Introduction to Theoretical Computer Science

Zdeněk Sawa

Department of Computer Science, FEI,
Technical University of Ostrava
17. listopadu 15, Ostrava-Poruba 708 33
Czech republic

February 6, 2014

Lecturer

Name: Ing. Zdeněk Sawa, Ph.D.
E-mail: zdenek.sawa@vsb.cz
Room: A1024
Web: http://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
- Actual information
- A link to a page with animations

Requirements

- **Credit** (22 points):
  - Written test (22 points) — it will be written on a tutorial

  The minimal requirement for obtaining the credit is 7 points.

  A correcting test for 14 points.

- **Exam** (78 points)
  - A written exam consisting of three parts (26 points for each part); it is necessary to obtain at least 10 points for each part.

Theoretical Computer Science

Different areas of theoretical computer science:

- algorithms
- computational complexity
- models of computation
- automata theory
- formal languages
- syntax and semantics of programming languages
- type theory
- parallel and distributed systems
- ...

Theoretical Computer Science

Overlapping with many other areas of mathematics and computer science:

- logic
- graph theory
- number theory
- cryptography
- computational geometry
- game theory
- numerical mathematics
- ...

Theoretical Computer Science

Some important characteristics of theoretical computer science:

- formal mathematical approach to problems
- the use of mathematical definitions and proofs
- rigorous mathematical proofs

The notion of a proof, aximatic approach.

Logic — the study of reasoning, proofs, formalisms

Conjunction

conjuction: $A \wedge B$

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Remark: also denoted as & or **and**

Rules for Conjunction

$$\wedge\text{i:}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\wedge\text{e}_1\text{:}\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \wedge\text{e}_2\text{:}\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}$$

$$\text{Assm:}\ \frac{}{\Gamma, A, \Delta \vdash A}$$

A Formally Written Proof

$$A \wedge B, C \wedge D \vdash B \wedge C$$

1. $A \wedge B, C \wedge D \vdash A \wedge B$ (Assm)
2. $A \wedge B, C \wedge D \vdash B$ ($\wedge \mathrm{e}_2$ 1)
3. $A \wedge B, C \wedge D \vdash C \wedge D$ (Assm)
4. $A \wedge B, C \wedge D \vdash C$ ($\wedge \mathrm{e}_1$ 3)
5. $A \wedge B, C \wedge D \vdash B \wedge C$ ($\wedge \mathrm{i}$ 2,4)

A Formally Written Proof

$$A \wedge B, C \wedge D \vdash B \wedge C$$

1. $\Gamma \vdash A \wedge B$ (Assm)
2. $\Gamma \vdash C \wedge D$ (Assm)
3. $\Gamma \vdash C$ ($\wedge \mathrm{e}_1$ 2)
4. $\Gamma \vdash B$ ($\wedge \mathrm{e}_2$ 1)
5. $\Gamma \vdash B \wedge C$ ($\wedge \mathrm{i}$ 4,3)

Remark: $\Gamma := A \wedge B, C \wedge D$

A Proof Represented as a Tree

$$A \wedge B, C \wedge D \vdash B \wedge C$$

$$\dfrac{\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \qquad \dfrac{\Gamma \vdash C \wedge D}{\Gamma \vdash C}}{\Gamma \vdash B \wedge C}$$

Remark: $\Gamma := A \wedge B, C \wedge D$

Derived Rule

$$\frac{\Gamma \vdash (A \wedge B) \wedge C}{\Gamma \vdash A \wedge (B \wedge C)}$$

1.	$\Gamma \vdash (A \wedge B) \wedge C$	(premise)
2.	$\Gamma \vdash A \wedge B$	$(\wedge \mathrm{e}_1\ 1)$
3.	$\Gamma \vdash A$	$(\wedge \mathrm{e}_1\ 2)$
4.	$\Gamma \vdash B$	$(\wedge \mathrm{e}_2\ 2)$
5.	$\Gamma \vdash C$	$(\wedge \mathrm{e}_2\ 1)$
6.	$\Gamma \vdash B \wedge C$	$(\wedge \mathrm{i}\ 4{,}5)$
7.	$\Gamma \vdash A \wedge (B \wedge C)$	$(\wedge \mathrm{i}\ 3{,}6)$

Assumptions

permutation:

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

weakening:

$$\frac{\Gamma, \Delta \vdash B}{\Gamma, A, \Delta \vdash B}$$

contraction:

$$\frac{\Gamma, A, A, \Delta \vdash B}{\Gamma, A, \Delta \vdash B}$$

$$\text{Ant:} \quad \frac{\Gamma \vdash A}{\Gamma' \vdash A}(\Gamma \subseteq \Gamma')$$

$$\text{Ch:} \quad \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}$$

Disjuction

Disjunction: $A \vee B$

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Remark: also denoted as **or**

XOR

xor (exclusive or): $A \oplus B$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Rules for Disjuction

$$\vee\mathrm{i}_1\text{:}\ \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \qquad \vee\mathrm{i}_2\text{:}\ \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$\vee\mathrm{e}\text{:}\ \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}$$

An Example of a Proof

$$(A \vee B) \vee C \vdash A \vee (B \vee C)$$

1.	$\varphi \vdash (A \vee B) \vee C$	(Assm)
2.	$\varphi, A \vee B \vdash A \vee B$	(Assm)
3.	$\varphi, A \vee B, A \vdash A$	(Assm)
4.	$\varphi, A \vee B, A \vdash A \vee (B \vee C)$	($\vee \mathrm{i}_1$ 3)
5.	$\varphi, A \vee B, B \vdash B$	(Assm)
6.	$\varphi, A \vee B, B \vdash B \vee C$	($\vee \mathrm{i}_1$ 5)
7.	$\varphi, A \vee B, B \vdash A \vee (B \vee C)$	($\vee \mathrm{i}_2$ 6)
8.	$\varphi, A \vee B \vdash A \vee (B \vee C)$	($\vee \mathrm{e}$ 2,4,7)
9.	$\varphi, C \vdash C$	(Assm)
10.	$\varphi, C \vdash B \vee C$	($\vee \mathrm{i}_2$ 9)
11.	$\varphi, C \vdash A \vee (B \vee C)$	($\vee \mathrm{i}_2$ 10)
12.	$\varphi \vdash A \vee (B \vee C)$	($\vee \mathrm{e}$ 1,8,11)

Remark: $\varphi := (A \vee B) \vee C$

Nagation

Negation: $\neg A$

A	$\neg A$
0	1
1	0

Remark: also denoted by $\sim$ or **not**

Proofs by Contradiction

$$\text{Ctr:}\ \frac{\Gamma, A \vdash B \qquad \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A}$$

$$\text{CtrN:}\ \frac{\Gamma, \neg A \vdash B \qquad \Gamma, \neg A \vdash \neg B}{\Gamma \vdash A}$$

Double Negation

$$\neg\neg\text{i:}\ \frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A} \qquad \neg\neg\text{e:}\ \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$$

1.	$\Gamma \vdash A$	(premise)
2.	$\Gamma, \neg A \vdash A$	(Ant 1)
3.	$\Gamma, \neg A \vdash \neg A$	(Assm)
4.	$\Gamma \vdash \neg\neg A$	(Ctr 2,3)

1.	$\Gamma \vdash \neg\neg A$	(premise)
2.	$\Gamma, \neg A \vdash \neg A$	(Assm)
3.	$\Gamma, \neg A \vdash \neg\neg A$	(Ant 1)
4.	$\Gamma \vdash A$	(CtrN 2,3)

Double Negation

The proof of CtrN by Ctr and $\neg\neg$e:

$$\text{CtrN:}\ \frac{\Gamma, \neg A \vdash B \qquad \Gamma, \neg A \vdash \neg B}{\Gamma \vdash A}$$

1. $\Gamma, \neg A \vdash B$ (premise)
2. $\Gamma, \neg A \vdash \neg B$ (premise)
3. $\Gamma \vdash \neg\neg A$ (Ctr 1,2)
4. $\Gamma \vdash A$ ($\neg\neg$e 3)

From a Contradiction Anything Follows

$$\text{CtrA:}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash B}$$

1. $\Gamma \vdash A$ (premise)
2. $\Gamma \vdash \neg A$ (premise)
3. $\Gamma, \neg B \vdash A$ (Ant 1)
4. $\Gamma, \neg B \vdash \neg A$ (Ant 2)
5. $\Gamma \vdash B$ (CtrN 3,4)

Contraposition

$$\text{Cp (a)}: \quad \frac{\Gamma, A \vdash B}{\Gamma, \neg B \vdash \neg A} \qquad \text{Cp (b)}: \quad \frac{\Gamma, A \vdash \neg B}{\Gamma, B \vdash \neg A}$$

$$\text{Cp (c)}: \quad \frac{\Gamma, \neg A \vdash B}{\Gamma, \neg B \vdash A} \qquad \text{Cp (d)}: \quad \frac{\Gamma, \neg A \vdash \neg B}{\Gamma, B \vdash A}$$

1.	$\Gamma, A \vdash B$	(premise)
2.	$\Gamma, \neg B, A \vdash B$	(Ant 1)
3.	$\Gamma, \neg B, A \vdash \neg B$	(Assm)
4.	$\Gamma, \neg B \vdash \neg A$	(Ctr 2,3)

1.	$\Gamma, A \vdash \neg B$	(premise)
2.	$\Gamma, B, A \vdash B$	(Assm)
3.	$\Gamma, B, A \vdash \neg B$	(Ant 1)
4.	$\Gamma, B \vdash \neg A$	(Ctr 2,3)

Chaining of Arguments

$$\text{Ch:}\ \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}$$

1.	$\Gamma \vdash A$	(premise)
2.	$\Gamma, A \vdash B$	(premise)
3.	$\Gamma, \neg B \vdash A$	(Ant 1)
4.	$\Gamma, \neg B \vdash \neg A$	(Cp (a) 2)
5.	$\Gamma \vdash B$	(CtrN 3,4)

Proof by Cases

$$\text{PC:}\ \frac{\Gamma, A \vdash B \qquad \Gamma, \neg A \vdash B}{\Gamma \vdash B}$$

1. $\Gamma, A \vdash B$ (premise)
2. $\Gamma, \neg A \vdash B$ (premise)
3. $\Gamma, \neg B \vdash A$ (Cp (c) 2)
4. $\Gamma, \neg B \vdash \neg A$ (Cp (a) 1)
5. $\Gamma \vdash B$ (CtrN 3,4)

Law of Exluded Middle (Tertium Non Datur)

$$\vdash A \vee \neg A$$

1. $A \vdash A$ (Assm)
2. $A \vdash A \vee \neg A$ ($\vee \mathrm{i}_1$ 1)
3. $\neg A \vdash \neg A$ (Assm)
4. $\neg A \vdash A \vee \neg A$ ($\vee \mathrm{i}_2$ 3)
5. $\vdash A \vee \neg A$ (PC 2,4)

Other Rules for Disjunction

$$\frac{\Gamma, \neg A \vdash B}{\Gamma \vdash A \vee B} \qquad \frac{\Gamma, \neg B \vdash A}{\Gamma \vdash A \vee B}$$

1. $\Gamma, A \vdash A$ (Assm)
2. $\Gamma, A \vdash A \vee B$ ($\vee \mathrm{i}_1$ 1)
3. $\Gamma, \neg A \vdash B$ (premise)
4. $\Gamma, \neg A \vdash A \vee B$ ($\vee \mathrm{i}_2$ 3)
5. $\Gamma \vdash A \vee B$ (PC 2,4)

Implication

Implication: $A \to B$

A	B	$A \to B$
0	0	1
0	1	1
1	0	0
1	1	1

Remark: also denoted by $\Rightarrow$ or $\supset$

$$\to\mathrm{i}: \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \qquad\qquad \to\mathrm{e}: \quad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

Remark: $\to\mathrm{e}$ is better known as *modus ponens*

Examples of Proofs with Implication

$$\frac{\Gamma \vdash A \to B}{\Gamma, A \vdash B}$$

$$\text{Ch:}\ \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}$$

1. $\Gamma \vdash A \to B$ (premise)
2. $\Gamma, A \vdash A \to B$ (Ant 1)
3. $\Gamma, A \vdash A$ (Assm)
4. $\Gamma, A \vdash B$ ($\to$e 2,3)

1. $\Gamma \vdash A$ (premise)
2. $\Gamma, A \vdash B$ (premise)
3. $\Gamma \vdash A \to B$ ($\to$i 2)
4. $\Gamma \vdash B$ ($\to$e 3,1)

Equivalence

equivalence: $A \leftrightarrow B$

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

Remark: also denoted by $\Leftrightarrow$ or $\equiv$ or **iff**

Rules for Equivalence

$$\leftrightarrow \mathrm{i}: \quad \frac{\Gamma, A \vdash B \qquad \Gamma, B \vdash A}{\Gamma \vdash A \leftrightarrow B}$$

$$\leftrightarrow \mathrm{e}_1: \quad \frac{\Gamma \vdash A \leftrightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

$$\leftrightarrow \mathrm{e}_2: \quad \frac{\Gamma \vdash A \leftrightarrow B \qquad \Gamma \vdash B}{\Gamma \vdash A}$$

Logical Constants

Logical Constants: $\bot$ (false), $\top$ (true)

Remark: also denoted by 0, 1 or **ff**, **tt**

$$\bot\text{e:}\ \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \qquad\qquad \top\text{i:}\ \frac{}{\vdash \top}$$

$\bot$ and Negation

$$\neg\text{i:}\quad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}$$

1. $\Gamma, A \vdash \bot$ (premise)
2. $\Gamma, A \vdash \neg\bot$ ($\bot$e 1)
3. $\Gamma \vdash \neg A$ (Ctr 1,2)

$$\neg\text{e:}\quad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}$$

$\bot$ and Negation

$$\text{Ctr:}\ \frac{\Gamma, A \vdash B \qquad \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A}$$

1. $\Gamma, A \vdash B$ (premise)
2. $\Gamma, A \vdash \neg B$ (premise)
3. $\Gamma, A \vdash \bot$ ($\neg$e 2,1)
4. $\Gamma \vdash \neg A$ ($\neg$i 3)

$$\text{CtrA:}\ \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash B}$$

1. $\Gamma \vdash A$ (premise)
2. $\Gamma \vdash \neg A$ (premise)
3. $\Gamma \vdash \bot$ ($\neg$e 2,1)
4. $\Gamma \vdash B$ ($\bot$e 3)

Syntax — Logical Connectives

- If φ is a well-formed formula, then also $\neg\varphi$ is a well-formed formula.
- If φ and ψ are well-formed formulas, then also $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are well-formed formulas.
- $\bot$ and $\top$ are well-formed formulas.

Abstract syntax tree.

Conventions for omitting parentheses.

Propositional Logic — Syntax

At — the set of atomic propositions

Alphabet — the set of symbols:

- atomic propositions: all elements from At
- logical connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\bot$, $\top$
- parentheses: $)$, $($

Language — which sequences of symbols are formulas:

- If $p \in At$, then p is a formula.
- $\bot$ and $\top$ are formulas.
- If φ is a formula, then also $\neg\varphi$ is a formula.
- If φ and ψ are formulas, then also $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are formulas.
- There are no other formulas than those created according to the previous rules.

Propositional Logic — Syntax

A more succinct desciption of the syntax using so called Backus-Naur form:

$$\varphi \ ::= \ p \mid \bot \mid \top \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi)$$

p represents an arbitrary atomic proposition from the set At

In the so called abstract syntax, we abstract from details such as parentheses, priorities, etc.:

$$\varphi \ ::= \ p \mid \bot \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi$$

An alternative notation for the abstract syntax:

$$\varphi \ ::= \ p \mid \bot \mid \top \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2$$

Propsitional Logic — Semantics

Truth valuation: $\nu : At \to \{0, 1\}$

- $\nu \models \varphi$ — fomula φ is true in valuation ν
- $\nu \not\models \varphi$ — formula φ is not true in valuation ν

The definition of when a formula is true in valuation ν:

- $\nu \models p$, where $p \in At$, holds if and only if $\nu(p) = 1$.
- $\nu \models \bot$ never holds (i.e., it always holds that $\nu \not\models \bot$).
- $\nu \models \top$ always holds.
- $\nu \models \neg\varphi$ holds if and only if $\nu \not\models \varphi$.
- $\nu \models \varphi \wedge \psi$ holds if and only if $\nu \models \varphi$ and $\nu \models \psi$.
- $\nu \models \varphi \vee \psi$ holds if and only if $\nu \models \varphi$ or $\nu \models \psi$.
- $\nu \models \varphi \to \psi$ holds if and only if $\nu \not\models \varphi$ or $\nu \models \psi$.
- $\nu \models \varphi \leftrightarrow \psi$ holds if and only if $\nu \models \varphi$ and $\nu \models \psi$, or if $\nu \not\models \varphi$ and $\nu \not\models \psi$.

Propositional Logic — Semantics

$$h_\neg : \{0,1\} \to \{0,1\}$$

x	$h_\neg(x)$
0	1
1	0

$$h_* : \{0,1\} \times \{0,1\} \to \{0,1\} \quad \text{for } * \in \{\wedge, \vee, \to, \leftrightarrow\}$$

x	y	$h_\wedge(x,y)$	$h_\vee(x,y)$	$h_\to(x,y)$	$h_\leftrightarrow(x,y)$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Propositional Logic — Semantics

$\mathcal{L}_{At}$ — the set of all well-formed formulas

To each truth valuation $\nu : At \to \{0, 1\}$, there is assigned a function

$$\hat{\nu} : \mathcal{L}_{At} \to \{0, 1\}$$

- $\hat{\nu}(p) = \nu(p)$ for $p \in At$
- $\hat{\nu}(\bot) = 0$
- $\hat{\nu}(\top) = 1$
- $\hat{\nu}(\neg\varphi) = h_{\neg}(\hat{\nu}(\varphi))$
- $\hat{\nu}(\varphi * \psi) = h_{*}(\hat{\nu}(\varphi), \hat{\nu}(\psi))$ for $* \in \{\wedge, \vee, \to, \leftrightarrow\}$

- $\nu \models \varphi$ — when $\hat{\nu}(\varphi) = 1$
- $\nu \not\models \varphi$ — when $\hat{\nu}(\varphi) = 0$

Interpretations and Models

In general, when discussing a semantics of a logic, we talk about **interpretations**.

- In the case of propositional logic, an interpretation is just a truth valuation.
- For other logics, interpretations can be something different.
- If ν is an interpretation, the notation $\nu \models \varphi$ denotes that formula φ is true in the intepretation ν.
- An interpretation, where formula φ holds, is called a **model** of the formula φ.
- When Γ is a set of formulas, a **model** of this set of formulas is such interpretation that is a model of each formula from Γ.

Logical Entailment

φ – formula, Γ – a set of formulas

Fomula φ **logically follows** from formulas Γ, which is written as

$$\Gamma \models \varphi,$$

if $\nu \models \varphi$ hold for each interpretation ν, where for all formulas $\psi \in \Gamma$ it holds that $\nu \models \psi$.

Remark: The set Γ can be infinite.

Instead of $\{\psi_1, \psi_2, \ldots, \psi_n\} \models \varphi$, we usually write $\psi_1, \psi_2, \ldots, \psi_n \models \varphi$.

Instead of $\emptyset \models \varphi$, we usually write $\models \varphi$.

Tautologies, Contradictions, and Satisfiable Formulas

Formula φ is:

- **tautology** — for every interpretation ν it holds that $\nu \models \varphi$
- **contradiction** — for every interpretation ν it holds that $\nu \not\models \varphi$
- **satisfiable** — there exists at least one interpretation, for which it holds that $\nu \models \varphi$

$\models \varphi$ — denotes that φ is a tautology

Remark:

$$A_1, A_2, \ldots, A_n \models B$$

iff

$$\models A_1 \to (A_2 \to (\ \cdots\ \to (A_n \to B) \cdots))$$

Table Method

$$p \wedge (q \rightarrow \neg r)$$

p	q	r	$\neg r$	$q \rightarrow \neg r$	$p \wedge (q \rightarrow \neg r)$
0	0	0	1	1	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0

Table Method

$$((A \to B) \to A) \to A$$

A	B	$A \to B$	$(A \to B) \to A$	$((A \to B) \to A) \to A$
0	0	1	0	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

Deduction Systems

Deduction system — a set of rules that determine, how proofs can look and what kinds of steps can be done in these proofs .

Example: A deduction system for propositional logic based on sequents (sequent calculus), containing the following rules:

Assm	Ant
$\wedge\mathrm{i}$	$\wedge\mathrm{e}_1$, $\wedge\mathrm{e}_2$
$\vee\mathrm{i}_1$, $\vee\mathrm{i}_2$	$\vee\mathrm{e}$
$\rightarrow\mathrm{i}$	$\rightarrow\mathrm{e}$
$\leftrightarrow\mathrm{i}$	$\leftrightarrow\mathrm{e}_1$, $\leftrightarrow\mathrm{e}_2$
$\neg\mathrm{i}$	$\neg\mathrm{e}$
$\top\mathrm{i}$	$\bot\mathrm{e}$
	$\neg\neg\mathrm{e}$

Remark: This is one of variants of so called **natural deduction**.

Deduction Systems

φ – a formula, Γ – a set of formulas (it can be infinite)

For a given deduction system $\mathcal{D}$, the notation

$$\Gamma \vdash_{\mathcal{D}} \varphi$$

denotes that in the system $\mathcal{D}$, there exists a proof of formula φ from assumptions Γ.

For a given deduction system $\mathcal{D}$, it must specified precisely when $\Gamma \vdash_{\mathcal{D}} \varphi$ holds.

A system based on sequents (sequent calculus):

- $\Gamma \vdash_{\mathcal{D}} \varphi$ holds iff there are some formulas $\psi_1, \psi_2, \ldots, \psi_n$ from set Γ such that using the rules of the system it is possible to derive

  $$\psi_1, \psi_2, \ldots, \psi_n \vdash \varphi$$

  (i.e., there exists a proof of this sequent).

Deduction Systems

A deduction system $\mathcal{D}$ is:

- **sound** — if it holds that if $\Gamma \vdash_{\mathcal{D}} \varphi$ then $\Gamma \models \varphi$

  I.e., it is not possible to prove an incorrect conclusion in this system. What is derived, it really holds.

- **complete** — if it holds that if $\Gamma \models \varphi$ then $\Gamma \vdash_{\mathcal{D}} \varphi$

  I.e., everything that holds can be also proved in the system.

Deduction Systems

For proving that a given system is sound, it is sufficient to prove that every rule is sound.

Example: For proving that rule

$$\wedge\text{i:}\quad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

is sound, it is sufficient to show that:

- if $\Gamma \models A$ and $\Gamma \models B$, then $\Gamma \models A \wedge B$.

Table Method in the Deduction System

It is possible to simulate the table method in the deduction system:

- The fact that φ has truth value 1 in a given truth valuation, is represented by formula φ.
- The fact that φ has truth value 0 in a given truth valuation, is represented by formula $\neg\varphi$.

Table Method in the Deduction System

Example: Let us have atomic propositions p, q, r, and a formula φ constructed from those atomic propositions using logical connectives.

- If for example $\nu_1 \models \varphi$ for valuation ν_1, where
$$\nu_1(p) = 1, \qquad \nu_1(q) = 0, \qquad \nu_1(r) = 0,$$
it is possible to derive sequent
$$p, \neg q, \neg r \vdash \varphi$$

- If for example $\nu_2 \not\models \varphi$ for valuation ν_2, where
$$\nu_2(p) = 0, \qquad \nu_2(q) = 1, \qquad \nu_2(r) = 0,$$
it is possible to derive sequent
$$\neg p, q, \neg r \vdash \neg\varphi$$

Table Method in the Deduction System

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

For example, it is possible to derive the following rules for conjunction:

$$\frac{\Gamma \vdash \neg A \qquad \Gamma \vdash \neg B}{\Gamma \vdash \neg(A \wedge B)} \qquad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash B}{\Gamma \vdash \neg(A \wedge B)}$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash \neg B}{\Gamma \vdash \neg(A \wedge B)} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

Table Method in the Deduction System

The table method can be viewed as an example of a proof by cases:

p_1	p_2	p_3	φ
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

1. $\neg p_1, \neg p_2, \neg p_3 \vdash \varphi$ (...)
2. $\neg p_1, \neg p_2, p_3 \vdash \varphi$ (...)
3. $\neg p_1, p_2, \neg p_3 \vdash \varphi$ (...)
4. $\neg p_1, p_2, p_3 \vdash \varphi$ (...)
5. $p_1, \neg p_2, \neg p_3 \vdash \varphi$ (...)
6. $p_1, \neg p_2, p_3 \vdash \varphi$ (...)
7. $p_1, p_2, \neg p_3 \vdash \varphi$ (...)
8. $p_1, p_2, p_3 \vdash \varphi$ (...)
9. $\neg p_1, \neg p_2 \vdash \varphi$ (PC 2,1)
10. $\neg p_1, p_2 \vdash \varphi$ (PC 4,3)
11. $p_1, \neg p_2 \vdash \varphi$ (PC 6,5)
12. $p_1, p_2 \vdash \varphi$ (PC 8,7)
13. $\neg p_1 \vdash \varphi$ (PC 10,9)
14. $p_1 \vdash \varphi$ (PC 12,11)
15. $\vdash \varphi$ (PC 14,13)

Equivalence of Formulas

Formulas φ and ψ are **logicaly equivalent**, which is denoted $\varphi \Leftrightarrow \psi$, if for every interpretation ν it holds

$$\nu \models \varphi \qquad \text{iff} \qquad \nu \models \psi.$$

Remark: $\varphi \Leftrightarrow \psi$ holds iff $\varphi \models \psi$ and $\psi \models \varphi$.

Formulas φ and ψ are **provably equivalent** in a given deduction system, which is denoted $\varphi \dashv\vdash \psi$, if

$$\varphi \vdash \psi \qquad \text{and} \qquad \psi \vdash \varphi.$$

Remark: If the deduction system is sound, then from $\varphi \dashv\vdash \psi$ follows $\varphi \Leftrightarrow \psi$, if it is complete, then from $\varphi \Leftrightarrow \psi$ follows $\varphi \dashv\vdash \psi$.

Equivalence of Formulas

$\varphi \Leftrightarrow \psi$ iff $\models \varphi \leftrightarrow \psi$.

$\varphi \dashv\vdash \psi$ iff $\vdash \varphi \leftrightarrow \psi$.

For arbitrary formulas φ, ψ, and χ, it holds:

- $\varphi \Leftrightarrow \varphi$.
- If $\varphi \Leftrightarrow \psi$, then $\psi \Leftrightarrow \varphi$.
- If $\varphi \Leftrightarrow \psi$ and $\psi \Leftrightarrow \chi$, then $\varphi \Leftrightarrow \chi$.

Similarly:

- $\varphi \dashv\vdash \varphi$.
- If $\varphi \dashv\vdash \psi$, then $\psi \dashv\vdash \varphi$.
- If $\varphi \dashv\vdash \psi$ and $\psi \dashv\vdash \chi$, then $\varphi \dashv\vdash \chi$.

Equivalence of Formulas

When $A \dashv\vdash A'$ holds:

- It is possible to prove $\Gamma \vdash A'$ from $\Gamma \vdash A$ (and vice versa).
- It is possible to prove $\Gamma, A', \Delta \vdash B$ from $\Gamma, A, \Delta \vdash B$ (and vice versa).

When $A \dashv\vdash A'$ then it also holds that:

- $\neg A \dashv\vdash \neg A'$
- $A \wedge B \dashv\vdash A' \wedge B, \quad B \wedge A \dashv\vdash B \wedge A'$
- $A \vee B \dashv\vdash A' \vee B, \quad B \vee A \dashv\vdash B \vee A'$
- $A \to B \dashv\vdash A' \to B, \quad B \to A \dashv\vdash B \to A'$
- $A \leftrightarrow B \dashv\vdash A' \leftrightarrow B, \quad B \leftrightarrow A \dashv\vdash B \leftrightarrow A'$

(similarly for $\Leftrightarrow$)

Some Important Equivalences

Remark: All equivalences mentioned below are also provable ($\dashv\vdash$).

$$A \Leftrightarrow \neg\neg A\,,$$

Associativity of $\wedge$, $\vee$, $\leftrightarrow$:

$$\begin{aligned}(A \wedge B) \wedge C &\Leftrightarrow A \wedge (B \wedge C)\\ (A \vee B) \vee C &\Leftrightarrow A \vee (B \vee C)\\ (A \leftrightarrow B) \leftrightarrow C &\Leftrightarrow A \leftrightarrow (B \leftrightarrow C)\end{aligned}$$

Commutativity of $\wedge$, $\vee$, $\leftrightarrow$:

$$A \wedge B \Leftrightarrow B \wedge A \qquad A \vee B \Leftrightarrow B \vee A \qquad A \leftrightarrow B \Leftrightarrow B \leftrightarrow A$$

Idempotence of $\wedge$ and $\vee$:

$$A \wedge A \Leftrightarrow A \qquad A \vee A \Leftrightarrow A$$

Some Important Equivalences

Distributive laws for $\wedge$ and $\vee$:

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C) \qquad A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$$

De Morgan's laws:

$$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B \qquad \neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$$

Conjuction, resp. disjunction, of formulas A and $\neg A$ is equivalent to $\bot$, resp. $\top$:

$$A \wedge \neg A \Leftrightarrow \bot \qquad A \vee \neg A \Leftrightarrow \top$$

Some Important Equivalences

Equvalences involving $\bot$ and $\top$:

$$\neg\bot \Leftrightarrow \top \qquad \neg\top \Leftrightarrow \bot$$
$$A \wedge \bot \Leftrightarrow \bot \qquad A \wedge \top \Leftrightarrow A$$
$$A \vee \bot \Leftrightarrow A \qquad A \vee \top \Leftrightarrow \top$$
$$A \rightarrow \bot \Leftrightarrow \neg A \qquad A \rightarrow \top \Leftrightarrow \top$$
$$\bot \rightarrow A \Leftrightarrow \top \qquad \top \rightarrow A \Leftrightarrow A$$
$$A \leftrightarrow \bot \Leftrightarrow \neg A \qquad A \leftrightarrow \top \Leftrightarrow A$$

Equivalences for replacing the logical connective implication:

$$A \rightarrow B \Leftrightarrow \neg A \vee B \qquad \neg(A \rightarrow B) \Leftrightarrow A \wedge \neg B$$

Equivalences for replacing the logical connective equivalence:

$$A \leftrightarrow B \Leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$$
$$A \leftrightarrow B \Leftrightarrow (A \wedge B) \vee (\neg A \wedge \neg B)$$
$$A \leftrightarrow B \Leftrightarrow (A \vee \neg B) \wedge (\neg A \vee B)$$

Equivalent Transformations

Instead of $A_1 \Leftrightarrow A_2$, $A_2 \Leftrightarrow A_3$, $\ldots$, $A_{n-1} \Leftrightarrow A_n$, it is possible to write

$$A_1 \Leftrightarrow A_2 \Leftrightarrow A_3 \Leftrightarrow \cdots \Leftrightarrow A_{n-1} \Leftrightarrow A_n$$

For arbitrary A_i, A_j, where $i, j \in \{1, 2, \ldots, n\}$, it holds that $A_i \Leftrightarrow A_j$.

Similarly, instead of $A_1 \dashv\vdash A_2$, $A_2 \dashv\vdash A_3$, $\ldots$, $A_{n-1} \dashv\vdash A_n$, it is possible to write

$$A_1 \dashv\vdash A_2 \dashv\vdash A_3 \dashv\vdash \cdots \dashv\vdash A_{n-1} \dashv\vdash A_n$$

For arbitrary A_i, A_j, where $i, j \in \{1, 2, \ldots, n\}$, it holds that $A_i \dashv\vdash A_j$.

Normal Forms of Fomulas

Let us consider only formulas of propositional logic.

- **Literal** — an atomic proposition or its negation, e.g., p or $\neg r$

$$L \ ::= \ p \ \mid \ \neg p$$

- **Elementary conjunction** — a conjunction of one of more literals, e.g., $p \wedge \neg q$, r, $q \wedge \neg r \wedge p$

$$C \ ::= \ L \ \mid \ L \wedge C$$

- **Elementary disjunction** (**clause**) — a disjunction of one or more literals, e.g., $p \vee \neg q$, r, $q \vee \neg r \vee p$

$$D \ ::= \ L \ \mid \ L \vee D$$

Normal Forms of Fomulas

- **Disjunctive normal form** (**DNF**) — a disjunction of one or more elementary conjunctions, e.g., $(p \wedge \neg q) \vee (\neg r) \vee (\neg r \wedge \neg p \wedge \neg q)$

$$E \ ::= \ C \ \mid \ C \vee E$$

- **Conjunctive normal form** (**CNF**) — a conjunction of one or more elementary disjunctions (clauses),
e.g., $(p \vee \neg q) \wedge (\neg r) \wedge (\neg r \vee \neg p \vee \neg q)$

$$F \ ::= \ D \ \mid \ D \wedge F$$

Remark: Formulas $\bot$ and $\top$ will be also considered to be in DNF and CNF.

Normal Forms of Fomulas

- For formulas in CNF, it is easy to determine if they are tautologies or not:

  In a tautology in CNF, every clause constains as literals some atomic proposition p and its negation $\neg p$.

- For formulas in DNF, it is easy to determine if they are contradictions or not:

  In a contradiction in DNF, every elementary conjunction contains as literals some atomic proposition p and its negation $\neg p$.

Normal Forms of Fomulas

p	q	r	φ
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

DNF:

$(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge r)$

KNF:

$(p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$

Normal Forms of Fomulas

When we consider a fixed **finite** set of atomic propositions At:

- **Complete disjunctive normal form** (**CDNF**) — a formula in DNF, where every elementary conjunction contains every atomic proposition from At exactly once.

  Example: $(p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r)$

- **Complete conjunctive normal form** (**CCNF**) — a formula in CNF, where every clause contains every atomic proposition from At exactly once.

  Example: $(p \vee \neg q \vee \neg r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$

Remark: In the examples is $At = \{p, q, r\}$.

Sets

Set — a collection of elements

Element a is a member of set S:

$$a \in S$$

Element a is not a member of set S: $a \notin S$

A finite set can be expressed by a list of elements of the set:

$$S = \{a, b, c\}$$

Subset: $S \subseteq T$ — every element of set S is a member of set T

Ordered n-tuples

Ordered n-tuple: $(a_1, a_2, \ldots, a_n)$ or $\langle a_1, a_2, \ldots, a_n \rangle$

- Ordered pair: (a, b) or $\langle a, b \rangle$
- Ordered triple: (a, b, c) or $\langle a, b, c \rangle$
- ...

Cartesian product:

$$S_1 \times S_2 \times \cdots \times S_n$$

— the set of all ordered n-tuples

$$(a_1, a_2, \ldots, a_n)$$

where $a_1 \in S_1$, $a_2 \in S_2$, ..., $a_n \in S_n$

Relations

Relation: $R \subseteq S_1 \times S_2 \times \cdots \times S_n$

- to express relationships between n-tuples of elements
- n — the arity of the relation
  - $n = 1$ — unary
  - $n = 2$ — binary
  - $n = 3$ — ternary

Example: Binary relation $R_1 \subseteq \mathbb{N} \times \mathbb{N}$ consisting of pairs of numbers (m, n), where $m < n$

$$(m, n) \in R_1 \qquad \text{iff} \qquad m < n$$

Remark: $\mathbb{N} = \{0, 1, 2, \ldots\}$

Relations

Predicate — assigns a truth value to a given n-tuple of elements $(a_1, a_2, \ldots, a_n)$, depending on whether this n-tuple is or is not in a given relation R:

$$R(a_1, a_2, \ldots, a_n)$$

I.e., $R(a_1, a_2, \ldots, a_n)$ holds iff $(a_1, a_2, \ldots, a_n) \in R$.

Example: Predicate R_1, where

$$R_1(m, n)$$

holds iff $(m, n) \in R_1$, i.e., when $m < n$

Relations

Unary relation — $R \subseteq S$ represents some property of elements from set S

Example: Unary relation $R_2 \subseteq \mathbb{N}$ consisting of those numbers that are primes

Predicate R_2 expressing that n is a prime

$$R_2(n)$$

Functions

Function — a binary relations $f \subseteq S \times T$ satisfying:

- if $(x, y_1) \in f$ and $(x, y_2) \in f$, then $y_1 = y_2$

I.e., for every element $x \in S$, there exists at most one element $y \in T$ such that $(x, y) \in f$.

This element is denoted

$$f(x)$$

Function f — represents a mapping that assignes elements from set T to elements from set S:

$$f : S \to T$$

Total vs. parcial functions

Functions

Function $f : S \to T$, where $S = A_1 \times A_2 \times \cdots \times A_n$:

$$f : A_1 \times A_2 \times \cdots \times A_n \to T$$

n — the arity of function f

Instead of $f((a_1, a_2, \ldots, a_n))$, we write $f(a_1, a_2, \ldots, a_n)$.

Example: Binary function $f_1 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which assigns to a pair of numbers their sum

$$f_1(x, y) = x + y$$

Structures

Structure — a non-empty set of elements together with several relations and functions over the elelents of this set

Example: Set $\mathbb{N} = \{0, 1, 2, \ldots\}$ together with the following relations and functions:

- unary function $f : \mathbb{N} \to \mathbb{N}$, where $f(x) = x + 1$
- binary function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $g(x, y) = x + y$
- binary function $h : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $h(x, y) = x \cdot y$
- binary relation $P \subseteq \mathbb{N} \times \mathbb{N}$, where $(x, y) \in P$ iff $x = y$
- binary relation $Q \subseteq \mathbb{N} \times \mathbb{N}$, where $(x, y) \in Q$ iff $x < y$

Structures

Example: Set $A = \{a, b, c\}$ together with the following functions f and g and relation R:

- unary function $f : A \to A$ and binary function $g : A \times A \to A$

x	$f(x)$
a	b
b	a
c	b

g	a	b	c
a	c	a	a
b	a	b	c
c	a	c	c

- binary relation $R \subseteq A \times A$, where

$$R = \{(a, a), (a, c), (b, b), (c, a), (c, b)\}$$

Signatures

Signature — tuple

$$(\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})$$

- $\mathcal{P}$ — a set of **predicate symbols**
- $\mathcal{F}$ — a set of **function symbols**
- $\mathcal{C}$ — a set of **constant symbols**
- $\text{arity} : \mathcal{P} \cup \mathcal{F} \to \mathbb{N}$ — a function determining the **arity** of each predicate or function symbol

Example: Signature $S = (\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})$, where $\mathcal{P} = \{P, Q, R\}$, $\mathcal{F} = \{f, g\}$, $\mathcal{C} = \{c, d\}$, and where $\text{arity}(P) = 1$, $\text{arity}(Q) = 1$, $\text{arity}(R) = 2$, $\text{arity}(f) = 2$, $\text{arity}(g) = 1$

Interpretations

Signature $S = (\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})$

Interpretation: $\mathfrak{A} = (A, \mathfrak{a})$

- A — **universe** — an arbitrary non-empty set
- $\mathfrak{a}$ — a mapping assigning meaning to every symbol in signature S:
  - for $P \in \mathcal{P}$ (where $\text{arity}(P) = n$):
    $\mathfrak{a}(P) = P^{\mathfrak{A}}$, where $P^{\mathfrak{A}} \subseteq A^n$ is an arbitrary n-ary relation over set A
  - for $f \in \mathcal{F}$ (where $\text{arity}(f) = n$):
    $\mathfrak{a}(f) = f^{\mathfrak{A}}$, where $f^{\mathfrak{A}} : A^n \to A$ is an arbitrary n-ary (total) function over set A
  - for $c \in \mathcal{C}$:
    $\mathfrak{a}(c) = c^{\mathfrak{A}}$, where $c^{\mathfrak{A}} \in A$ is an arbitrary element of the universe A

Valuations

Variables: $Var = \{x_0, x_1, x_2, \ldots\}$

Remark: Variables will be denoted x, y, z

Assume an interpretation $\mathfrak{A} = (A, \mathfrak{a})$.

Valuation — determines values of variables, it assignes elements of the universe to variables:

$$v : Var \to A$$

An interpretation and a valuation:

$$\mathfrak{I} = (\mathfrak{A}, v)$$

Terms

Term — an expression denoting an element of the universe:

- x — where $x \in Var$
- c — where $c \in \mathcal{C}$
- $f(t_1, t_2, \ldots, t_n)$ — where $f \in \mathcal{F}$, $\text{arity}(f) = n$, and where $t_1, t_2, \ldots, t_n$ are terms

Examples of terms: x c $f(x, y)$ $f(g(x), f(c, y))$

Syntax:

$$t \quad ::= \quad x \mid c \mid f(t, t, \ldots, t)$$

Terms

An interpretation and a valuation $\mathfrak{I} = (\mathfrak{A}, v)$

$\mathfrak{I}(t)$ — a value of term t in interpretation $\mathfrak{A}$ a valuation v

- for $x \in Var$: $\quad \mathfrak{I}(x) = v(x)$
- for $c \in \mathcal{C}$: $\quad \mathfrak{I}(c) = c^{\mathfrak{A}}$
- for $f \in \mathcal{F}$ (where $\text{arity}(f) = n$) and terms $t_1, t_2, \ldots, t_n$:
  $\mathfrak{I}(f(t_1, t_2, \ldots, t_n)) = f^{\mathfrak{A}}(\mathfrak{I}(t_1), \mathfrak{I}(t_2), \ldots, \mathfrak{I}(t_n))$

Atomic Formulas

Atomic formula:

$$P(t_1, t_2, \ldots, t_n)$$

where $P \in \mathcal{P}$ (where $\text{arity}(P) = n$) and $t_1, t_2, \ldots, t_n$ are terms

Examples of atomic formulas: $P(x)$ $Q(g(g(c)))$ $R(f(x, y), x)$

An interpretation and a valuation $\mathfrak{I} = (\mathfrak{A}, v)$

Formula φ is true in interpretation $\mathfrak{A}$ and valuation v:

$$\mathfrak{I} \models \varphi$$

- $\mathfrak{I} \models P(t_1, t_2, \ldots, t_n)$ iff $(\mathfrak{I}(t_1), \mathfrak{I}(t_2), \ldots, \mathfrak{I}(t_n)) \in P^{\mathfrak{A}}$.

Equality

Equality (**identity**): denoted by $=$

Atomic formula:

$$t_1 = t_2$$

Examples of atomic formulas with equality:

$$x = y \qquad f(f(x, y), z) = g(x)$$

- $\mathfrak{I} \models t_1 = t_2$ iff $\mathfrak{I}(t_1) = \mathfrak{I}(t_2)$

Quantifiers

It is possible to create formulas from smaller subformulas using logical connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\bot$, $\top$

It is also possible to use **quantifiers**:

- universal quantifier: $\forall$
- existential quantifier: $\exists$

If φ is a formula and x is a variable ($x \in \mathit{Var}$), then also:

- $\forall x \varphi$ is a formula

  — it represents propositions "for every element x, φ holds ", "for each x, φ holds", etc.

- $\exists x \varphi$ is a formula

  — it represents propositions "there exists an element x, for which φ holds", "for some x, φ holds", etc.

Quantifiers

Interpretation $\mathfrak{A} = (A, \mathfrak{a})$, valuation $v : Var \to A$

$x \in Var$, $a \in A$

Notation

$$v[x \mapsto a]$$

denotes the valuation $v' : Var \to A$, which assignes to every variable the value as valuation v, except that it assignes value a to variable x

I.e., for $y \in Var$ is

$$v'(y) = \begin{cases} a & \text{if } y = x \\ v(y) & \text{otherwise} \end{cases}$$

$\mathfrak{I} = (\mathfrak{A}, v)$

$$\mathfrak{I}[x \mapsto a] = (\mathfrak{A}, v[x \mapsto a])$$

Quantifiers

- $\mathfrak{I} \models \forall x \varphi$ holds iff for every $a \in A$ it holds that $\mathfrak{I}[x \mapsto a] \models \varphi$
- $\mathfrak{I} \models \exists x \varphi$ holds iff there exists some $a \in A$ such that $\mathfrak{I}[x \mapsto a] \models \varphi$

Examples of formulas:

- $\forall x \exists y R(x, y)$ — for each x there exists some y such that x and y are in relation R
- $\neg \exists x (P(x) \wedge Q(x))$ — there does not exist such x, for which both $P(x)$ and $Q(x)$ would be true
- $\exists x P(x) \rightarrow \forall y Q(y)$ — if there is some x, for which $P(x)$ holds, then for every y, $Q(y)$ holds

First order predicate logic — summary

Symbols:

- Logical symbols:
  - **variables**: $x \in Var$, where $Var = \{x_0, x_1, x_2, \ldots\}$
  - **logical connectives**: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
  - **quantifiers**: $\forall$, $\exists$
  - **parentheses**:), (
  - **equality**: $=$
- Non-logical symbols — given by signature $S = (\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})$:
  - **predicate symbols**: $P \in \mathcal{P}$
  - **function symbols**: $f \in \mathcal{F}$
  - **constant symbols**: $c \in \mathcal{C}$

Syntax:

$$
\begin{array}{rcl}
t & ::= & x \mid c \mid f(t, t, \ldots, t) \\
\varphi & ::= & P(t, t, \ldots, t) \mid t = t \mid \bot \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \\
& & \quad \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \forall x \varphi \mid \exists x \varphi
\end{array}
$$

First order predicate logic — summary

Semantics:

A value of a term for $\mathfrak{I} = (\mathfrak{A}, v)$:

- for $x \in Var$: $\quad \mathfrak{I}(x) = v(x)$
- for $c \in \mathcal{C}$: $\quad \mathfrak{I}(c) = c^{\mathfrak{A}}$
- for $f \in \mathcal{F}$ (where $\text{arity}(f) = n$) and terms $t_1, t_2, \ldots, t_n$:
  $\mathfrak{I}(f(t_1, t_2, \ldots, t_n)) = f^{\mathfrak{A}}(\mathfrak{I}(t_1), \mathfrak{I}(t_2), \ldots, \mathfrak{I}(t_n))$

First order predicate logic — summary

A truth value of a formula for $\mathfrak{I} = (\mathfrak{A}, \mathfrak{a})$:

- For $P \in \mathcal{P}$, where $\text{arity}(P) = n$, and for terms $t_1, t_2, \ldots, t_n$, $\mathfrak{I} \models P(t_1, t_2, \ldots, t_n)$ holds iff $(\mathfrak{I}(t_1), \mathfrak{I}(t_2), \ldots, \mathfrak{I}(t_n)) \in P^{\mathfrak{A}}$
- For terms t_1, t_2, $\mathfrak{I} \models t_1 = t_2$ holds iff $\mathfrak{I}(t_1) = \mathfrak{I}(t_2)$
- $\mathfrak{I} \models \bot$ never holds, i.e., $\mathfrak{I} \not\models \bot$ always holds
- $\mathfrak{I} \models \top$ always holds
- $\mathfrak{I} \models \neg\varphi$ holds iff $\mathfrak{I} \not\models \varphi$
- $\mathfrak{I} \models \varphi \wedge \psi$ holds iff $\mathfrak{I} \models \varphi$ and $\mathfrak{I} \models \psi$
- $\mathfrak{I} \models \varphi \vee \psi$ holds iff $\mathfrak{I} \models \varphi$ or $\mathfrak{I} \models \psi$
- $\mathfrak{I} \models \varphi \to \psi$ holds iff $\mathfrak{I} \not\models \varphi$ or $\mathfrak{I} \models \psi$
- $\mathfrak{I} \models \varphi \leftrightarrow \psi$ holds iff $\mathfrak{I} \models \varphi$ and $\mathfrak{I} \models \psi$, or if $\mathfrak{I} \not\models \varphi$ and $\mathfrak{I} \not\models \psi$
- $\mathfrak{I} \models \forall x \varphi$ holds iff for every $a \in A$ it holds that $\mathfrak{I}[x \mapsto a] \models \varphi$
- $\mathfrak{I} \models \exists x \varphi$ holds iff there exists some $a \in A$ such that $\mathfrak{I}[x \mapsto a] \models \varphi$

Free and Bound Occurrences of Variables

Every occurrence of variable x in a subformula of the form $\exists x\varphi$ or $\forall x\varphi$ is **bound**.

An occurrence of a variable, which is not bound, is **free**.

$\mathrm{free}(t)$ — the set of variables, which occur as free variables in term t:

- $\mathrm{free}(x) = \{x\}$ for $x \in \mathit{Var}$
- $\mathrm{free}(c) = \emptyset$ for $c \in \mathcal{C}$
- $\mathrm{free}(f(t_1, t_2, \ldots, t_n)) = \mathrm{free}(t_1) \cup \mathrm{free}(t_2) \cup \ldots \cup \mathrm{free}(t_n)$ for $f \in \mathcal{F}$

$\mathrm{free}(\varphi)$ — the set variables, which occur as free variables in formula φ:

- $\mathrm{free}(P(t_1, t_2, \ldots, t_n)) = \mathrm{free}(t_1) \cup \mathrm{free}(t_2) \cup \cdots \cup \mathrm{free}(t_n)$ for $P \in \mathcal{P}$
- $\mathrm{free}(t_1 = t_2) = \mathrm{free}(t_1) \cup \mathrm{free}(t_2)$
- $\mathrm{free}(\bot) = \mathrm{free}(\top) = \emptyset$
- $\mathrm{free}(\neg\varphi) = \mathrm{free}(\varphi)$
- $\mathrm{free}(\varphi * \psi) = \mathrm{free}(\varphi) \cup \mathrm{free}(\psi)$ for $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
- $\mathrm{free}(\mathrm{Q}x\varphi) = \mathrm{free}(\varphi) - \{x\}$ for $\mathrm{Q} \in \{\exists, \forall\}$ and $x \in \mathit{Var}$

Free and Bound Occurrences of Variables

The truth value of formula φ for $\mathfrak{I} = (\mathfrak{A}, v)$ depends only on $\mathfrak{A}$ a values assigned by v to variables from the set $\text{free}(\varphi)$.

In particular, in the case when $\text{free}(\varphi) = \emptyset$, the truth value of φ in $\mathfrak{I} = (\mathfrak{A}, v)$ depends only on the interpretation $\mathfrak{A}$.

For formalas φ, where $\text{free}(\varphi) = \emptyset$, we can write

$$\mathfrak{A} \models \varphi \qquad \text{or} \qquad \mathfrak{A} \not\models \varphi$$

Formula φ, where $\text{free}(\varphi) = \emptyset$, is called a **closed formula** or a **sentence**.

Substitution

φ – formula, x – variable, t – term

The formula obtained when we substiture term t for **free** occurrences of variable x in formula φ:

$$\varphi[t/x]$$

Example: $\varphi := \forall x(P(x) \to R(f(x,z),y)), \quad t := g(f(y,w))$

Formula $\varphi[t/z]$:

$$\forall x(P(x) \to R(f(x,g(f(y,w))),y))$$

Remark: It is necessary to avoid the situation when a free occurrence of some variable in term t becomes bound after the substitution.
In such case, it is necessary to rename the bound variable.

Deduction System for First Order Predicate Logic

A deduction system consisting of the same rules as the system for propositional logic.

Some additional rules for quantifiers and equality.

$$\forall \mathrm{e}: \quad \frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]}$$

Example:

$$\forall x(P(x) \to Q(x)),\ P(c) \vdash Q(c)$$

1.	$\Gamma \vdash \forall x(P(x) \to Q(x))$	(Assm)
2.	$\Gamma \vdash P(c) \to Q(c)$	($\forall$e 1)
3.	$\Gamma \vdash P(c)$	(Assm)
4.	$\Gamma \vdash Q(c)$	($\to$e 2,3)

$\Gamma := \forall x(P(x) \to Q(x)),\ P(c)$

Universal Quantifier

$$\forall \text{i:} \quad \frac{\Gamma \vdash A[y/x]}{\Gamma \vdash \forall x A}(y \notin \text{free}(\Gamma, \forall x A))$$

A special case:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}(x \notin \text{free}(\Gamma))$$

Universal Quantifier

Example: $\forall x(P(x) \vee Q(x)), \forall x(Q(x) \to R(x)) \vdash \forall x(P(x) \vee R(x))$

1.	$\Gamma \vdash \forall x(P(x) \vee Q(x))$	(Assm)
2.	$\Gamma \vdash P(x) \vee Q(x)$	($\forall$e 1)
3.	$\Gamma, P(x) \vdash P(x)$	(Assm)
4.	$\Gamma, P(x) \vdash P(x) \vee R(x)$	($\vee$i$_1$ 3)
5.	$\Gamma, Q(x) \vdash \forall x(Q(x) \to R(x))$	(Assm)
6.	$\Gamma, Q(x) \vdash Q(x) \to R(x)$	($\forall$e 5)
7.	$\Gamma, Q(x) \vdash Q(x)$	(Assm)
8.	$\Gamma, Q(x) \vdash R(x)$	($\to$e 6,7)
9.	$\Gamma, Q(x) \vdash P(x) \vee R(x)$	($\vee$i$_2$ 8)
10.	$\Gamma \vdash P(x) \vee R(x)$	($\vee$e 2,4,9)
11.	$\Gamma \vdash \forall x(P(x) \vee R(x))$	($\forall$i 10)

$\Gamma := \forall x(P(x) \vee Q(x)), \forall x(Q(x) \to R(x))$

Existential Quantifier

$$\exists\text{i:}\quad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

$$\exists\text{e:}\quad \frac{\Gamma \vdash \exists x A \qquad \Gamma, A[y/x] \vdash B}{\Gamma \vdash B}(y \notin \mathsf{free}(\Gamma, \exists x A, B))$$

A special case:

$$\frac{\Gamma \vdash \exists x A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}(x \notin \mathsf{free}(\Gamma, B))$$

Equality

$$=\mathrm{i}: \quad \frac{}{\vdash t = t}$$

$$=\mathrm{e}: \quad \frac{\Gamma \vdash t = t' \qquad \Gamma \vdash A[t/x]}{\Gamma \vdash A[t'/x]}$$

Example:

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash t_2 = t_1}$$

1.	$\Gamma \vdash t_1 = t_2$	(premise)
2.	$\vdash t_1 = t_1$	$(=\mathrm{i})$
3.	$\Gamma \vdash t_1 = t_1$	(Ant 2)
4.	$\Gamma \vdash (x = t_1)[t_1/x]$	(rep. 3, $x \notin \mathsf{free}(t_1)$)
5.	$\Gamma \vdash (x = t_1)[t_2/x]$	$(=\mathrm{e}$ 1,4)
6.	$\Gamma \vdash t_2 = t_1$	(rep. 5)

Some Important Equivalences

Remark: All following equivalences are also provable ($\dashv\vdash$)

$$\begin{aligned} \neg\forall x\varphi &\Leftrightarrow \exists x\neg\varphi \\ \neg\exists x\varphi &\Leftrightarrow \forall x\neg\varphi \end{aligned}$$

When $x \notin \mathsf{free}(\psi)$:

$$\begin{aligned} (\forall x\varphi) \wedge \psi &\Leftrightarrow \forall x(\varphi \wedge \psi) \\ (\forall x\varphi) \vee \psi &\Leftrightarrow \forall x(\varphi \vee \psi) \\ (\exists x\varphi) \wedge \psi &\Leftrightarrow \exists x(\varphi \wedge \psi) \\ (\exists x\varphi) \vee \psi &\Leftrightarrow \exists x(\varphi \vee \psi) \end{aligned}$$

Some Important Equivalences

$$(\forall x \varphi) \wedge (\forall x \psi) \Leftrightarrow \forall x (\varphi \wedge \psi)$$
$$(\exists x \varphi) \vee (\exists x \psi) \Leftrightarrow \exists x (\varphi \vee \psi)$$

$$\forall x \forall y \varphi \Leftrightarrow \forall y \forall x \varphi$$
$$\exists x \exists y \varphi \Leftrightarrow \exists y \exists x \varphi$$

When $y \notin \text{free}(\varphi)$:

$$\forall x \varphi \Leftrightarrow \forall y (\varphi[y/x])$$
$$\exists x \varphi \Leftrightarrow \exists y (\varphi[y/x])$$

When $x \notin \text{free}(\varphi)$:

$$\forall x \varphi \Leftrightarrow \varphi$$
$$\exists x \varphi \Leftrightarrow \varphi$$

There Exists Exactly One

Let us assume that:

- φ is a fomula
- variable y is a different variable than x
- $y \in \text{free}(\varphi)$

Proposition "there is exacly one element x, for which φ holds" can be represented by formula

$$\exists x(\varphi \wedge \forall y(\varphi[y/x] \rightarrow x = y))$$

It can be denoted also:

$$\exists_{=1} x\varphi$$

Remark: With the same meaning as $\exists_{=1}$, the following symbols are also used: $\exists_1$, $\exists^{=1}$, $\exists!$.

Natural Numbers

Signature $(\{<\}, \{\sigma, +, \cdot\}, \{0\}, \text{arity})$, where $\text{arity}(<) = 2$, $\text{arity}(\sigma) = 1$, $\text{arity}(+) = 2$, $\text{arity}(\cdot) = 2$

Examples of Formulas:

- $\forall x \exists y (x < y)$ — for each natural number, there exists a greater natural number
- $\sigma(0) + \sigma(0) = \sigma(\sigma(0))$ — it holds that $1 + 1 = 2$
- $\sigma(0) < x \wedge \neg \exists y \exists z (\sigma(0) < y \wedge \sigma(0) < z \wedge y \cdot z = x)$ — formula with free variable x, representing proposition that x je a prime number

Natural Numbers — Robinson and Peano Arithmetic

Axioms:

- $\forall x(\sigma(x) \neq 0)$
- $\forall x \forall y(\sigma(x) = \sigma(y) \rightarrow x = y)$
- $\forall x(x = 0 \vee \exists y(\sigma(y) = x))$
- $\forall x(x + 0 = x)$
- $\forall x \forall y(x + \sigma(y) = \sigma(x + y))$
- $\forall x(x \cdot 0 = 0)$
- $\forall x \forall y(x \cdot \sigma(y) = (x \cdot y) + x)$
- $\forall x \forall y(x < y \leftrightarrow \exists z(\sigma(z) + x = y))$

Induction axiom schema:

- $\forall y_1 \cdots \forall y_n(\varphi[0/x] \wedge \forall x(\varphi \rightarrow \varphi[\sigma(x)/x]) \rightarrow \forall x \varphi)$

— φ is an arbitrary formula, where $\text{free}(\varphi) \subseteq \{x, y_1, \ldots, y_n\}$

Definitions

Let us assume a given set of axioms Γ.

- The definition of a predicate symbol P, where $\text{arity}(P) = n$:

$$\forall x_1 \cdots \forall x_n \, (P(x_1, \ldots, x_n) \ \leftrightarrow \ \varphi)$$

  where $\text{free}(\varphi) \subseteq \{x_1, \ldots, x_n\}$

- The definition of a function symbol f, where $\text{arity}(f) = n$:

$$\forall x_1 \cdots \forall x_n \forall y \, (f(x_1, \ldots, x_n) = y \ \leftrightarrow \ \varphi)$$

  where $\text{free}(\varphi) \subseteq \{x_1, \ldots, x_n, y\}$, and where $\Gamma \vdash \forall x_1 \cdots \forall x_n \exists_{=1} y(\varphi)$

- The definition of a constant symbol c:

$$\varphi[c/x]$$

  where $\text{free}(\varphi) \subseteq \{x\}$, and where $\Gamma \vdash \exists_{=1} x(\varphi)$

Group Theory

Signature $(\emptyset, \{\circ\}, \{e\}, \text{arity})$, where $\text{arity}(\circ) = 2$

Axioms:

- $\forall x \forall y \forall z((x \circ y) \circ z = x \circ (y \circ z))$
- $\forall x(x \circ e = x)$
- $\forall x \exists y(x \circ y = e)$

Set Theory

Signature $(\{\in\}, \emptyset, \emptyset, \text{arity})$, where $\text{arity}(\in) = 2$

Examples of some of axioms of set theory:

- **Axiom of extensionality**:

$$\forall x \forall y (\forall z (z \in x \leftrightarrow z \in y) \rightarrow x = y)$$

- **Axiom schema of specification**:

$$\forall y \exists z \forall x (x \in z \leftrightarrow (x \in y \wedge \varphi))$$

where φ is an arbitrary formula, where $z \notin \text{free}(\varphi)$

The set of those elements x of set y, for which it holds that φ:

$$\{x \in y \mid \varphi\}$$

Often Used Abbreviations

- Proposition "φ holds for some element x of set A":

$$\exists x(x \in A \ \wedge \ \varphi)$$

Abbreviation:

$$(\exists x \in A)(\varphi)$$

- Proposition "φ holds for each element x of set A":

$$\forall x(x \in A \ \rightarrow \ \varphi)$$

Abbreviation:

$$(\forall x \in A)(\varphi)$$

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: HELLO ABRACADABRA ERROR

Alphabet and Word

Example 2:

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

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

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 $\Sigma_5 = \{a, b\}$: *aababb*, *abbabbba*, *aaab*

Alphabet and Word

Example 6:

Alphabet Σ_6 is the set of all ASCII characters.

Example of a word:

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

class␣HelloWorld␣{ ↩ ␣␣␣␣public␣static␣void␣main(Str ···

Theory of Formal Languages – Motivation

Language — a set of (some) words of symbols from a given alphabet

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

Representation of Formal Languages

To describe a language, there are several possibilities:

- We can enumerate all words of the language (however, this is possible only for small finite languages).

  Example: $L = \{aab, babba, aaaaaa\}$

- We can specify a property of the words of the language:

  Example: The language over alphabet $\{0, 1\}$ containing all words with even number of occurrences of symbol 1.

Representation of Formal Languages

In particular, the following two approaches are used in the theory of formal languages:

- To describe an (idealized) machine, device, algorithm, that recognizes words of the given language – approaches based on **automata**.
- To describe some mechanism that allows to generate all words of the given language – approaches based on **grammars** or **regular expressions**.

Some Basic Concepts

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

For example, the length of word $abaab$ is 5.

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

For example, if $w = abaab$ then $|w| = 5$.

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

For word $w = ababb$ we have $|w|_a = 2$ and $|w|_b = 3$.

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.

(Remark: In literature, sometimes λ (lambda) is used to denote the empty word instead of ε .)

$$|\varepsilon| = 0$$

Concatenation of Words

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

For example, the concatenation of words `OST` and `RAVA` is the word `OSTRAVA`.

The operation of concatenation is denoted by symbol $\cdot$ (similarly to multiplication). It is possible to omit this symbol.

$$\texttt{OST} \cdot \texttt{RAVA} = \texttt{OSTRAVA}$$

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 $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$.

Concatenation of Words

Concatenation is not **commutative**, i.e., the following equality does not hold in general

$$u \cdot v = v \cdot u$$

Example:

$$\texttt{OST} \cdot \texttt{RAVA} \neq \texttt{RAVA} \cdot \texttt{OST}$$

It is obvious that the following holds for any words v and w:

$$|v \cdot w| = |v| + |w|$$

For every word w we also have:

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

Prefixes, Suffixes, and Subwords

Definition

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

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

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

Example:

- Prefixes of the word abaab are ε, a, ab, aba, abaa, abaab.
- Suffixes of the word abaab are ε, b, ab, aab, baab, abaab.
- Subwords of the word abaab are ε, a, b, ab, ba, aba, baa, aab, abaa, baab, abaab.

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.

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: It is assumed the languages involved in these operations use the same 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: $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \notin L_2\}$

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

Set Operations on Languages

Example:

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

- L_1 — the set of all words containing subword $\mathtt{baa}$
- L_2 — the set of all words with an even number of occurrences of symbol $\mathtt{b}$

Then

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

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

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

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

Example:

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

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

$$abba \qquad abbab \qquad abbbbb \qquad baa \qquad baab \qquad babbb$$

Iteration of a Language

Definition

The **iteration (Kleene star) of language** L, denoted L^*, is the language consisting of words created by concatenation of some arbitrary number of words from language L.
I.e. $w \in L^*$ iff

$$\exists n \in \mathbb{N} : \exists w_1, w_2, \ldots, w_n \in L : w = w_1 w_2 \cdots w_n$$

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

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

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 – Alternative Definition

At first, for a language L and a number $k \in \mathbb{N}$ we define the language L^k:

$$L^0 = \{\varepsilon\}, \qquad L^k = L^{k-1} \cdot L \quad \text{for } k \geq 1$$

This means

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

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

$$aaaaaa \quad aaaab \quad aabaa \quad aabb \quad baaaa \quad baab \quad bbaa \quad bbb$$

Iteration of a Language – Alternative Definition

Alternative definition

The **iteration (Kleene star) of 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

Remark: Sometimes, notation L^+ is used as an abbreviation for $L \cdot L^*$, i.e.,

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

Reverse

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: $w = \texttt{HELLO}$ $w^R = \texttt{OLLEH}$

Formally, for $w = a_1 a_2 \cdots a_n$ (where $a_i \in \Sigma$) is $w^R = a_n a_{n-1} \cdots a_1$.

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

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

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

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

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

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 $<_L$.

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: abeceda $\Sigma = \{0, 1\}$

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

Regular Expressions

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

- 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: abeceda $\Sigma = \{0, 1\}$

- 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: abeceda $\Sigma = \{0, 1\}$

- 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: abeceda $\Sigma = \{0, 1\}$

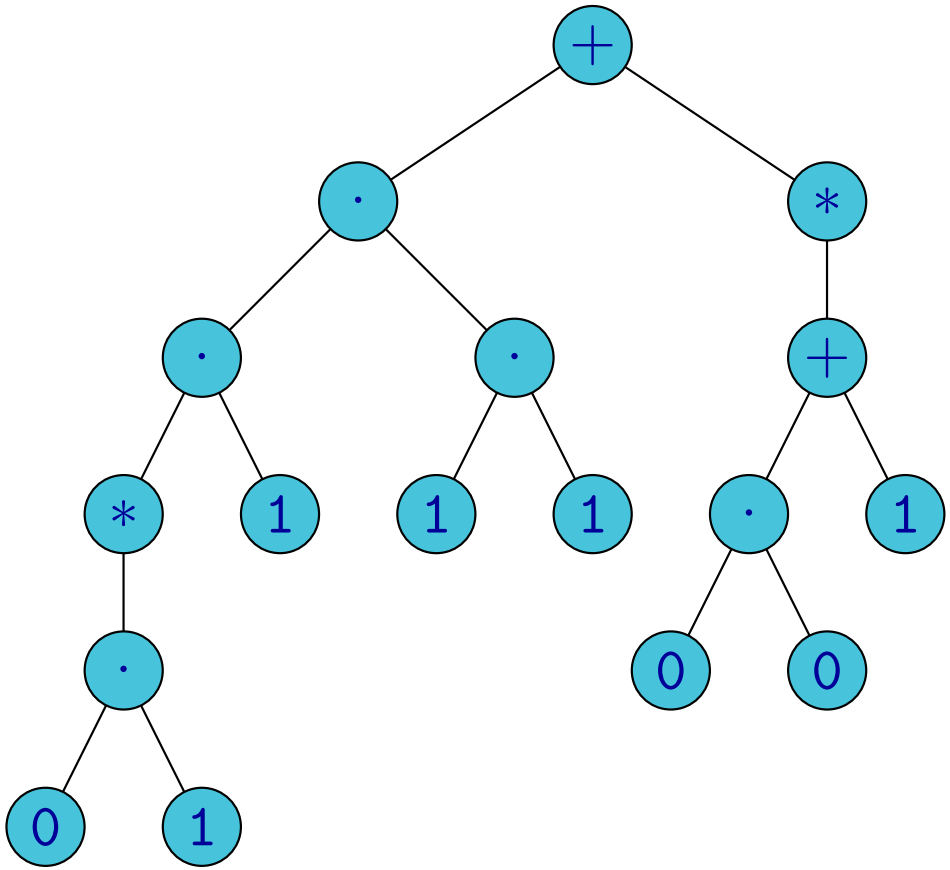
- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, (0^*) is also a regular expression.
- Since $(0 + 1)$ and (0^*) are regular expressions, $((0 + 1) \cdot (0^*))$ is also a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

Remark: If α is a regular expression, by $[\alpha]$ we denote the language defined by the regular expression α.

$$[((0 + 1) \cdot (0^*))] = \{0,\ 1,\ 00,\ 10,\ 000,\ 100,\ 0000,\ 1000,\ 00000,\ \dots\}$$

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

A description of an (abstract) syntax of regular expressions using Backus-Naur form:

$$\alpha \ ::= \ \emptyset \ \mid \ \varepsilon \ \mid \ a \ \mid \ \alpha^* \ \mid \ \alpha \cdot \alpha \ \mid \ \alpha + \alpha$$

The formal definition of semantics of regular expressions:

- $[\emptyset] = \emptyset$
- $[\varepsilon] = \{\varepsilon\}$
- $[a] = \{a\}$
- $[\alpha^*] = [\alpha]^*$
- $[\alpha \cdot \beta] = [\alpha] \cdot [\beta]$
- $[\alpha + \beta] = [\alpha] \cup [\beta]$

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 $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

$0 + 1$ ... the language containing two words 0 and 1

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

$0 + 1$ ... the language containing two words 0 and 1

0^* ... the language containing words ε, 0, 00, 000, ...

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

$0 + 1$ ... the language containing two words 0 and 1

0^* ... the language containing words ε, 0, 00, 000, ...

$(01)^*$ ... the language containing words ε, 01, 0101, 010101, ...

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

$0 + 1$ ... the language containing two words 0 and 1

0^* ... the language containing words ε, 0, 00, 000, ...

$(01)^*$ ... the language containing words ε, 01, 0101, 010101, ...

$(0 + 1)^*$ ... the language containing all words over the alphabet $\{0, 1\}$

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

$0 + 1$ ... the language containing two words 0 and 1

0^* ... the language containing words ε, 0, 00, 000, ...

$(01)^*$ ... the language containing words ε, 01, 0101, 010101, ...

$(0 + 1)^*$ ... the language containing all words over the alphabet $\{0, 1\}$

$(0 + 1)^*00$ ... the language containing all words ending with 00

Regular Expressions

Examples: In all examples $\Sigma = \{0, 1\}$.

0 ... the language containing the only word 0

01 ... the language containing the only word 01

$0 + 1$ ... the language containing two words 0 and 1

0^* ... the language containing words ε, 0, 00, 000, ...

$(01)^*$ ... the language containing words ε, 01, 0101, 010101, ...

$(0 + 1)^*$ ... the language containing all words over the alphabet $\{0, 1\}$

$(0 + 1)^*00$ ... the language containing all words ending with 00

$(01)^*111(01)^*$ ... the language containing all words that contain a subword 111 preceded and followed by an arbitrary number of copies of the word 01

Regular Expressions

$(0+1)^*00 + (01)^*111(01)^*$ ... the language containing all words that either end with 00 or contain a subwords 111 preceded and followed with some arbitrary number of words 01

Regular Expressions

$(0+1)^*00+(01)^*111(01)^*$ ... the language containing all words that either end with 00 or contain a subwords 111 preceded and followed with some arbitrary number of words 01

$(0+1)^*1(0+1)^*$ ... the language of all words that contain at least one occurrence of symbol 1

Regular Expressions

$(0+1)^*00+(01)^*111(01)^*$ ... the language containing all words that either end with 00 or contain a subwords 111 preceded and followed with some arbitrary number of words 01

$(0+1)^*1(0+1)^*$ ... the language of all words that contain at least one occurrence of symbol 1

$0^*(10^*10^*)^*$ ... the language containg all words with an even number of occurrences of symbol 1

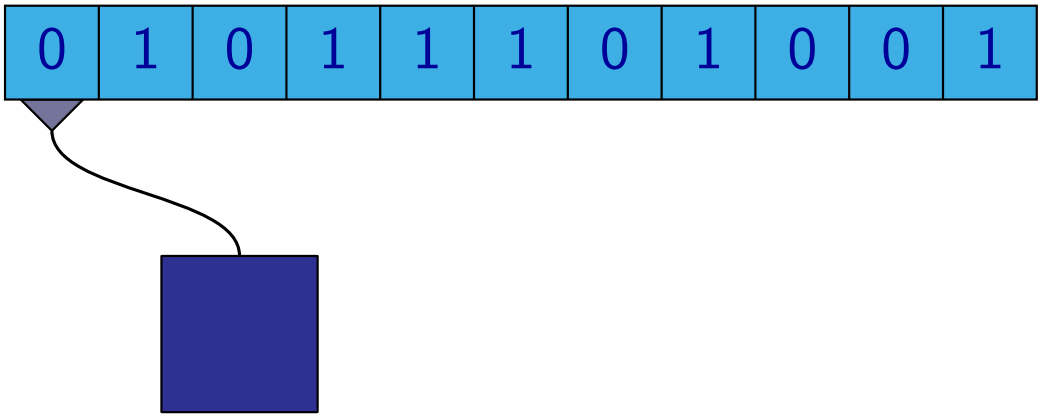
Finite Automata

Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

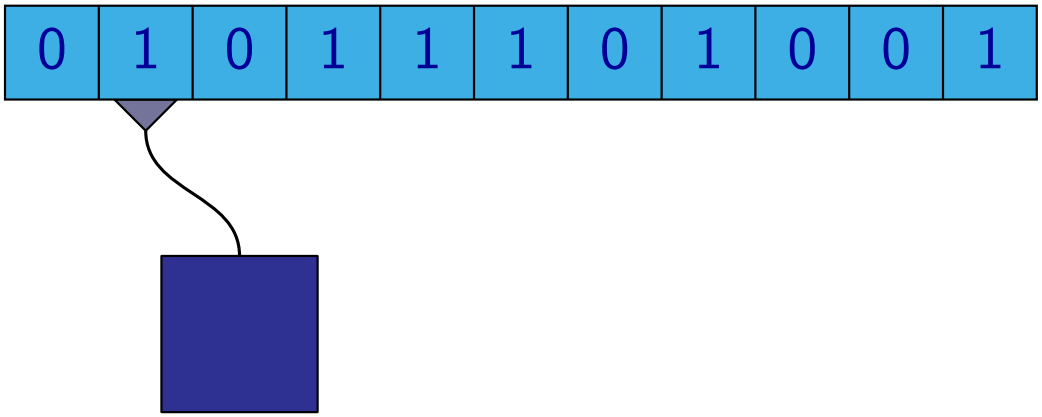


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

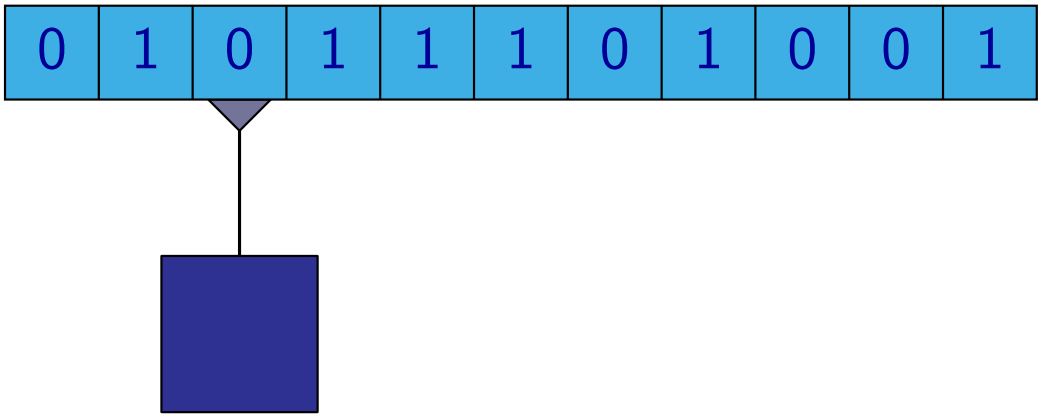


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

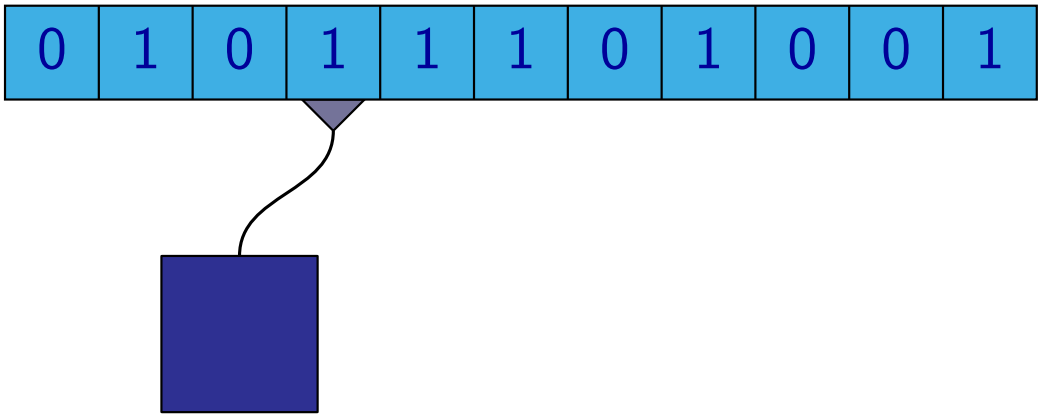


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

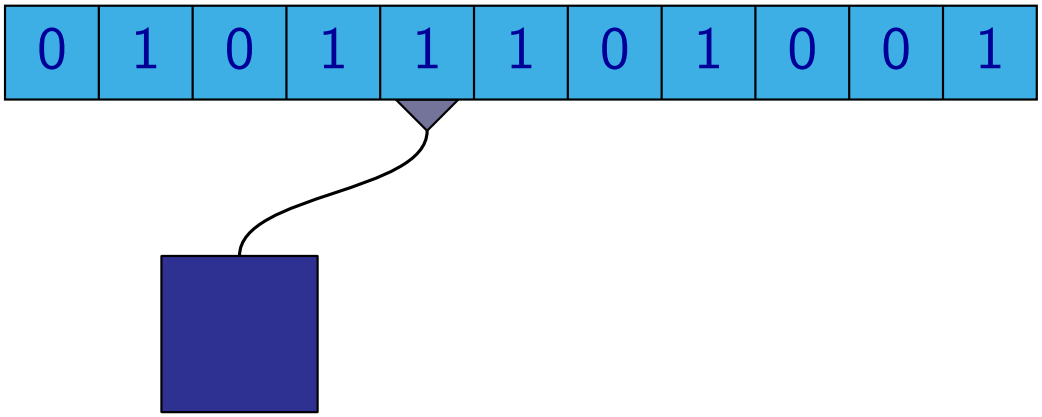


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

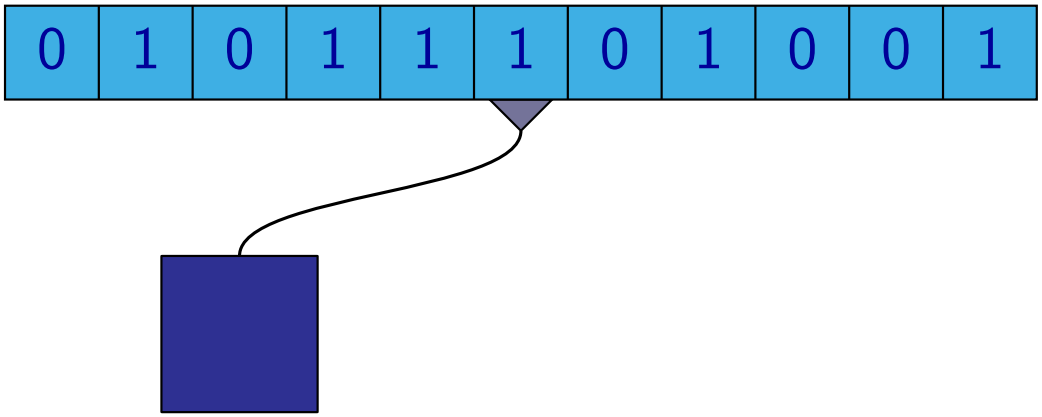


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

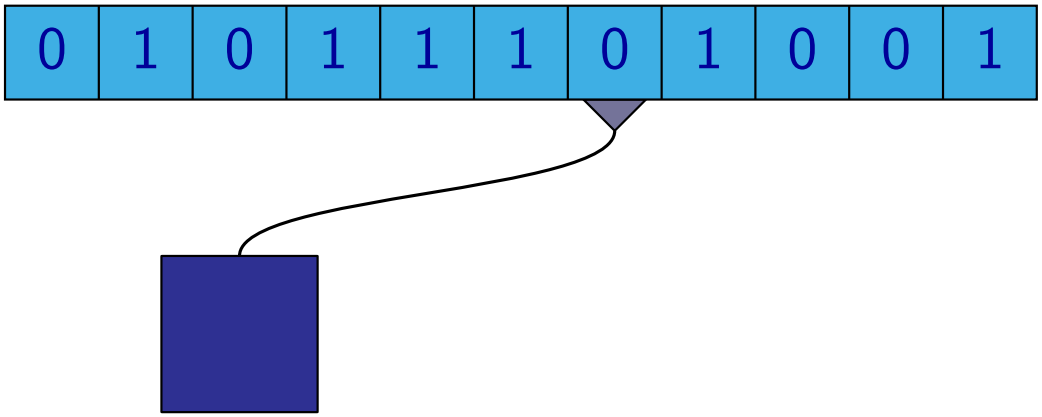


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

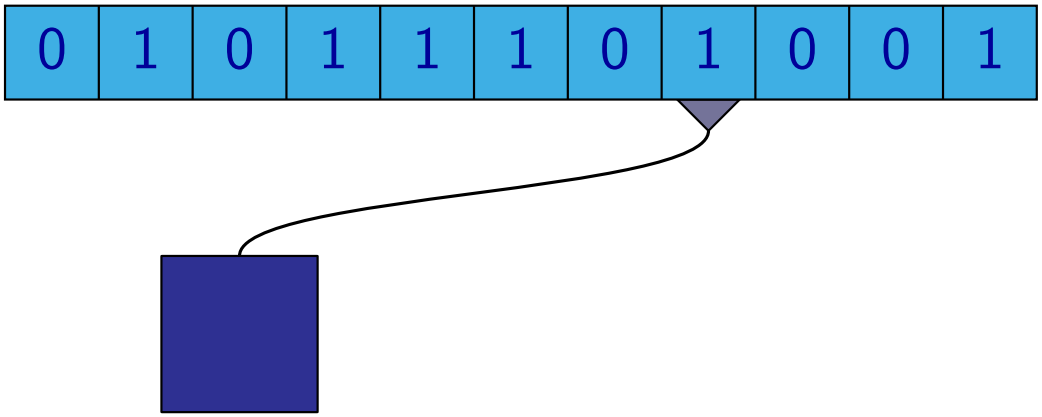


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

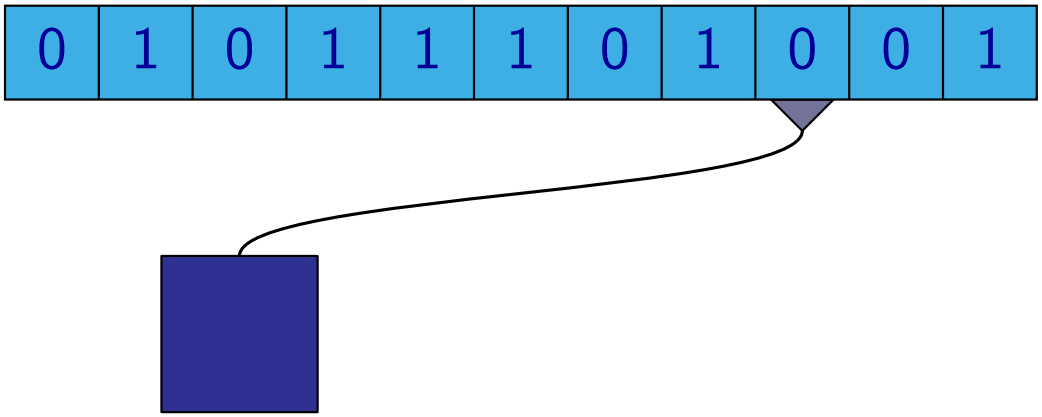


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

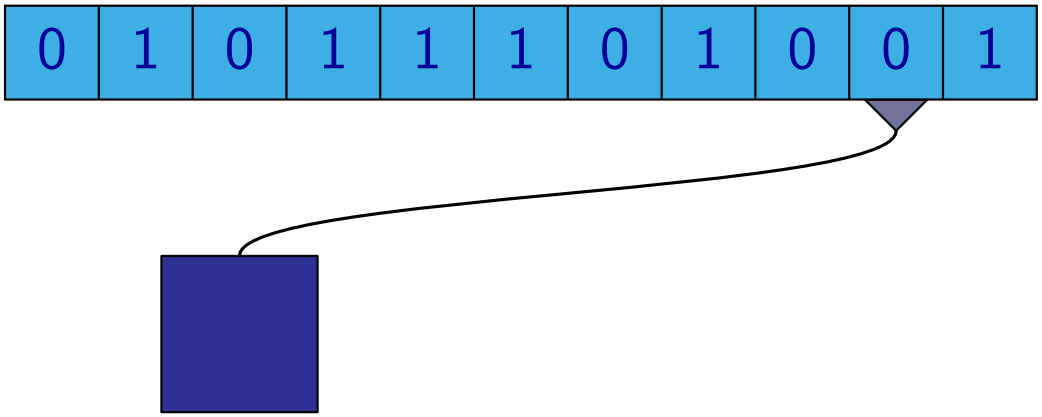


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

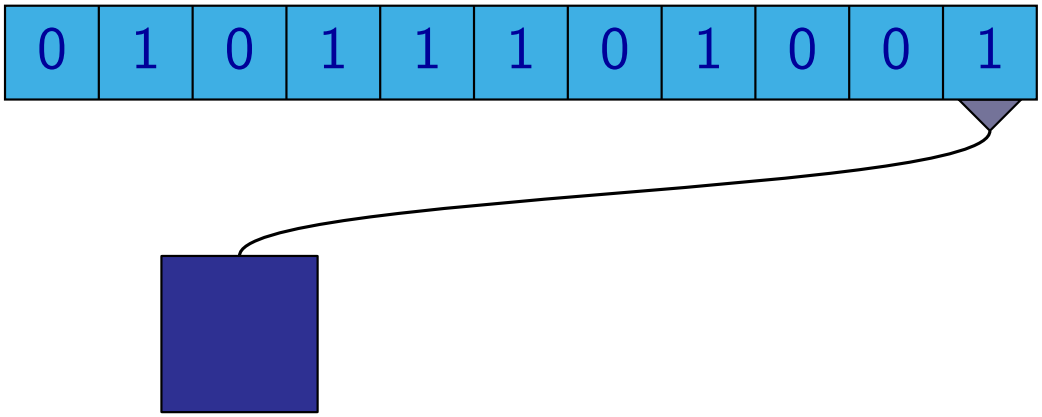


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

We would like to recognize a language L consisting of words with even number of symbols 1.

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

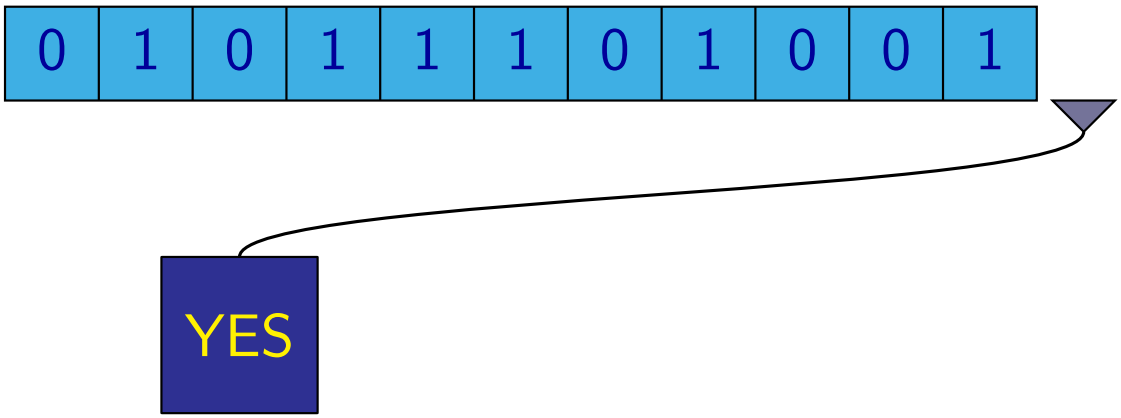


Recognition of a Language

Example: Consider words over alphabet $\{0, 1\}$.

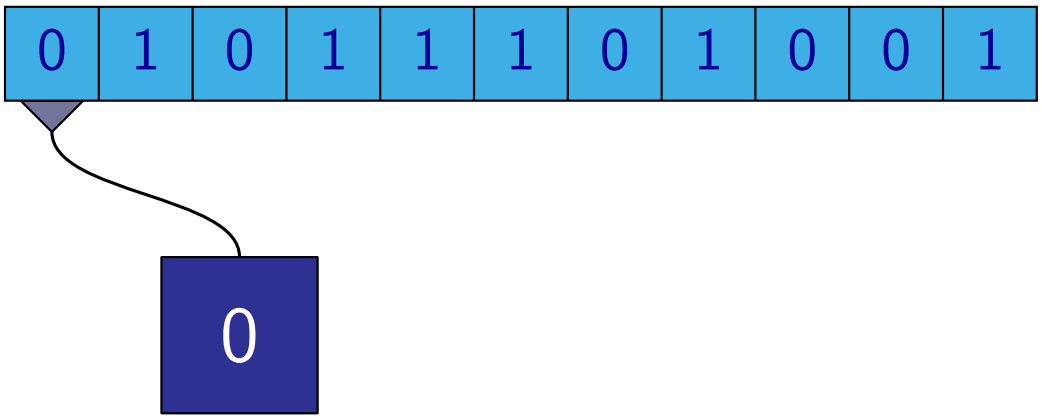
We would like to recognize a language L consisting of words with even number of symbols 1.

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



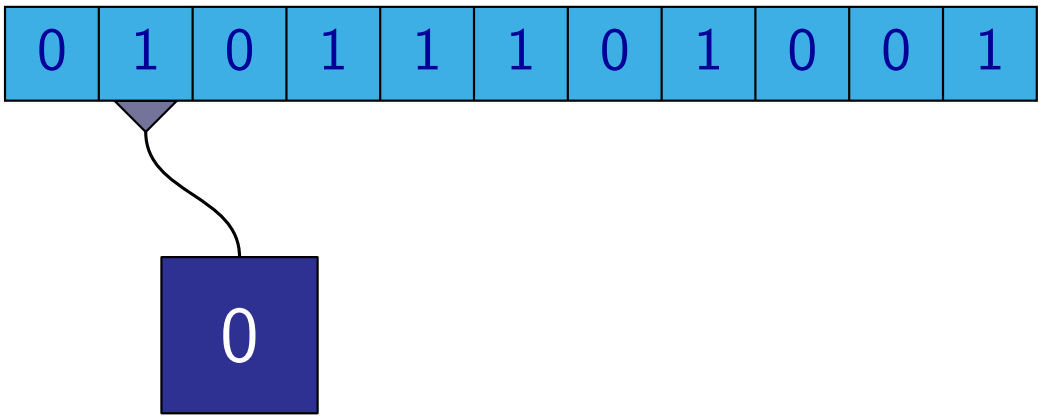
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



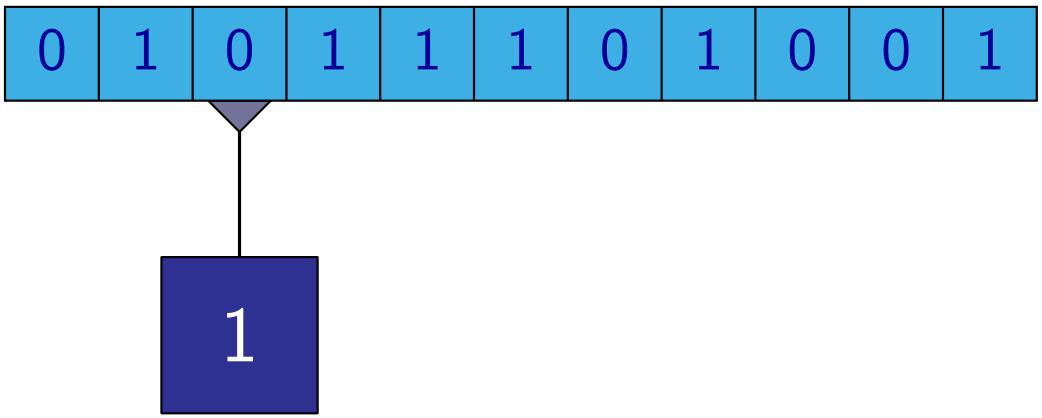
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



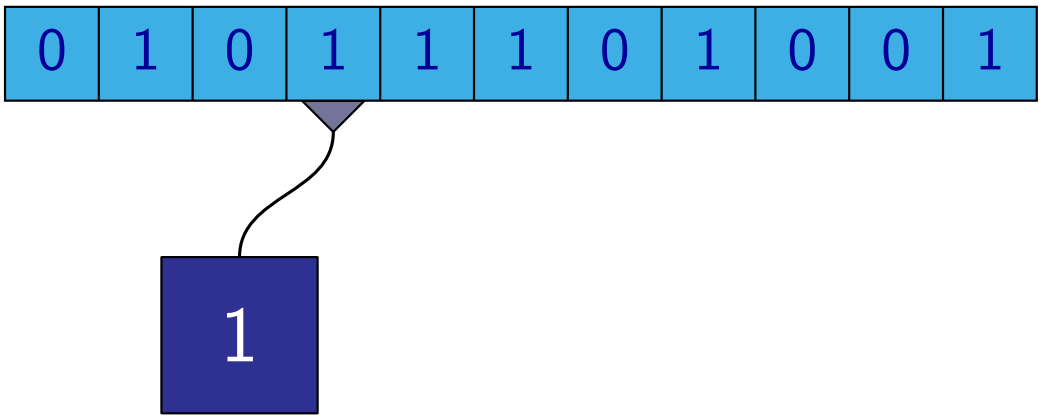
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



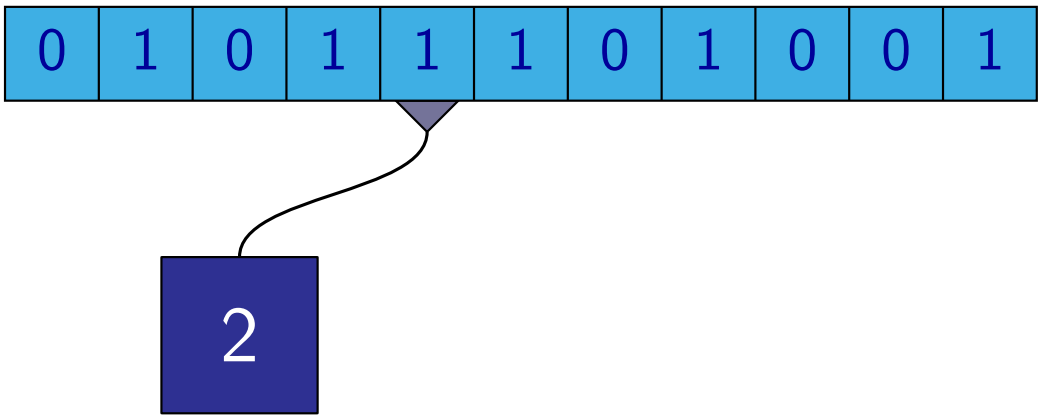
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



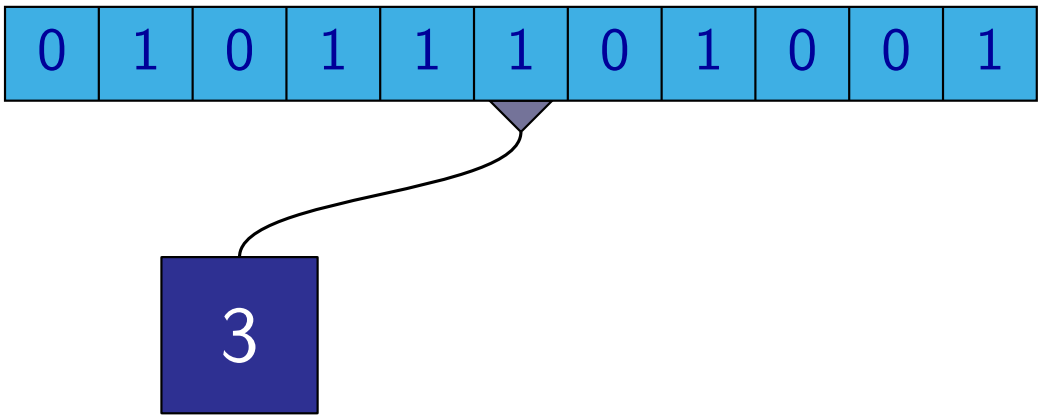
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



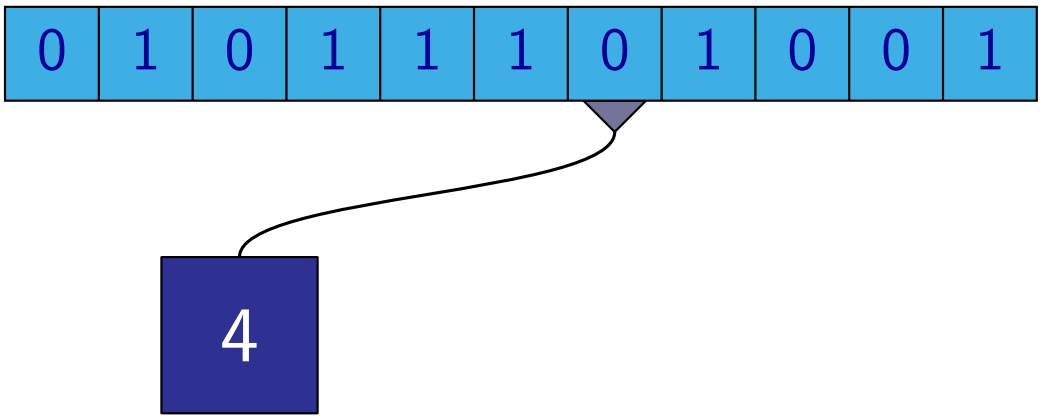
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



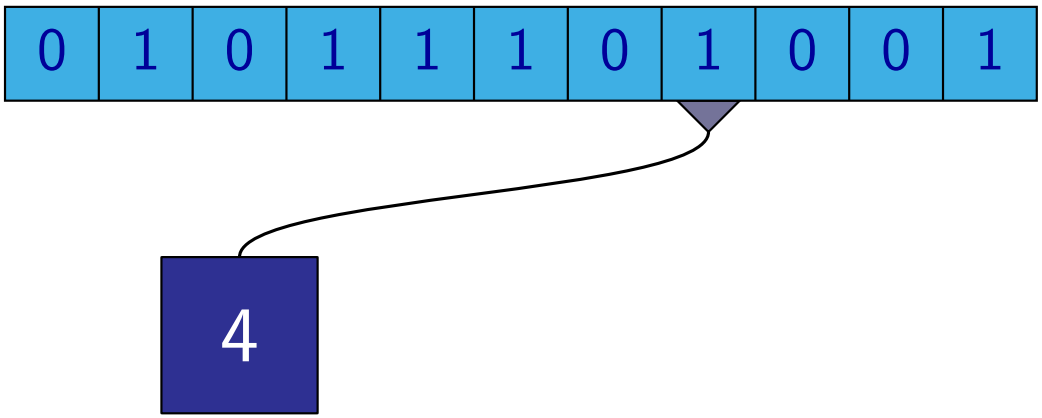
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



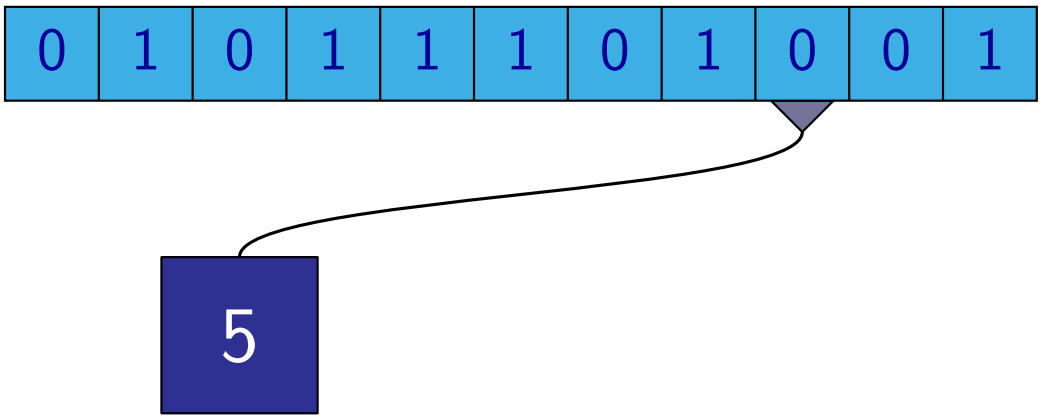
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



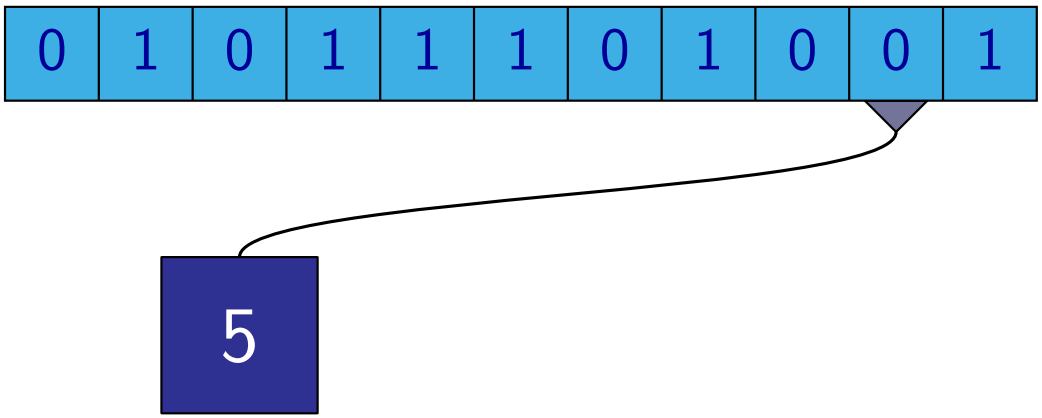
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



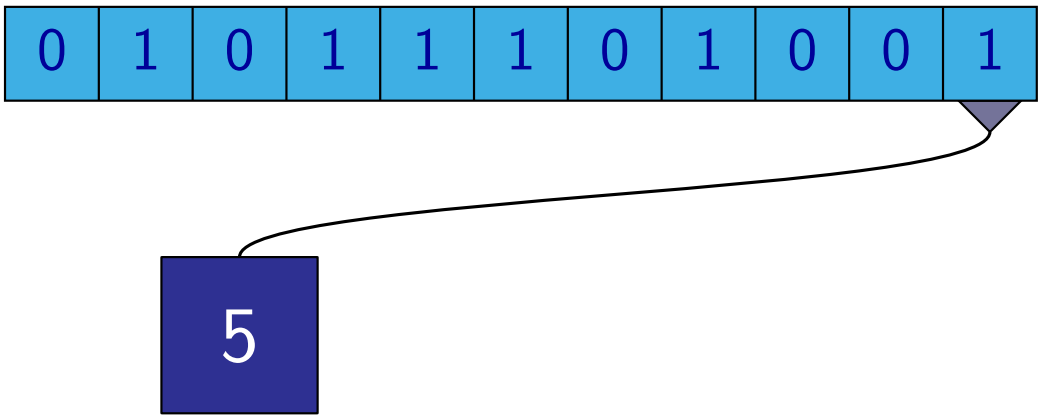
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



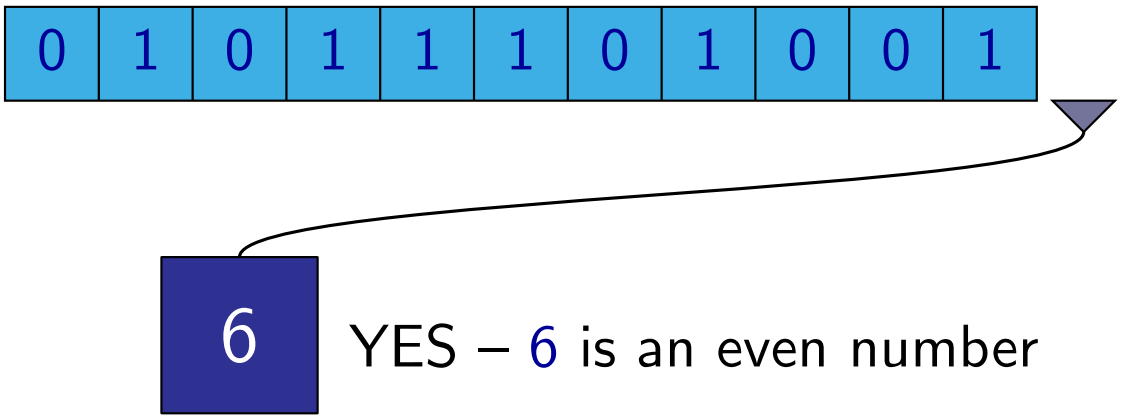
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



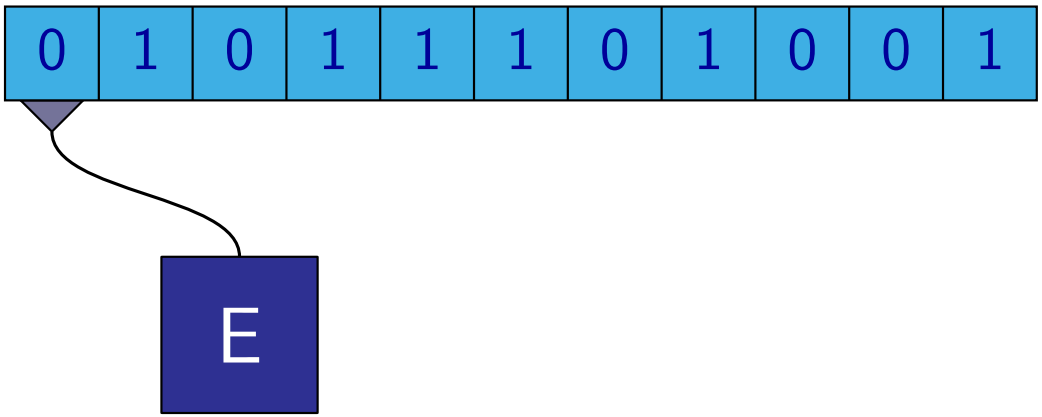
Recognition of a Language

The first idea: To count the number of occurrences of symbol 1.



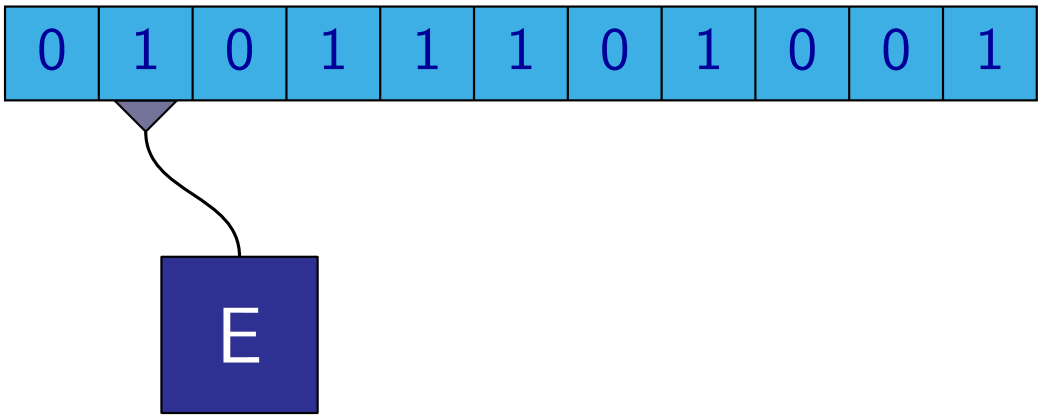
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



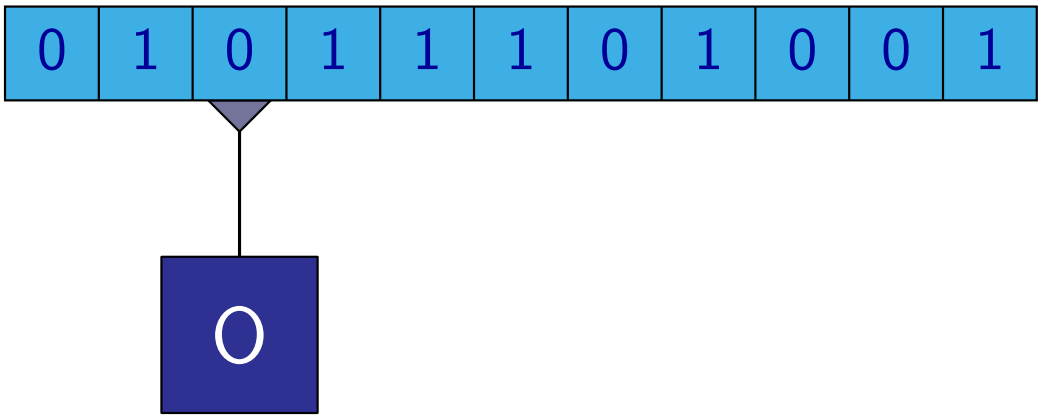
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



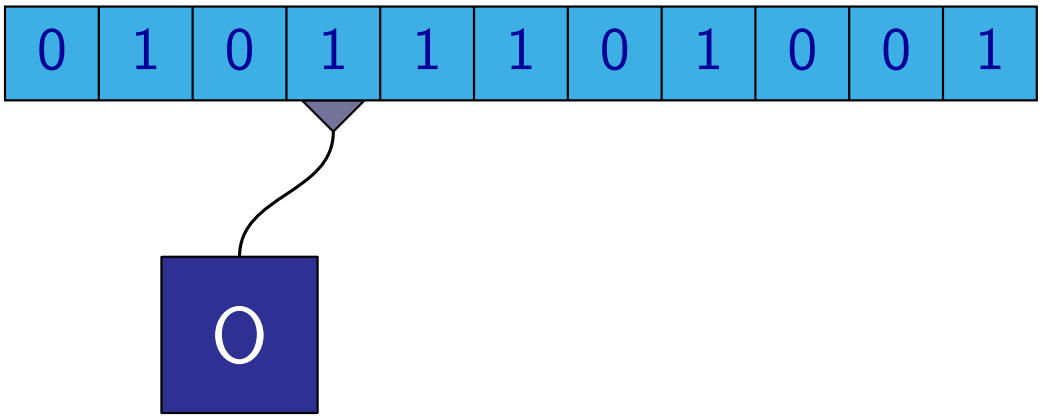
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



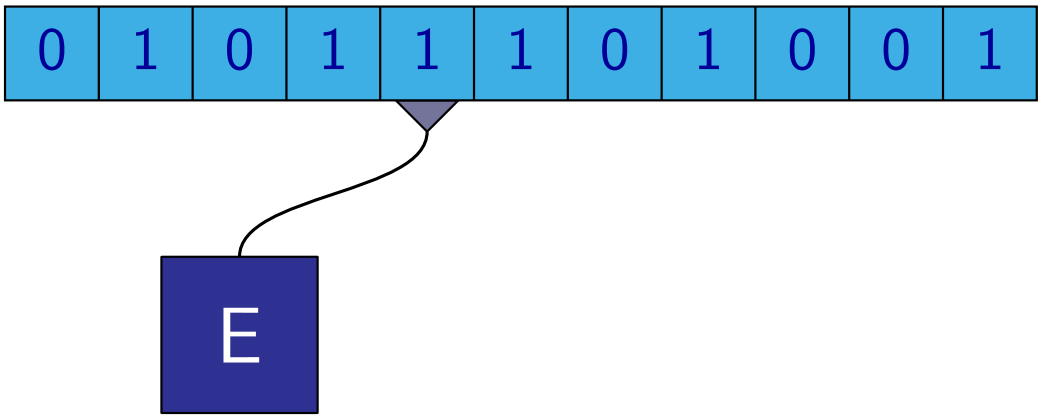
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



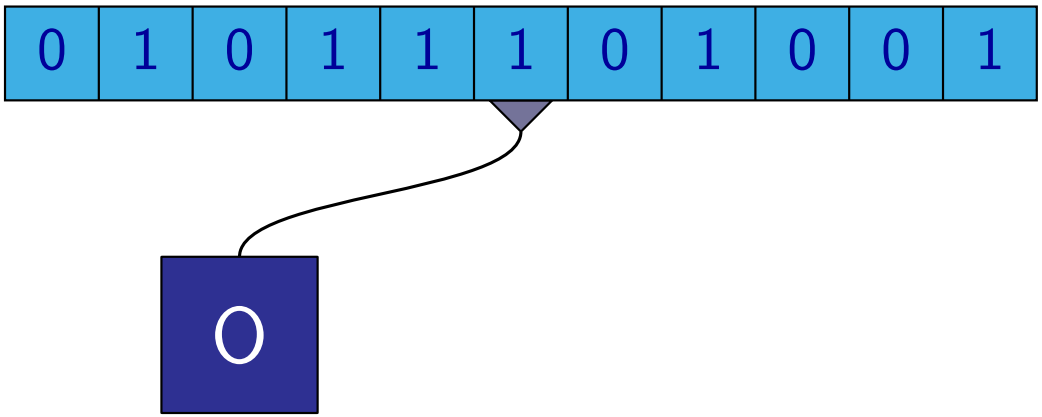
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



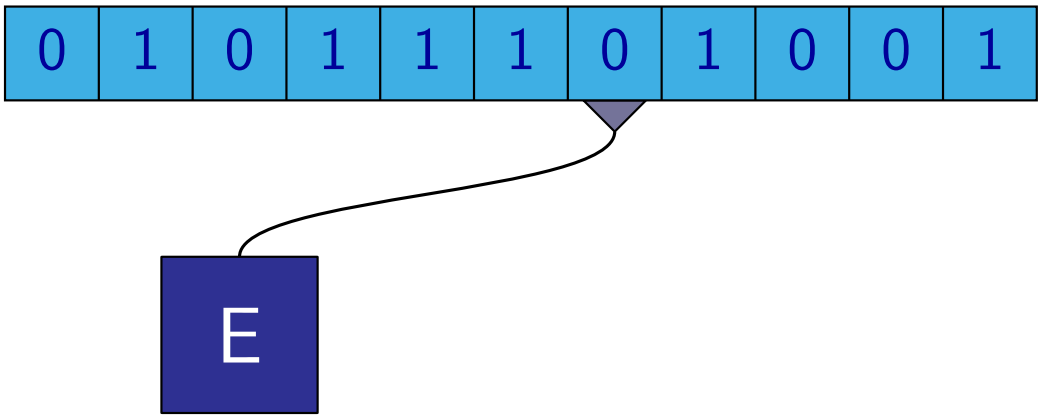
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



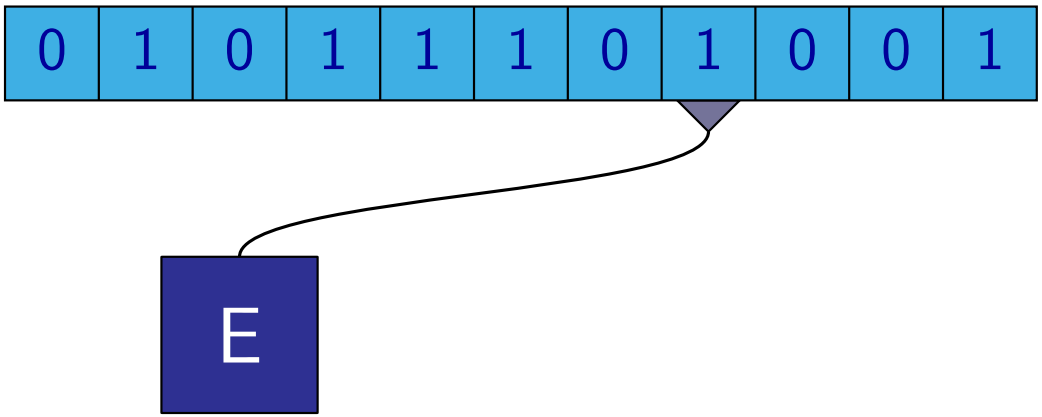
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



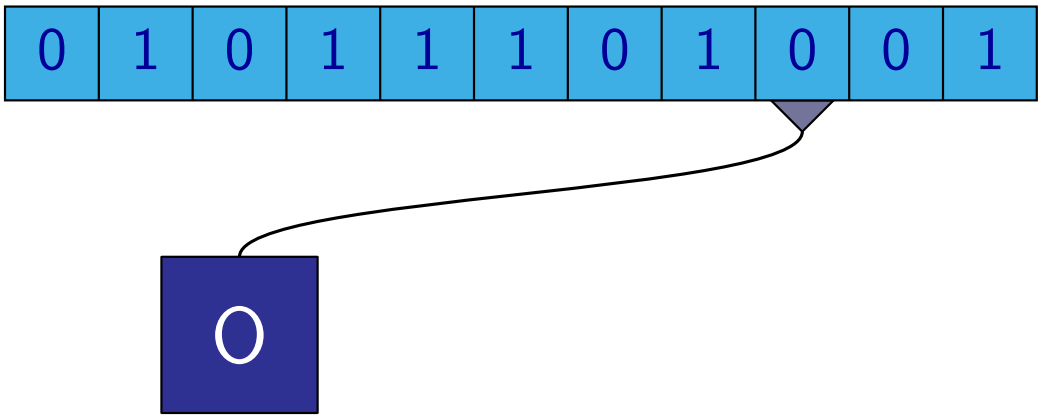
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



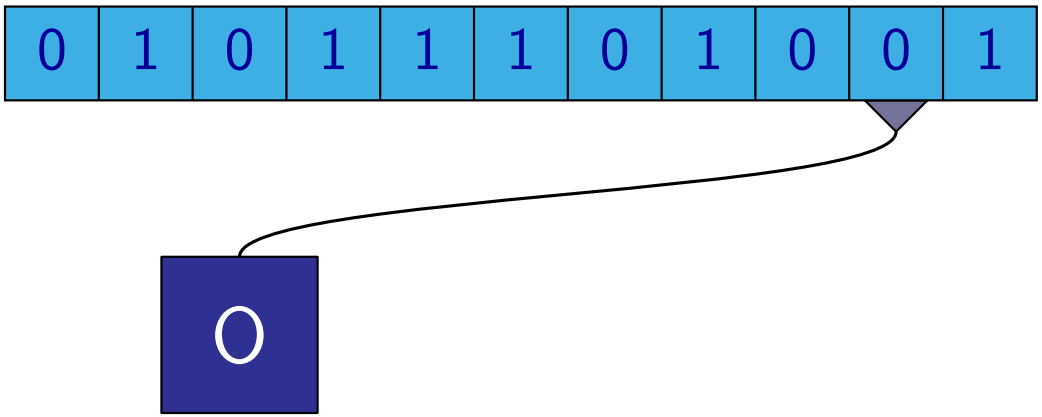
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



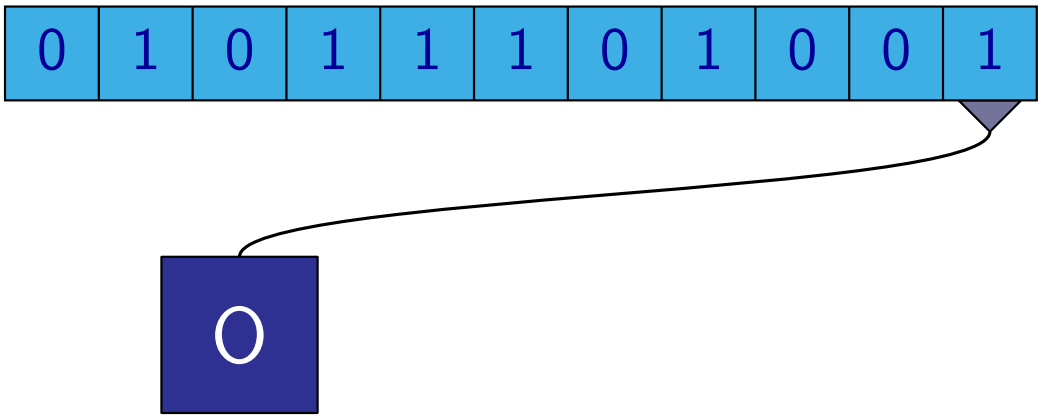
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).



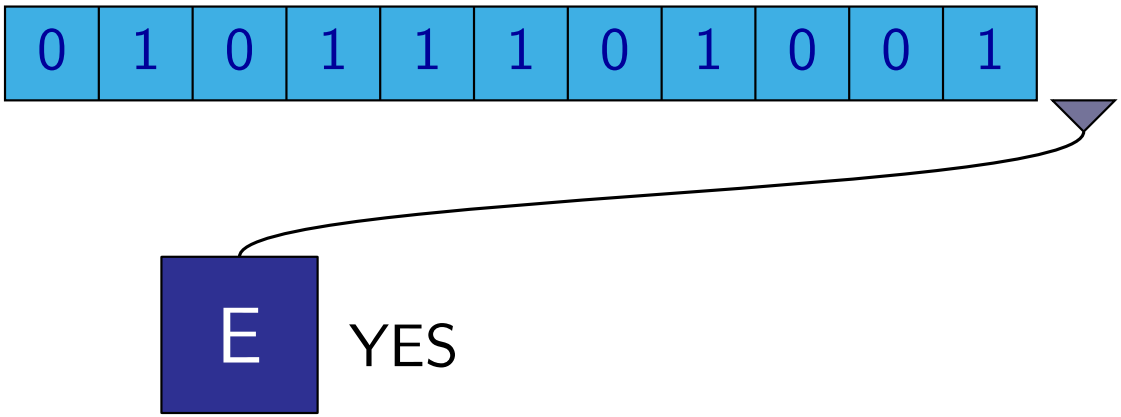
Recognition of a Language

The second idea: In fact, we just need to remember if the number of symbols 1 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 1 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

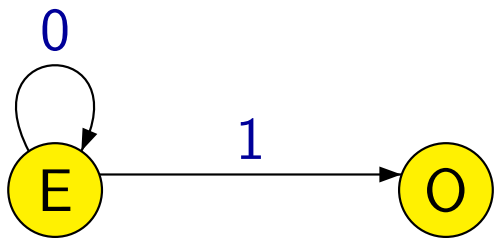
The behaviour of the device can be described by the following graph:

Recognition of a Language

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

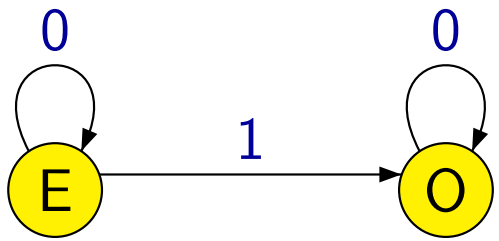
Recognition of a Language

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



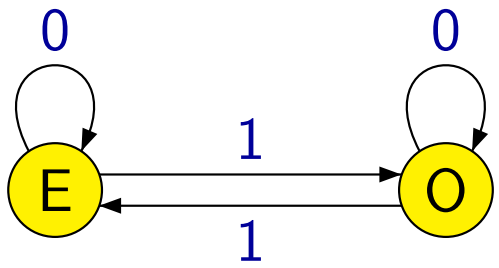
Recognition of a Language

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



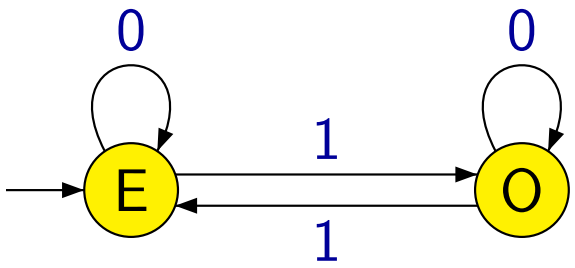
Recognition of a Language

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



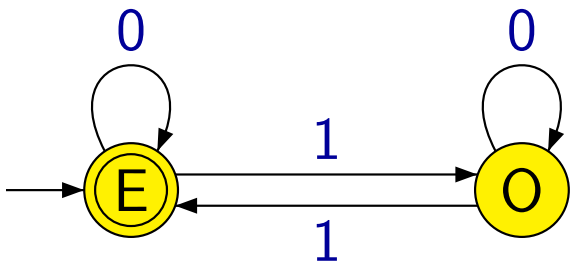
Recognition of a Language

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



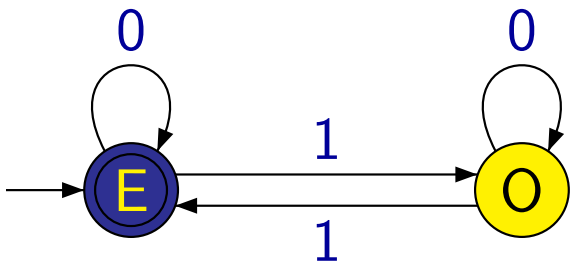
Recognition of a Language

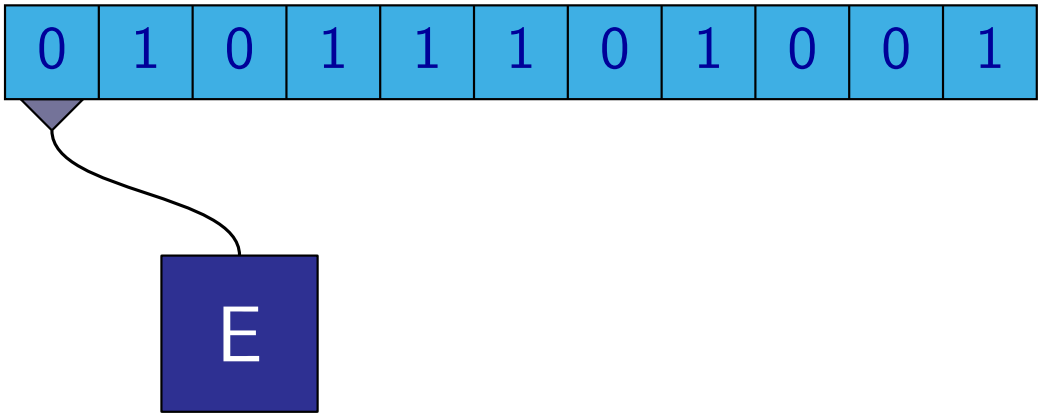
The behaviour of the device can be described by the following graph:



Recognition of a Language

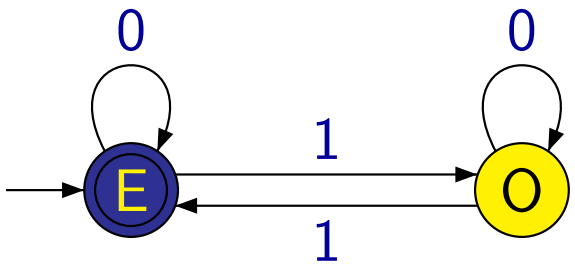
The behaviour of the device can be described by the following graph:

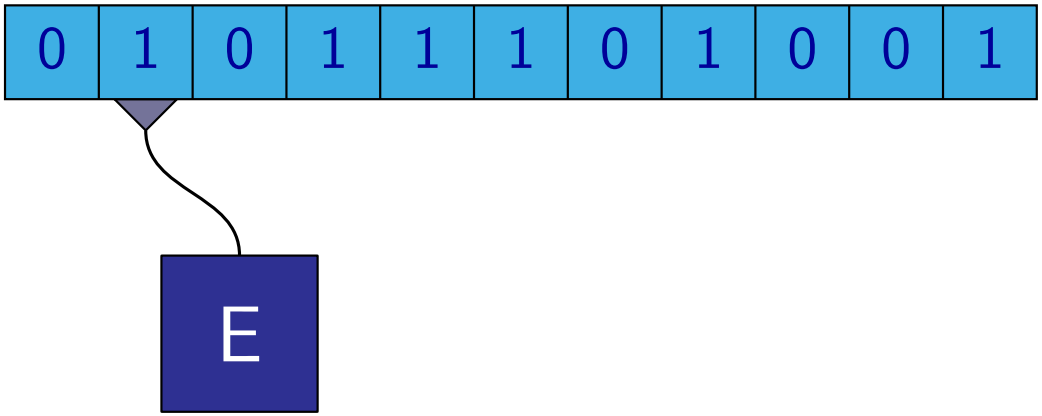




Recognition of a Language

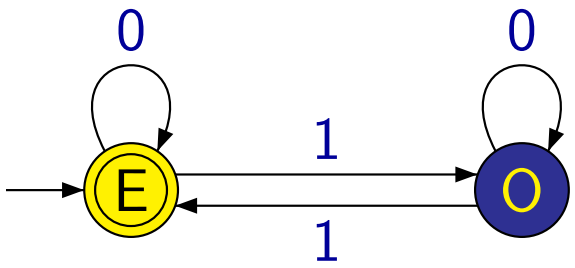
The behaviour of the device can be described by the following graph:

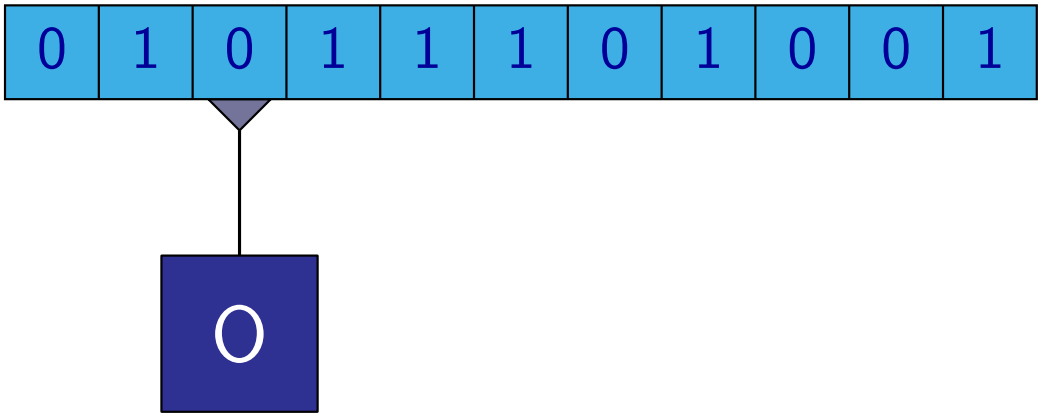




Recognition of a Language

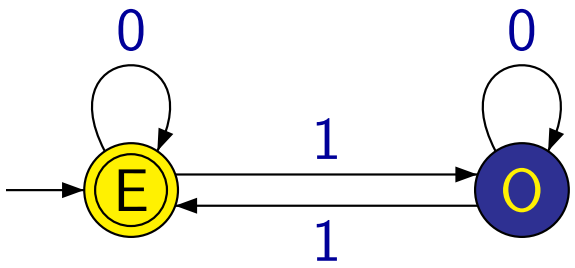
The behaviour of the device can be described by the following graph:

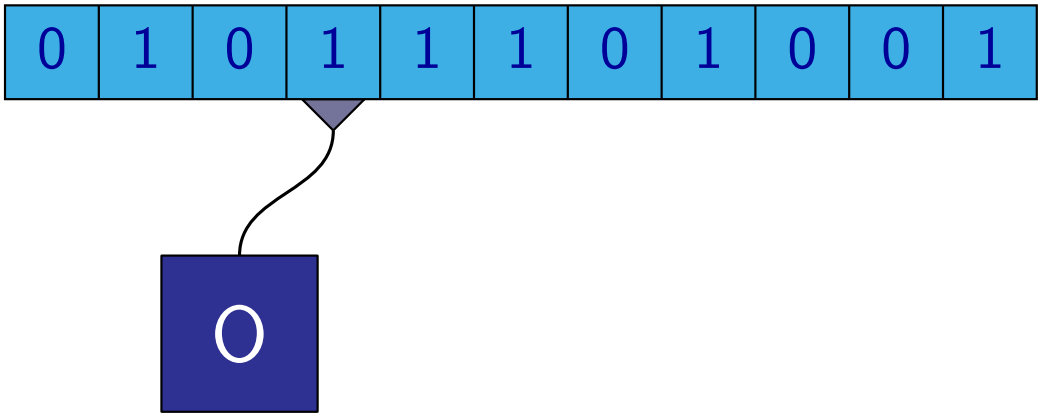




Recognition of a Language

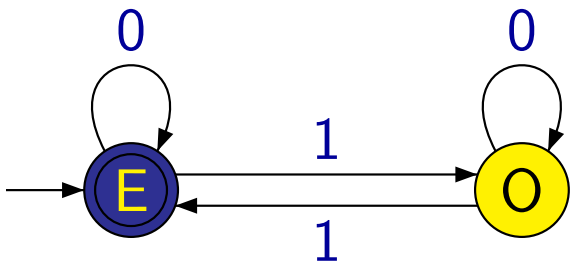
The behaviour of the device can be described by the following graph:

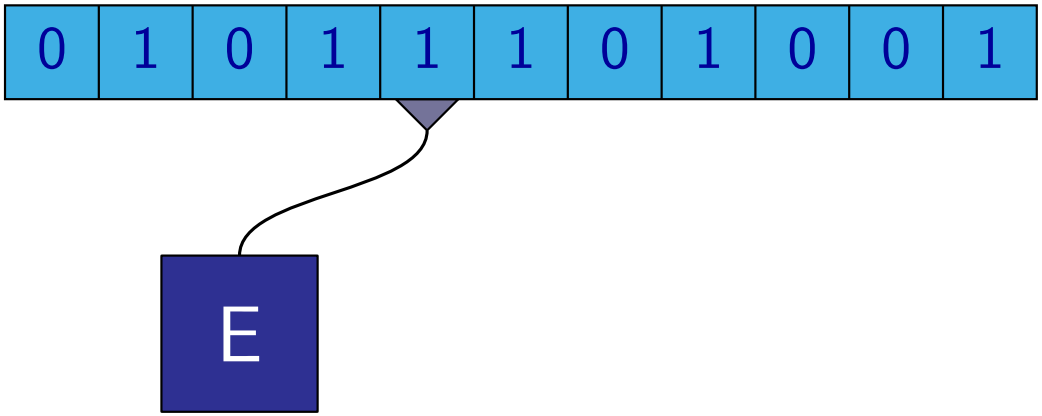




Recognition of a Language

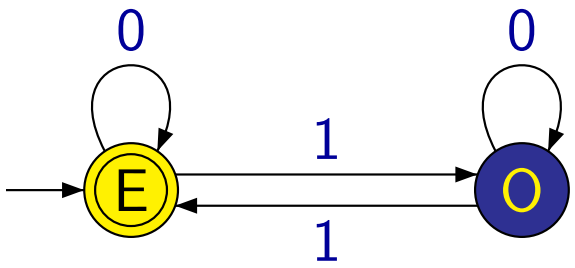
The behaviour of the device can be described by the following graph:

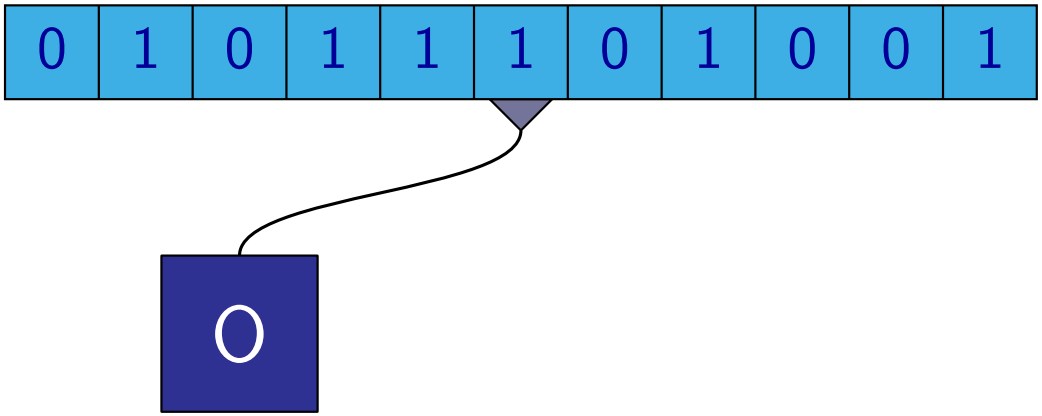




Recognition of a Language

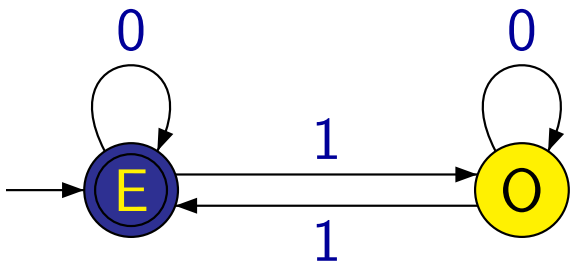
The behaviour of the device can be described by the following graph:

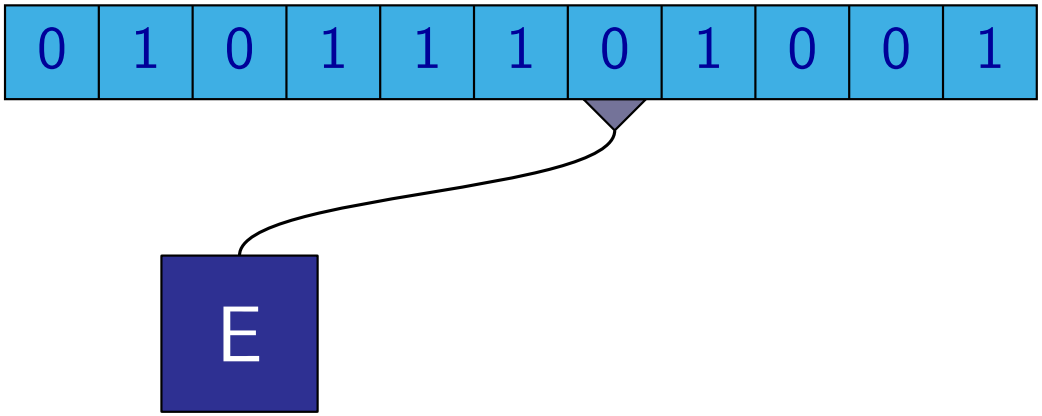




Recognition of a Language

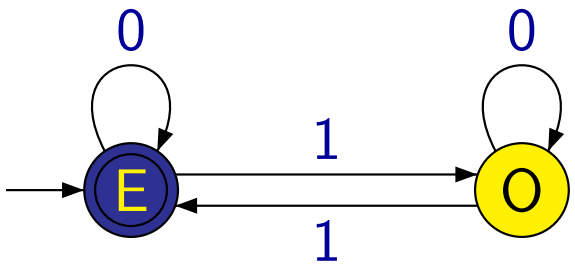
The behaviour of the device can be described by the following graph:

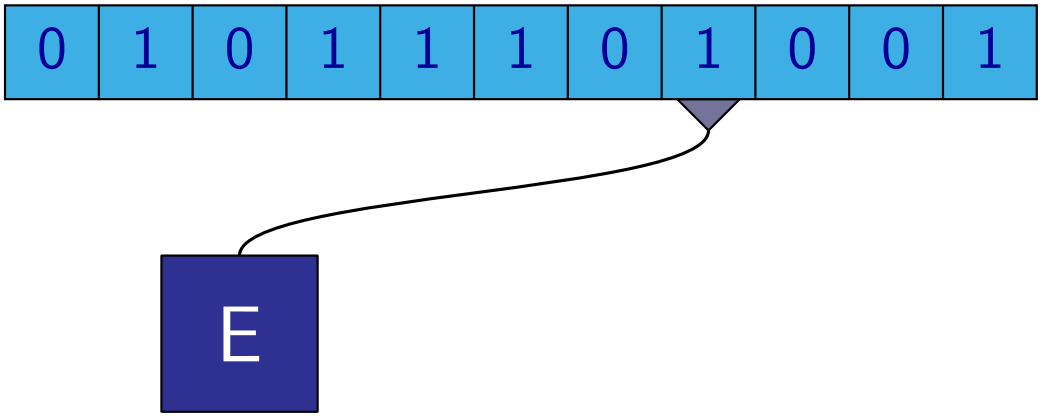




Recognition of a Language

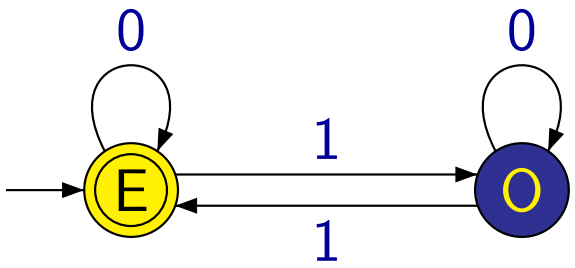
The behaviour of the device can be described by the following graph:

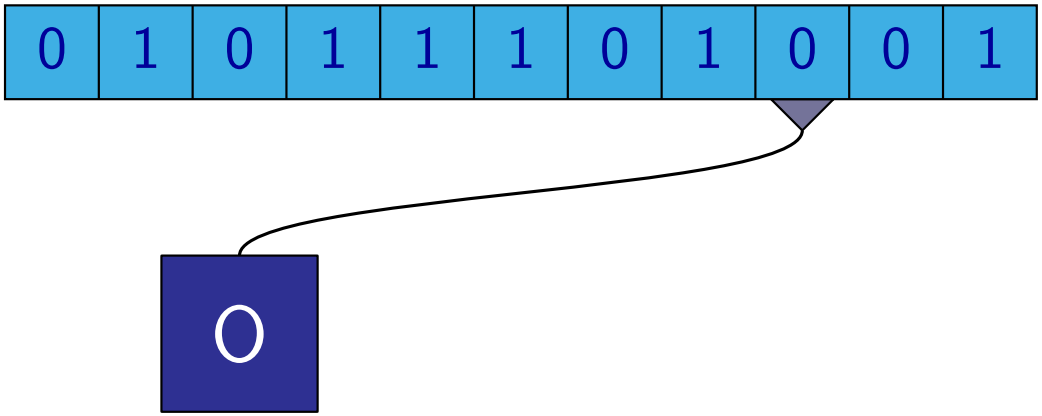




Recognition of a Language

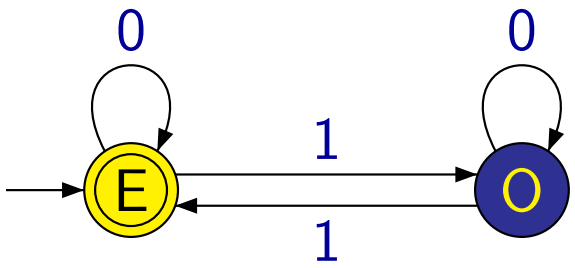
The behaviour of the device can be described by the following graph:

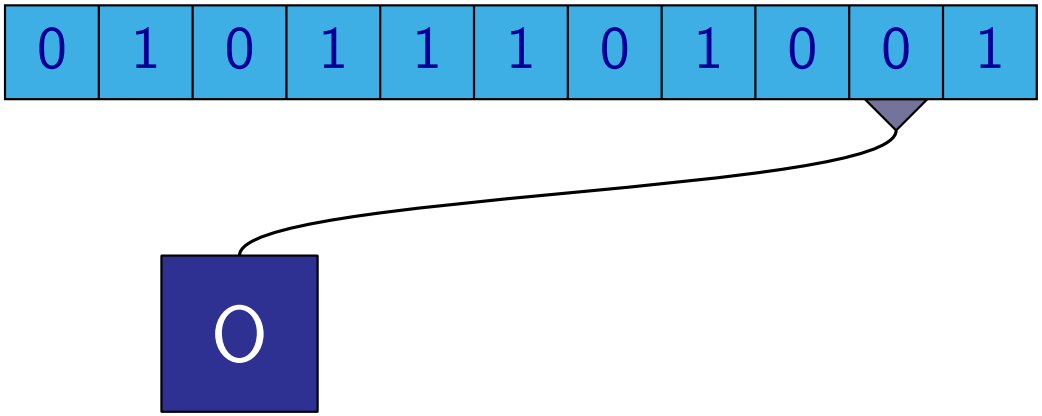




Recognition of a Language

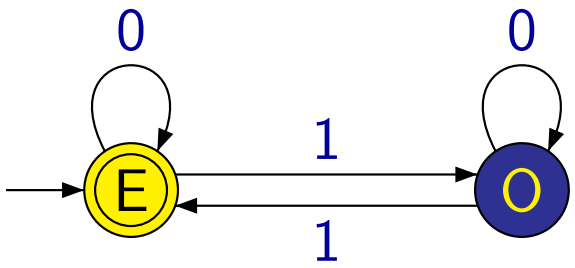
The behaviour of the device can be described by the following graph:

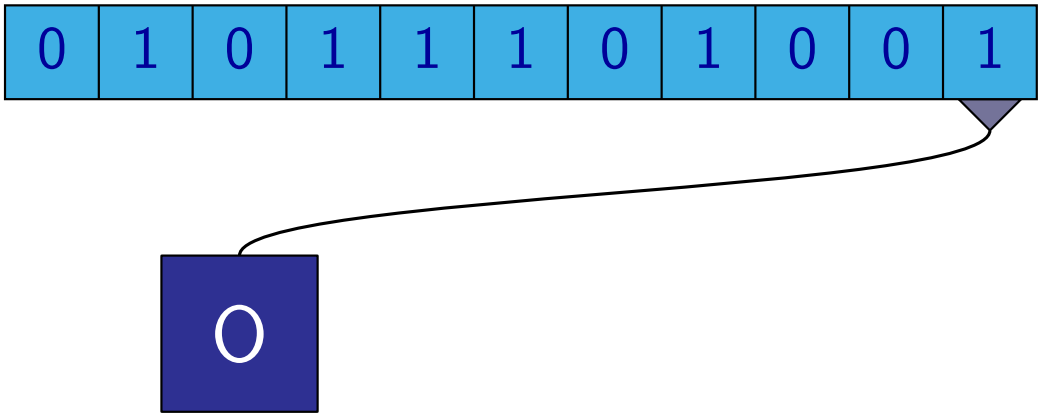




Recognition of a Language

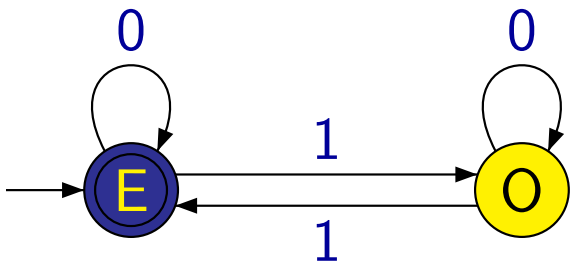
The behaviour of the device can be described by the following graph:

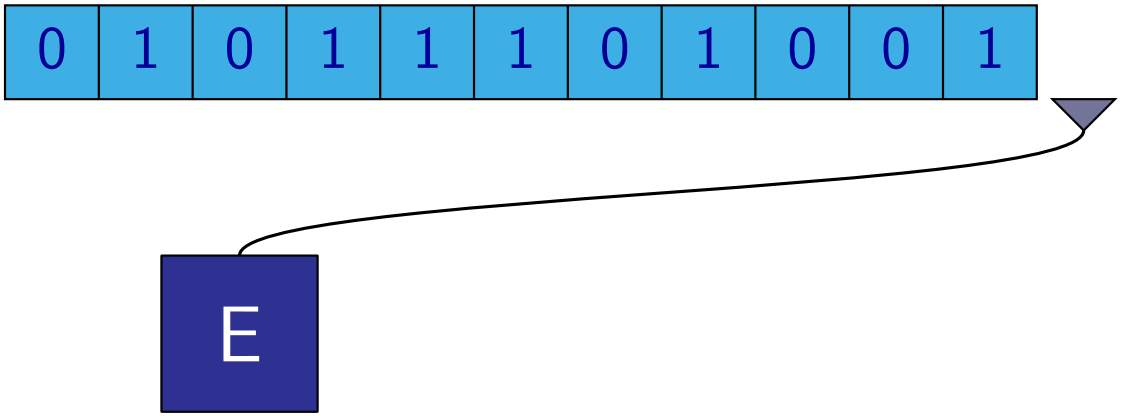




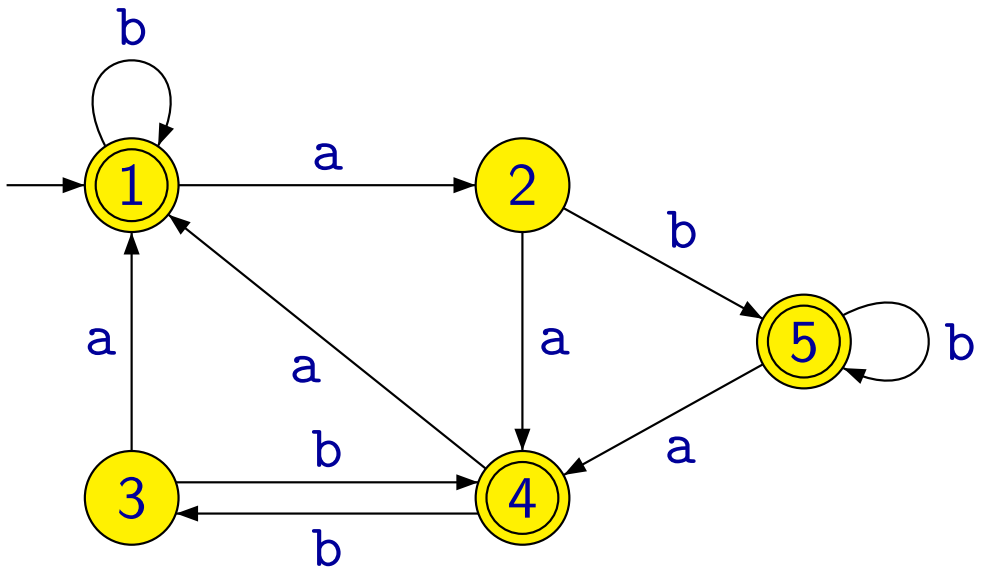
Recognition of a Language

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





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

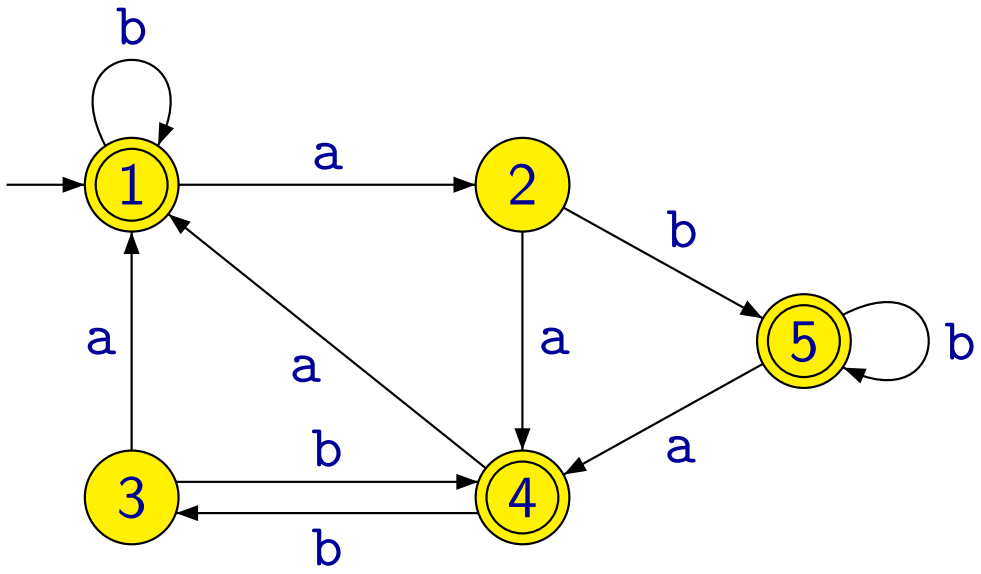
Formally, a **deterministic finite automaton** (**DFA**) is defined as a tuple

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

where:

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

Deterministic Finite Automaton



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

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

Deterministic Finite Automaton

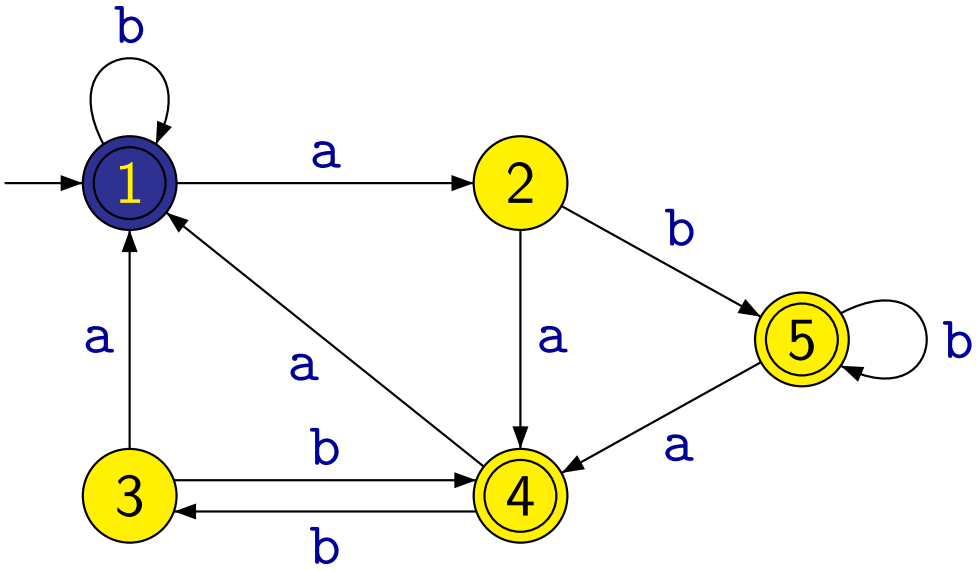
Instead of

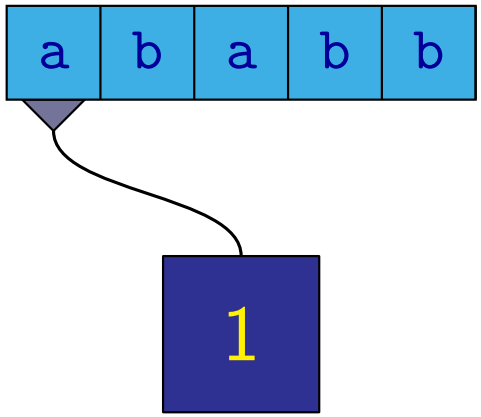
$$\begin{array}{ll} \delta(1,\mathtt{a}) = 2 & \delta(1,\mathtt{b}) = 1 \\ \delta(2,\mathtt{a}) = 4 & \delta(2,\mathtt{b}) = 5 \\ \delta(3,\mathtt{a}) = 1 & \delta(3,\mathtt{b}) = 4 \\ \delta(4,\mathtt{a}) = 1 & \delta(4,\mathtt{b}) = 3 \\ \delta(5,\mathtt{a}) = 4 & \delta(5,\mathtt{b}) = 5 \end{array}$$

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

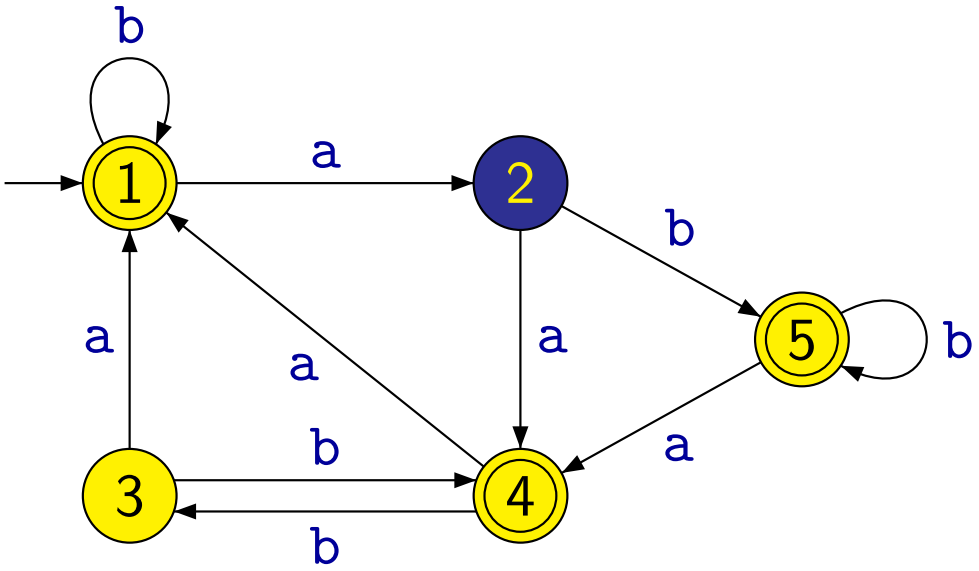
Deterministic Finite Automaton

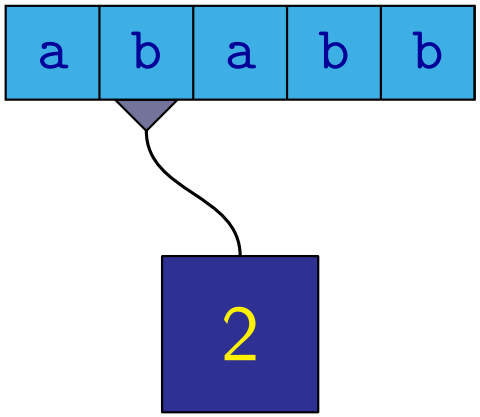




1

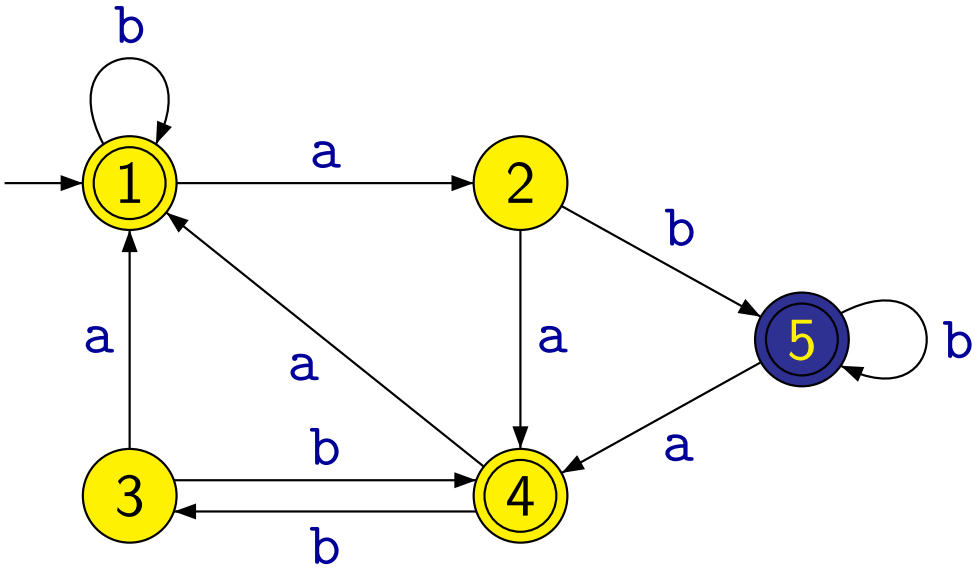
Deterministic Finite Automaton

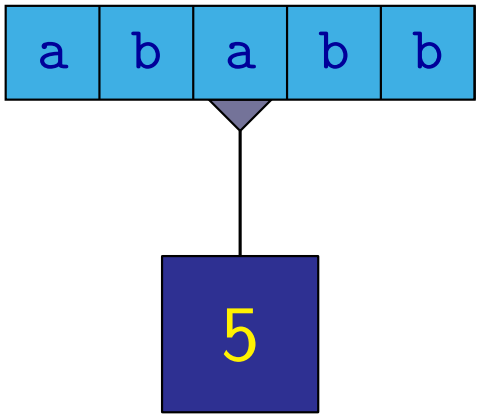




$1 \xrightarrow{a} 2$

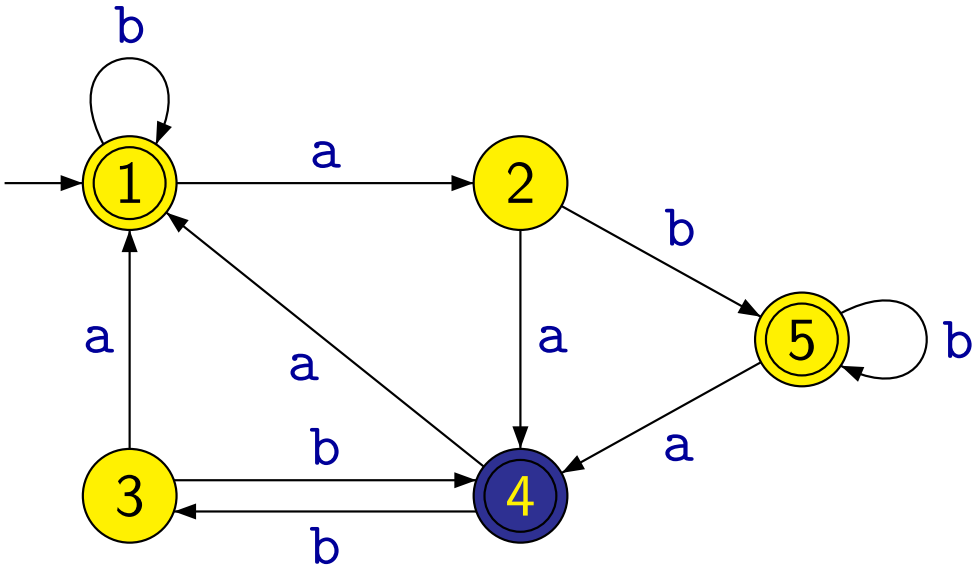
Deterministic Finite Automaton

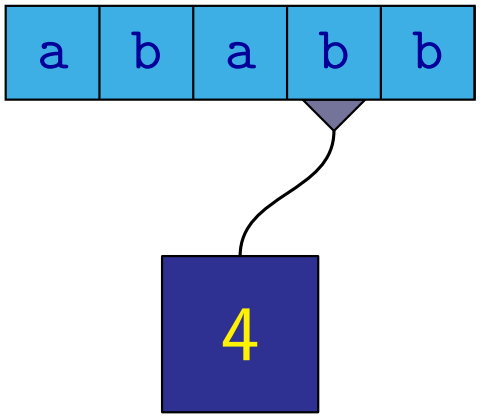




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5$$

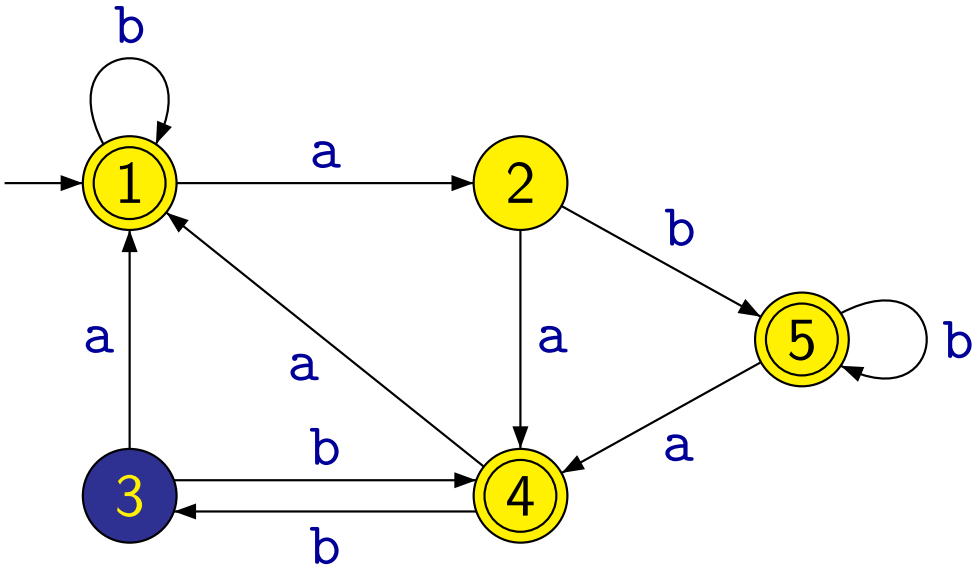
Deterministic Finite Automaton

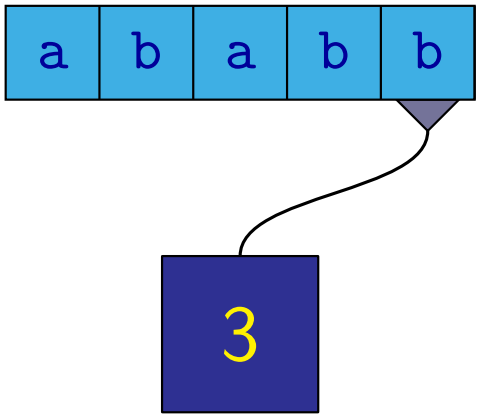




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4$$

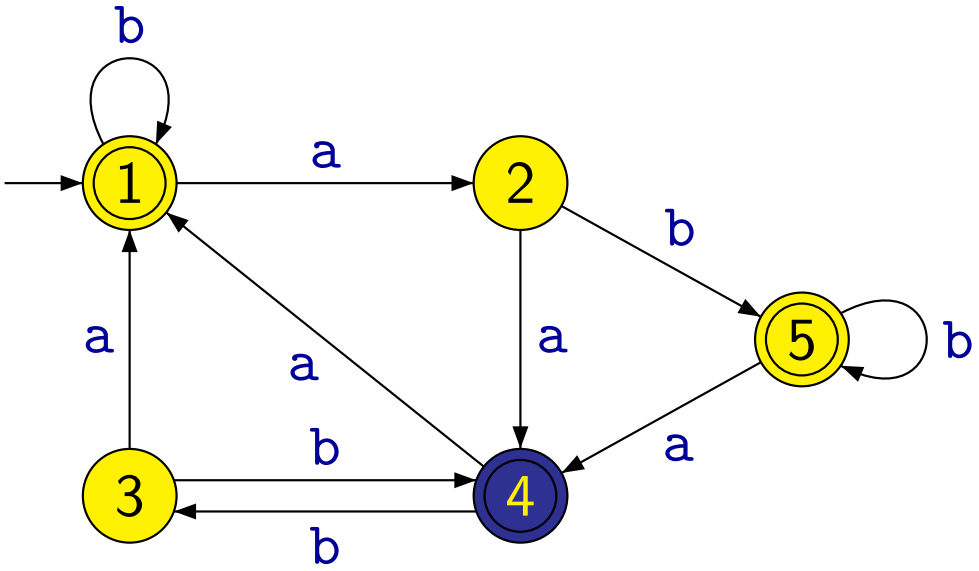
Deterministic Finite Automaton

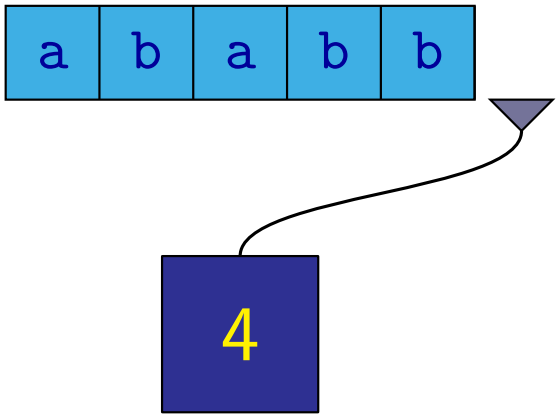




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3$$

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 $A = (Q, \Sigma, \delta, q_0, F)$.

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 $a \in \Sigma$ and $w \in \Sigma^*$:
  $q \xrightarrow{aw} q'$ iff there is $q'' \in Q$ such that $\delta(q, a) = q''$ and $q'' \xrightarrow{w} q'$.

Deterministic Finite Automaton

A word $w \in \Sigma^*$ is **accepted** by a deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ 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 $A = (Q, \Sigma, \delta, q_0, F)$, denoted $L(A)$, is the set of all words accepted by the automaton, i.e.,

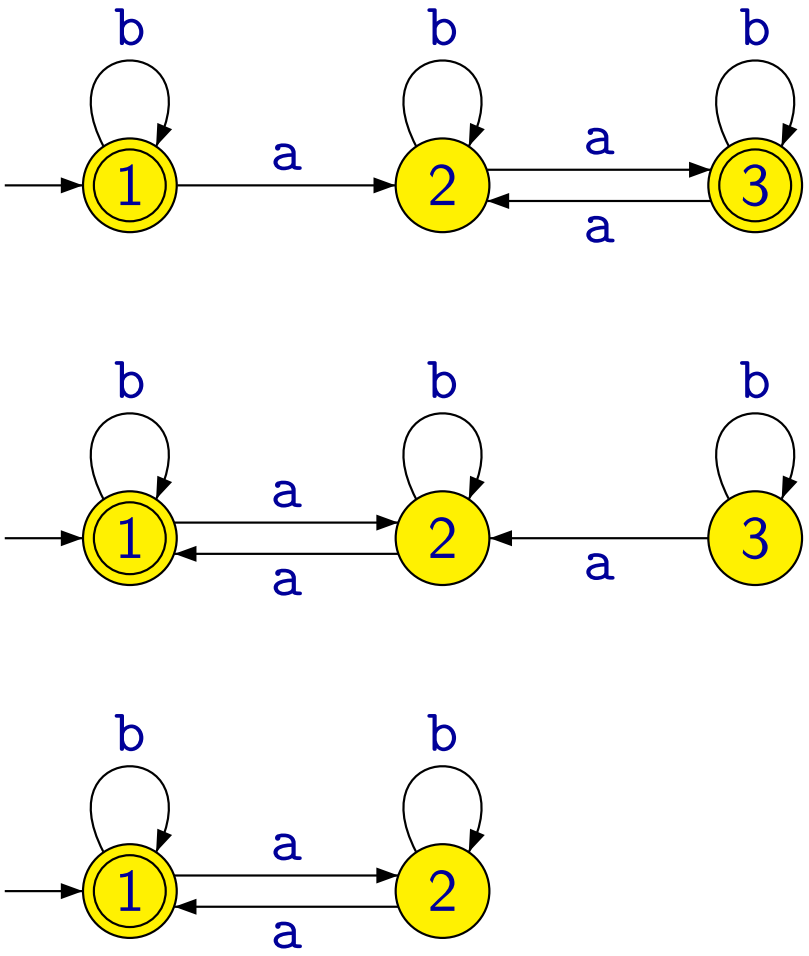
$$L(A) = \{w \in \Sigma^* \mid \exists q \in F : q_0 \xrightarrow{w} q\}$$

Regular languages

Definition

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

Equivalence of Automata



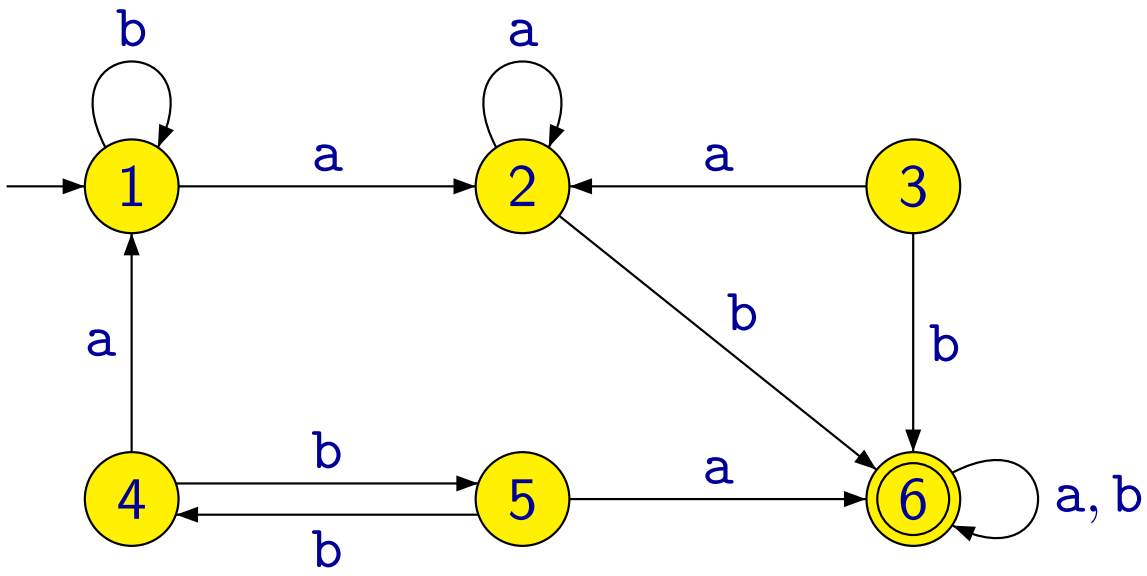
All 3 automata accept the language of all words with an even number of a's.

Equivalence of Automata

Definition

We say automata A_1, A_2 are **equivalent** if $L(A_1) = L(A_2)$.

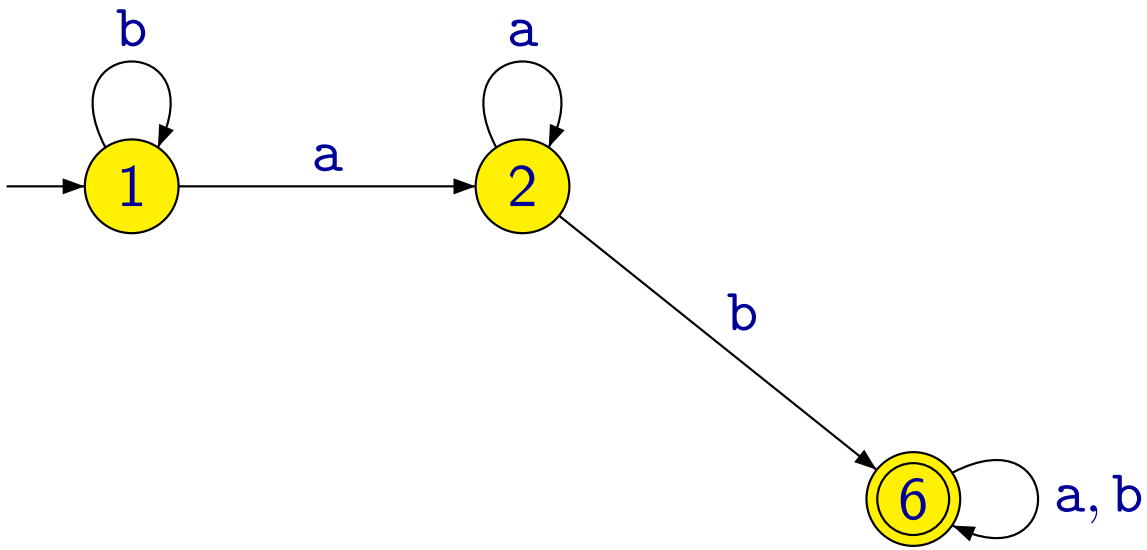
Unreachable States of an Automaton



- 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.

.

Unreachable States of an Automaton



- 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

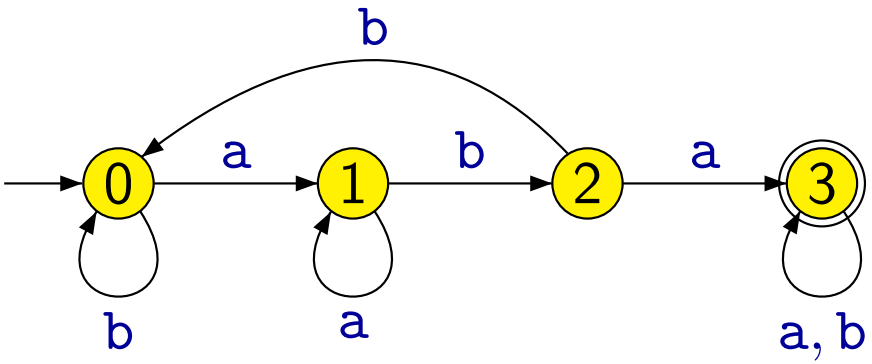
A state q of a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ is **reacheable** if there exists a word w such that $q_0 \xrightarrow{w} q$.

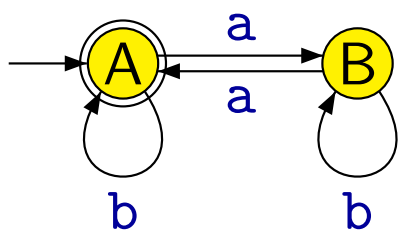
Otherwise the state is **unreachable**.

- 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.

An Automaton for Intersection of Languages

Let us have the following two automata:

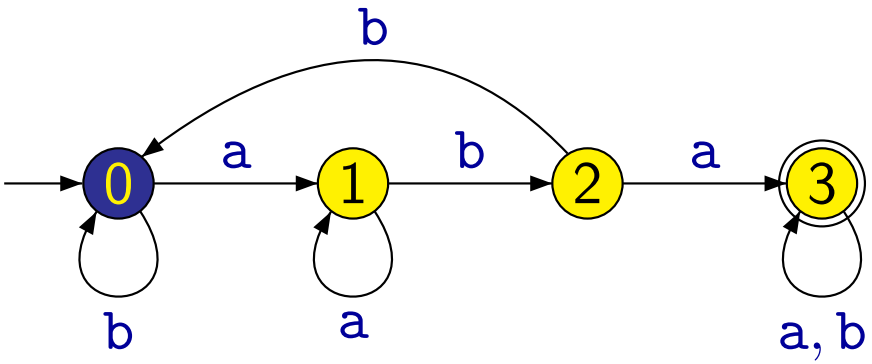


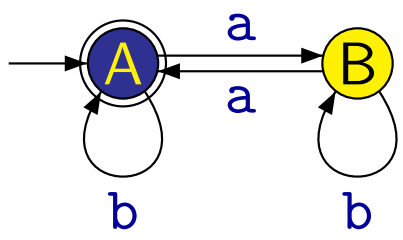


Do both of them accept the word `ababb`?

An Automaton for Intersection of Languages

Let us have the following two automata:

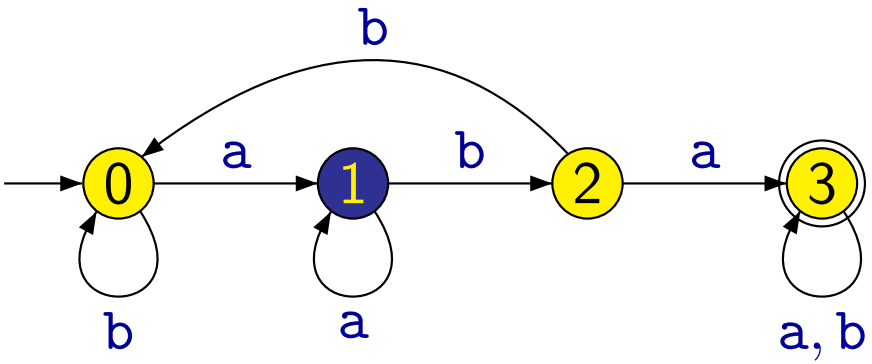


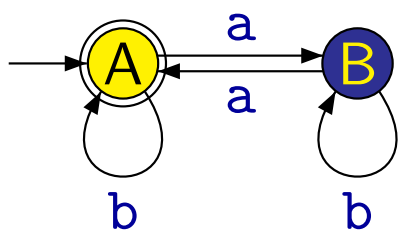


Do both of them accept the word ababb?

An Automaton for Intersection of Languages

Let us have the following two automata:

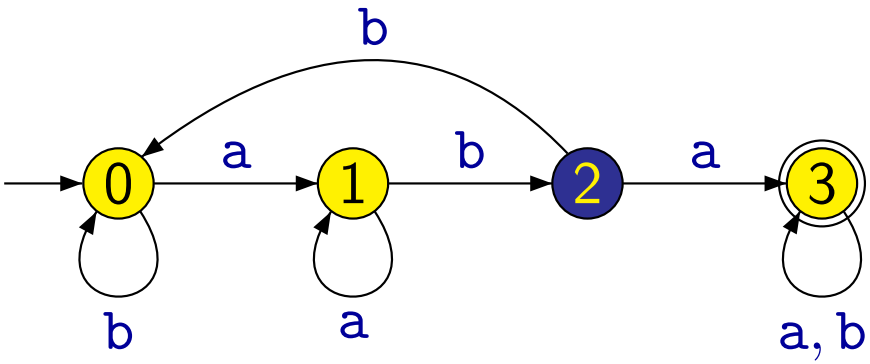


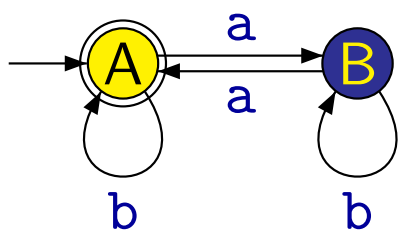


Do both of them accept the word ababb?

An Automaton for Intersection of Languages

Let us have the following two automata:

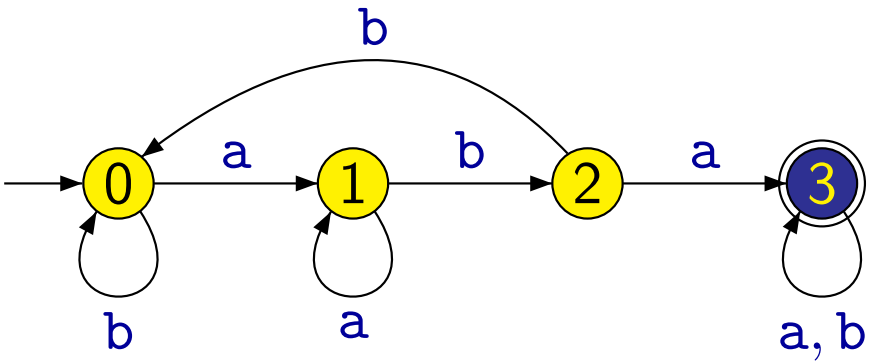


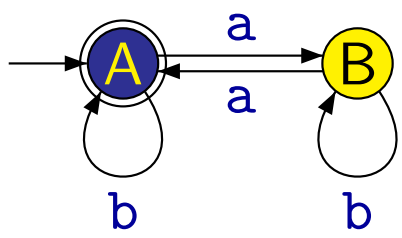


Do both of them accept the word ababb?

An Automaton for Intersection of Languages

Let us have the following two automata:

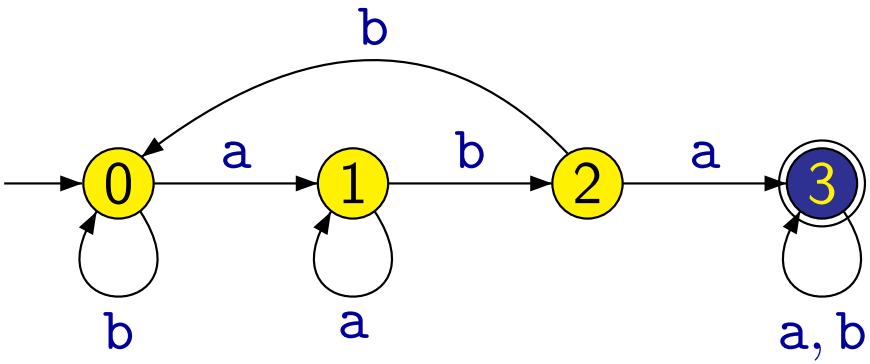


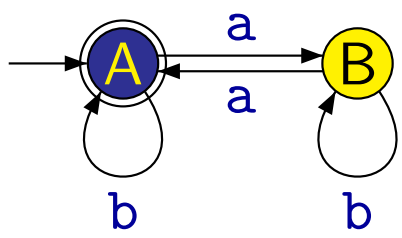


Do both of them accept the word ababb?

An Automaton for Intersection of Languages

Let us have the following two automata:

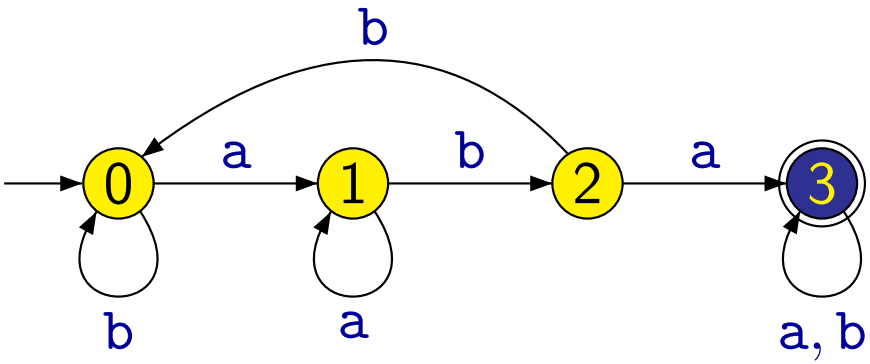


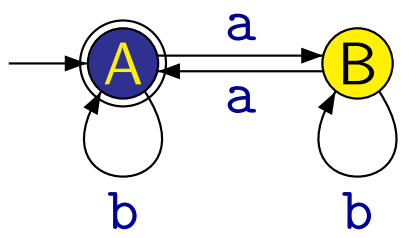


Do both of them accept the word ababb?

An Automaton for Intersection of Languages

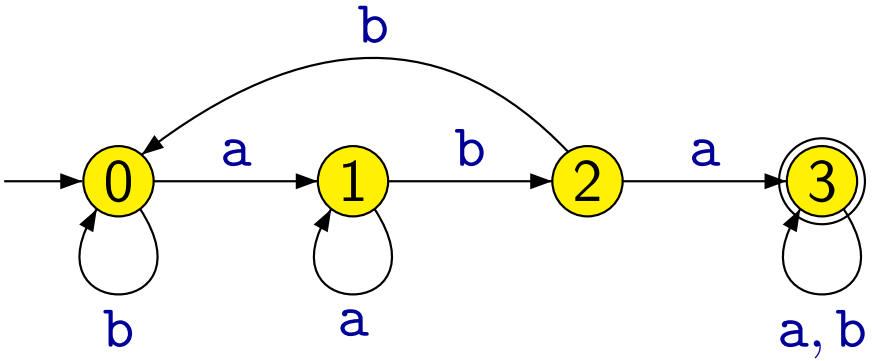
Let us have the following two automata:

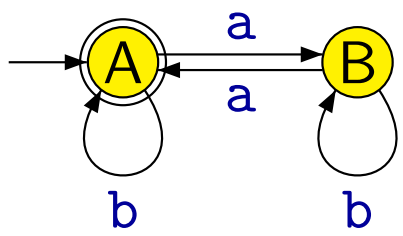




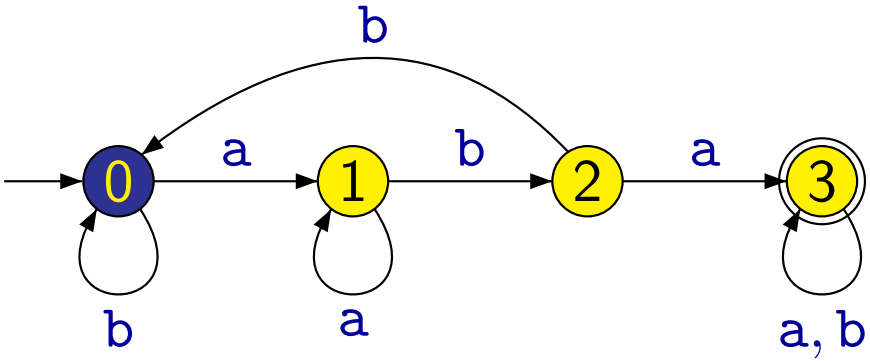
Do both of them accept the word `ababb`?

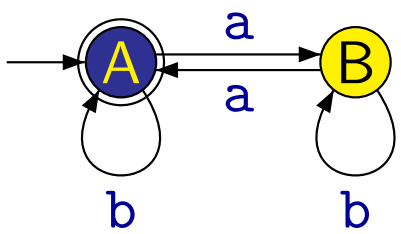
An Automaton for Intersection of Languages

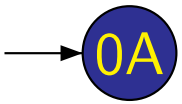




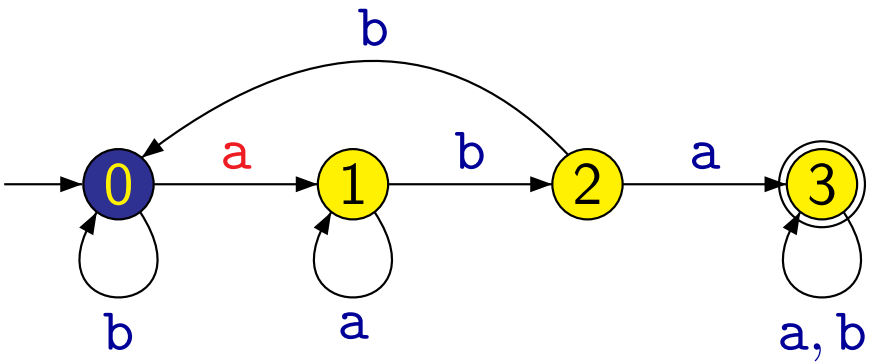
An Automaton for Intersection of Languages

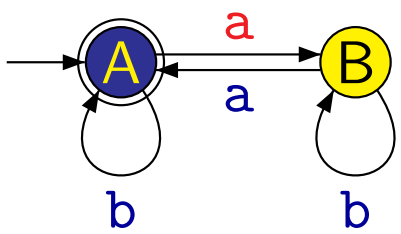


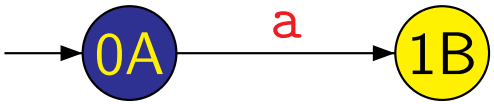




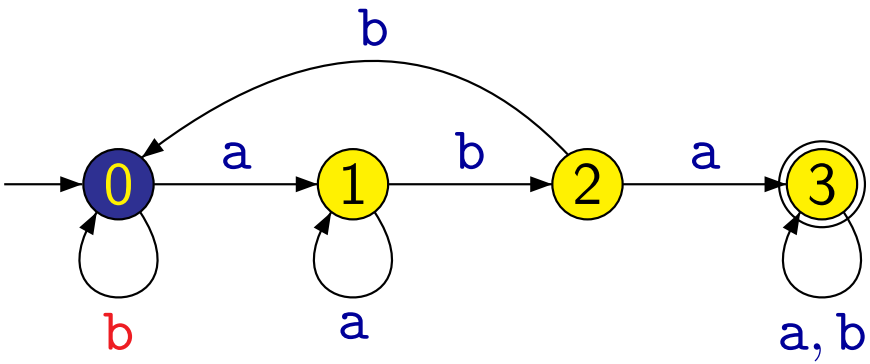
An Automaton for Intersection of Languages

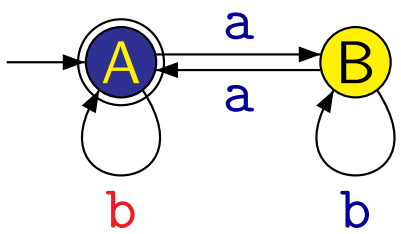


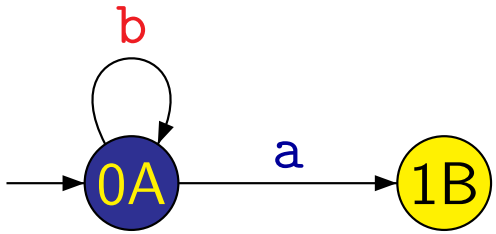




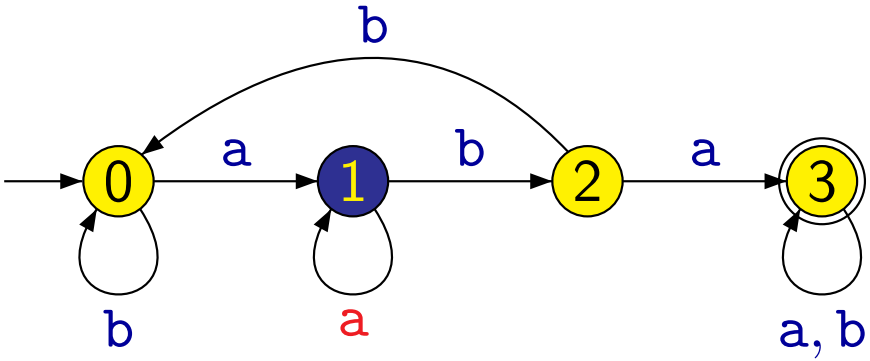
An Automaton for Intersection of Languages

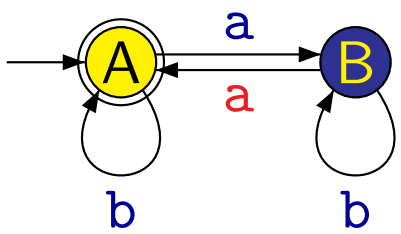


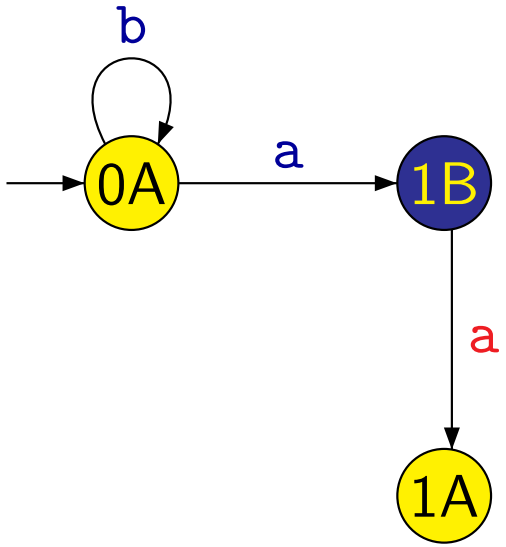




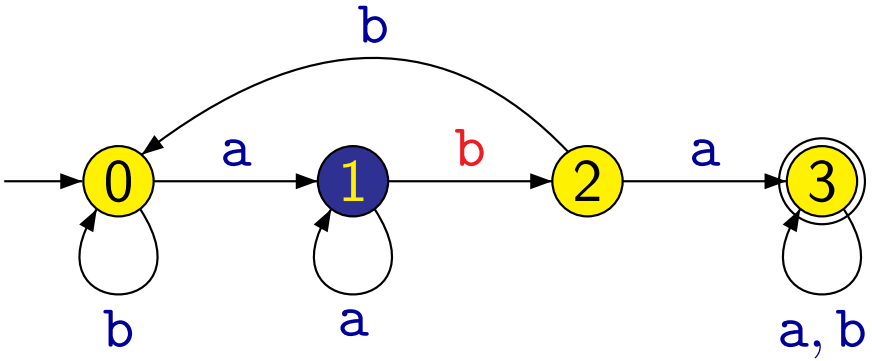
An Automaton for Intersection of Languages

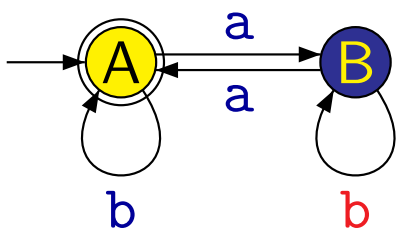


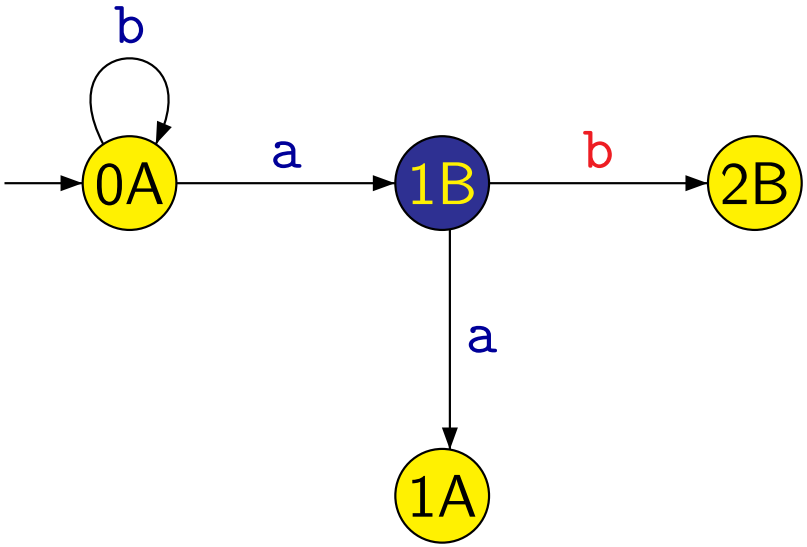




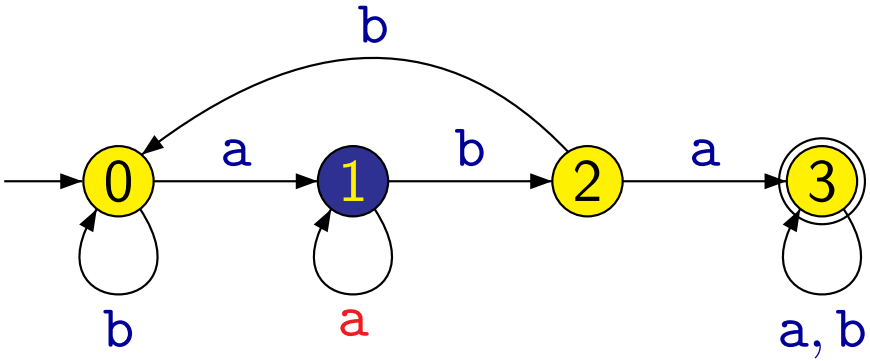
An Automaton for Intersection of Languages

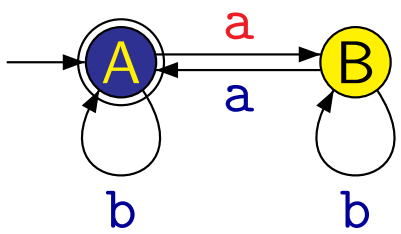


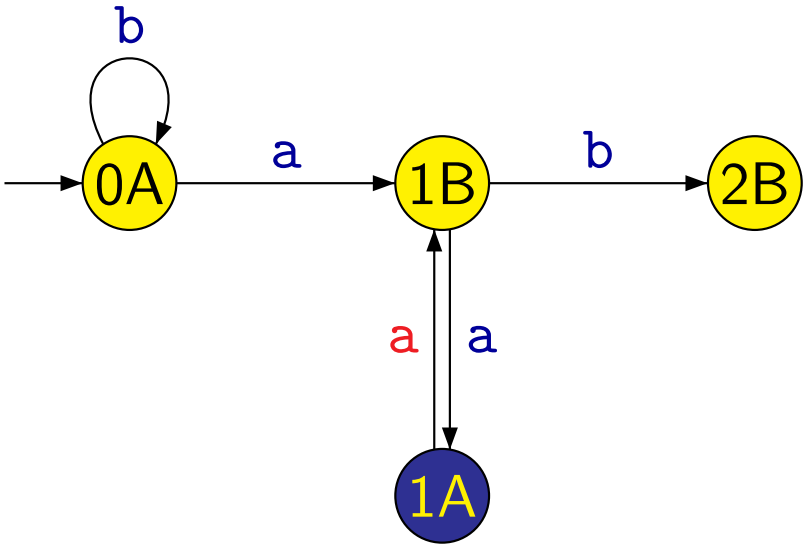




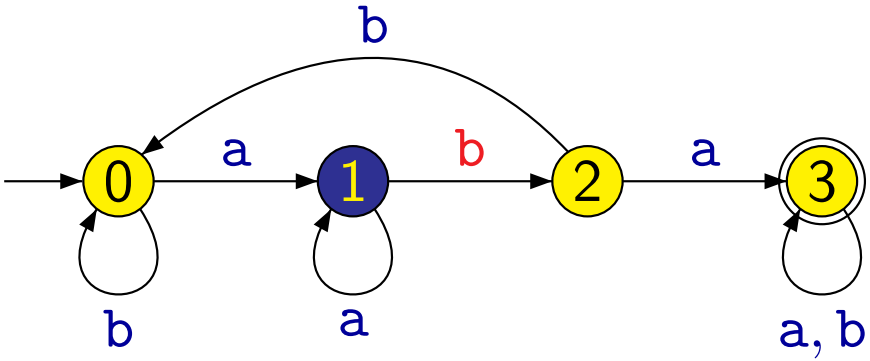
An Automaton for Intersection of Languages

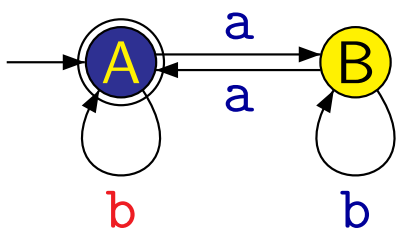


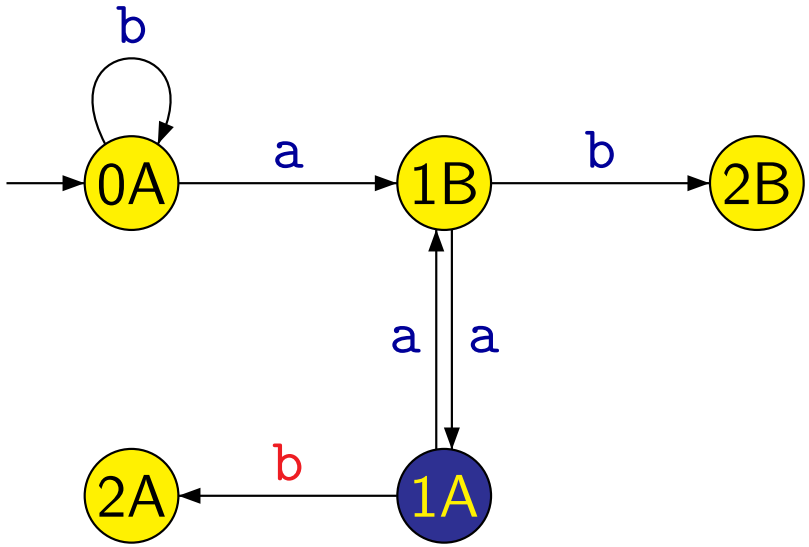




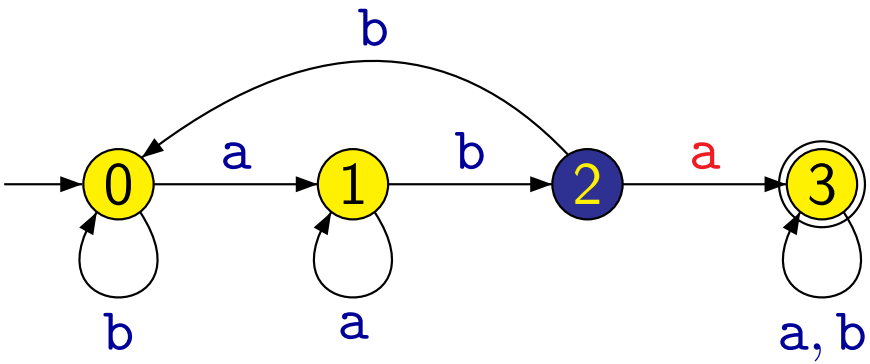
An Automaton for Intersection of Languages

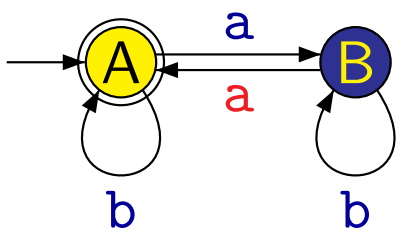


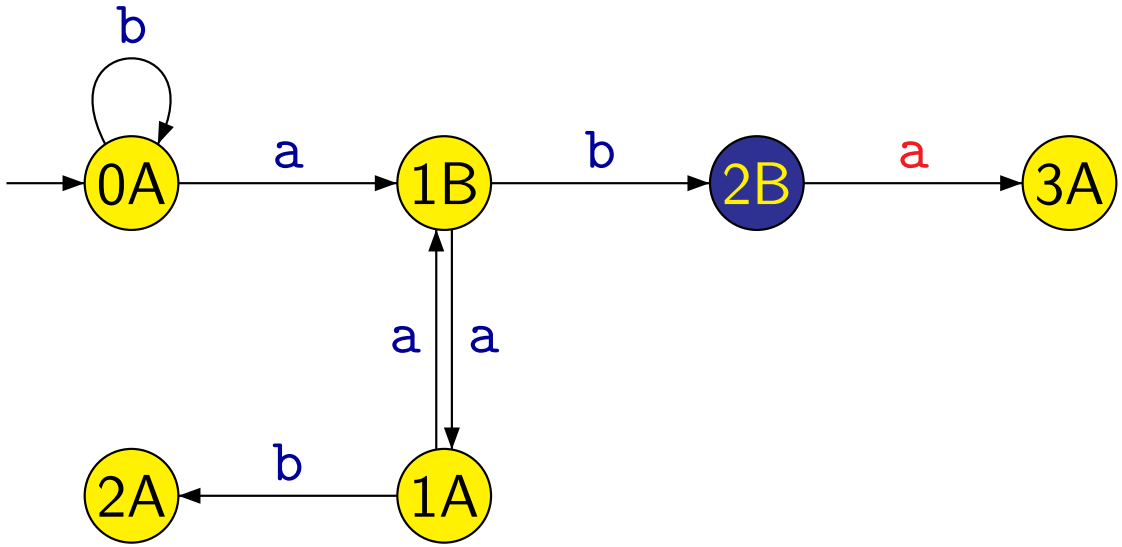




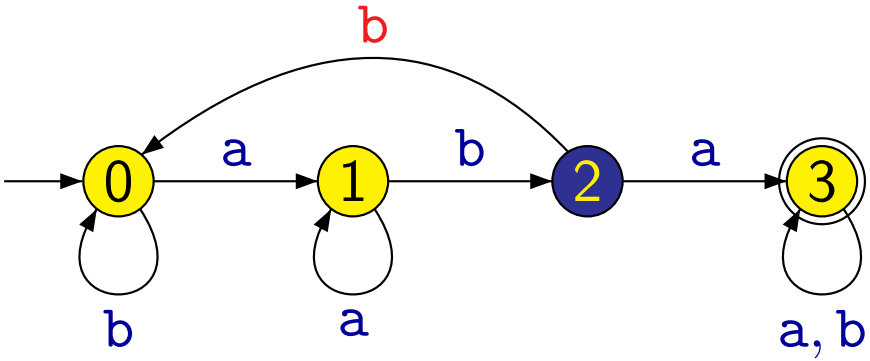
An Automaton for Intersection of Languages

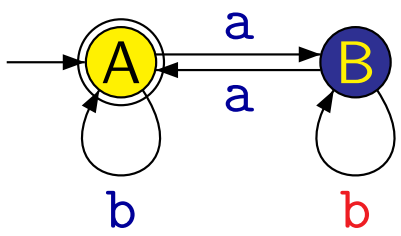


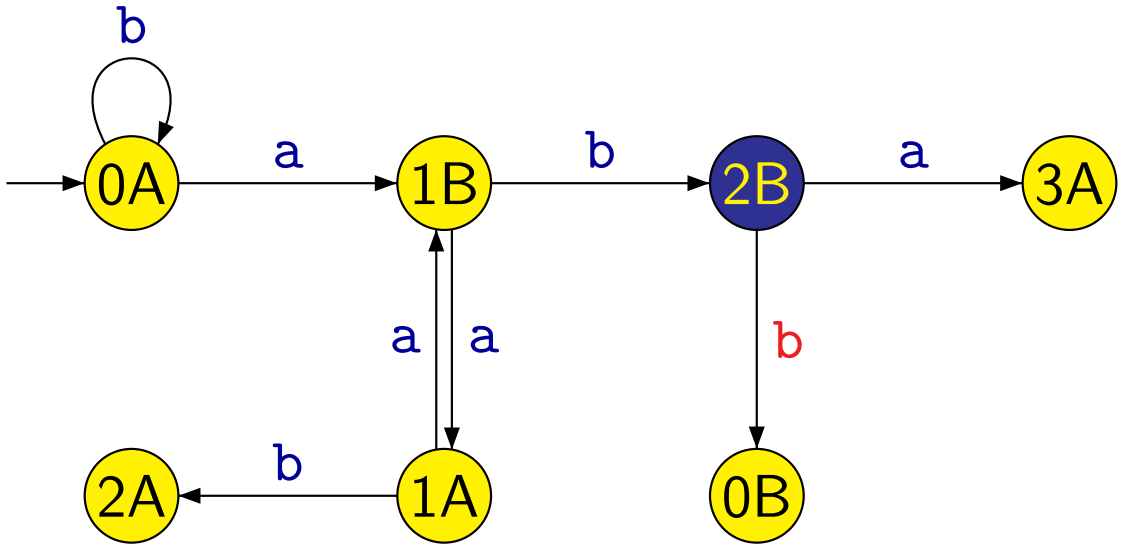




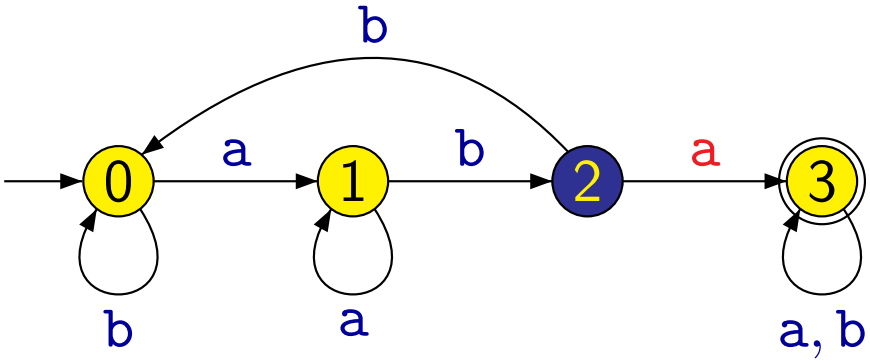
An Automaton for Intersection of Languages

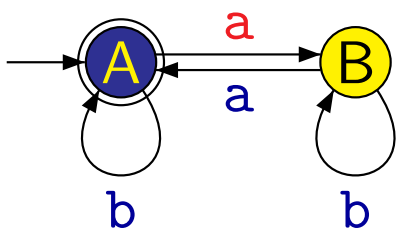


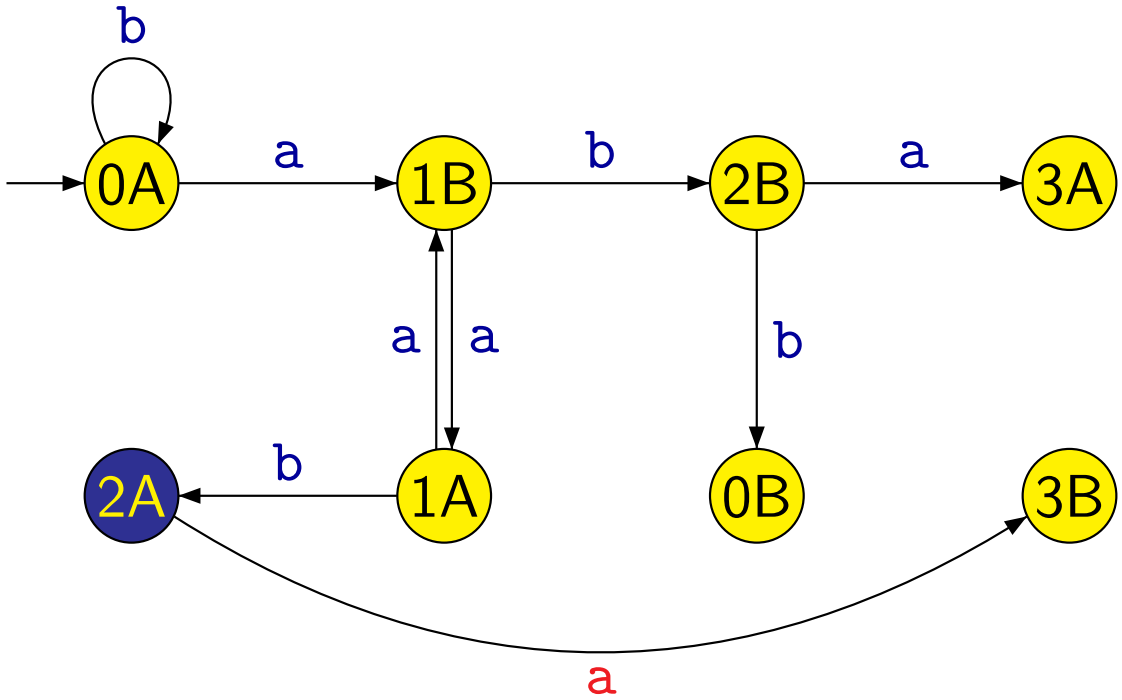




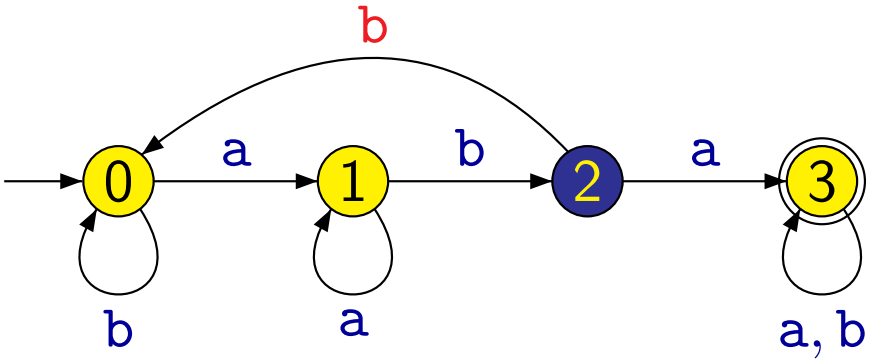
An Automaton for Intersection of Languages

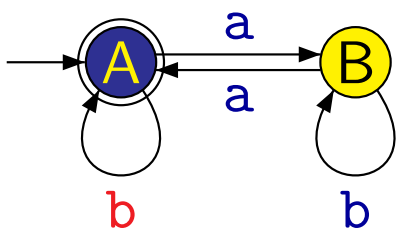


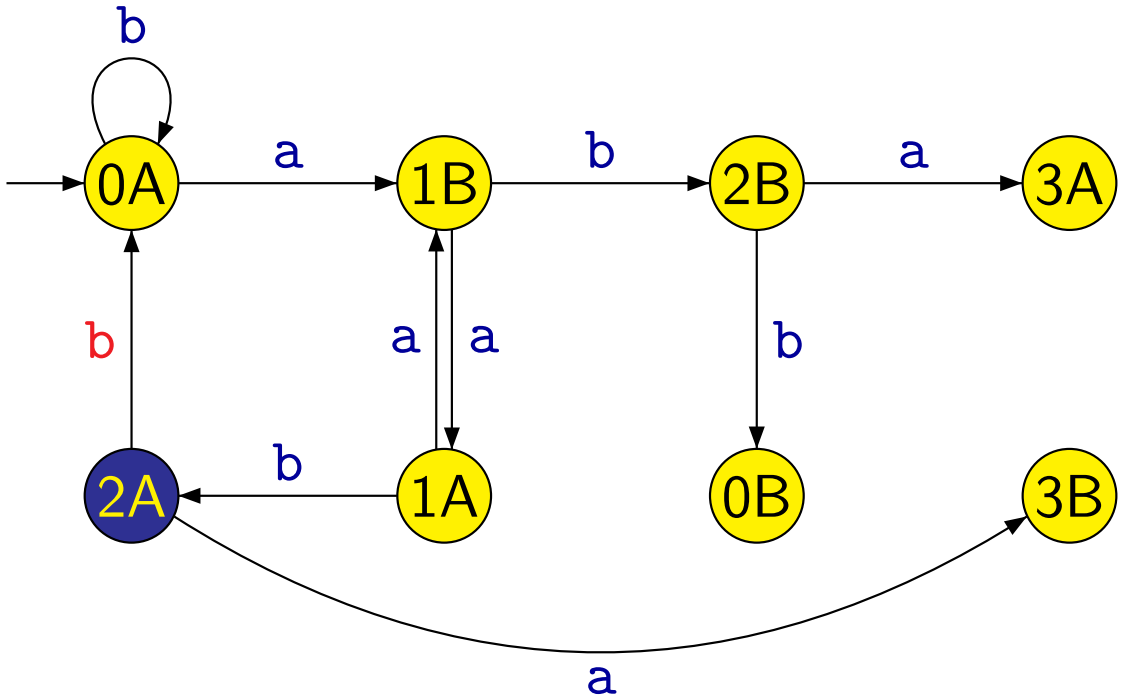




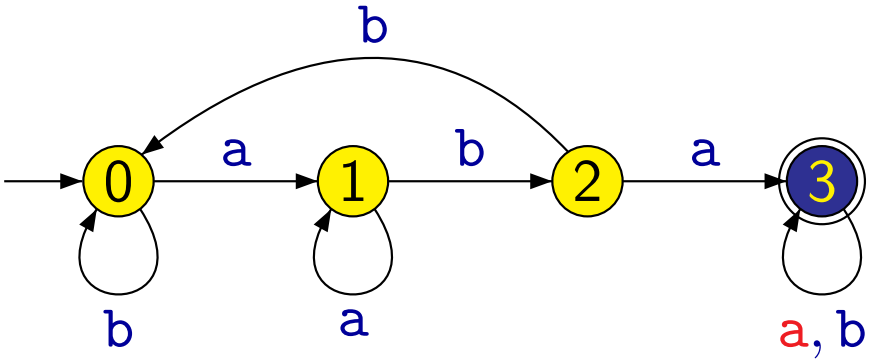
An Automaton for Intersection of Languages

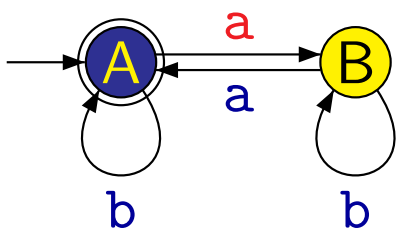


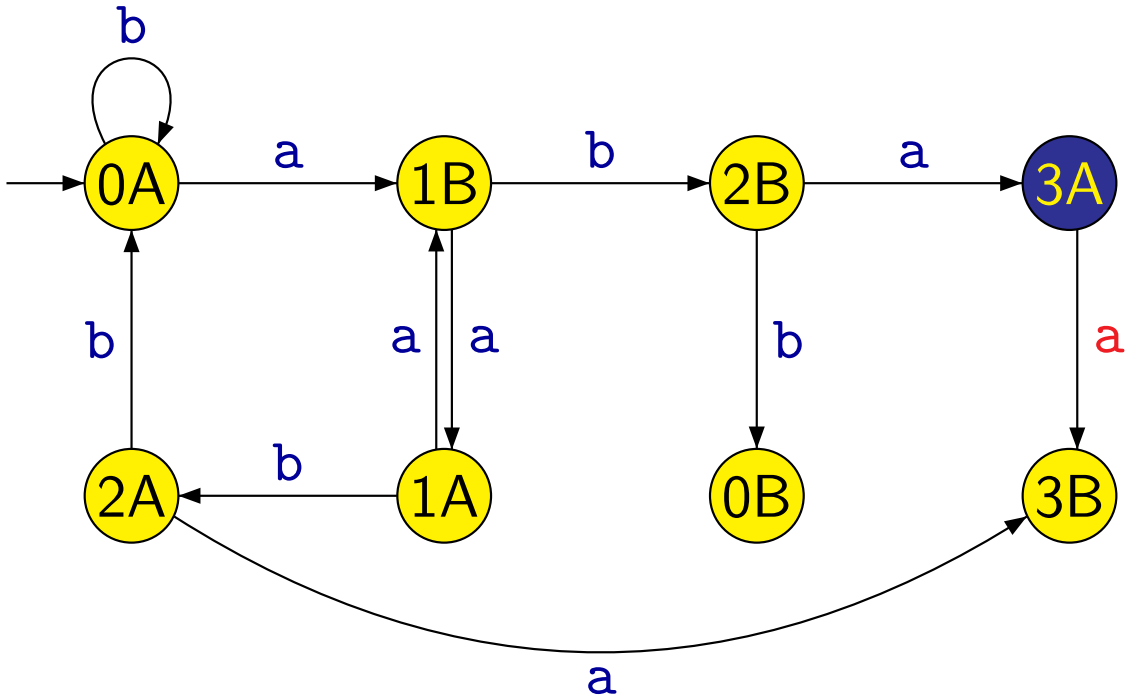




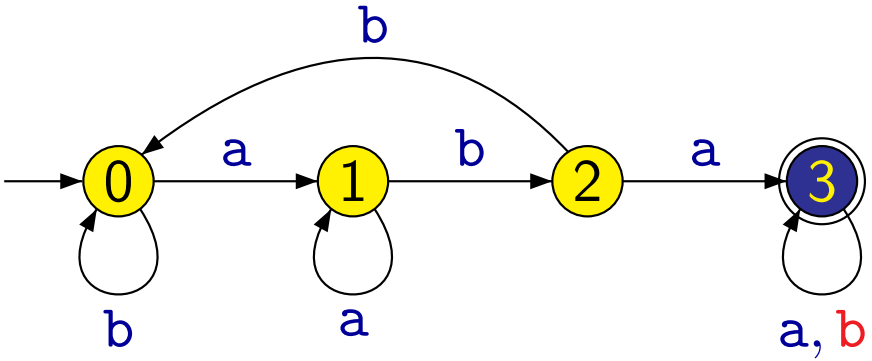
An Automaton for Intersection of Languages

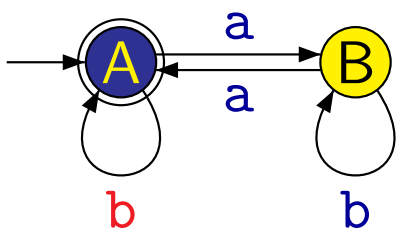


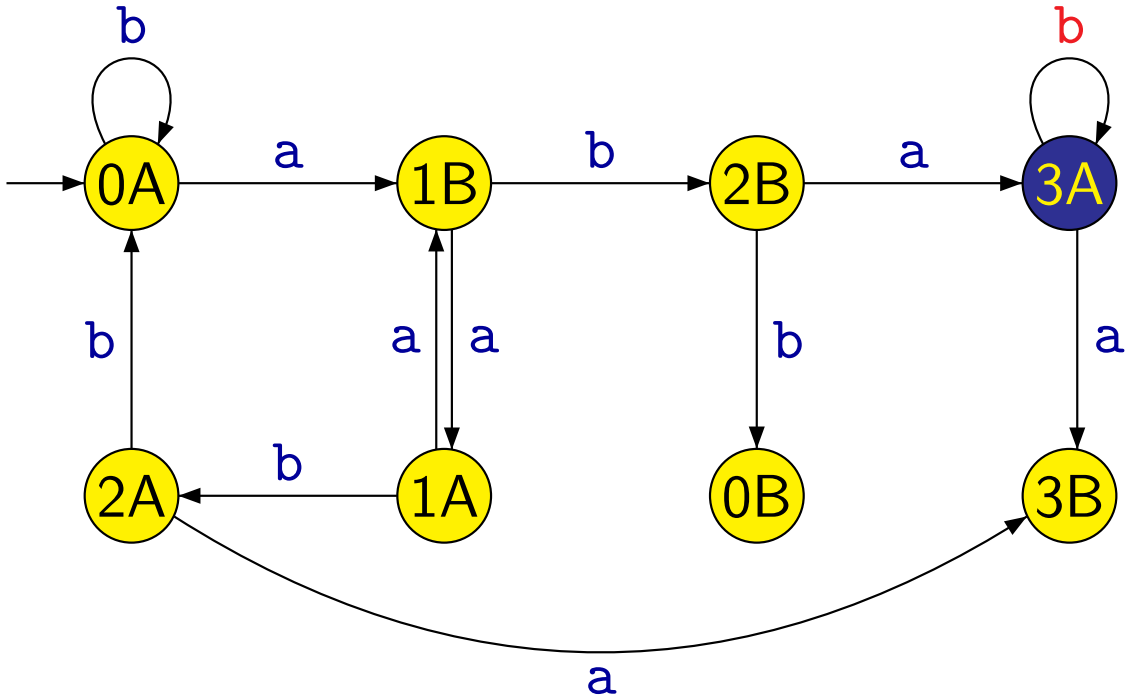




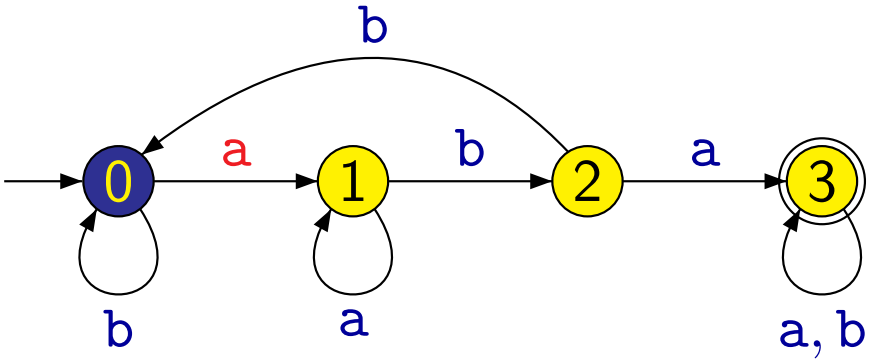
An Automaton for Intersection of Languages

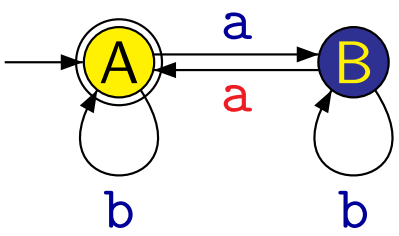


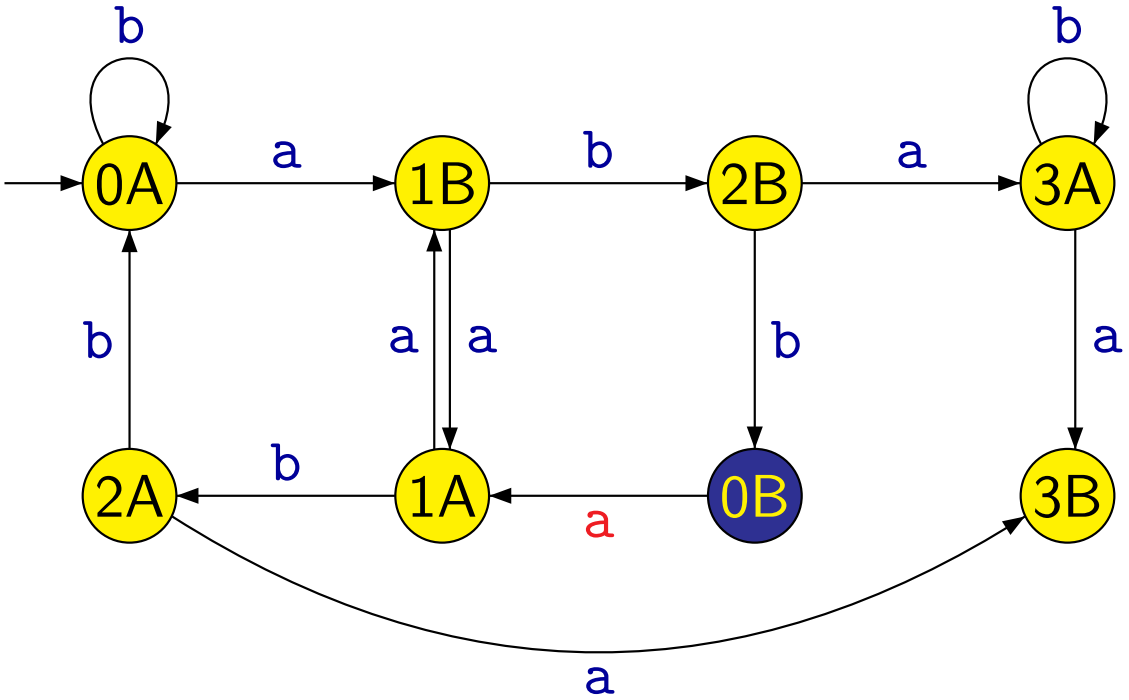




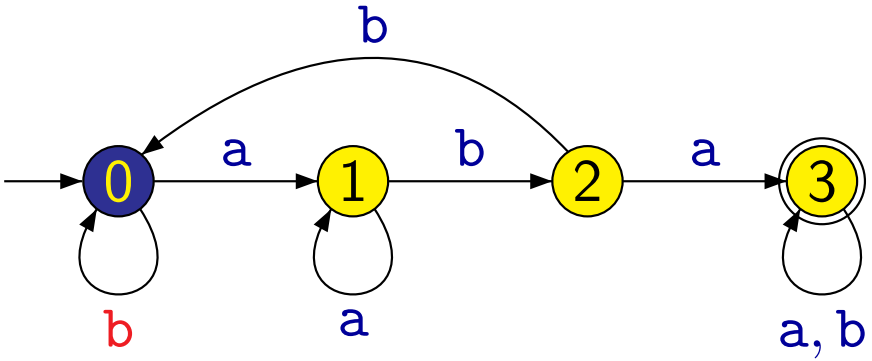
An Automaton for Intersection of Languages

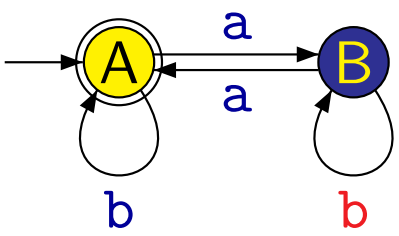


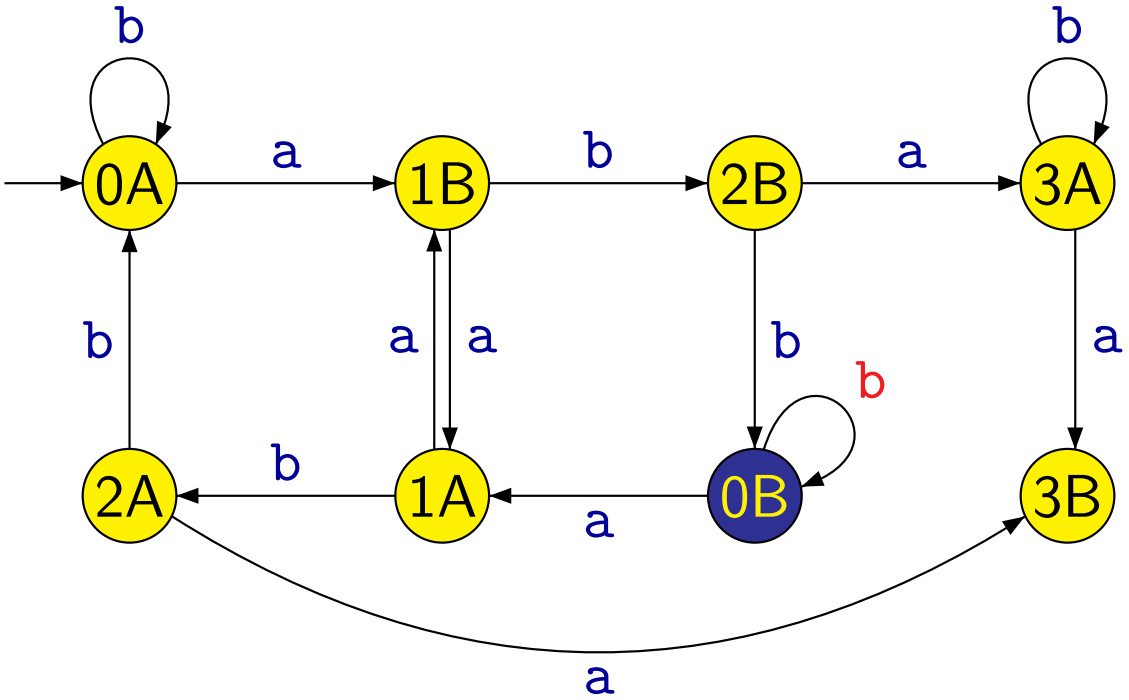




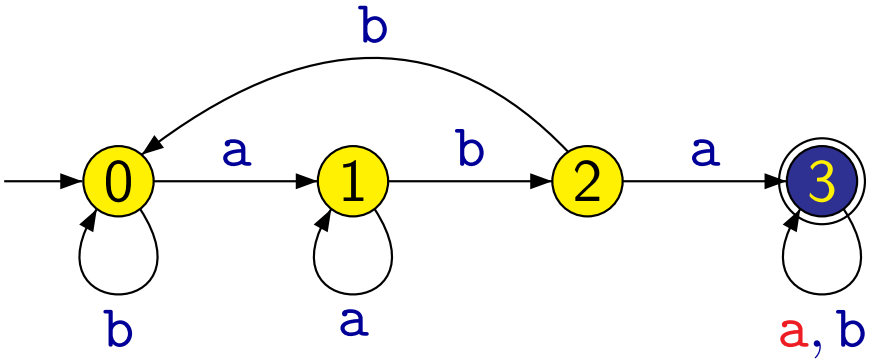
An Automaton for Intersection of Languages

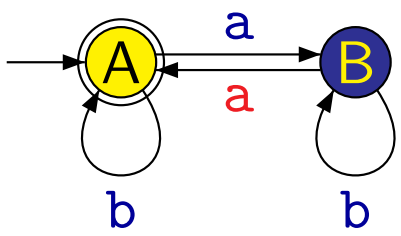


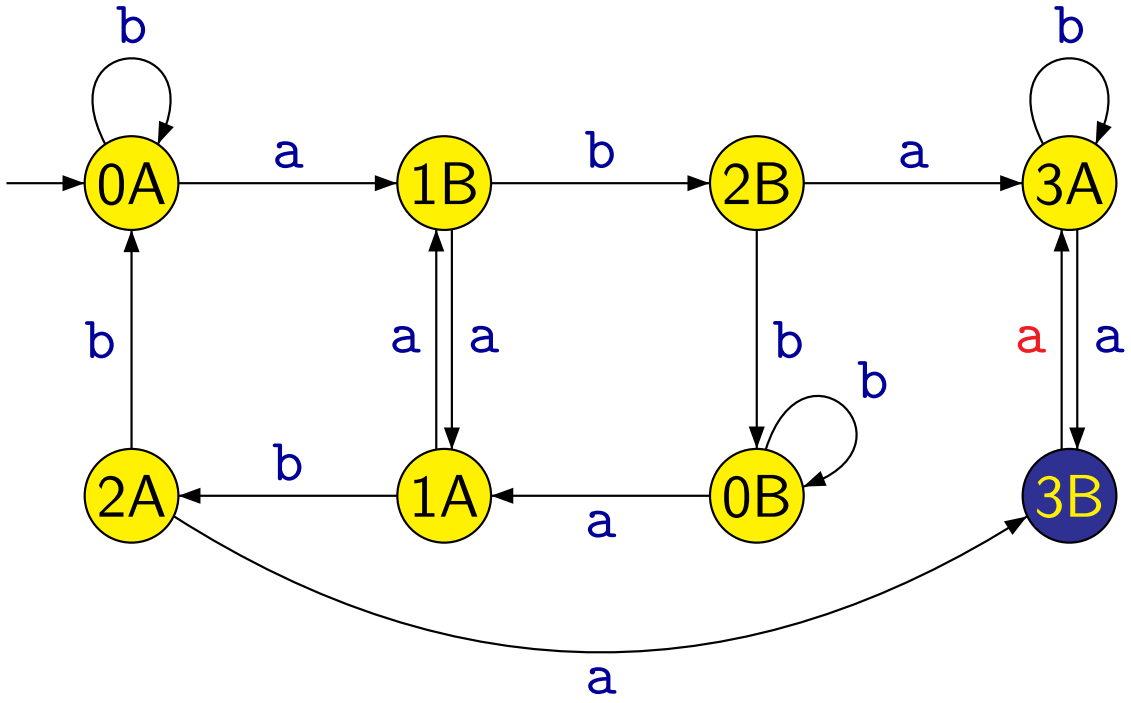




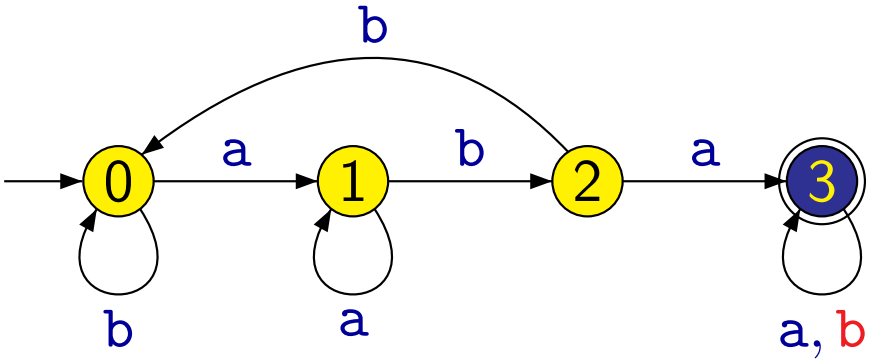
An Automaton for Intersection of Languages

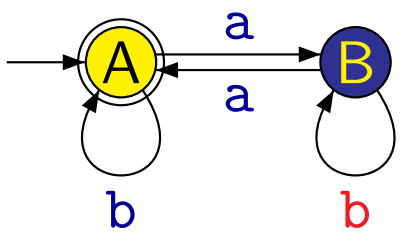


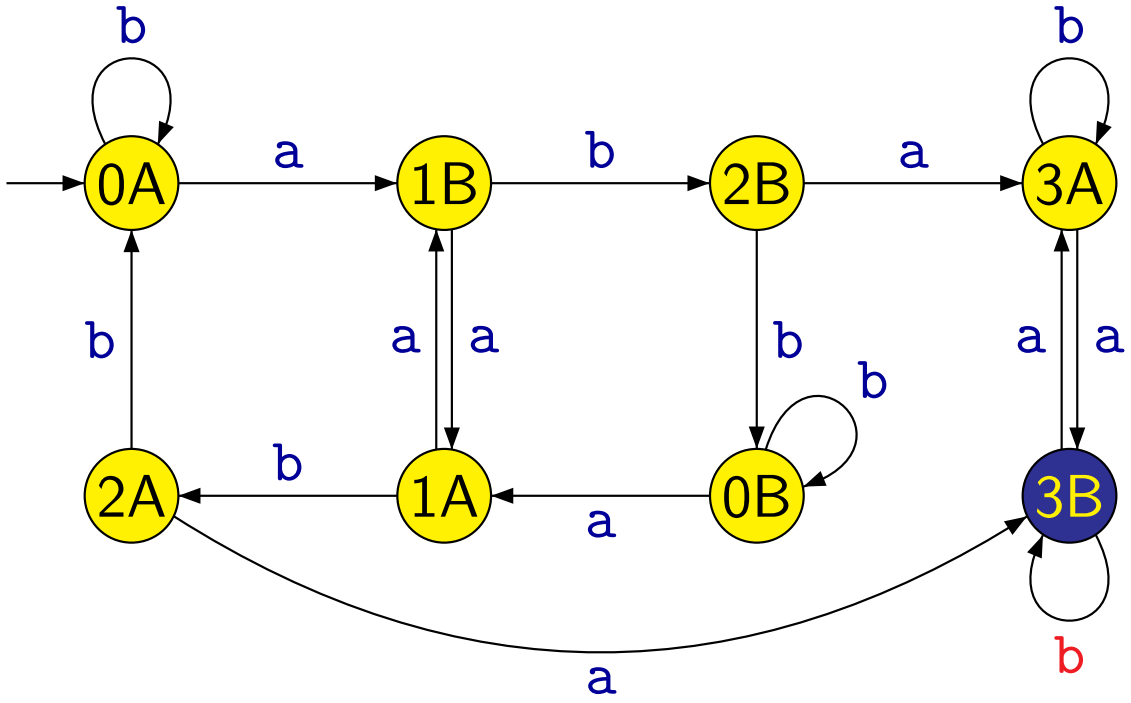




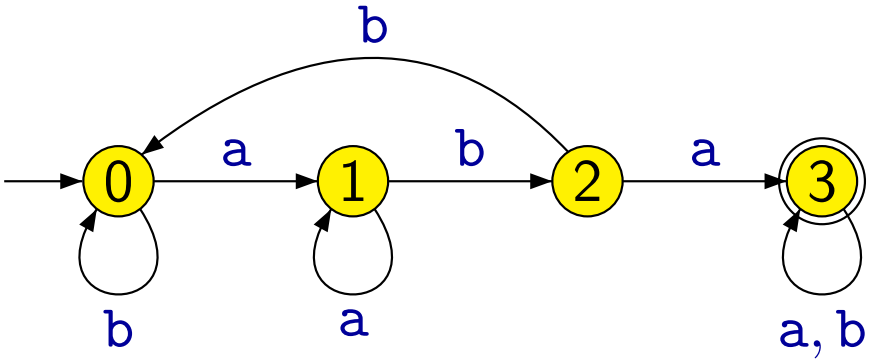
An Automaton for Intersection of Languages

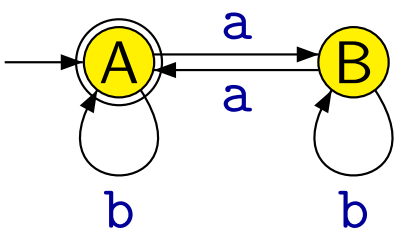


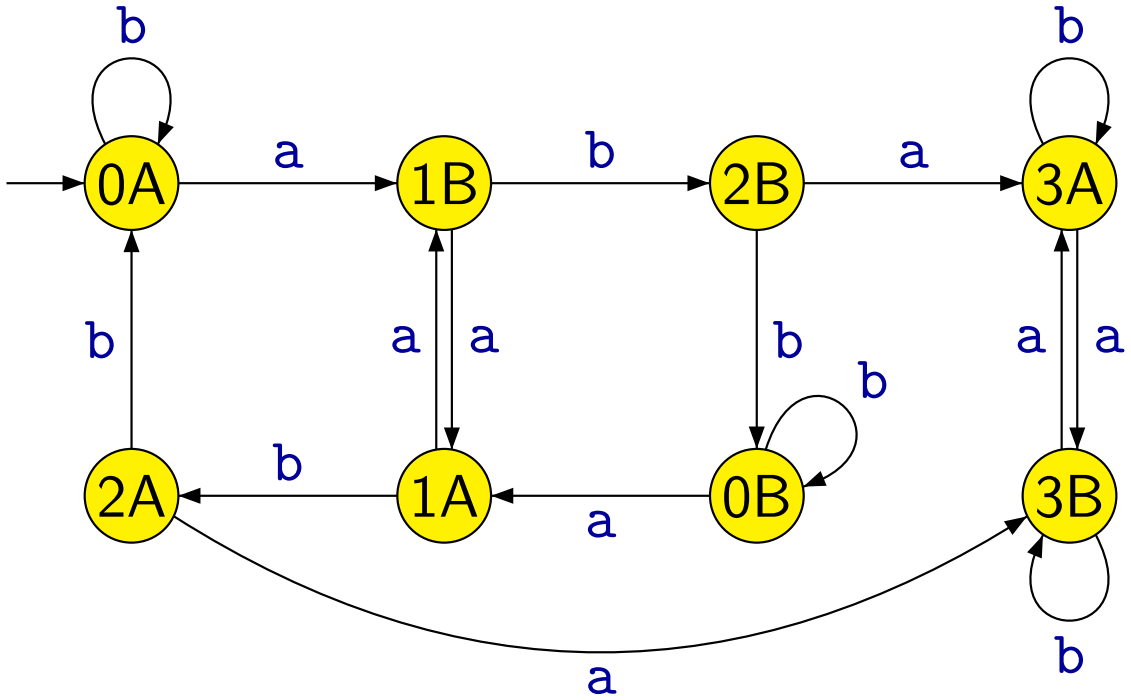




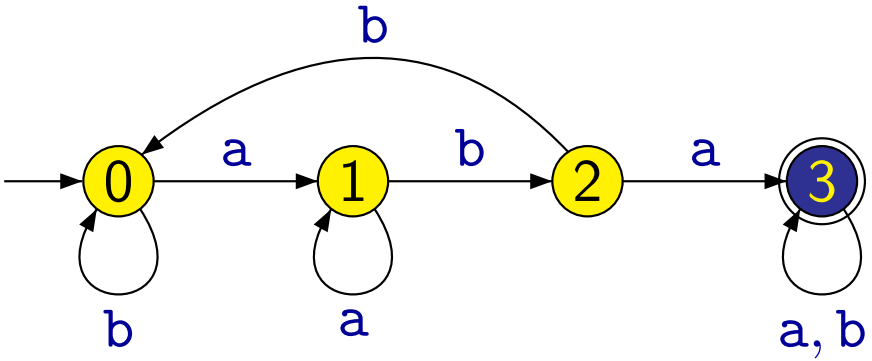
An Automaton for Intersection of Languages

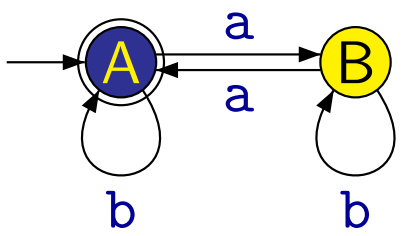


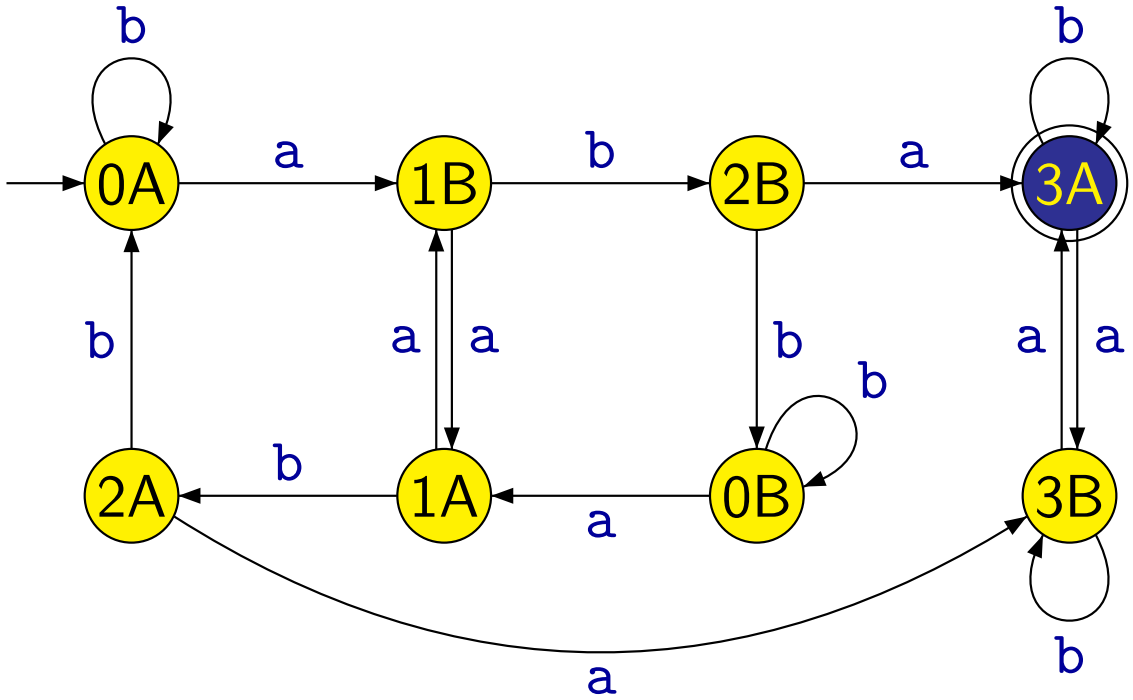




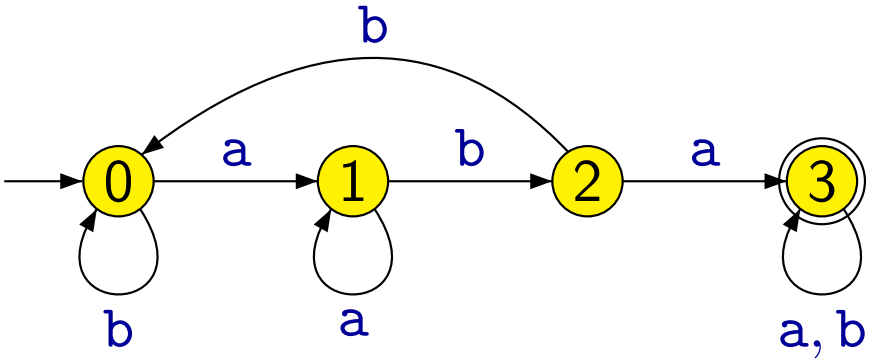
An Automaton for Intersection of Languages

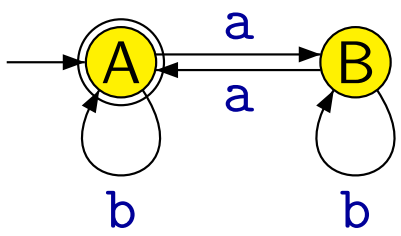


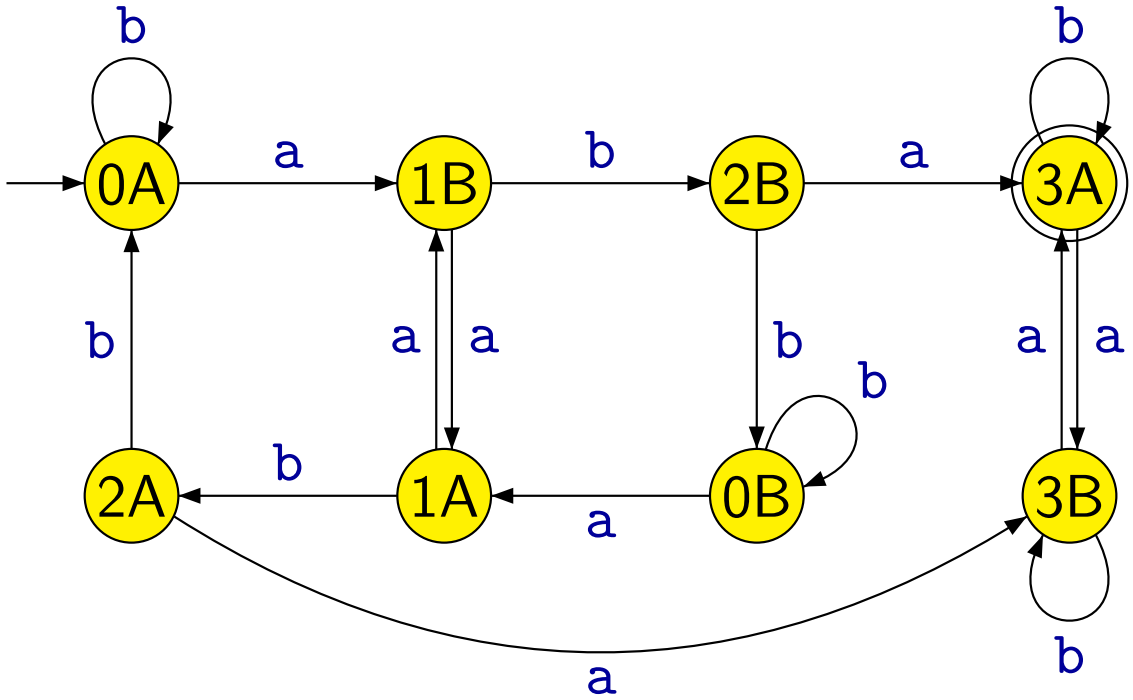




An Automaton for Intersection of Languages







An Automaton for Intersection of Languages

Formally, the construction can be described as follows:

We assume we have two deterministic finite automata $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$.

We construct DFA $A = (Q, \Sigma, \delta, q_0, F)$ 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 $w \in L(A)$ iff $w \in L(A_1)$ and $w \in L(A_2)$, i.e.,

$$L(A) = L(A_1) \cap L(A_2)$$

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.

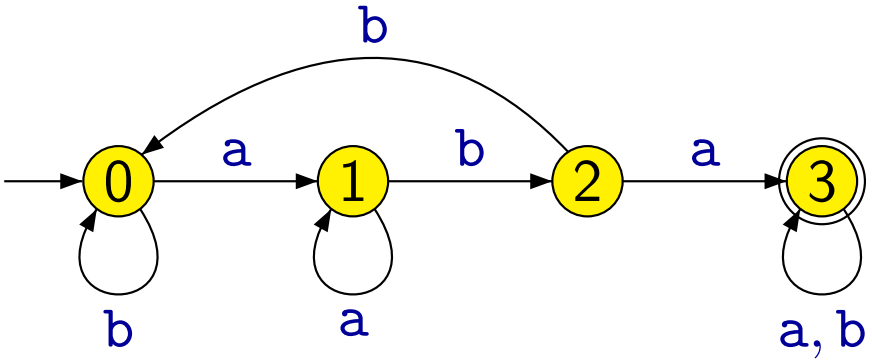
Proof: Let us assume that A_1 and A_2 are deterministic finite automata such that

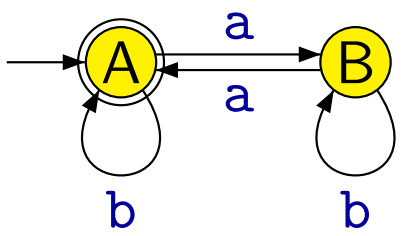
$$L_1 = L(A_1) \qquad L_2 = L(A_2)$$

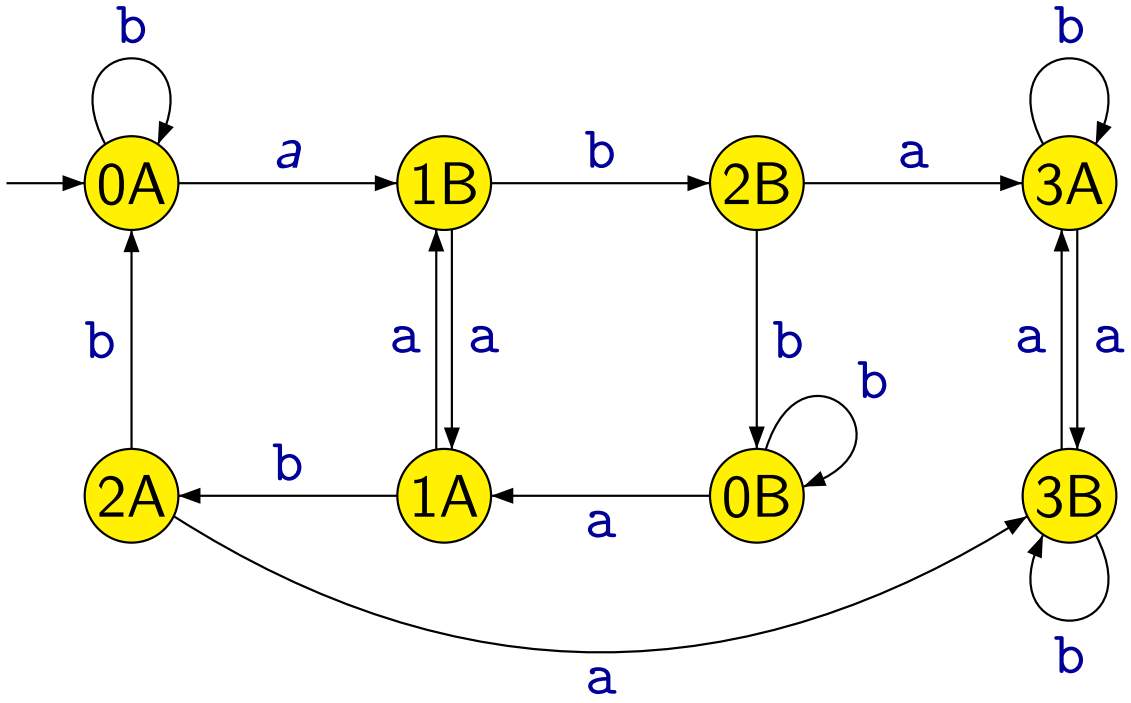
Using the described construction, we can construct a deterministic finite automaton A such that

$$L(A) = L(A_1) \cap L(A_2) = L_1 \cap L_2$$

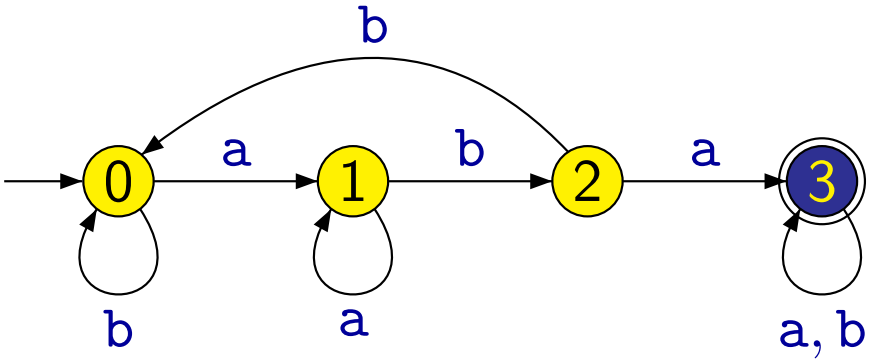
An Automaton for the Union of Languages

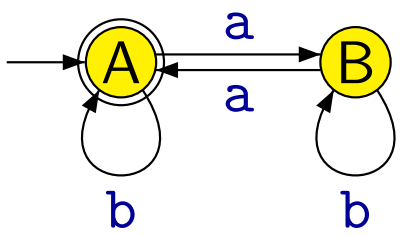


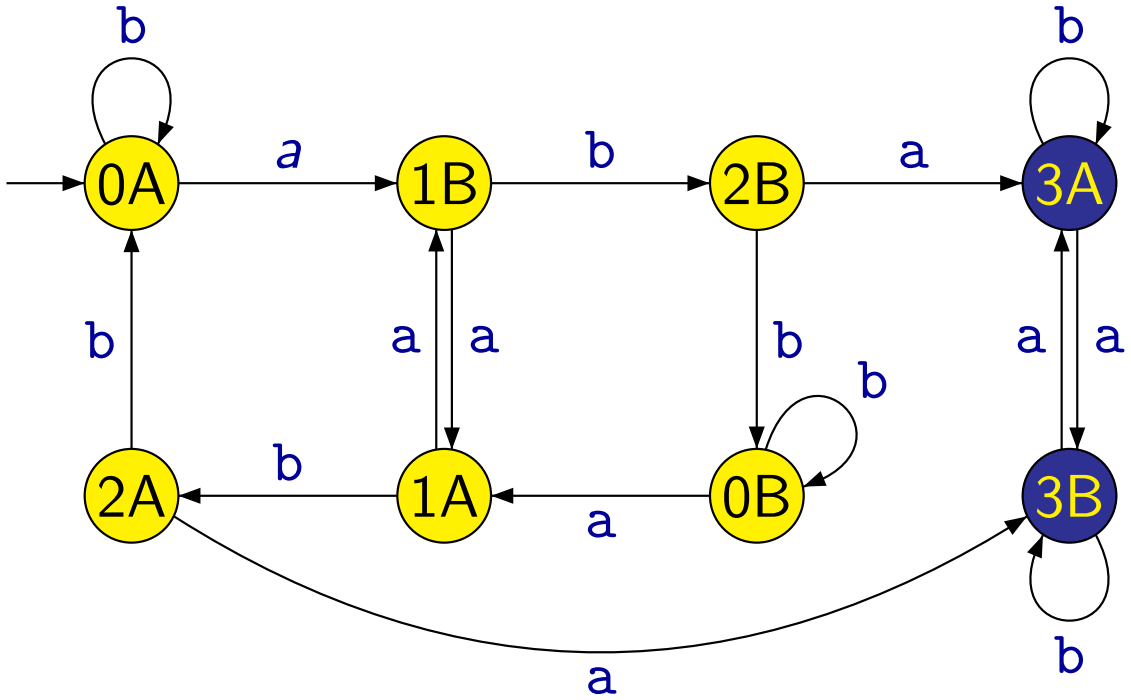




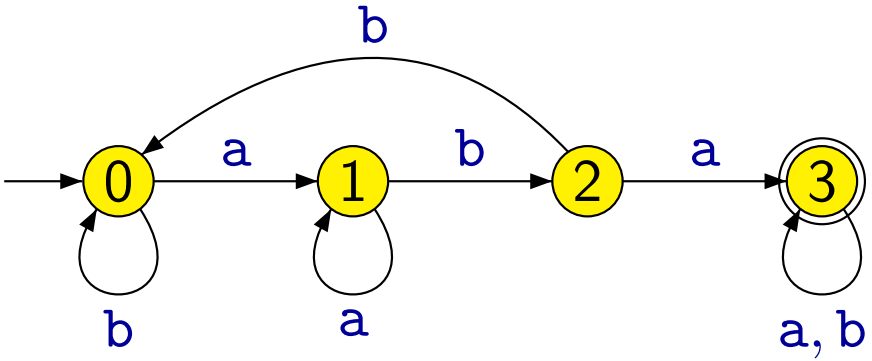
An Automaton for the Union of Languages

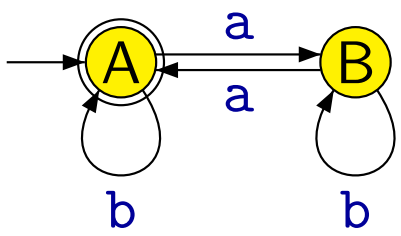


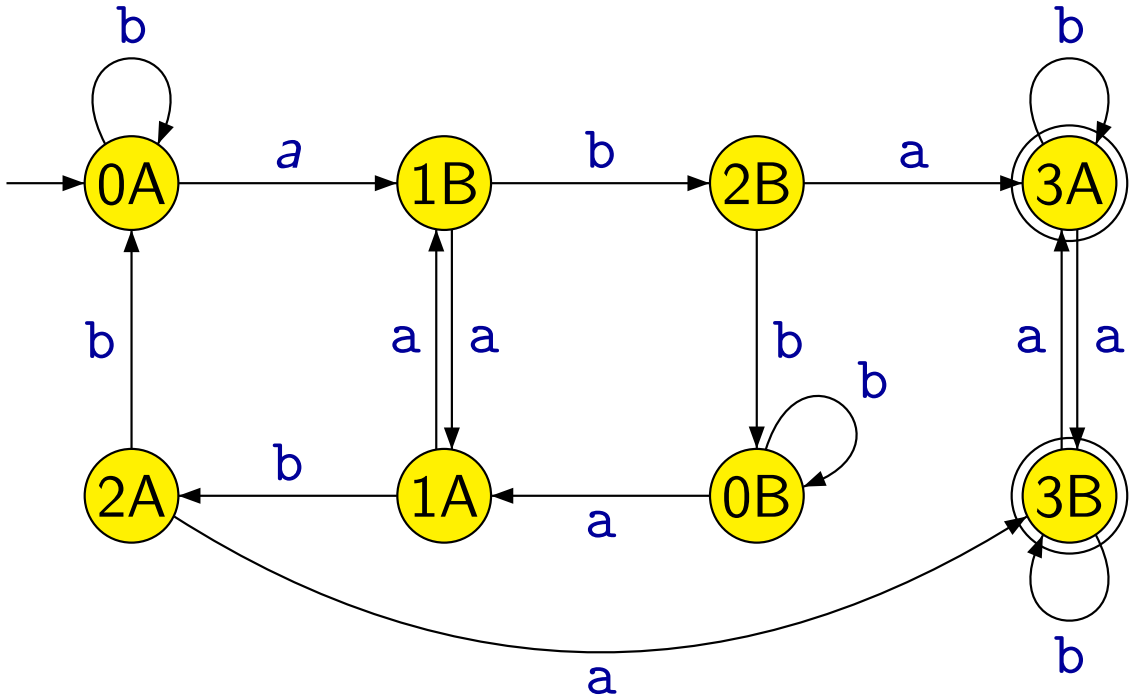




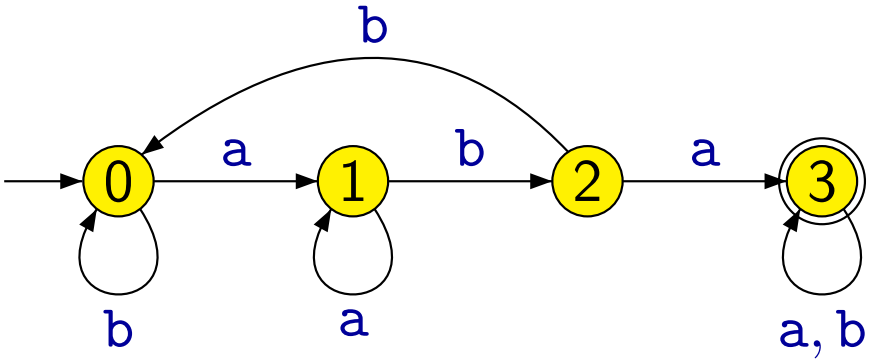
An Automaton for the Union of Languages

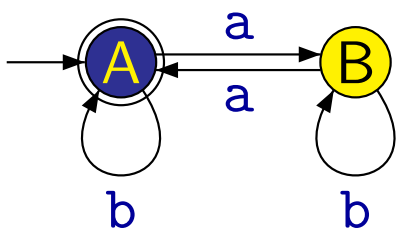


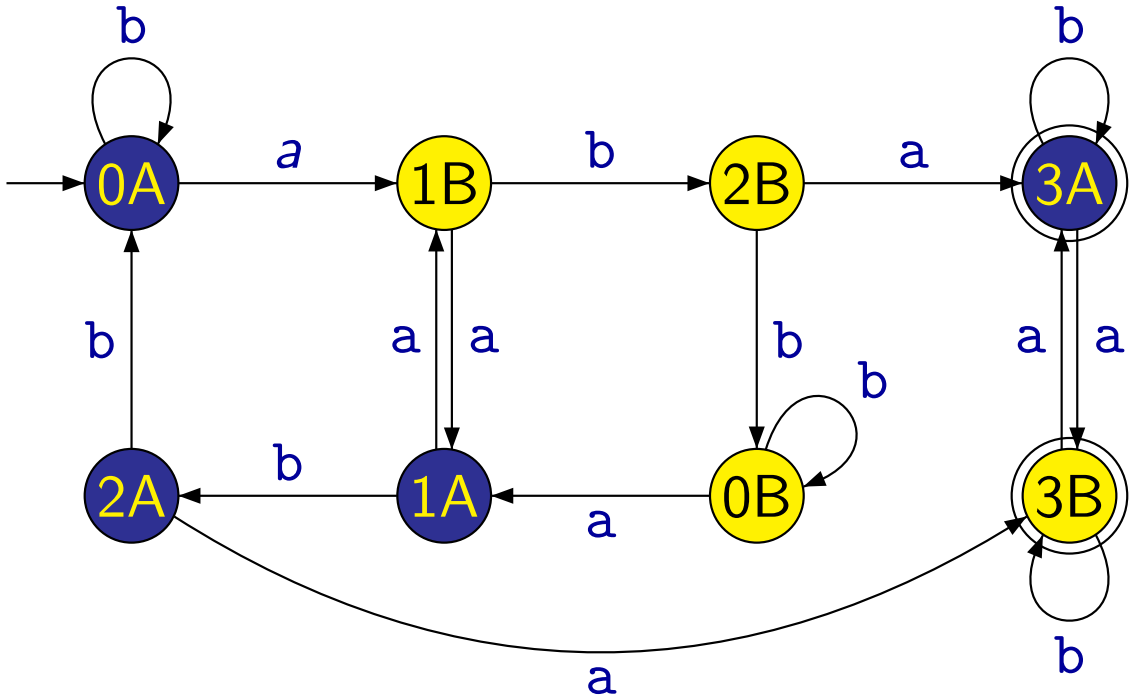




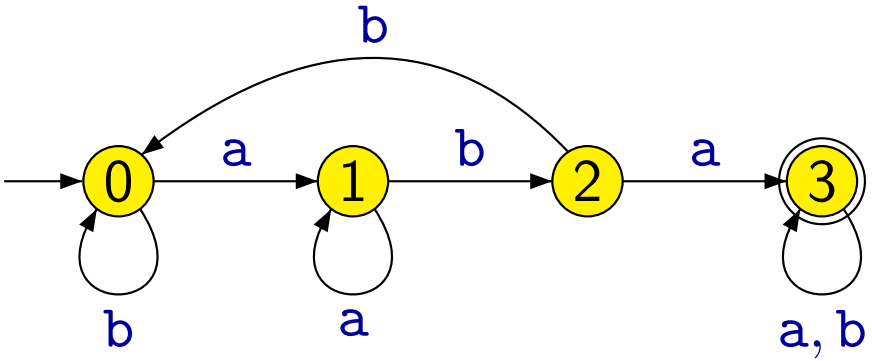
An Automaton for the Union of Languages

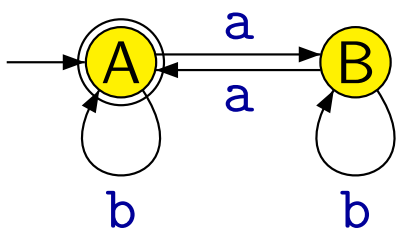


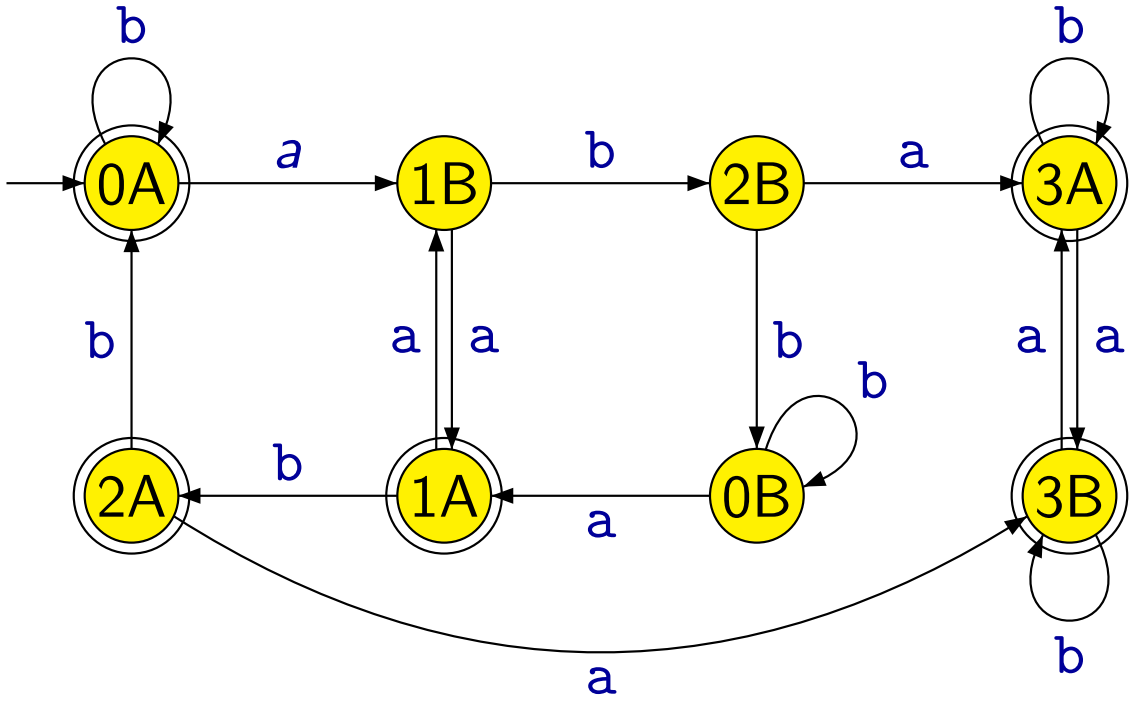




An Automaton for the Union of Languages







Union of Regular Languages

The construction of an automaton A that accepts the **union** of languages accepted by automata A_1 and A_2, i.e., the language

$$L(A_1) \cup L(A_1)$$

is almost identical as in the case of the automaton accepting $L(A_1) \cap L(A_2)$.

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 A that accepts the **union** of languages accepted by automata A_1 and A_2, i.e., the language

$$L(A_1) \cup L(A_1)$$

is almost identical as in the case of the automaton accepting $L(A_1) \cap L(A_2)$.

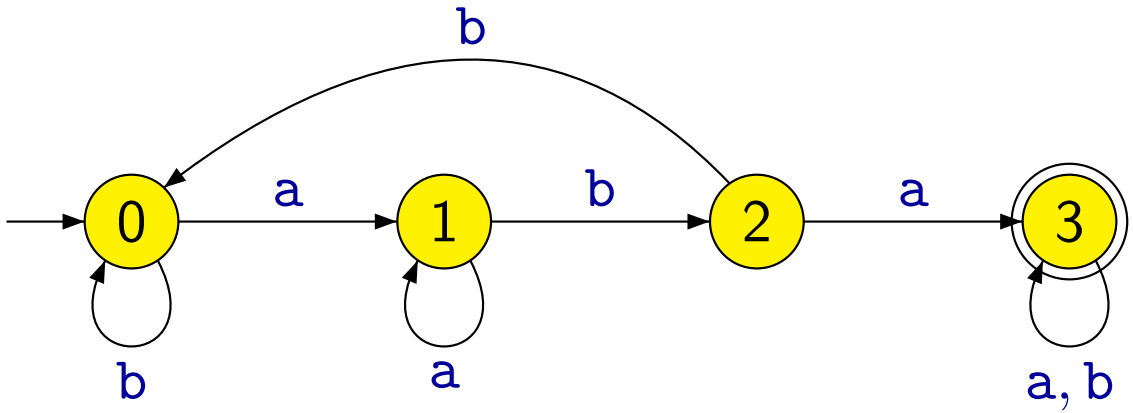
The only difference is the set of accepting states:

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

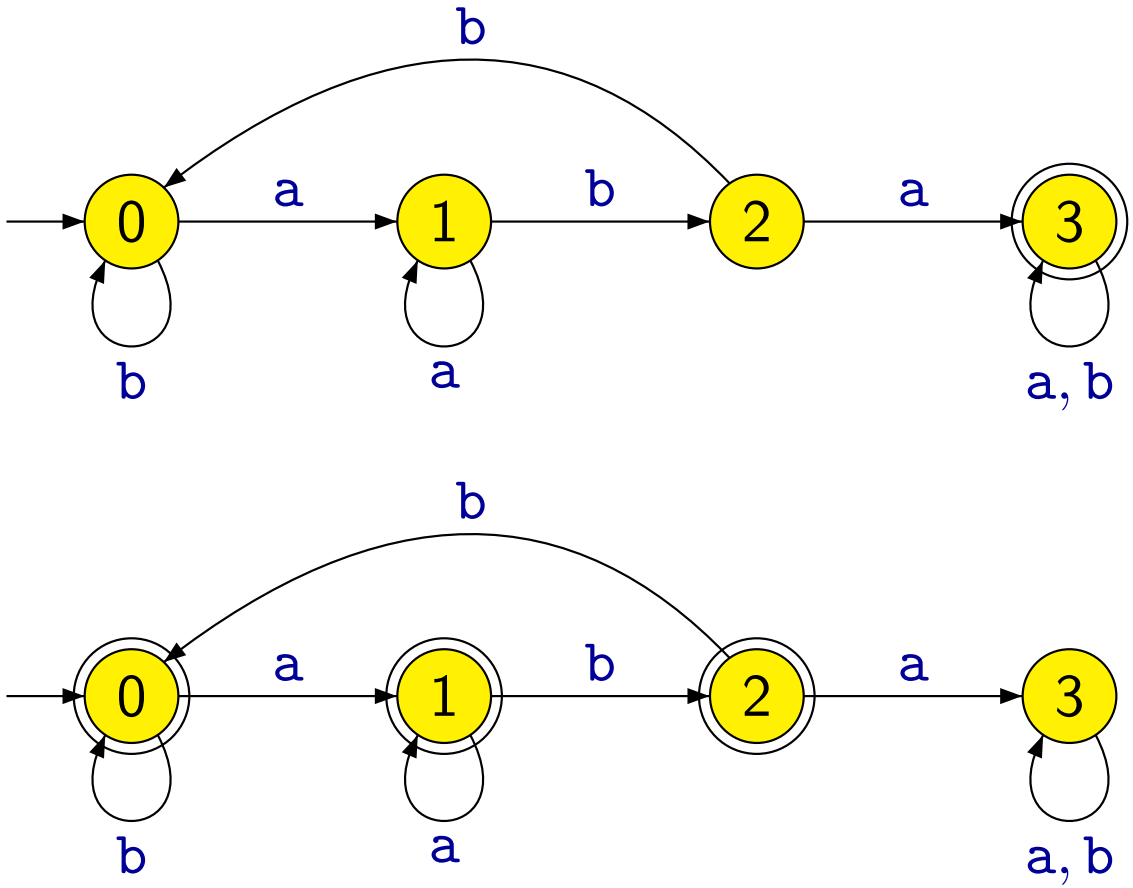
Theorem

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



An Automaton for the Complement of a Language



Complement of a Regular Language

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

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

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

Complement of a Regular Language

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

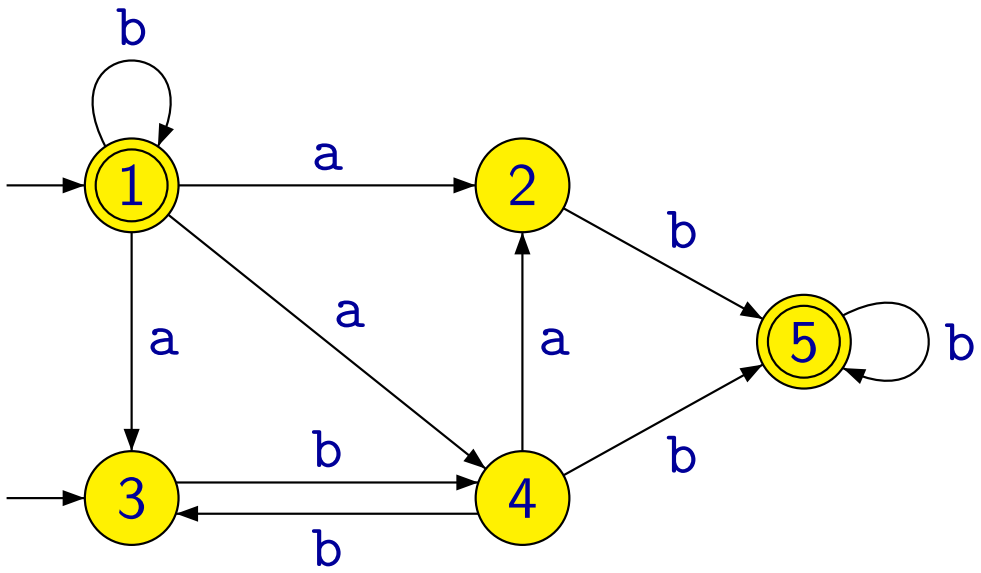
It is obvious that for each word $w \in \Sigma^*$ we have $w \in L(A')$ iff $w \notin L(A)$, i.e.,

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

Theorem

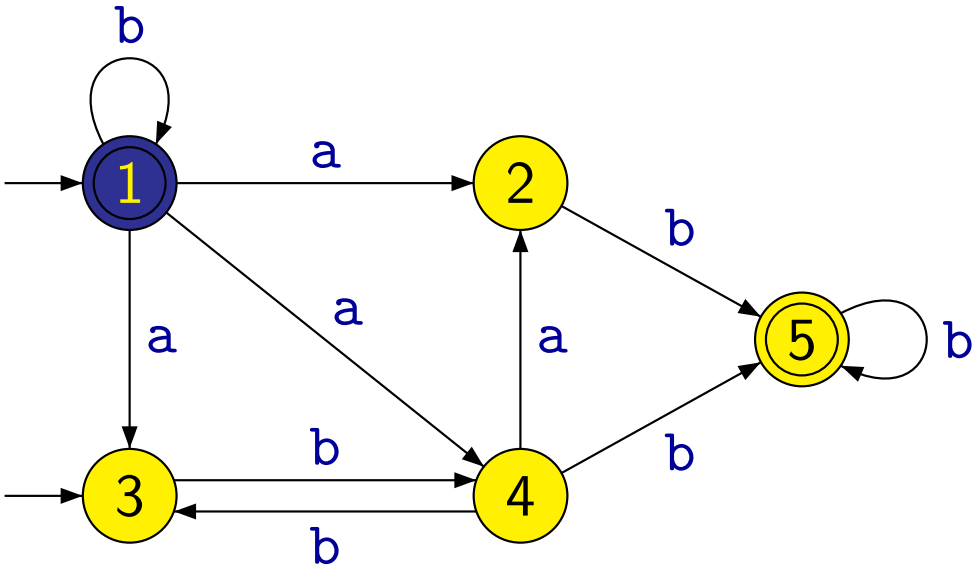
If a language L is regular then also its complement $\overline{L}$ is regular.

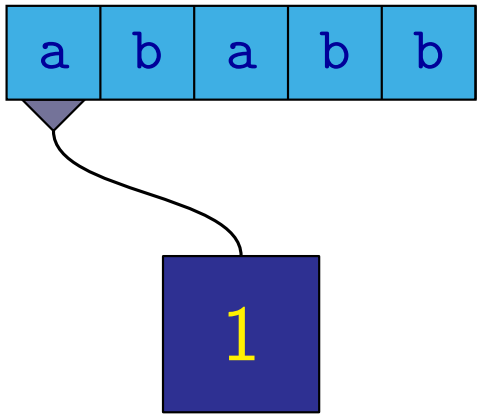
Nondeterministic Finite Automaton



- 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.

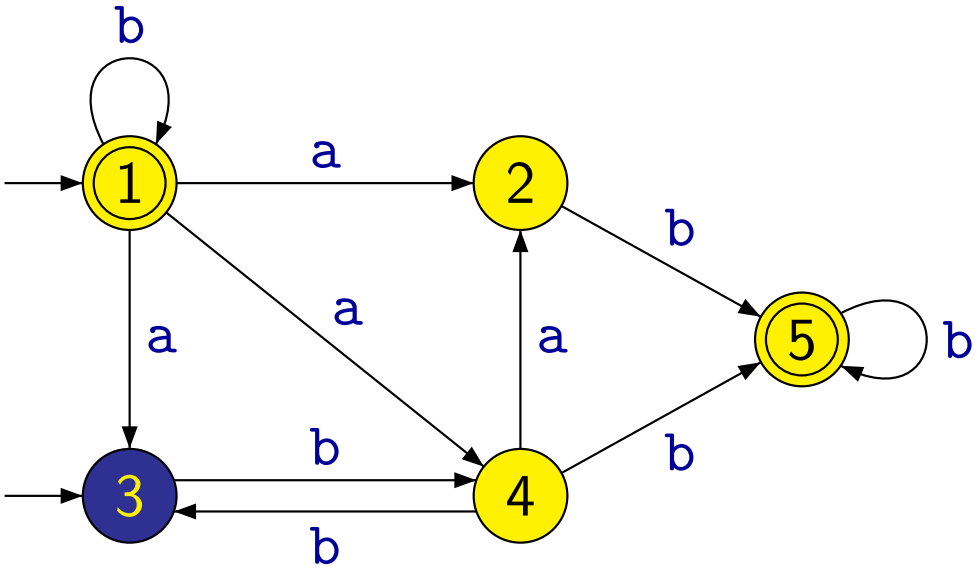
Nondeterministic Finite Automaton

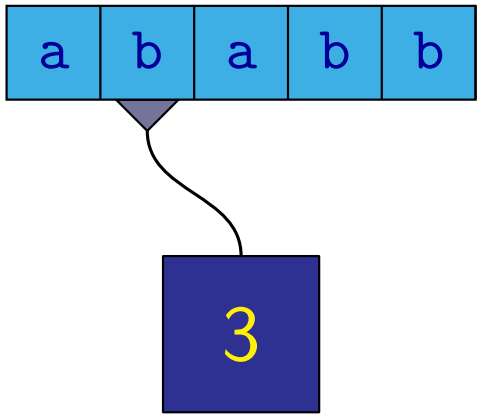




1

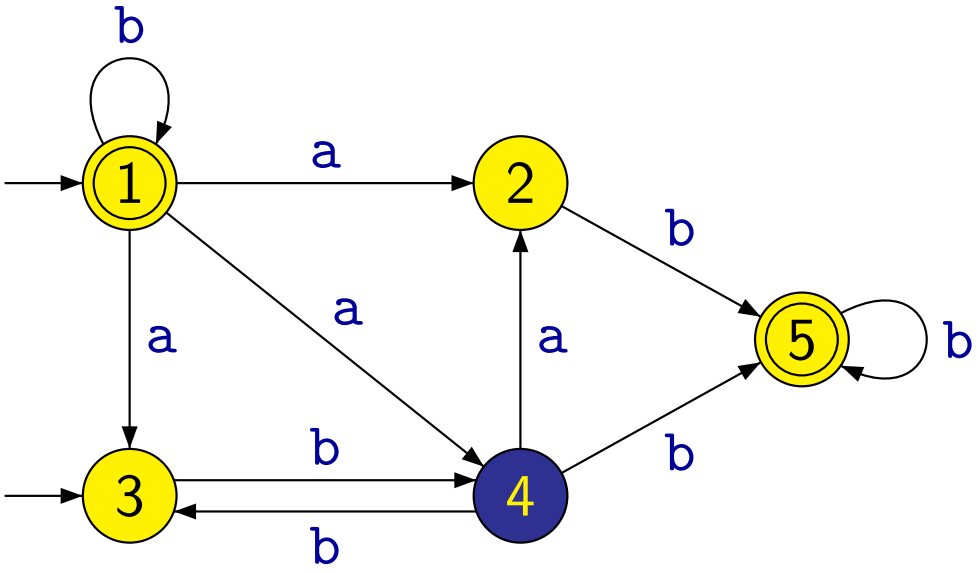
Nondeterministic Finite Automaton

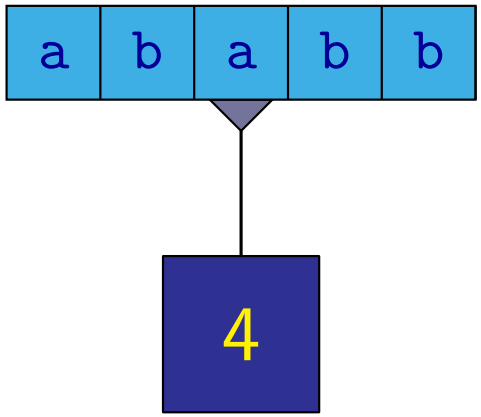




$1 \xrightarrow{a} 3$

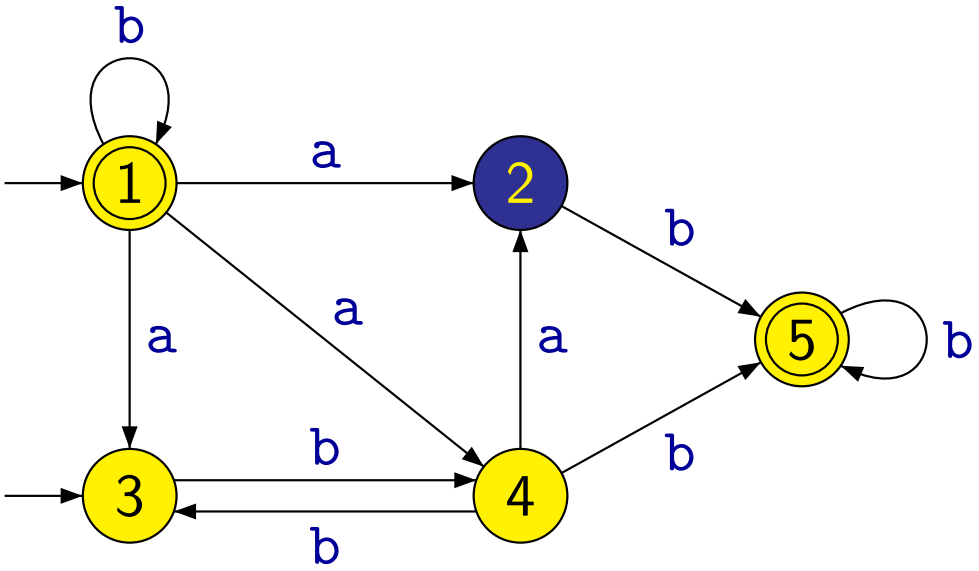
Nondeterministic Finite Automaton

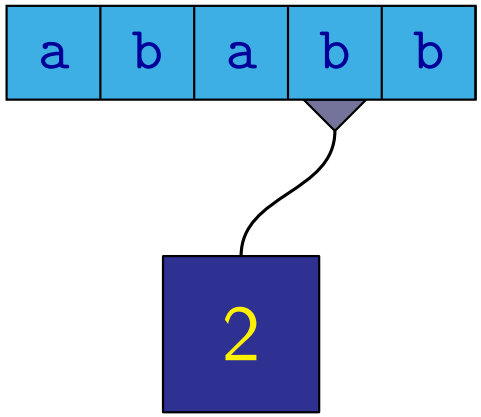




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

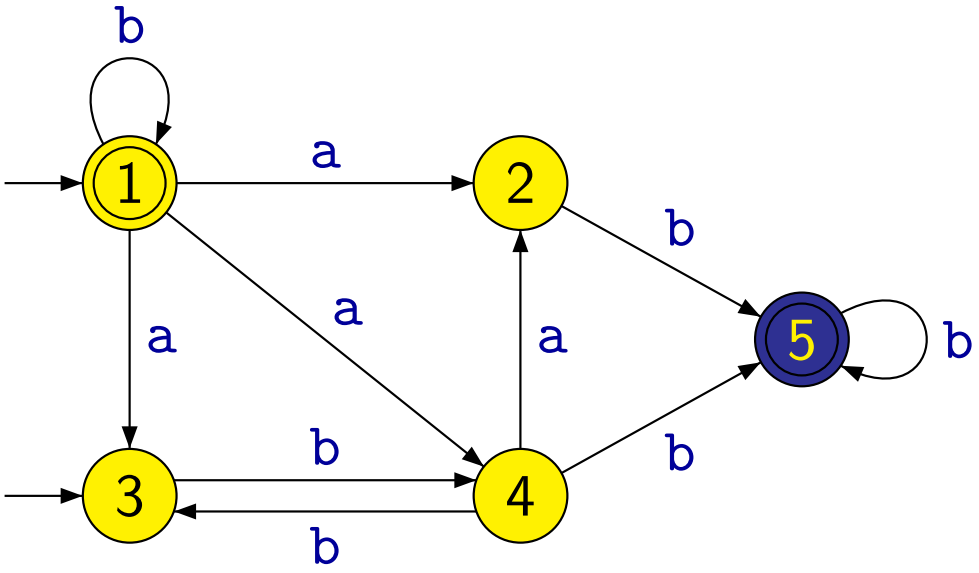
Nondeterministic Finite Automaton

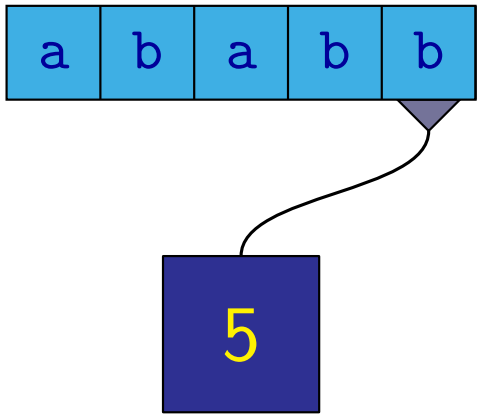




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2$$

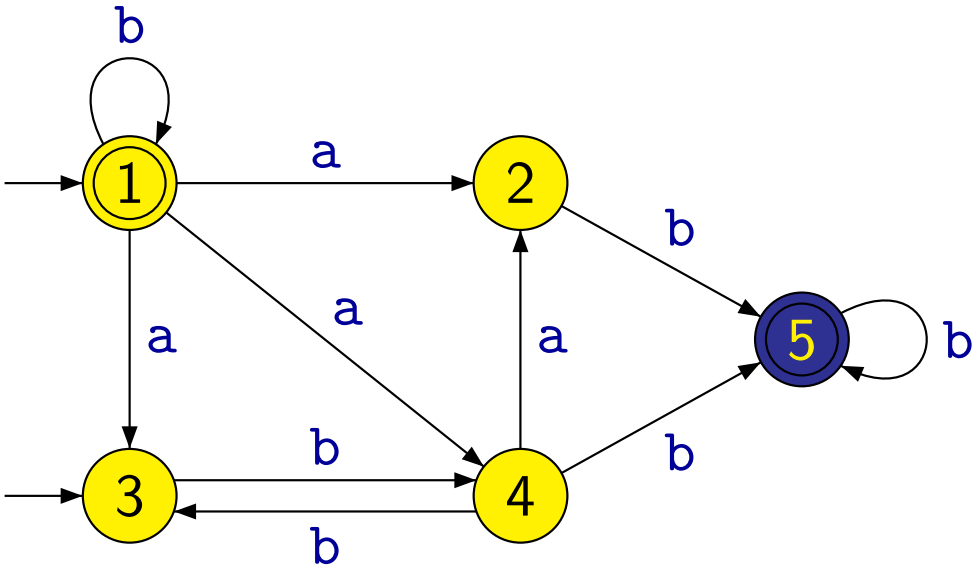
Nondeterministic Finite Automaton

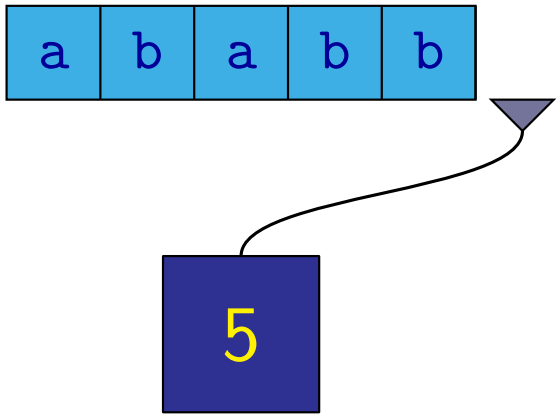




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5$$

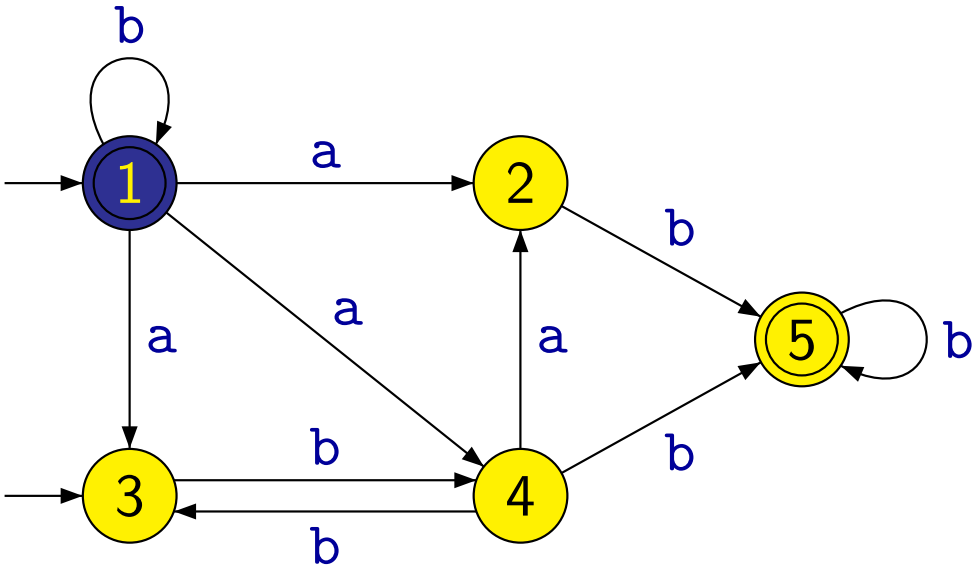
Nondeterministic Finite Automaton

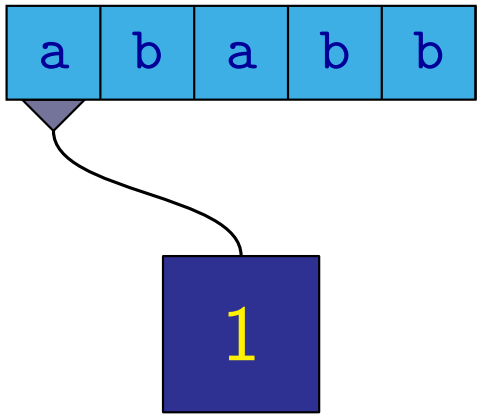




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{b} 5$$

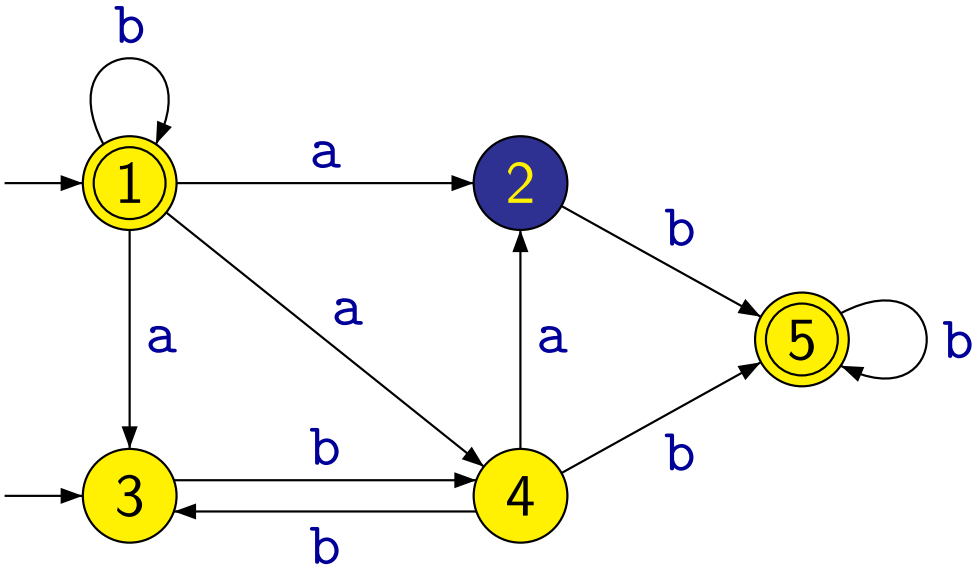
Nondeterministic Finite Automaton

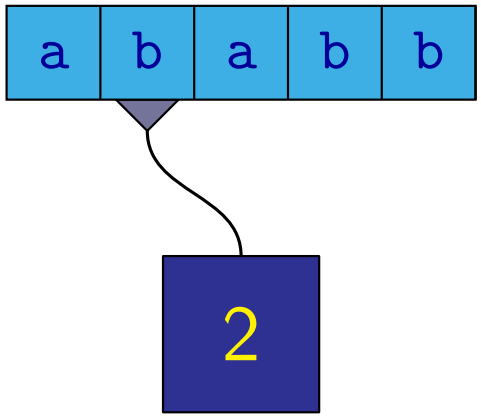




1

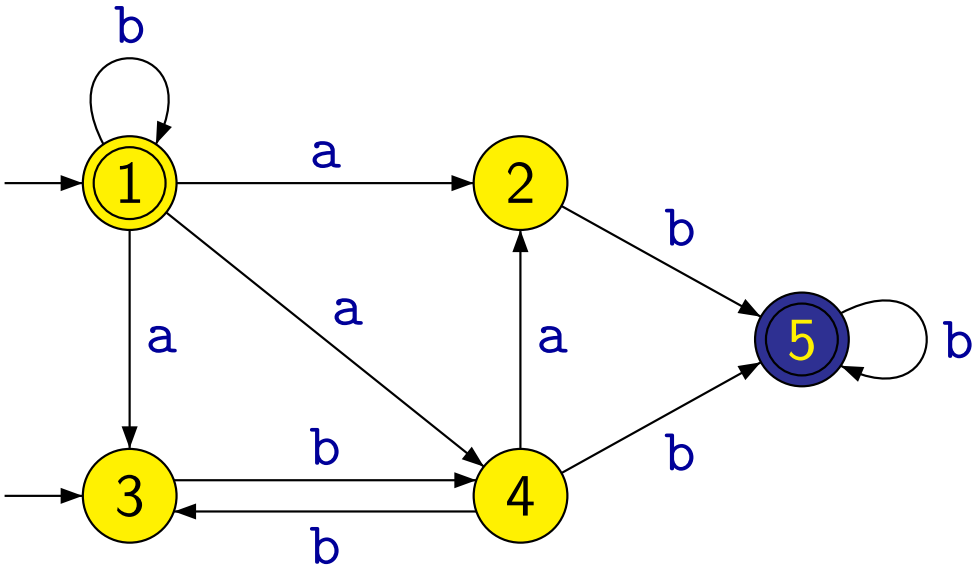
Nondeterministic Finite Automaton

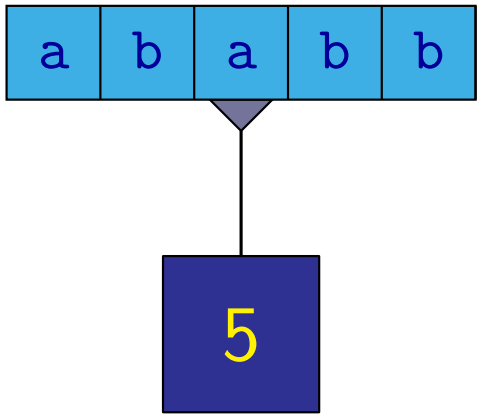




$1 \xrightarrow{a} 2$

Nondeterministic Finite Automaton

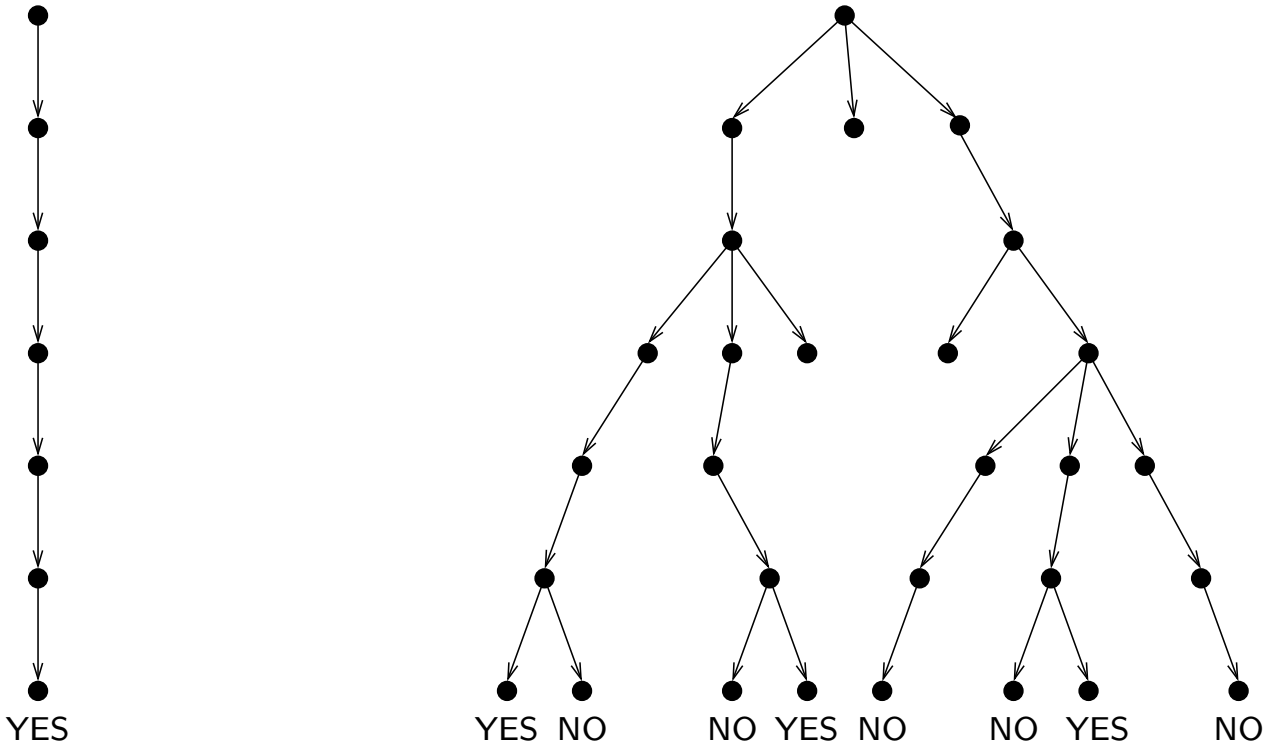




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5$$

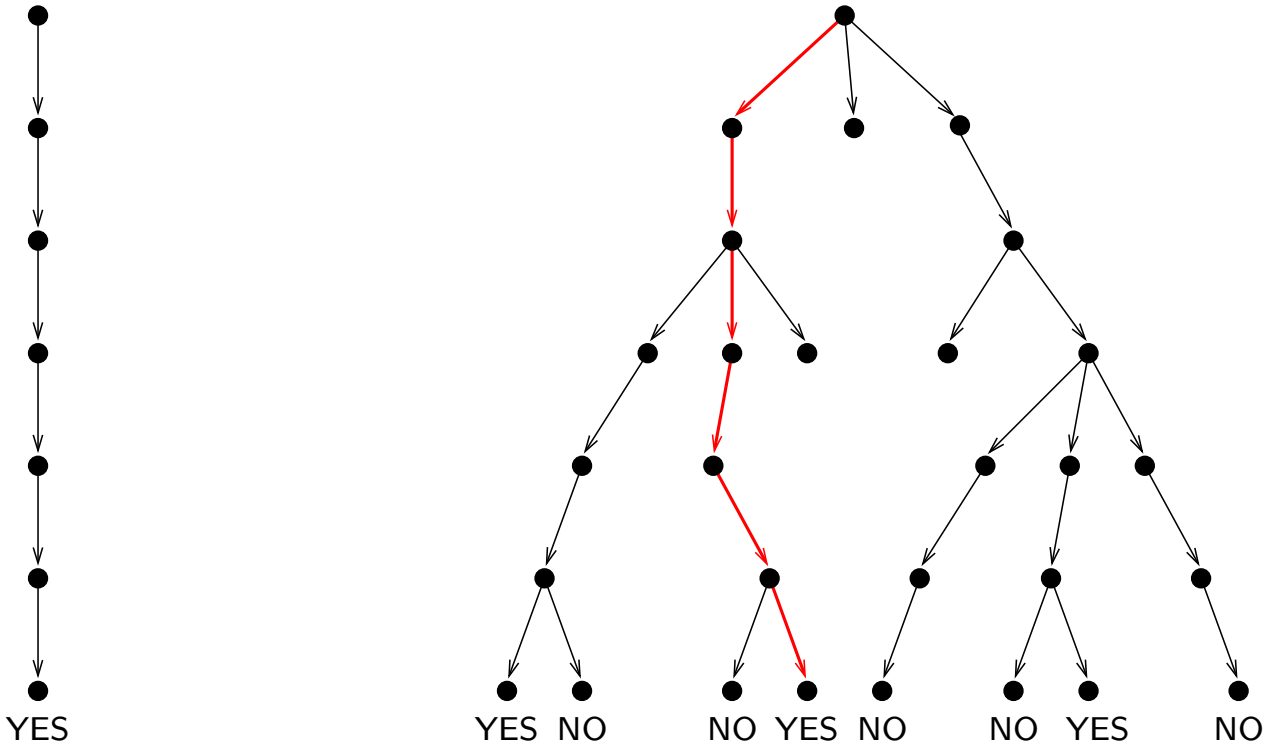
Nondeterministic Finite Automaton

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



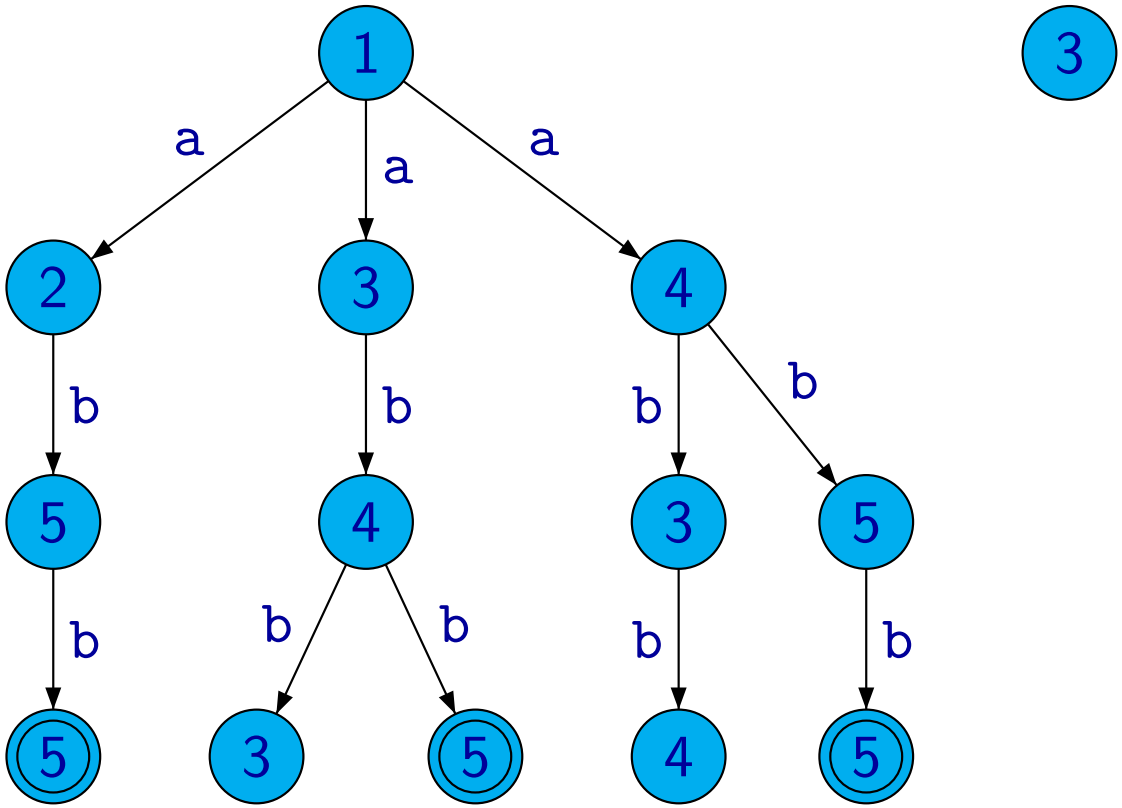
Nondeterministic Finite Automaton

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	2	$3, 5$
$\leftarrow 5$	$-$	5



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

Nondeterministic Finite Automaton

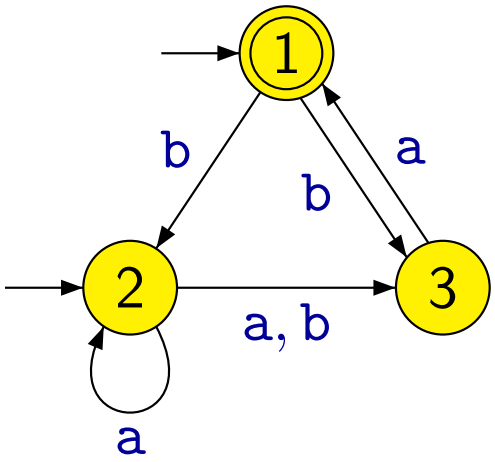
Formally, a **nondeterministic finite automaton** (**NFA**) is defined as a tuple

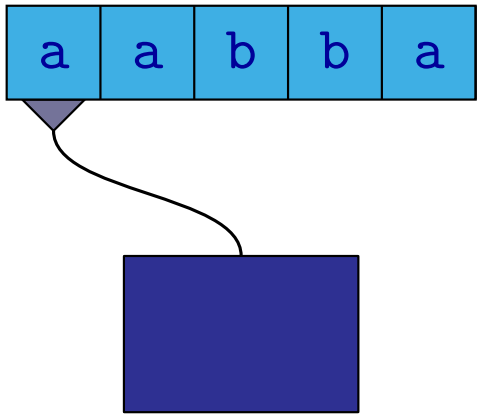
$$(Q, \Sigma, \delta, I, F)$$

where:

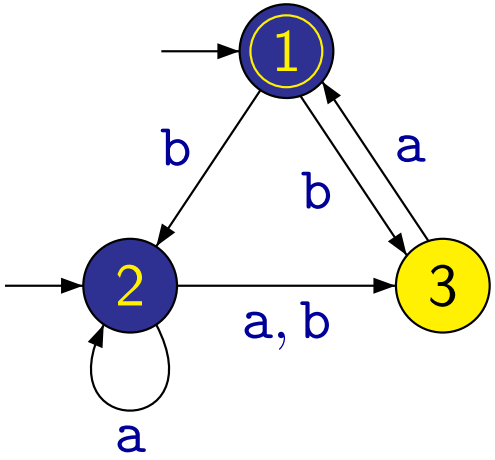
- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is a **transition fuction**
- $I \subseteq Q$ is a set of **initial states**
- $F \subseteq Q$ is a set of **accepting states**

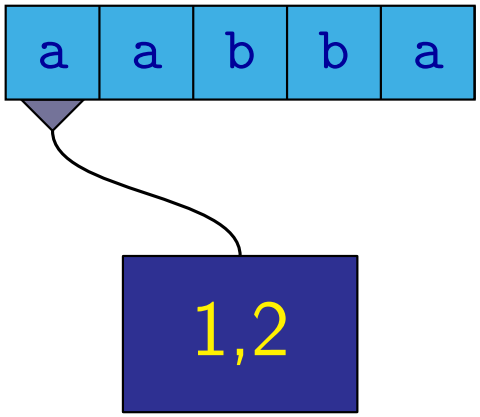
Transformation of NFA to DFA



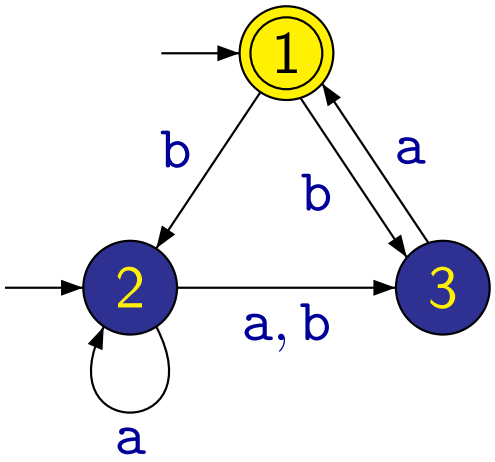


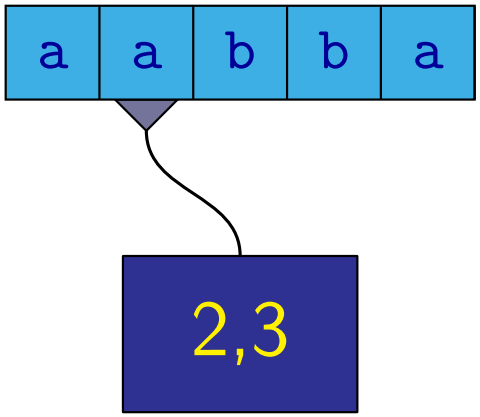
Transformation of NFA to DFA



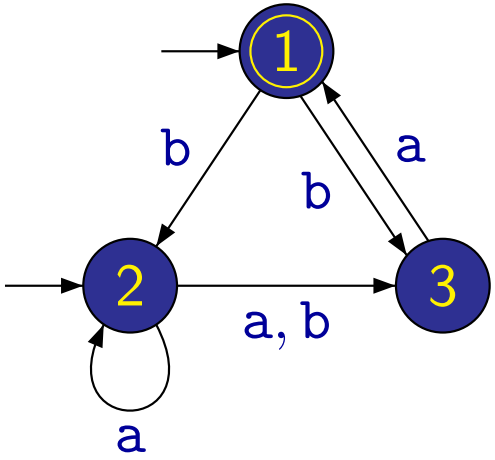


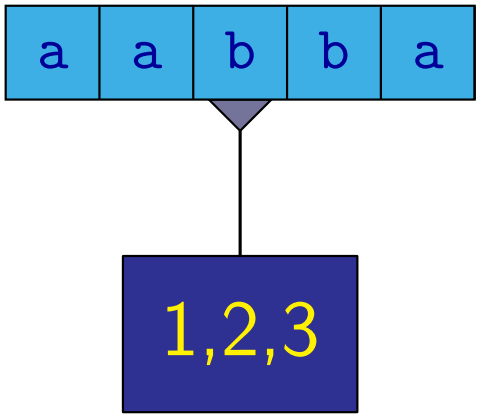
Transformation of NFA to DFA



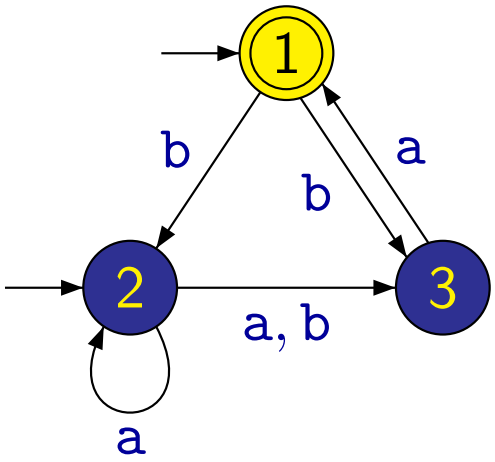


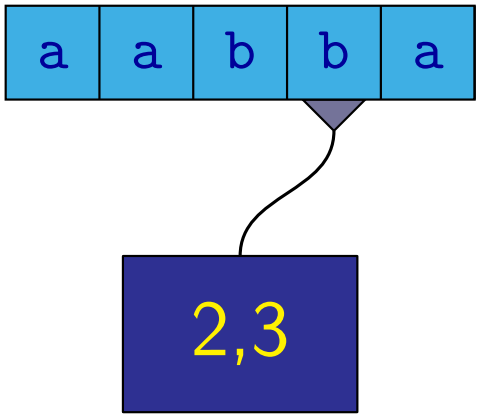
Transformation of NFA to DFA



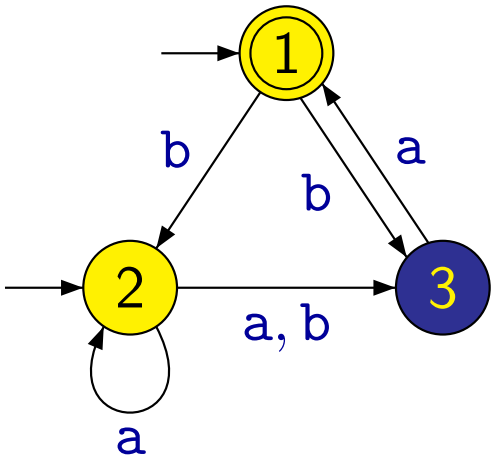


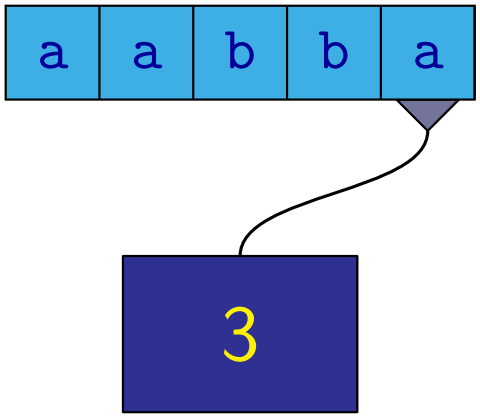
Transformation of NFA to DFA



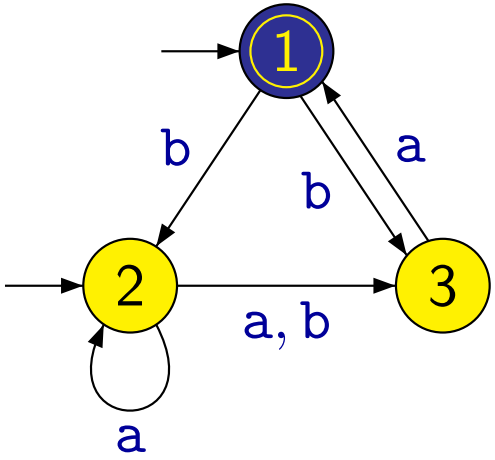


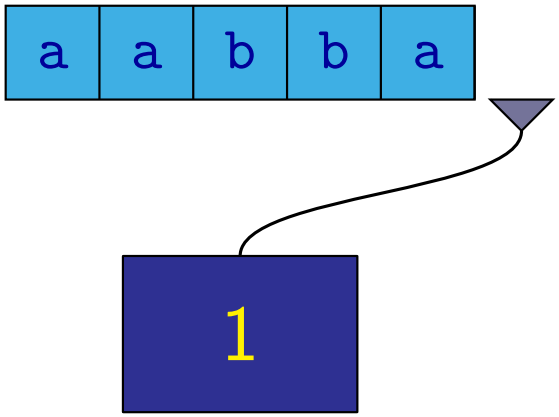
Transformation of NFA to DFA



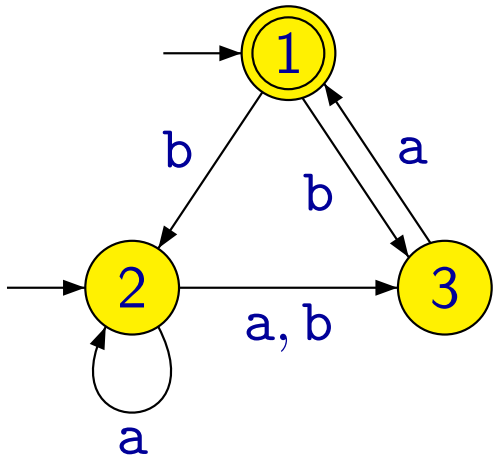


Transformation of NFA to DFA

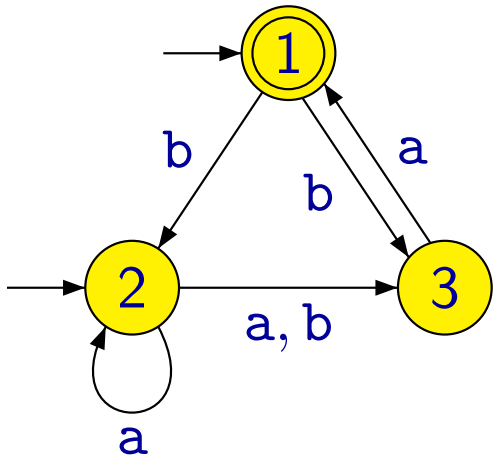




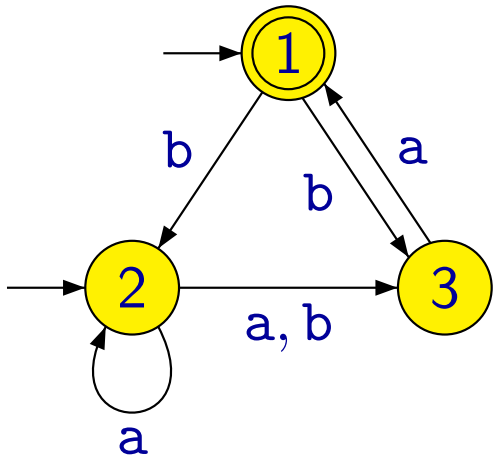
Transformation of NFA to DFA

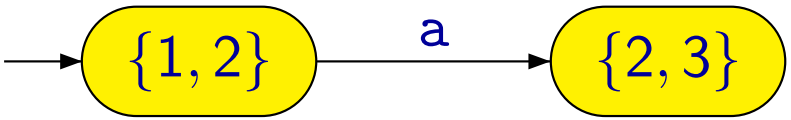


Transformation of NFA to DFA

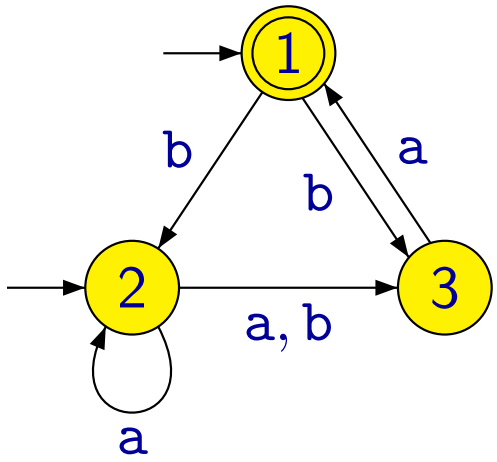


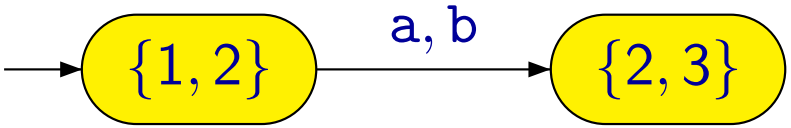
Transformation of NFA to DFA



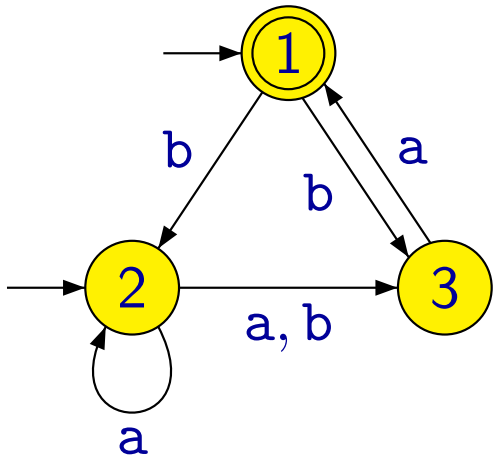


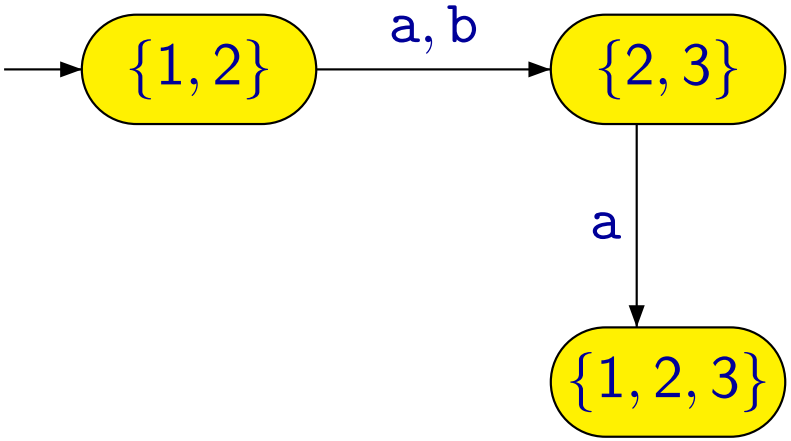
Transformation of NFA to DFA



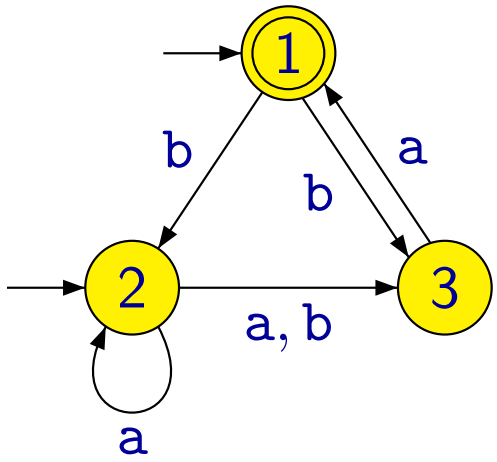


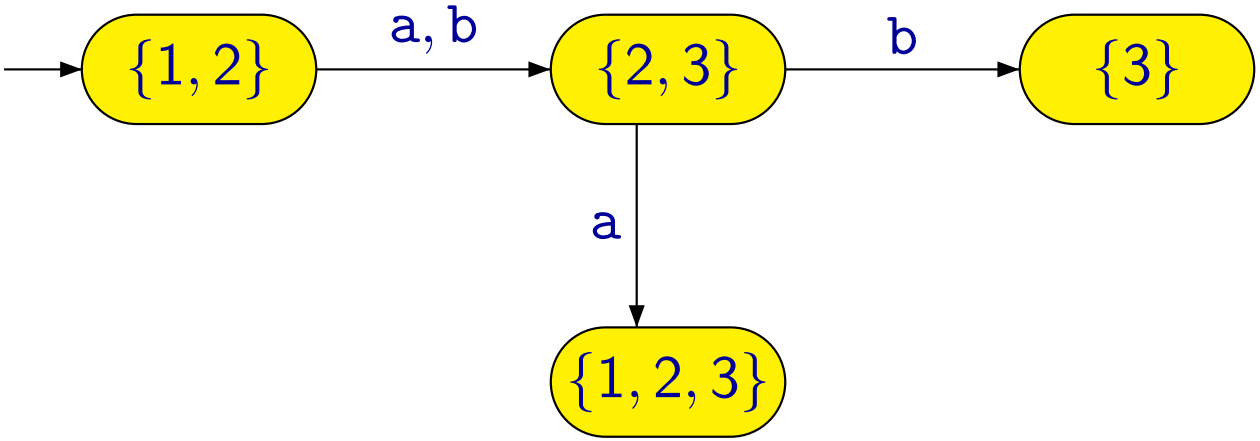
Transformation of NFA to DFA



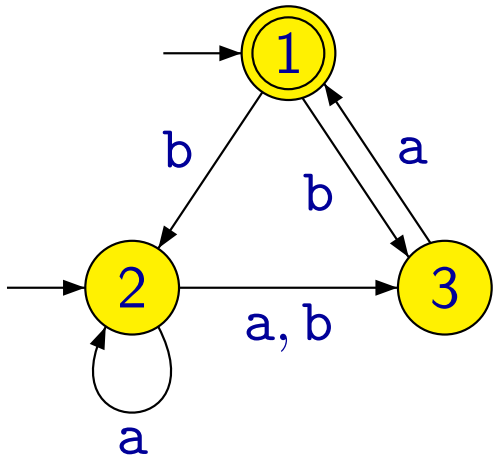


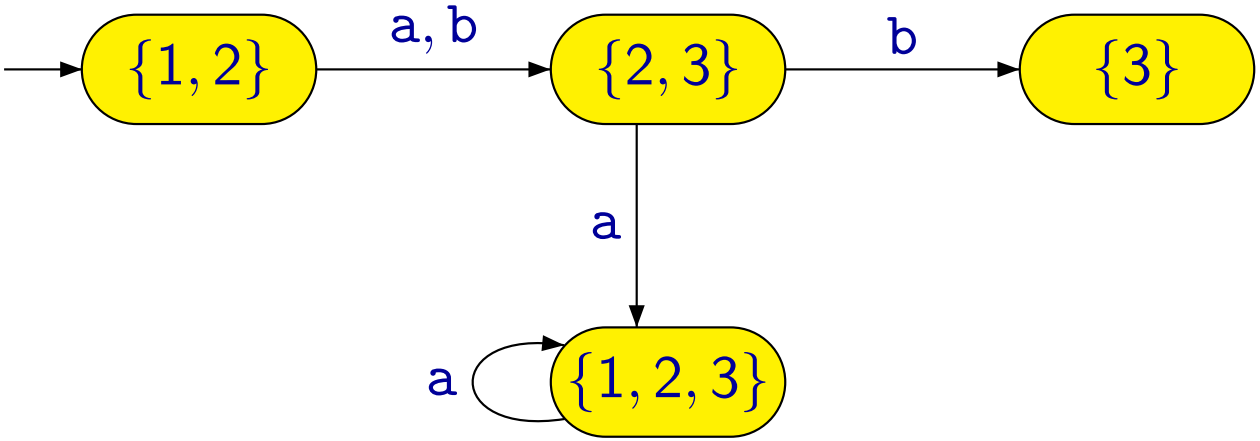
Transformation of NFA to DFA



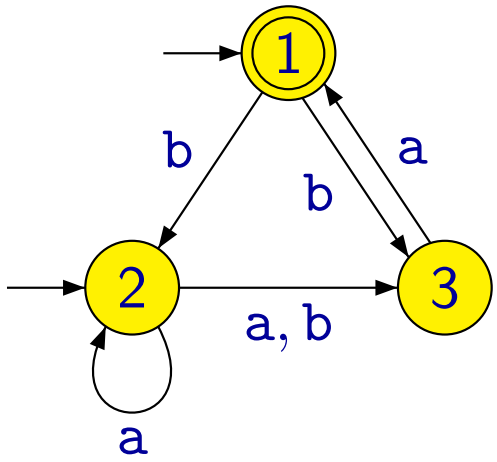


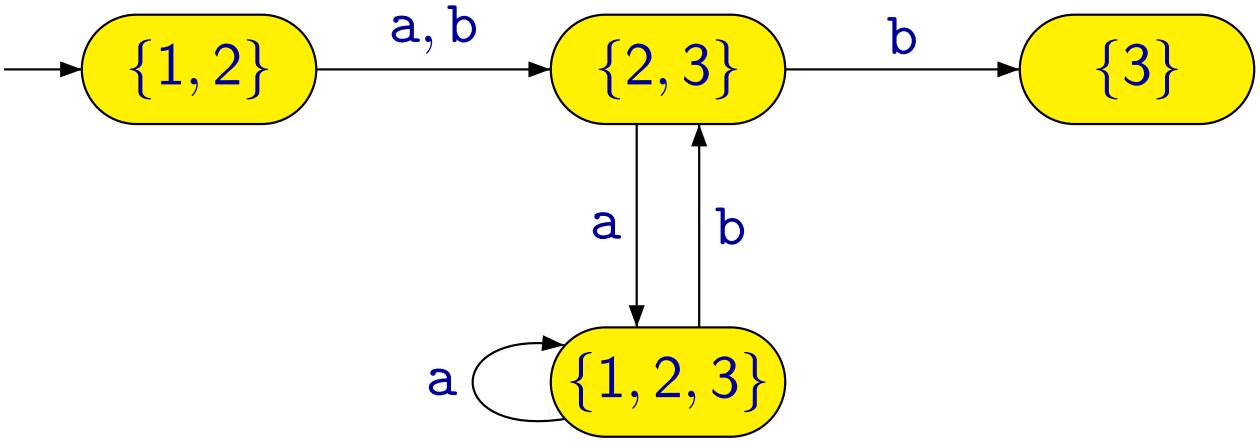
Transformation of NFA to DFA



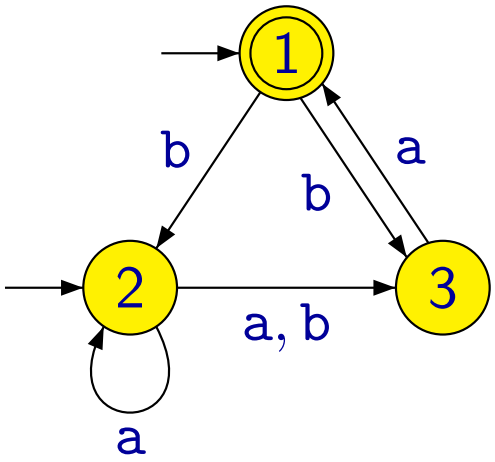


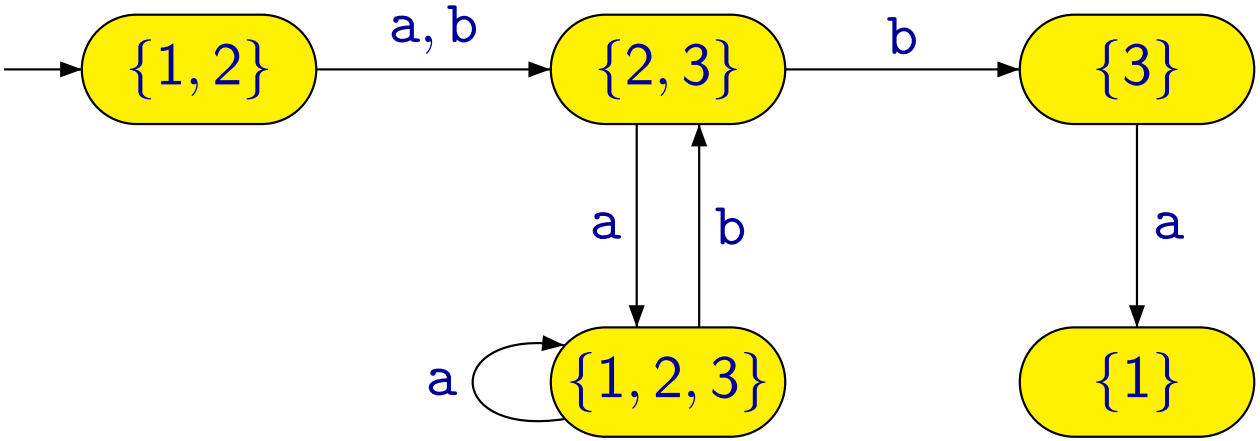
Transformation of NFA to DFA



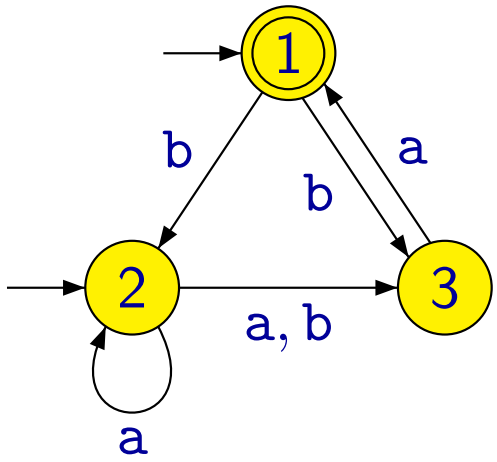


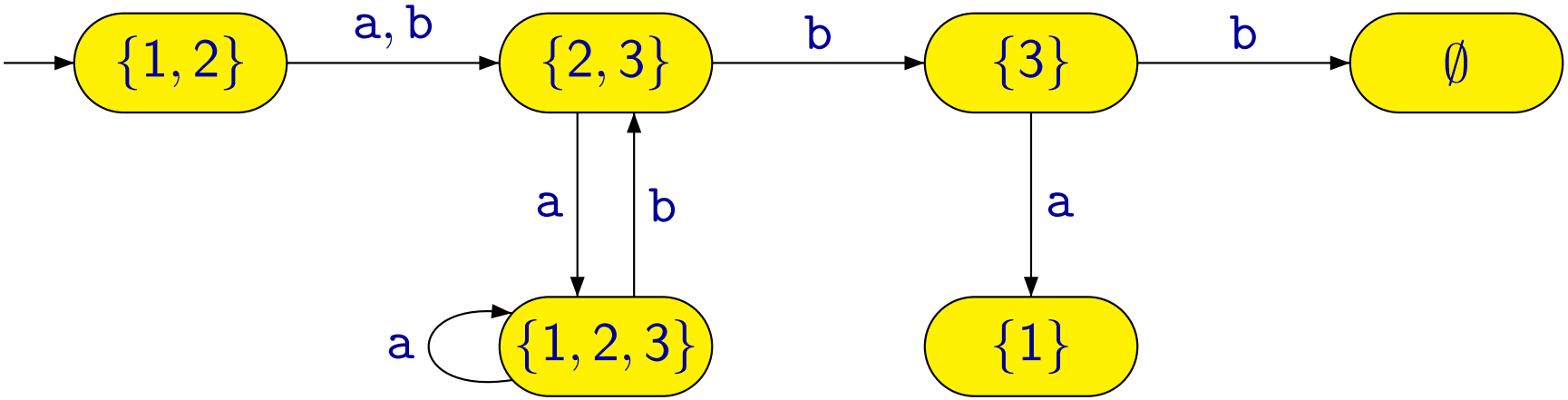
Transformation of NFA to DFA



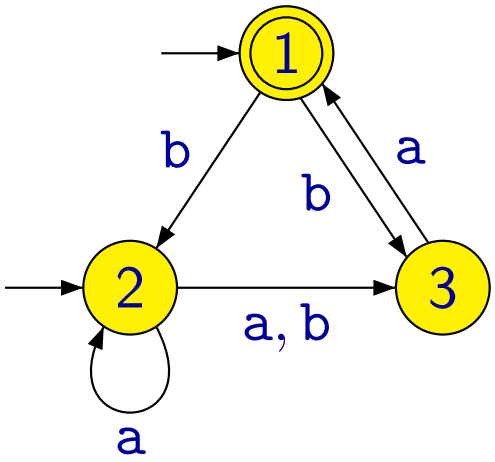


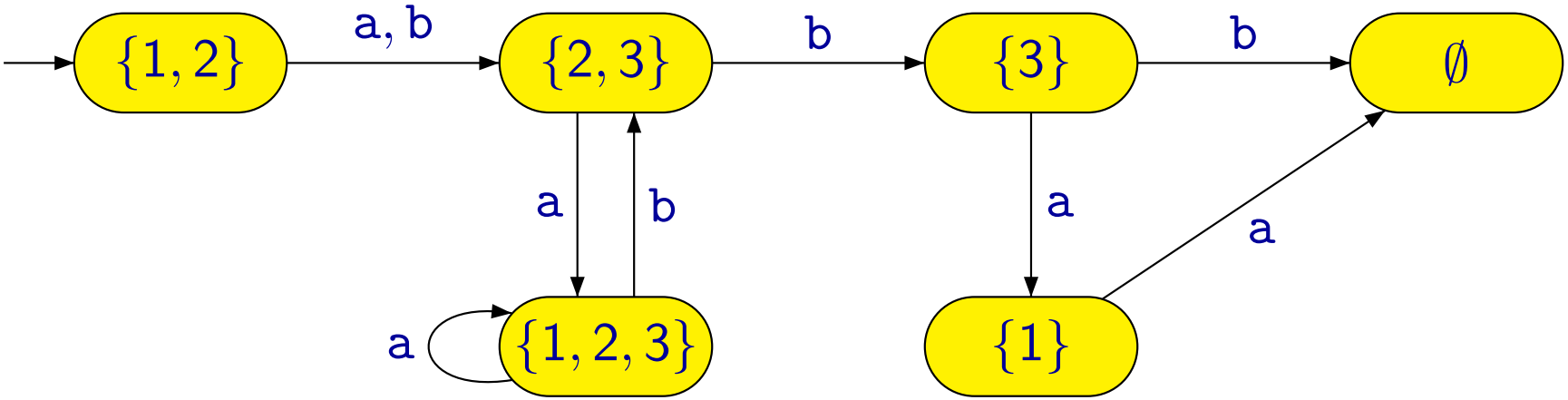
Transformation of NFA to DFA



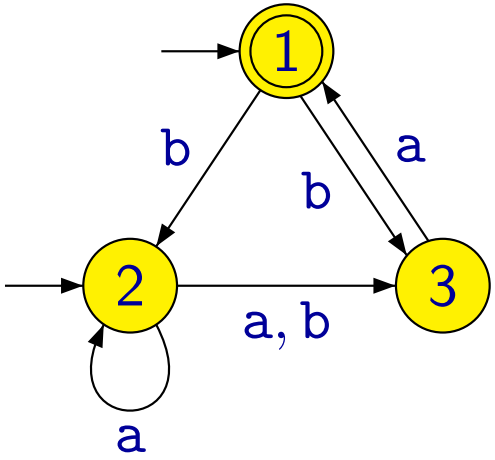


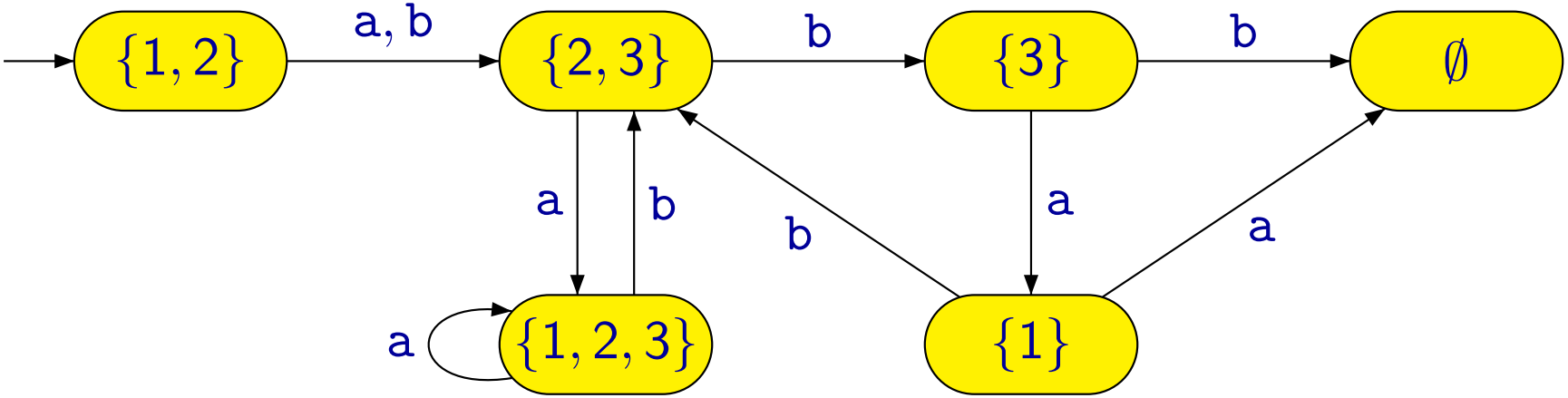
Transformation of NFA to DFA



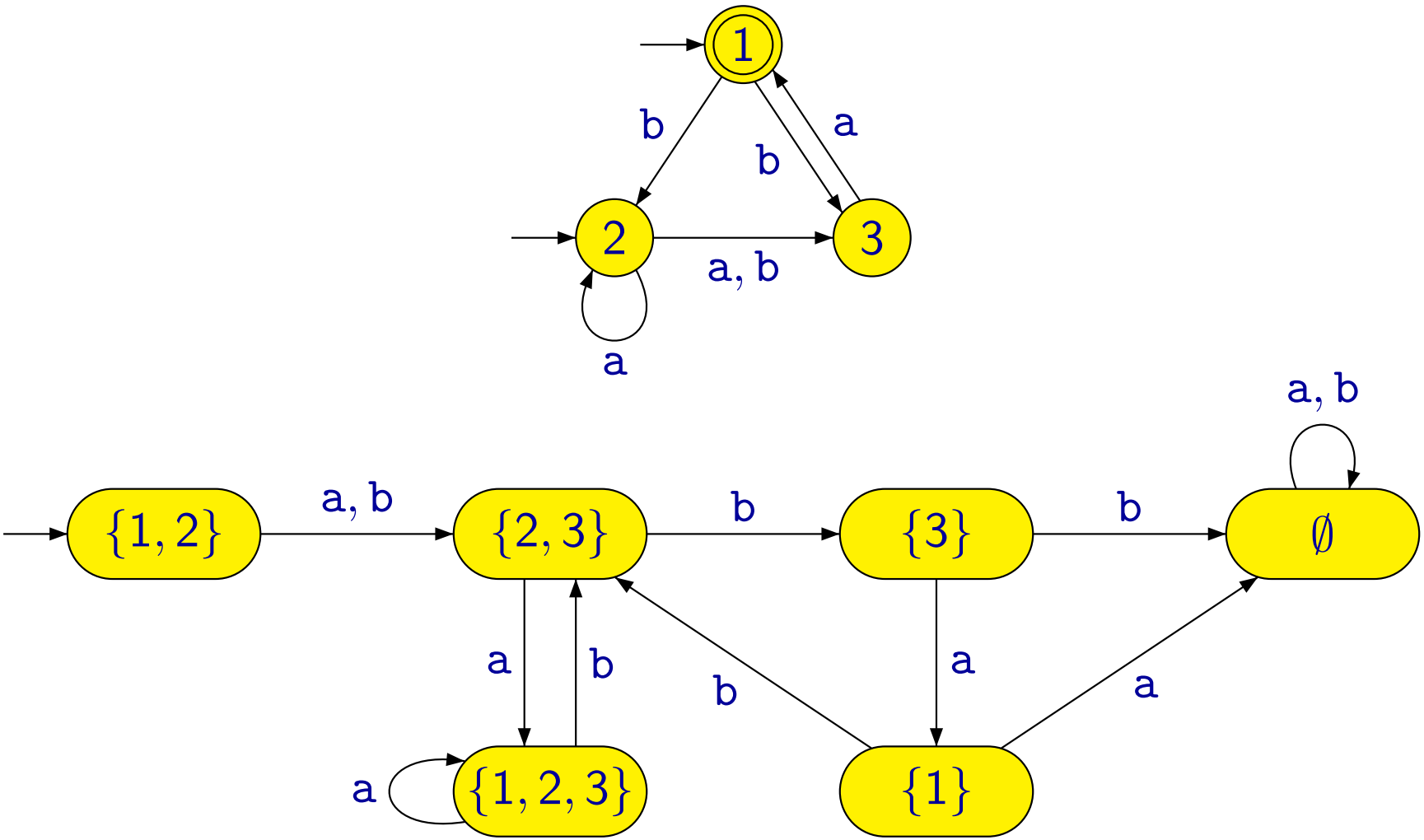


Transformation of NFA to DFA

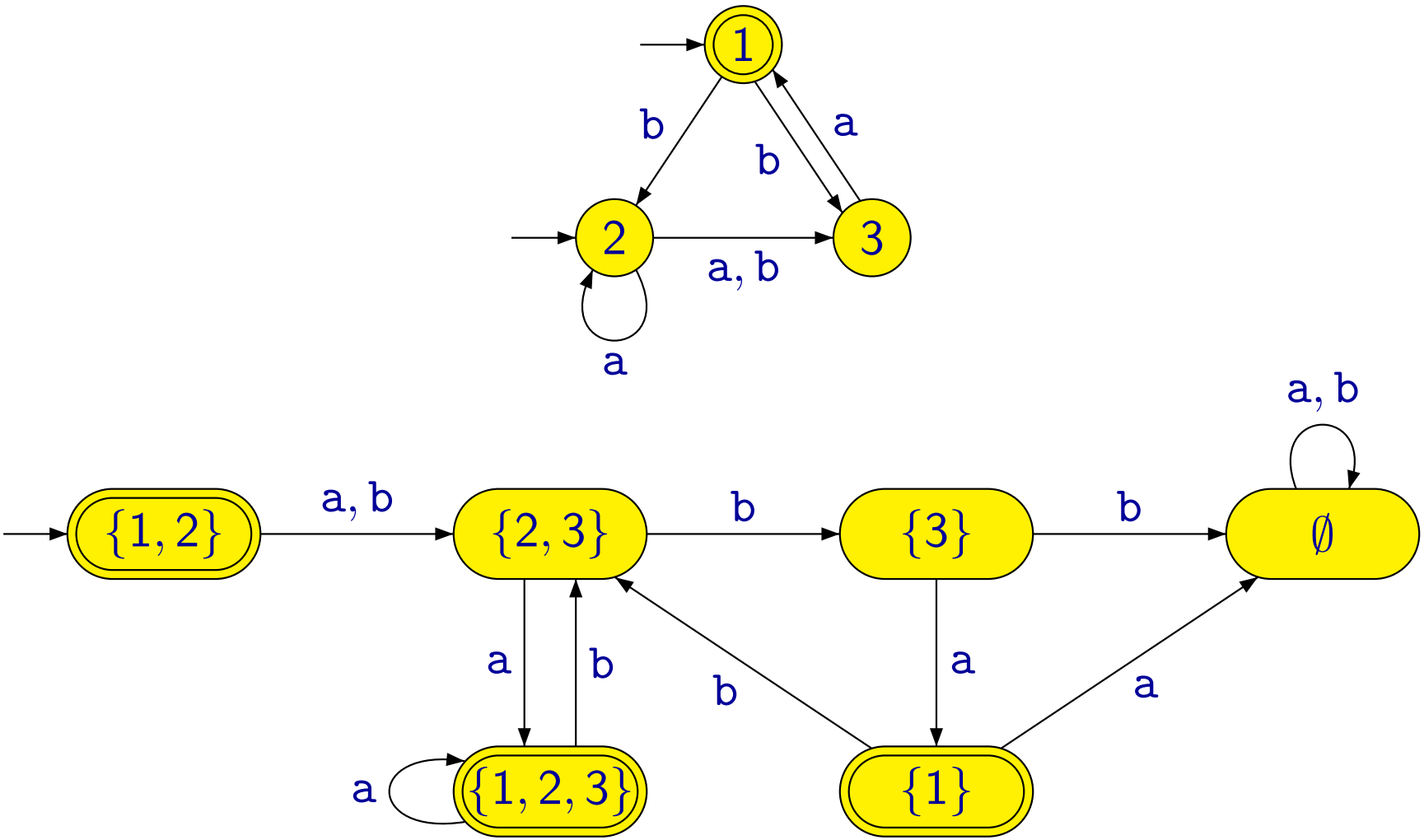




Transformation of NFA to DFA



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

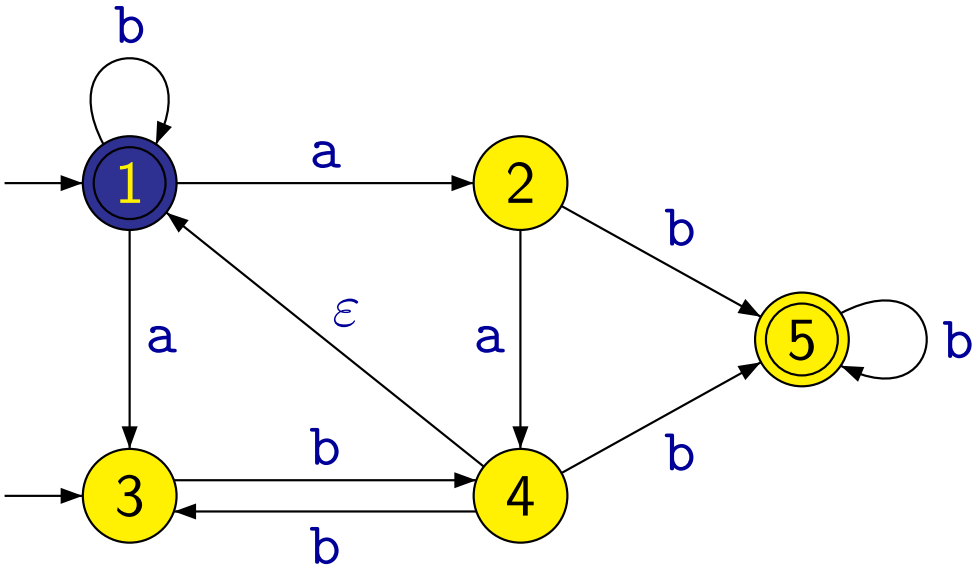
Transformation of NFA to DFA

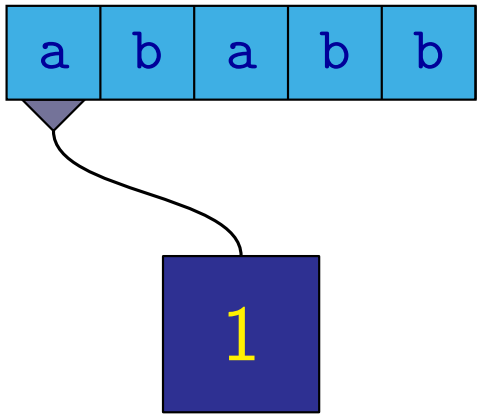
Remark: When a nondeterministic automaton with n states is transformed into a deterministic one, the resulting automaton can have 2^n states.

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.

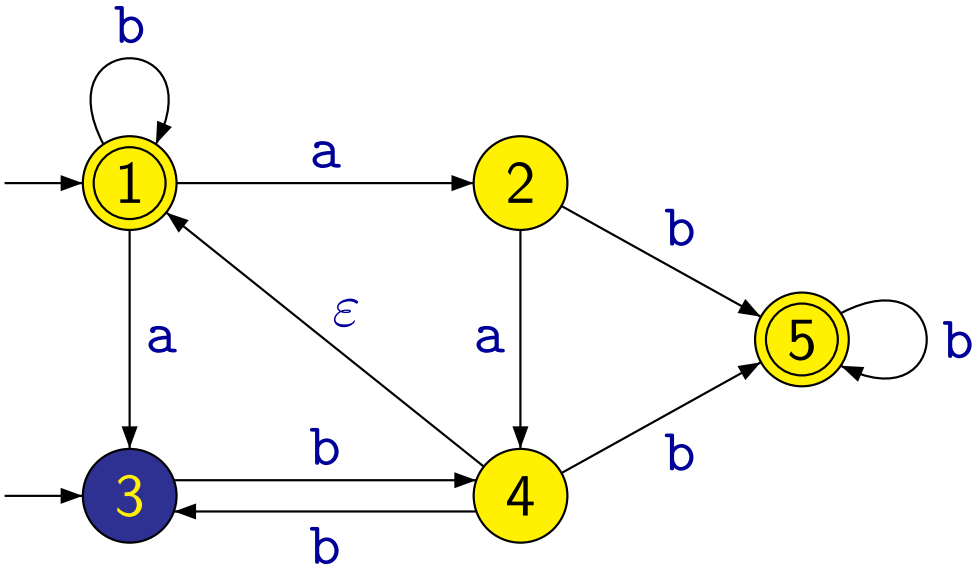
Generalized Nondeterministic Finite Automaton

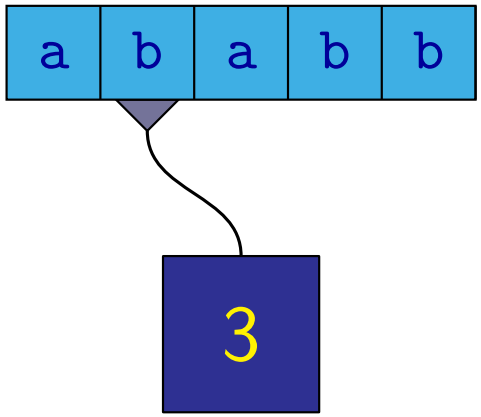




1

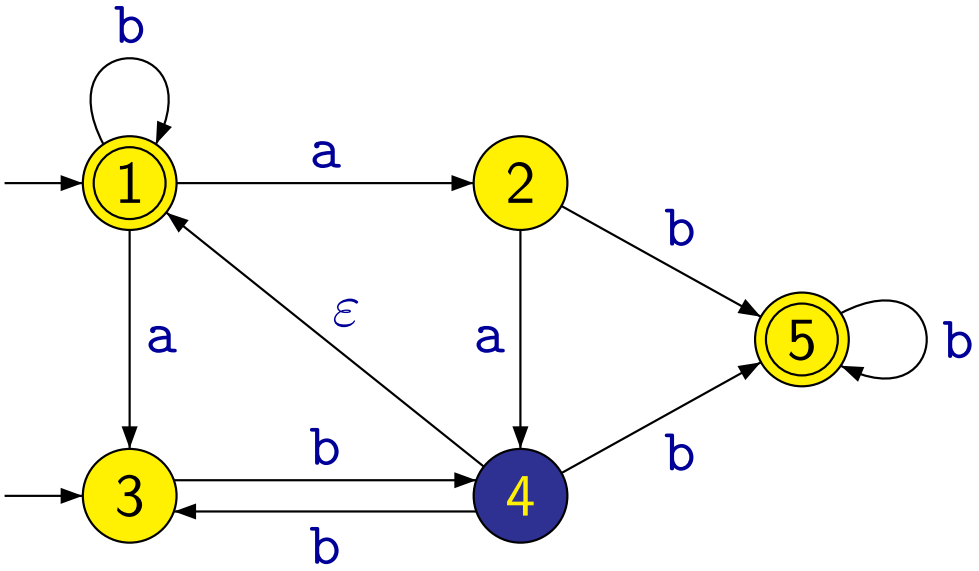
Generalized Nondeterministic Finite Automaton

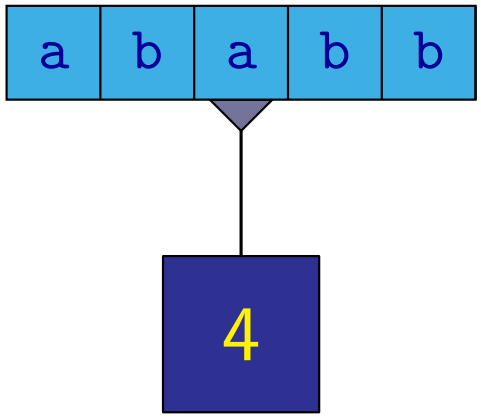




$1 \xrightarrow{a} 3$

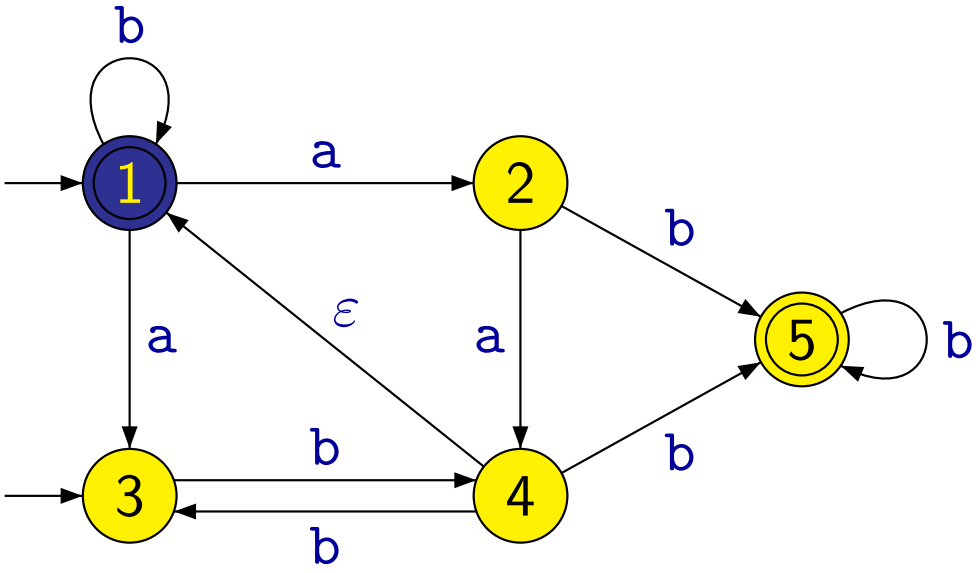
Generalized Nondeterministic Finite Automaton

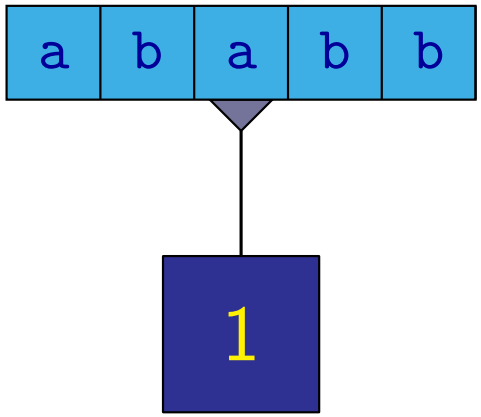




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

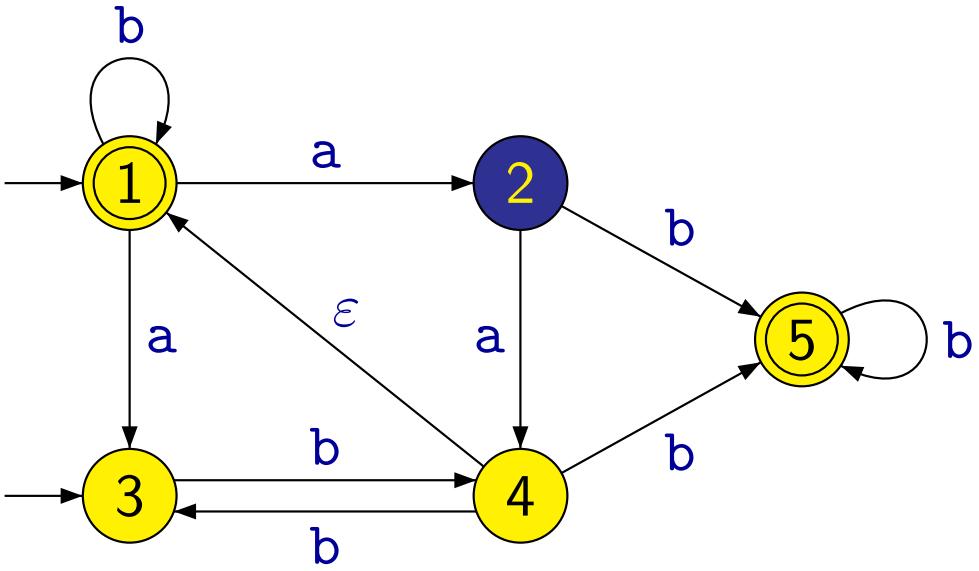
Generalized Nondeterministic Finite Automaton

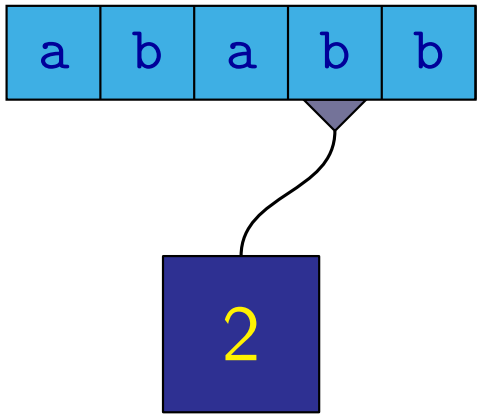




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1$$

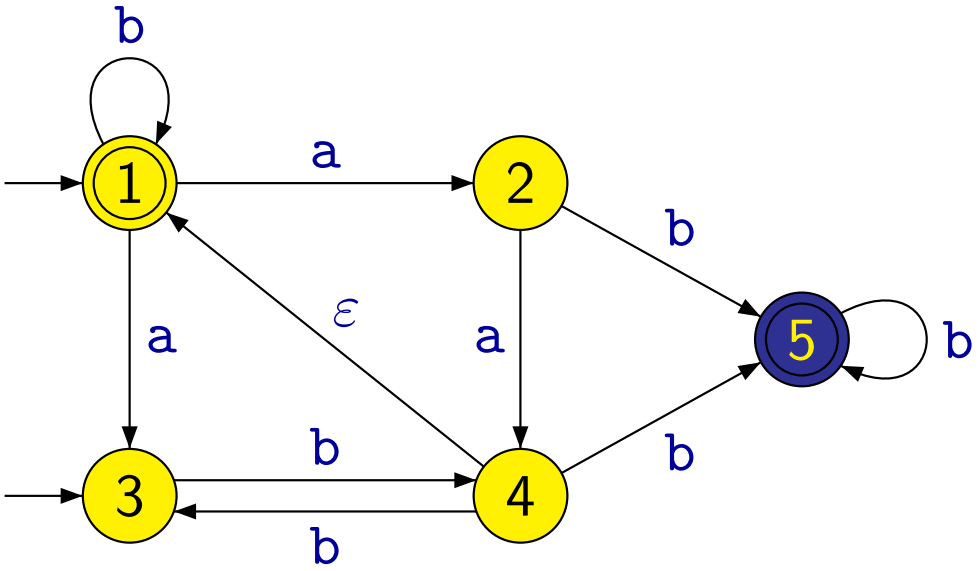
Generalized Nondeterministic Finite Automaton

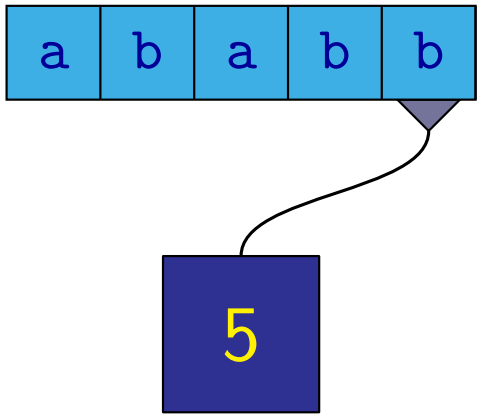




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2$$

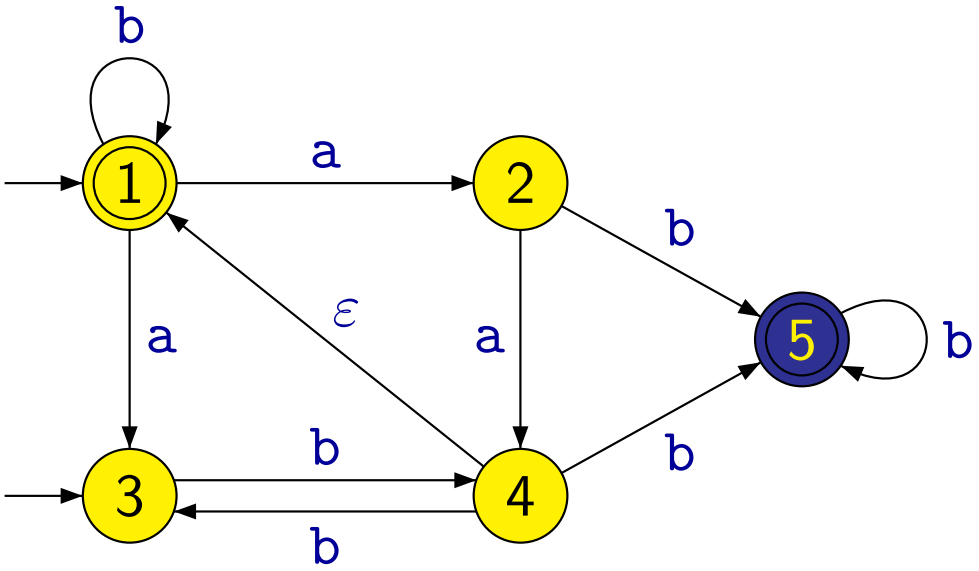
Generalized Nondeterministic Finite Automaton

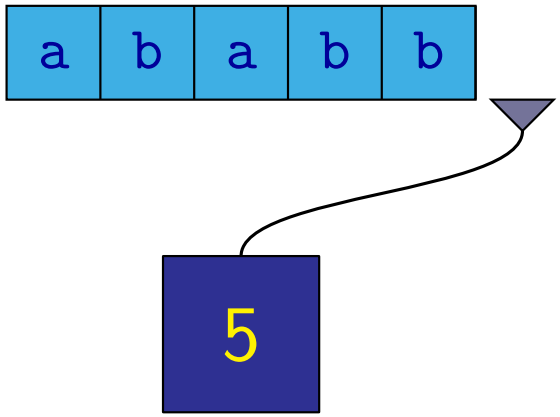




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5$$

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

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

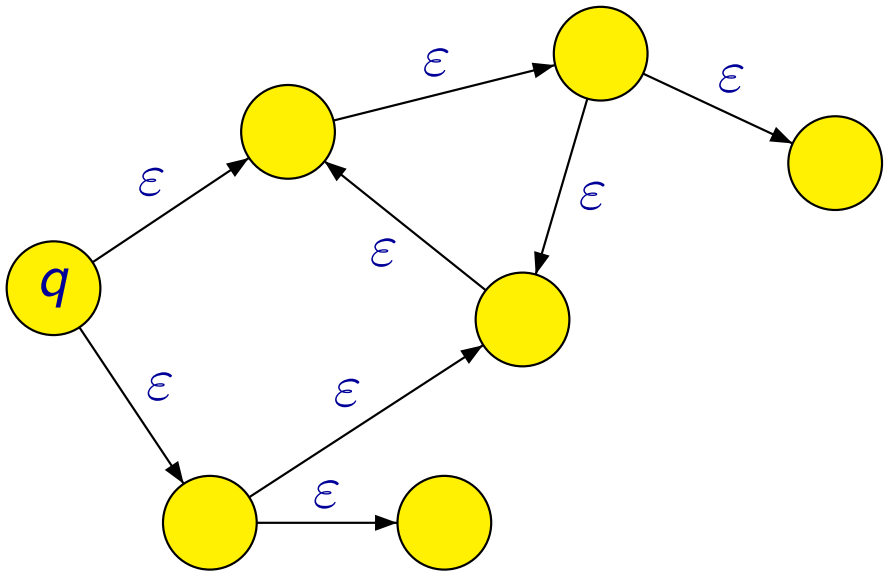
where:

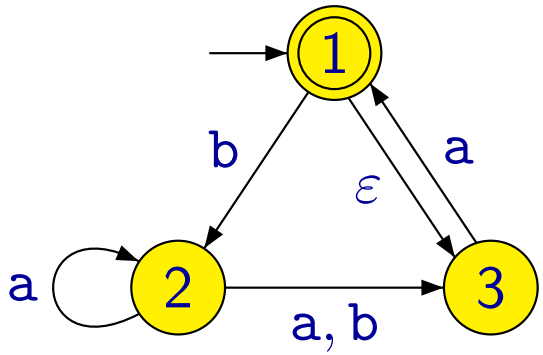
- Q is a finite set of **states**
- Σ is a finite **alphabet**
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ is a **transition function**
- $I \subseteq Q$ is a set of **initial states**
- $F \subseteq Q$ is a set of **accepting states**

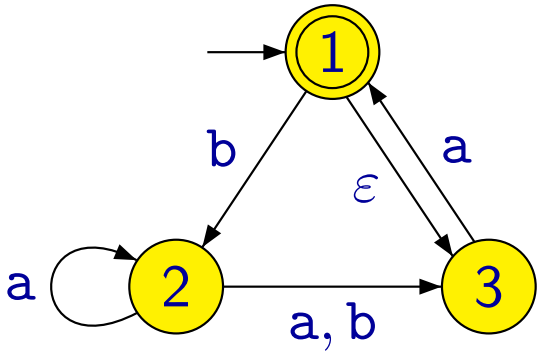
Remark: NFA can be viewed as a special case of GNFA, where $\delta(q, \varepsilon) = \emptyset$ for all $q \in Q$.

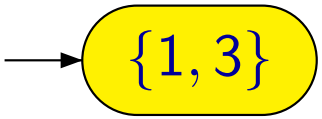
Transformation to a Deterministic Finite Automaton

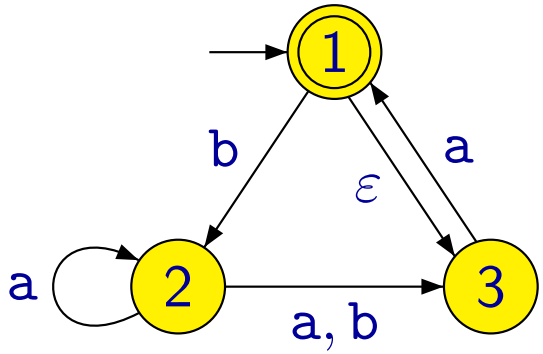
A generalized nondeterministic finite automaton can be transformed into a deterministic one using a similar construction as a nondeterministic finite automaton with the difference that we add to sets of states also all states that are reachable from already added states by some sequence of ε-transitions.

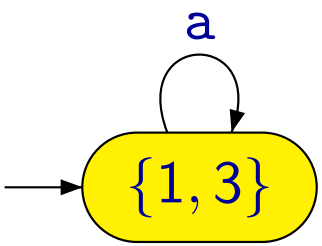


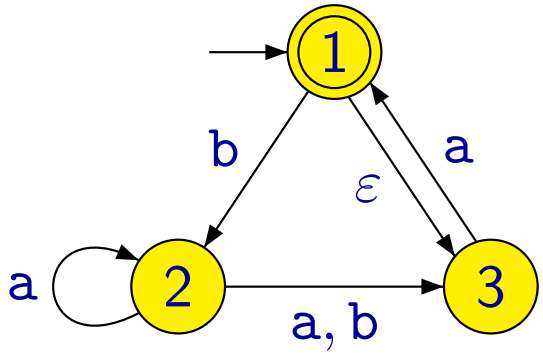

1
2
3
b
a
ε
a
a, b

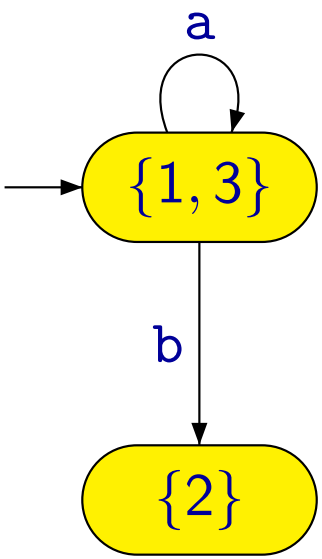

1
b
a
ε
a
2
3
a, b

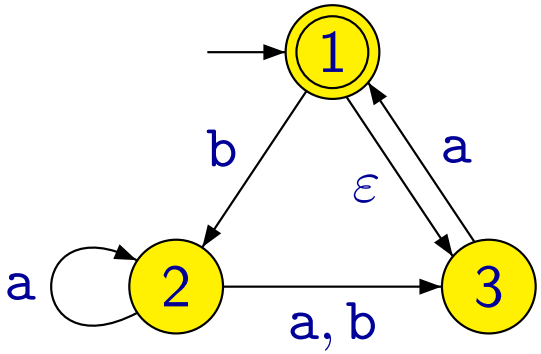

{1, 3}

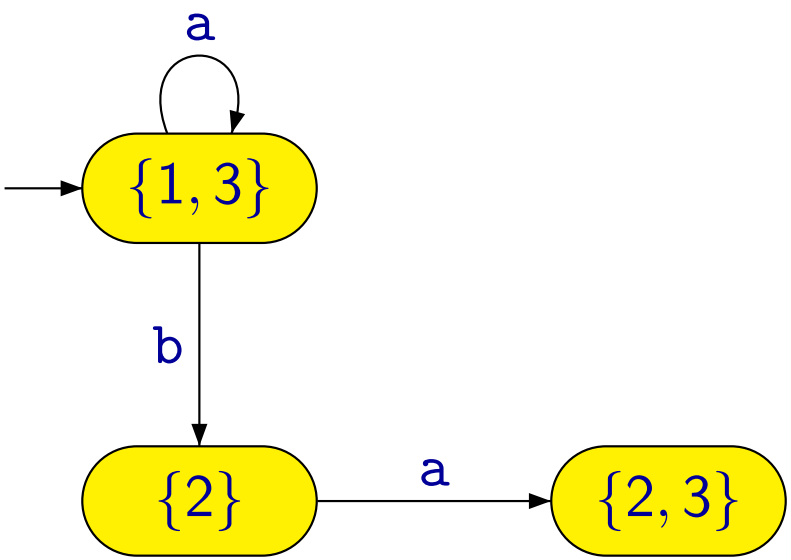

1
b
a
ε
a
2
3
a, b

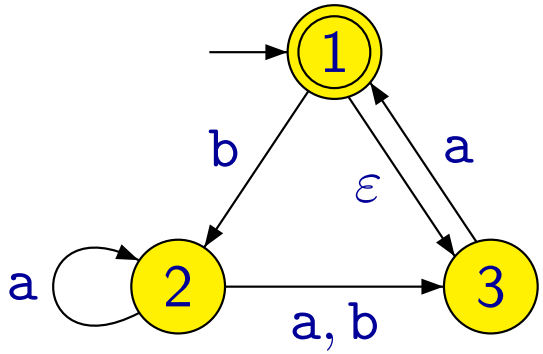

a
{1, 3}

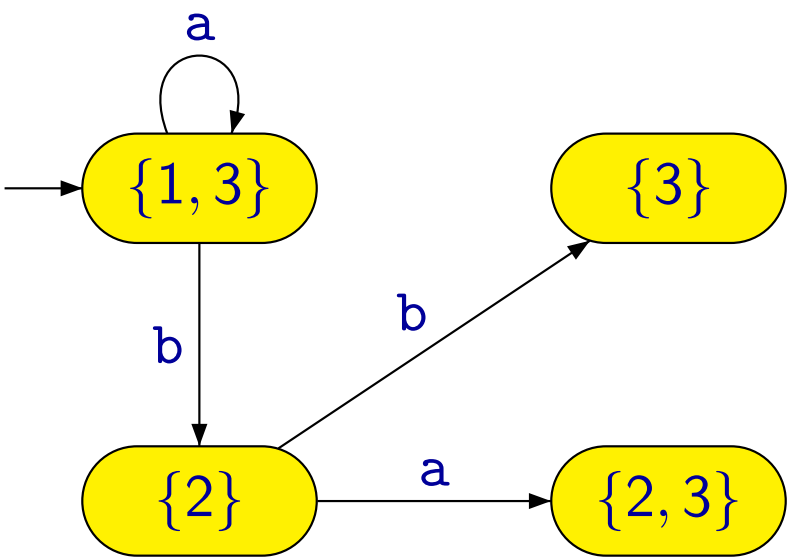

1
b
a
ε
a
2
3
a, b

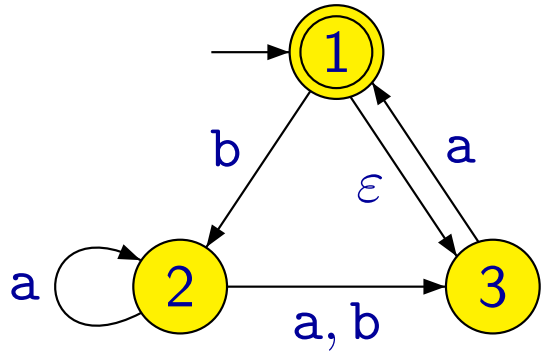

a
{1, 3}
b
{2}

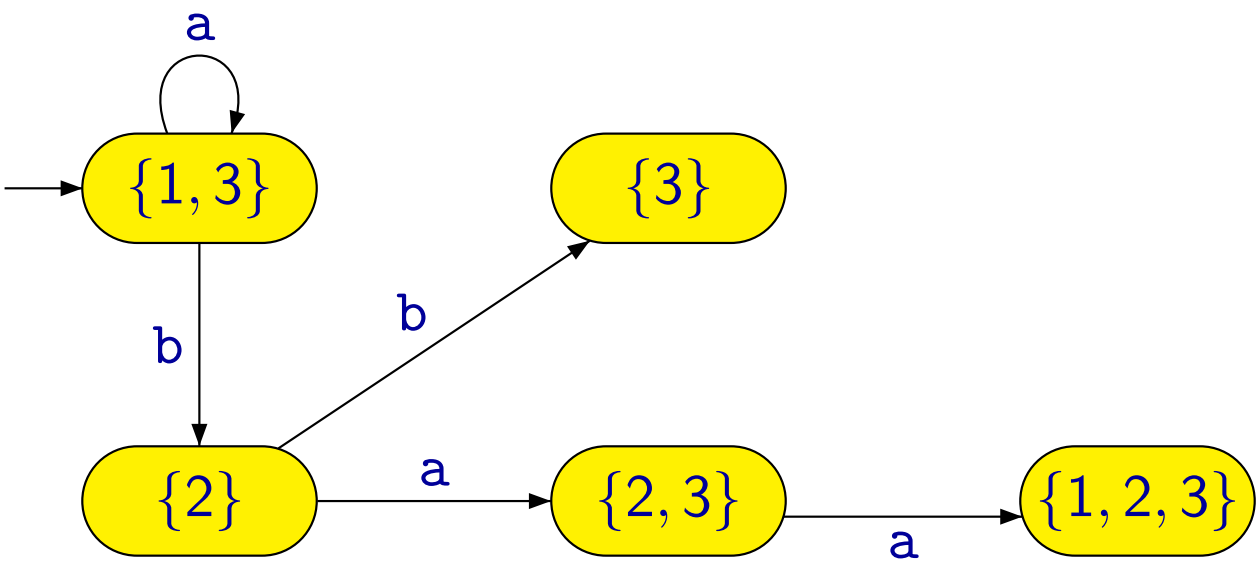

1
b
a
ε
a
2
3
a, b

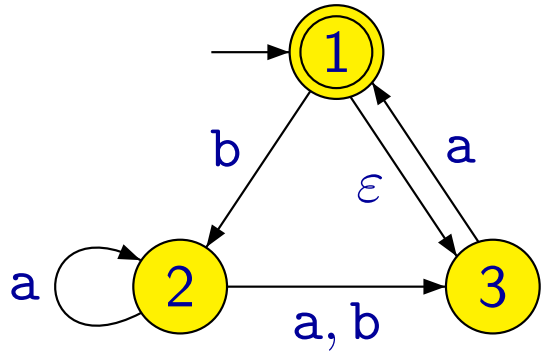

a
{1, 3}
b
{2}
a
{2, 3}

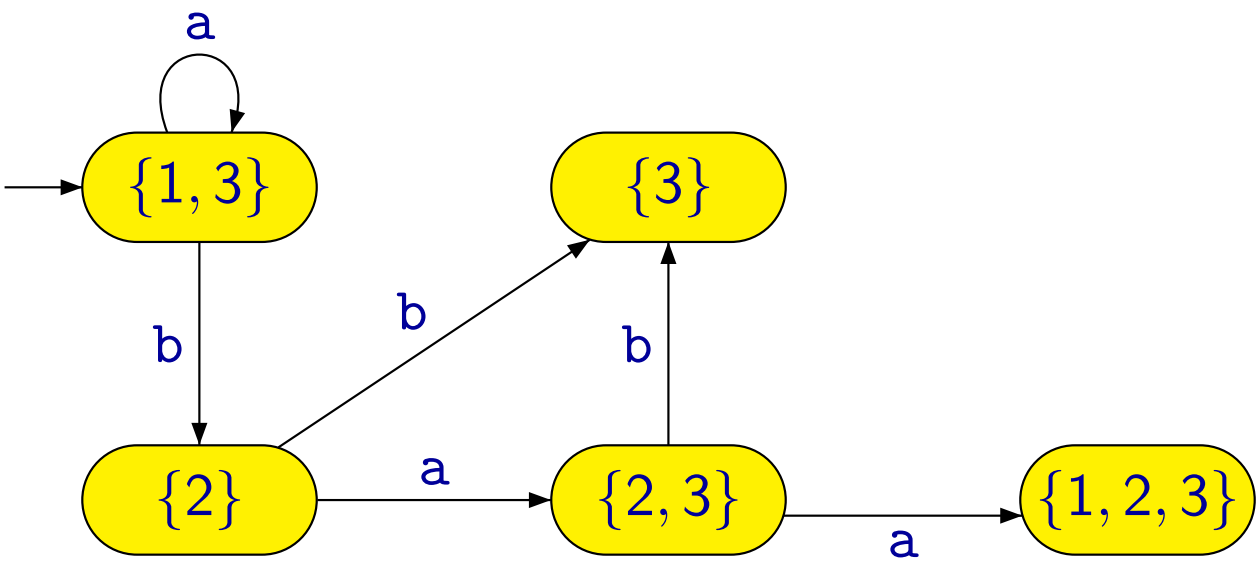

1
b
a
ε
a
2
3
a, b

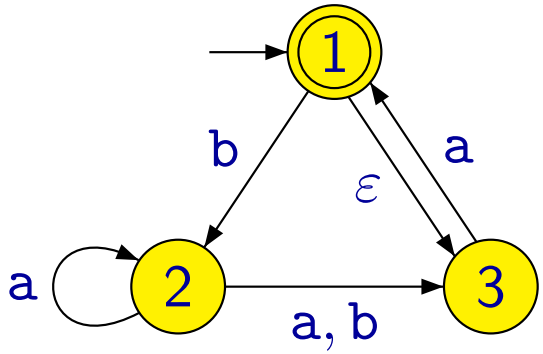

a
{1, 3}
{3}
b
b
{2}
a
{2, 3}

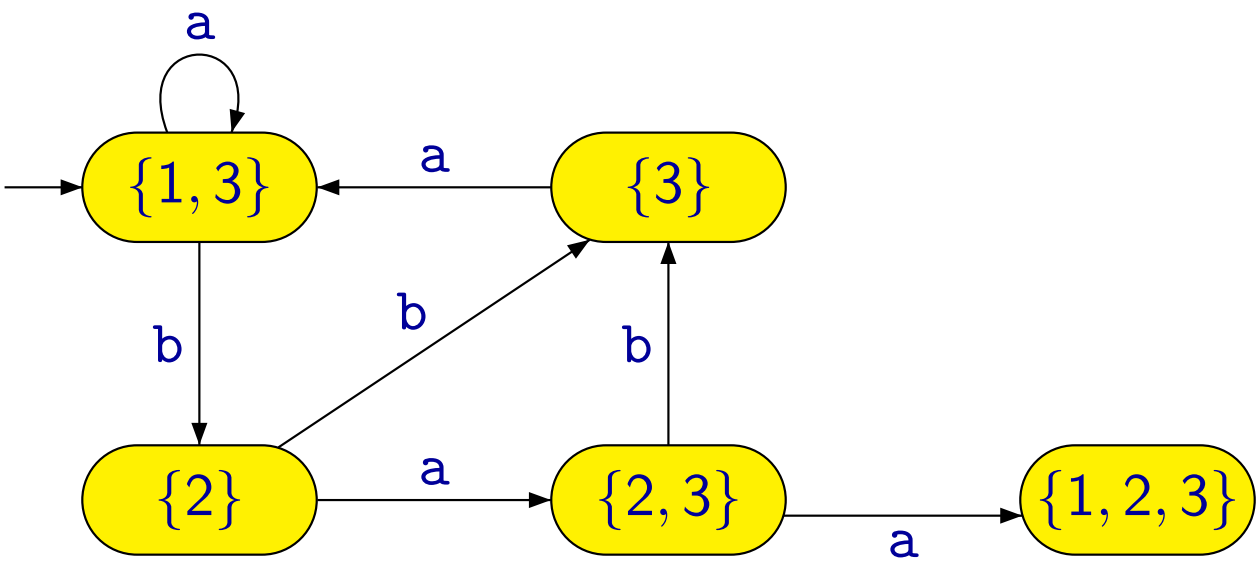

1
b
a
ε
a
2
3
a, b

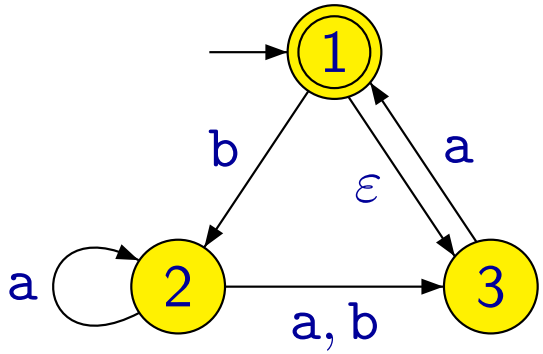

a
{1, 3}
{3}
b
b
{2}
a
{2, 3}
{1, 2, 3}
a

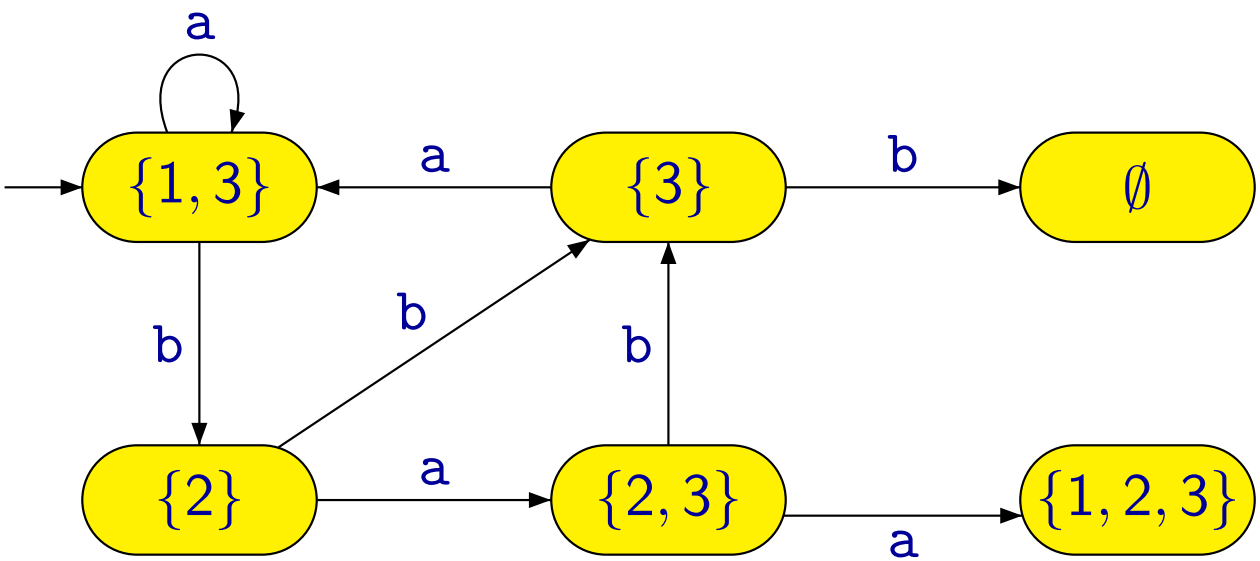

1
b
a
ε
a
2
3
a, b

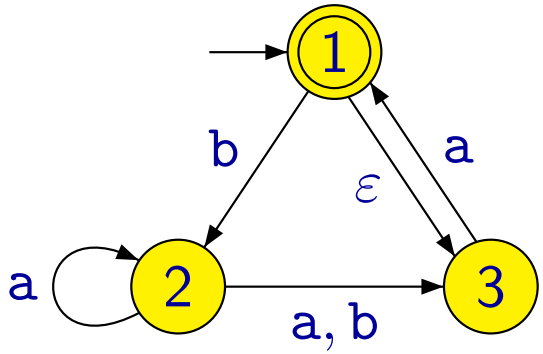

a
{1, 3}
{3}
b
b
b
{2}
a
{2, 3}
{1, 2, 3}
a

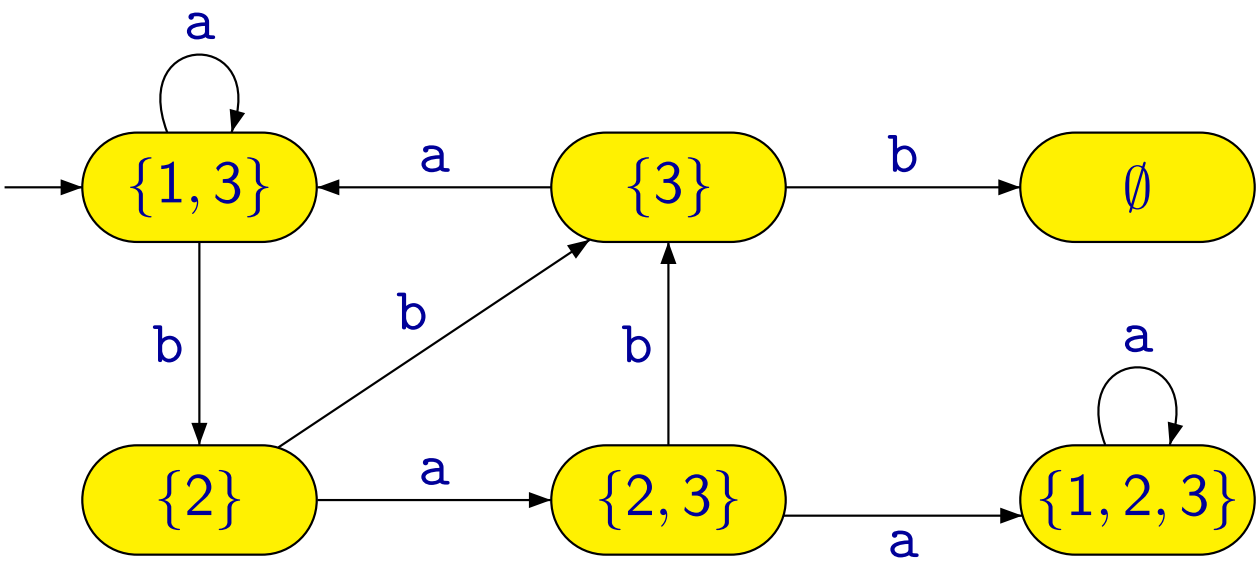

1
b
a
ε
a
2
3
a, b

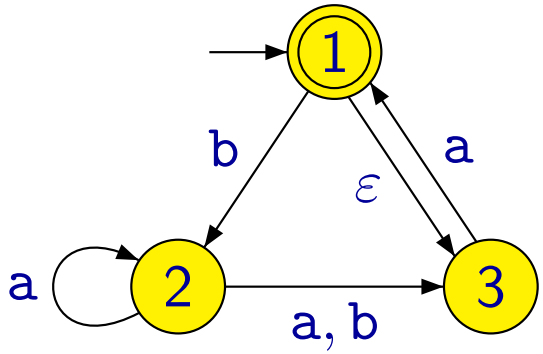

a
{1, 3}
a
{3}
b
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

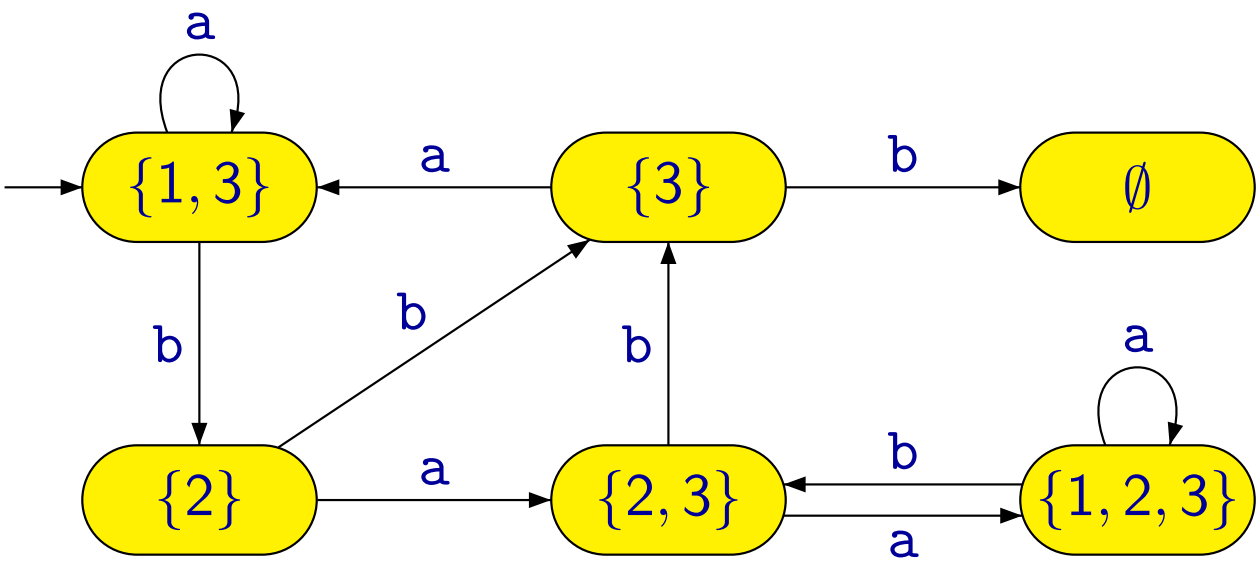

1
b
a
ε
a
2
3
a, b

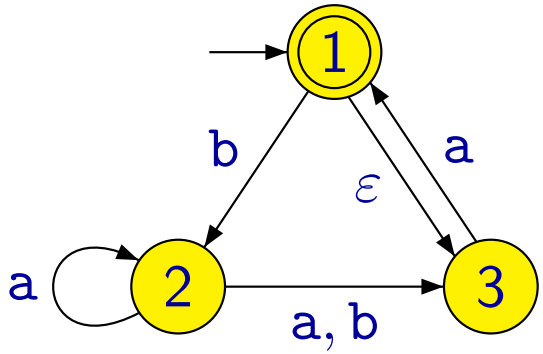

a
{1, 3}
a
{3}
b
∅
b
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

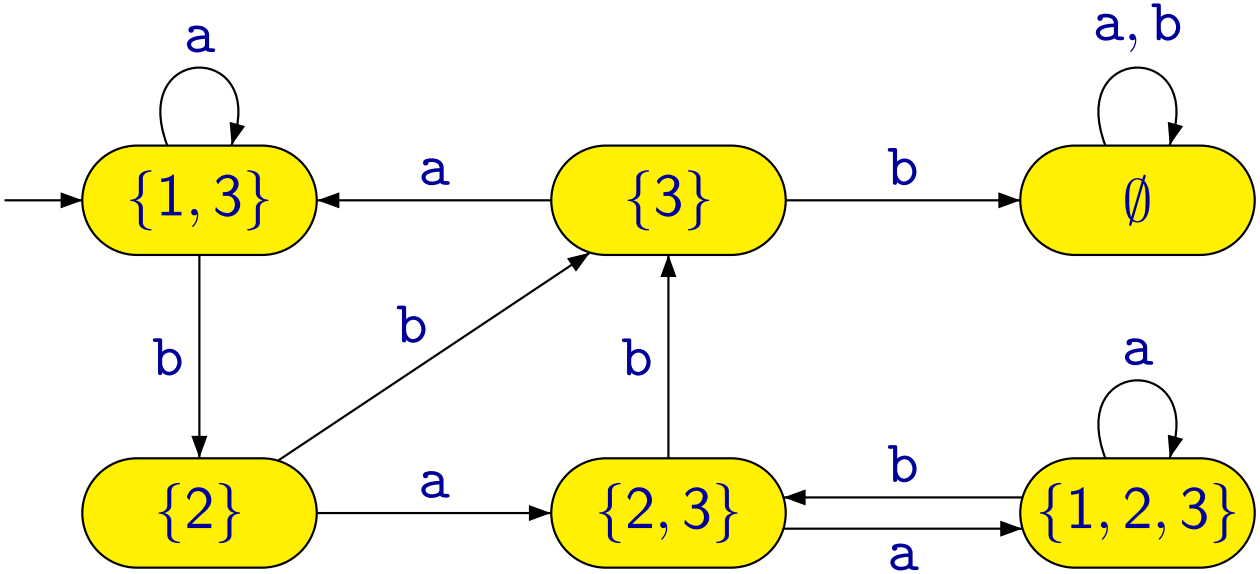

1
b
a
ε
a
2
3
a, b

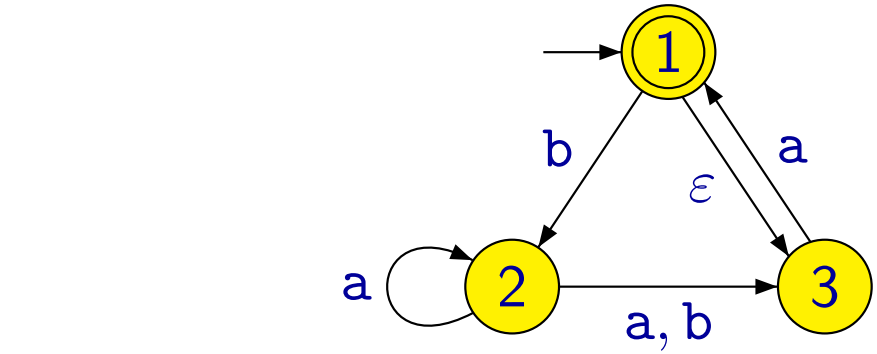

a
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
{1, 2, 3}
a

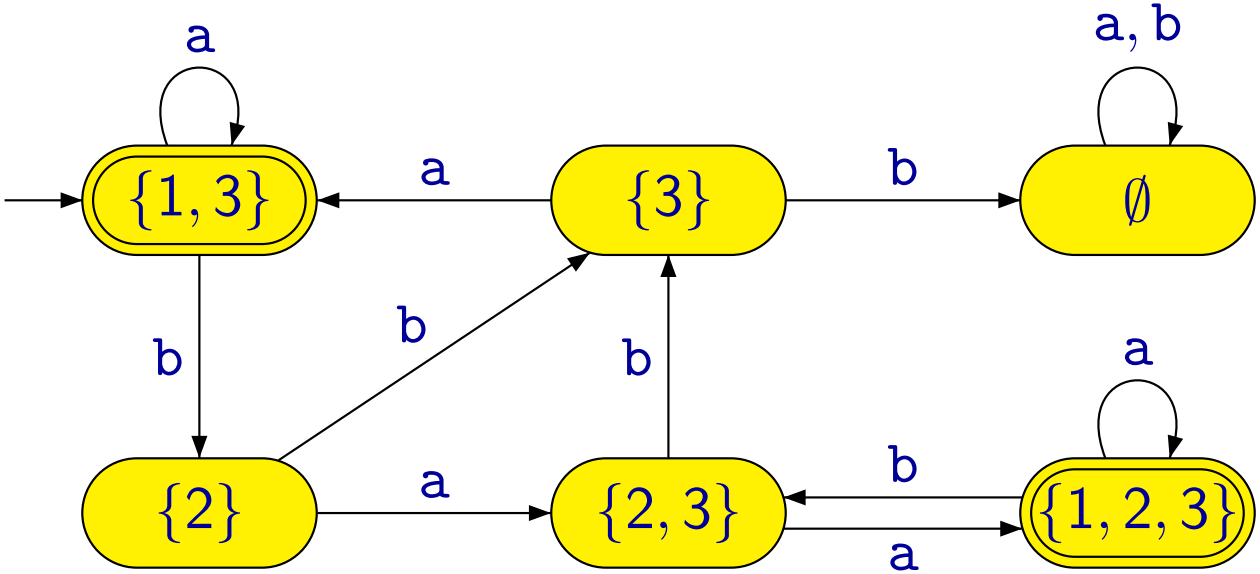

1
b
a
ε
a
2
3
a, b


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


1
b
a
ε
a
2
3
a, b


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


1
b
a
ε
a
2
3
a, b


a
a, b
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
b
{1, 2, 3}
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 $A = (Q, \Sigma, \delta, I, F)$.

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 the all 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 $A = (Q, \Sigma, \delta, I, F)$ we can now construct DFA $A' = (Q', \Sigma, \delta', q_0', F')$, 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 $a \in \Sigma$:

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

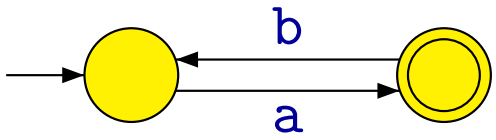
- $q_0' = Cl_\varepsilon(I)$
- $F' = \{K \in Q' \mid Cl_\varepsilon(K) \cap F \neq \emptyset\}$

It is not difficult to verify that $L(A) = L(A')$.

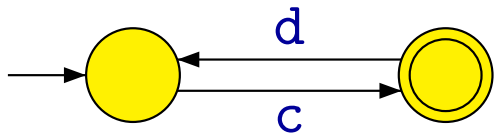
Concatenation of Languages

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

A_1:



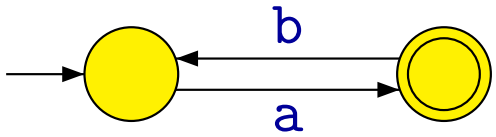
A_2:



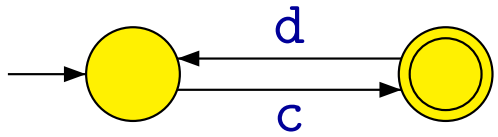
Concatenation of Languages

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

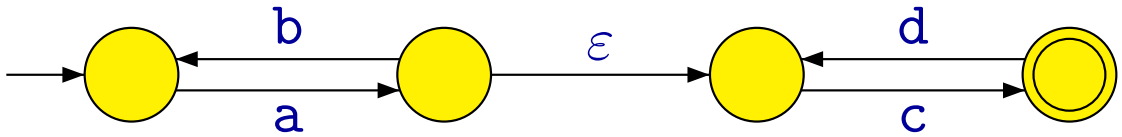
A_1:



A_2:



A:

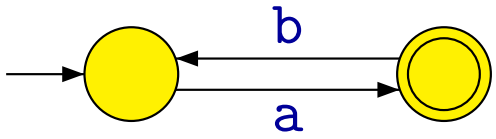


$L(A) = L(A_1) \cdot L(A_2)$

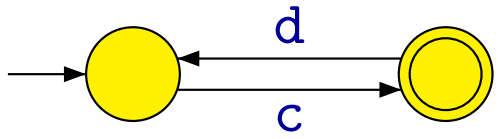
Concatenation of Languages

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

A_1:

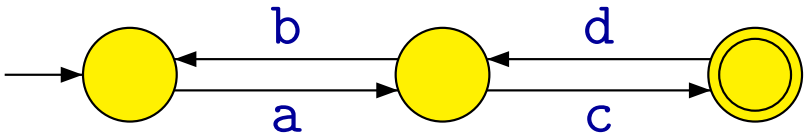


A_2:



An incorrect construction:

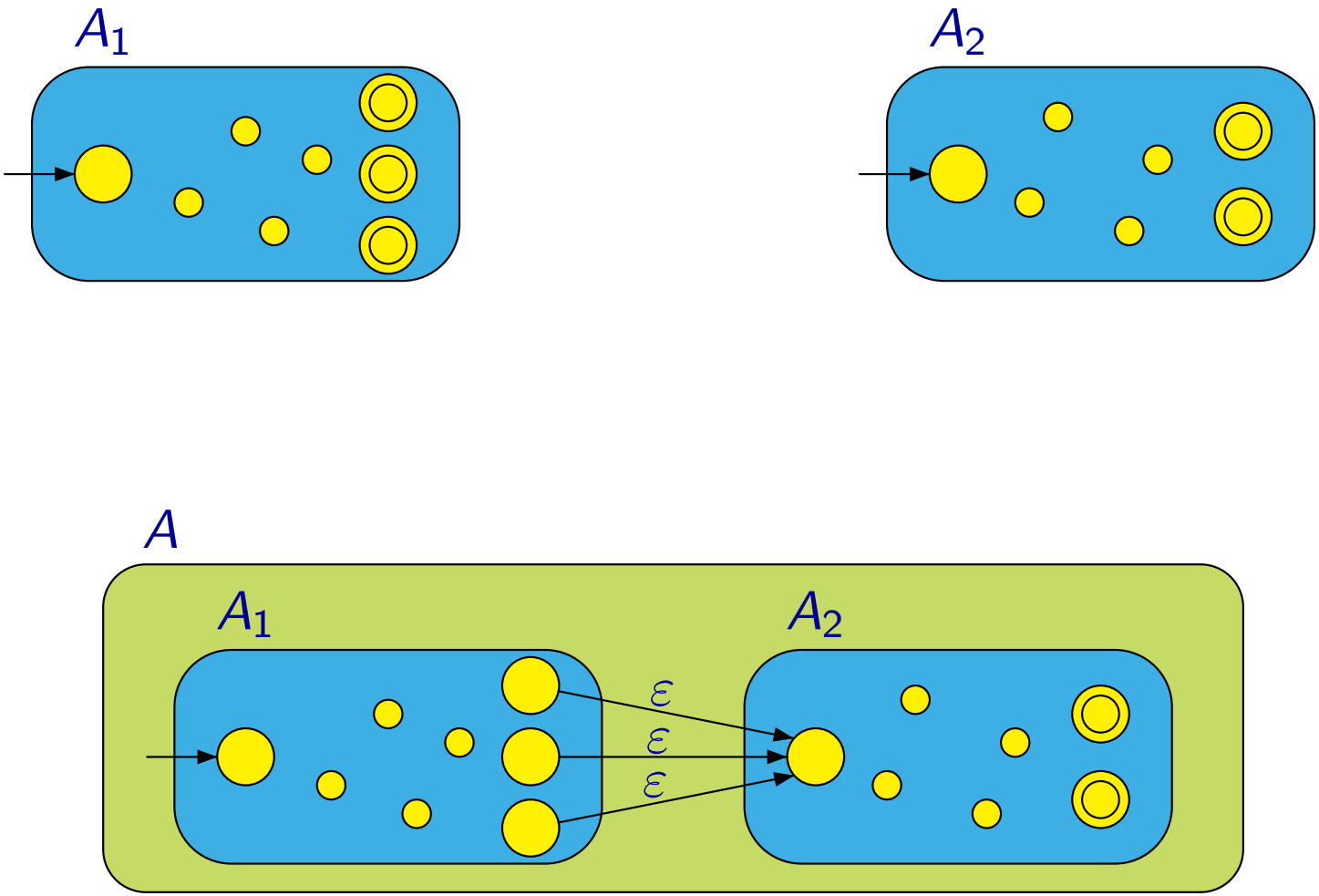
A:



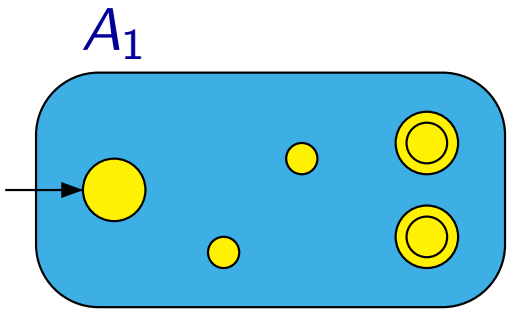
$\mathtt{acdbac} \in L(A)$ but $\mathtt{acdbac} \notin L(A_1) \cdot L(A_2)$

Concatenation of Languages

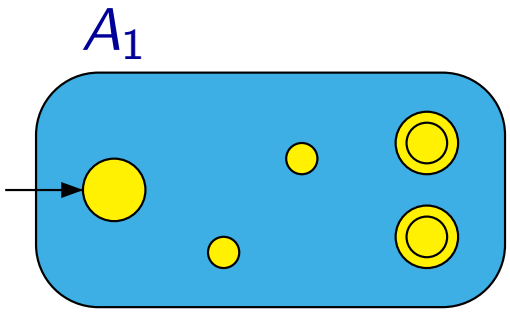
Concatenation of Languages

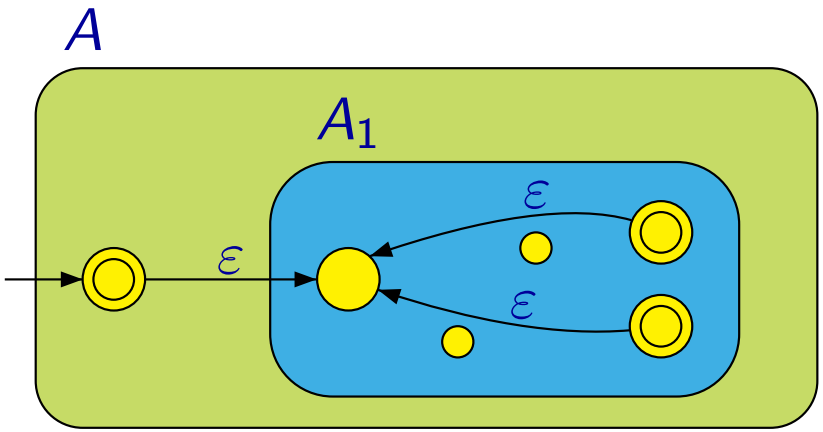


Iteration of a Language



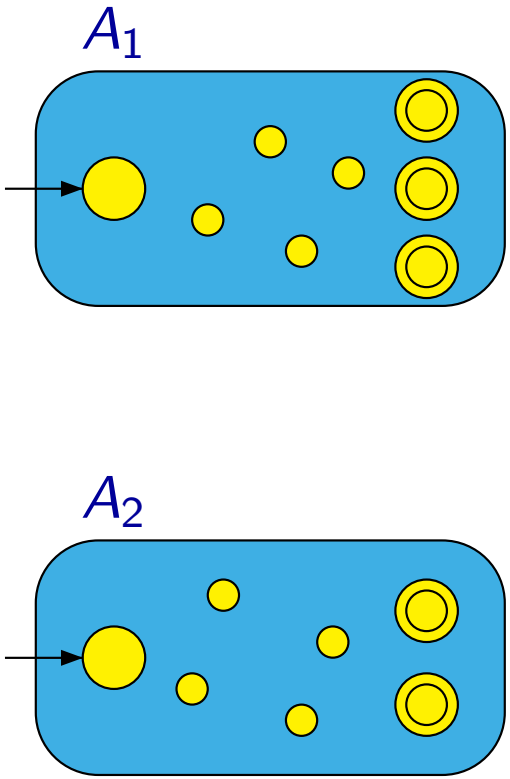
Iteration of a Language





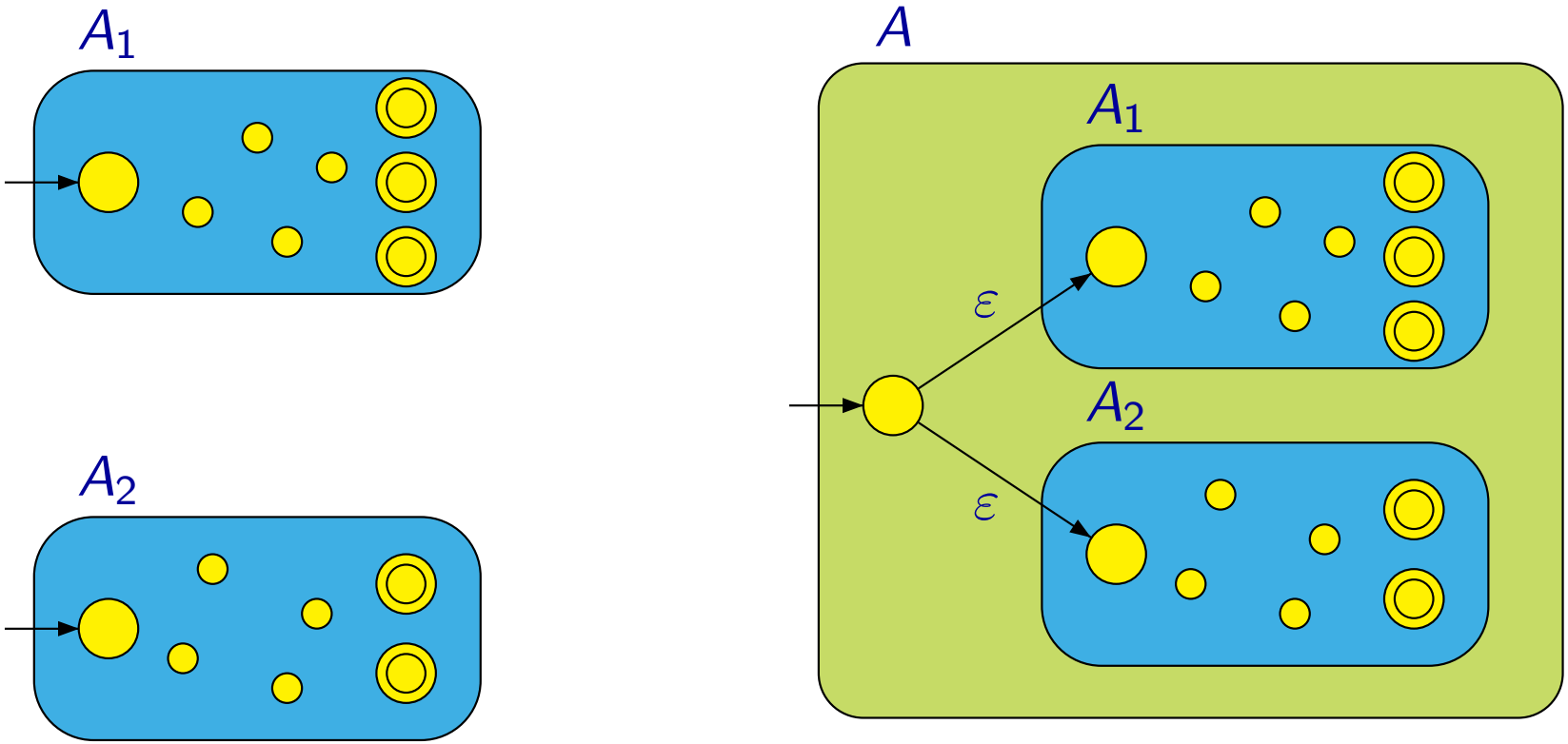
Union of Languages

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).

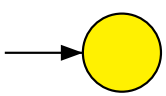
Proof: It is sufficient to show how to construct for a given regular expression α a finite automaton accepting the language $[\alpha]$.

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

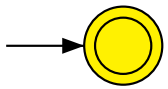
- If α is a elementary expression (i.e., $\emptyset$, ε or a):
  - We construct the corresponding automaton directly.
- If α is of the form $(\beta + \gamma)$, $(\beta \cdot \gamma)$ or (β^*):
  - We construct automata accepting languages $[\beta]$ and $[\gamma]$ recursively.
  - Using these two automata,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

Transformation of a Regular Expression to a Finite Automaton

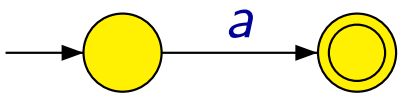
The automata for the elementary expressions:



$\emptyset$



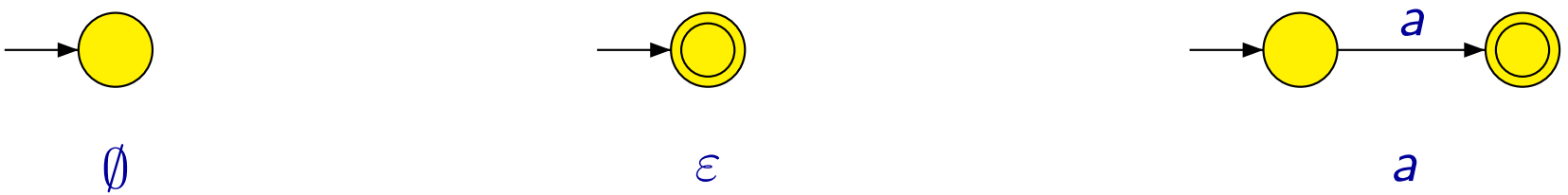
ε



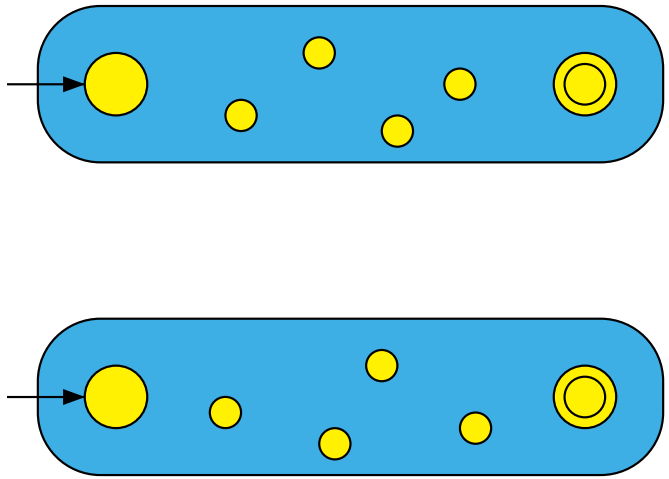
a

Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

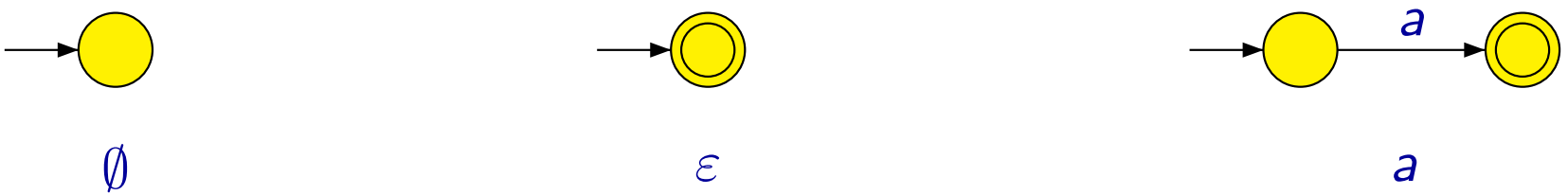


The construction for the union:

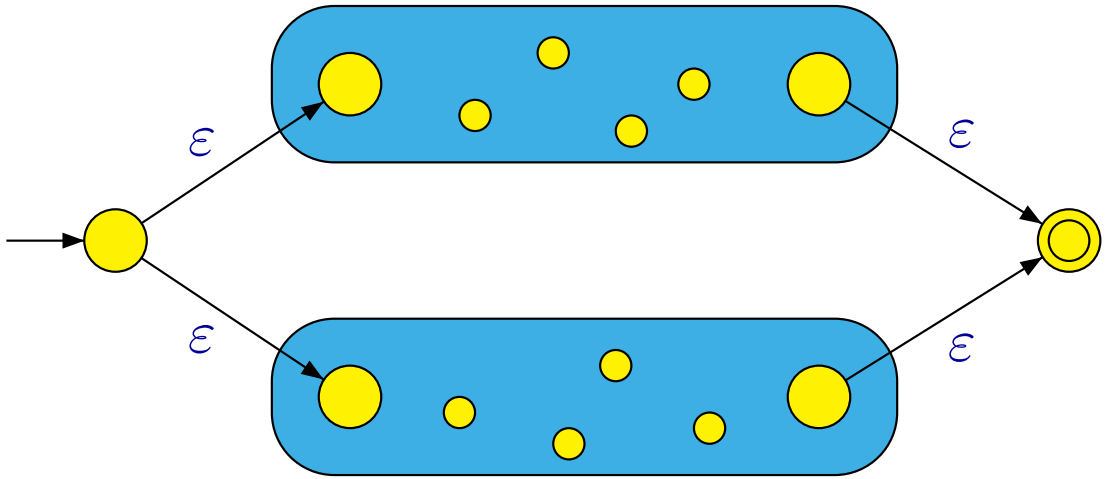


Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

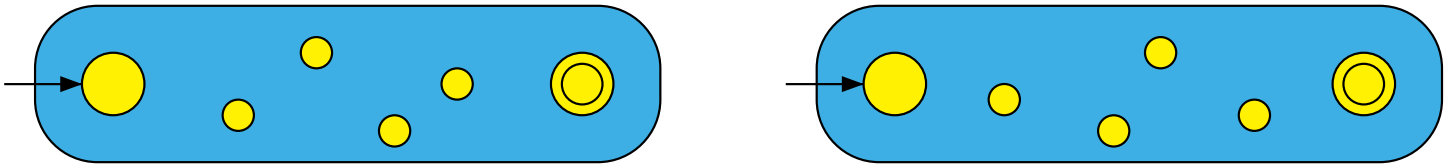


The construction for the union:



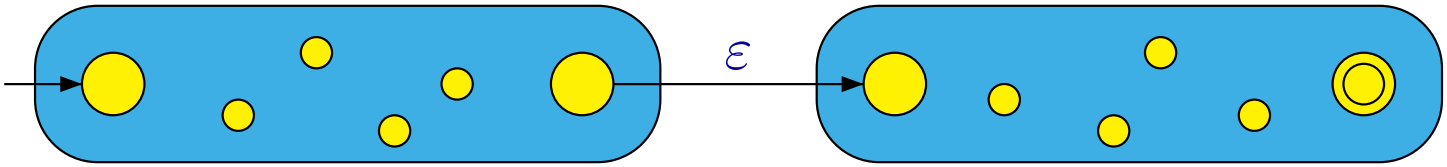
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:



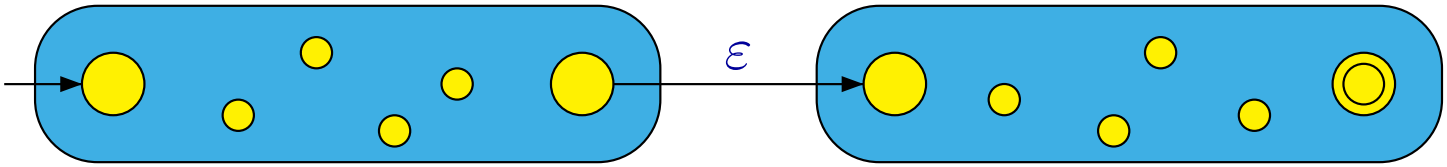
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

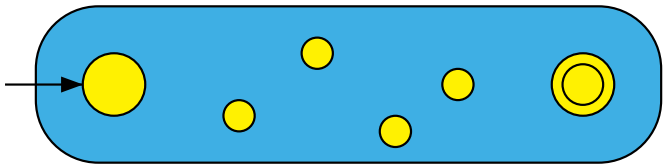


Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

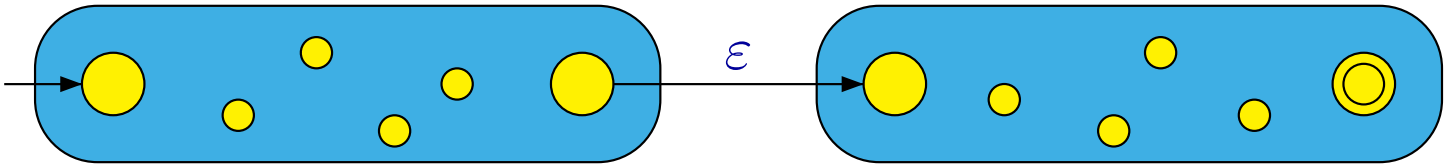


The construction for the iteration:

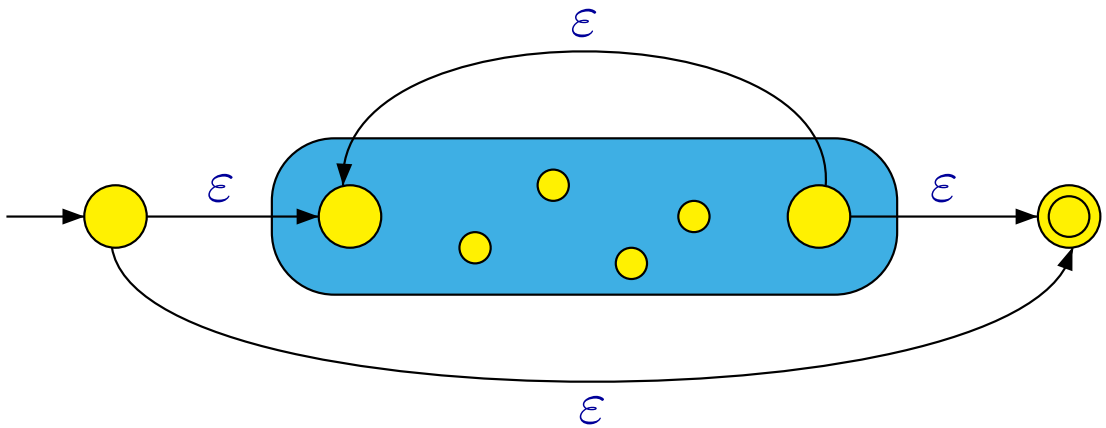


Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:



The construction for the iteration:

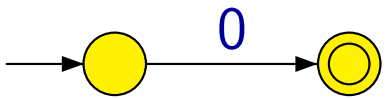


Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:

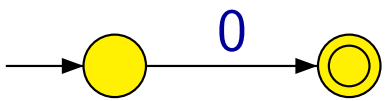
Transformation of a Regular Expression to a Finite Automaton

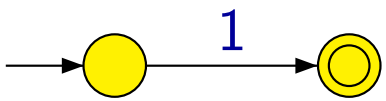
Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:



Transformation of a Regular Expression to a Finite Automaton

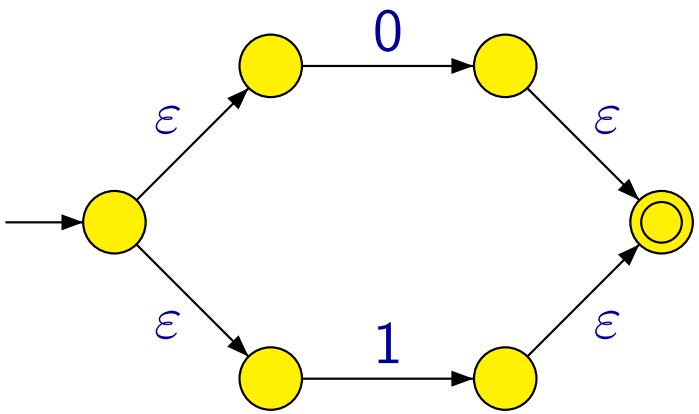
Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:





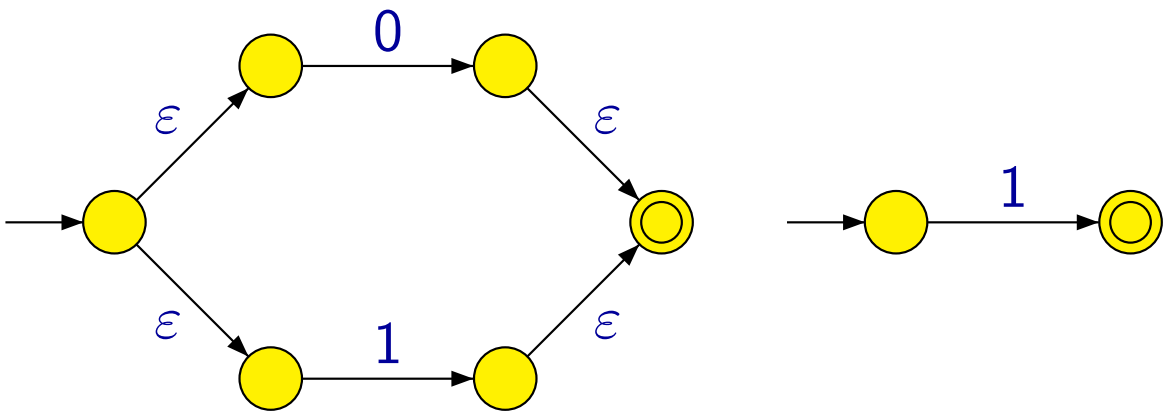
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:



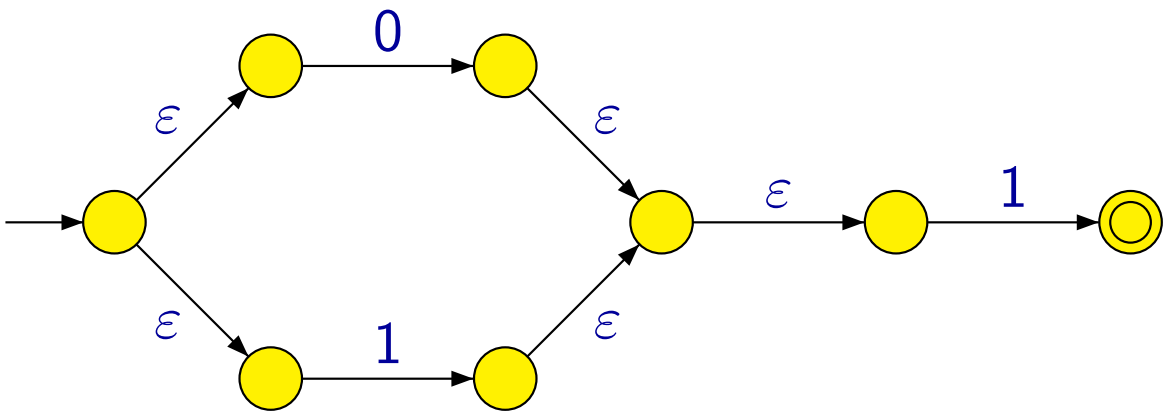
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:



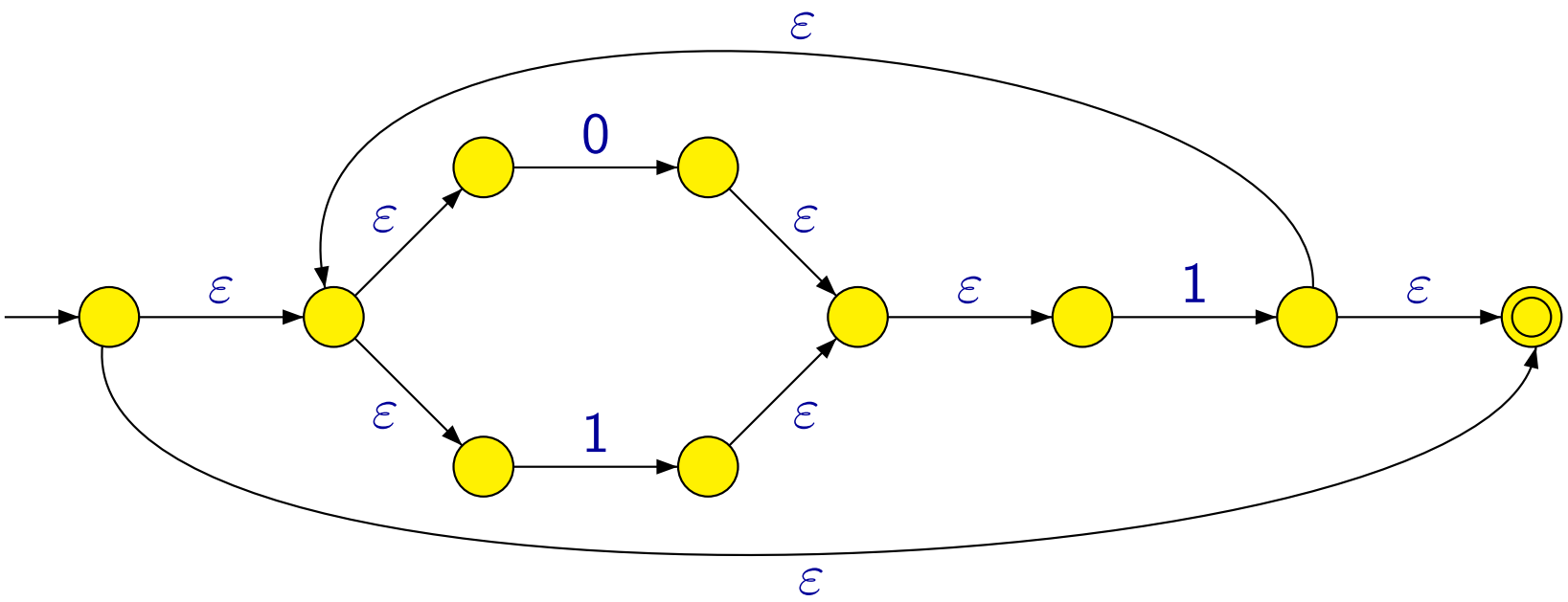
Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:



Transformation of a Regular Expression to a Finite Automaton

Example: The construction of an automaton for expression $((0+1)\cdot 1)^*$:



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

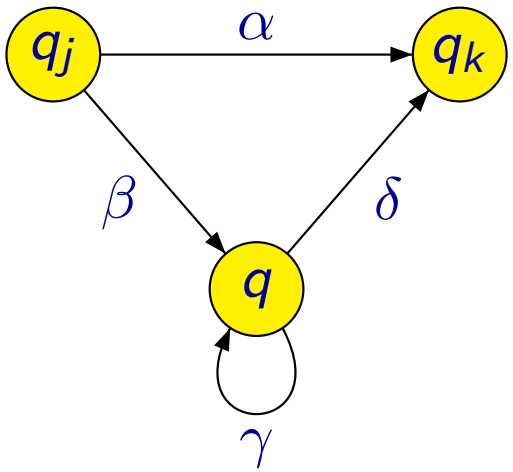
Every regular language can be represented by some regular expression.

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

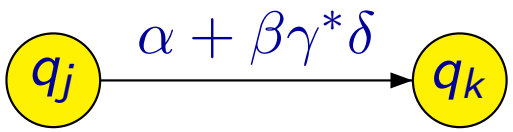
- We modify A in such a way that ensures it has exactly one initial and exactly one accepting state.
- Its states will be removed one by one.
- Its transitions will be labelled with regular expressions.
- The resulting automaton will have only two states – the initial and the accepting, and only one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

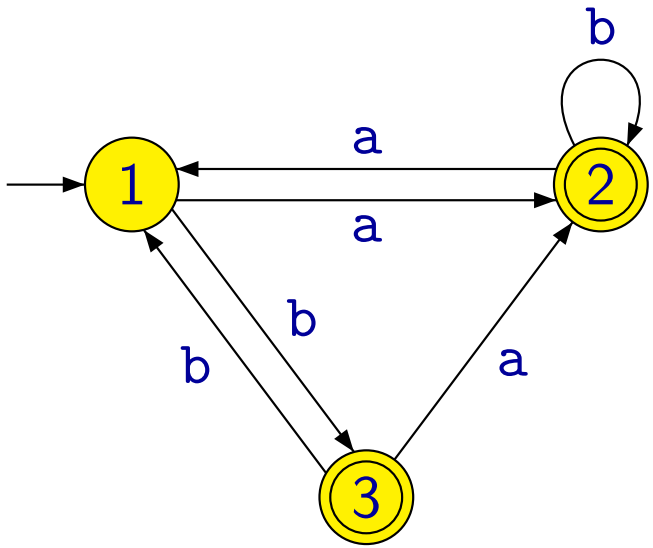


After removing of the state q:



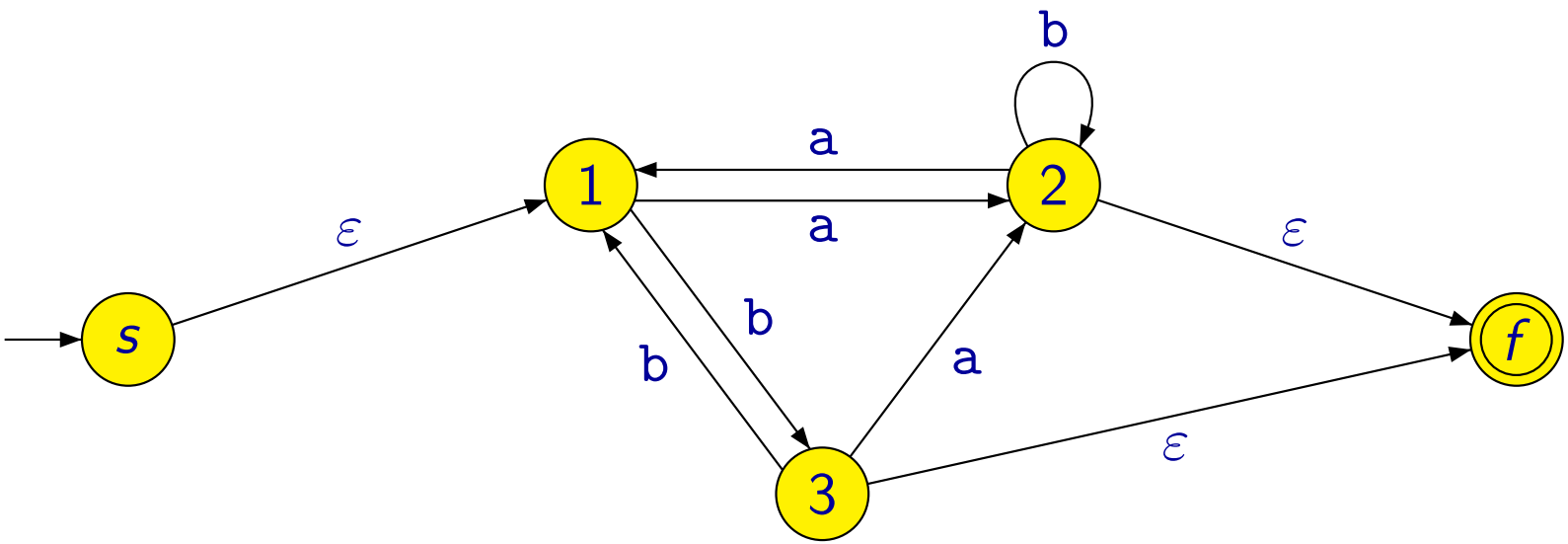
Transformation of an Automaton to a Regular Expression

Example:



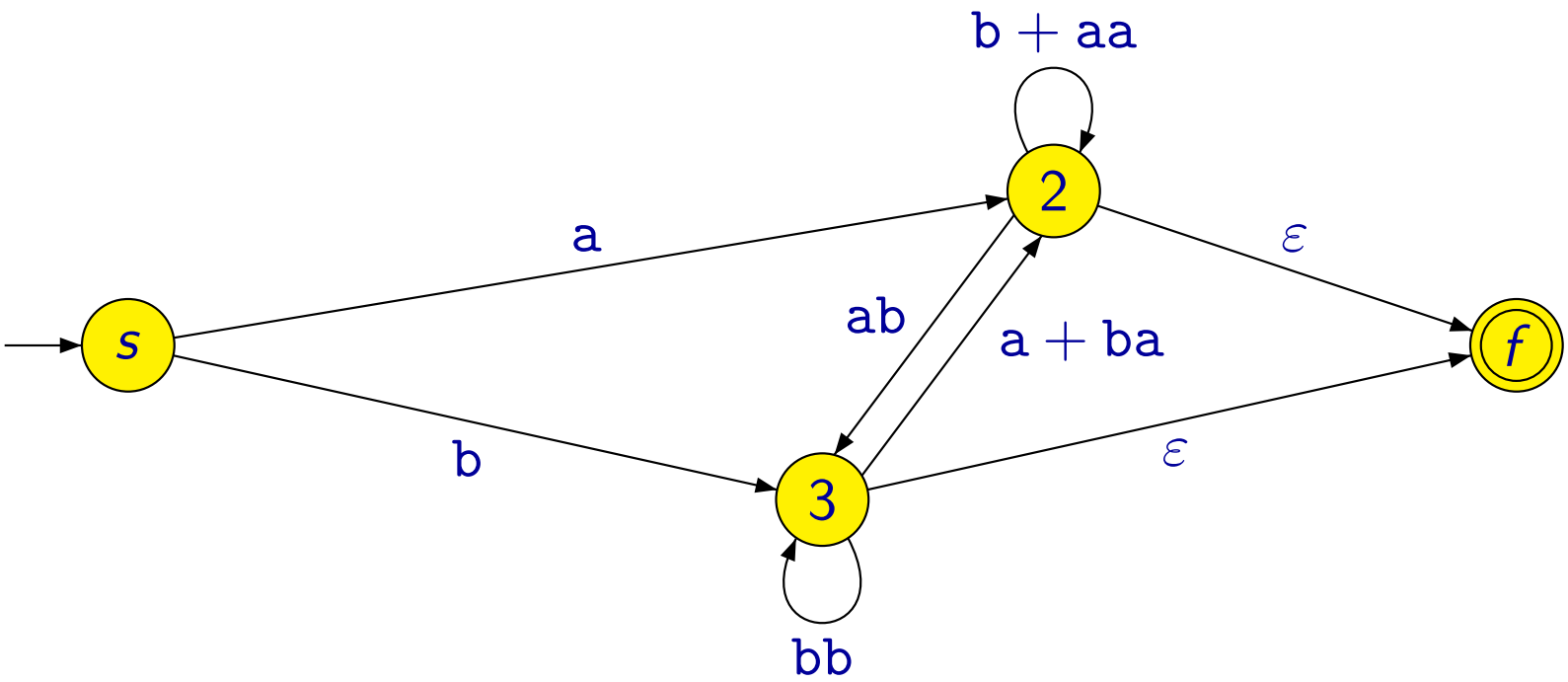
Transformation of an Automaton to a Regular Expression

Example:



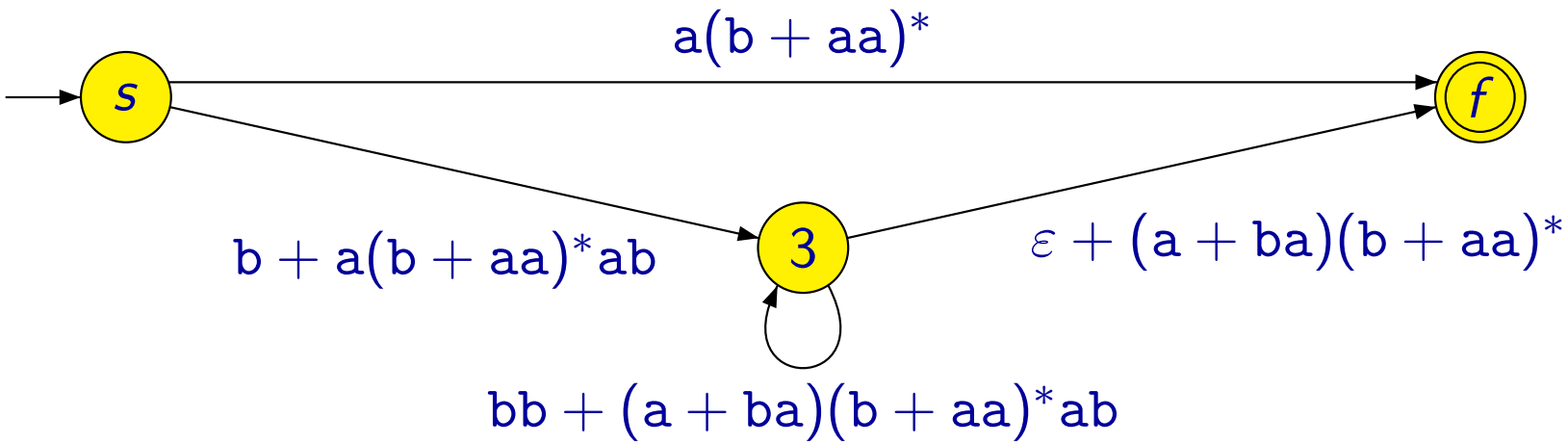
Transformation of an Automaton to a Regular Expression

Example:



Transformation of an Automaton to a Regular Expression

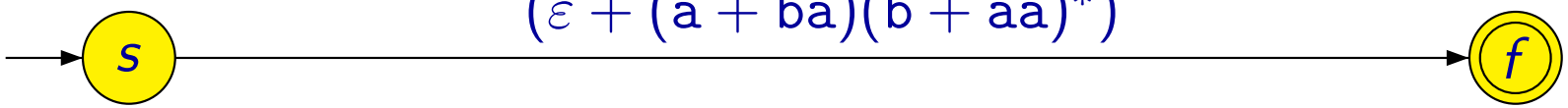
Example:



Transformation of an Automaton to a Regular Expression

Example:

$$\begin{gathered} \mathtt{a}(\mathtt{b}+\mathtt{aa})^*+ \\ (\mathtt{b}+\mathtt{a}(\mathtt{b}+\mathtt{aa})^*\mathtt{ab}) \\ (\mathtt{bb}+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})^* \\ (\varepsilon+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*) \end{gathered}$$



Equivalence of Finite Automata and Regular Expressions

Theorem

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

Context-Free Grammars

Context-Free Grammars

Example:

$$
\begin{aligned}
\langle \text{STMT} \rangle &\to \langle \text{IF-STMT} \rangle \mid \langle \text{WHILE-STMT} \rangle \mid \langle \text{BLOCK-STMT} \rangle \mid \\
&\quad \langle \text{ASSG-STMT} \rangle \\
\langle \text{IF-STMT} \rangle &\to \textbf{if } \langle \text{BOOL-EXPR} \rangle \textbf{ then } \langle \text{STMT} \rangle \textbf{ else } \langle \text{STMT} \rangle \\
\langle \text{WHILE-STMT} \rangle &\to \textbf{while } \langle \text{BOOL-EXPR} \rangle \textbf{ do } \langle \text{STMT} \rangle \\
\langle \text{BLOCK-STMT} \rangle &\to \textbf{begin } \langle \text{STMT-LIST} \rangle \textbf{ end} \\
\langle \text{STMT-LIST} \rangle &\to \langle \text{STMT} \rangle \mid \langle \text{STMT} \rangle \, ; \, \langle \text{STMT-LIST} \rangle \\
\langle \text{ASSG-STMT} \rangle &\to \langle \text{VAR} \rangle := \langle \text{ARITH-EXPR} \rangle \\
\langle \text{BOOL-EXPR} \rangle &\to \langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \\
\langle \text{COMPARE-OP} \rangle &\to < \mid > \mid \leq \mid \geq \mid = \mid \neq \\
\langle \text{ARITH-EXPR} \rangle &\to \langle \text{VAR} \rangle \mid \langle \text{CONST} \rangle \mid \\
&\quad (\langle \text{ARITH-EXPR} \rangle \langle \text{ARITH-OP} \rangle \langle \text{ARITH-EXPR} \rangle) \\
\langle \text{ARITH-OP} \rangle &\to + \mid - \mid * \mid / \\
\langle \text{CONST} \rangle &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
\langle \text{VAR} \rangle &\to a \mid b \mid c \mid \cdots \mid x \mid y \mid z
\end{aligned}
$$

Context-Free Grammars

The symbols, that have the form $\langle \text{XXX} \rangle$ in the previous example, are called **nonterminal symbols** (**nonterminals**).

The rules describe the strings represented by a given nonterminal.

From nonterminal $\langle \text{STMT} \rangle$ we can for example obtain the text

$$\textbf{while } x \leq y \textbf{ do begin } x := (x + 1)\,;\; y := (y - 1) \textbf{ end}$$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

Context-Free Grammars

$$\textbf{while } x \leq y \textbf{ do begin } x := (x+1)\,;\; y := (y-1) \textbf{ end}$$

$\langle \text{STMT} \rangle$

Context-Free Grammars

$$\textbf{while } x \leq y \textbf{ do begin } x := (x+1)\,;\; y := (y-1) \textbf{ end}$$

$\langle \text{STMT} \rangle$
$\langle \text{WHILE-STMT} \rangle$

Context-Free Grammars

$$\textbf{while } x \leq y \textbf{ do begin } x := (x+1)\,;\; y := (y-1) \textbf{ end}$$

$\langle \text{STMT} \rangle$
$\langle \text{WHILE-STMT} \rangle$
while $\langle \text{BOOL-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle \text{STMT} \rangle$
$\langle \text{WHILE-STMT} \rangle$
while $\langle \text{BOOL-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle$STMT$\rangle$
$\langle$WHILE-STMT$\rangle$
while $\langle$BOOL-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$ARITH-EXPR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle$STMT$\rangle$
$\langle$WHILE-STMT$\rangle$
while $\langle$BOOL-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$ARITH-EXPR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle$STMT$\rangle$
$\langle$WHILE-STMT$\rangle$
while $\langle$BOOL-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$ARITH-EXPR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle$STMT$\rangle$
$\langle$WHILE-STMT$\rangle$
while $\langle$BOOL-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$ARITH-EXPR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$
while $x \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle \text{STMT} \rangle$
$\langle \text{WHILE-STMT} \rangle$
while $\langle \text{BOOL-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{VAR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{VAR} \rangle \leq \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{VAR} \rangle \leq \langle \text{VAR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $x \leq \langle \text{VAR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $x \leq y$ **do** $\langle \text{STMT} \rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle \text{STMT} \rangle$
$\langle \text{WHILE-STMT} \rangle$
while $\langle \text{BOOL-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{VAR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{VAR} \rangle \leq \langle \text{ARITH-EXPR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $\langle \text{VAR} \rangle \leq \langle \text{VAR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $x \leq \langle \text{VAR} \rangle$ **do** $\langle \text{STMT} \rangle$
while $x \leq y$ **do** $\langle \text{STMT} \rangle$
while $x \leq y$ **do** $\langle \text{BLOCK-STMT} \rangle$

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle$STMT$\rangle$
$\langle$WHILE-STMT$\rangle$
while $\langle$BOOL-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$ARITH-EXPR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$
while $x \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$
while $x \leq y$ **do** $\langle$STMT$\rangle$
while $x \leq y$ **do** $\langle$BLOCK-STMT$\rangle$
while $x \leq y$ **do begin** $\langle$STMT-LIST$\rangle$ **end**

Context-Free Grammars

while $x \leq y$ **do begin** $x := (x + 1)\,;\; y := (y - 1)$ **end**

$\langle$STMT$\rangle$
$\langle$WHILE-STMT$\rangle$
while $\langle$BOOL-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$ARITH-EXPR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle\langle$COMPARE-OP$\rangle\langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$ARITH-EXPR$\rangle$ **do** $\langle$STMT$\rangle$
while $\langle$VAR$\rangle \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$
while $x \leq \langle$VAR$\rangle$ **do** $\langle$STMT$\rangle$
while $x \leq y$ **do** $\langle$STMT$\rangle$
while $x \leq y$ **do** $\langle$BLOCK-STMT$\rangle$
while $x \leq y$ **do begin** $\langle$STMT-LIST$\rangle$ **end**
...

Context-Free Grammars

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

$$G = (\Pi, \Sigma, S, P)$$

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)

$$A \to \alpha$$

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

Context-Free Grammars

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

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

$$A \to aBBb$$
$$A \to AaA$$
$$B \to \varepsilon$$
$$B \to bCA$$
$$C \to AB$$
$$C \to a$$
$$C \to b$$

Context-Free Grammars

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

$$A \to \alpha_1 \qquad A \to \alpha_2 \qquad A \to \alpha_3$$

we can write them in a more succinct way as

$$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$$

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

A

Context-Free Grammars

Grammars are used for generating words.

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

$$\underline{A} \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

$\underline{A}$

Context-Free Grammars

Grammars are used for generating words.

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

$$\underline{A} \to \underline{aBBb} \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

$A \Rightarrow aBBb$

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

$A \Rightarrow aBBb \Rightarrow abCABb$

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

$A \Rightarrow aBBb \Rightarrow abCABb$

Context-Free Grammars

Grammars are used for generating words.

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

$$\underline{A} \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$\underline{A} \to \underline{aBBb} \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \underline{\varepsilon} \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

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

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

Context-Free Grammars

Grammars are used for generating words.

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

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$\underline{C} \to AB \mid a \mid \underline{b}$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \underline{\varepsilon} \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \underline{\varepsilon} \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and P contains rules

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar 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

$$aCBbA \Rightarrow aCbCAbA$$

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** $L(G)$ generated by a grammar $G = (\Pi, \Sigma, S, P)$ 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.,

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

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 $G = (\Pi, \Sigma, S, P)$ where $\Pi = \{S\}$, $\Sigma = \{a, b\}$, and P contains

$$S \to aSb \mid \varepsilon$$

Context-Free Grammars

Example: We want to construct a grammar generating the language

$$L = \{a^n b^n \mid n \geq 0\}$$

Grammar $G = (\Pi, \Sigma, S, P)$ where $\Pi = \{S\}$, $\Sigma = \{a, b\}$, and P contains

$$S \to aSb \mid \varepsilon$$

$$S \Rightarrow \varepsilon$$
$$S \Rightarrow aSb \Rightarrow ab$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$
$$\ldots$$

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:

$$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$$

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:

$$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaaba$$

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:

$$S \to \varepsilon \mid (S) \mid SS$$

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:

$$S \to \varepsilon \mid (S) \mid SS$$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (SS)(S) \Rightarrow ((S)S)(S) \Rightarrow$
$(()S)(S) \Rightarrow (()(S))(S) \Rightarrow (()())(S) \Rightarrow (()())((S)) \Rightarrow$
$(()())(())$

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:

$$E \to a \mid E + E \mid E * E \mid (E)$$

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:

$$E \to a \mid E + E \mid E * E \mid (E)$$

$$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow (E) * E + E \Rightarrow (E + E) * E + E \Rightarrow$$
$$(a + E) * E + E \Rightarrow (a + a) * E + E \Rightarrow (a + a) * a + E \Rightarrow (a + a) * a + (E) \Rightarrow$$
$$(a + a) * a + (E * E) \Rightarrow (a + a) * a + (a * E) \Rightarrow (a + a) * a + (a * a)$$

Derivation Tree

$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

Derivation Tree

A

$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

A

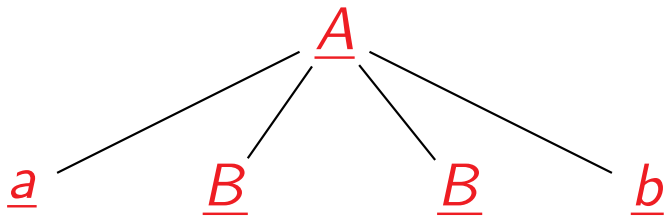
Derivation Tree

$\underline{A}$

$\underline{A} \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

$\underline{A}$

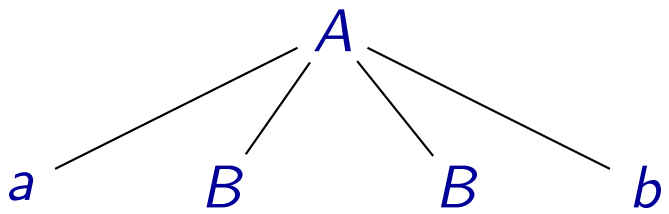
Derivation Tree



$\underline{A} \to \underline{aBBb} \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

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

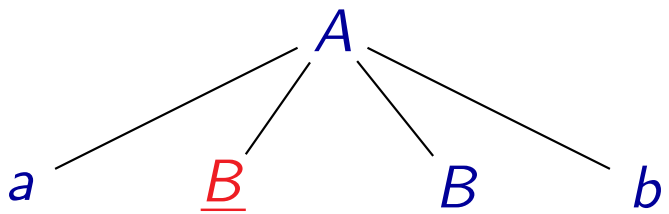
Derivation Tree



$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

$A \Rightarrow aBBb$

Derivation Tree

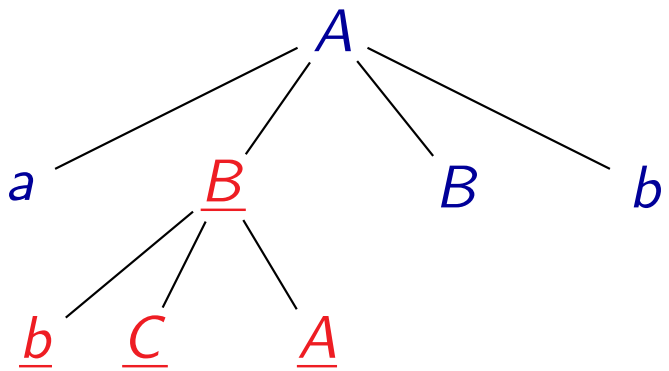


$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

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

Derivation Tree

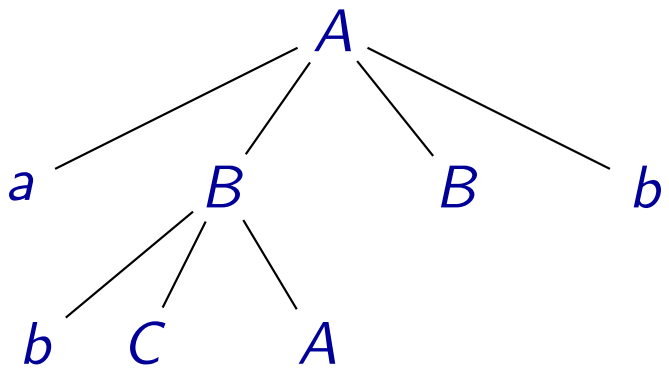
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid \underline{bCA}$
$C \to AB \mid a \mid b$



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

Derivation Tree

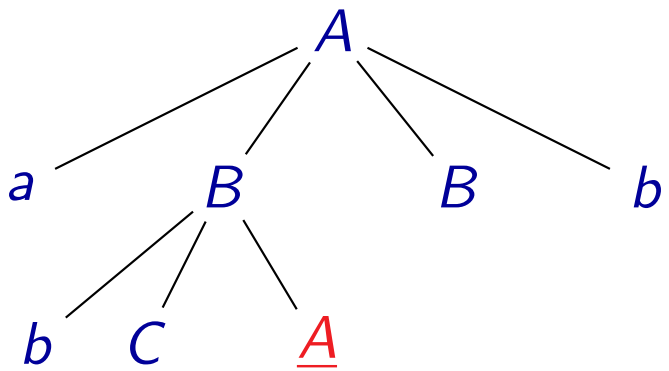
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb$

Derivation Tree

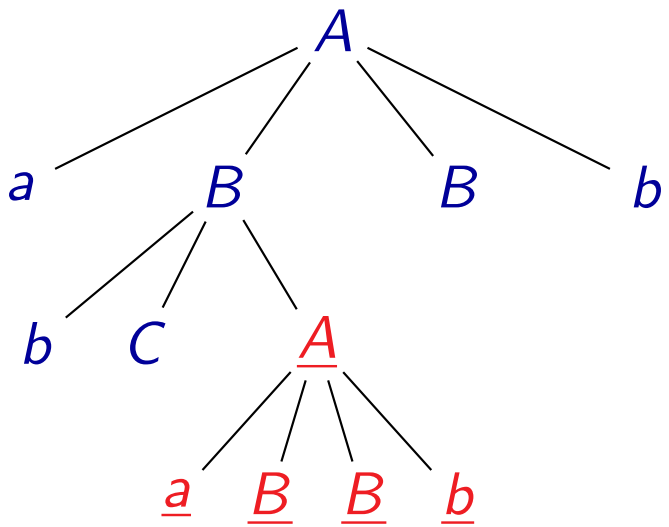
$\underline{A} \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

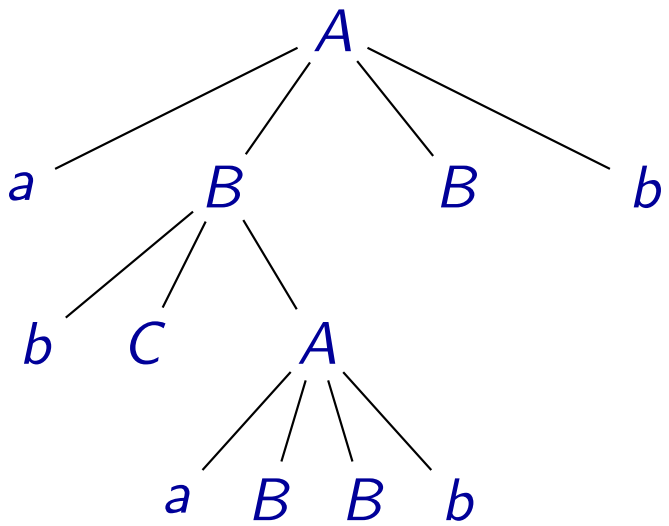
$\underline{A} \to \underline{aBBb} \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

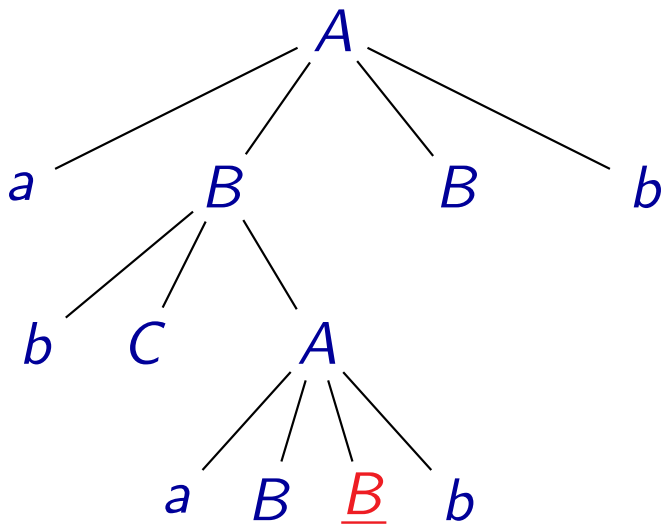
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

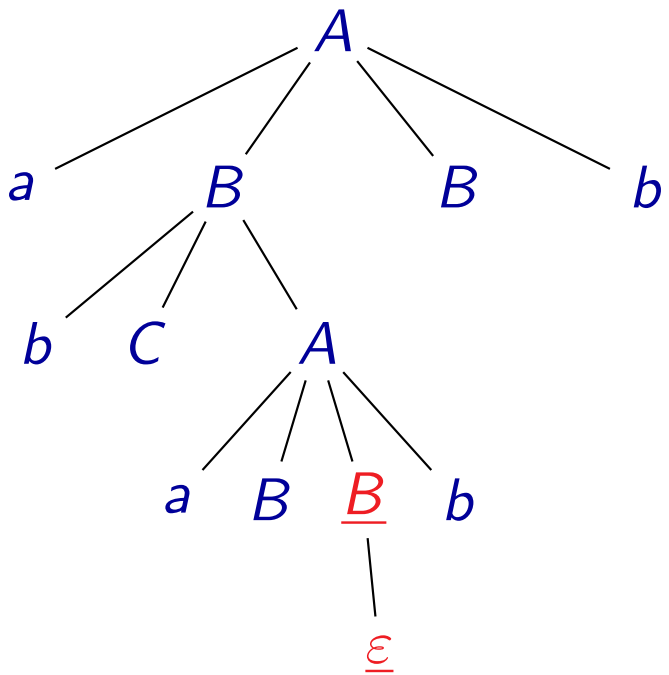
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

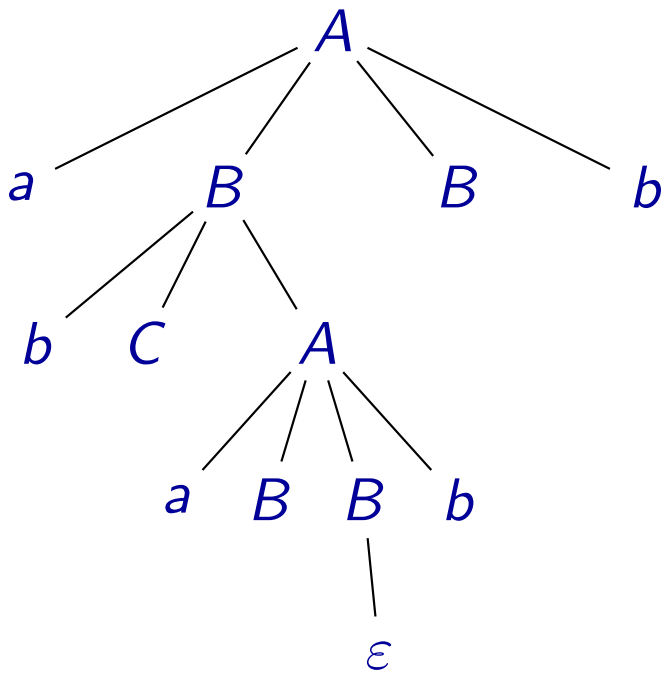
$A \to aBBb \mid AaA$
$\underline{B} \to \underline{\varepsilon} \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

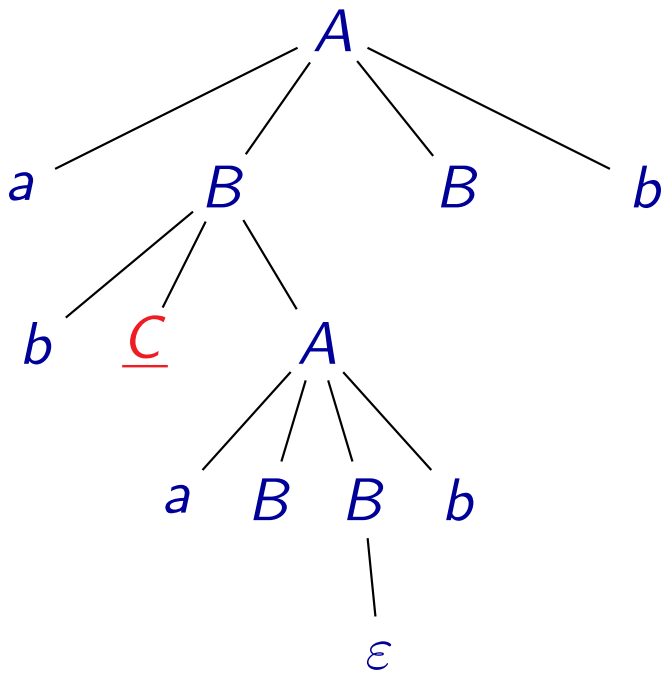
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

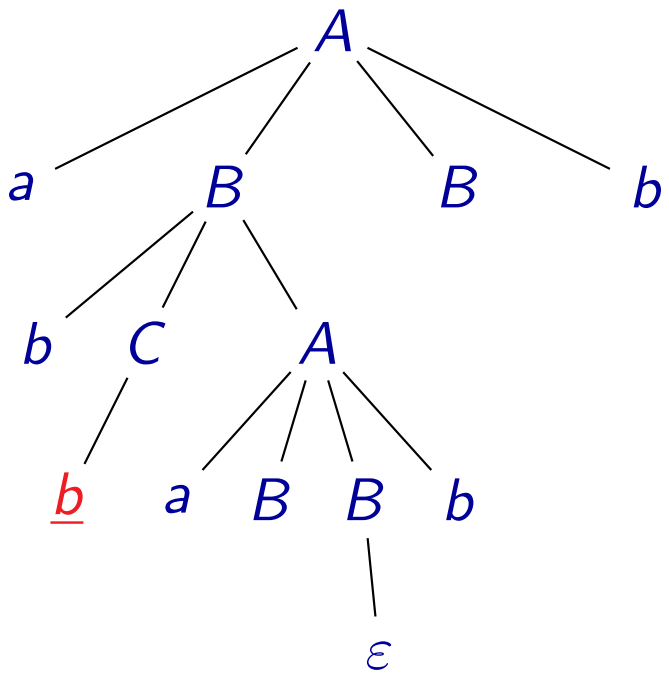
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$\underline{C} \to AB \mid a \mid b$



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

Derivation Tree

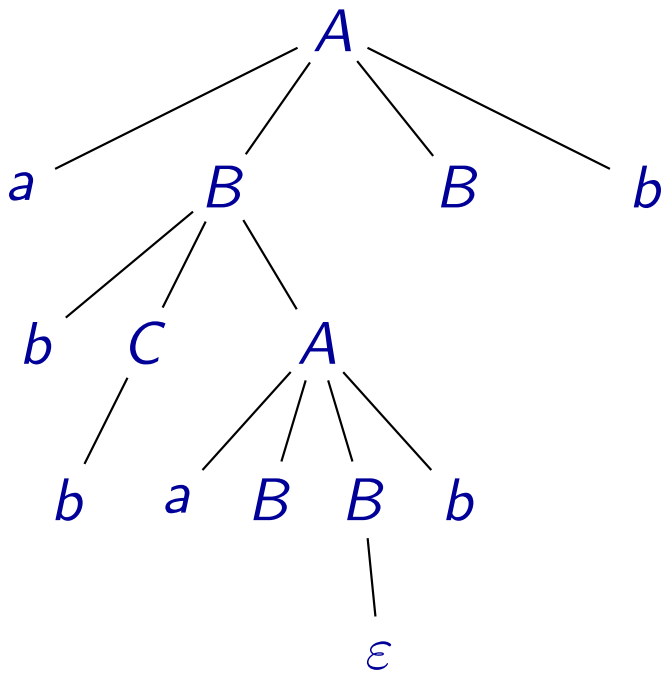
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$\underline{C} \to AB \mid a \mid \underline{b}$



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

Derivation Tree

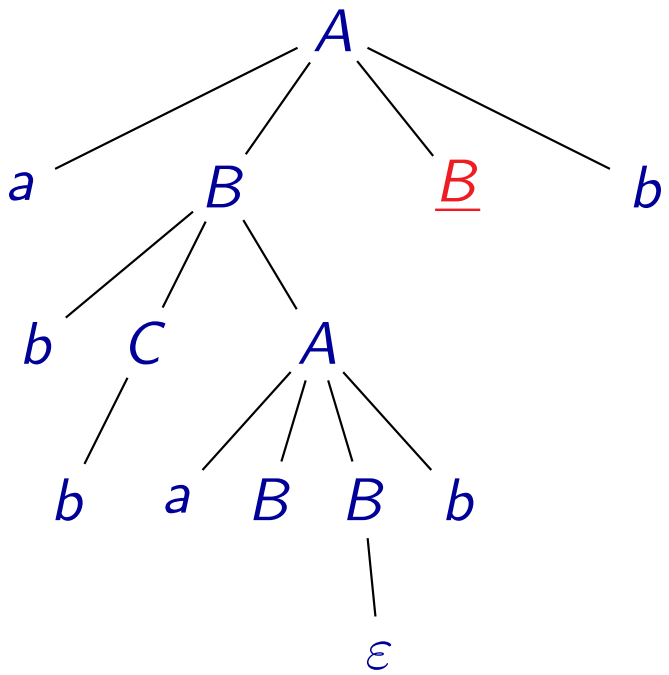
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

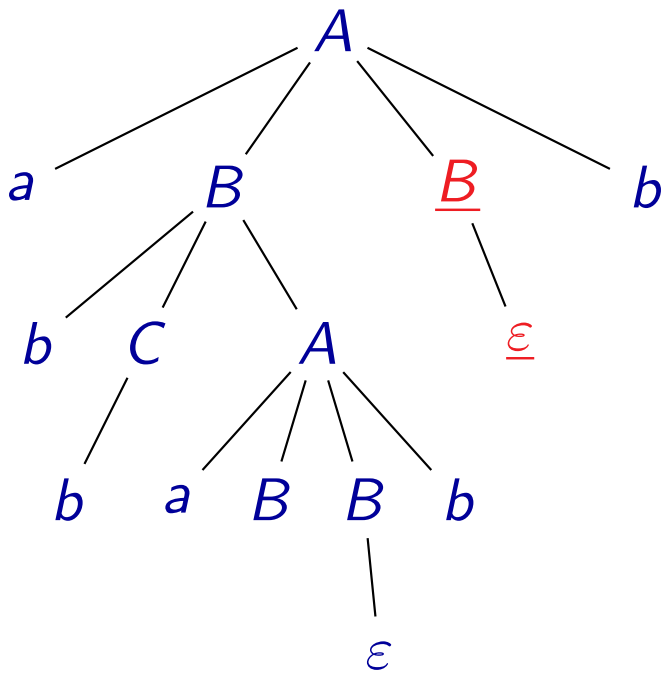
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

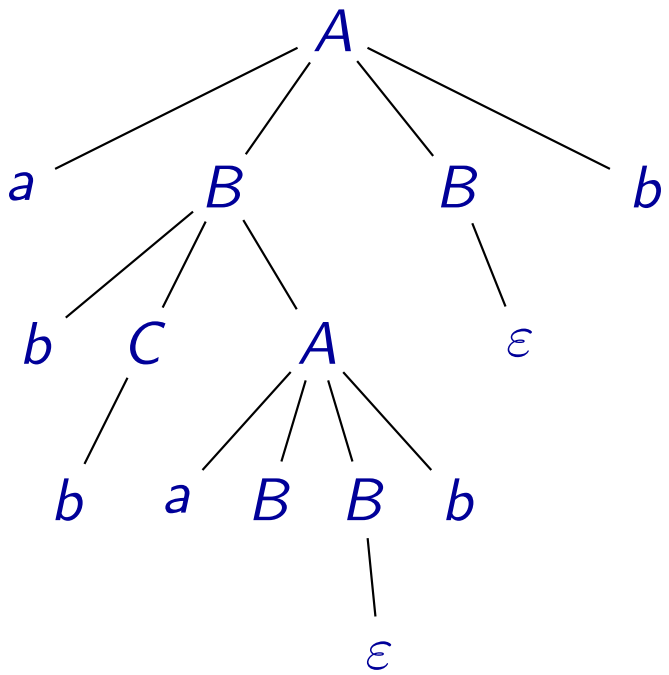
$A \to aBBb \mid AaA$
$\underline{B} \to \underline{\varepsilon} \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

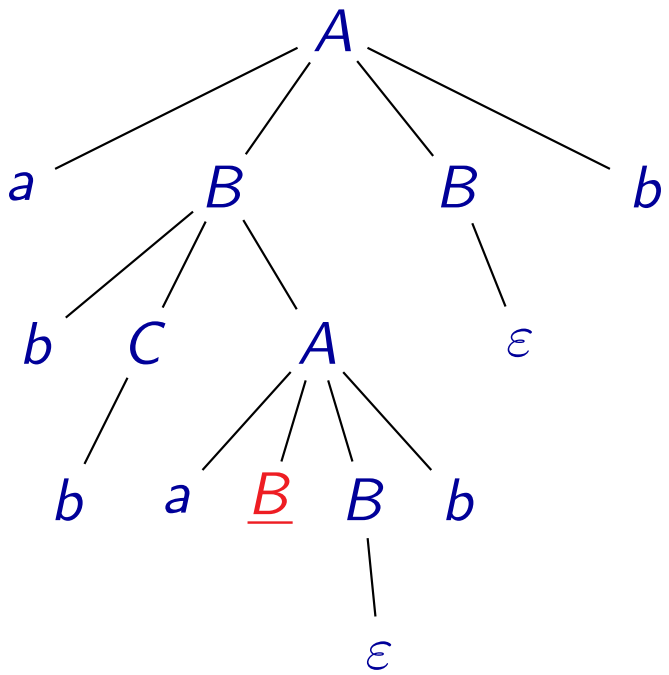
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

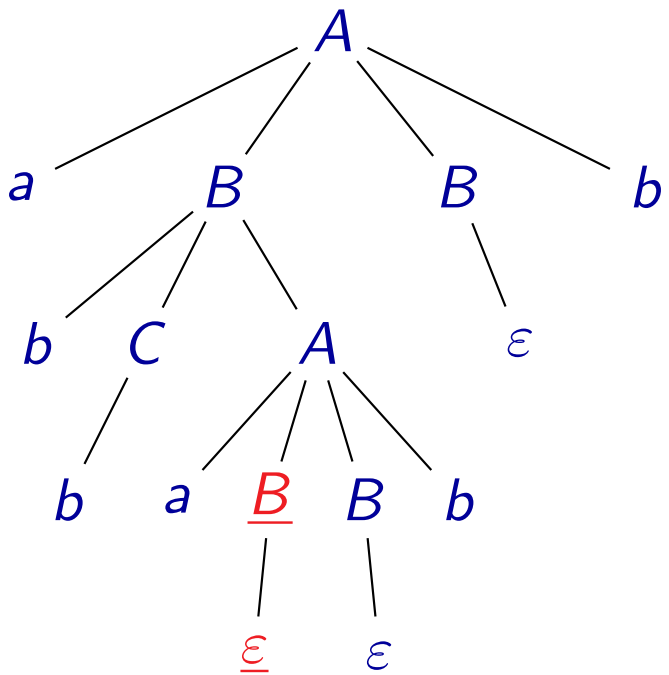
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

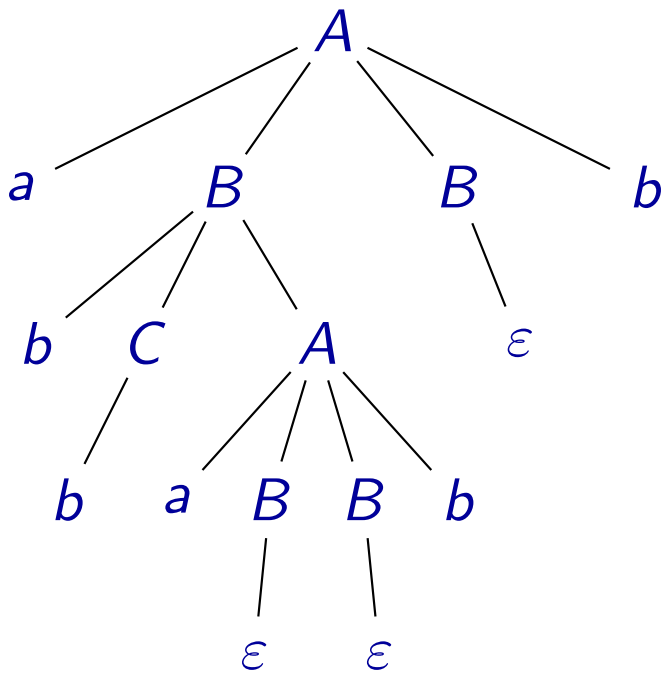
$A \to aBBb \mid AaA$
$\underline{B} \to \underline{\varepsilon} \mid bCA$
$C \to AB \mid a \mid b$



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

Derivation Tree

$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



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

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 $A \to \alpha$.

Left and Right Derivation

$$E \to E + E \mid E * E \mid (E) \mid a$$

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 G_1 and G_2 are **equivalent** if they generate the same language, i.e., if $L(G_1) = L(G_2)$.

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.

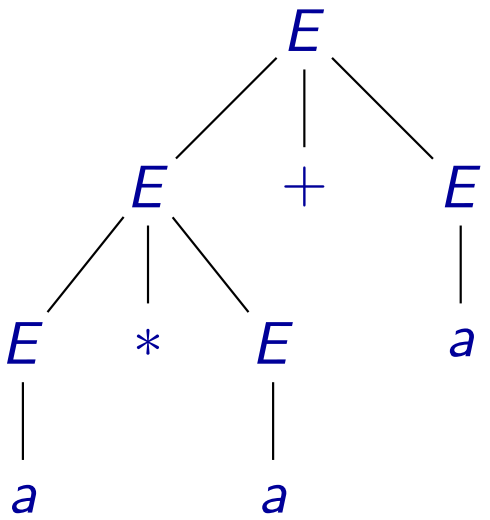
Ambiguous Grammars

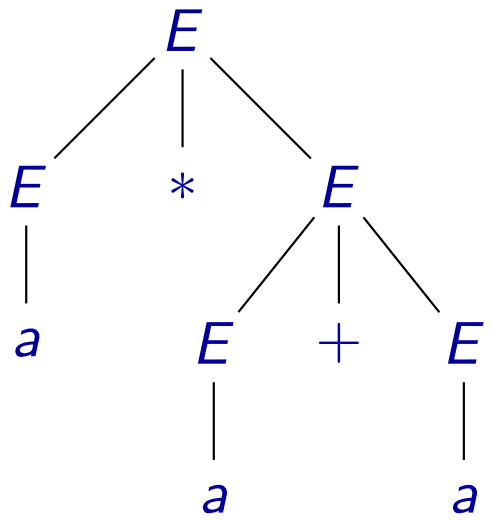
A grammar G is **ambiguous** if there is a word $w \in L(G)$ that has two different derivation trees, resp. two different left or two different right derivations.

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

$$E \to E + E \mid E * E \mid (E) \mid a$$

can be replaced with the equivalent grammar

$$\begin{aligned} E &\to T \mid T + E \\ T &\to F \mid F * T \\ F &\to a \mid (E) \end{aligned}$$

Remark: If there is no unambiguous grammar equivalent to a given ambiguous grammar, we say it is **inherently ambiguous**.

Context-Free Languages

Definition

A language L is **context-free** if there exists some context-free grammar G such that $L = L(G)$.

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 $G_1 = (\Pi_1, \Sigma, S_1, P_1)$ and $G_2 = (\Pi_2, \Sigma, S_2, P_2)$, and can assume that $\Pi_1 \cap \Pi_2 = \emptyset$ and $S \notin \Pi_1 \cup \Pi_2$.

- Grammar G such that $L(G) = L(G_1)L(G_2)$:

$$G = (\Pi_1 \cup \Pi_2 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\})$$

- Grammar G such that $L(G) = L(G_1) \cup L(G_2)$:

$$G = (\Pi_1 \cup \Pi_2 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\})$$

- Grammar G such that $L(G) = L(G_1)^*$:

$$G = (\Pi_1 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup \{S \to \varepsilon, S \to S_1 S\})$$

Computability and Complexity

What Is an Algorithm?

Algorithm

An **algorithm** is a mechanical procedure consisting of some simple elementary steps that for any given **input** produces an **output**.

An algorithm can be described:

- in plain English
- by a pseudocode
- as a computer program in a programming language
- as a hardware circuit
- ...

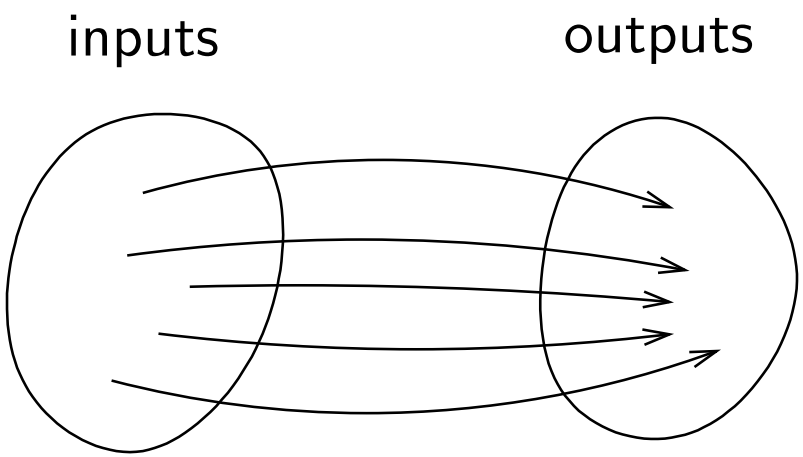
Algorithms are used for solving **problems**.

Problems

Problem

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

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



Examples of Problems

Problem "Sorting"

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

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

Example:

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

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

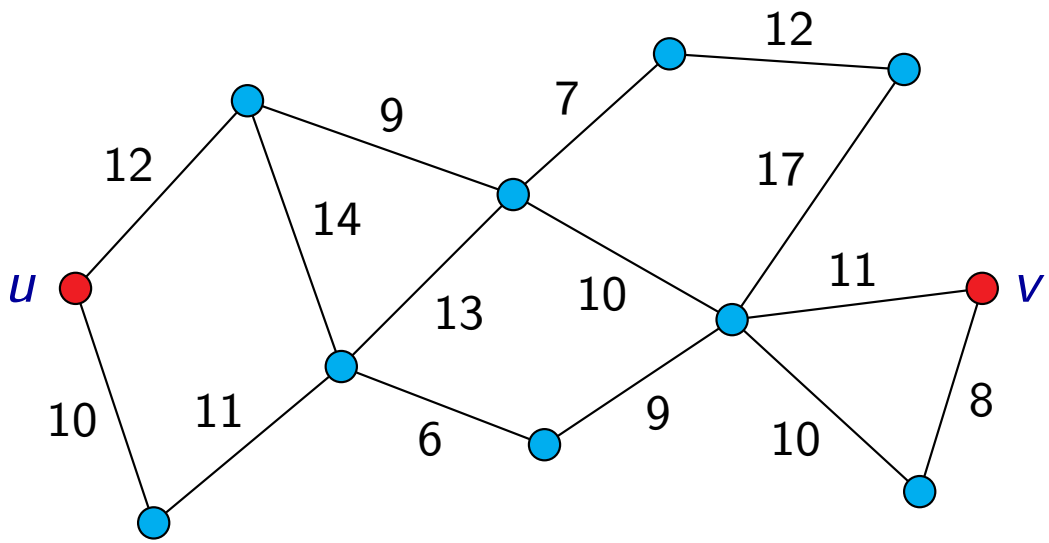
Examples of Problems

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

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

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

Example:



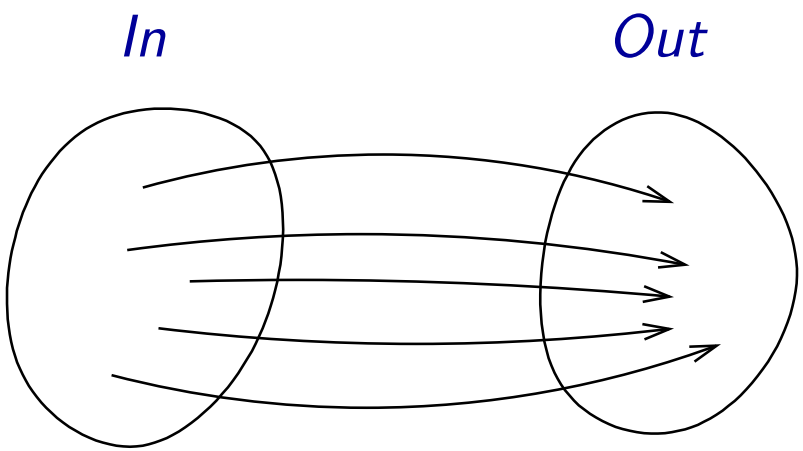
Problems

Problem

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

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

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



Encoding of Input and Output

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

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

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

Then an input can be for example the word

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

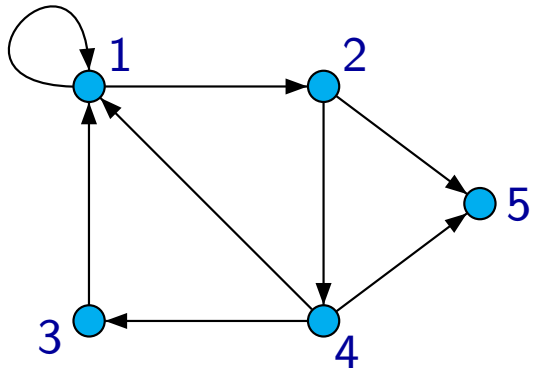
and the output is then the word

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

Encoding of Input and Output

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

For example, the following graph



can be represented as word

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

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

Encoding of Input and Output

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

Encoding of Input and Output

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

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

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

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

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

Other Examples of Problems

Problem "Primality"

Input: A natural number n.

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

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

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

Decision Problems

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

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

Example:

Problem "Primality"

Input: A natural number n.

Question: Is n a prime?

Decision Problems

Decision problem

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

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

Decision Problems and Languages

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

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

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

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

An Example of a Decision Problem

SAT problem (boolean satisfiability problem)

Input: Boolean formula φ.

Question: Is φ satisfiable?

Example:

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

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

Optimization Problems

Other special case are the so called optimization problems.

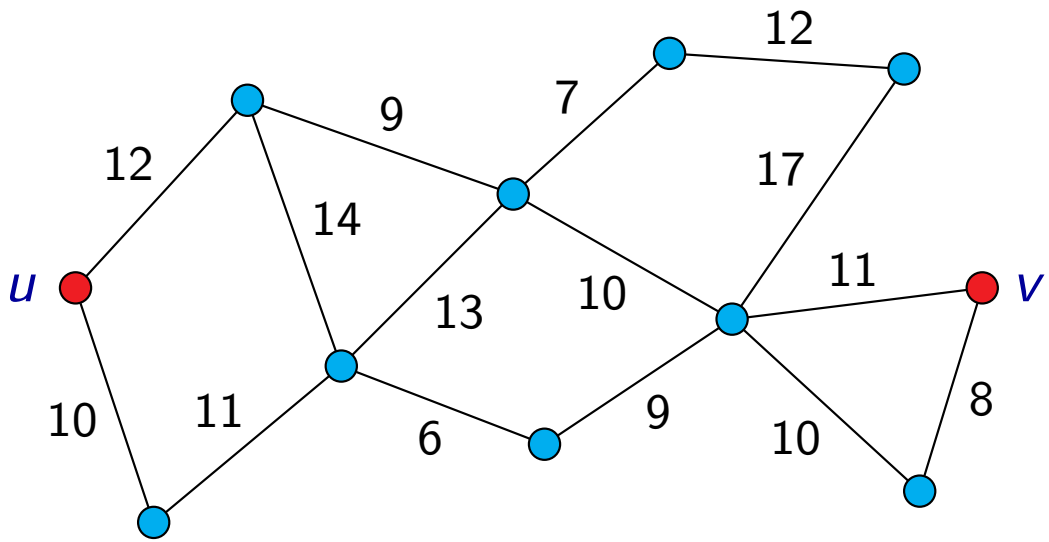
An **optimization problem** is a problem where the aim is to choose, from a set of feasible solutions, a solution that in some respect is optimal.

Optimization Problems

Other special case are the so called optimization problems.

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

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



Optimization Problems

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

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

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

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

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

Optimization Problems

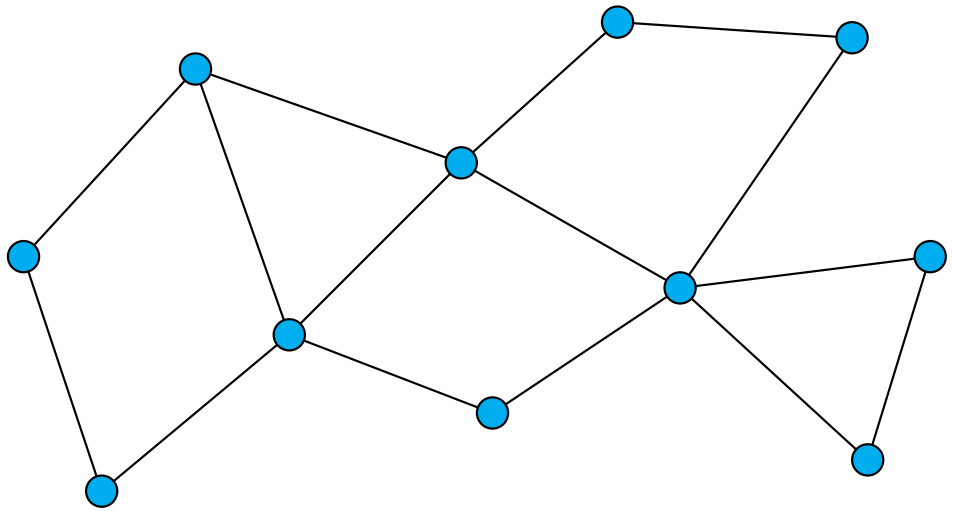
- The optimization problems, where g is min, are called **minimization problems**.
- The optimization problems, where g is max, are called **maximization problems**.

Examples of Problems

Problem "Coloring of a graph with k colors"

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

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



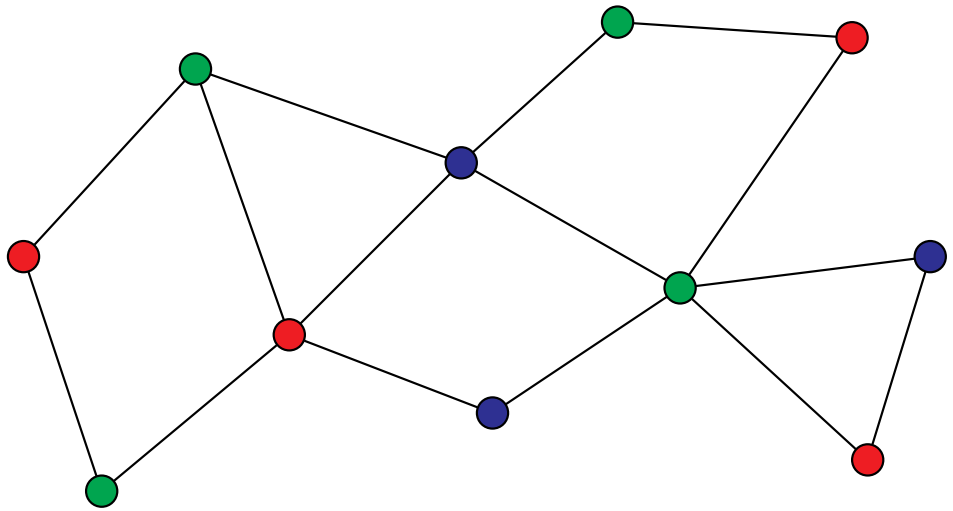
$k = 3$

Examples of Problems

Problem "Coloring of a graph with k colors"

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

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



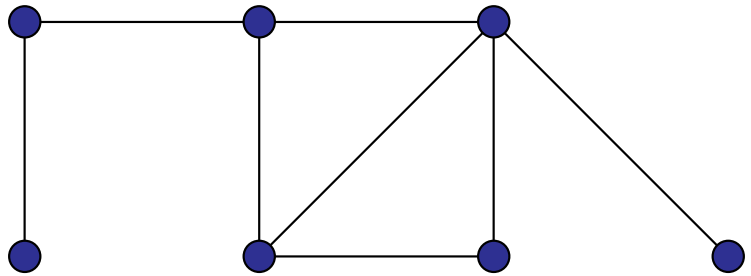
$k = 3$

Examples of Problems

Independent set (IS) problem

Input: An undirected graph G, a number k.

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



$k = 4$

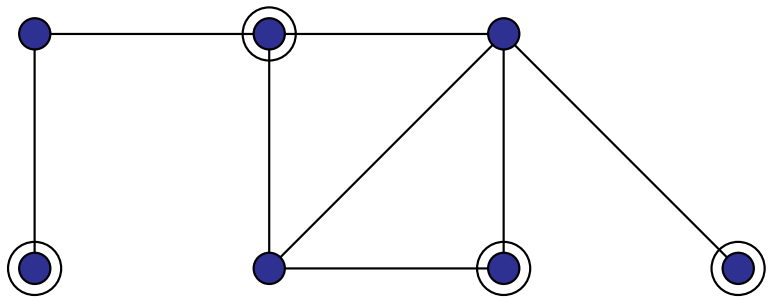
Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

Examples of Problems

Independent set (IS) problem

Input: An undirected graph G, a number k.

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

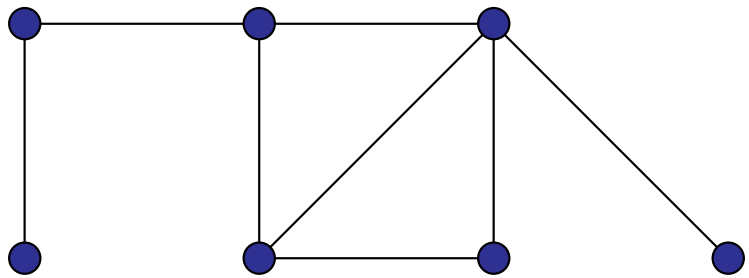


$k = 4$

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

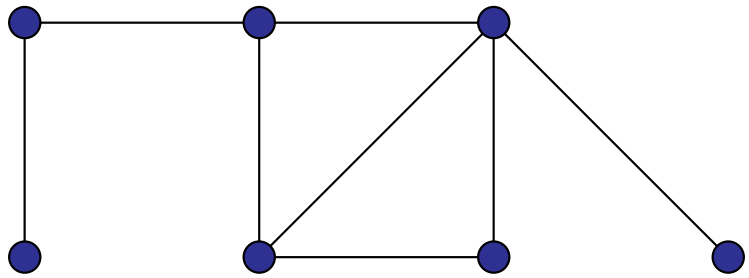
Independent Set (IS) Problem

An example of an instance where the answer is YES:



$k = 4$

An example of an instance where the answer is NO:



$k = 5$

ILP – Integer Linear Programming

Problem ILP (integer linear programming)

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

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

An example of an instance of the problem:

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

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

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

ILP – Integer Linear Programming

One of solutions of the system

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

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

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

because

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

So the answer for this instance is YES.

Examples of Problems

Problem

Input: Deterministic finite automata A_1 and A_2.

Question: Is $L(A_1) = L(A_2)$?

Problem

Input: Context-free grammars G_1 and G_2.

Question: Is $L(G_1) = L(G_2)$?

Solutions of Problems

Solving a problem

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

1. It halts after some finite number of steps for any input of the given problem (for any input instance).
2. It produces an output from the set of possible outputs that satisfies the conditions specified in the problem statement.

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

Remark: **correctness of an algorithm** — the algorithm solves the given problem

Solutions of Problems

To each algorithm A we can assign the function

$$f_A : In \rightarrow Out$$

where:

- In is the set of inputs for the algorithm A,
- Out is the set of outputs, which are generated by the algorithm A,
- $f_A(x)$ is the output, generated by the algorithm A for an input $x \in In$.

The function need not be **total** (i.e., the value of $f_A(x)$ need not be defined for each $x \in In$), it can be **partial**:

- the value of $f_A(x)$ is undefined if the computation of the algorithm A for an input x never halts, if an error occurs, etc.

Solutions of Problems

If we have a problem $P = (In, Out, R)$ and an algorithm A realizing a function $f_A : In \to Out$, we say that

$$\textit{the algorithm } A \textit{ solves the problem } P$$

if:

- the value of $f_A(x)$ is defined for each $x \in In$,
- for each $x \in In$ we have $(x, f_A(x)) \in R$

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**.

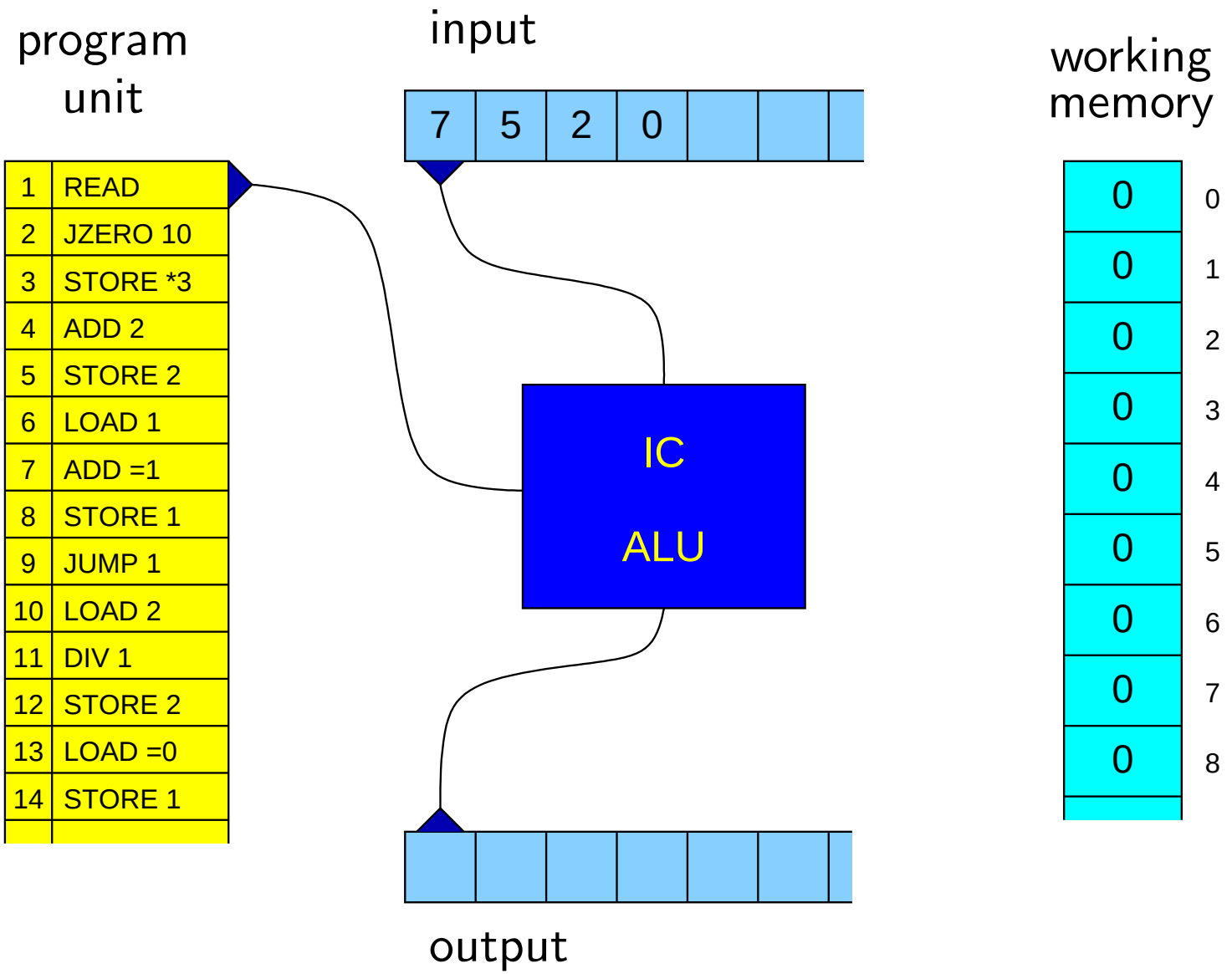
Random Access Machine

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 $0, 1, 2, \ldots$; the content of the cells can be read and written to
- **Input tape** – read-only
- **Output tape** – write-only

Random Access Machine



Random Access Machine

Cells 0 and 1 are special and are used as **registers** of the RAM:

- **Cell 0** – a **working register** (accumulator) – the register that is one of operands for most of instructions and to which the result of most of operations is stored.
- **Cell 1** – an **index register** – it is used for indirect addressing.

Forms of **operands** of instructions ($i \in \mathbb{N}$):

form	the value of the operand
$=i$	the number i
i	the number stored in the cell at address i
$*i$	the number stored in the cell at address $i + j$, where j is the current value of the index register

Random Access Machine

Example:

`LOAD <op>`

reads the value of the operand `<op>` into the working register (i.e., into cell 0).

- `LOAD =5` – loads 5 into the working register
- `LOAD 5` – loads the value in cell 5 into the working register
- `LOAD *5` – loads the value in cell $5 + j$, where j is the current value of the index register, into the working register

Random Access Machine

Input and output instructions (they have no operands):

- `READ` – the value from the cell of input tape on which is the reading head is stored into the working register and the reading head is moved right
- `WRITE` – the value in the working register is written on the output tape and the output head is moved right by one

Intructions working with memory:

- `LOAD <op>` – the value of the operand is loaded into the working register
- `STORE <op>` – the value of the operand is rewritten with the content of the working register (an operand of the form $=i$ is not allowed)

Random Access Machine

Instruction for arithmetic operations:

- `ADD <op>` – the value in the working register is increased by the value of the operand (i.e., the value of the operand is added to it)
- `SUB <op>` – the value of the operand subtracted from the value of the working register
- `MUL <op>` – the value in the working register is multiplied by the value of the operand
- `DIV <op>` – the value in the working register is divided by the value of the operand (the result is truncated to an integer value)

Random Access Machine

Jump instructions:

- `JUMP <label>` – the computation will continue with an instruction determined by the label
- `JZERO <label>` – if the working register contains value 0, the computation will continue with an instruction determined by the label; otherwise it will continue with the following instruction
- `JGTZ <label>` – if the working register contains a positive value, the computation will continue with an instruction determined by the label; otherwise it will continue with the following instruction

A halting instruction:

- `HALT` – the computation is halted

Random Access Machine

Problem "Searching"

Input: An integer x and a sequence of integers $a_1, a_2, \ldots, a_n$ (where $a_i \neq 0$) terminated by 0.

Output: If $a_i = x$ then the output is i (if there are several such i then the smallest one), otherwise the output is 0.

```
start:  READ
        STORE   3
        LOAD    =1
 loop:  STORE   2
        READ
        JZERO   output
        SUB     3
        JZERO   found
        LOAD    2
        ADD     =1
        JUMP    loop
 found: LOAD    2
output: WRITE
        HALT
```

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0
⋮

Output:

Instructions: 0

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 0
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0
⋮

Output:

Instructions: 0

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0
⋮

Output:

Instructions: 1

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 0
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 2

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 1
Cell 1: 0
Cell 2: 0
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 3

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 1
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 4

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 13
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 5

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 13
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 6

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 4
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 7

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 4
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 8

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 1
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 9

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 10

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 11

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 12

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 5
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 13

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 5
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 14

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 15

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 16

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 17

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 18

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 19

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 20

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 21

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 22

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 23

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 24

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 25

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output: 3

Instructions: 26

Random Access Machine

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

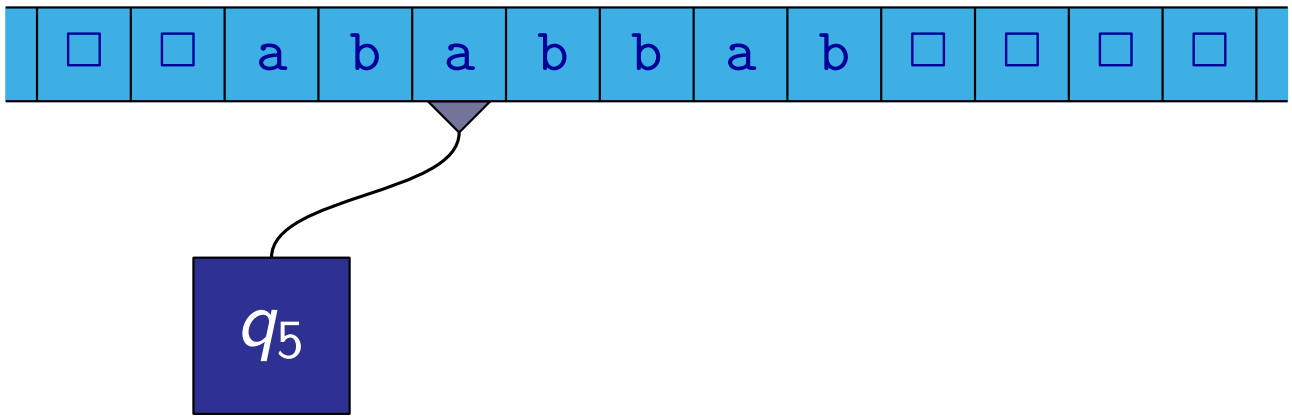
Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output: 3

Instructions: 27

Turing Machine

- We extend a deterministic finite automaton with the possibility of moving the head in both directions, of writing symbols on the tape, and we extend its tape into infinity.



Turing Machine

Definition

Formally, **Turing machine** is 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**

Turing Machine

- We assume that $\Gamma - \Sigma$ always contains a special element $\square$ denoting a **blank** symbol.
- A **configuration** is given by:
  - a state of the control unit
  - a content of the tape
  - a position of the head
- A computation over a word $w \in \Sigma^*$ starts in the **initial configuration** where:
  - 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

If the state of the control unit belongs to the set F then the next step is not defined and the computation halts.

We often choose the set of final states $F = \{q_{\text{yes}}, q_{\text{no}}\}$.

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_{yes}, the machine accepts the word w.
- If the state of the control unit after the computation over the word w is q_{no}, 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 $L(\mathcal{M})$ of a Turing machine $\mathcal{M}$ is the set of all words accepted by $\mathcal{M}$.

Turing Machine

- A Turing machine can give not only an answer YES or NO but it can also compute a function that assignes 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 $\Box$.

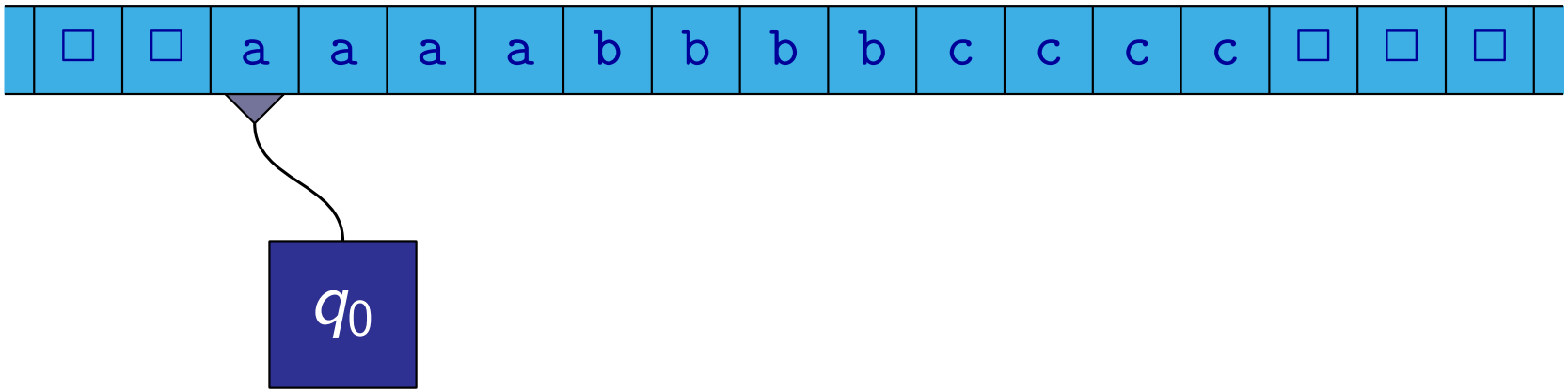
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



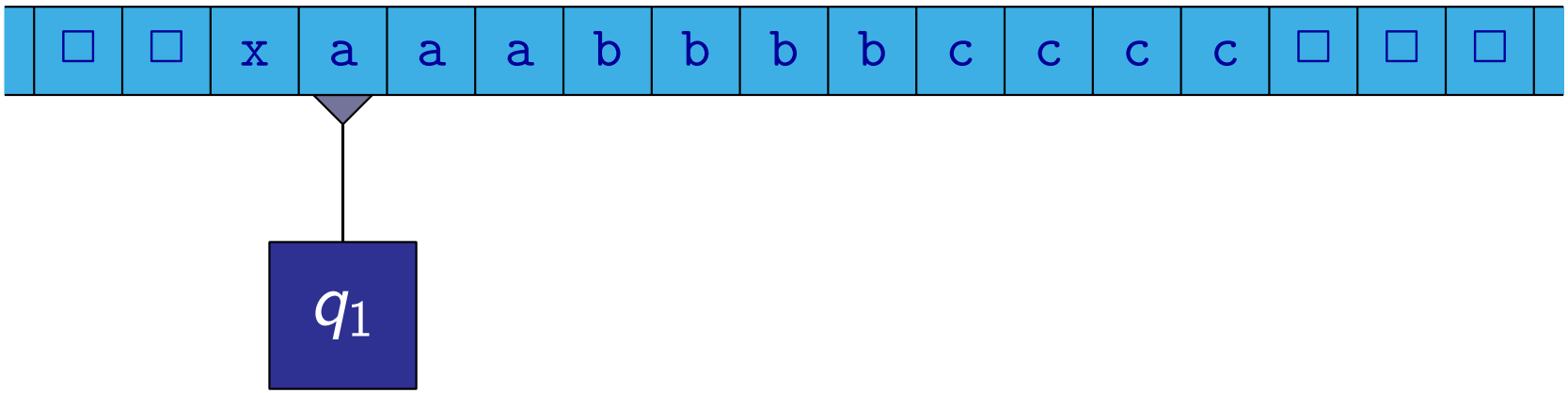
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



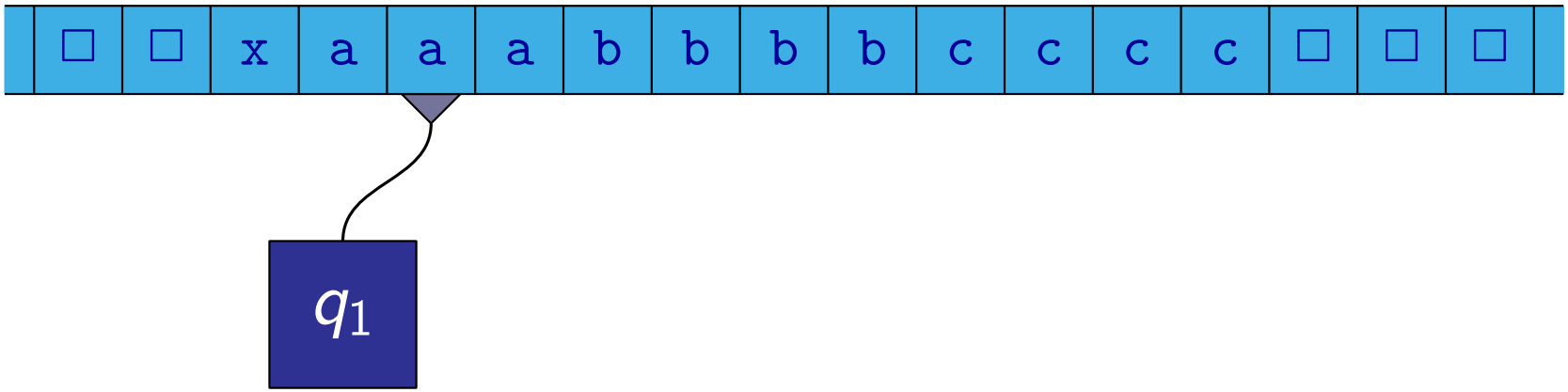
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



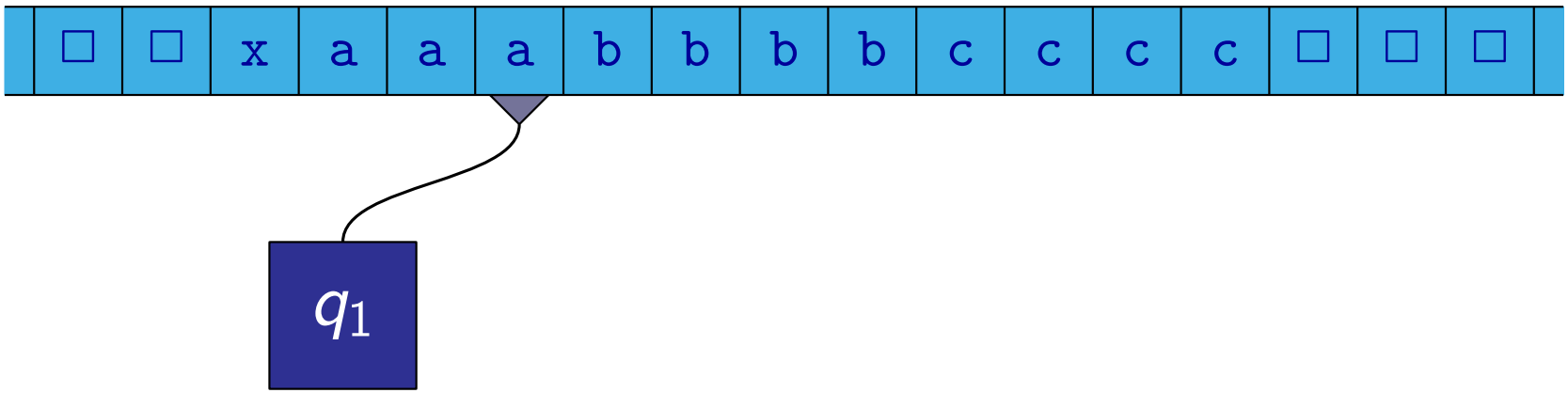
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



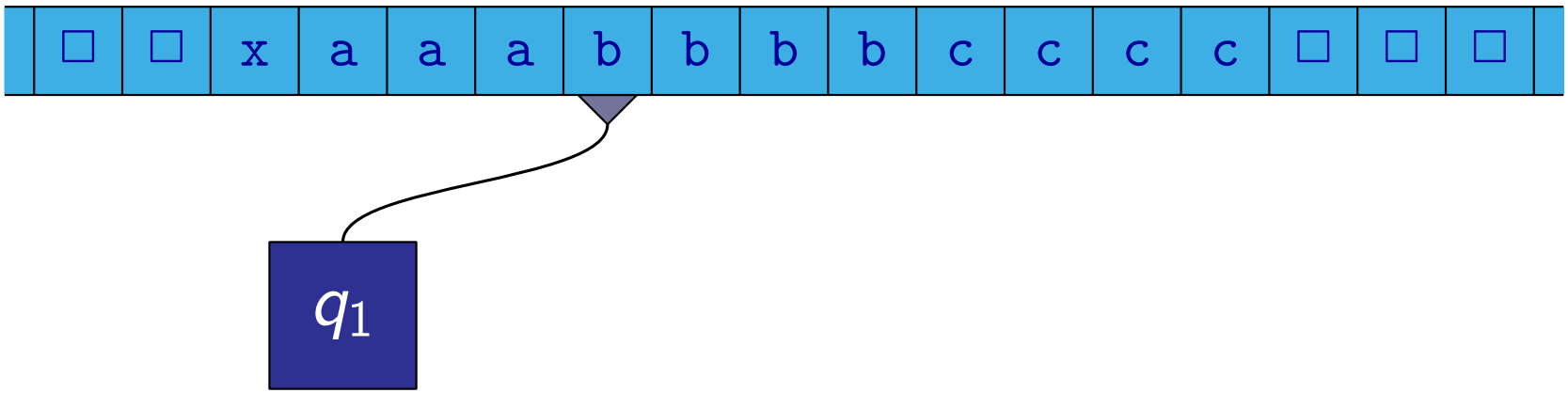
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



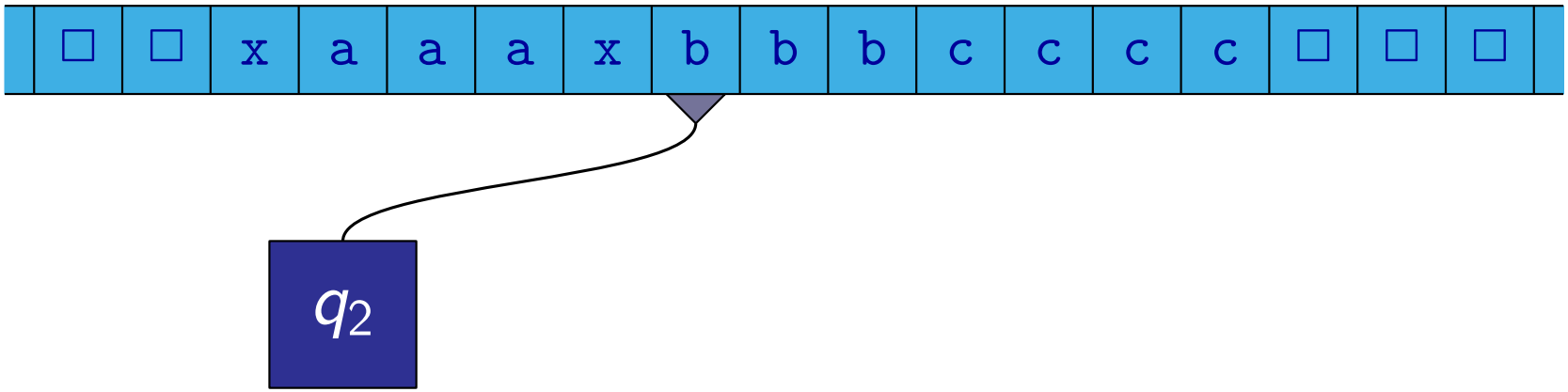
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



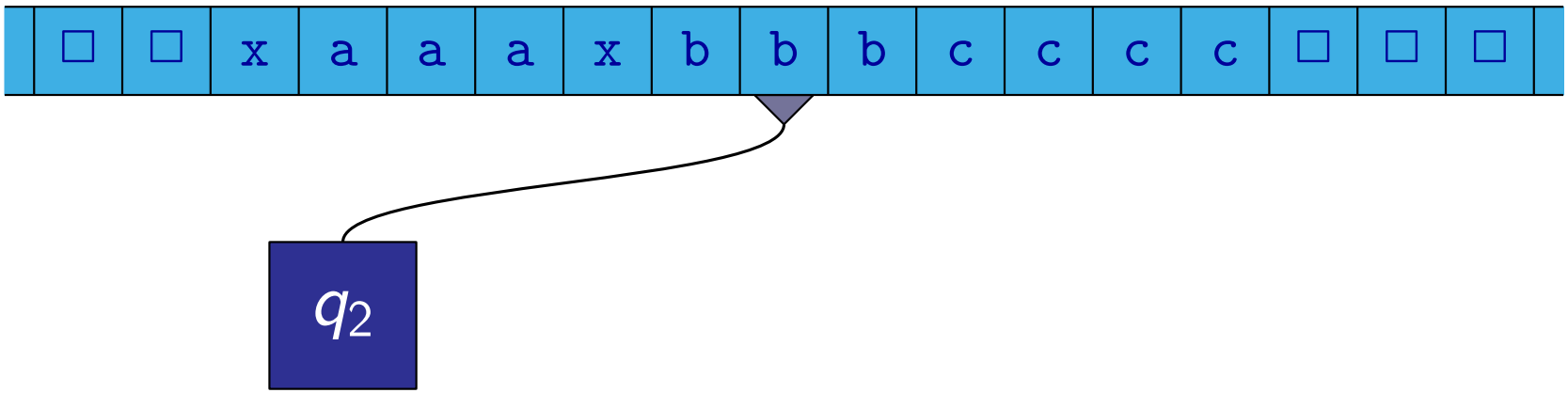
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



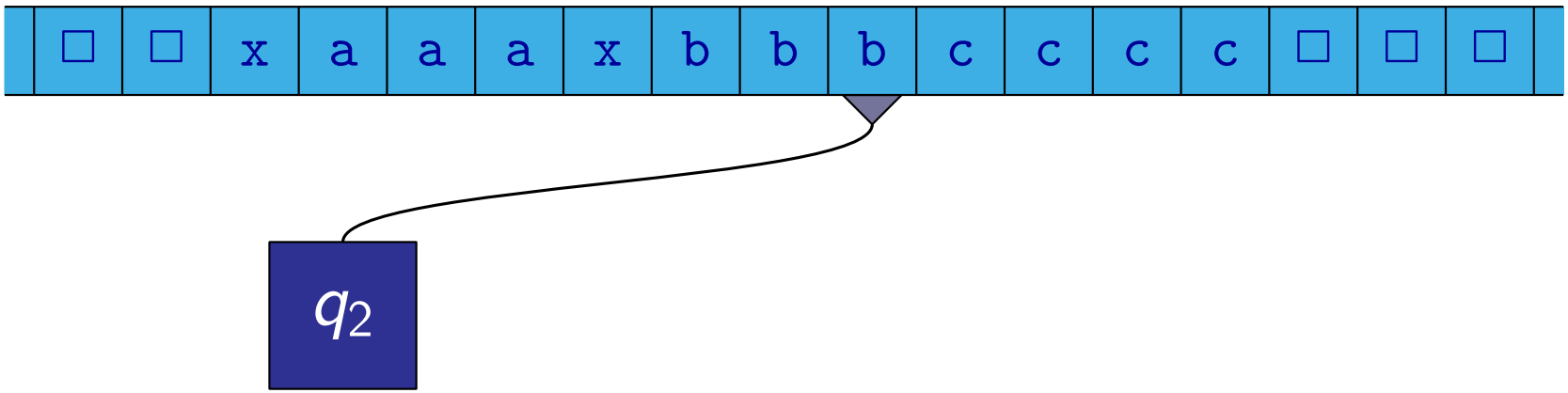
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



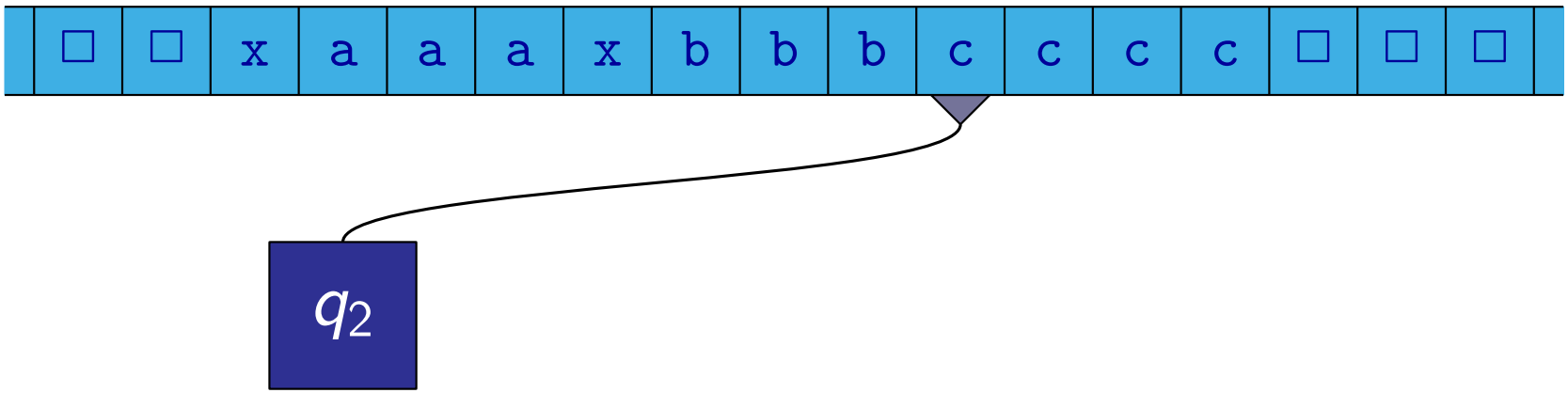
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



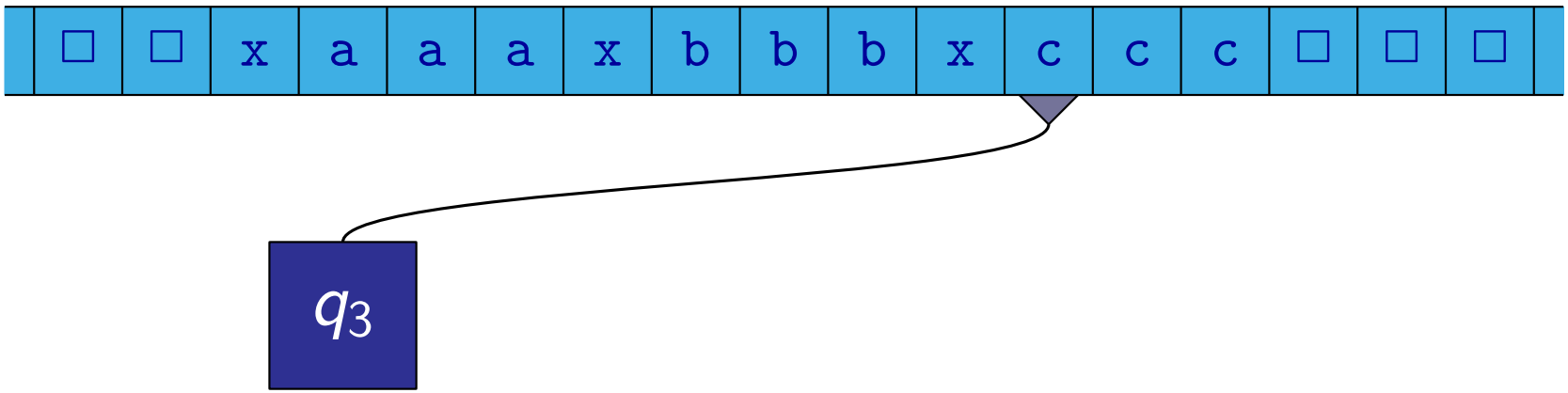
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



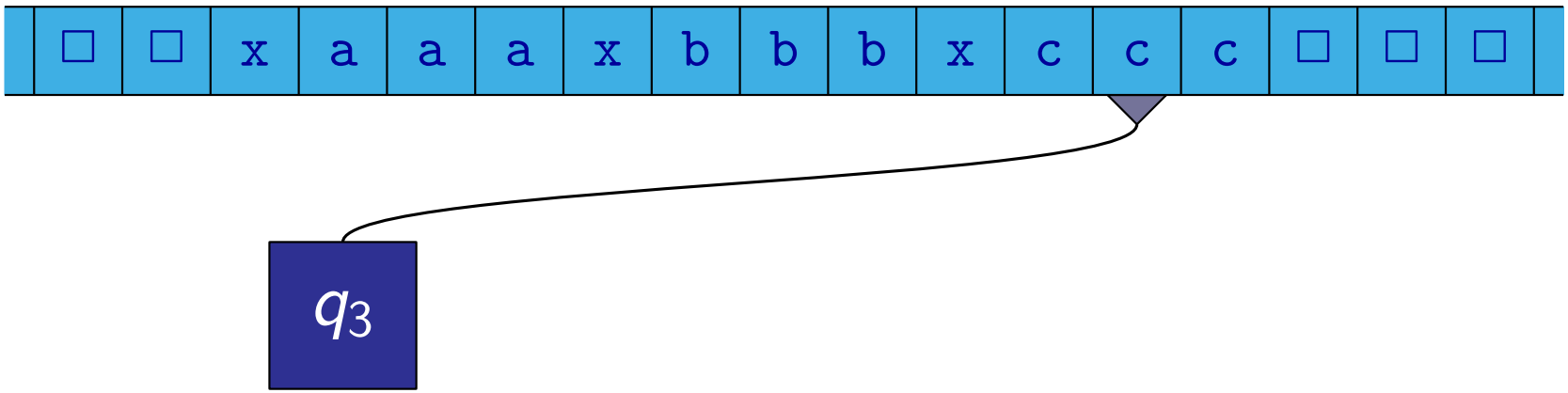
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



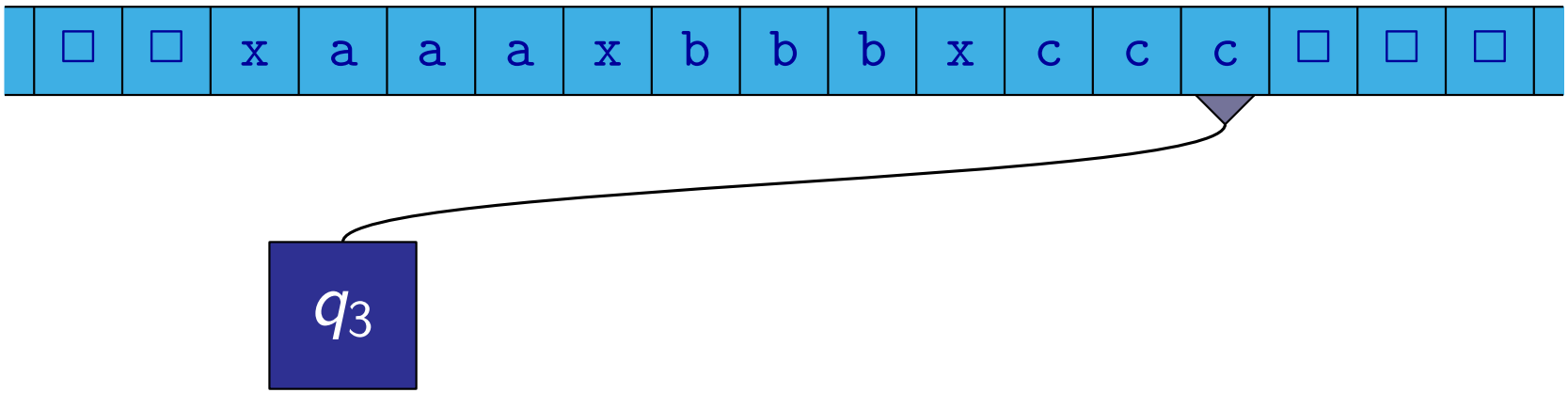
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



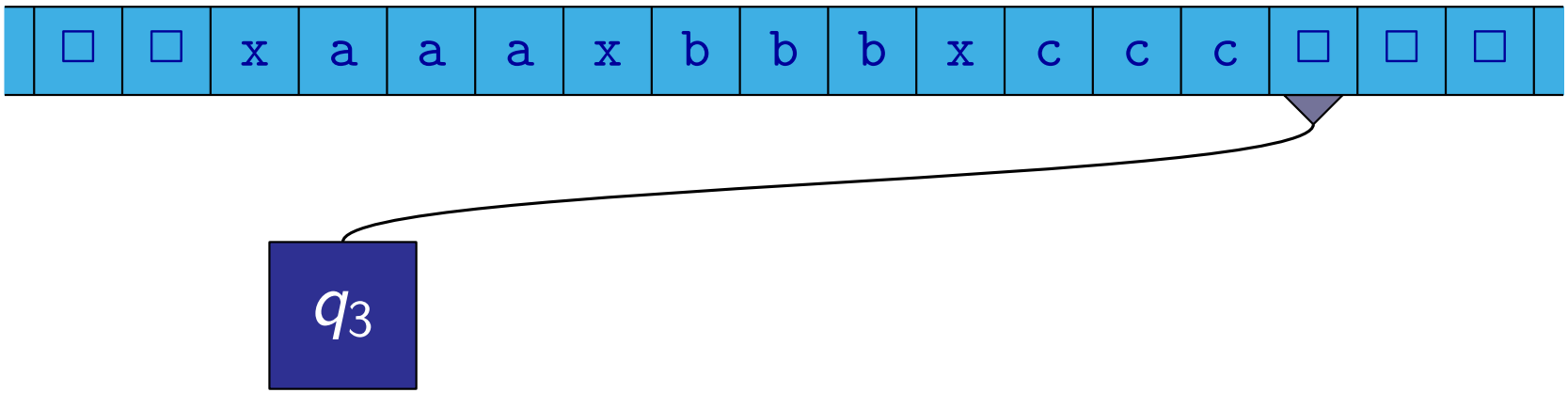
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



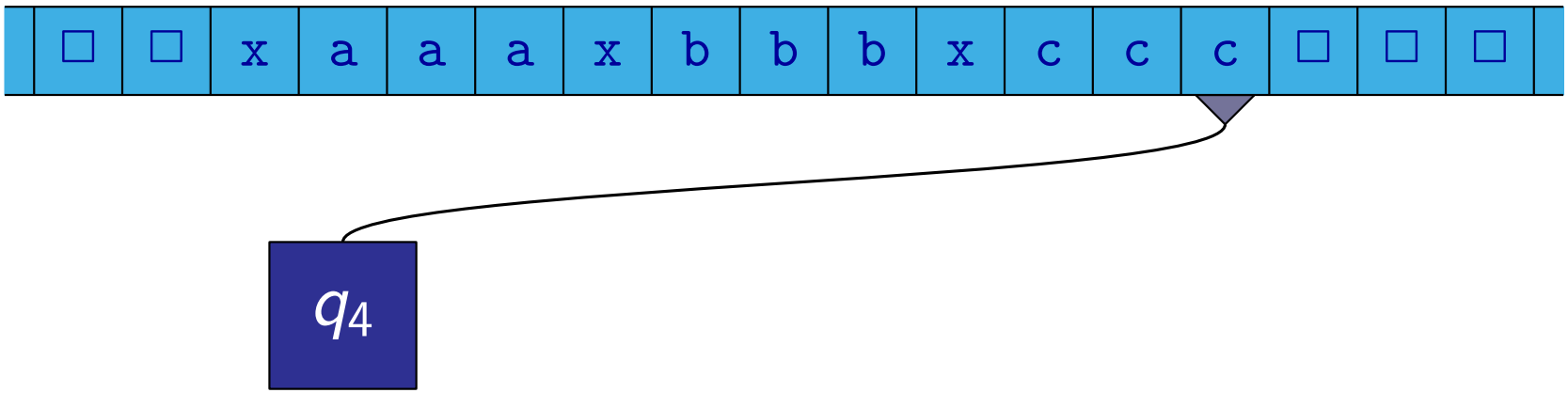
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



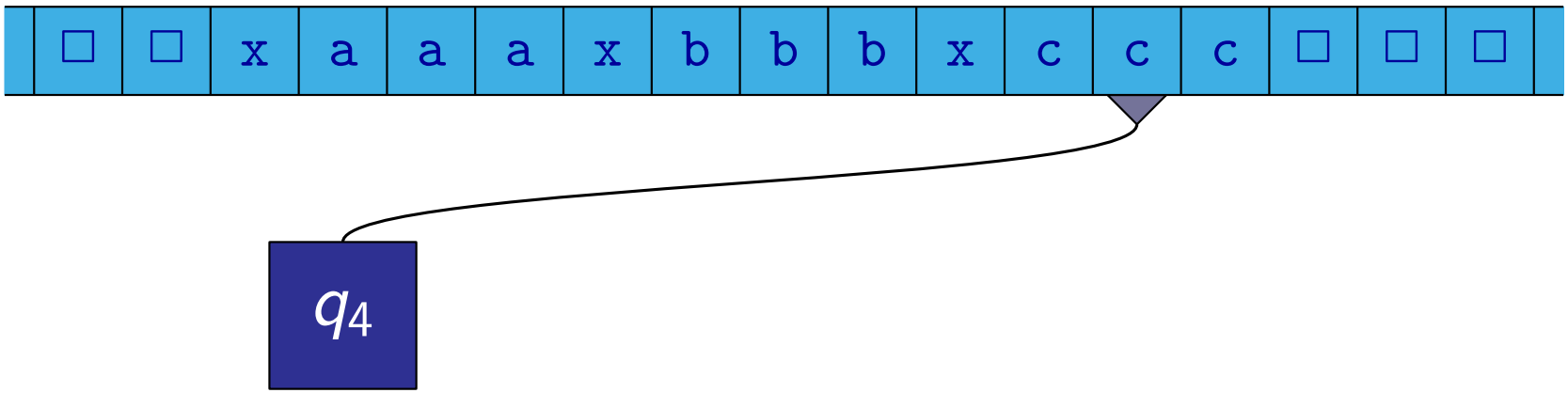
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



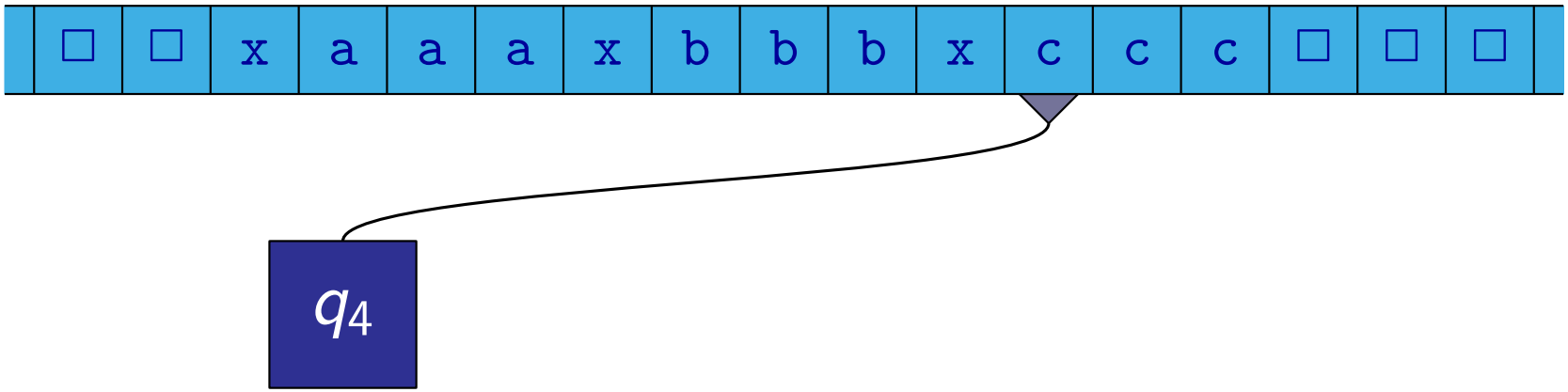
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



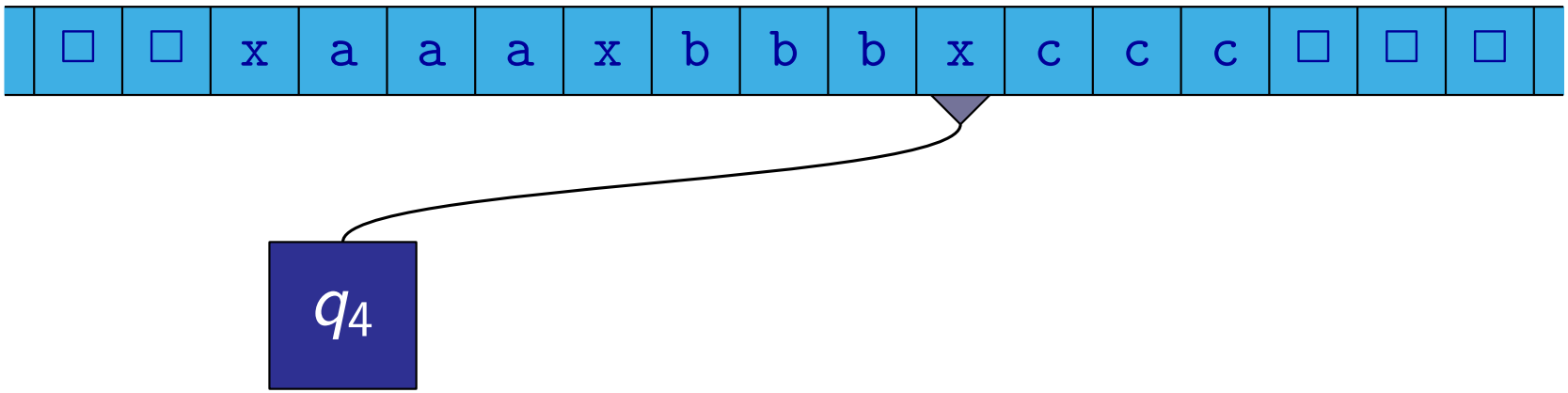
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



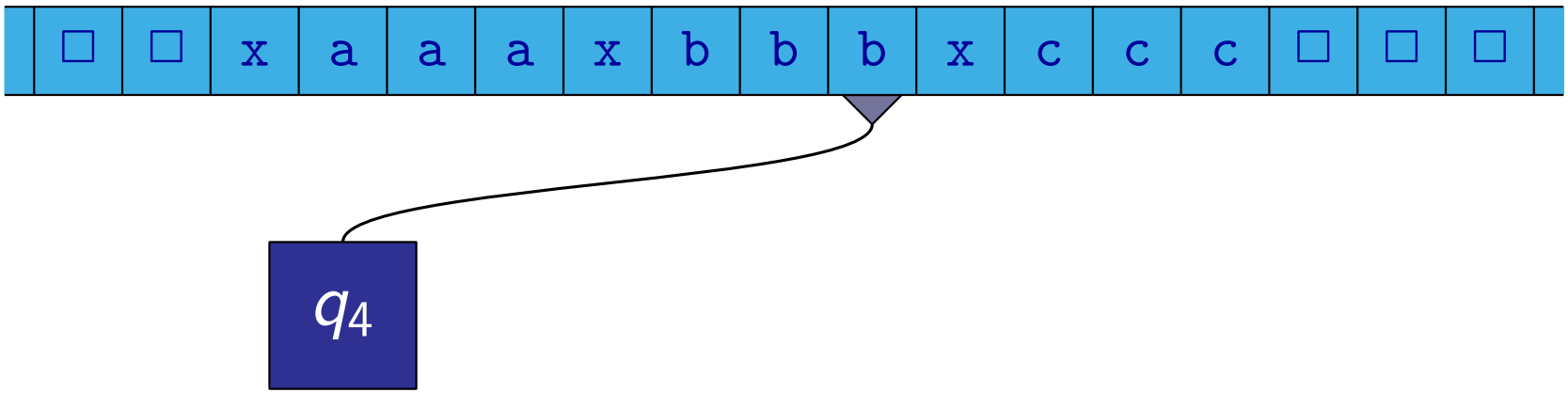
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



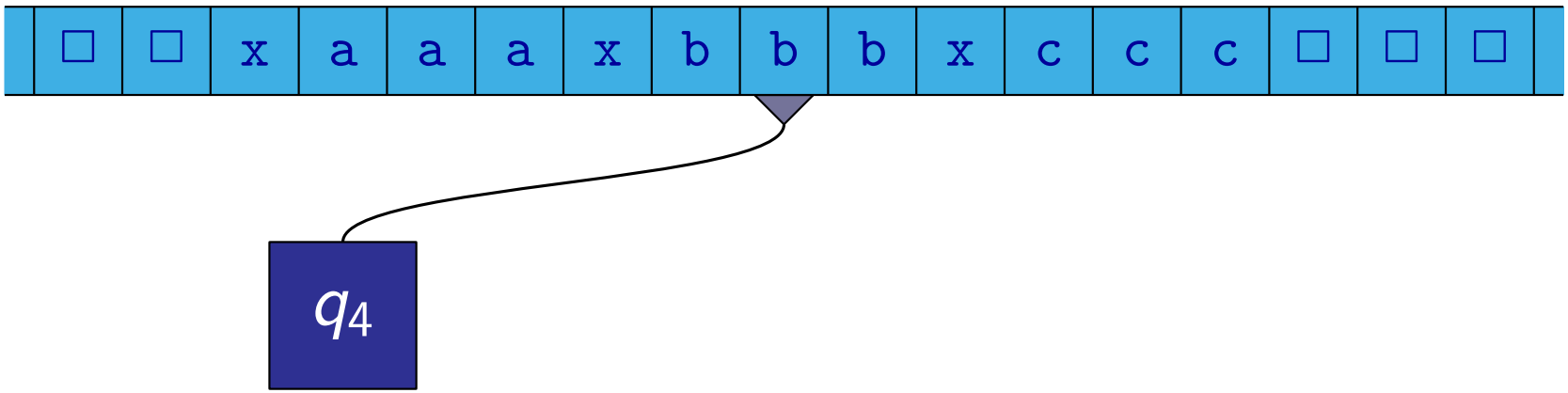
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



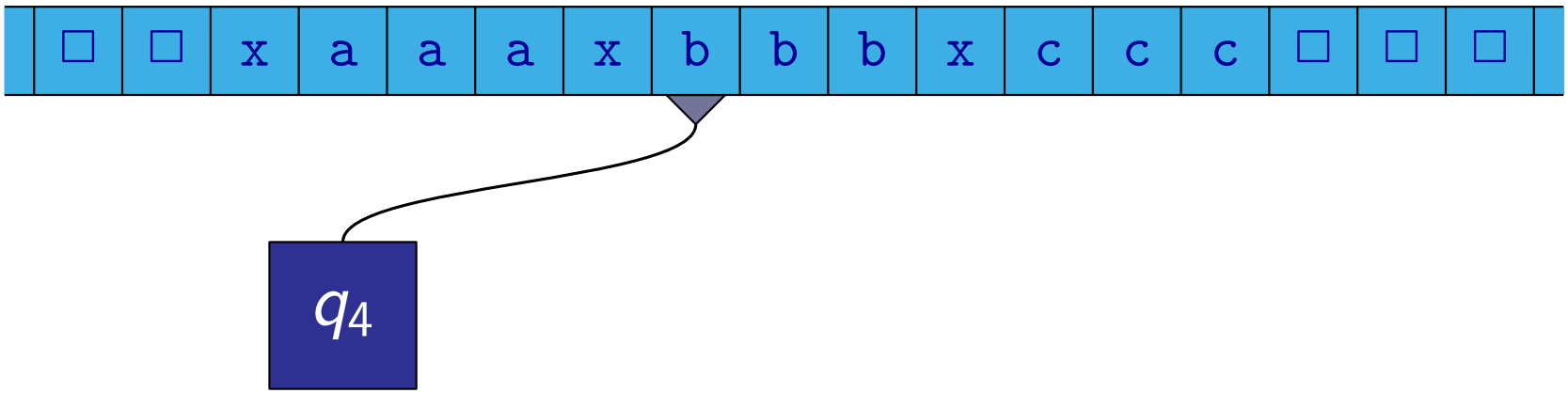
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



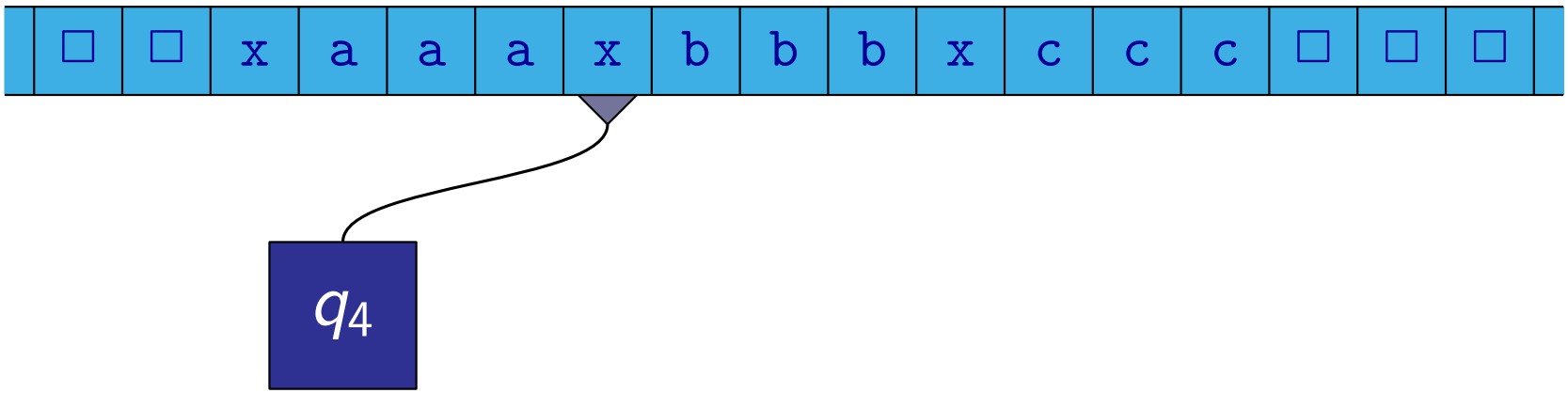
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



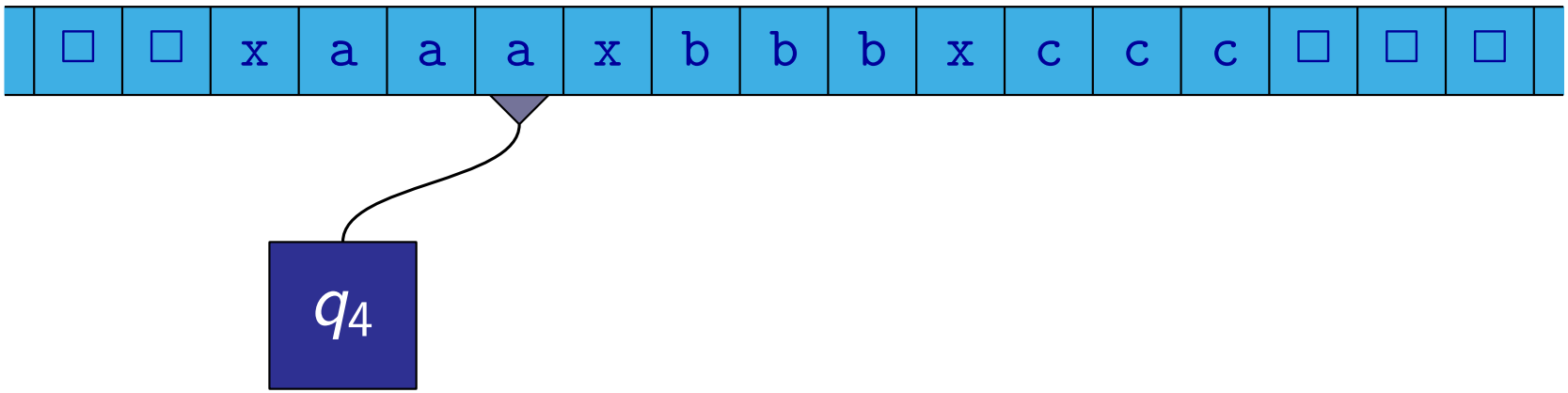
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



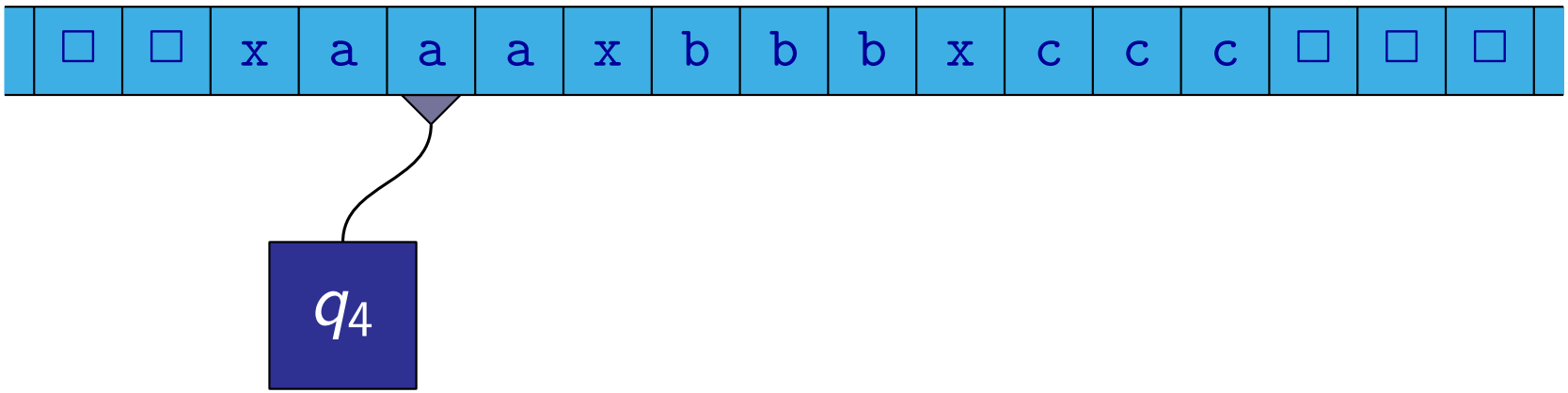
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



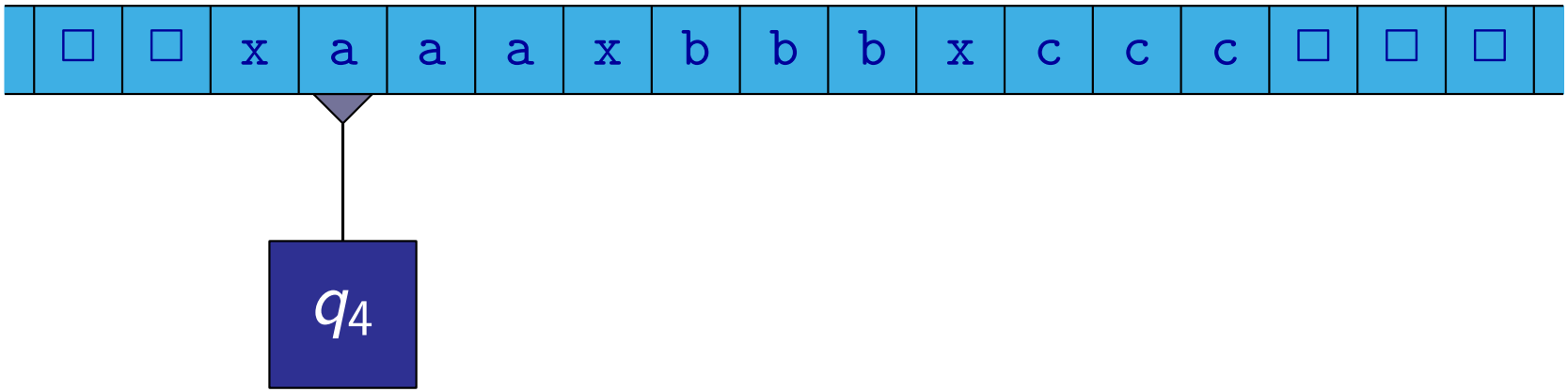
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



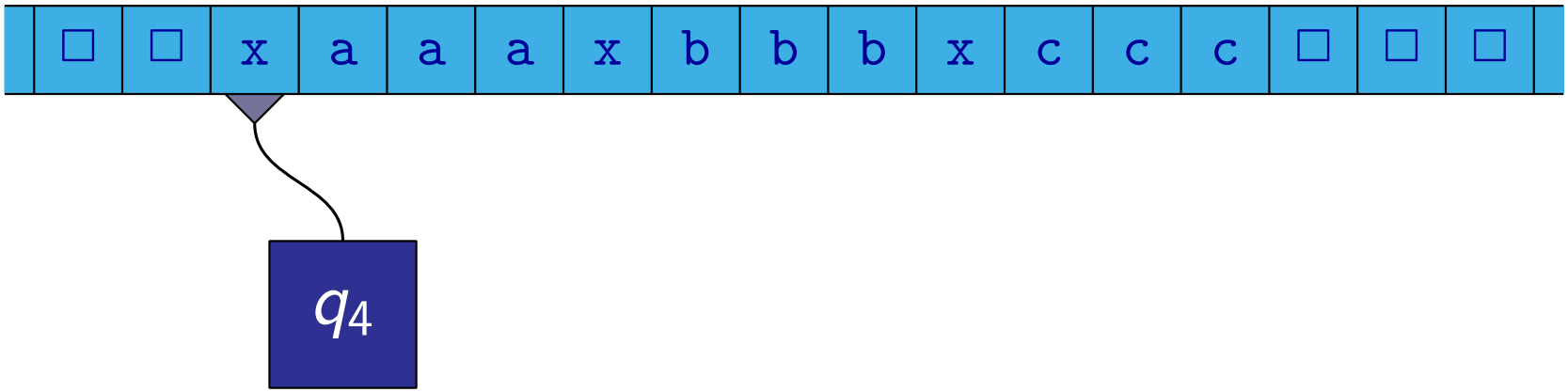
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



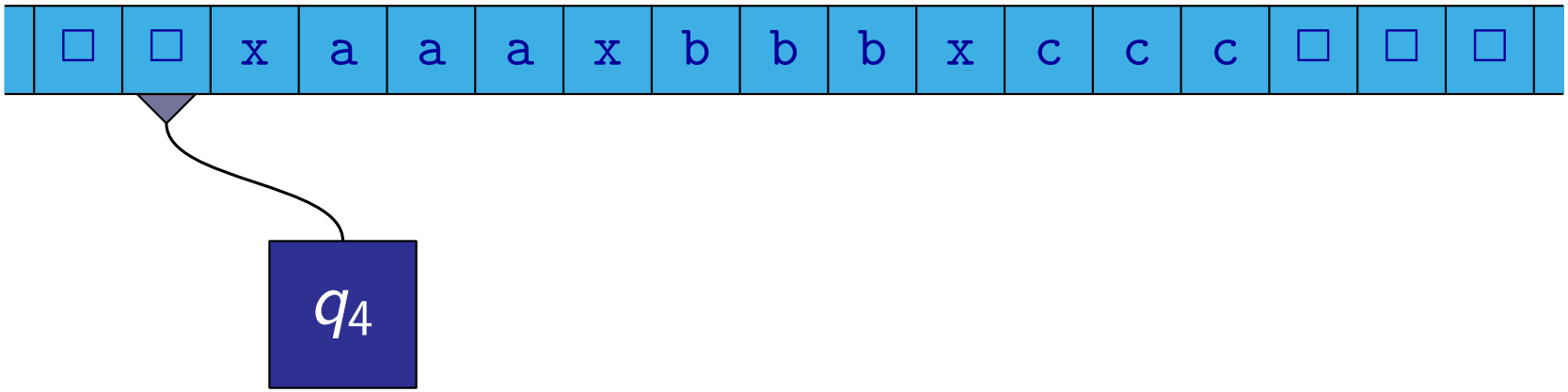
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



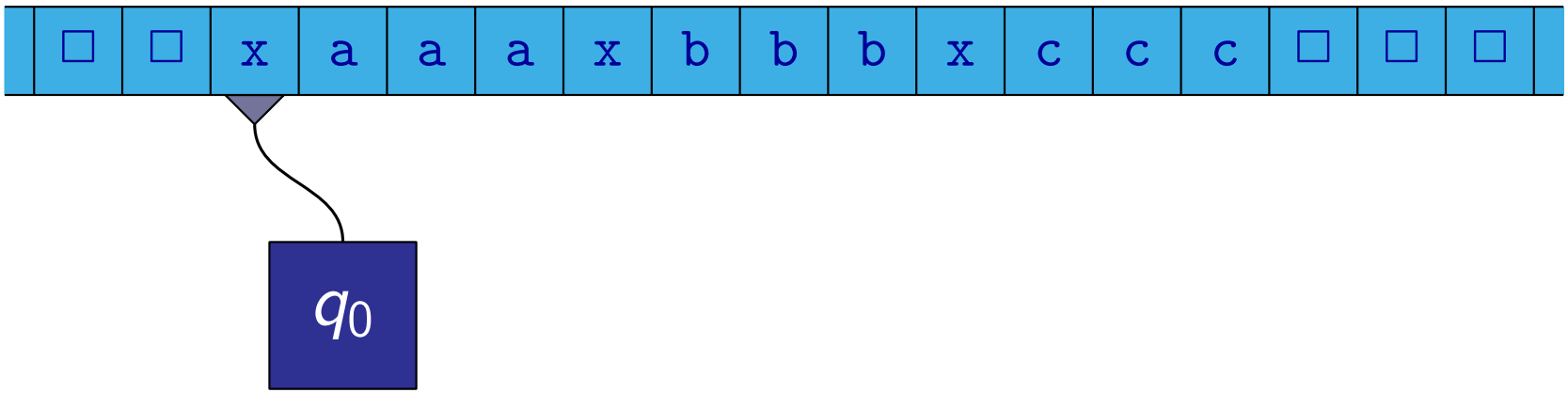
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



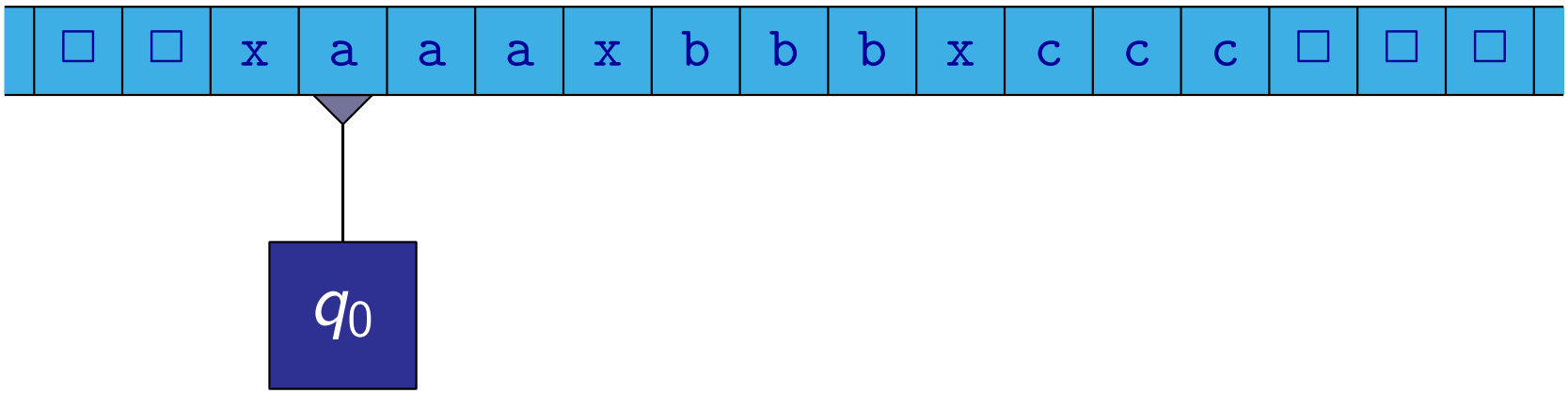
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



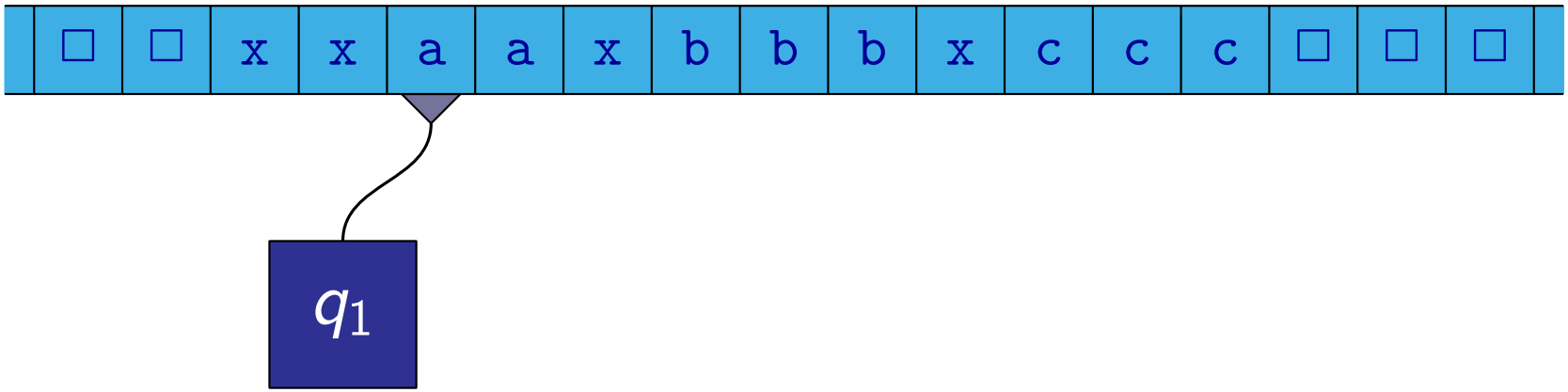
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



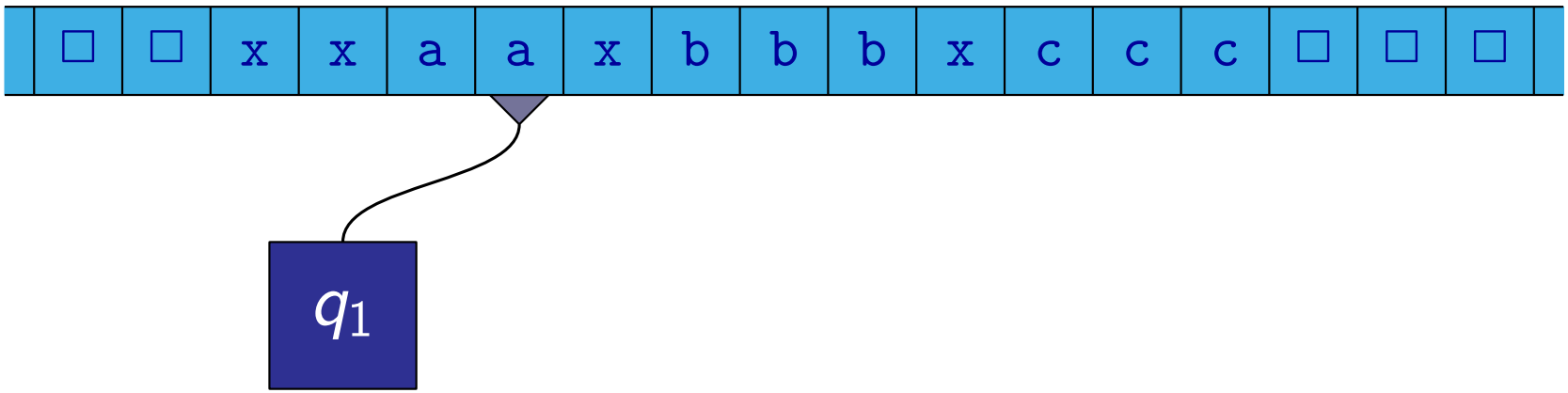
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



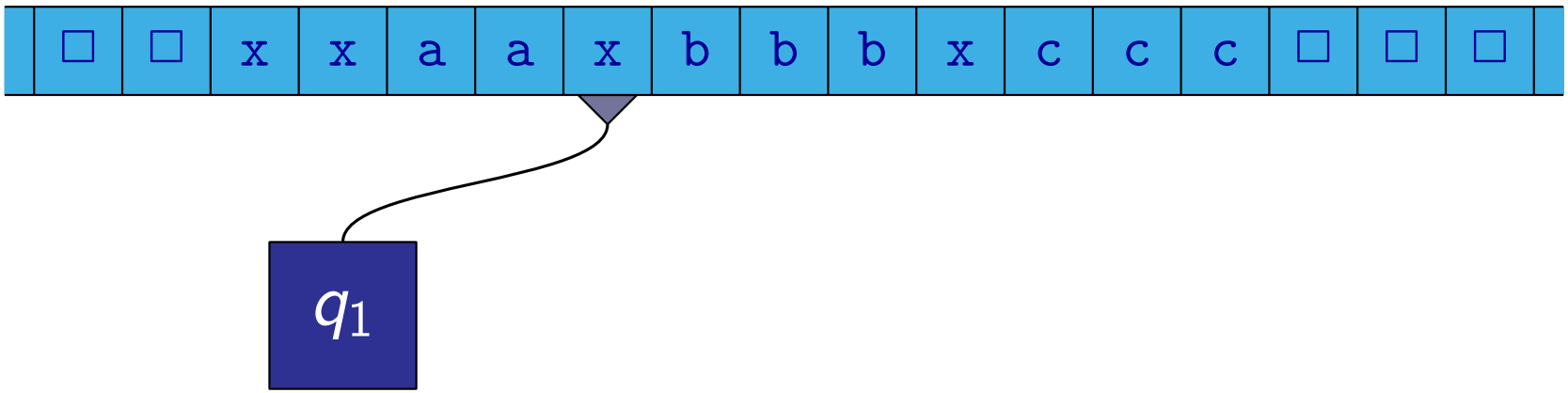
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



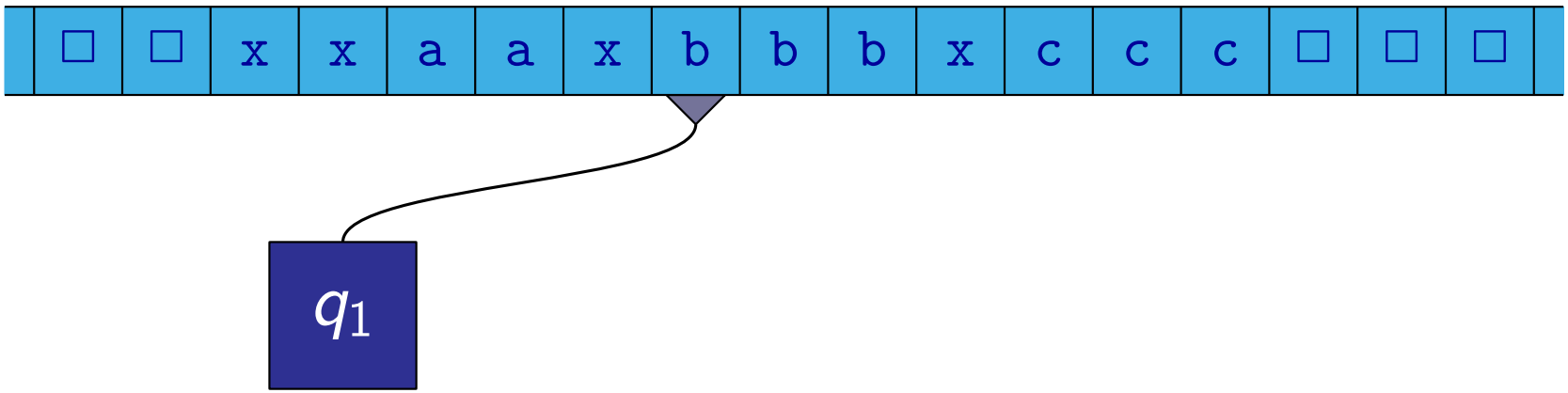
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



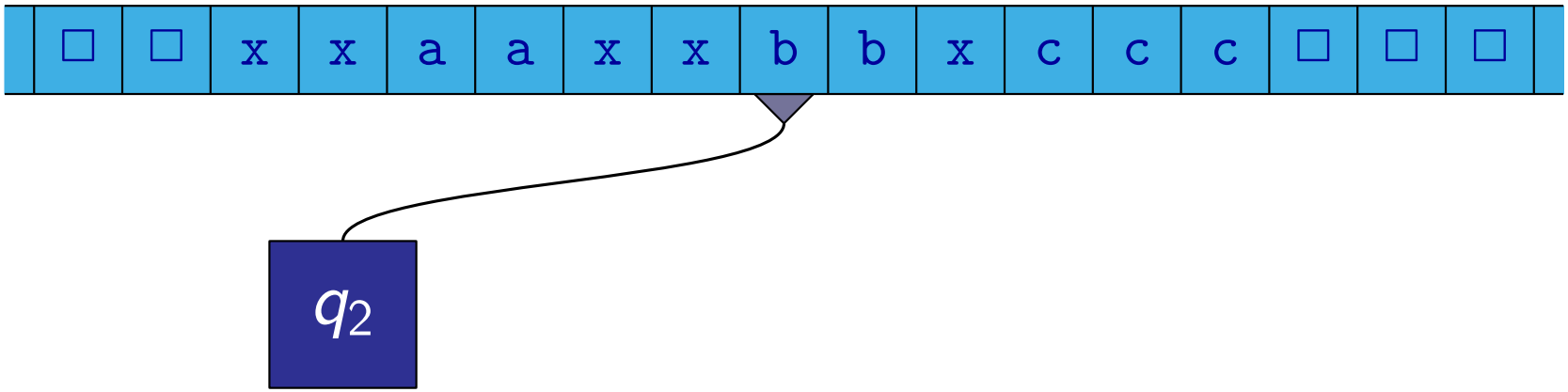
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



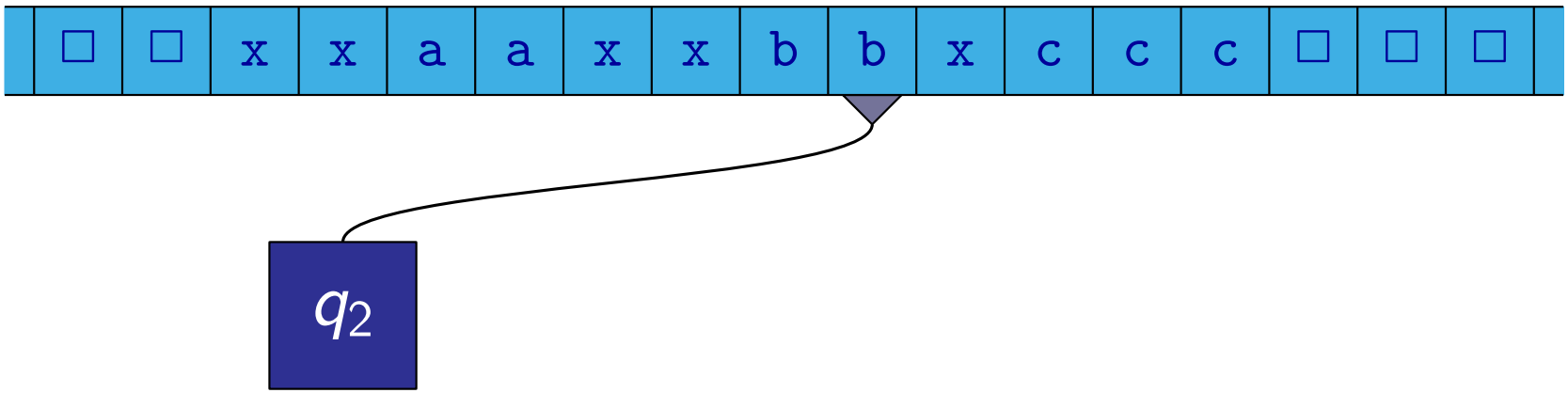
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



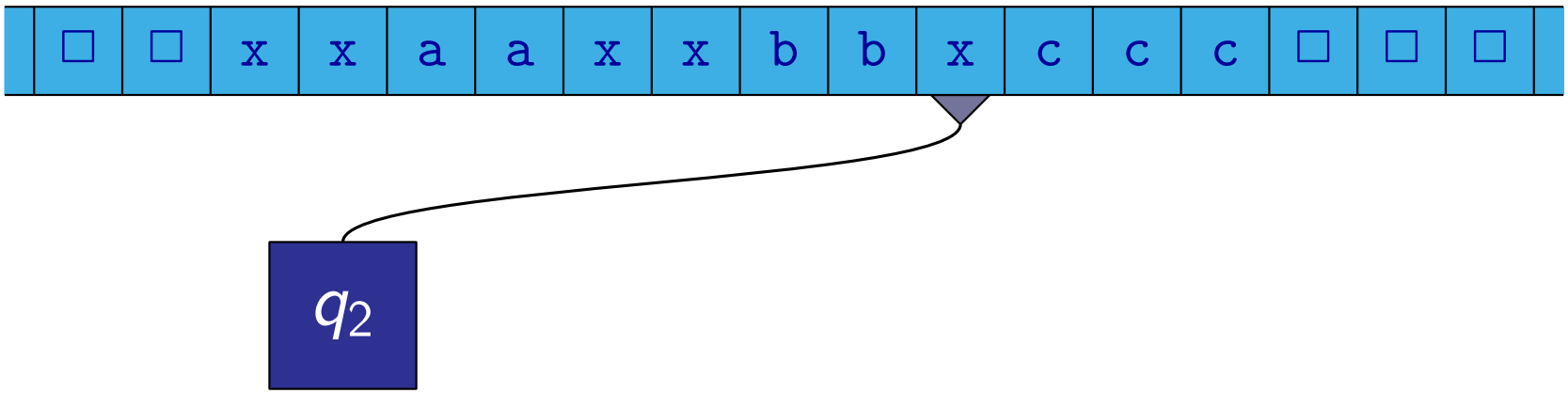
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



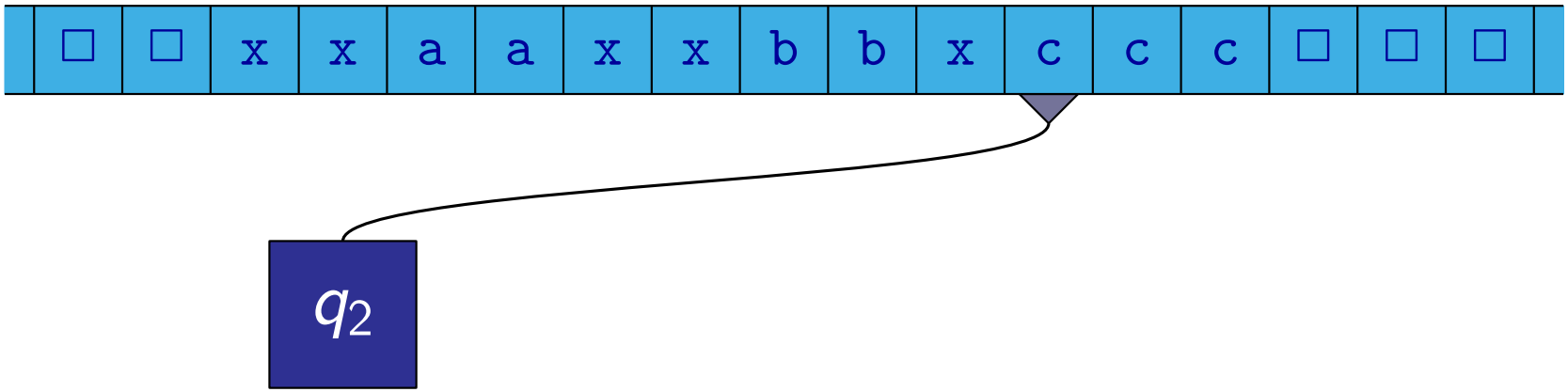
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



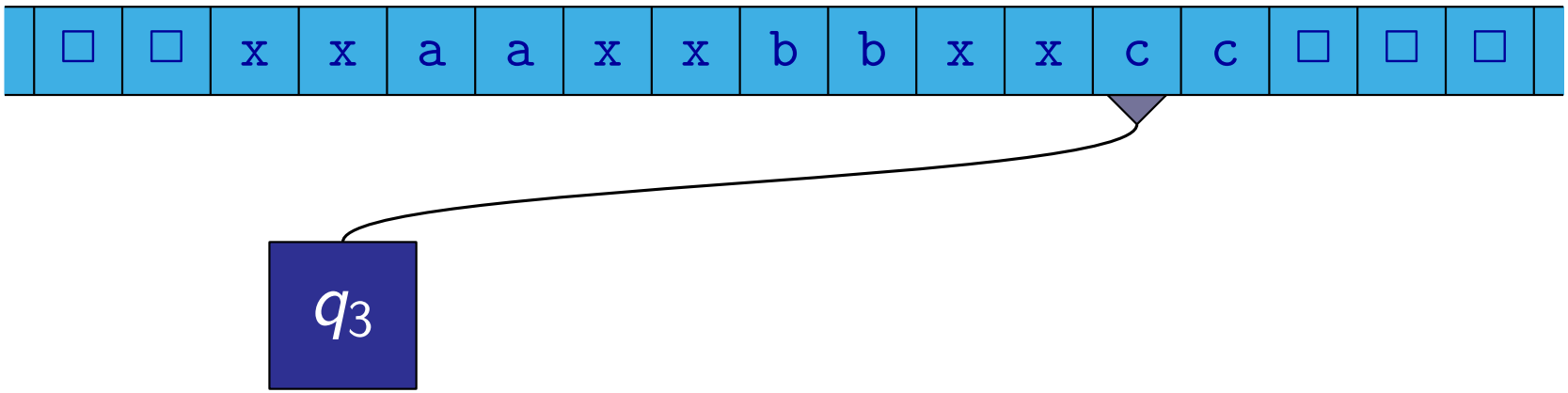
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



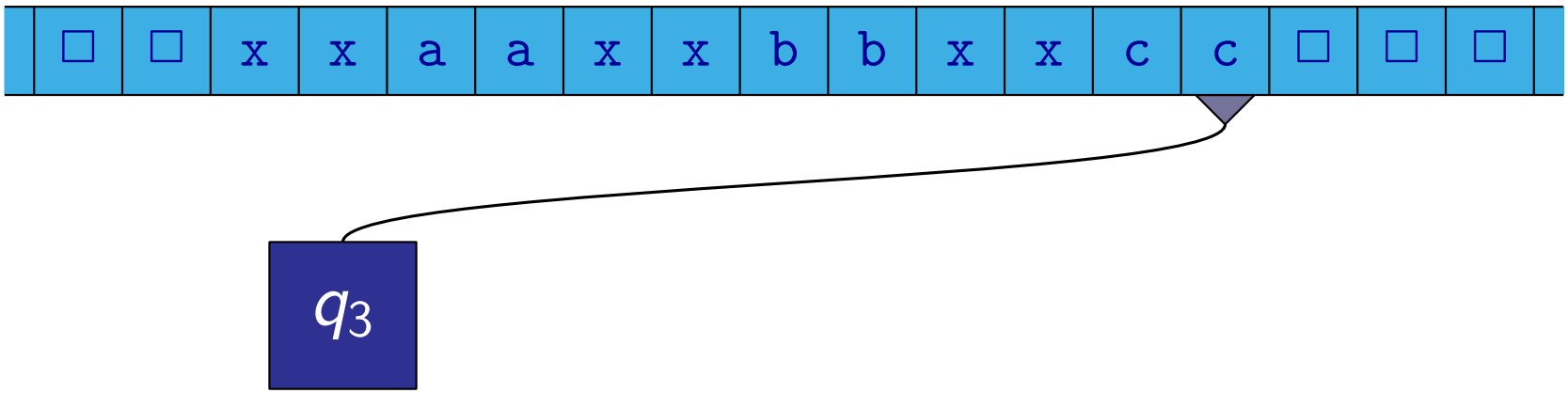
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



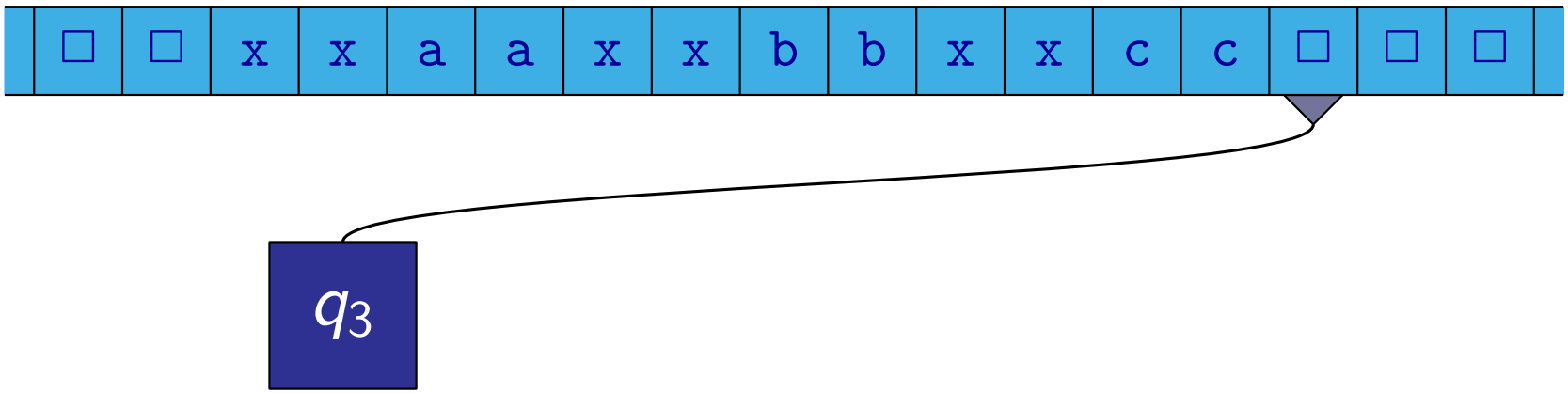
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



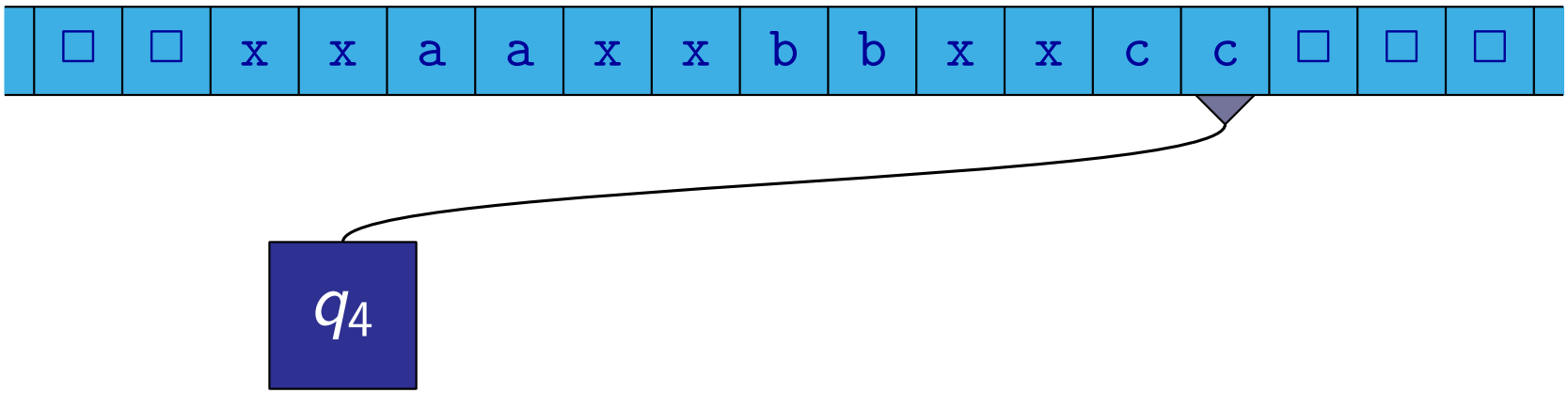
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



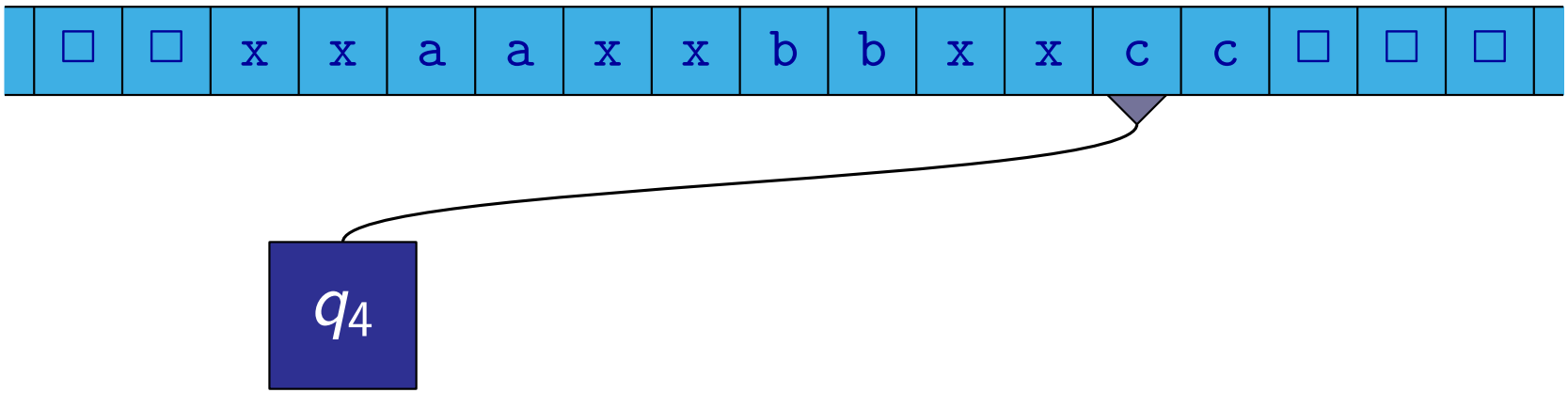
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



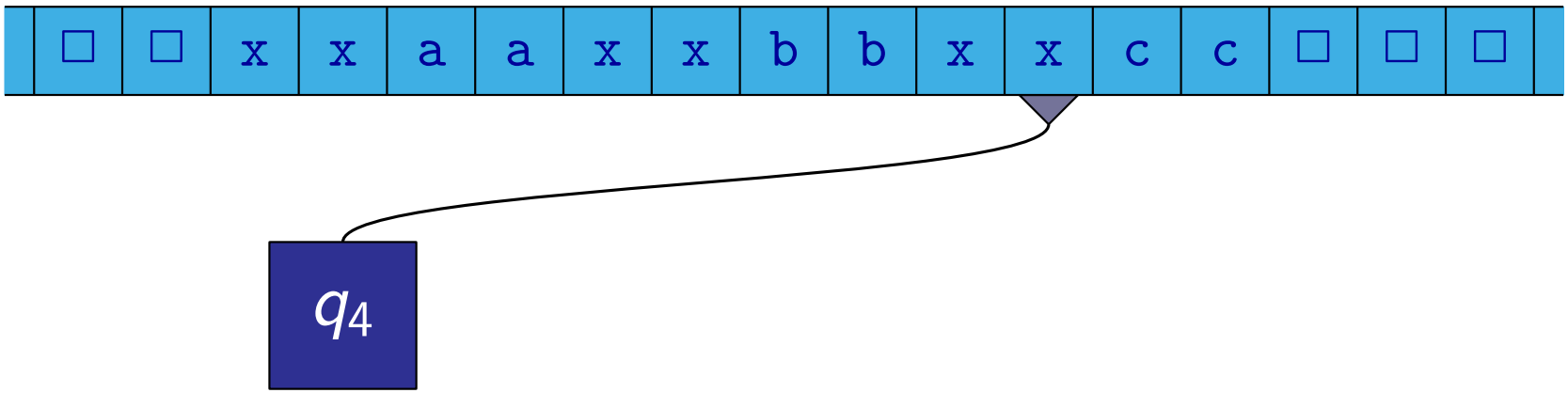
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



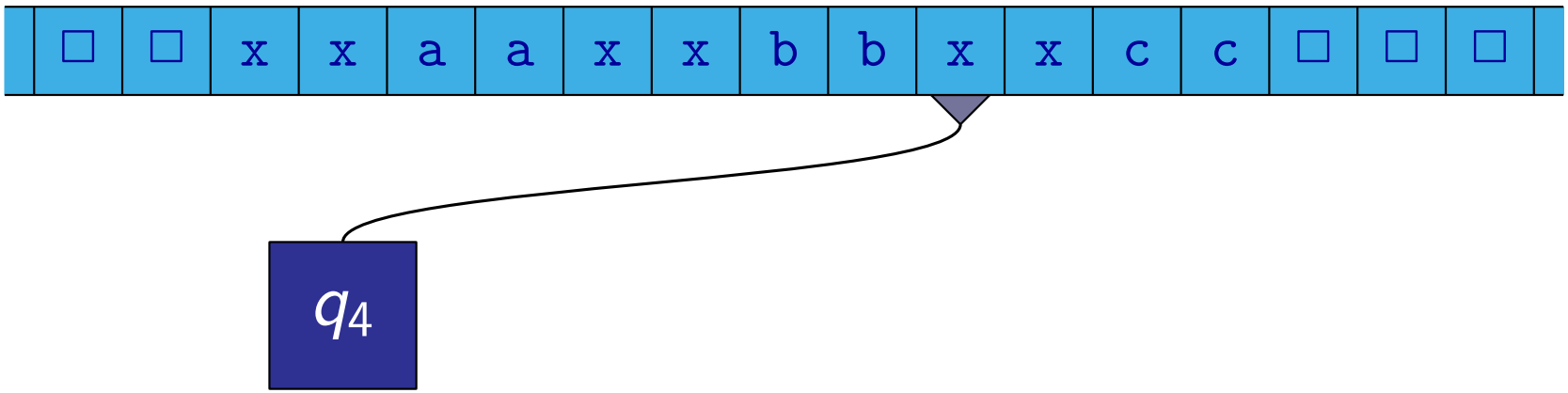
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



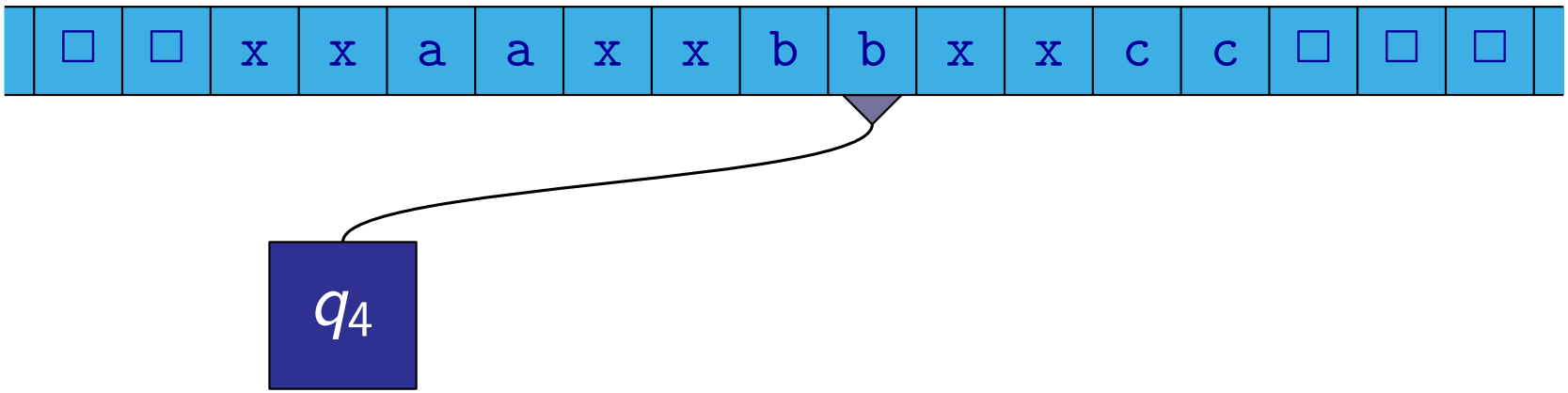
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



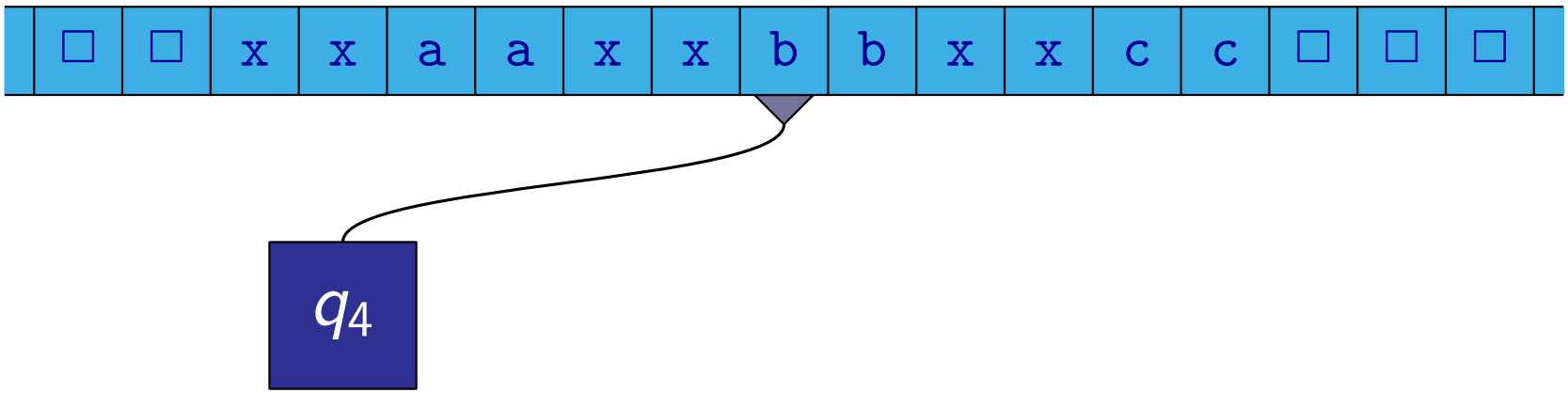
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



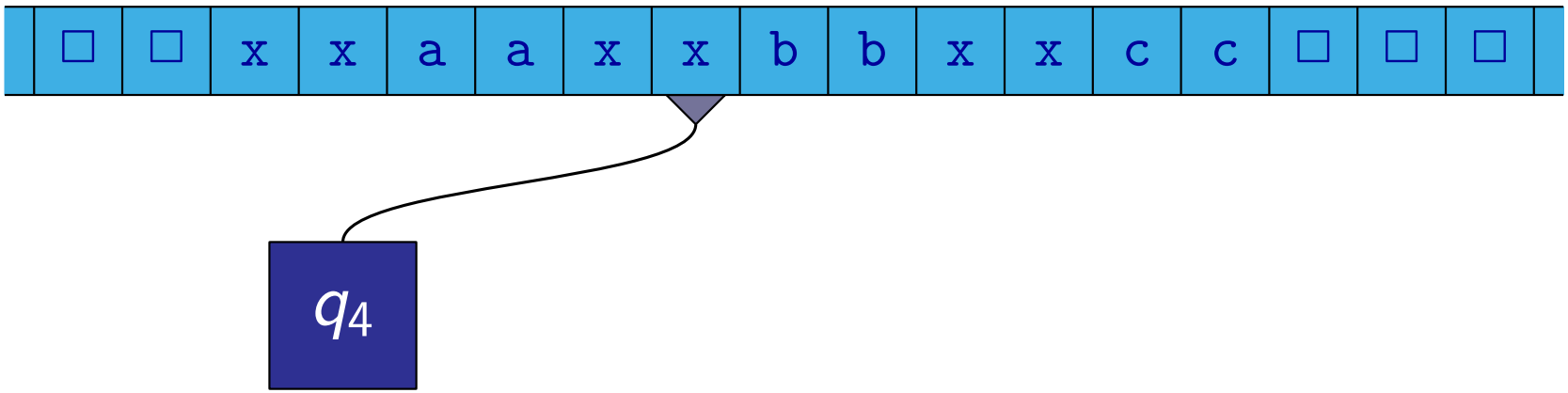
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



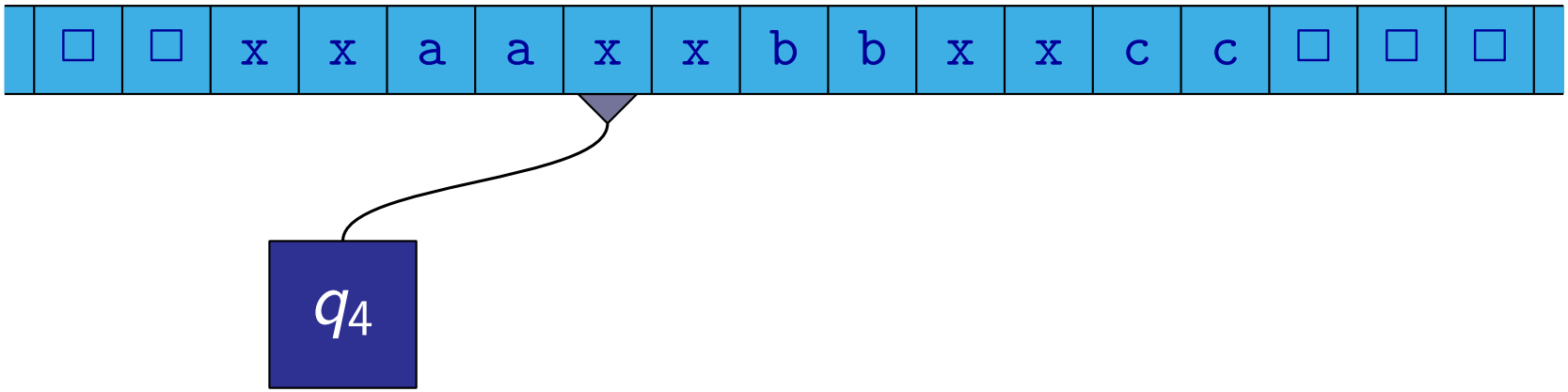
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



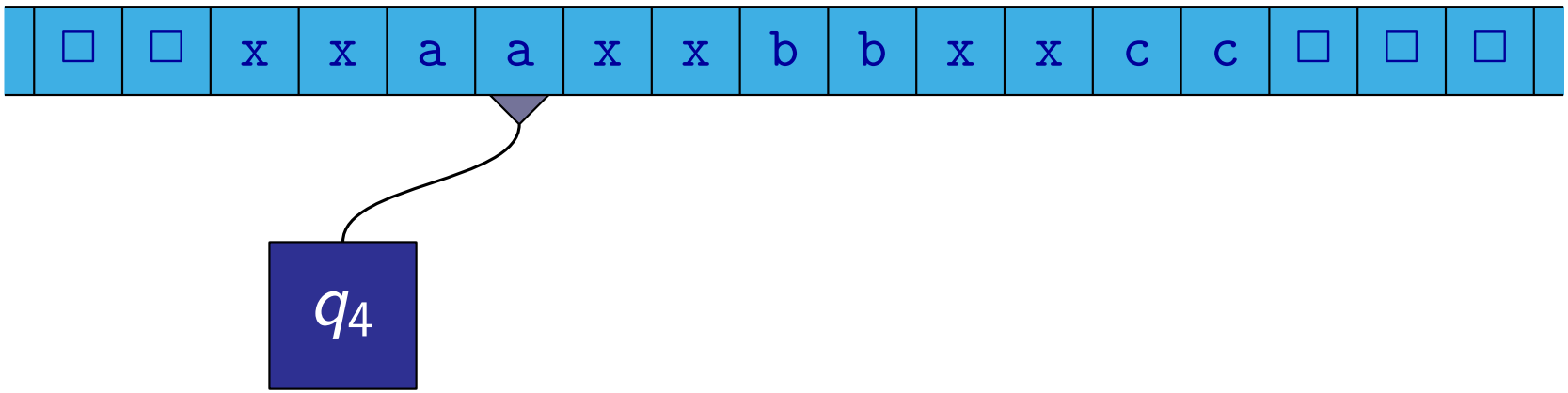
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



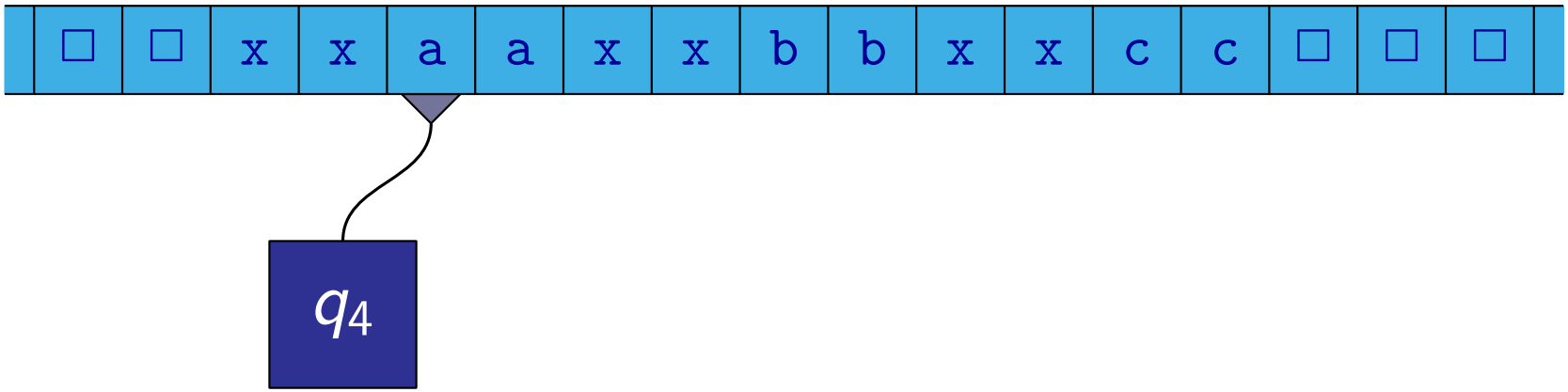
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



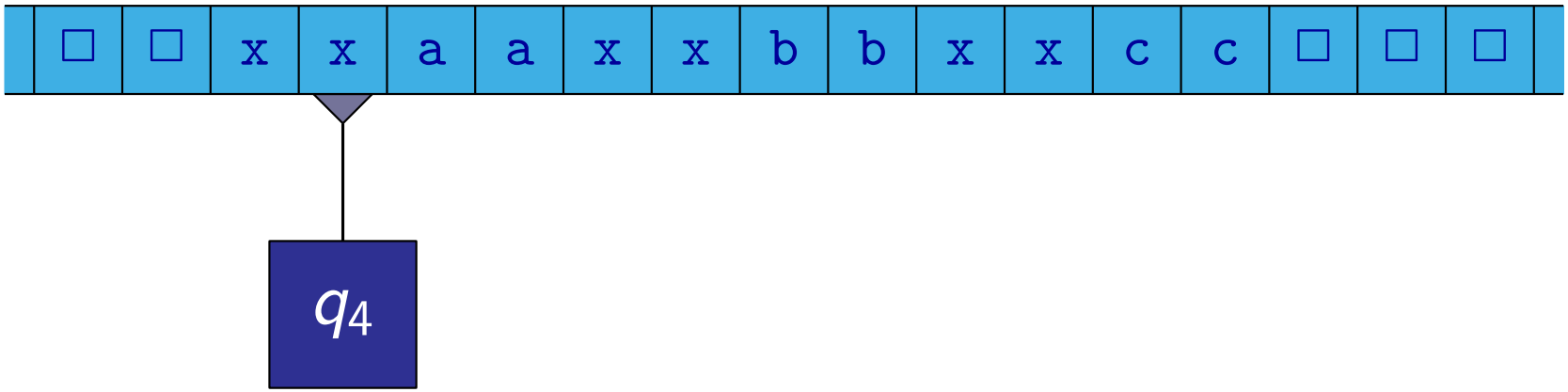
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



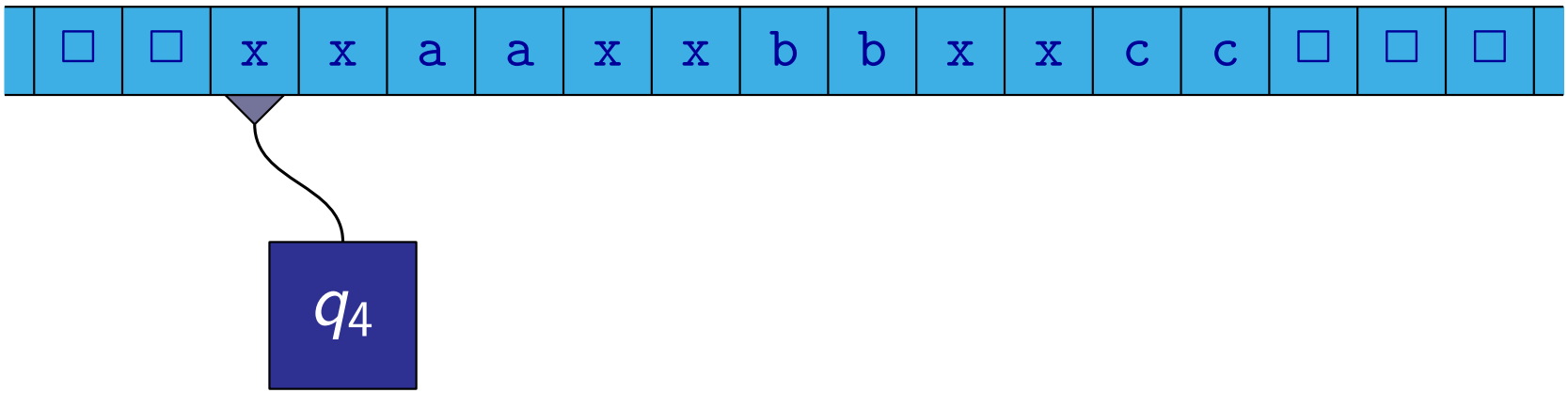
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



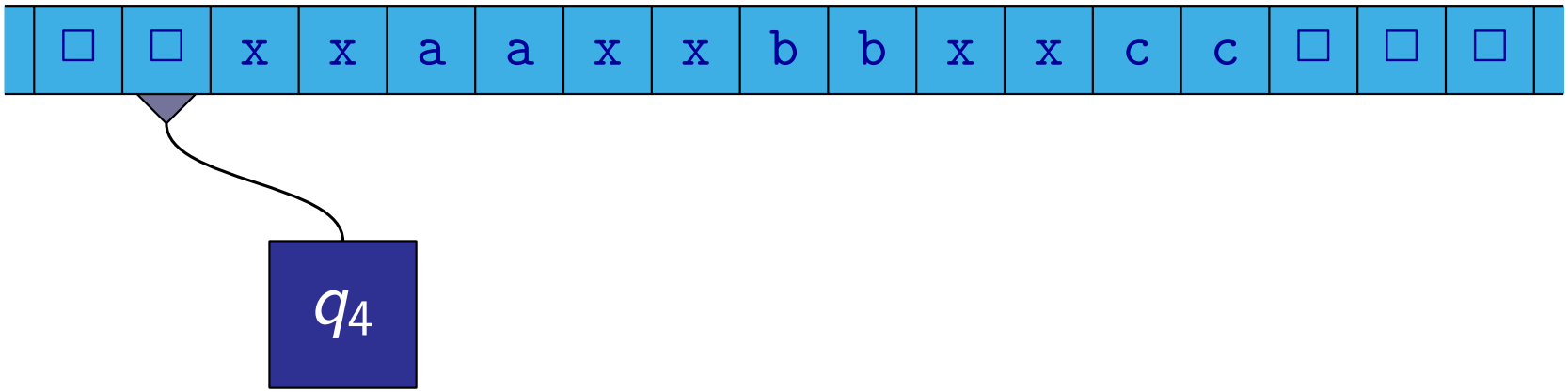
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



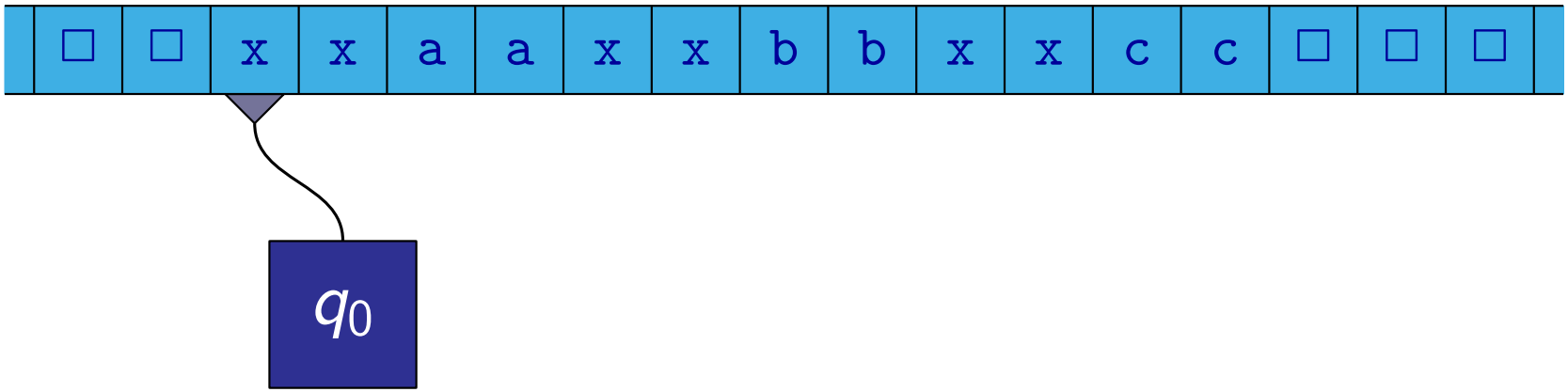
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



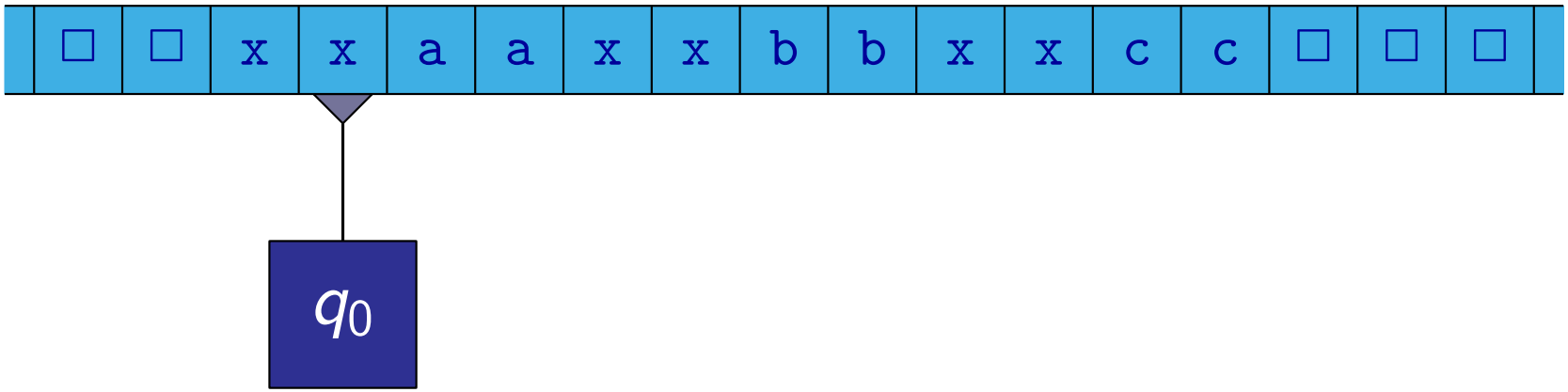
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



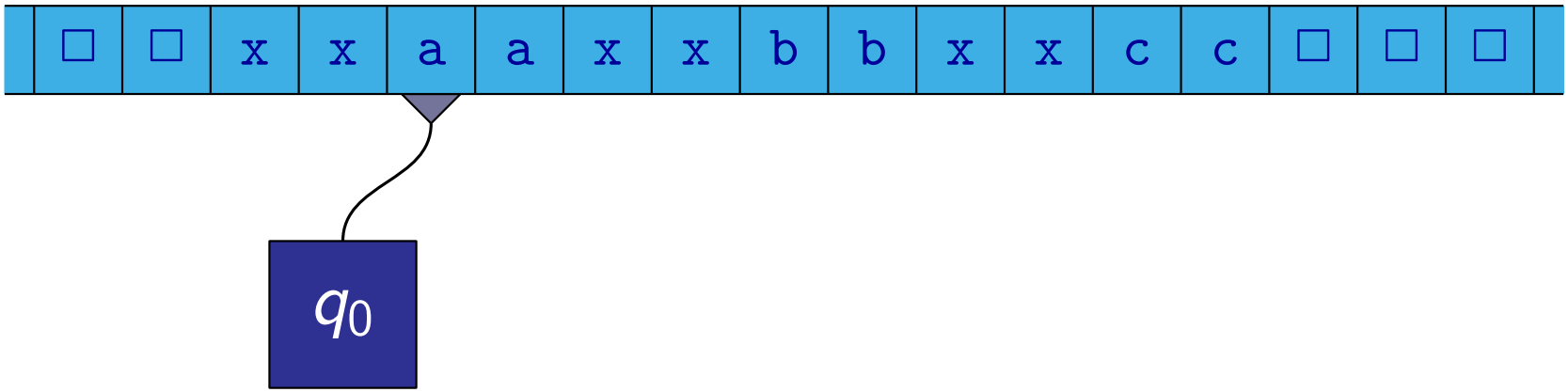
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



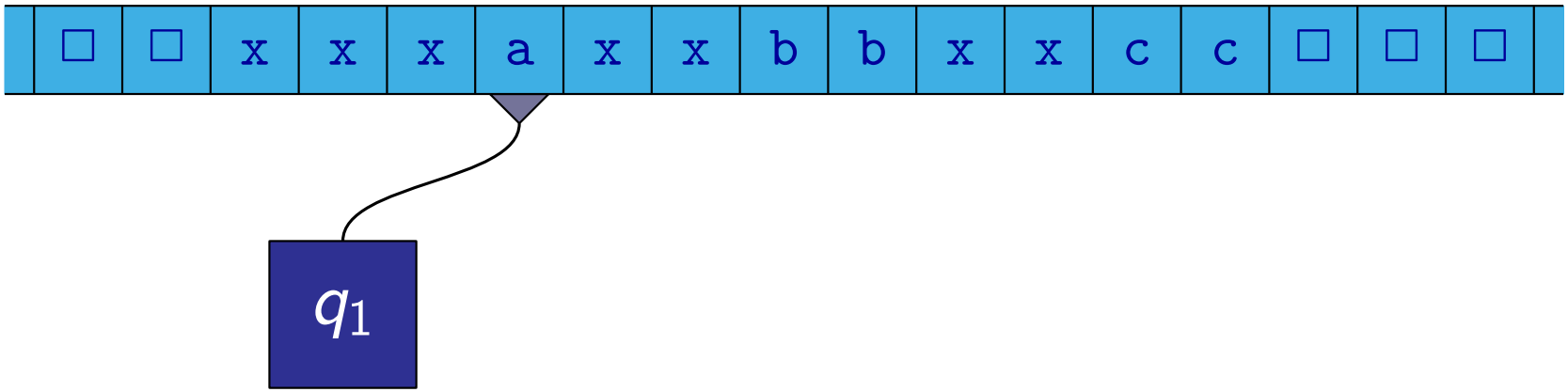
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



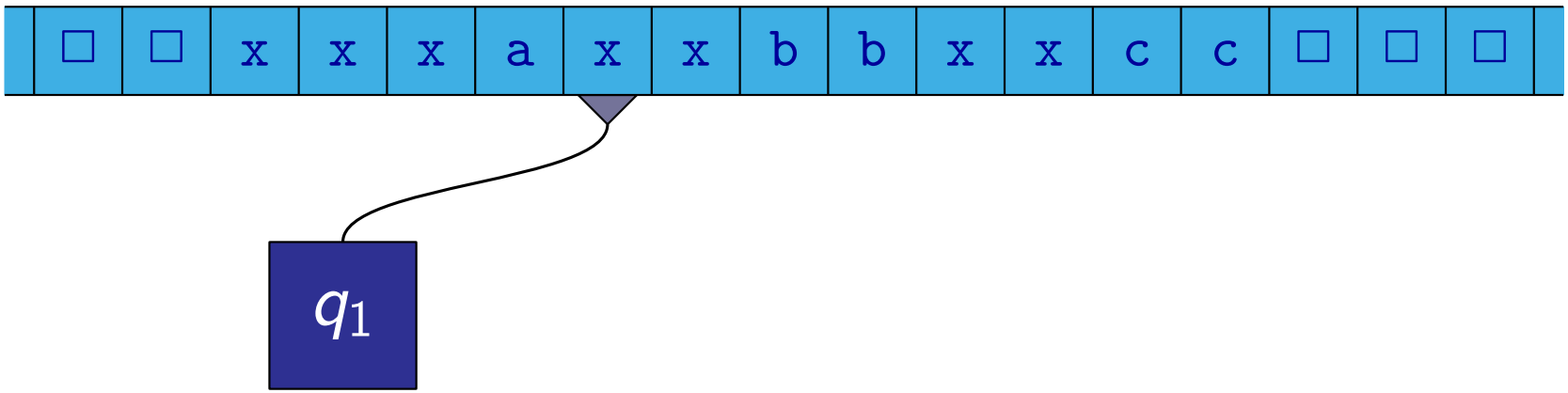
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



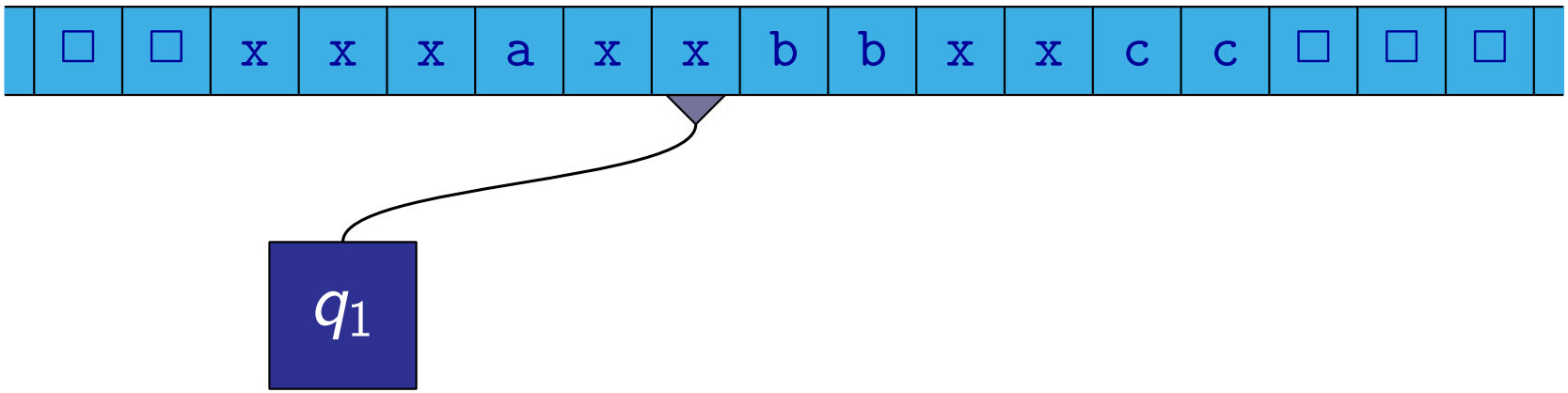
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



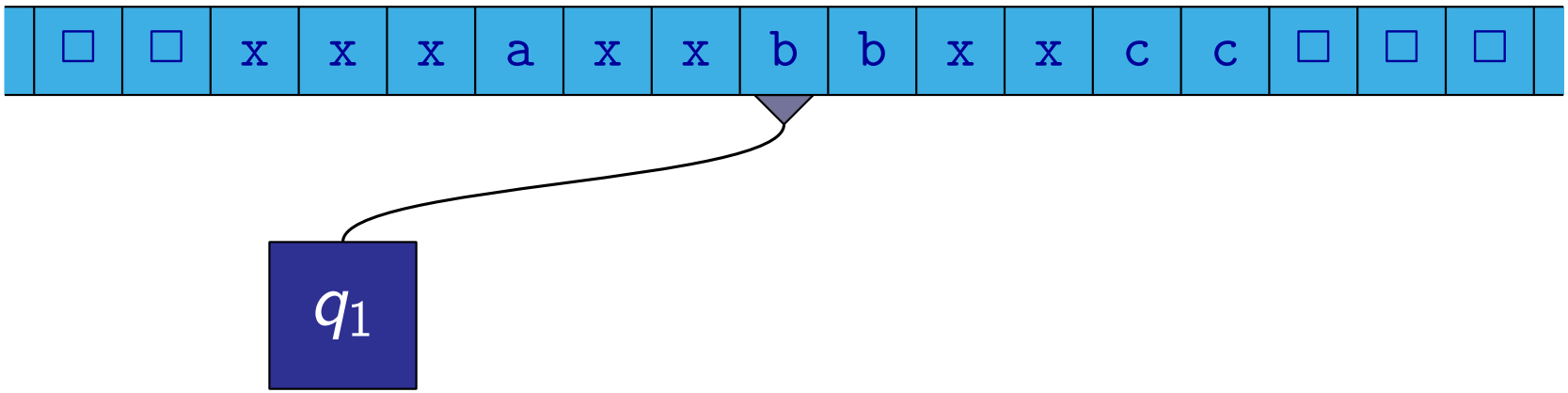
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



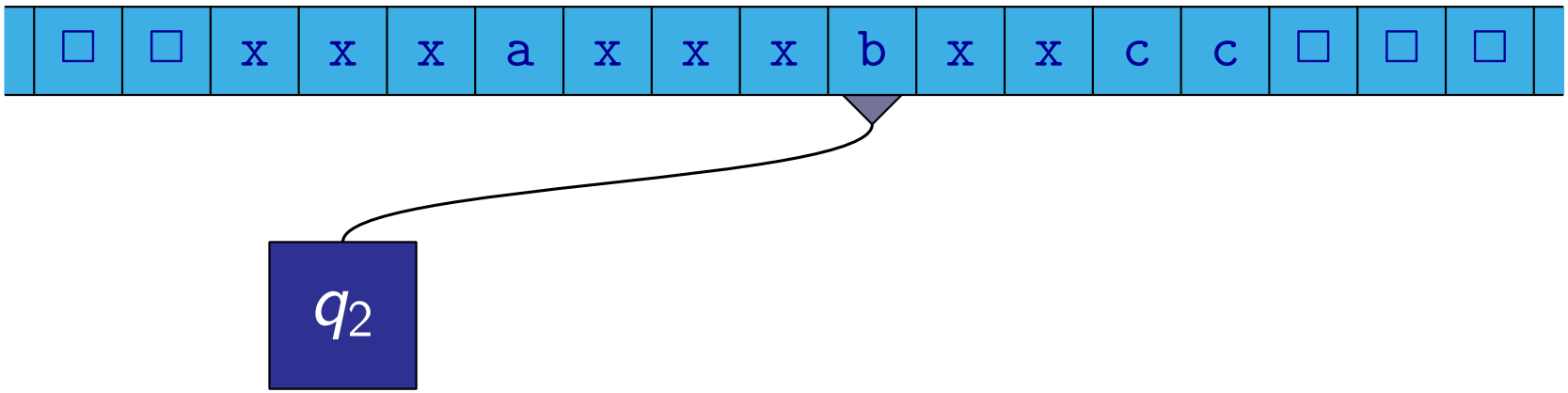
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



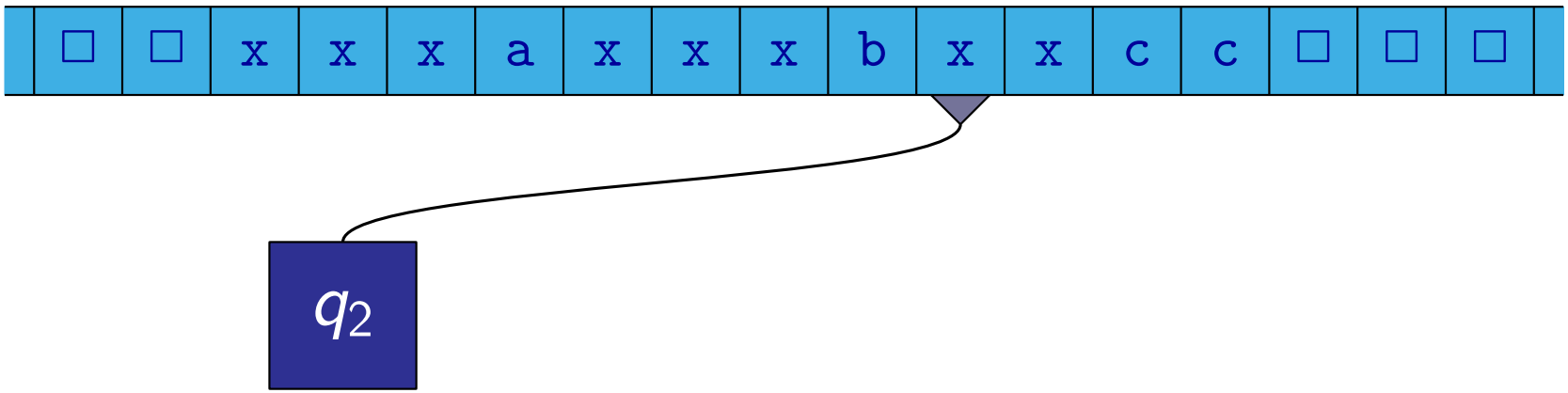
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



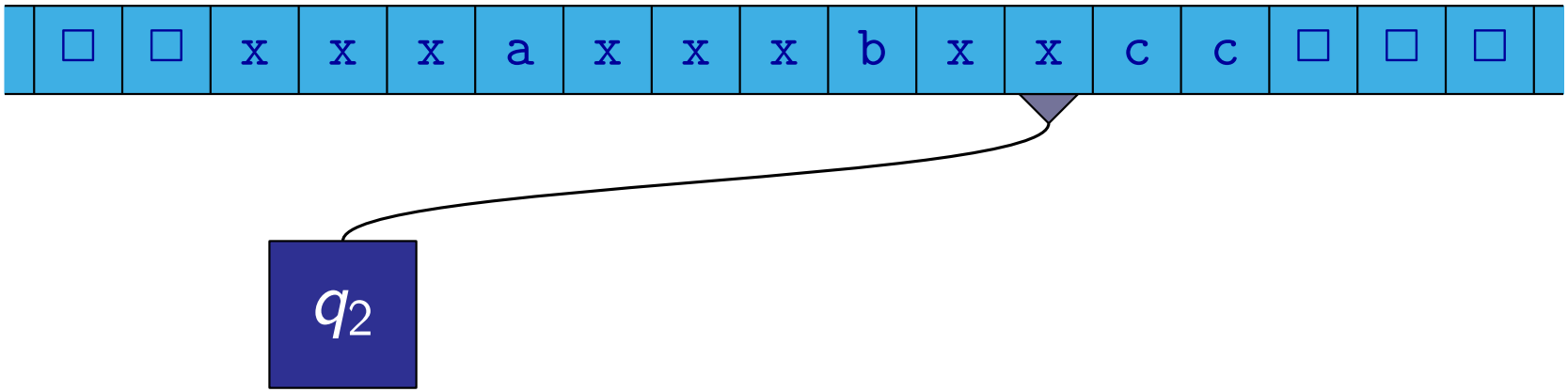
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



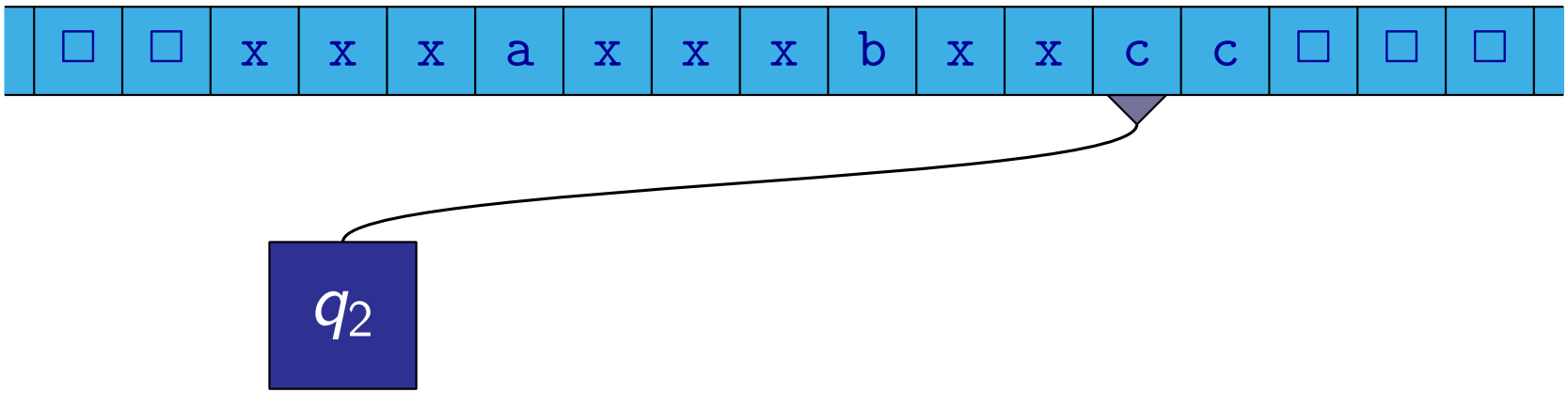
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



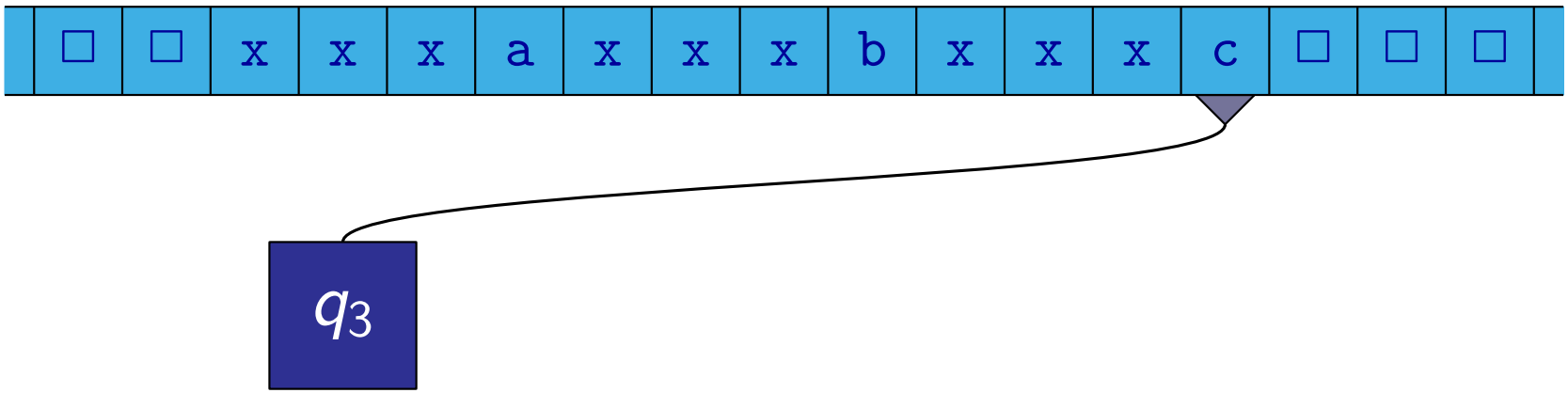
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



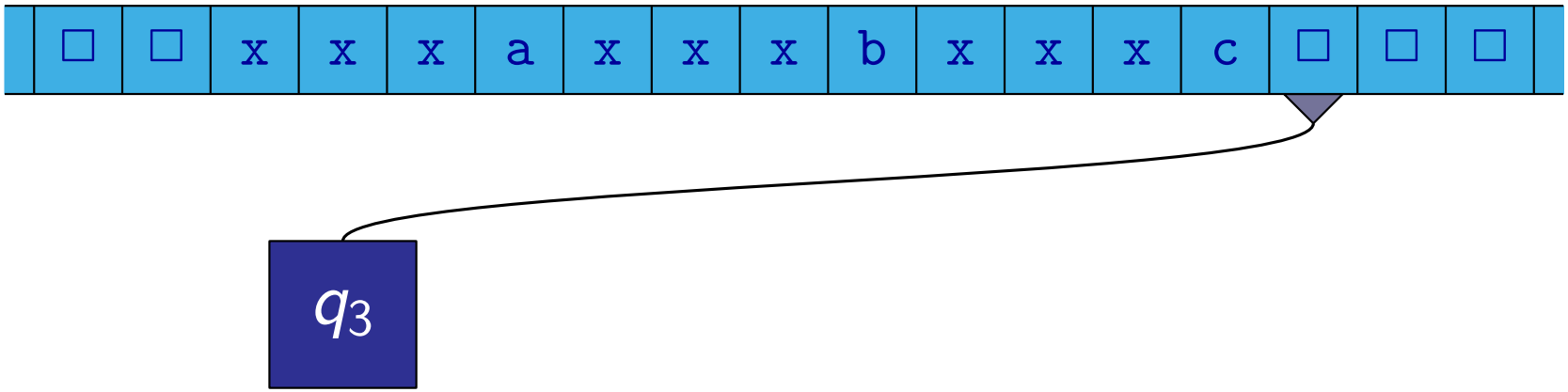
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



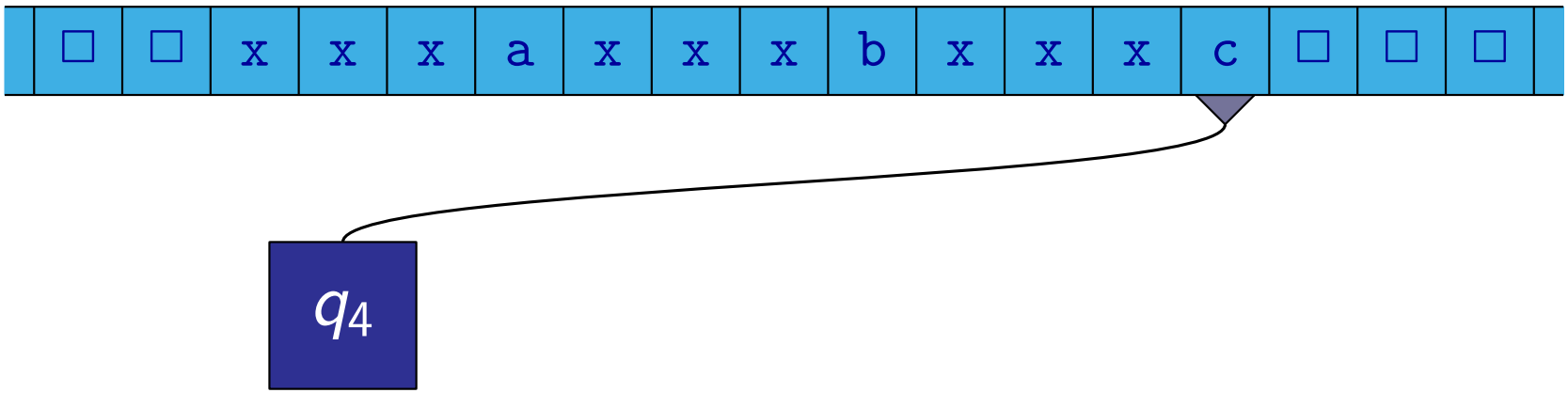
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



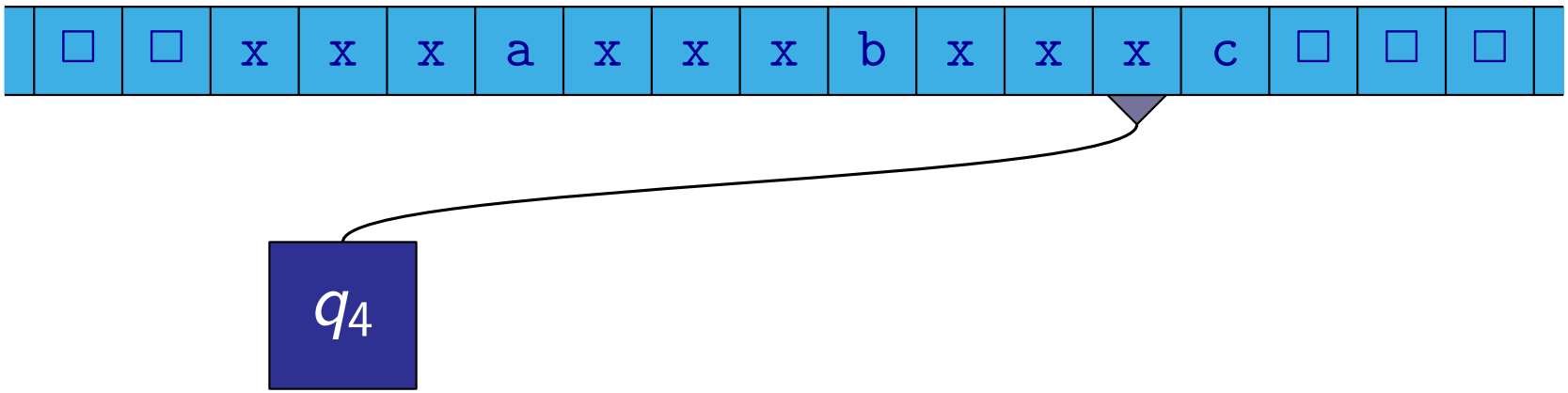
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



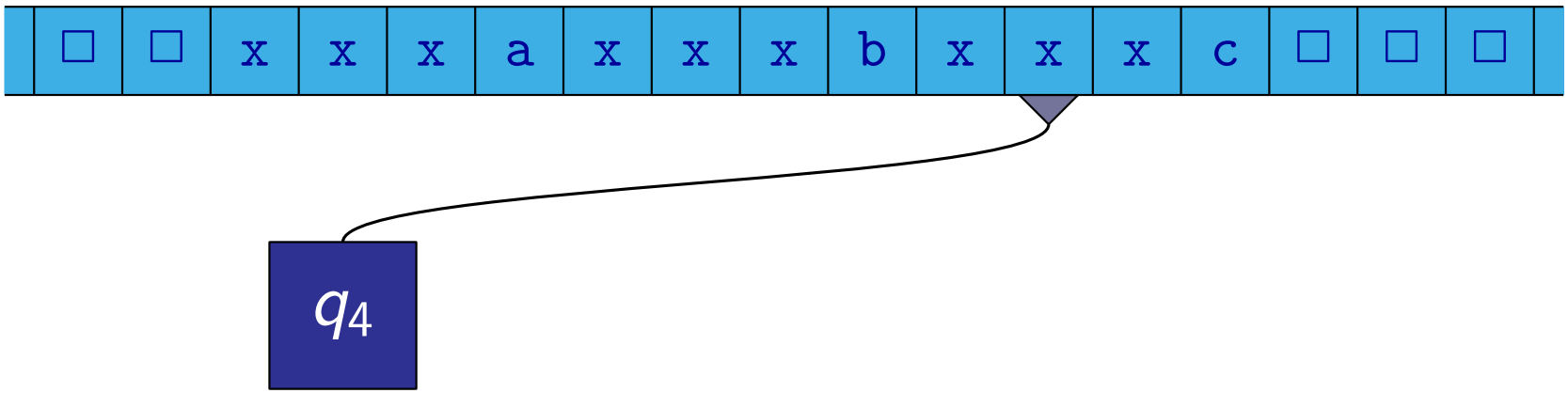
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



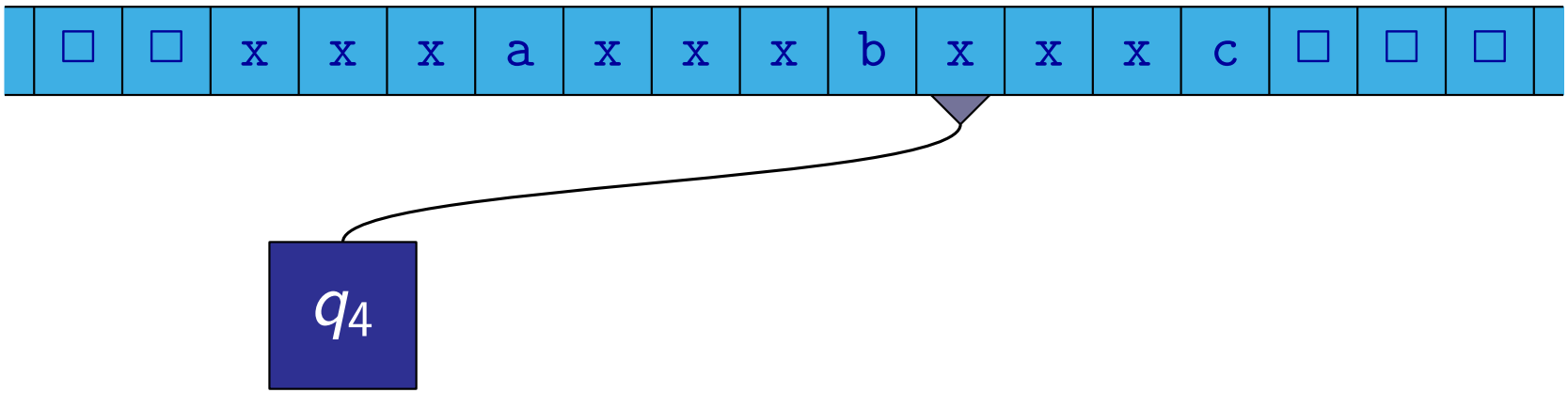
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



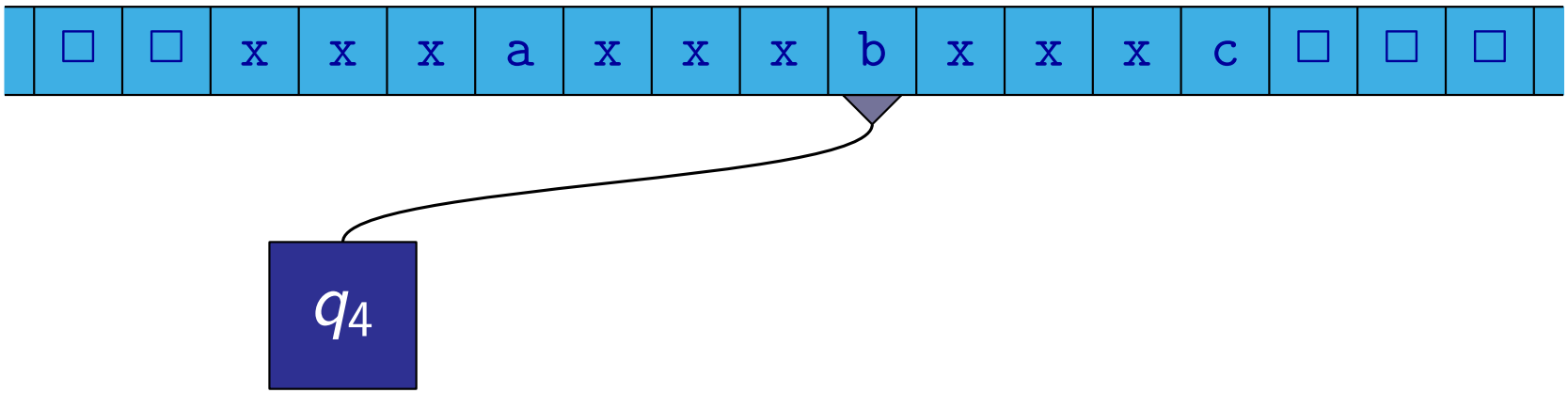
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



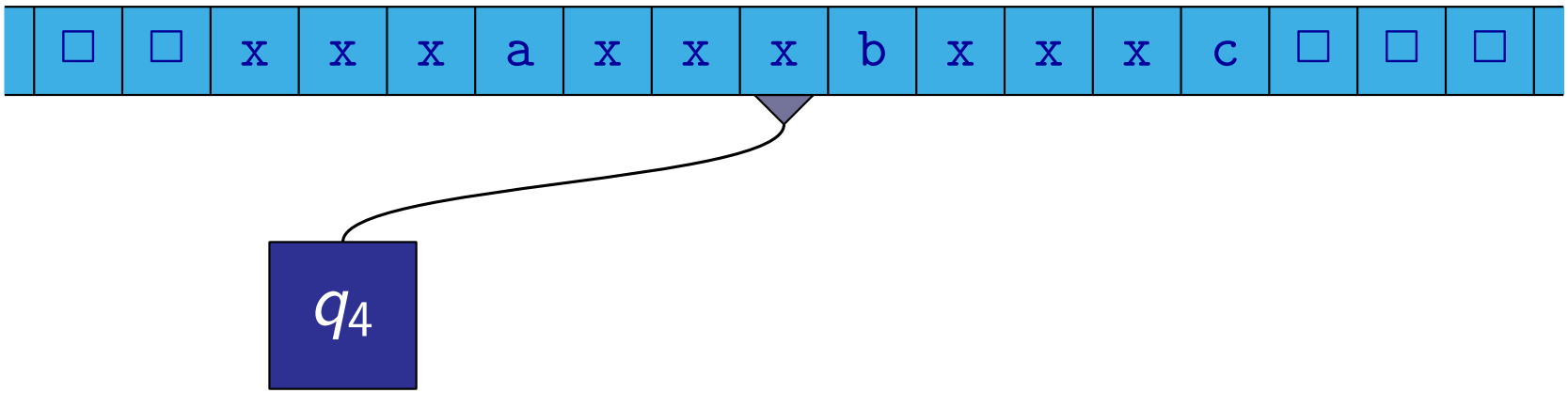
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



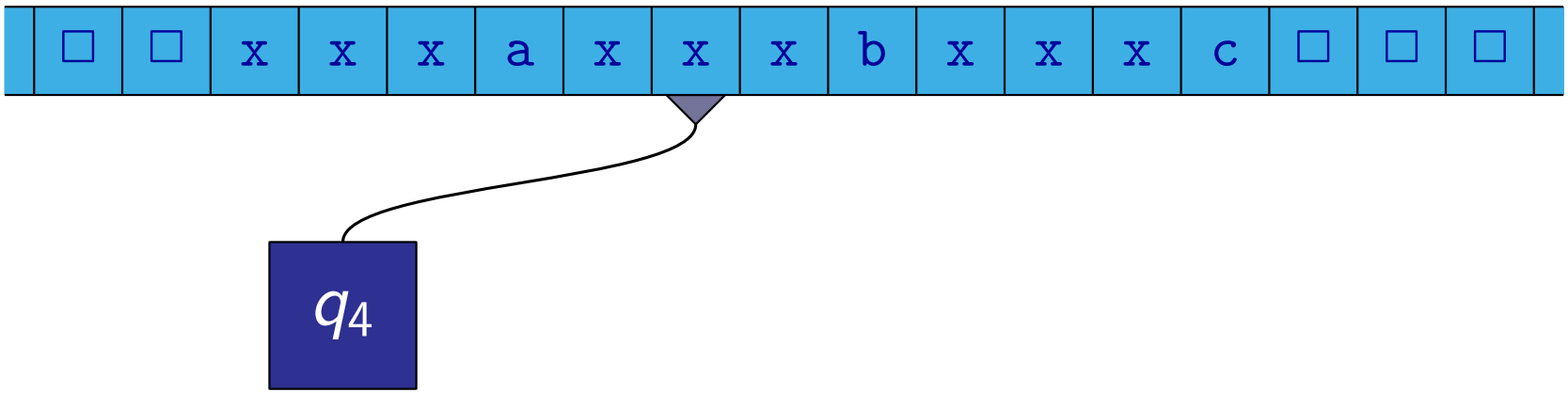
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



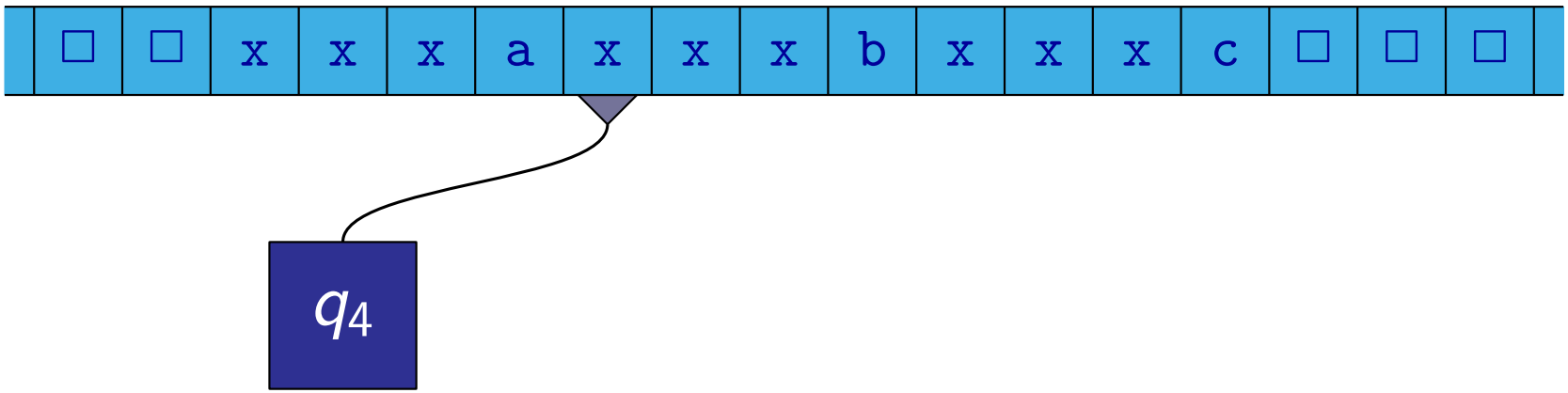
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



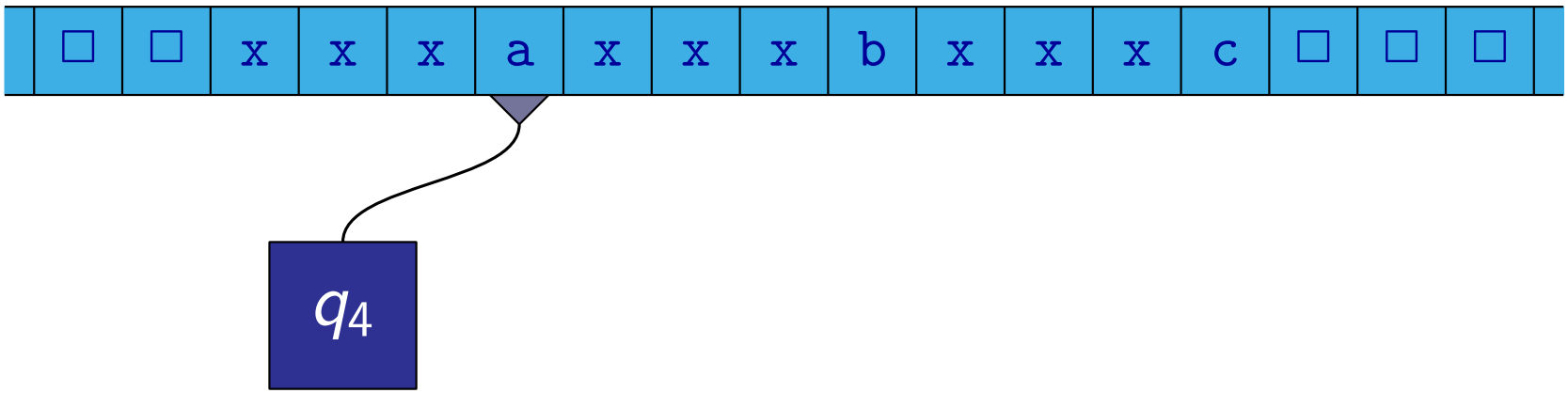
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



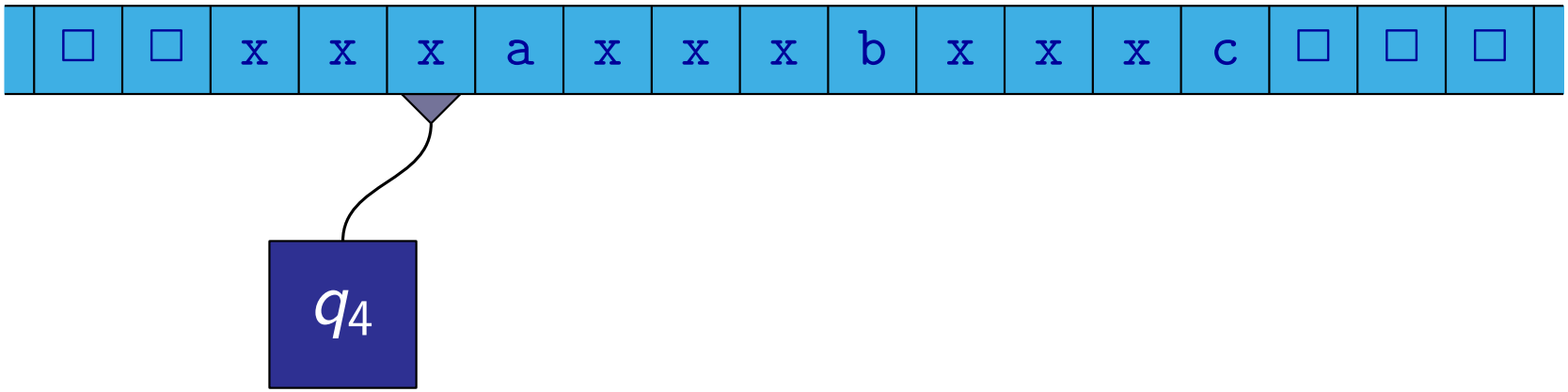
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



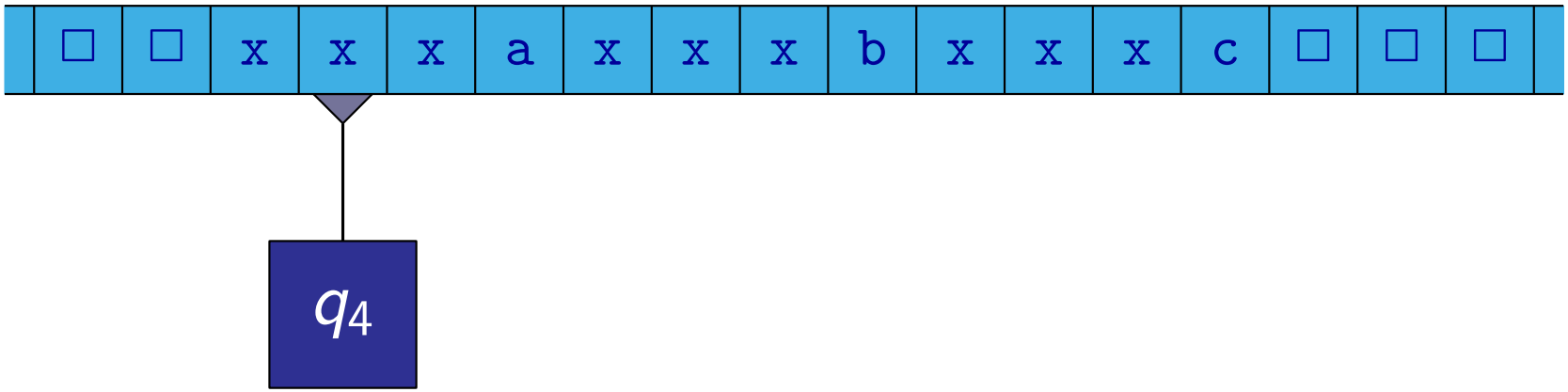
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



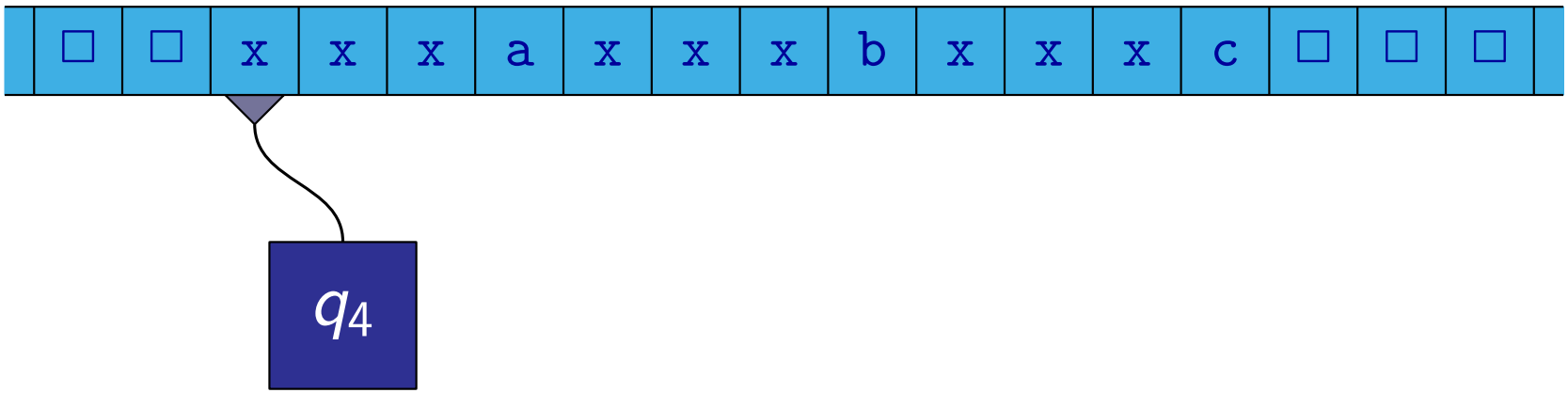
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



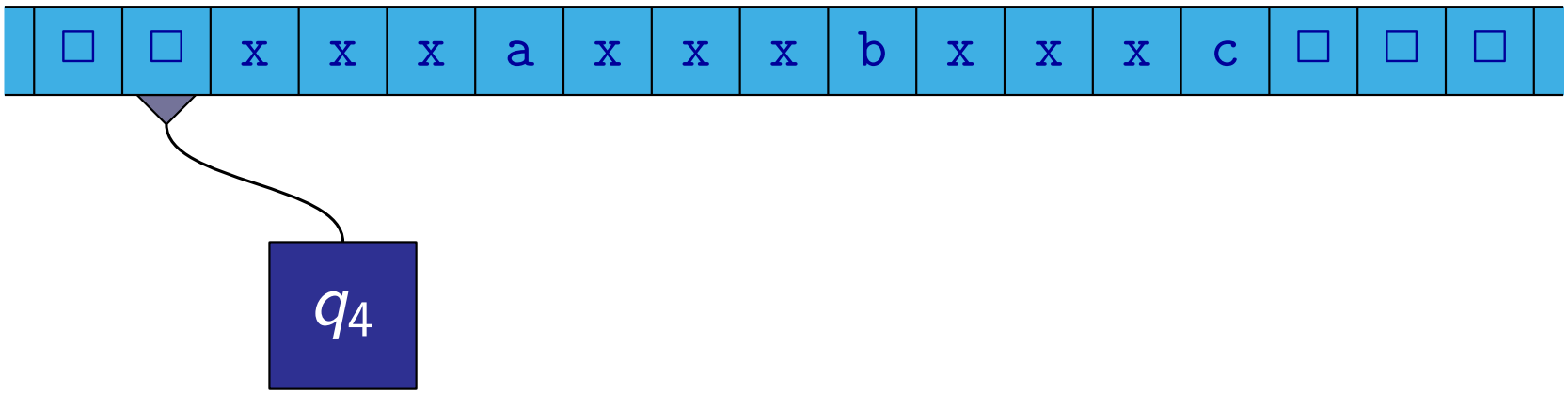
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



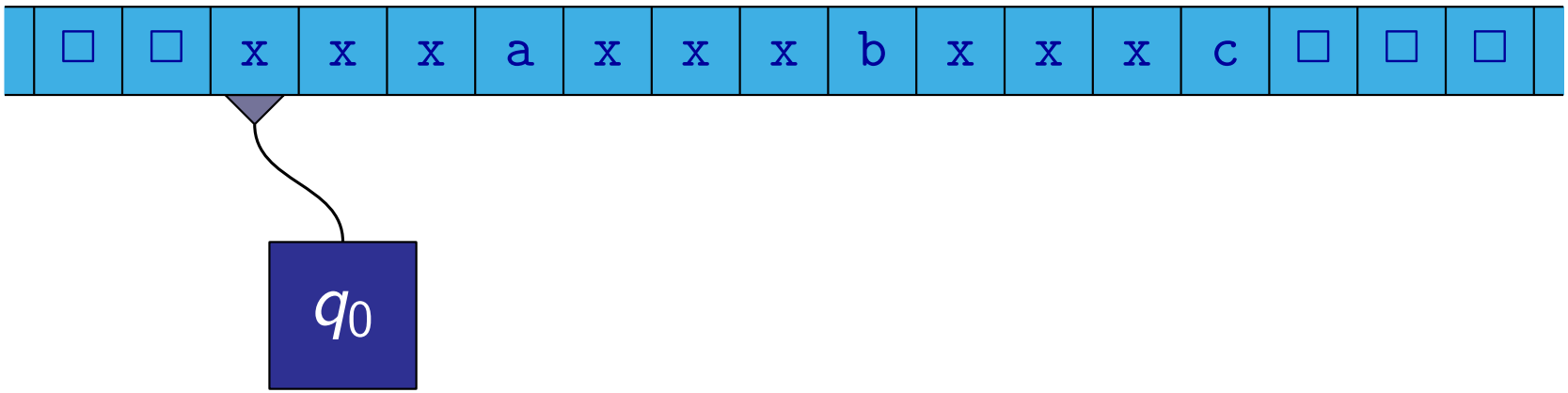
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



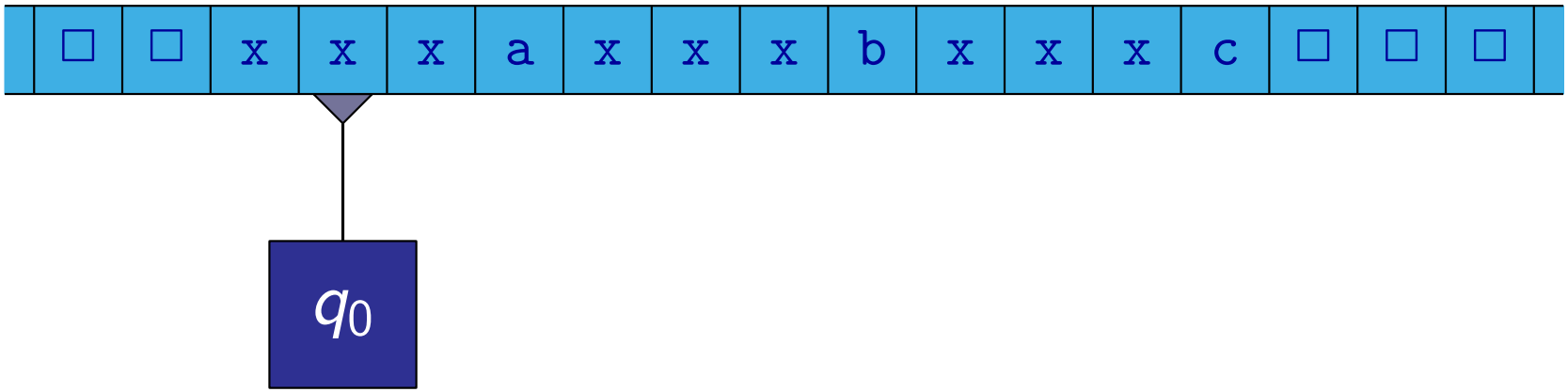
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



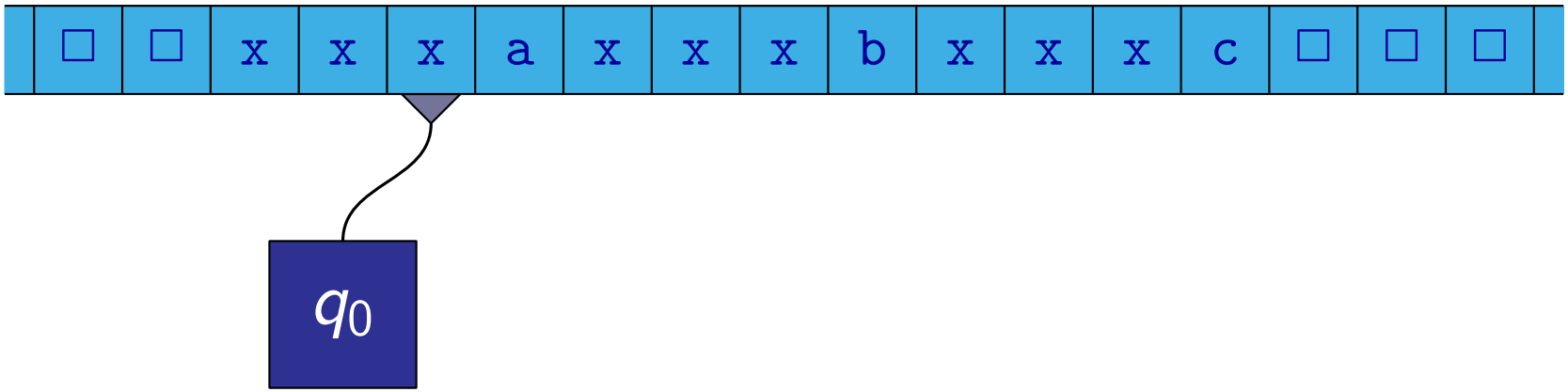
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



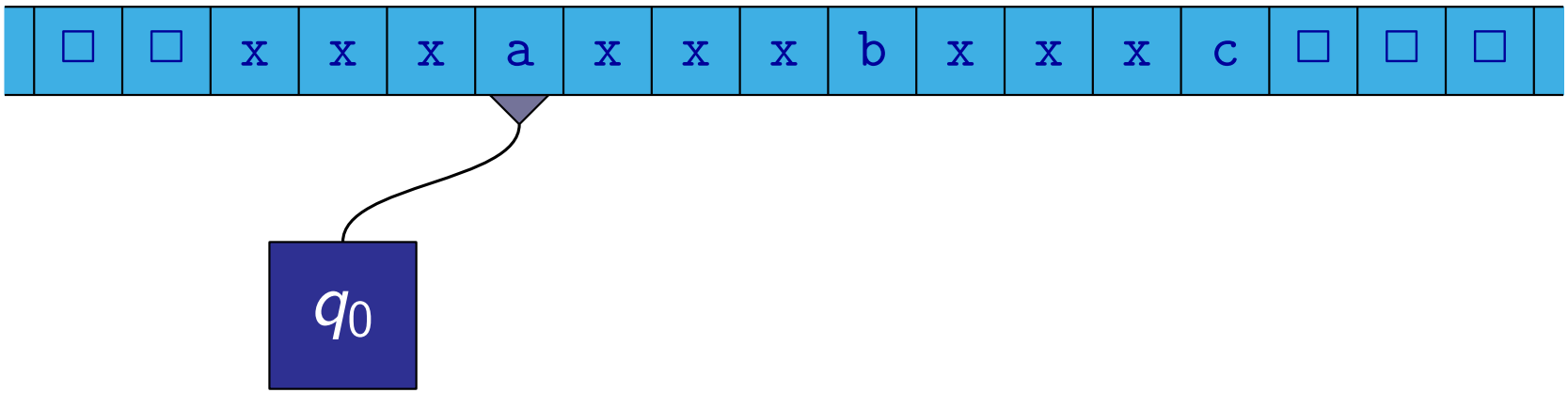
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



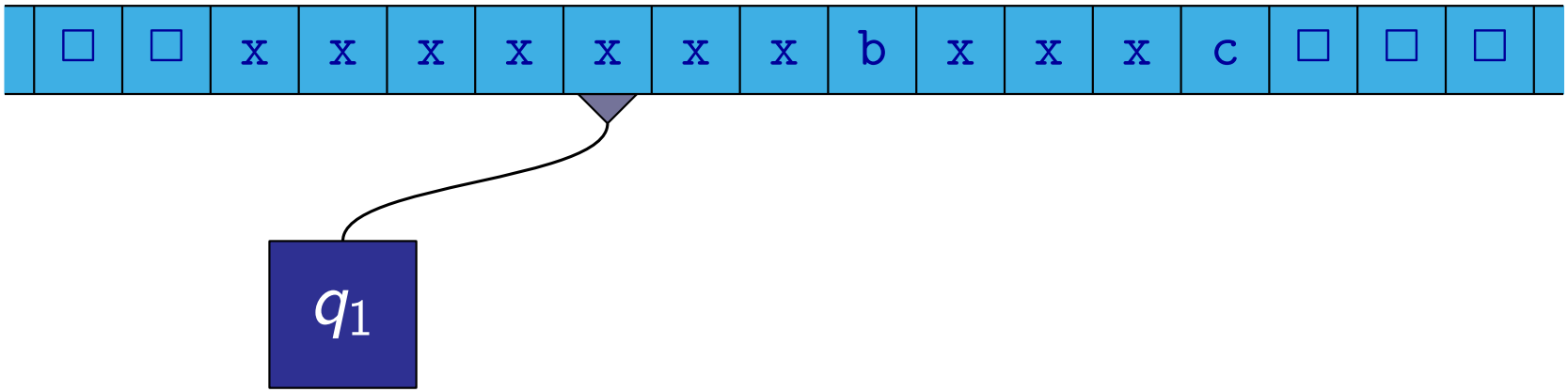
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



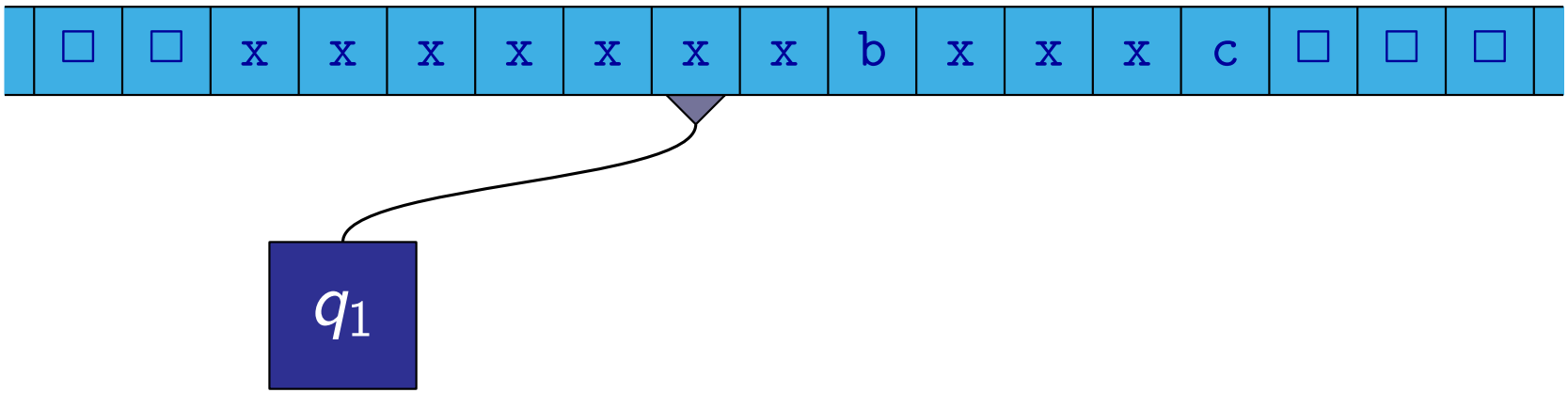
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



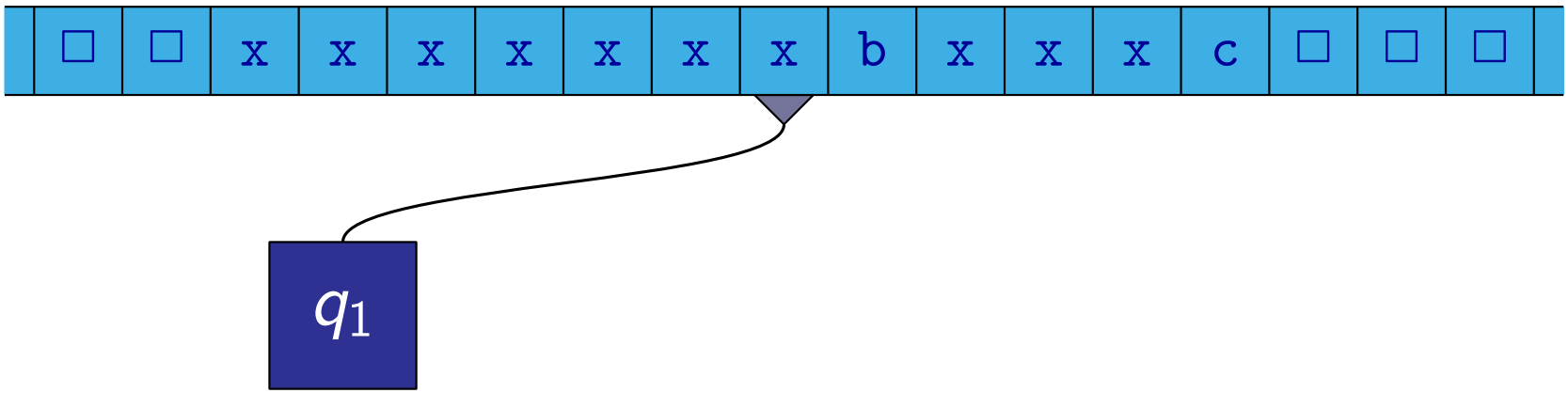
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



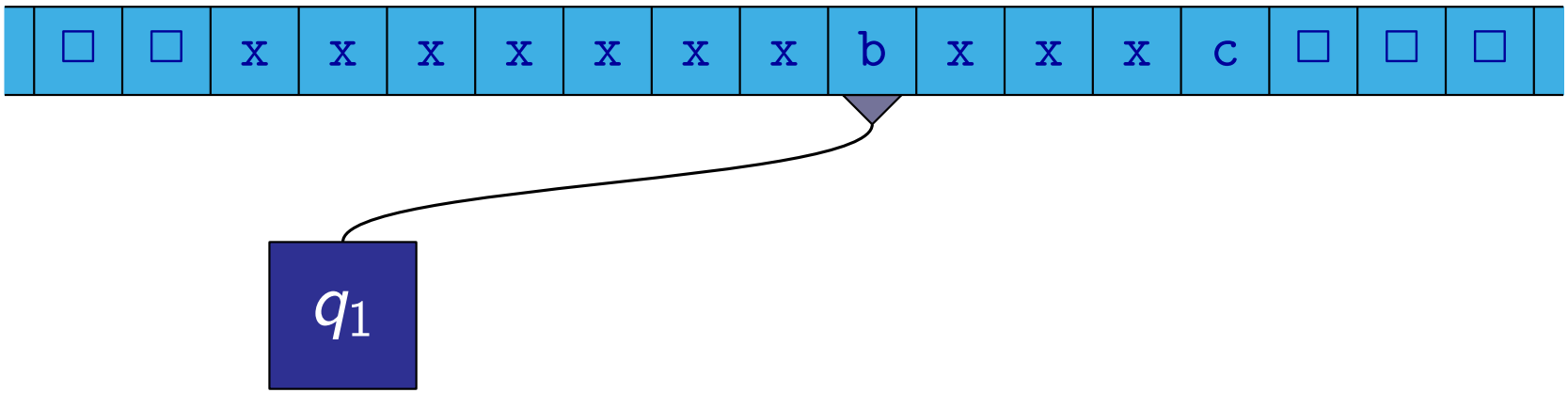
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



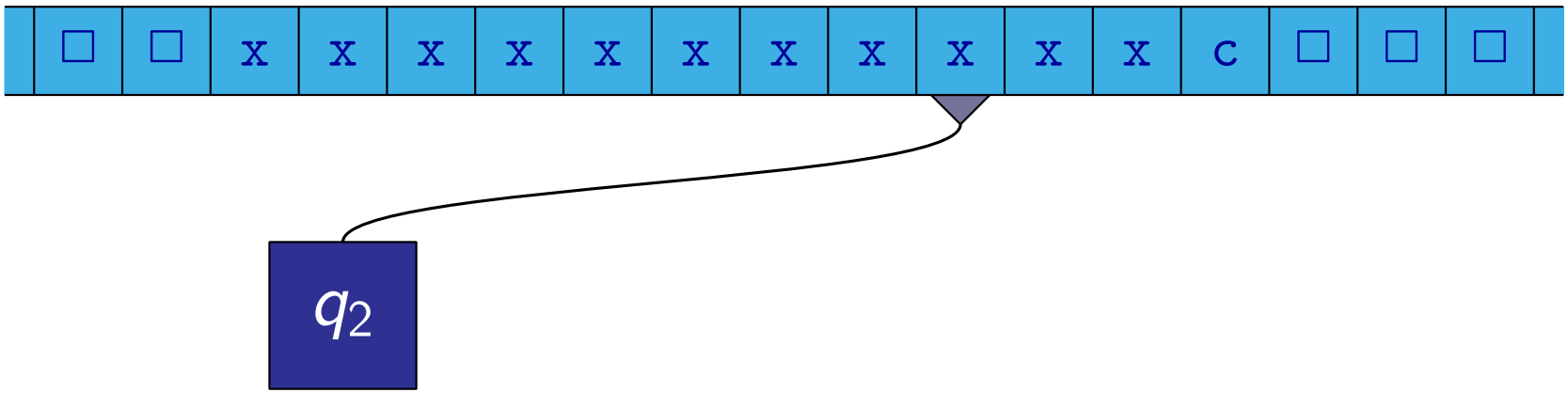
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



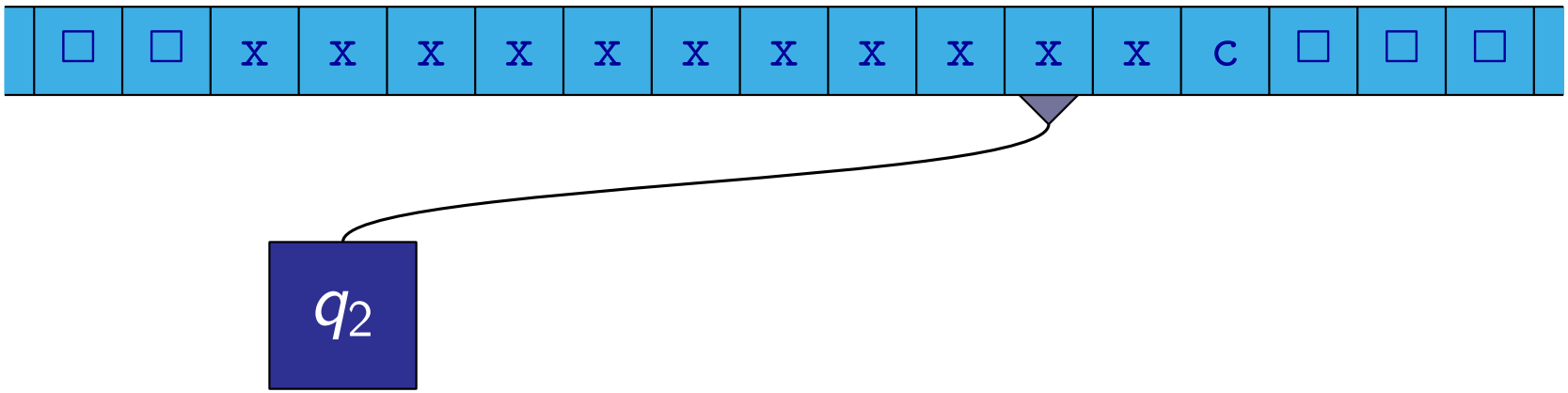
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



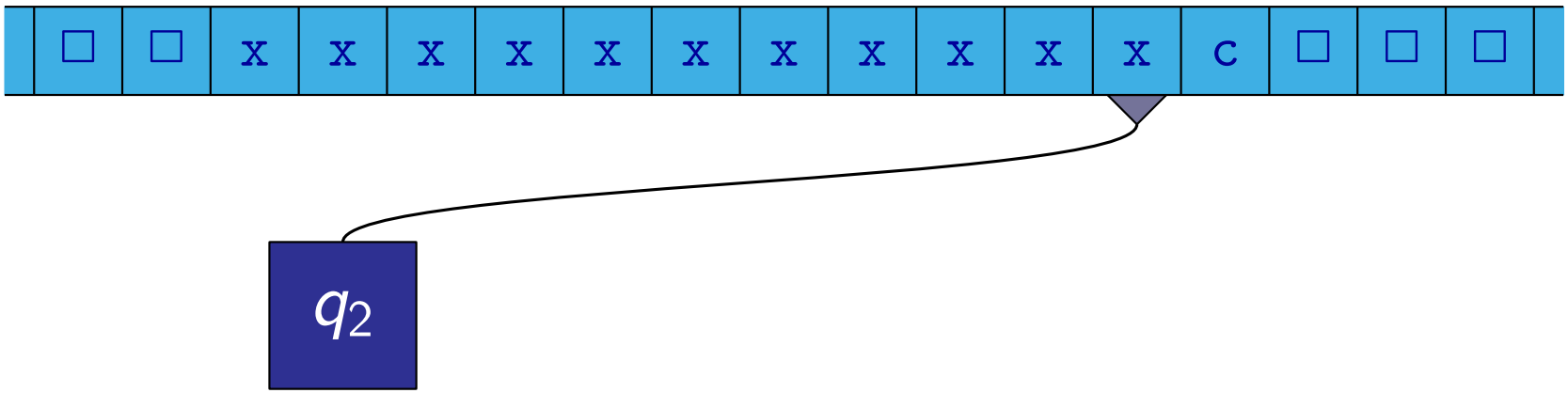
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



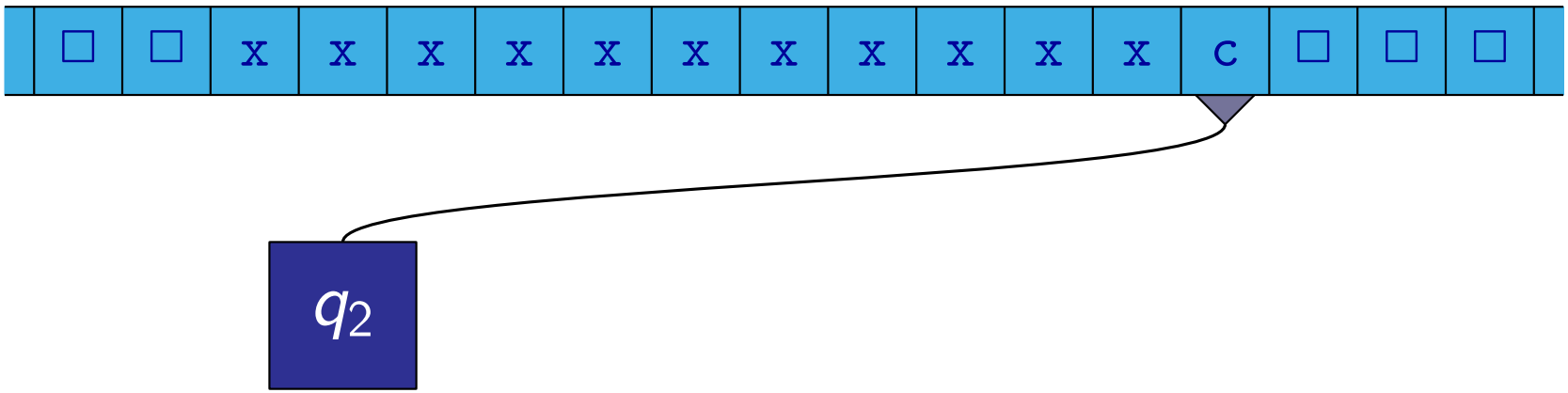
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



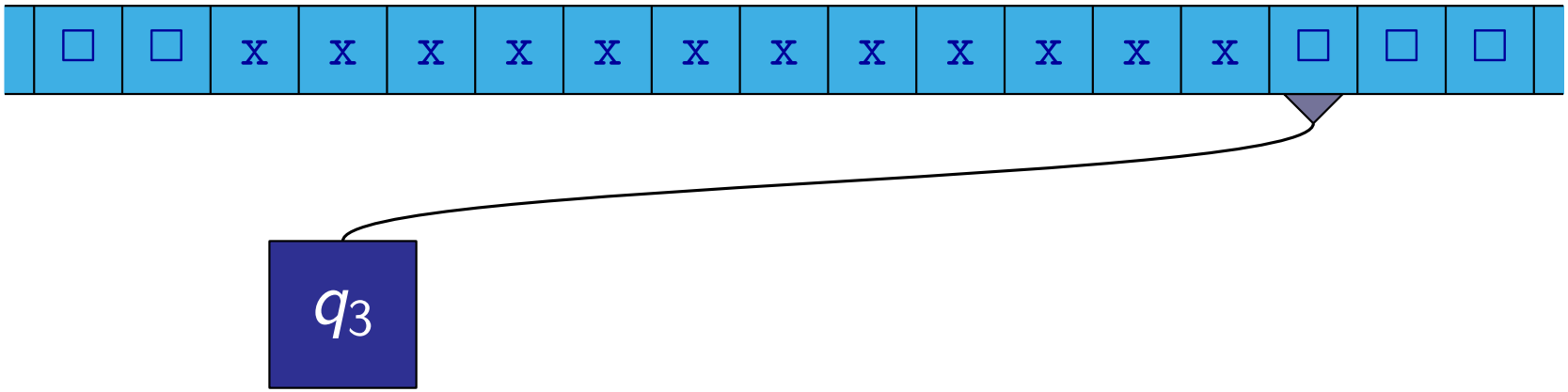
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



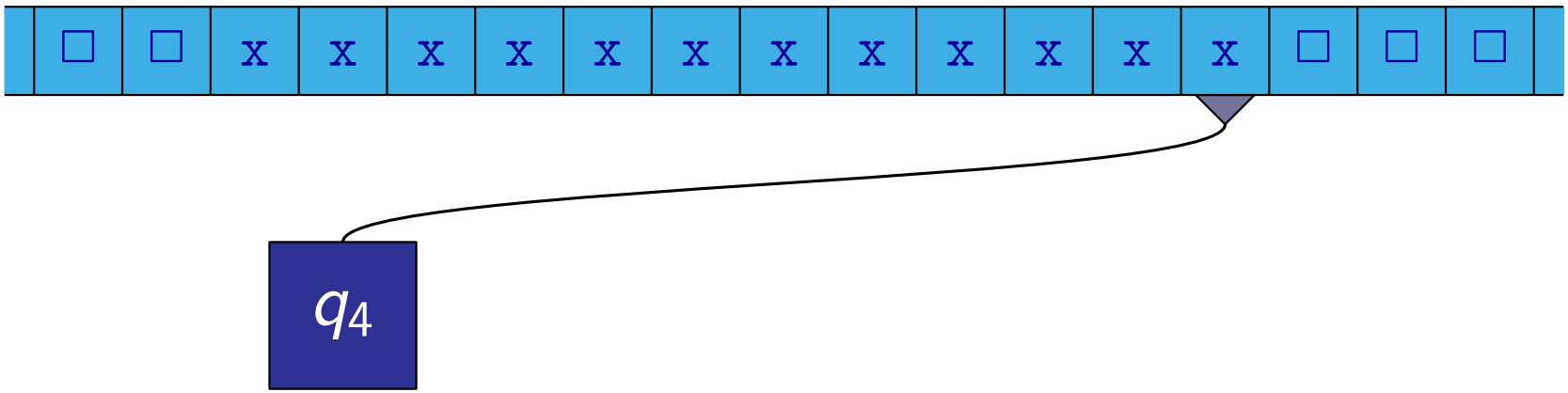
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



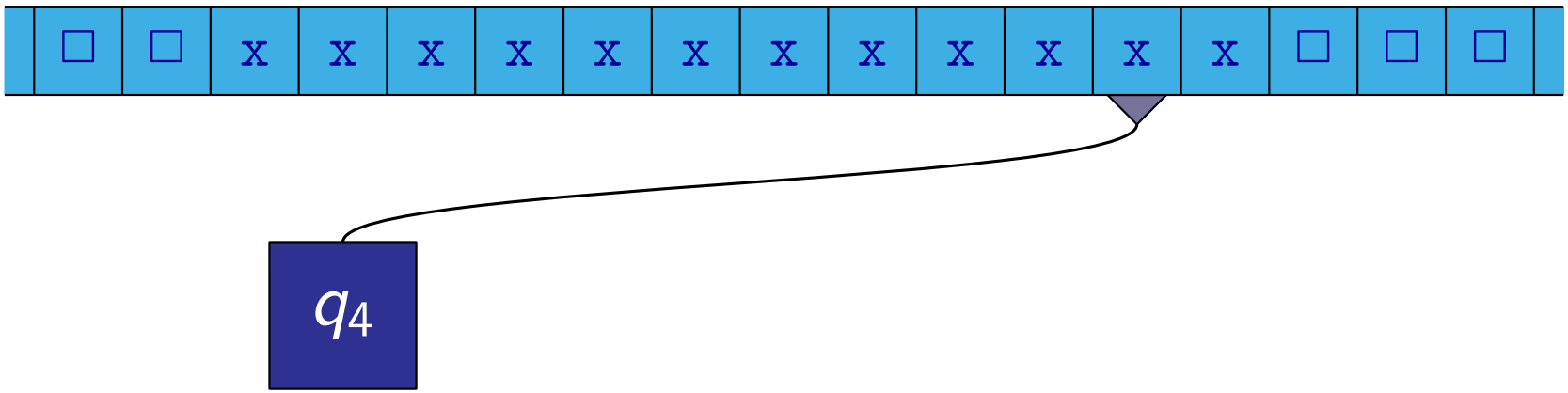
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



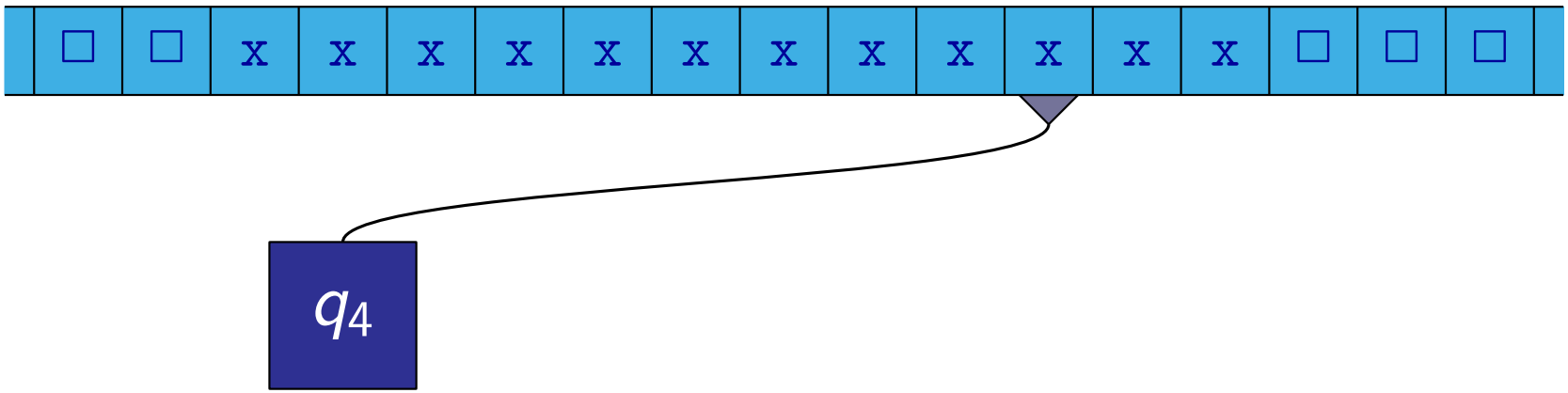
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



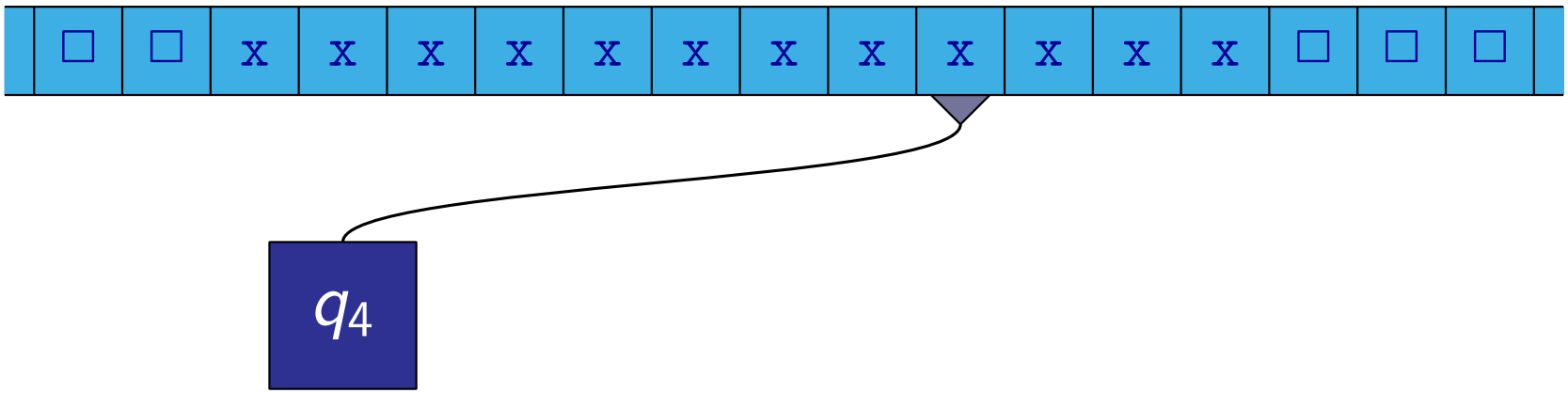
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



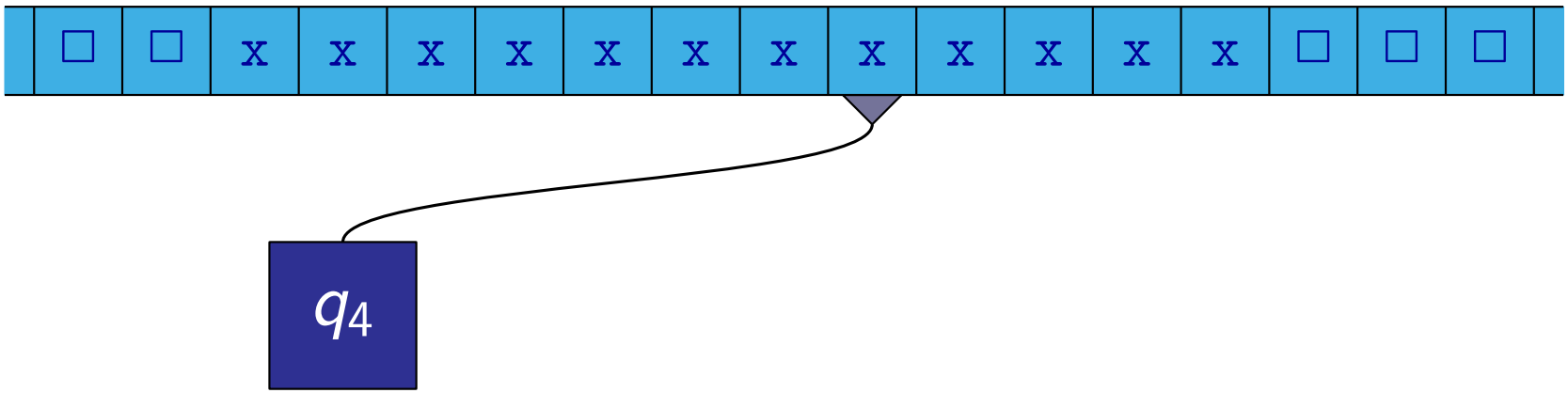
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



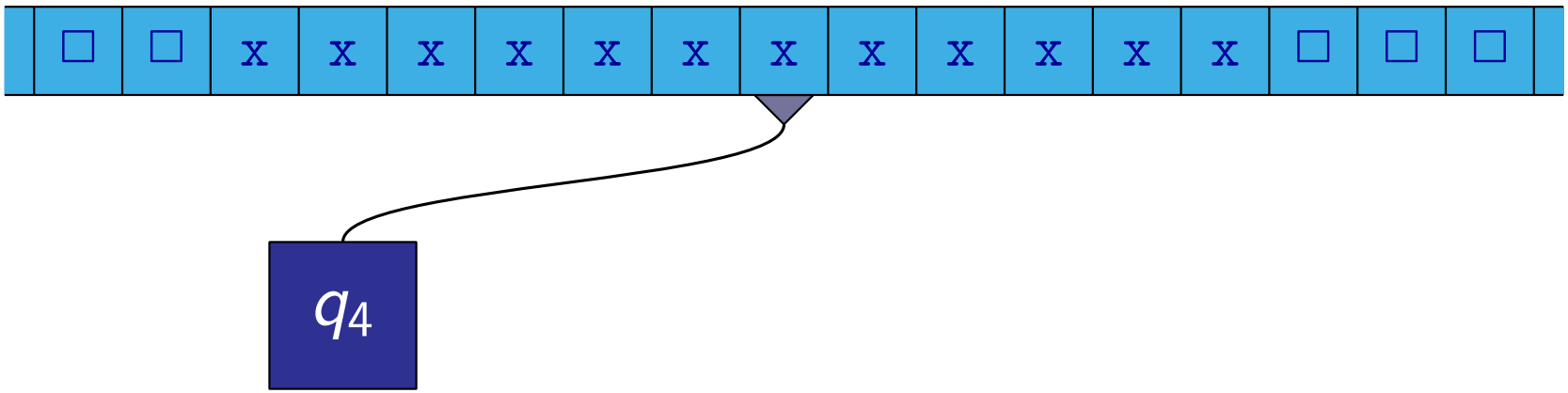
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



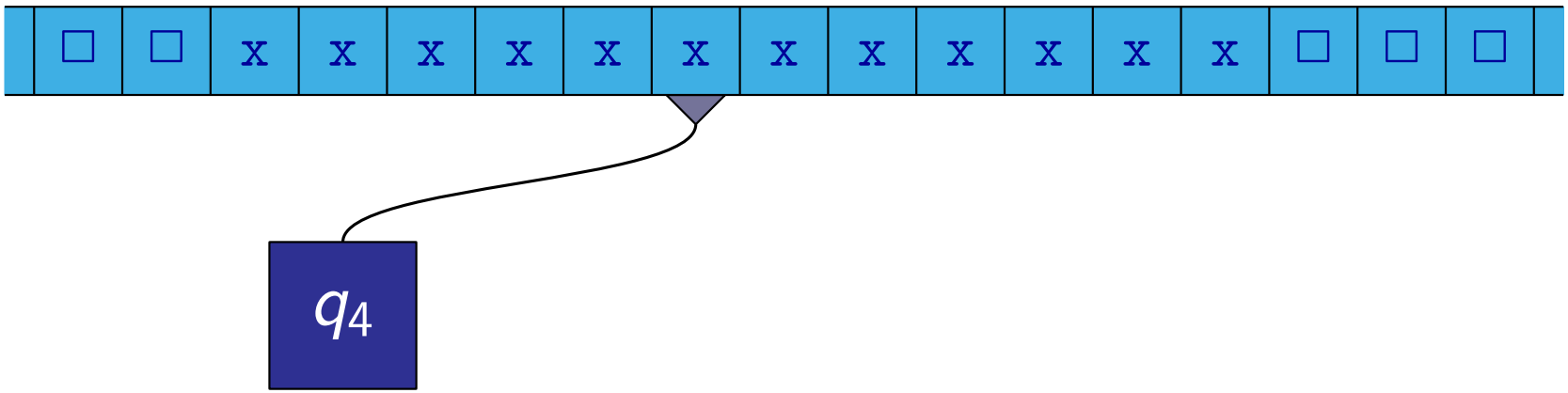
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



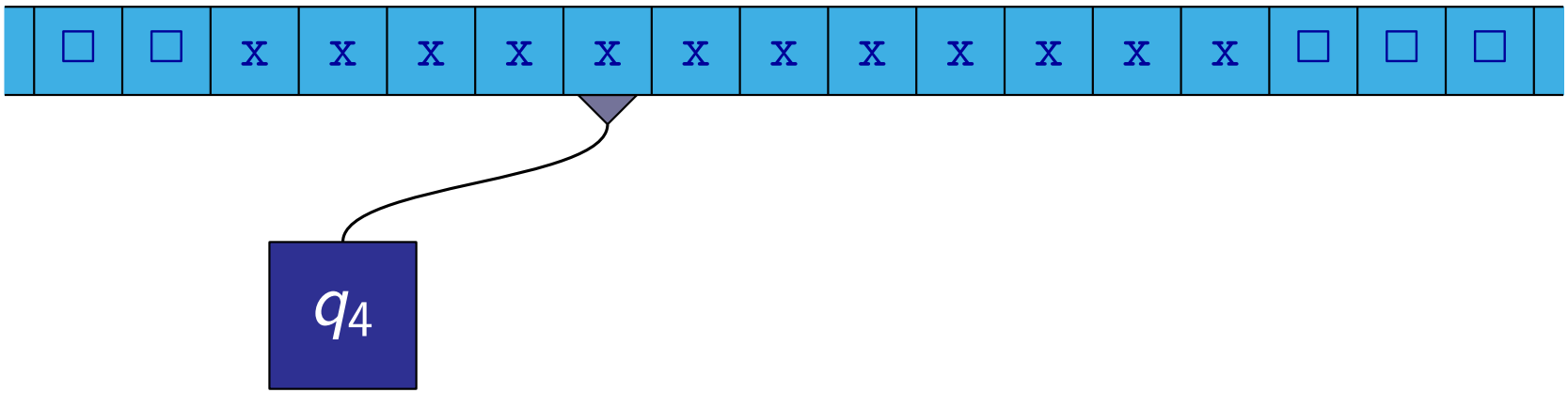
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



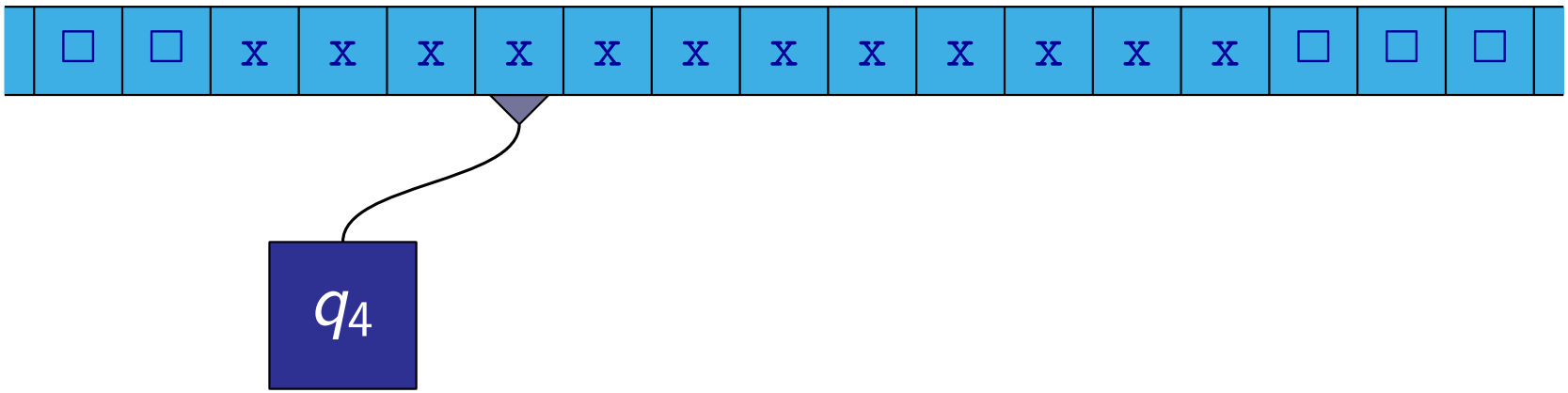
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



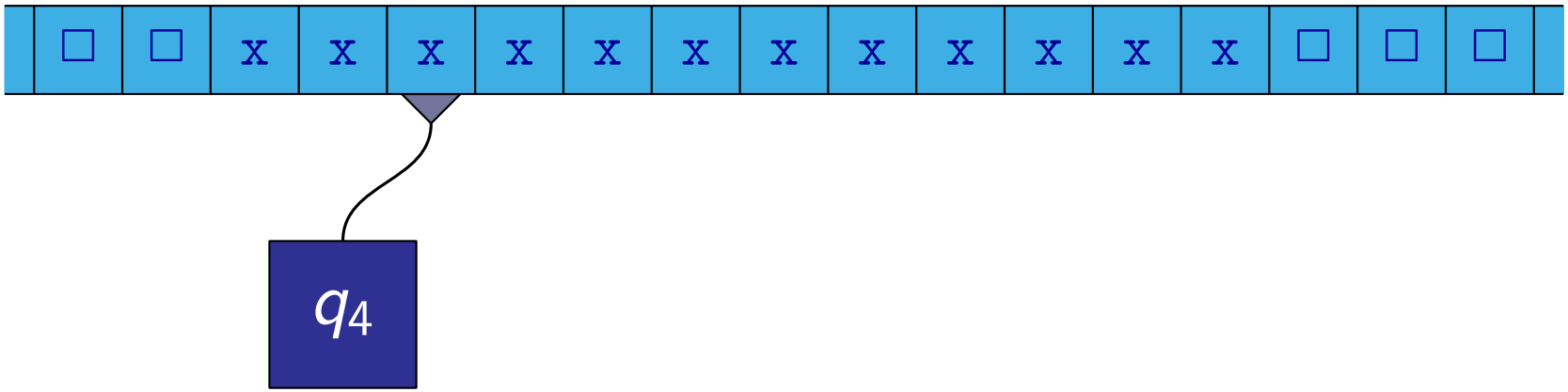
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



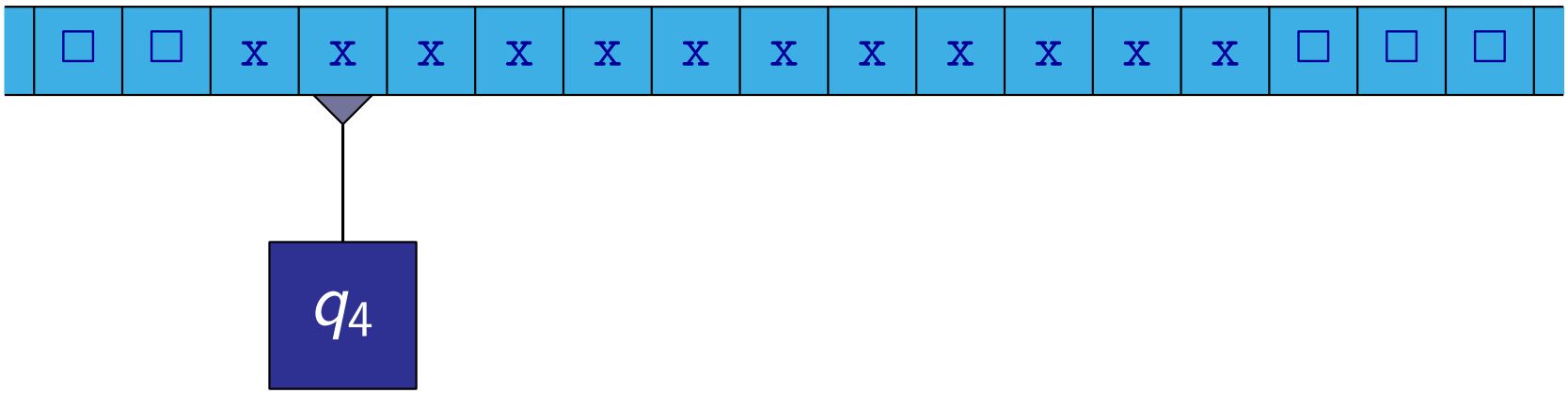
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



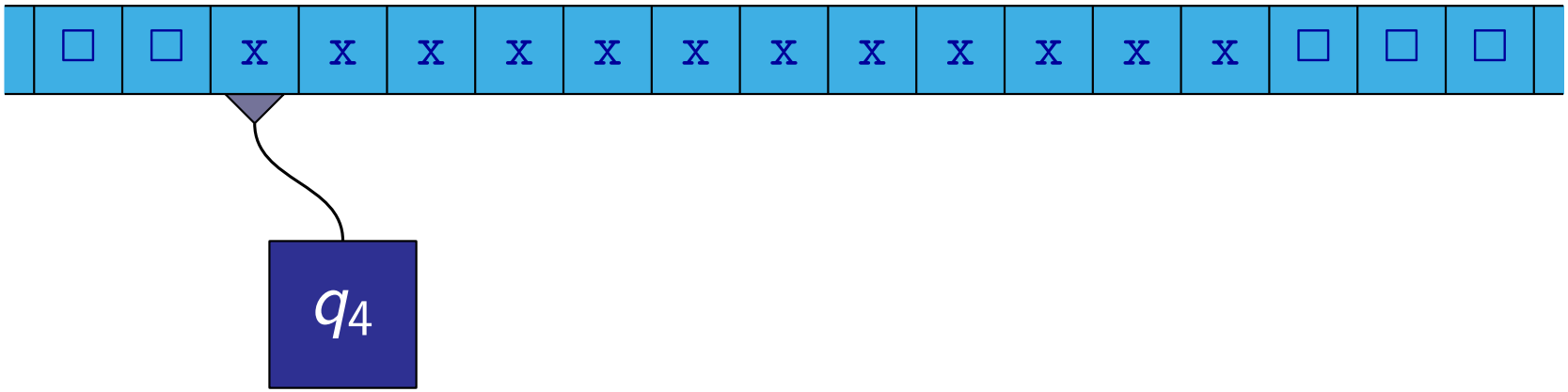
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



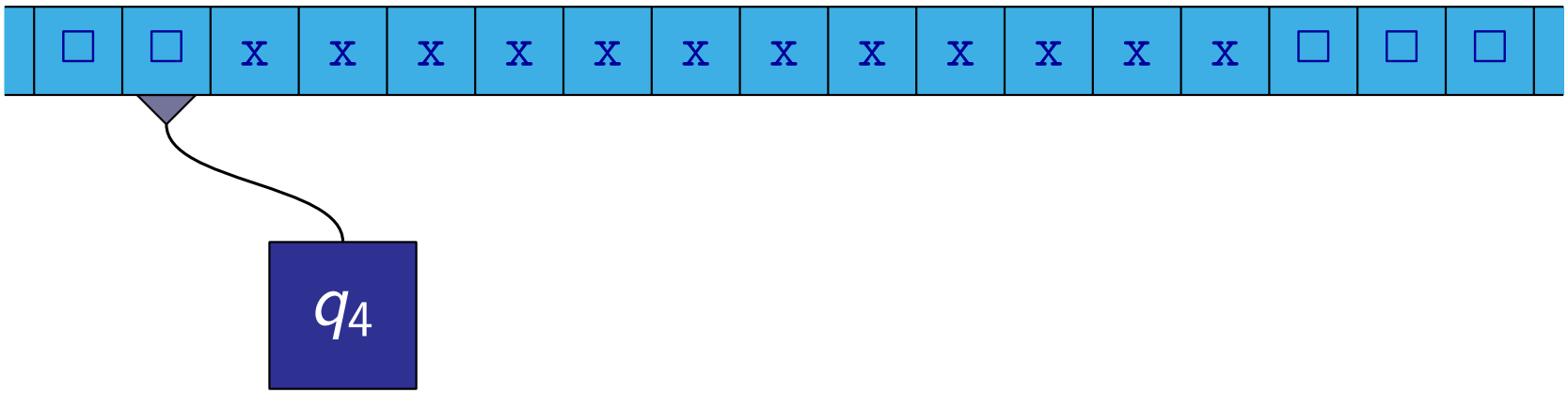
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



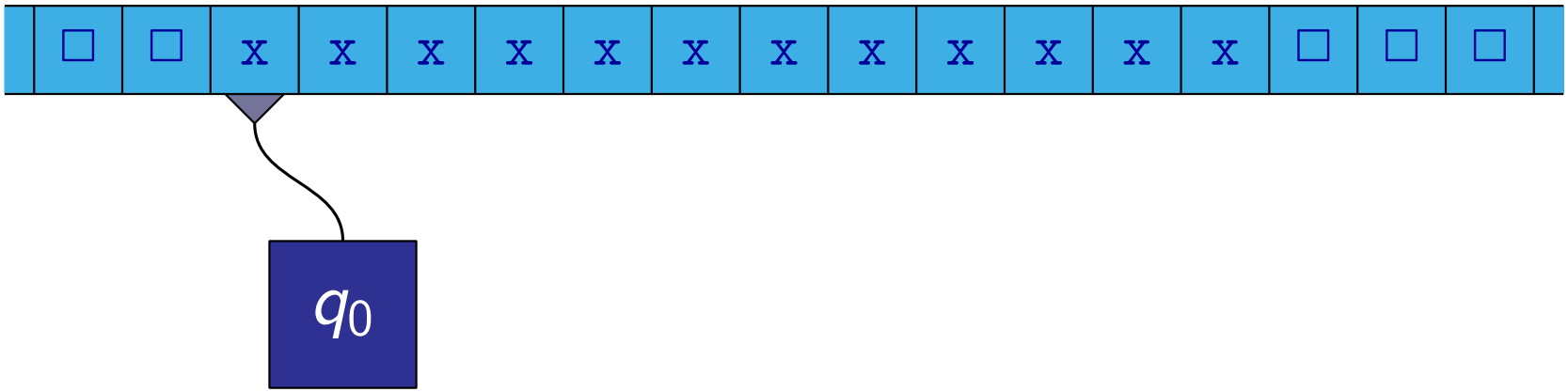
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



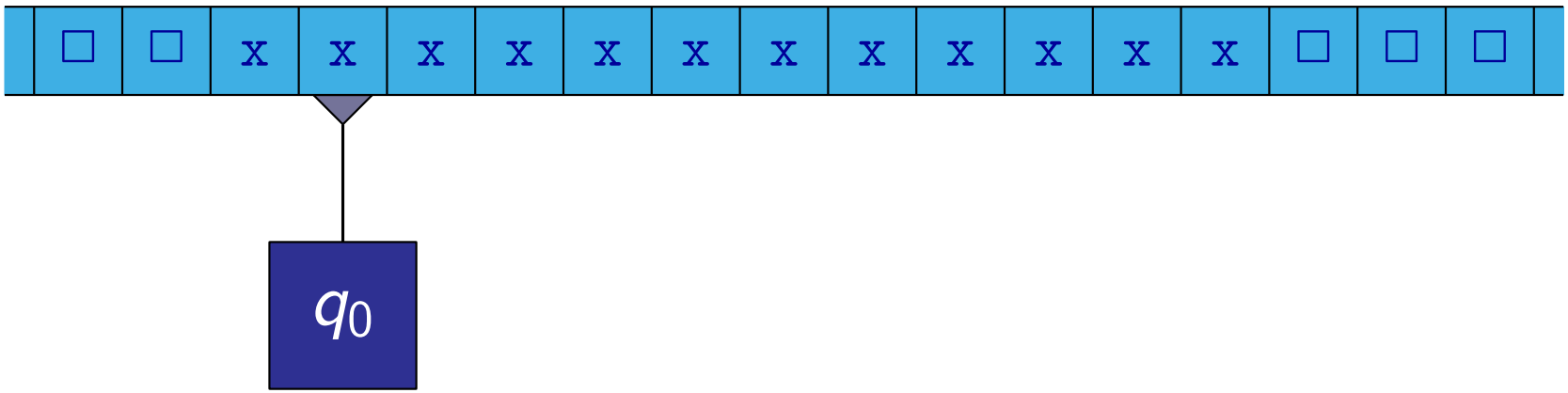
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



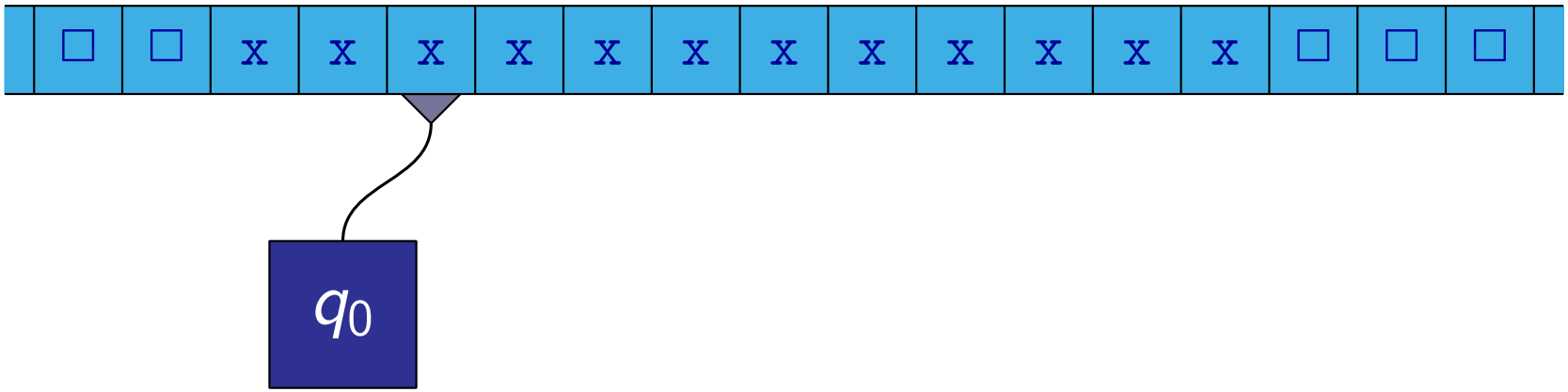
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



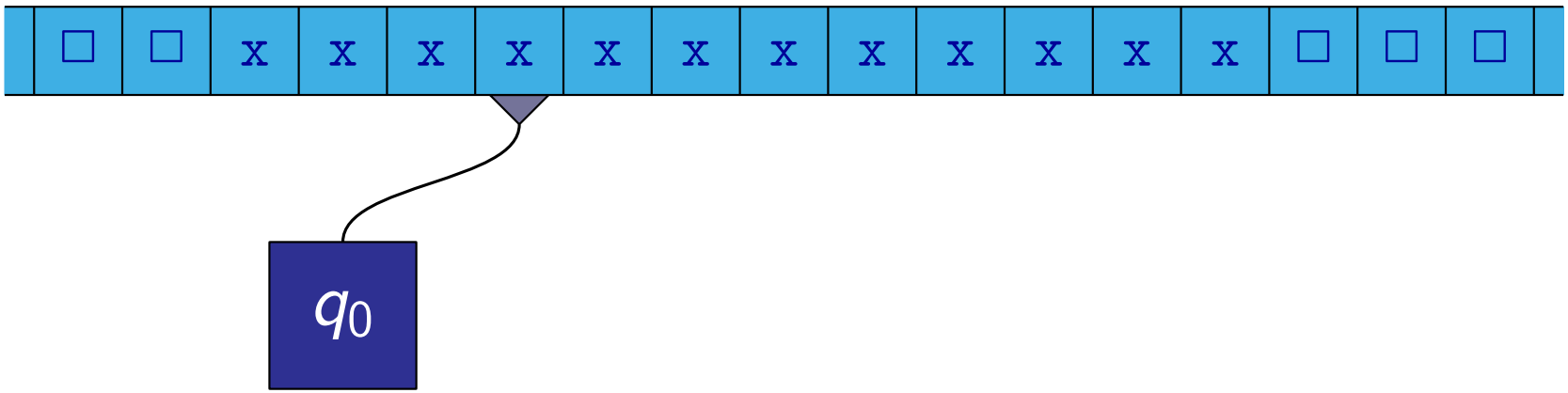
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



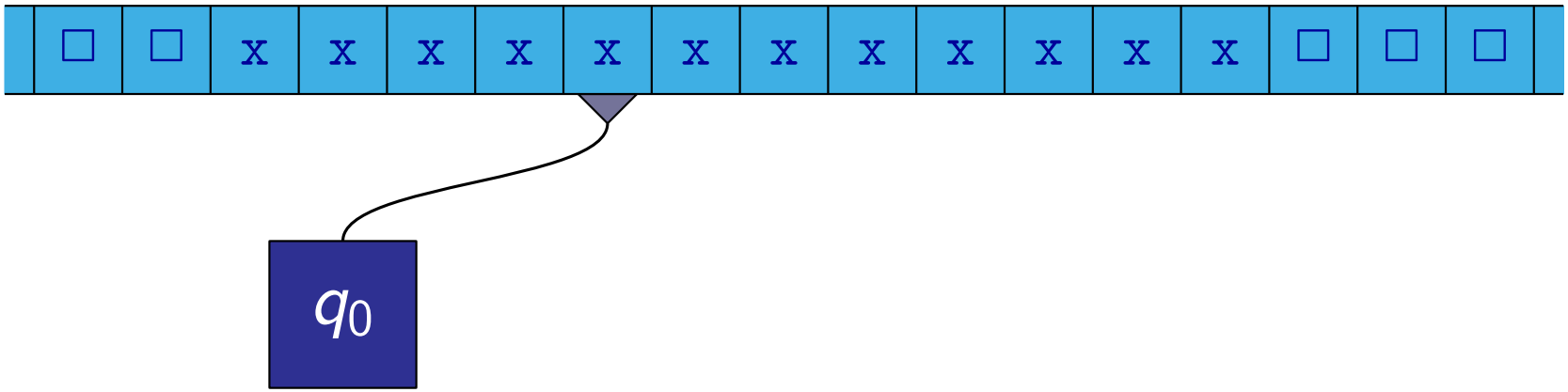
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



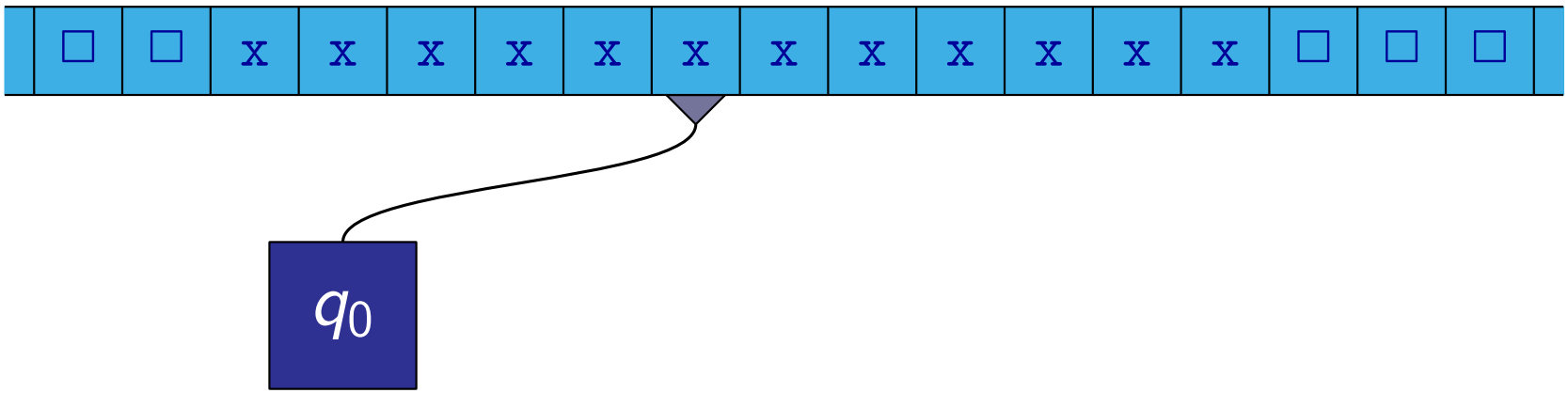
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



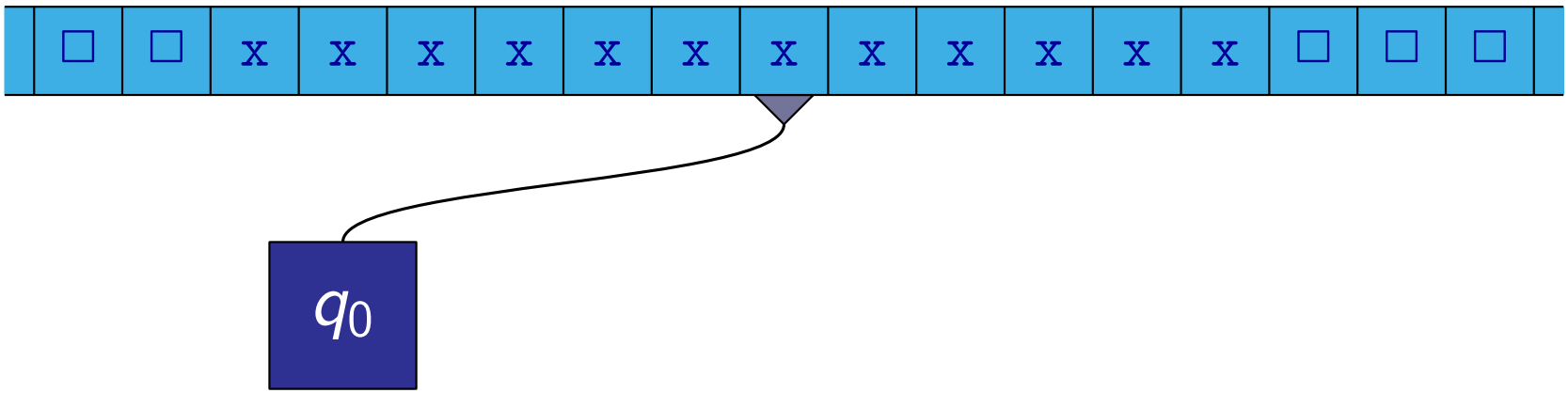
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



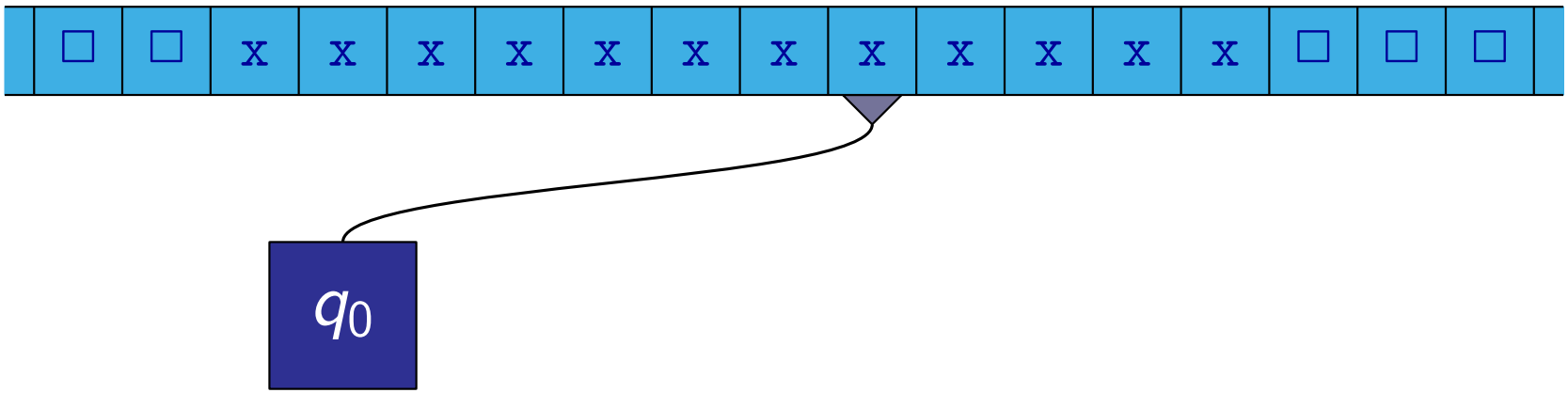
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



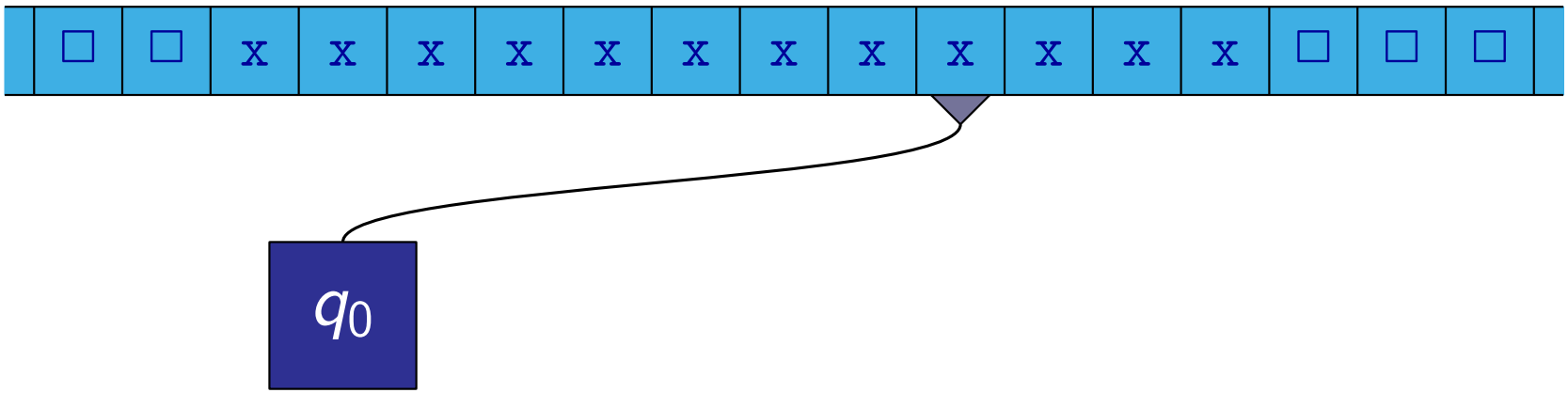
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



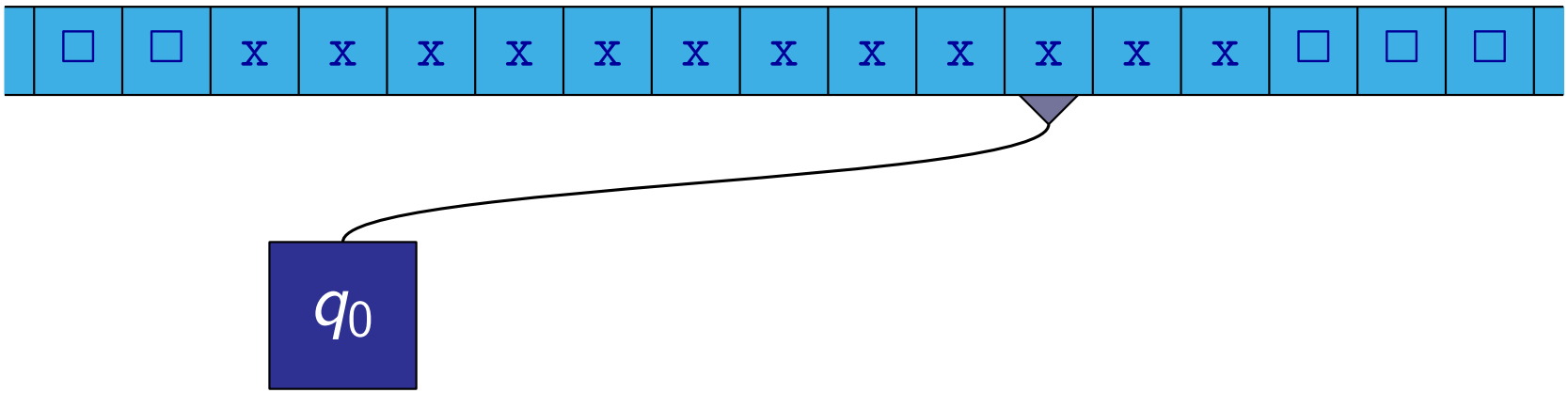
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



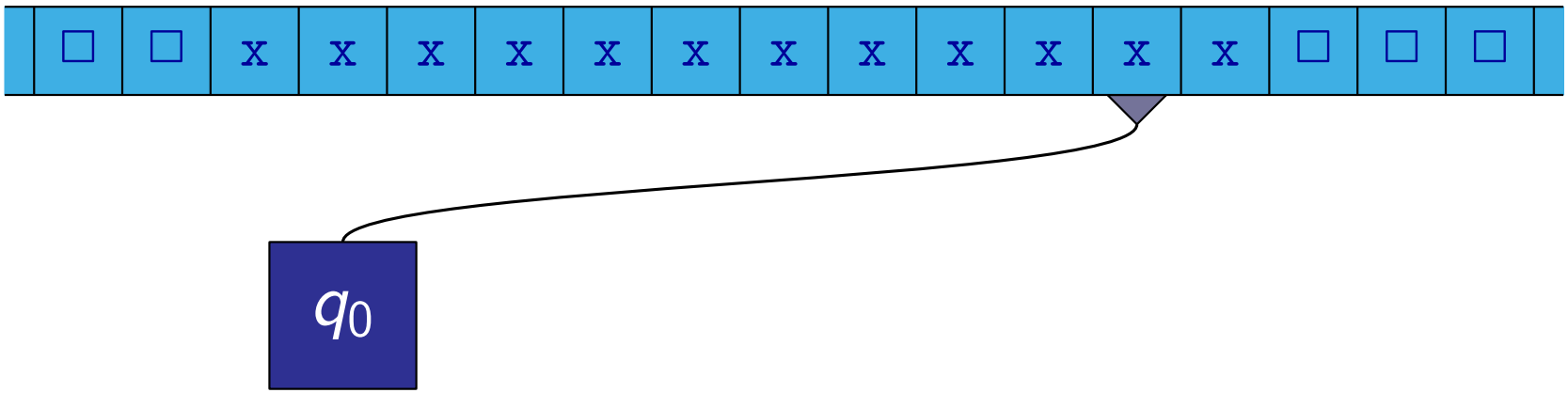
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



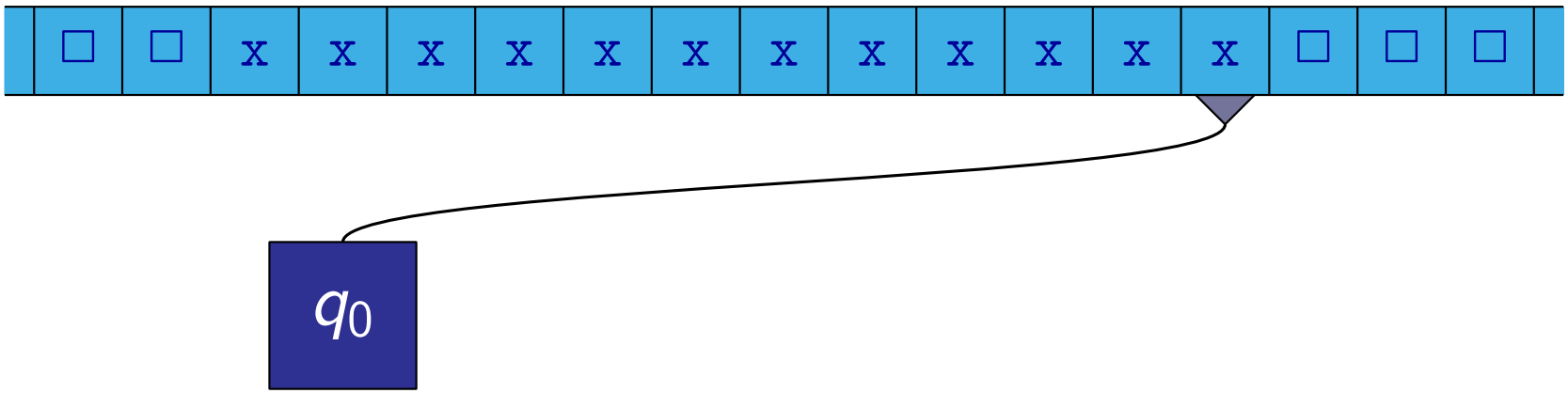
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



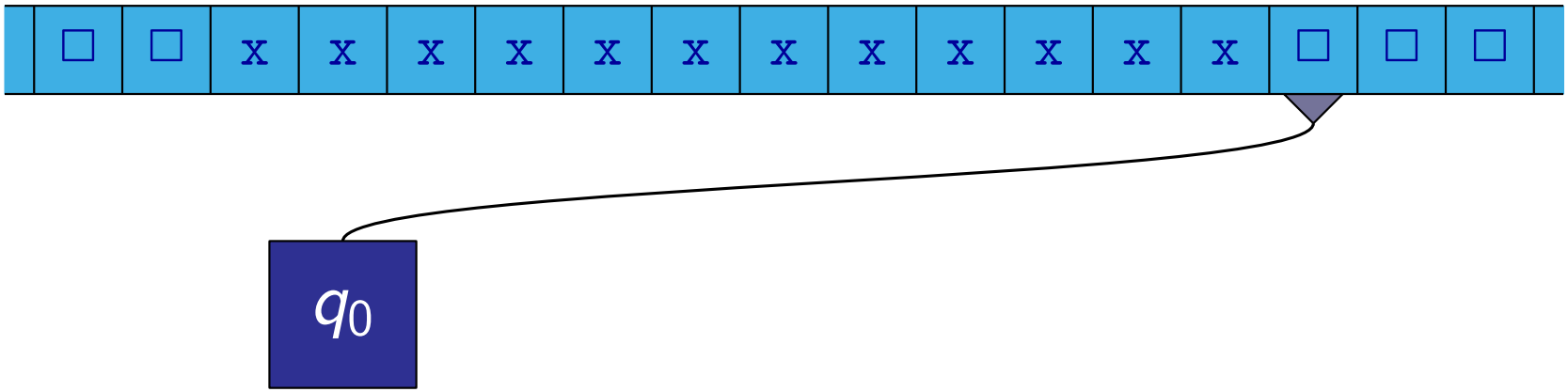
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$



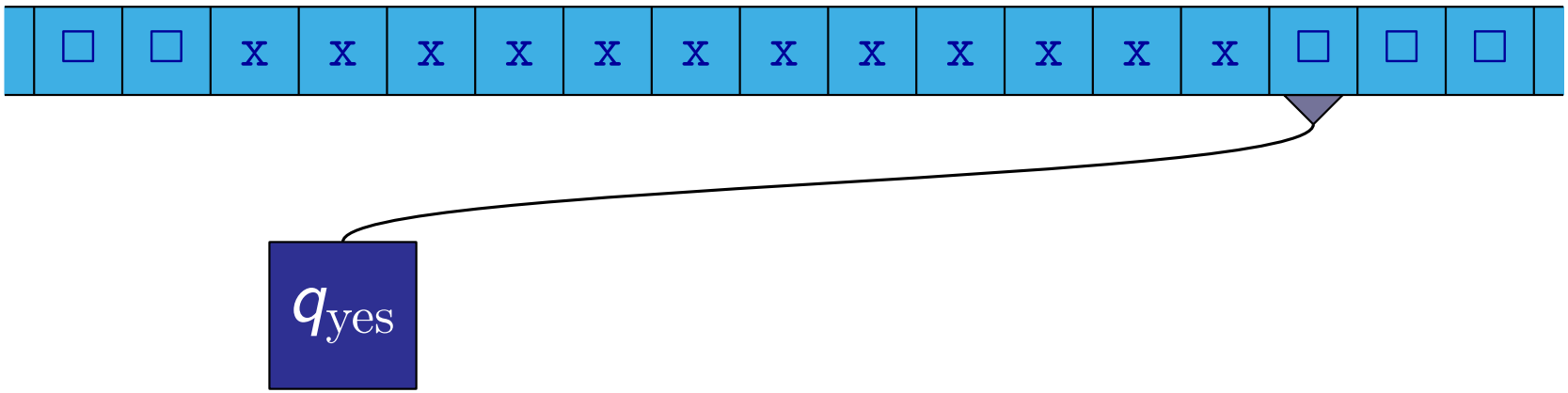
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

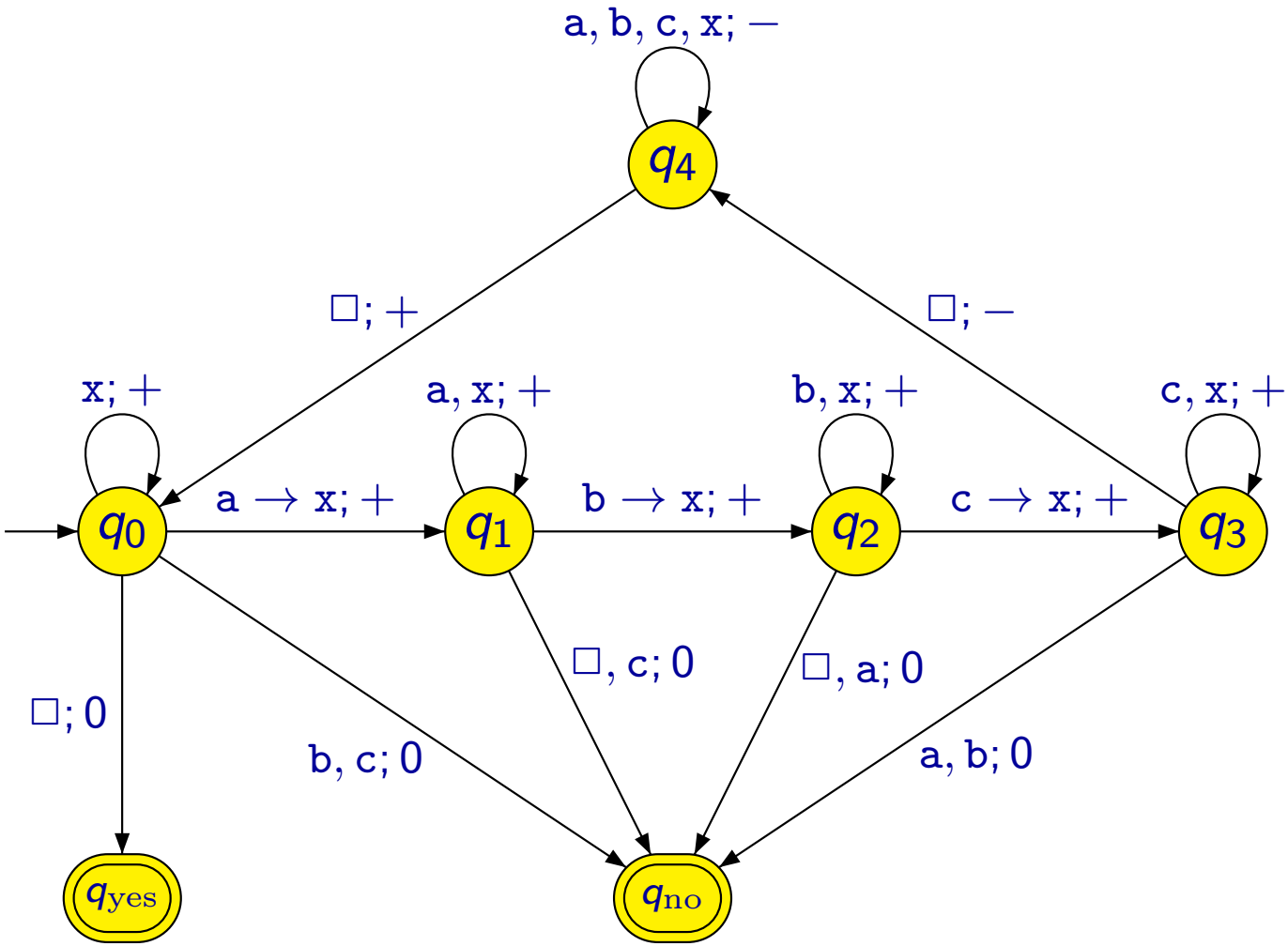
$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $\quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ $\quad \Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{x}\}$

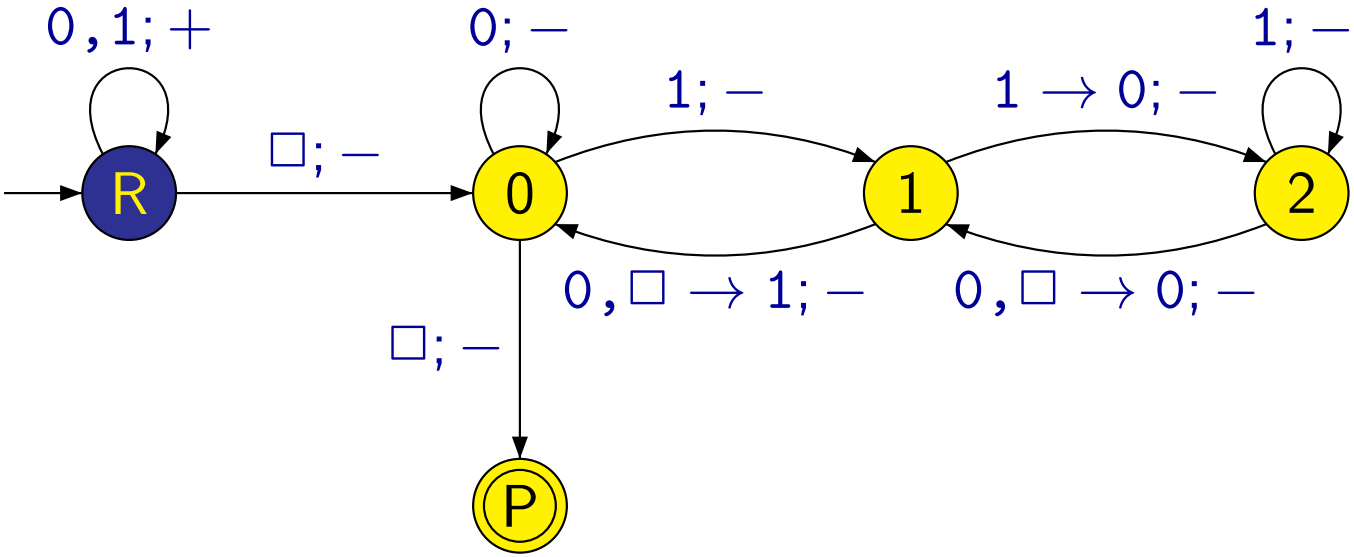
δ	$\square$	a	b	c	x
q_0	$(q_{\text{yes}}, \square, 0)$	$(q_1, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_0, \mathtt{x}, +1)$
q_1	$(q_{\text{no}}, \square, 0)$	$(q_1, \mathtt{a}, +1)$	$(q_2, \mathtt{x}, +1)$	$(q_{\text{no}}, \mathtt{c}, 0)$	$(q_1, \mathtt{x}, +1)$
q_2	$(q_{\text{no}}, \square, 0)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_2, \mathtt{b}, +1)$	$(q_3, \mathtt{x}, +1)$	$(q_2, \mathtt{x}, +1)$
q_3	$(q_4, \square, -1)$	$(q_{\text{no}}, \mathtt{a}, 0)$	$(q_{\text{no}}, \mathtt{b}, 0)$	$(q_3, \mathtt{c}, +1)$	$(q_3, \mathtt{x}, +1)$
q_4	$(q_0, \square, +1)$	$(q_4, \mathtt{a}, -1)$	$(q_4, \mathtt{b}, -1)$	$(q_4, \mathtt{c}, -1)$	$(q_4, \mathtt{x}, -1)$

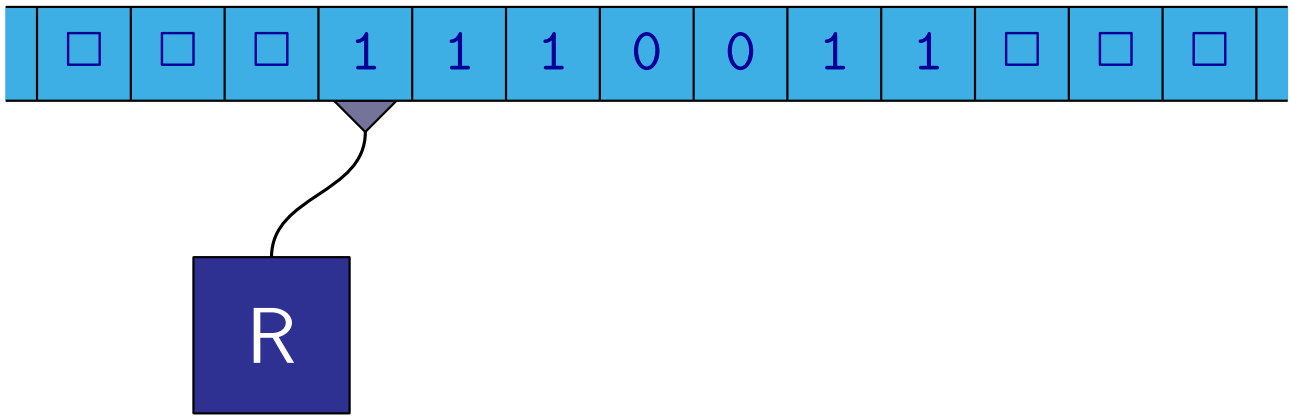


Turing Machine

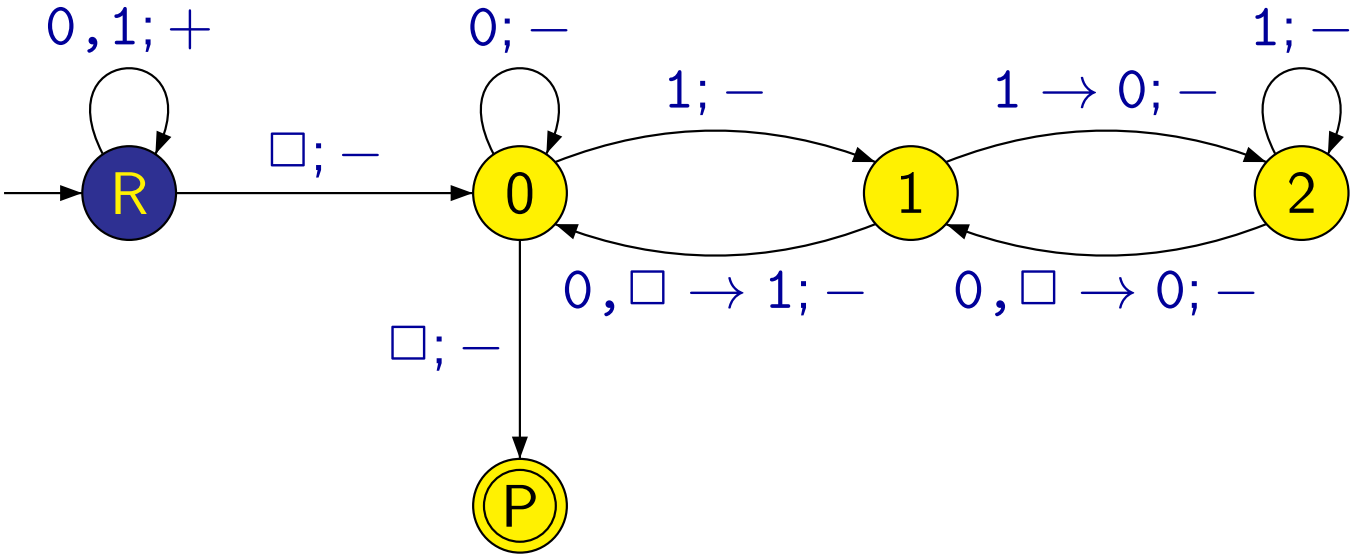


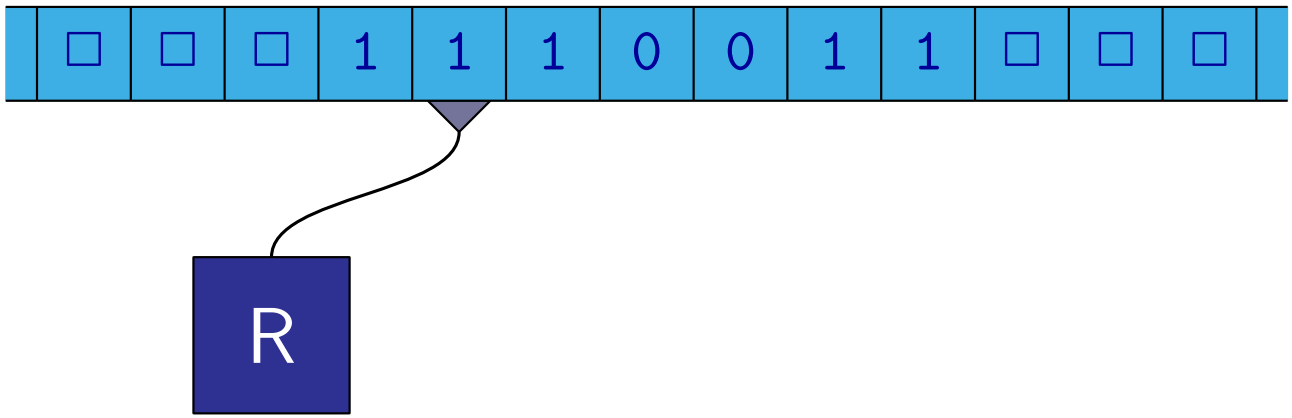
Turing Machine – Multiplication by Three



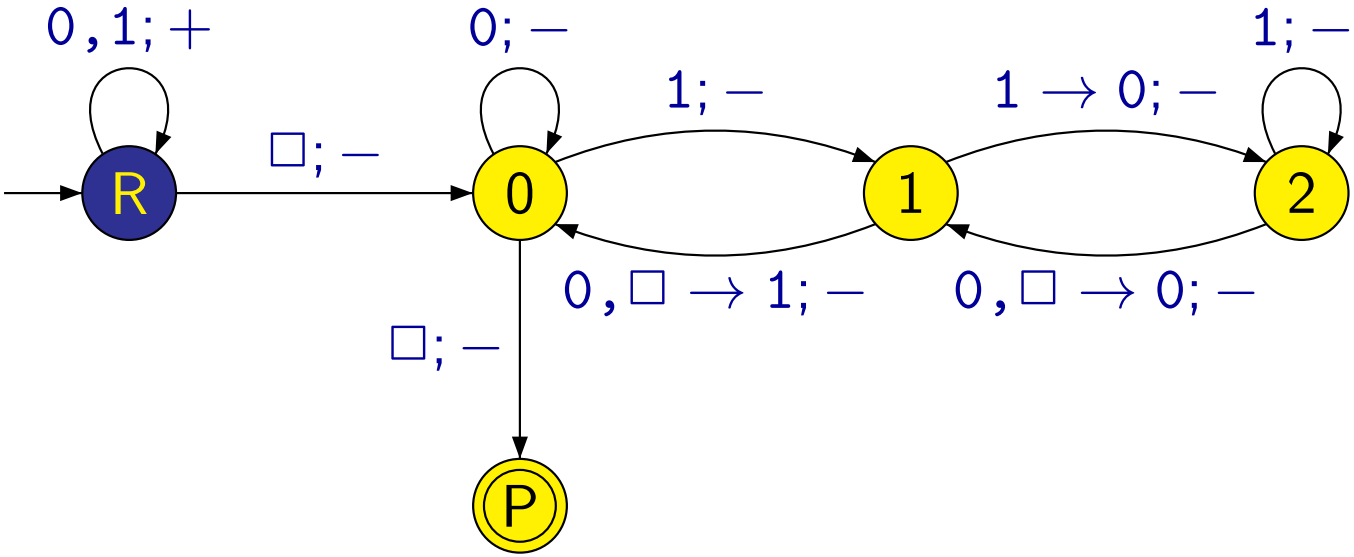


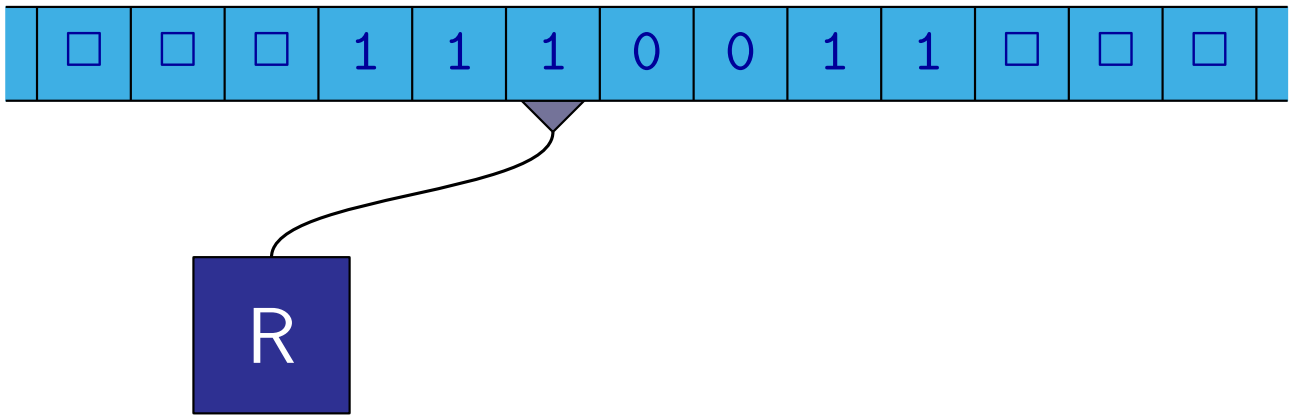
Turing Machine – Multiplication by Three



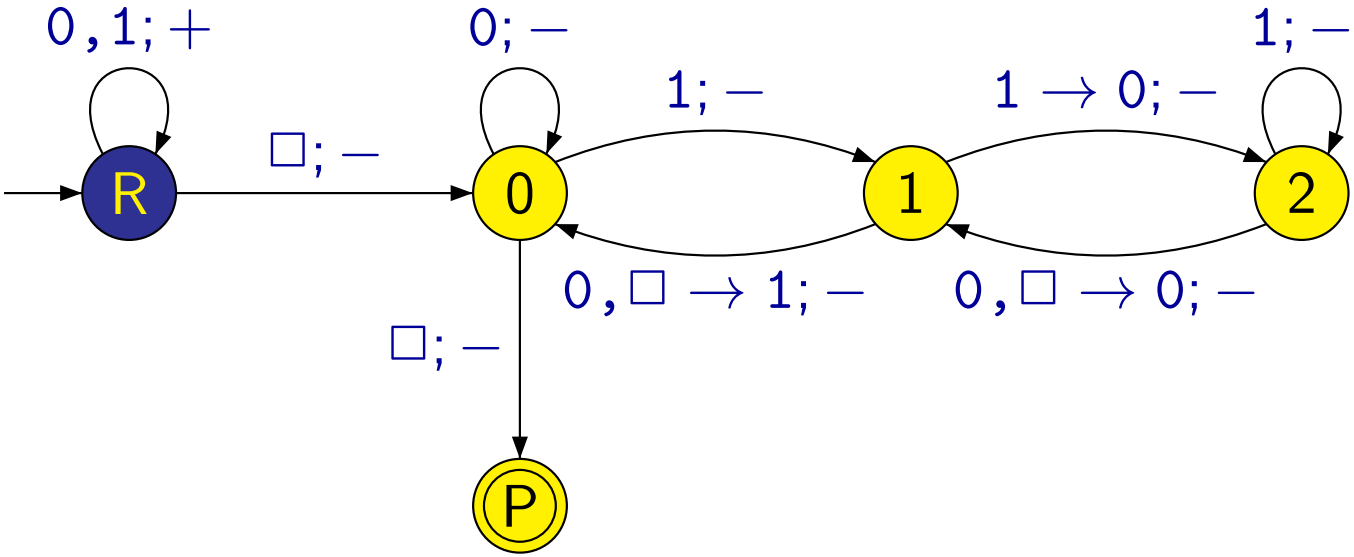


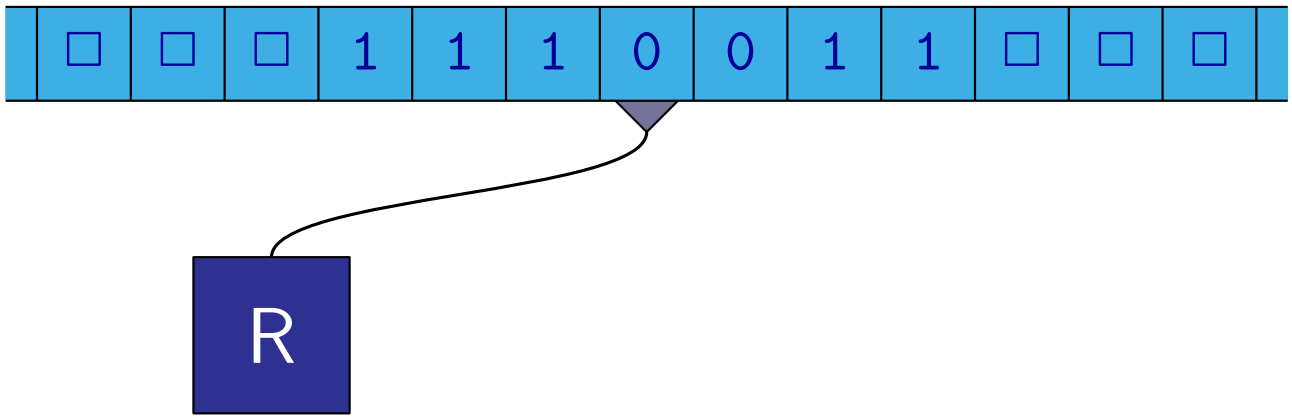
Turing Machine – Multiplication by Three



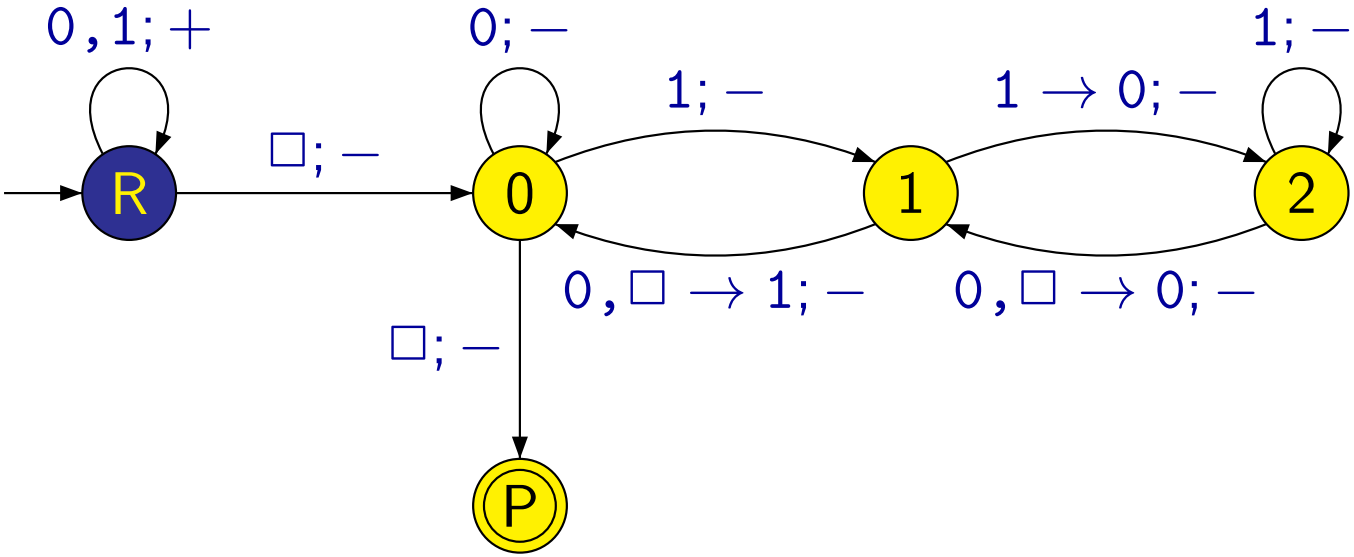


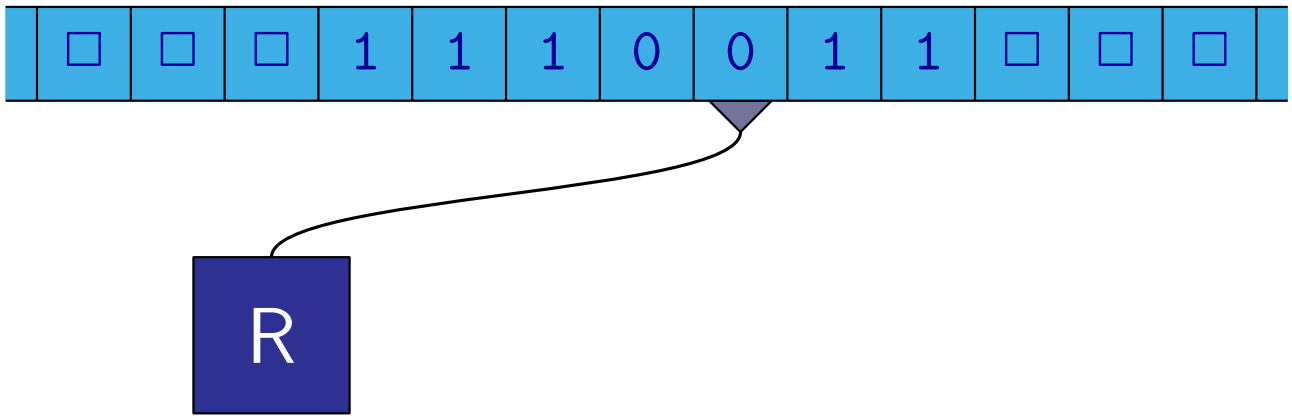
Turing Machine – Multiplication by Three



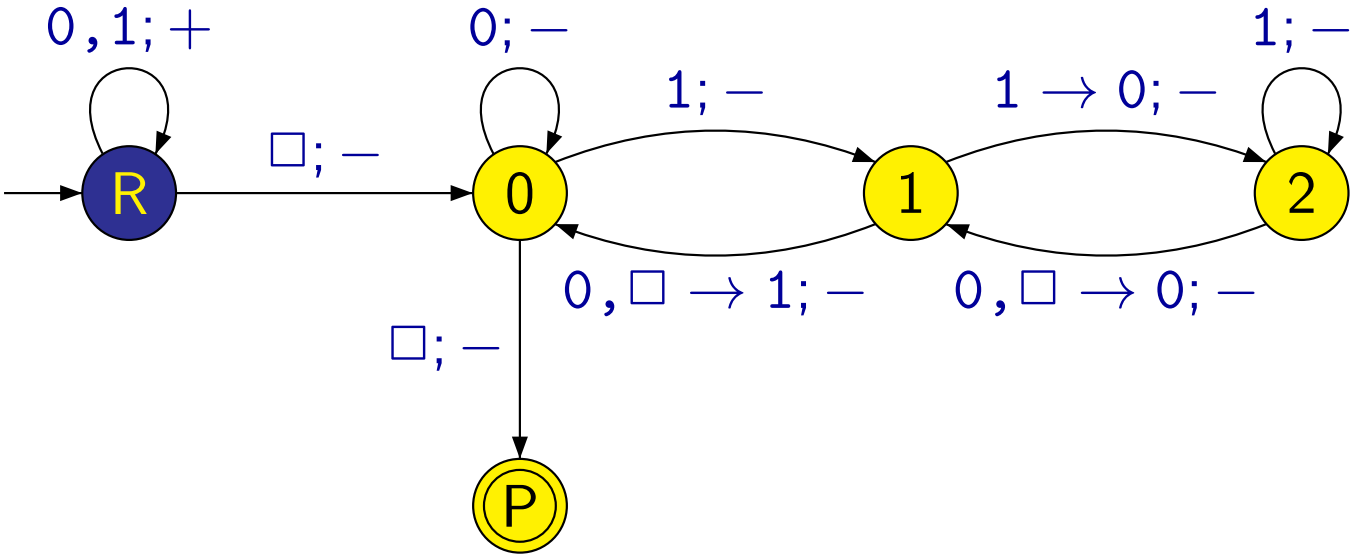


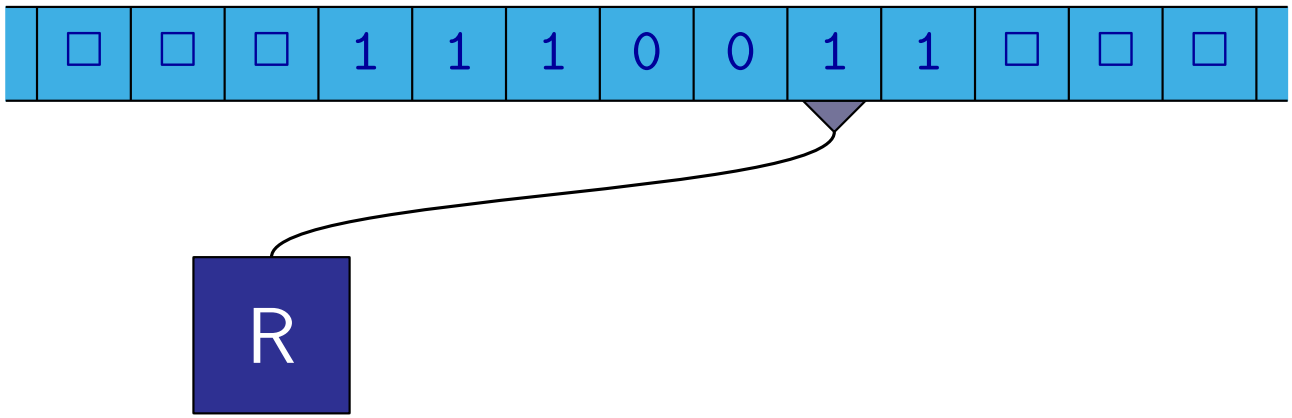
Turing Machine – Multiplication by Three



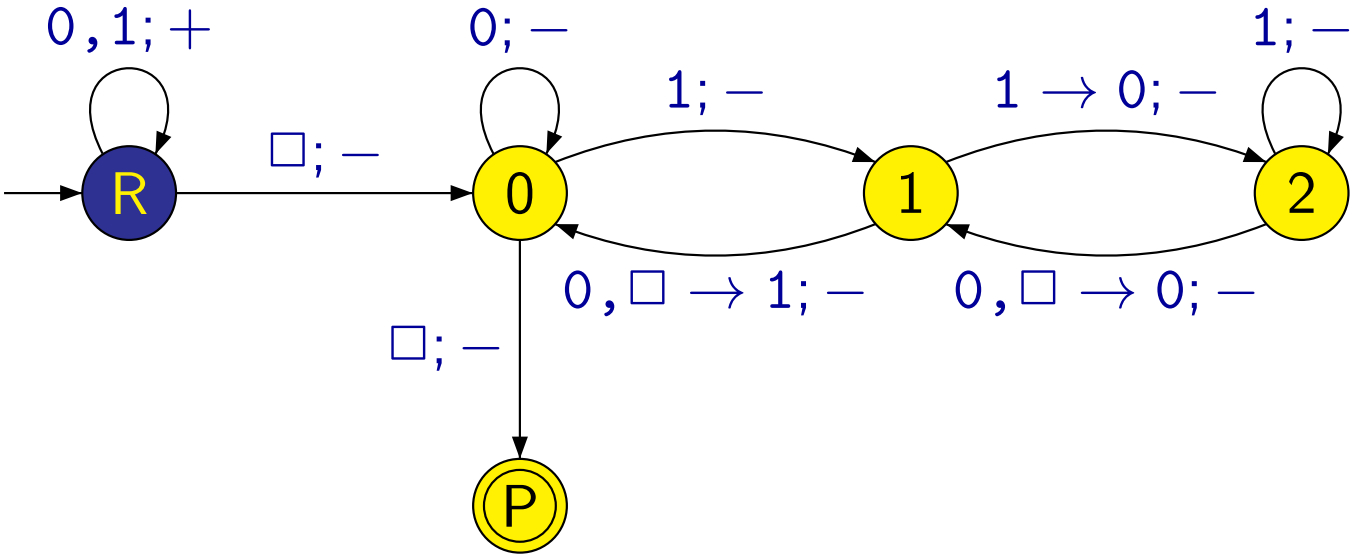


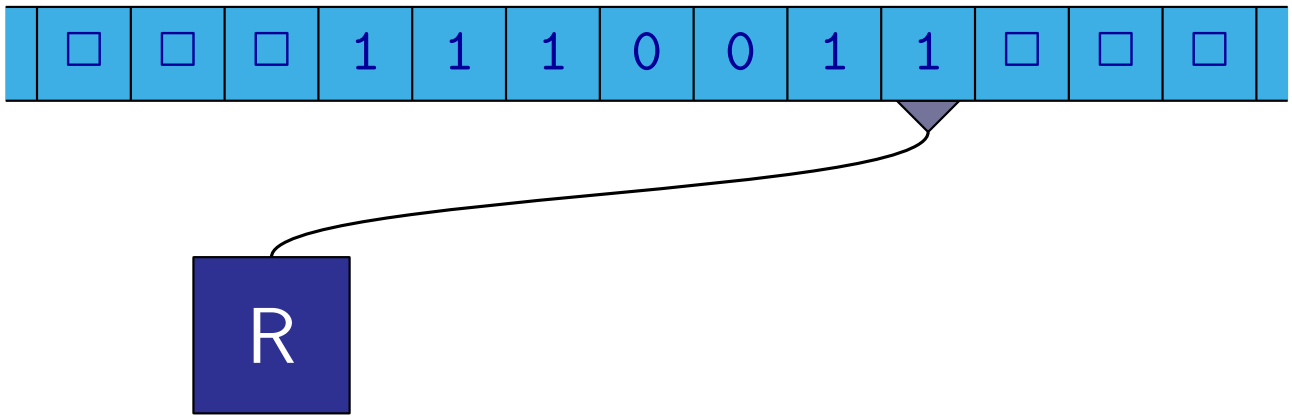
Turing Machine – Multiplication by Three



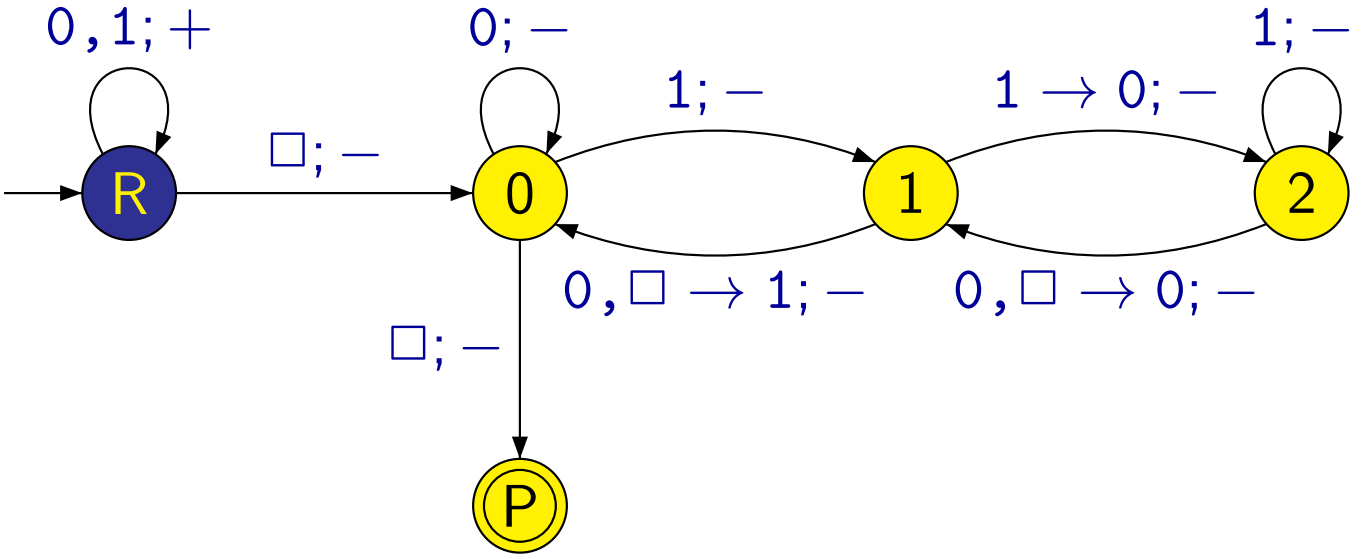


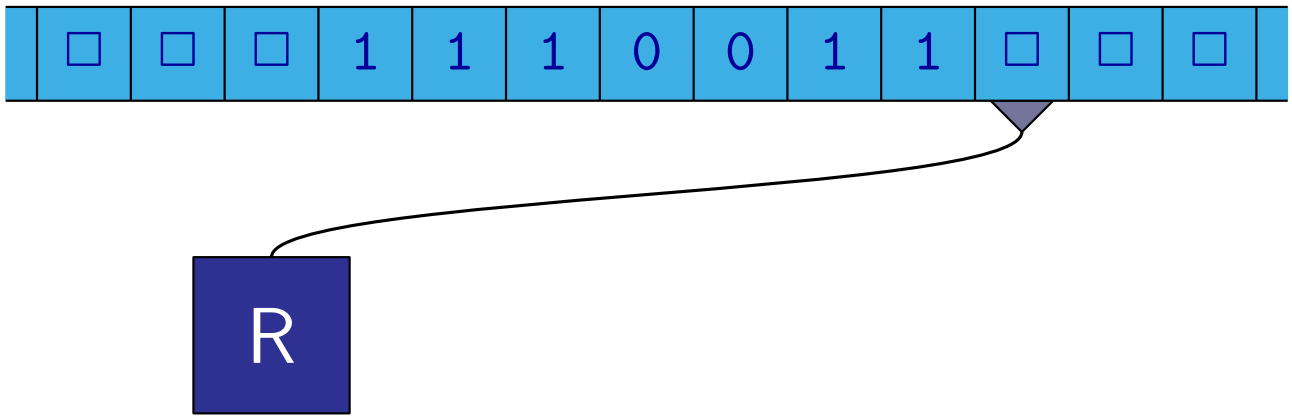
Turing Machine – Multiplication by Three



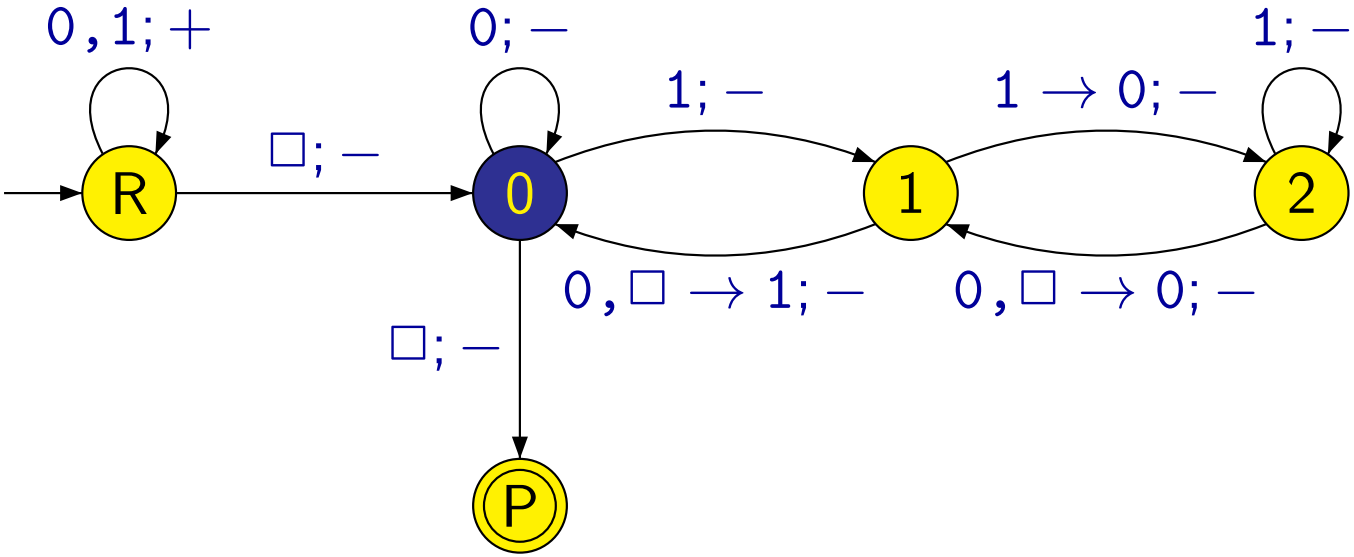


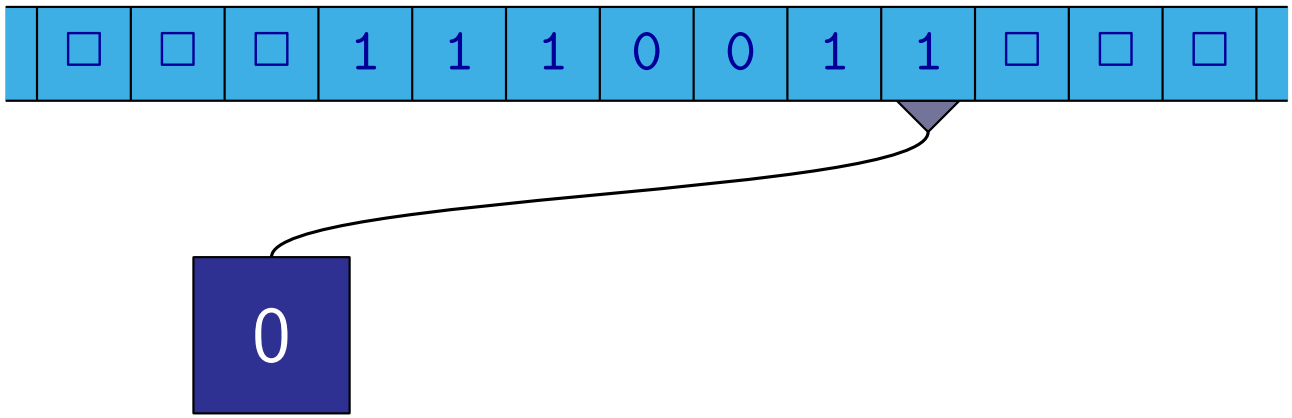
Turing Machine – Multiplication by Three



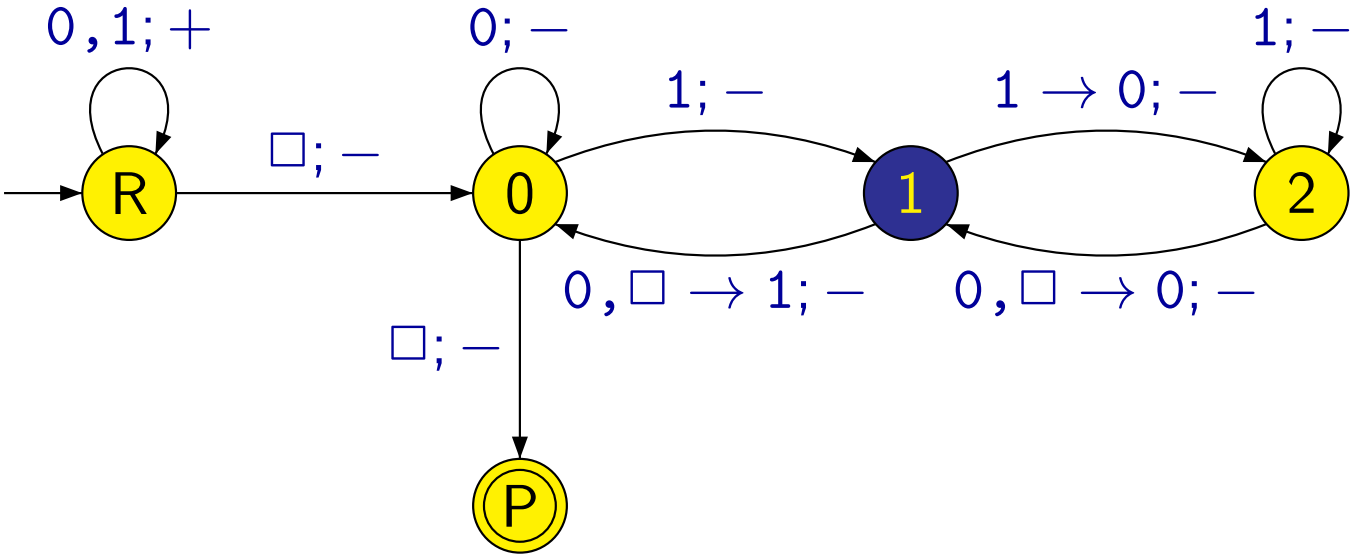


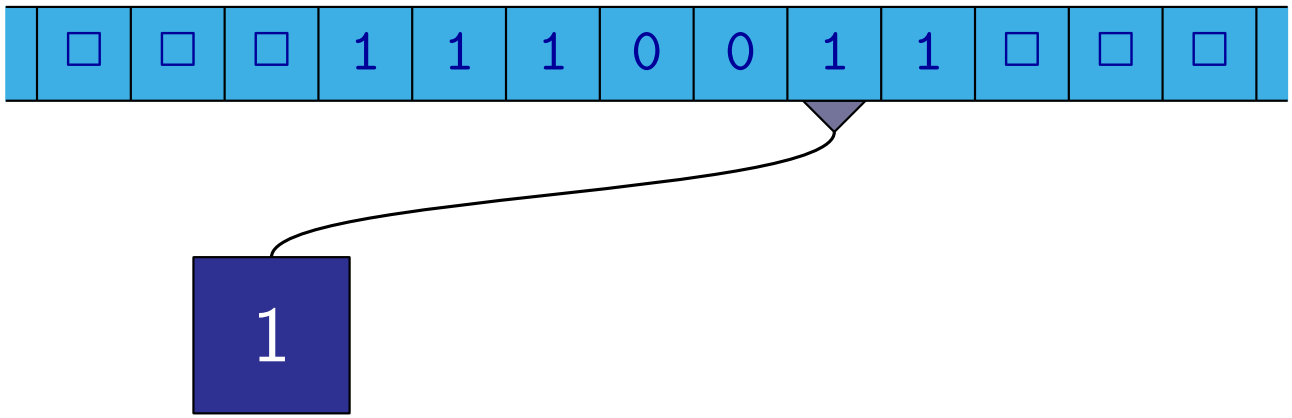
Turing Machine – Multiplication by Three



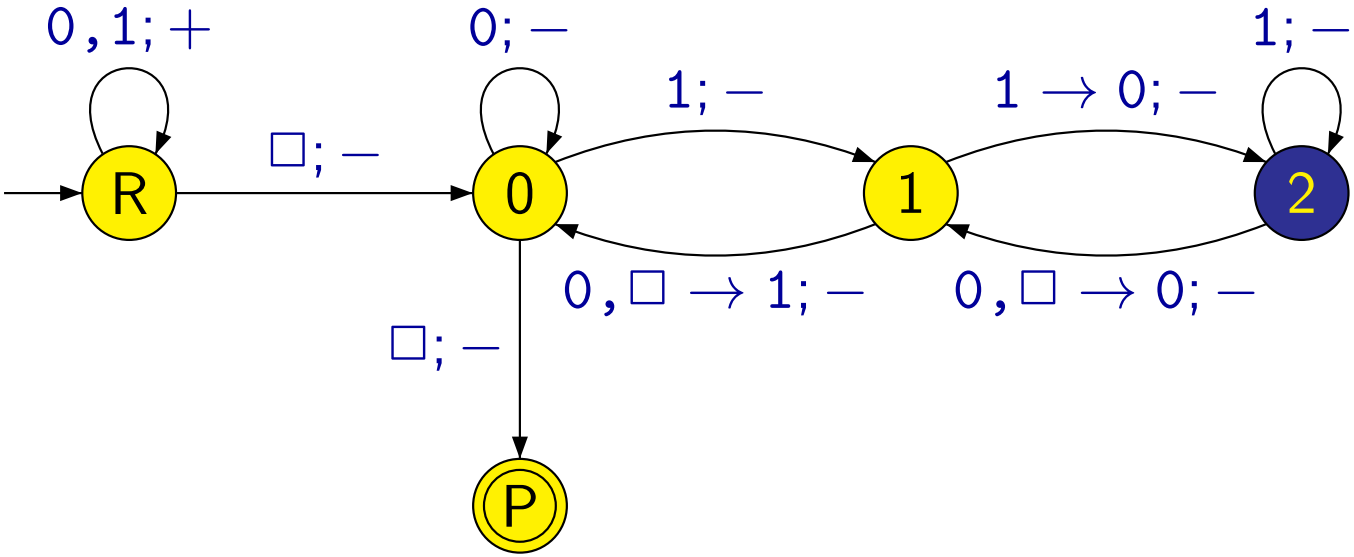


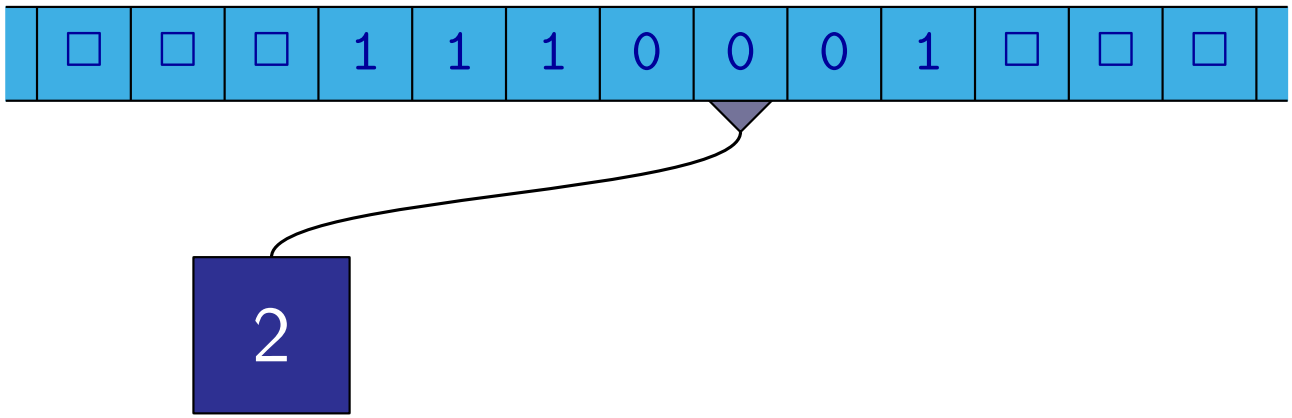
Turing Machine – Multiplication by Three



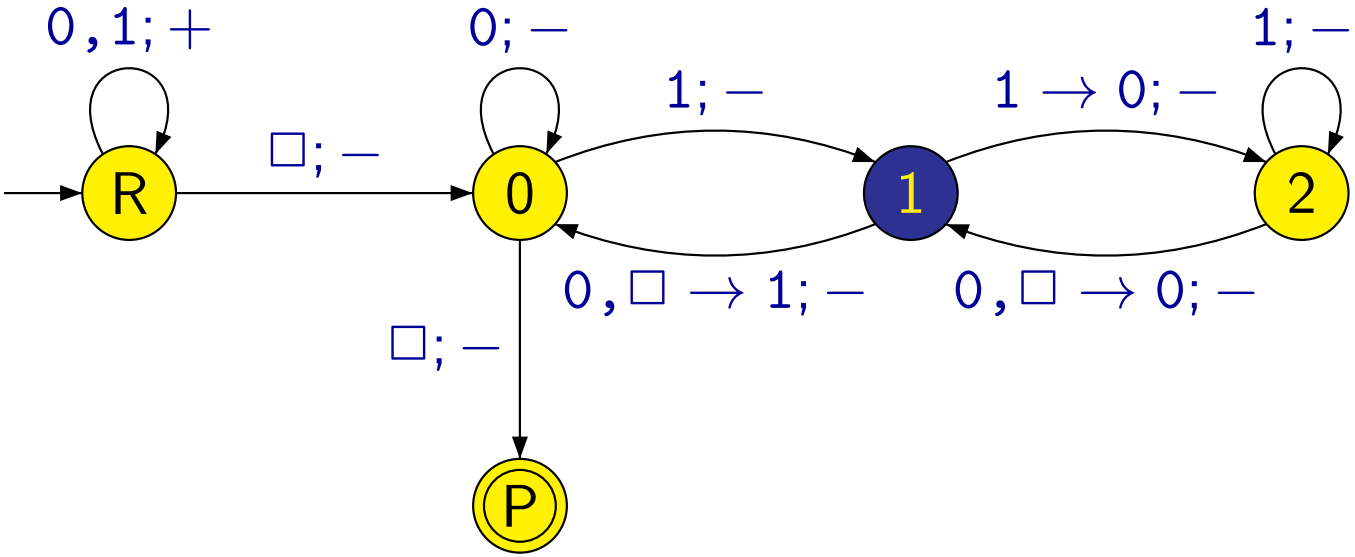


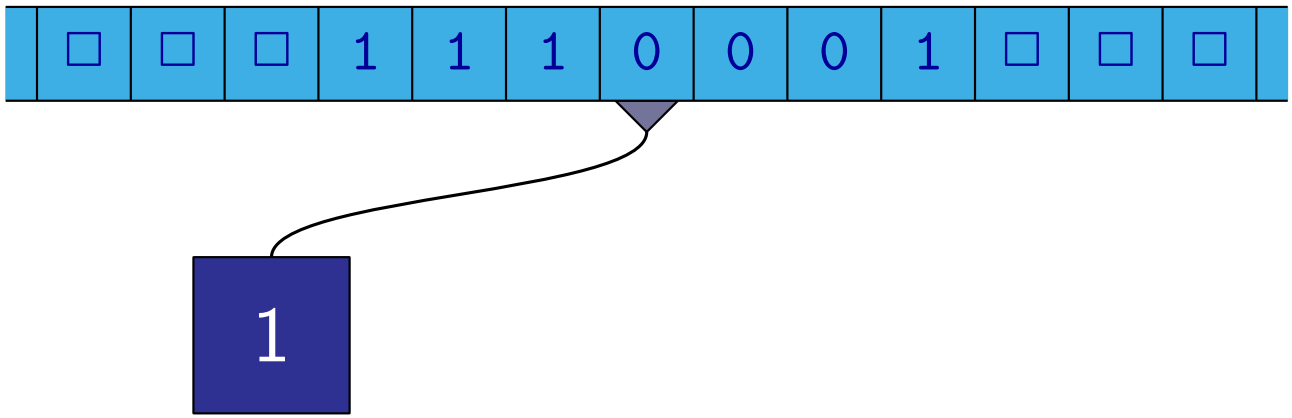
Turing Machine – Multiplication by Three



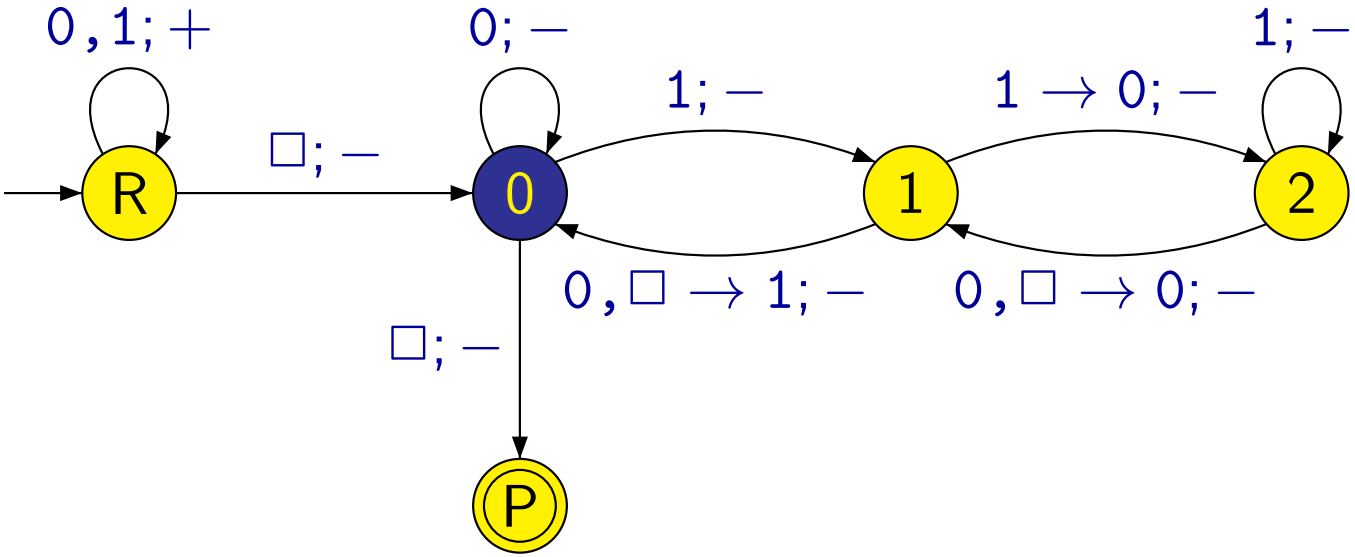


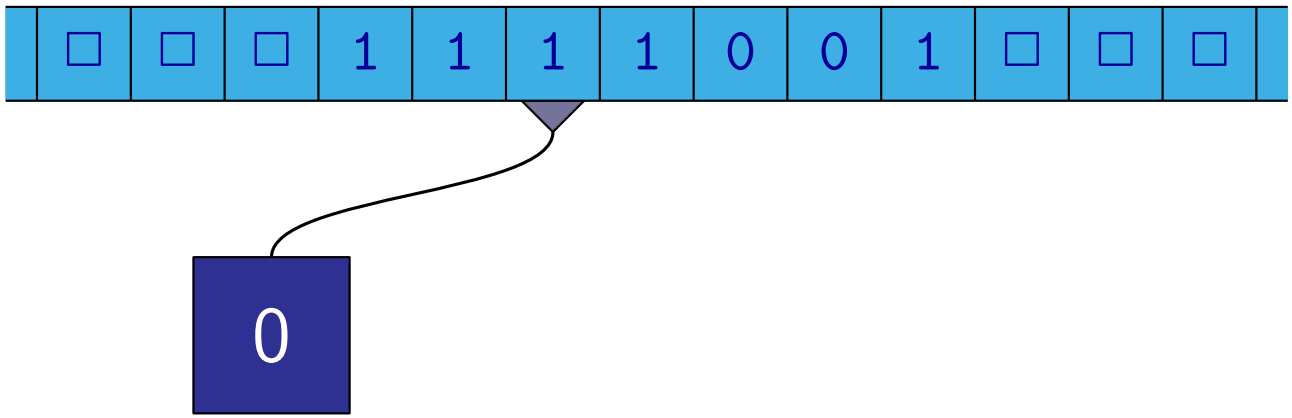
Turing Machine – Multiplication by Three



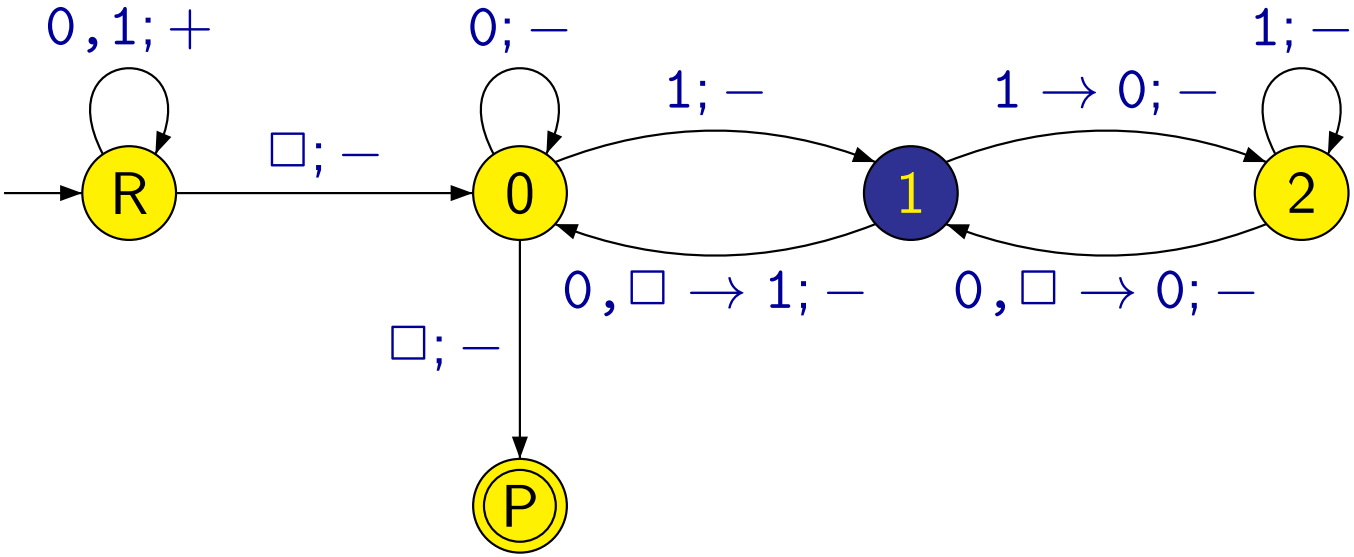


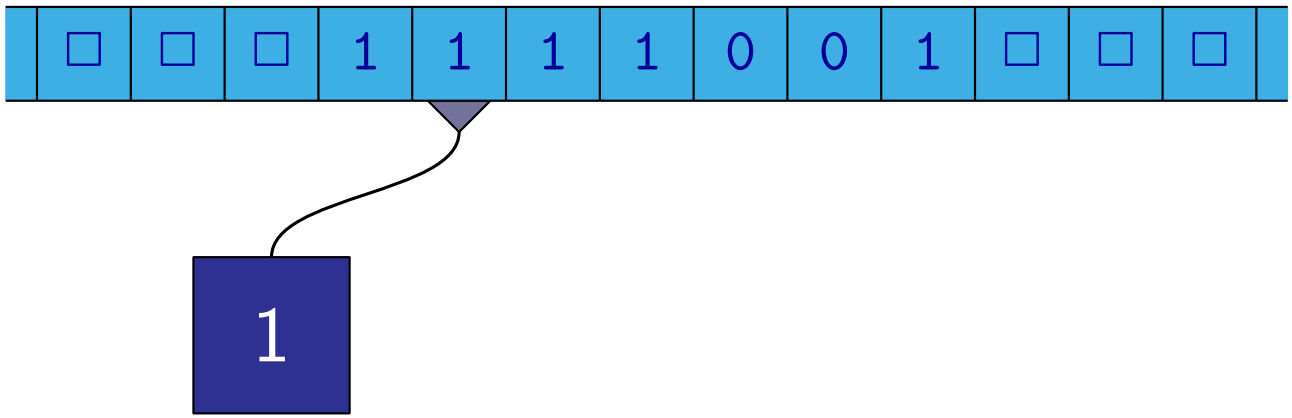
Turing Machine – Multiplication by Three



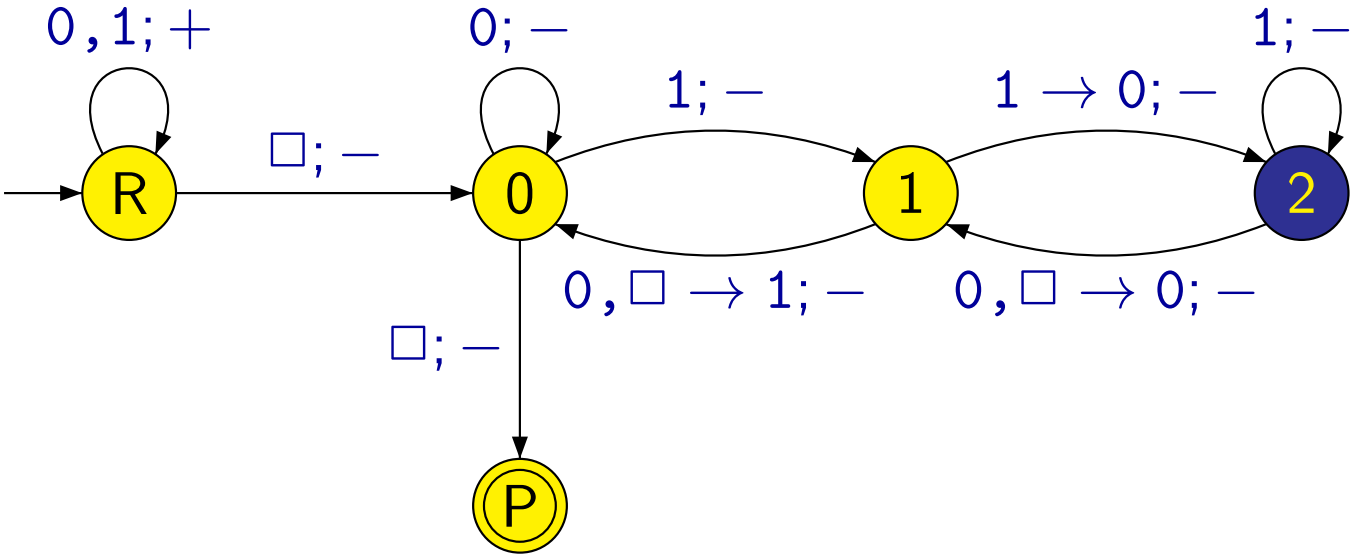


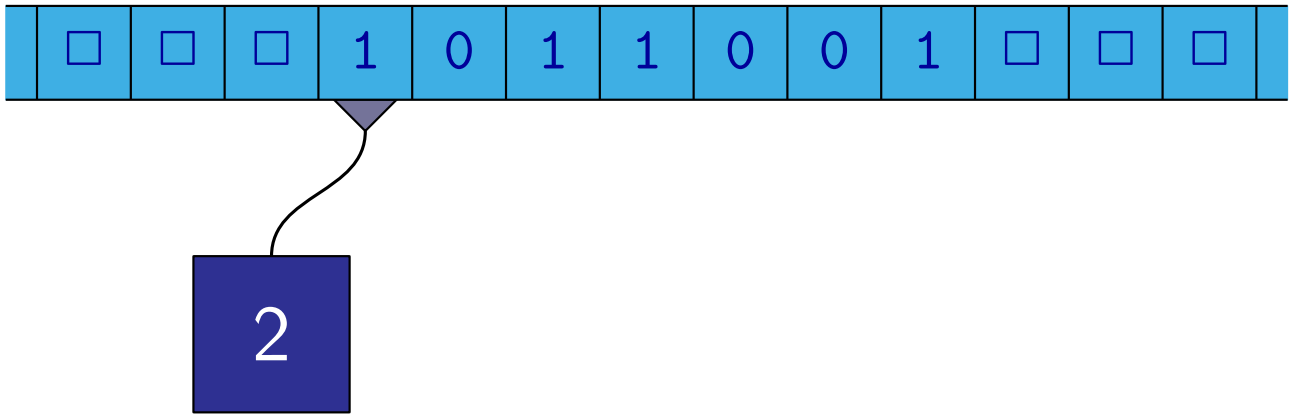
Turing Machine – Multiplication by Three



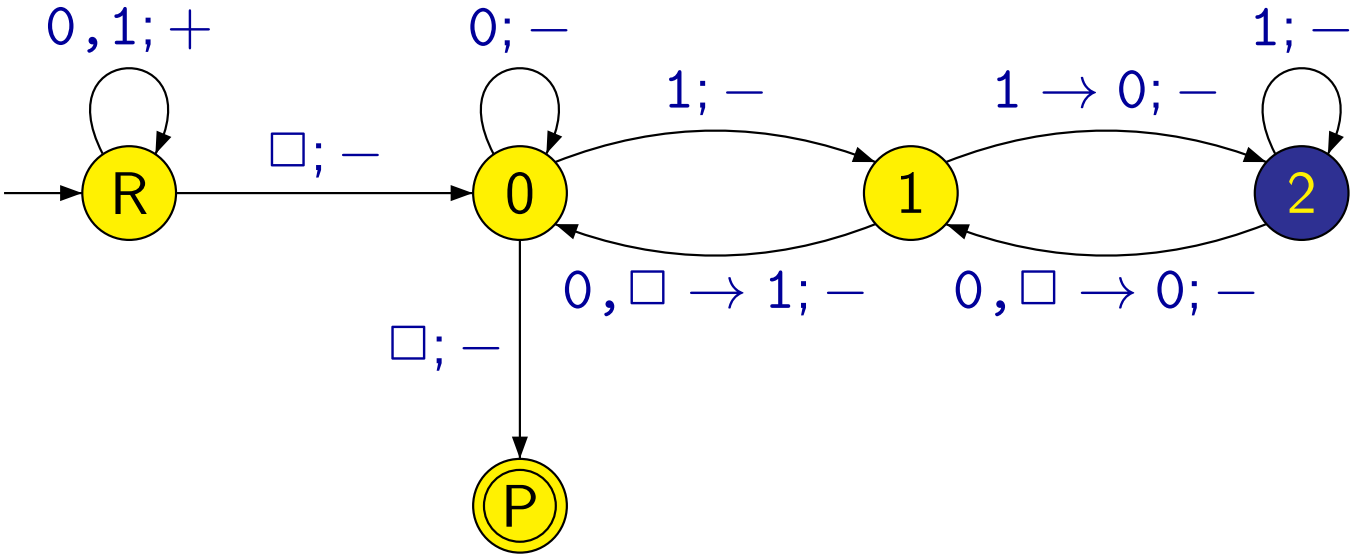


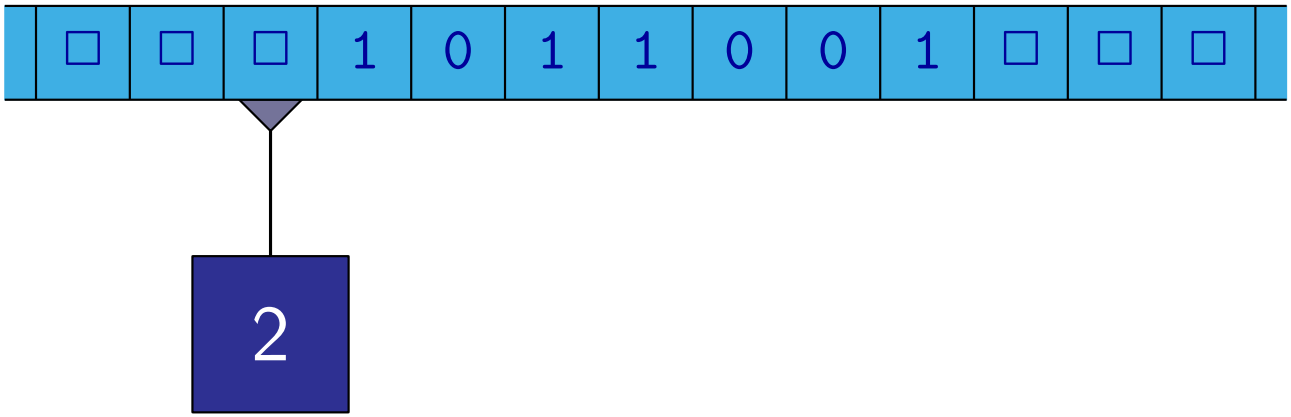
Turing Machine – Multiplication by Three



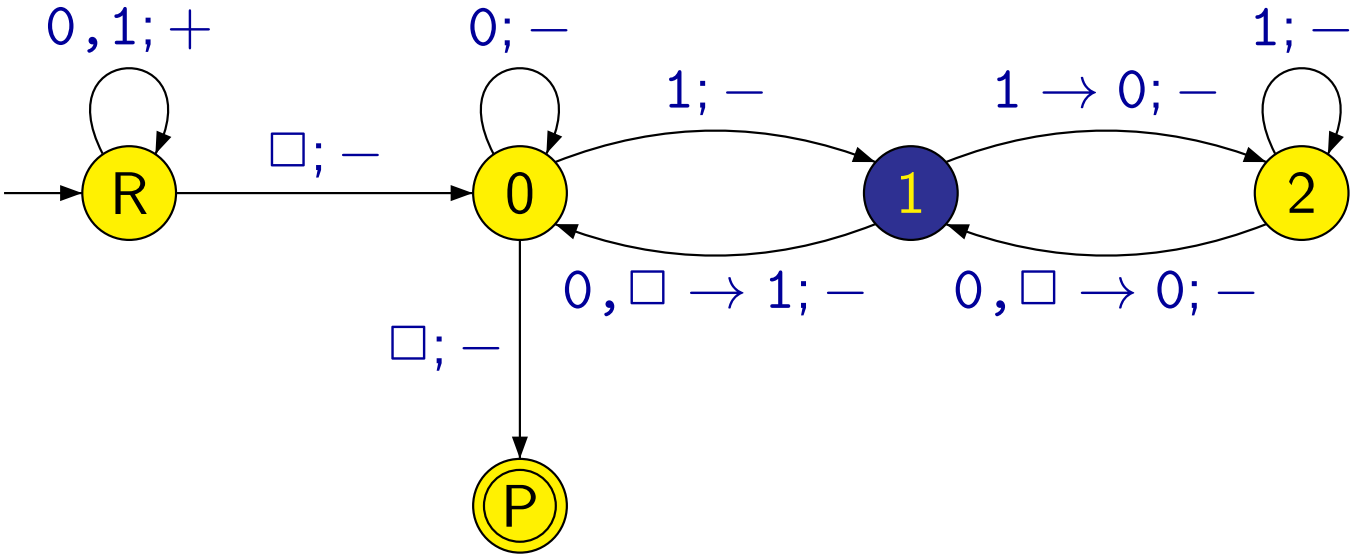


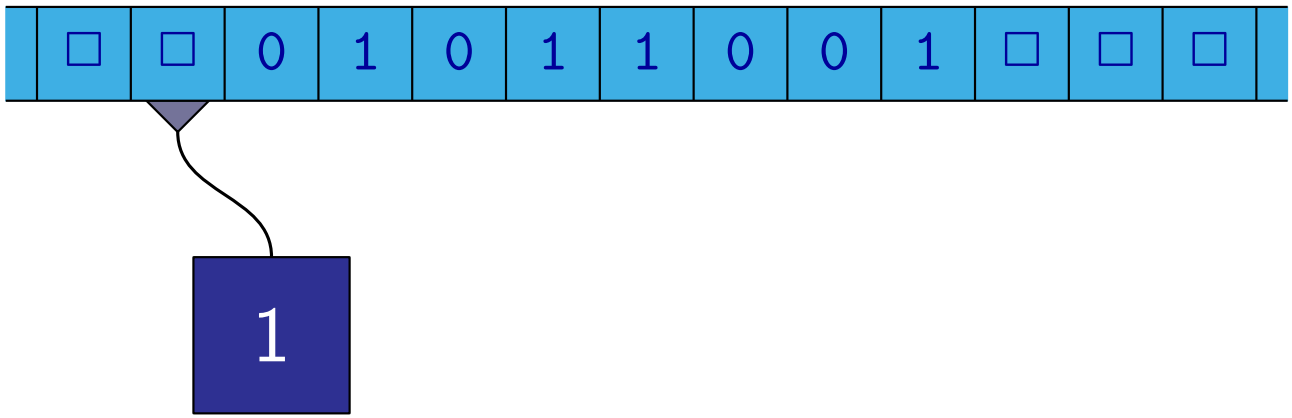
Turing Machine – Multiplication by Three



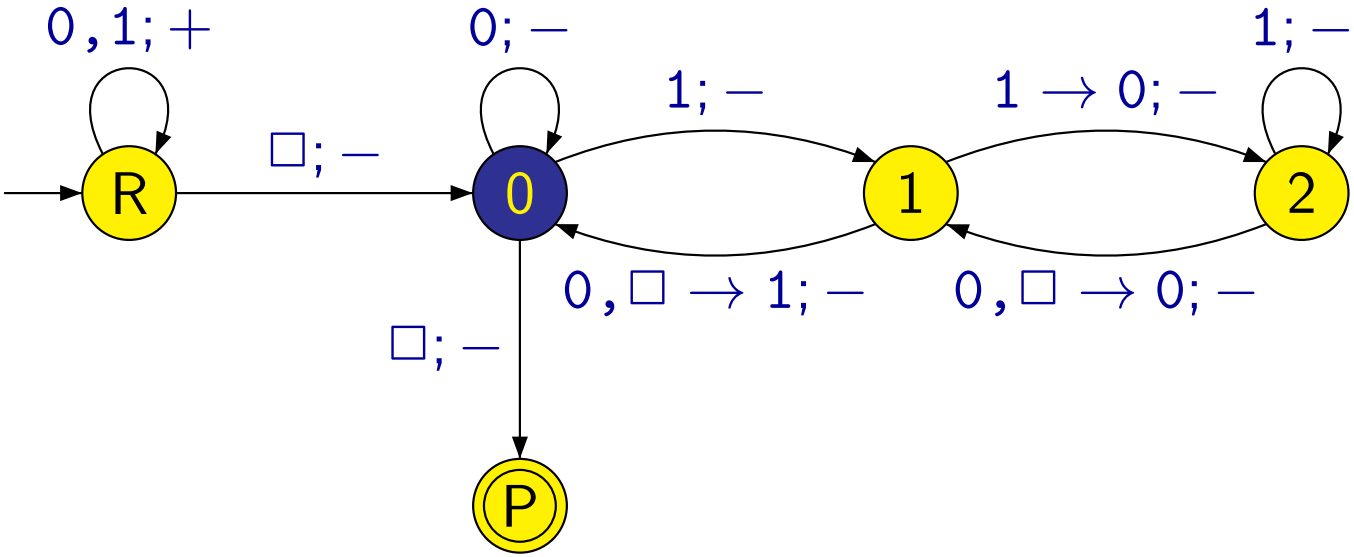


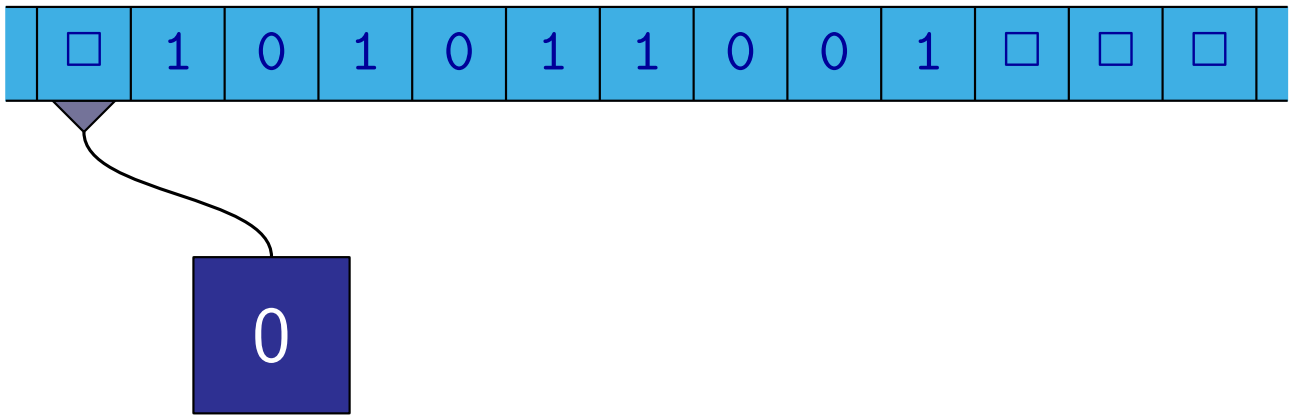
Turing Machine – Multiplication by Three



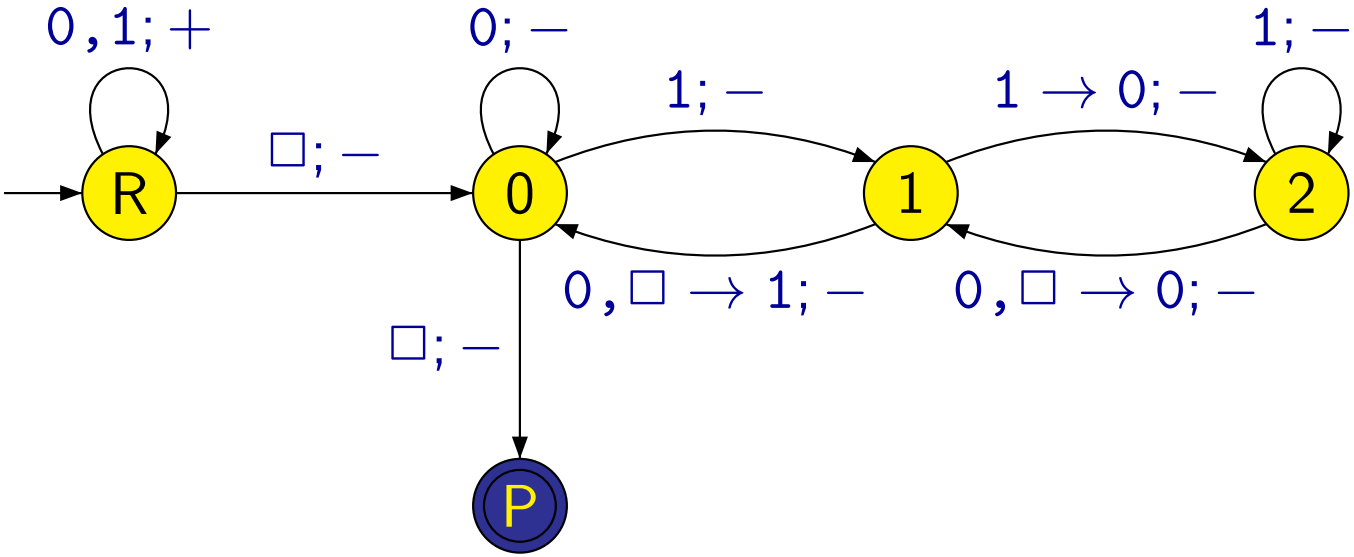


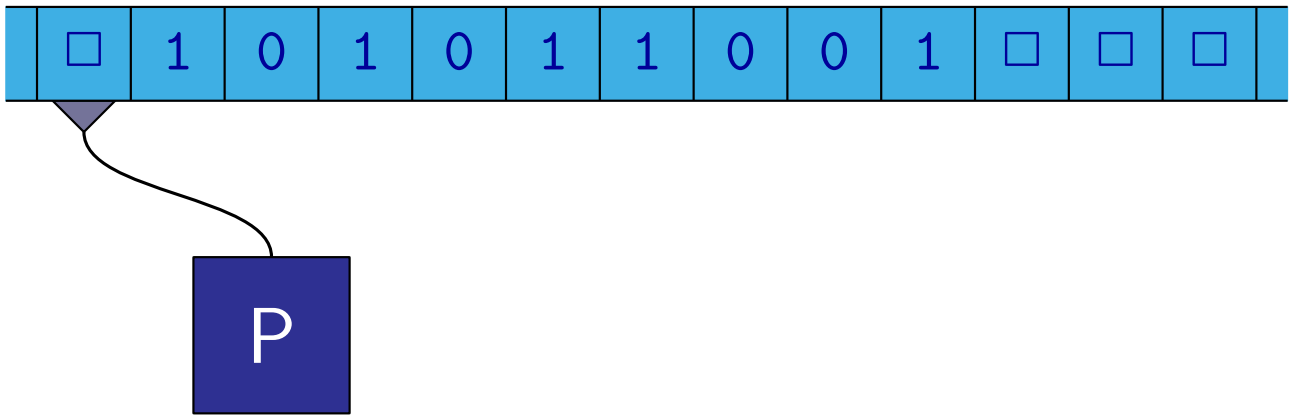
Turing Machine – Multiplication by Three





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 accepting w.

Remark: For every nondeterministic Turing machine, an equivalent deterministic Turing machine can be constructed.

Church-Turing Thesis

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:

- Random Access Machines
- Turing machines
- Lambda calculus
- Recursive functions
- ...

We can also mention:

- An arbitrary (general purpose) programming language (for example C, Java, Lisp, Haskell, Prolog, etc.).

All these models are equivalent with respect to algorithms that can be implemented by them.

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.

Time Complexity

Problem "Searching"

Input: An integer x and a sequence of integers $a_1, a_2, \ldots, a_n$ (where $a_i \neq 0$) terminated by 0.

Output: If $a_i = x$ then the output is i (if there are several such i then the smallest one), otherwise the output is 0.

```
start:  READ
        STORE   3
        LOAD    =1
 loop:  STORE   2
        READ
        JZERO   output
        SUB     3
        JZERO   found
        LOAD    2
        ADD     =1
        JUMP    loop
 found: LOAD    2
output: WRITE
        HALT
```

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0
⋮

Output:

Instructions: 0

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0
$\vdots$

Output:

Instructions: 0

Time Complexity

```
start:   READ
         STORE   3
         LOAD    =1
 loop:   STORE   2
         READ
         JZERO   output
         SUB     3
         JZERO   found
         LOAD    2
         ADD     =1
         JUMP    loop
 found:  LOAD    2
output:  WRITE
         HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0
⋮

Output:

Instructions: 1

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 0
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 2

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 1
Cell 1: 0
Cell 2: 0
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 3

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 1
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 4

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 13
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 5

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 13
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 6

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 4
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 7

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 4
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 8

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 1
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 9

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 10

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 11

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 12

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 5
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
$\vdots$

Output:

Instructions: 13

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0:	5
Cell 1:	0
Cell 2:	2
Cell 3:	9
Cell 4:	0
⋮	

Output:

Instructions: 14

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 15

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 16

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 17

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
$\vdots$

Output:

Instructions: 18

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 19

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 20

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 21

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 22

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 23

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 0
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 24

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output:

Instructions: 25

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output: 3

Instructions: 26

Time Complexity

```
start:  READ
        STORE  3
        LOAD   =1
 loop:  STORE  2
        READ
        JZERO  output
        SUB    3
        JZERO  found
        LOAD   2
        ADD    =1
        JUMP   loop
found:  LOAD   2
output: WRITE
        HALT
```

Input:
$9, 13, 5, 9, 7, 2, 0$

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
⋮

Output: 3

Instructions: 27

Time Complexity

- In this example, where the input is

$$9, 13, 5, 9, 7, 2, 0$$

the algorithm performed 27 instructions.

Time Complexity

- In this example, where the input is

$$9, 13, 5, 9, 7, 2, 0$$

the algorithm performed 27 instructions.

We can try to analyze how many instructions the algorithm performes in general for an input

$$x, a_1, a_2, \ldots, a_n, 0$$

Time Complexity

- In this example, where the input is

$$9, 13, 5, 9, 7, 2, 0$$

the algorithm performed 27 instructions.

We can try to analyze how many instructions the algorithm performes in general for an input

$$x, a_1, a_2, \ldots, a_n, 0$$

- If for all i (where $1 \leq i \leq n$) is $x \neq a_i$:

$$3 + 8 \cdot n + 3 + 2 = 8n + 8$$

Time Complexity

- In this example, where the input is

$$9, 13, 5, 9, 7, 2, 0$$

the algorithm performed 27 instructions.

We can try to analyze how many instructions the algorithm performes in general for an input

$$x, a_1, a_2, \ldots, a_n, 0$$

- If for all i (where $1 \leq i \leq n$) is $x \neq a_i$:

$$3 + 8 \cdot n + 3 + 2 = 8n + 8$$

- If for some i (where $1 \leq i \leq n$) is $x = a_i$:

$$3 + 8 \cdot (i - 1) + 5 + 3 = 8i + 3$$

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).

Example: For problem "Searching" where all inputs are of the form

$$x, a_1, a_2, \ldots, a_n, 0$$

we can choose the number n as the size of the input.

This way the size of the input $9, 13, 5, 9, 7, 2, 0$ is 5.

Size of Input

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.

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 $1, 2, 3, 4, \ldots$.

It is obvious that even for inputs of the same size the number of performed steps can be different.

Let us assume that X is the set of all possible inputs of the given problem, and $g(x)$ is the number of steps perfomed by the algorithm for an input $x \in X$.

Let us denote the size of input $x \in X$ as $|x|$.

Now we define a function $f : \mathbb{N} \to \mathbb{N}$ such that for $n \in \mathbb{N}$ is

$$f(n) = \max\{g(x) \mid x \in X, |x| = n\}$$

Time Complexity in the Worst Case

Such function $f(n)$ (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**.

Time Complexity in the Worst Case

Such function $f(n)$ (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**.

Example: As we found out, the previously described algorithm solving the problem "Searching" performs for an input $x, a_1, a_2, \ldots, a_n, 0$ the following number of steps:

- If for all i it holds that $x \neq a_i$, then the algorithm performs $8n + 8$ steps.
- If for some i it holds that $x = a_i$, then the algorithm performs $8i + 3$ steps.

Since in the second case it is always $i \leq n$, the time complexity of the algorithm in the worst case is $f(n) = 8n + 8$.

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 $f(n)$ as the maximum but as the arithmetic mean of the set

$$\{g(x) \mid x \in X, |x| = n\}$$

- 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.

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 $f(n)$ operations and that an execution of one operation takes $1\,\mu s$ ($10^{-6}\,s$).

	n							
$f(n)$	20	40	60	80	100	200	500	1000
n	$20\,\mu s$	$40\,\mu s$	$60\,\mu s$	$80\,\mu s$	0.1 ms	0.2 ms	0.5 ms	1 ms
$n \log n$	$86\,\mu s$	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	$38.3 \cdot 10^9$ years	$40.1 \cdot 10^{15}$ years	$50 \cdot 10^{45}$ years	$10.4 \cdot 10^{136}$ years	–
$n!$	77147 years	$2.59 \cdot 10^{34}$ years	$2.64 \cdot 10^{68}$ years	$2.27 \cdot 10^{105}$ years	$2.96 \cdot 10^{144}$ 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

- It is usually quite difficult to express the complexity exactly.
- The exact complexity depends on the used model of computation and on the particular implementation (on details of this implementation).
- We are interested in the complexity for big inputs. For small inputs usually even nonefficient algorithms work fast.
- We usually do not need to know the exact number of performed instructions and we will be satisfied with some estimation of how fast this number grows when the size of an input grows.
- So we use the so called **asymptotic notation**, which allows us to ignore unimportant details and to estimate approximately how fast the given function grows. This simplifies the analysis considerably.

Asymptotic Notation

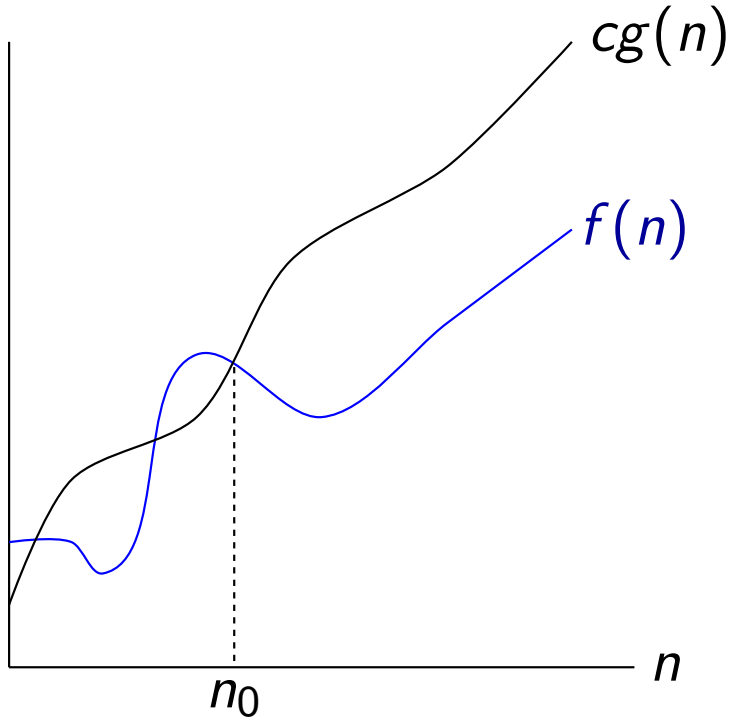
Let us take an arbitrary function $g : \mathbb{N} \to \mathbb{N}$. Expressions $O(g)$, $\Omega(g)$, and $\Theta(g)$ denote **sets of functions** of the type $\mathbb{N} \to \mathbb{N}$, where:

- $O(g)$ – 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

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"

Asymptotic Notation – Symbol O

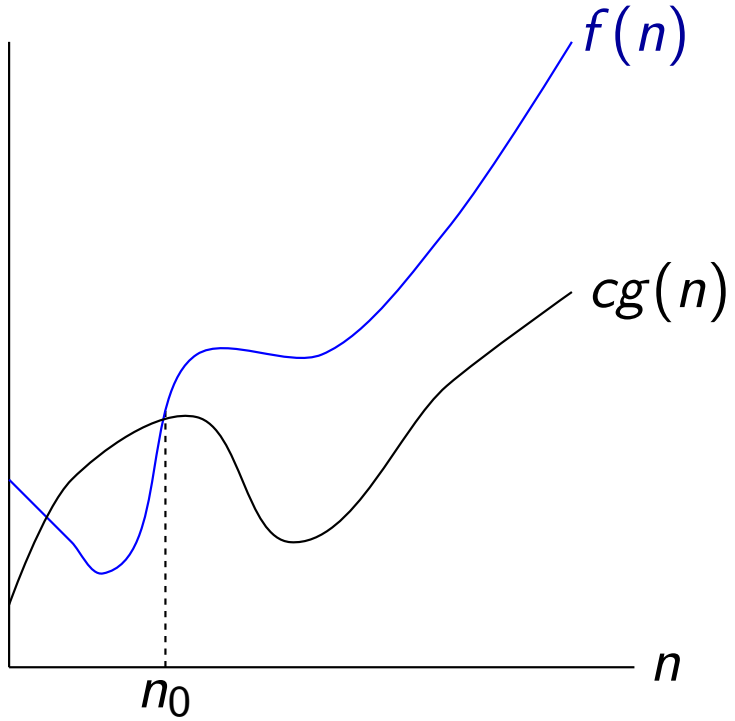


Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{N}$. For a function $f : \mathbb{N} \to \mathbb{N}$ we have $f \in O(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : f(n) \leq c\, g(n).$$

Asymptotic Notation – Symbol Ω

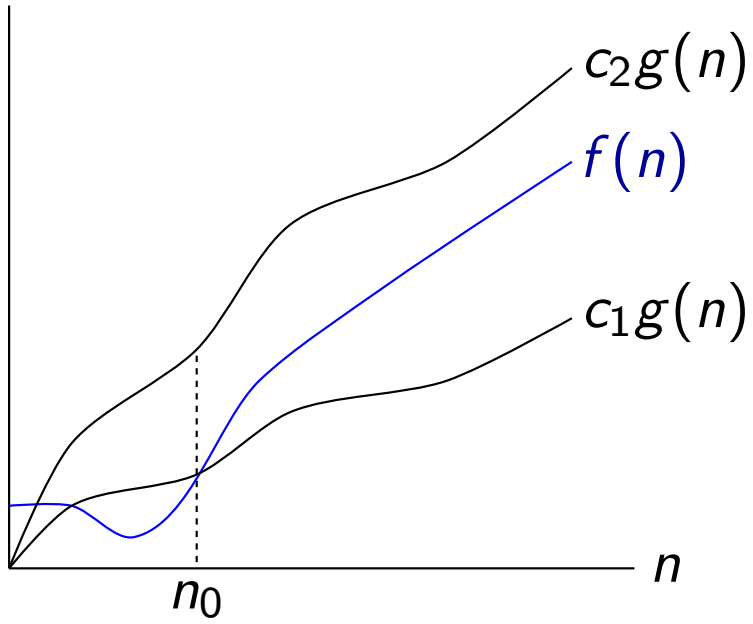


Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{N}$. For a function $f : \mathbb{N} \to \mathbb{N}$ we have $f \in \Omega(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : c\, g(n) \leq f(n).$$

Asymptotic Notation – Symbol Θ



Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{N}$. For a function $f : \mathbb{N} \to \mathbb{N}$ we have $f \in \Theta(g)$ iff

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : c_1\, g(n) \leq f(n) \leq c_2\, g(n).$$

Asymptotic Notation

For simplicity, we consider only functions of type $\mathbb{N} \to \mathbb{N}$ in the previous definitions.

In fact, these definitions could be extended to all **asymptotically nonnegative** functions of type $\mathbb{R}_+ \to \mathbb{R}$, which moreover can be undefined on some finite subinterval of its domain.

Function $f : \mathbb{R}_+ \to \mathbb{R}$ is **asymptotically nonnegative** if it satisfies:

$$(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \geq 0)$$

Remark: For $n < n_0$, the value of $f(n)$ can be undefined.

$\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$

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 any function $g : \mathbb{N} \to \mathbb{N}$ we have:

$$g \in O(g) \qquad g \in \Omega(g) \qquad g \in \Theta(g)$$

- For any pair of functions $f, g : \mathbb{N} \to \mathbb{N}$ we have:
  - $f \in O(g)$ iff $g \in \Omega(f)$
  - $f \in \Theta(g)$ iff $g \in \Theta(f)$
  - $f \in \Theta(g)$ iff $f \in O(g)$ and $f \in \Omega(g)$

- For any functions $f, g, h : \mathbb{N} \to \mathbb{N}$ 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

- There are pairs of functions $f, g : \mathbb{N} \to \mathbb{N}$ such that

  $$f \notin O(g) \qquad \text{and} \qquad g \notin O(f),$$

  for example

  $$f(n) = n \qquad\qquad g(n) = n^{1+\sin(n)}.$$

- $O(1)$ denotes the set of all **bounded** functions, i.e., functions whose function values can be bounded from above by a constant.

Asymptotic Notation

- For any pair of functions $f, g : \mathbb{N} \to \mathbb{N}$ we have:
  - $\max(f, g) \in \Theta(f + g)$
  - if $f \in O(g)$ then $f + g \in \Theta(g)$
- For any functions $f_1, f_2, g_1, g_2 : \mathbb{N} \to \mathbb{N}$ 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

- A function f is called:

  logarithmic, if $f(n) \in \Theta(\log n)$
  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 $k > 0$

- 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

- For any $k, \ell > 0$, such that $k < \ell$, and any $c > 1$ we have:

$$n^k \in O(n^\ell) \qquad n^\ell \notin O(n^k)$$

$$\log_c^k n \in O(\log_c^\ell n) \qquad \log_c^\ell n \notin O(\log_c^k n)$$

$$c^{n^k} \in O(c^{n^\ell}) \qquad c^{n^\ell} \notin O(c^{n^k})$$

- For any $k, \ell > 0$ and any $c > 1$ we have:

$$\log_c^k n \in O(n^\ell) \qquad n^\ell \notin O(\log_c^k n)$$

$$n^k \in O(c^{n^\ell}) \qquad c^{n^\ell} \notin O(n^k)$$

Remark: $\log_c^k n$ is a more succinct notation for $(\log_c n)^k$.

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 $\log_b n = \log_a n \cdot \log_b n$, from which the above mentioned conclusion follows directly. □

So for any constants $a, b > 1$ we have $\log_a n \in \Theta(\log_b n)$.

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

As mentioned before, expressions $O(g)$, $\Omega(g)$, and $\Theta(g)$ denote certain sets of functions.

In some texts, these expressions are sometimes used with a slightly different meaning:

- an expression $O(g)$, $\Omega(g)$ or $\Theta(g)$ 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 $f \in O(g)$.

Asymptotic Notation

- The asymptotic notation can be straightforwardly generalized to functions of several arguments:

  Let us consider, for example, a function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. For function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ we have $f(n, m) \in O(g(n, m))$ iff

  $$(\exists c > 0)(\exists n_0 \geq 0)(\exists m_0 \geq 0)(\forall n \geq n_0)(\forall m \geq m_0)(f(n, m) \leq c\, g(n, m))$$

- In addition to expressions $O(g)$, $\Omega(g)$, and $\Theta(g)$, expressions $o(g)$ and $\omega(g)$ are also sometimes used.
  - $o(g)$ – the set of all fuctions that grow slower than function g
  - $\omega(g)$ – the set of all functions that grow faster than function g

  We will not discuss their exact definitions and we will not deal with these expressions any further.

Complexity of Algorithms

Let us say we would like to analyze the time complexity $t(n)$ of some algorithm consisting of instructions $I_1, I_2, \ldots, I_k$:

- If $m_1, m_2, \ldots, m_k$ are the number of executions of individual instructions for some input x (i.e., the instruction I_i is performed m_i times for the input x), then the total number of executed instructions for input x is
$$m_1 + m_2 + \cdots + m_k.$$
- Let us consider functions $f_1, f_2, \ldots, f_k$, where $f_i : \mathbb{N} \to \mathbb{N}$, and where $f_i(n)$ is the maximum of numbers of executions of instruction I_i for all inputs of size n.
- Obviously, $t \in \Omega(f_i)$ for any function f_i.
- It is also obvious that $t \in O(f_1 + f_2 + \cdots + f_k)$.

Complexity of Algorithms

- Let us recall that if $f \in O(g)$ then $f + g \in O(g)$.
- 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
$$t \in O(f_i).$$
- This means that in an analysis of the time complexity $t(n)$, we can restrict our attention to the number of executions of the instruction that is performed most frequently (and which is performed at most $f_i(n)$ times for an input of size n), since we have
$$t \in \Theta(f_i).$$

Complexity of Algorithms

Example: In the analysis of the complexity of the searching of a number in a sequence we obtained

$$f(n) = 8n + 8\,.$$

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 the only cycle, which is performed at most n times, and there are some inputs, for which it is actually performed n times.
- 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 indepent of the size of the input.
- Other instructions are performed at most once, and so they contribute to the total running time by adding a constant.

Complexity of Algorithms

Let us try to analyze the time complexity of the following algorithm:

INSERTION-SORT(A, n):

for $j := 2$ **to** n **do**
  $x := A[j]$
  $i := j - 1$
  while $i > 0$ and $A[i] > x$ **do**
    $A[i+1] := A[i]$
    $i := i - 1$
  end while
  $A[i+1] := x$
end for

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

Let us consider inputs of size n:

- The outer cycle **for** is performed at most $n-1$ times.
- The inner cycle **while** is performed at most $(j-1)$ times for a given value j.
- There are inputs such that the cycle **while** is performed exactly $(j-1)$ times for each value j from 2 to n.
- So in the worst case, the cycle **while** is performed exactly m times, where

$$m = 1 + 2 + \cdots + (n-1) = (1 + (n-1)) \cdot \tfrac{n-1}{2} = \tfrac{1}{2}n^2 - \tfrac{1}{2}n$$

- This means that the total running time of the algorithm INSERTION-SORT in the worst case is $\Theta(n^2)$.

Complexity of Algorithms

In the previous case, we accurately computed the total number of executions of the cycle **while**.

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

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

- So far we have considered only the case when the execution of all instructions takes the same time.
- In practice, some of instructions can be more time consuming than others.
- If we know the times of an execution for each instruction and if we can count the number of their executions, we can compute the overall running time as

$$\sum_i m_i t_i$$

where n_i is the number of executions of instruction i and t_i is the time of one exection of i.

Complexity of Algorithms

Let us assume that we analyze the time complexity of some algorithm implemented as a RAM.

- So far, we have counted only the number of executed instructions in our analysis.
  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).

- 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.

Complexity of Algorithms

- 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 `ADD` and `SUB` is equal to the sum of the numbers of bits of their operands.
- The duration of an execution of instructions `MUL` and `DIV` is equal to the sum of the numbers of bits of their operands.

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.

Amount of memory of a RAM $\mathcal{M}$ used for an input x is the number of memory cells that are used by $\mathcal{M}$ during its computation on x.

Definition

A **space complexity** of a RAM $\mathcal{M}$ (in the worst case) is the function $s : \mathbb{N} \to \mathbb{N}$, 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 one cell beside the accumulator in each step).

Complexity of Problems

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 how many steps at most the given algorithm performs for an input 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.

Complexity Classes

Definition

For every function $f : \mathbb{N} \to \mathbb{N}$ we define $\mathcal{T}(f(n))$ as the class containing exactly those problems for which there exists an algorithm with time complexity $O(f(n))$.

Example:

- $\mathcal{T}(n)$ – the class of all problems for which there exists an algorithm with time complexity $O(n)$
- $\mathcal{T}(n^2)$ – the class of all problems for which there exists an algorithm with time complexity $O(n^2)$
- $\mathcal{T}(n \log n)$ – the class of all problems for which there exists an algorithm with time complexity $O(n \log n)$

Complexity Classes

Definition

For every function $f : \mathbb{N} \to \mathbb{N}$ we define $\mathcal{S}(f(n))$ as the class containing exactly those problems for which there exists an algorithm with space complexity $O(f(n))$.

Example:

- $\mathcal{S}(n)$ – the class of all problems for which there exists an algorithm with space complexity $O(n)$
- $\mathcal{S}(n^2)$ – the class of all problems for which there exists an algorithm with space complexity $O(n^2)$
- $\mathcal{S}(n \log n)$ – the class of all 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 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 all 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 problems for which there exists an algorithm with time complexity $2^{O(n^k)}$ where k is a constant

EXPSPACE – the set of all problems for which there exists an algorithm with space complexity $2^{O(n^k)}$ where k is a constant

LOGSPACE – the set of all 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.

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:

$$\text{PTIME} \subseteq \text{PSPACE}$$
$$\text{EXPTIME} \subseteq \text{EXPSPACE}$$

Since polynomial functions grow more slowly than exponential, we obviously have:

$$\text{PTIME} \subseteq \text{EXPTIME}$$

$$\text{LOGSPACE} \subseteq \text{PSPACE} \subseteq \text{EXPSPACE}$$

Relationships between Complexity Classes

- For every pair of real numbers $0 \leq \epsilon_1 < \epsilon_2$ is

$$\mathcal{S}(n^{\epsilon_1}) \subsetneq \mathcal{S}(n^{\epsilon_2})$$

- LOGSPACE $\subsetneq$ PSPACE
- PSPACE $\subsetneq$ EXPSPACE

- For every pair of real numbers $0 \leq \epsilon_1 < \epsilon_2$ is

$$\mathcal{T}(n^{\epsilon_1}) \subsetneq \mathcal{T}(n^{\epsilon_2})$$

- PTIME $\subsetneq$ 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 $\mathcal{S}(f(n)) \subseteq \mathcal{T}(2^{f(n)})$.

Relationships between Complexity Classes

The following results can be drawn from the previous discussion:

$$\text{LOGSPACE} \subseteq \text{PTIME}$$
$$\text{PSPACE} \subseteq \text{EXPTIME}$$

Relationships between Complexity Classes

Summary:

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$$

- $\text{PTIME} \subsetneq \text{EXPTIME}$
- $\text{LOGSPACE} \subsetneq \text{PSPACE}$

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.)

Nondeterminism

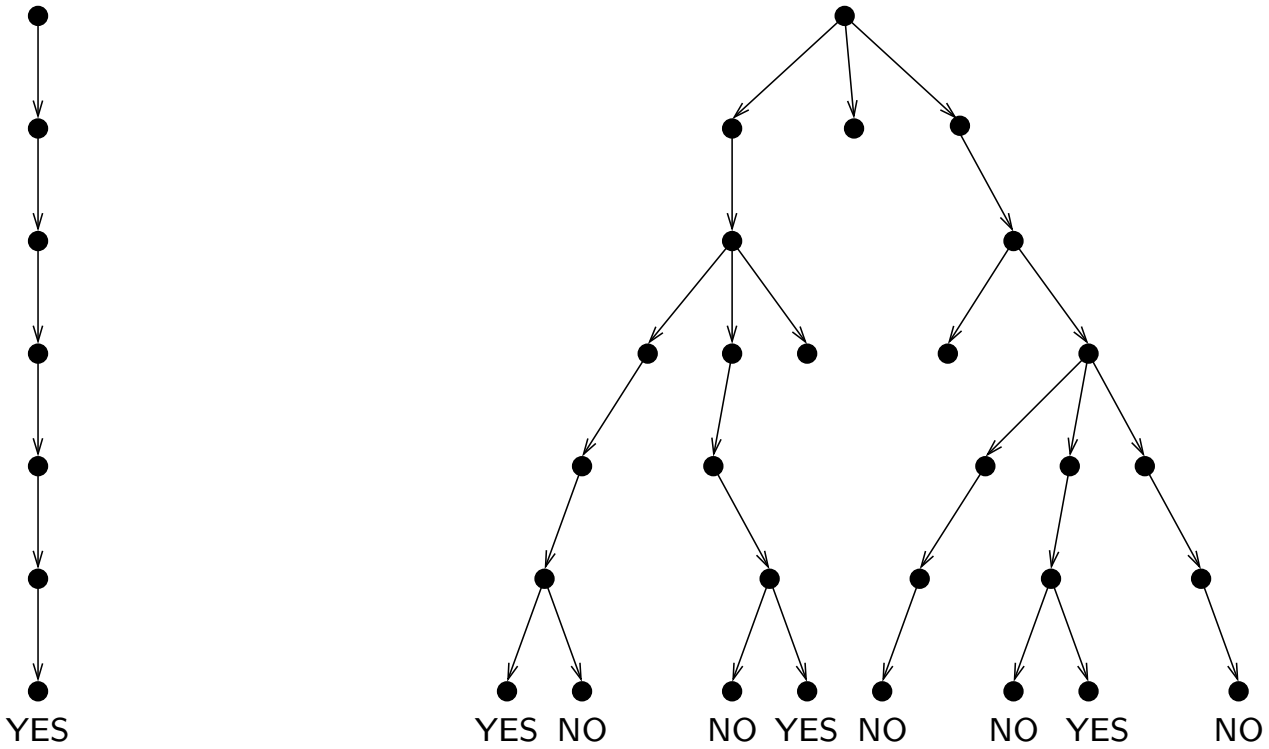
Nondeterminism

Nondeterministic RAM:

- Its definition is very similar to that of a deterministic RAM.
- Moreover, it has an instruction NDJUMP x OR y 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 YES, then the answer is YES.
- If all computations end with the answer NO then the answer is NO.

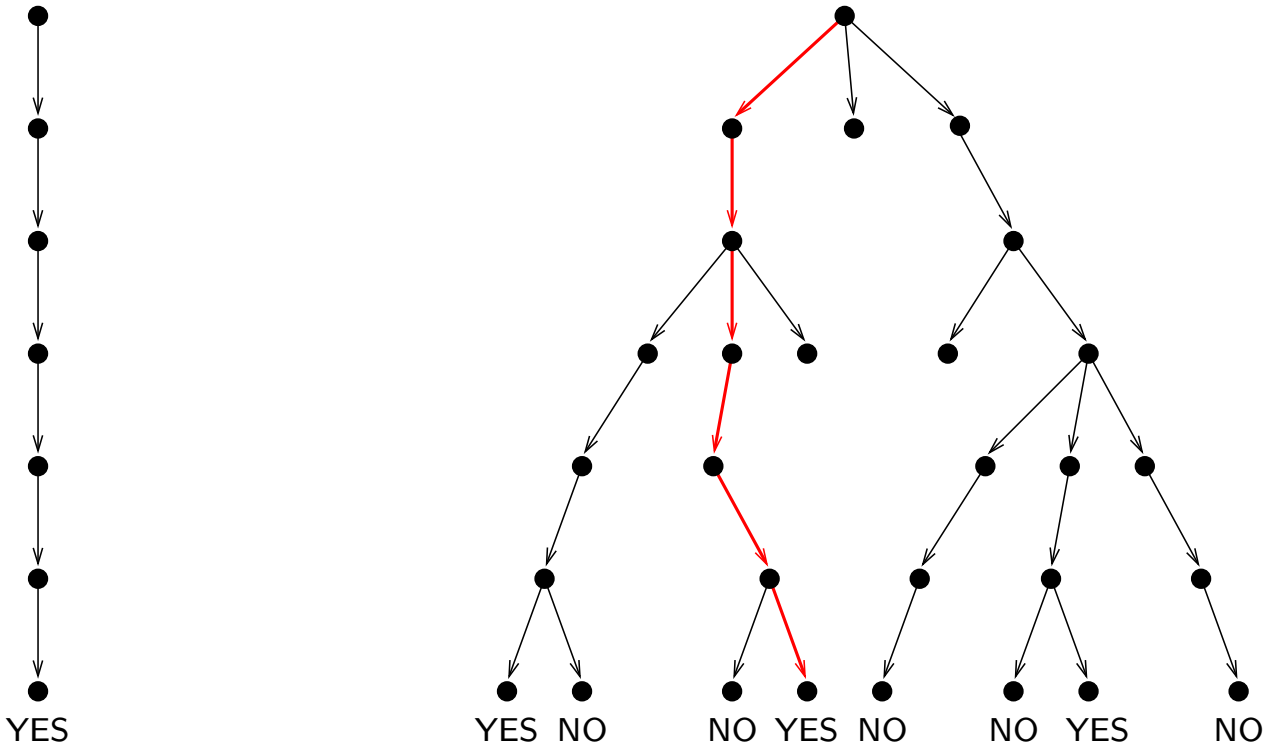
Nondeterministic versions of other computational models (such as nondeterministic Turing machines) are defined similarly.

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



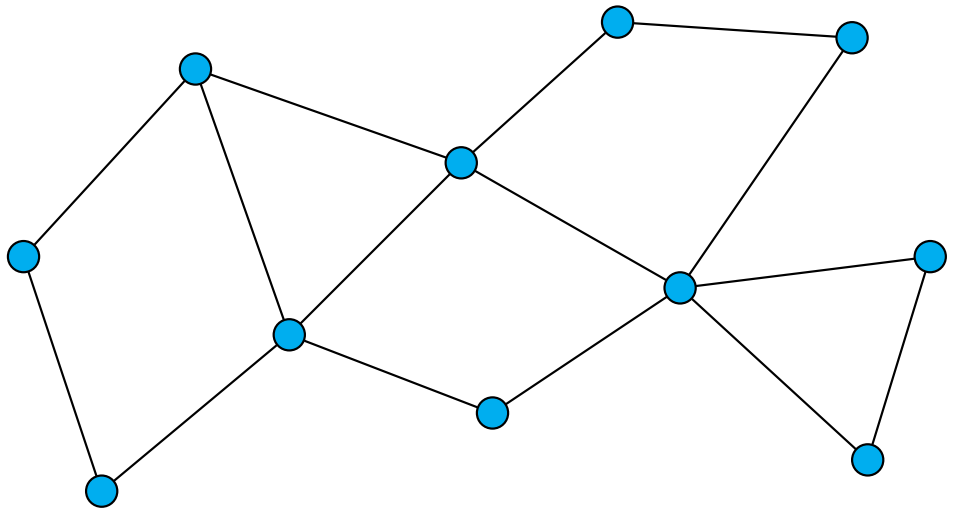
- The time required for a computation of a nondeterministic RAM (or other nondeterministic machine) on a given input is defined as the length of the longest computation on the input.

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?



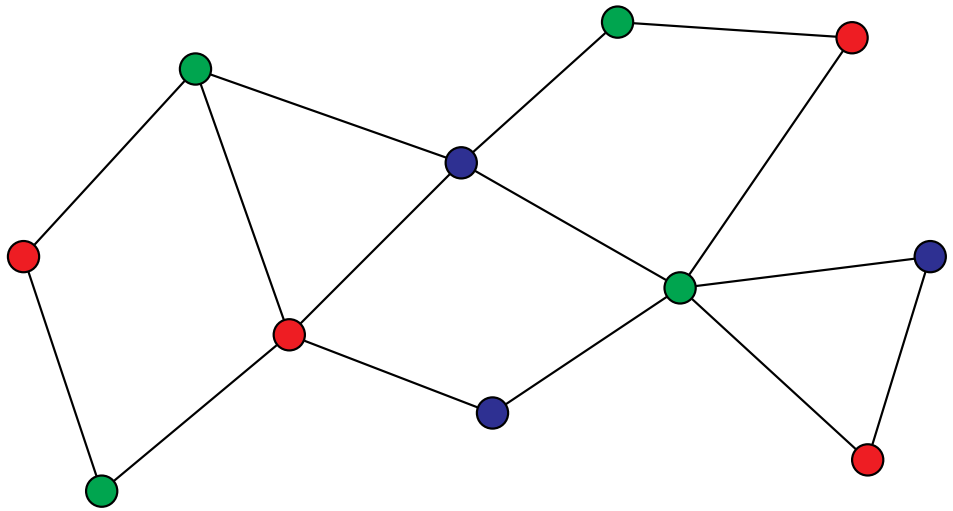
$k = 3$

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 NO. Otherwise it halts with the answer YES.

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.

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.

Nondeterministic Complexity Classes

Definition

For a function $f : \mathbb{N} \to \mathbb{N}$ 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 $f : \mathbb{N} \to \mathbb{N}$ 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 $O(f(n))$.

Class NPTIME

Definition

$$\text{NPTIME} = \bigcup_{k=0}^{\infty} \mathcal{NT}(n^k)$$

- 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.
- It is obvious that PTIME $\subseteq$ NPTIME, since deterministic algorithms can be viewed as a special case of nondeterministic algorithms.

Complexity Classes

- Class NPSPACE can be defined similarly.
- Similarly to the deterministic case, it holds that NPTIME $\subseteq$ NPSPACE.
- A nondeterministic RAM can be simulated by a deterministic RAM which tries all possible computations. Each of these possible computations uses at most a polynomial amount of memory and individual computations share no information so we can use the same memory for each of them. So we have NPTIME $\subseteq$ PSPACE.
- It is also known that if some problem is solved by a nondeterministic Turing machine with a space complexity $f(n)$ then it is also solved by a deterministic Turing machine with a space complexity $O((f(n))^2)$. Based on this it is evident that NPSPACE $\subseteq$ PSPACE.

$$\text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE} = \text{NPSPACE}$$

NP-Complete Problems

Polynomial Reduction between Problems

Let us have two decision problems A and B.

Problem A can be **reduced** to problem B if there is an algorithm P such that:

- It can get an arbitrary instance of problem A as an input.
- For an instance of a problem A obtained as an input (let us denote it as x) it produces an instance of a problem B as an output.
- It holds

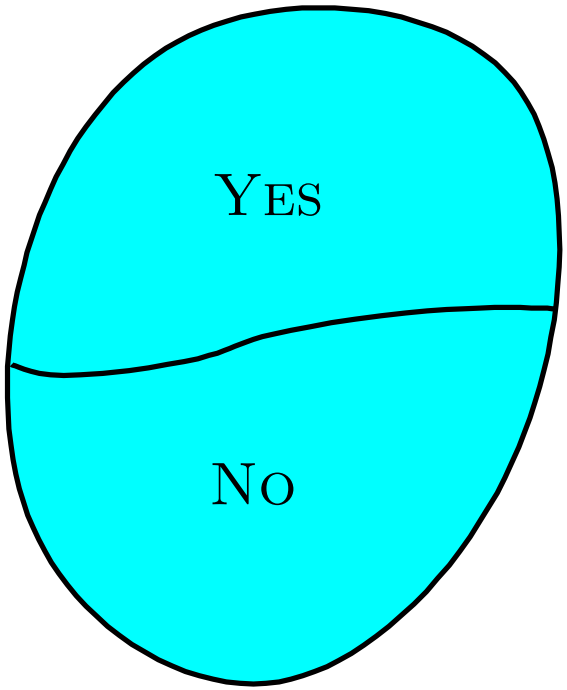
$$x \in A \quad \Leftrightarrow \quad P(x) \in B$$

  i.e., the answer for the input x of problem A is YES iff the answer for the input $P(x)$ of problem B is YES.

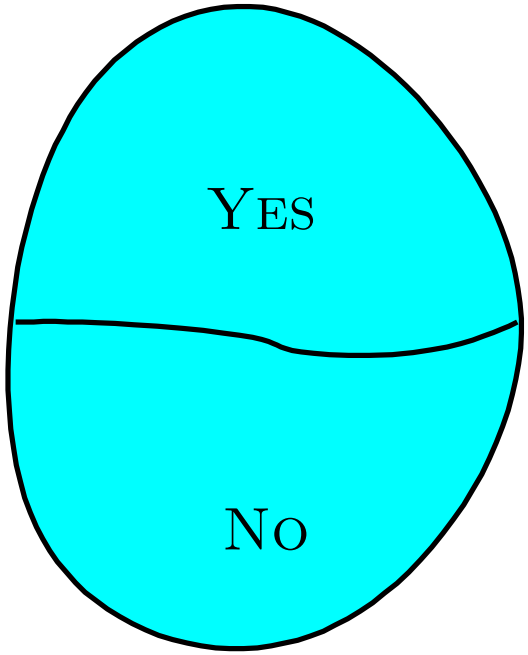
If, in addition, the algorithm P is polynomial, we say that problem A can be **reduced in polynomial time** to problem B.

Polynomial Reduction between Problems

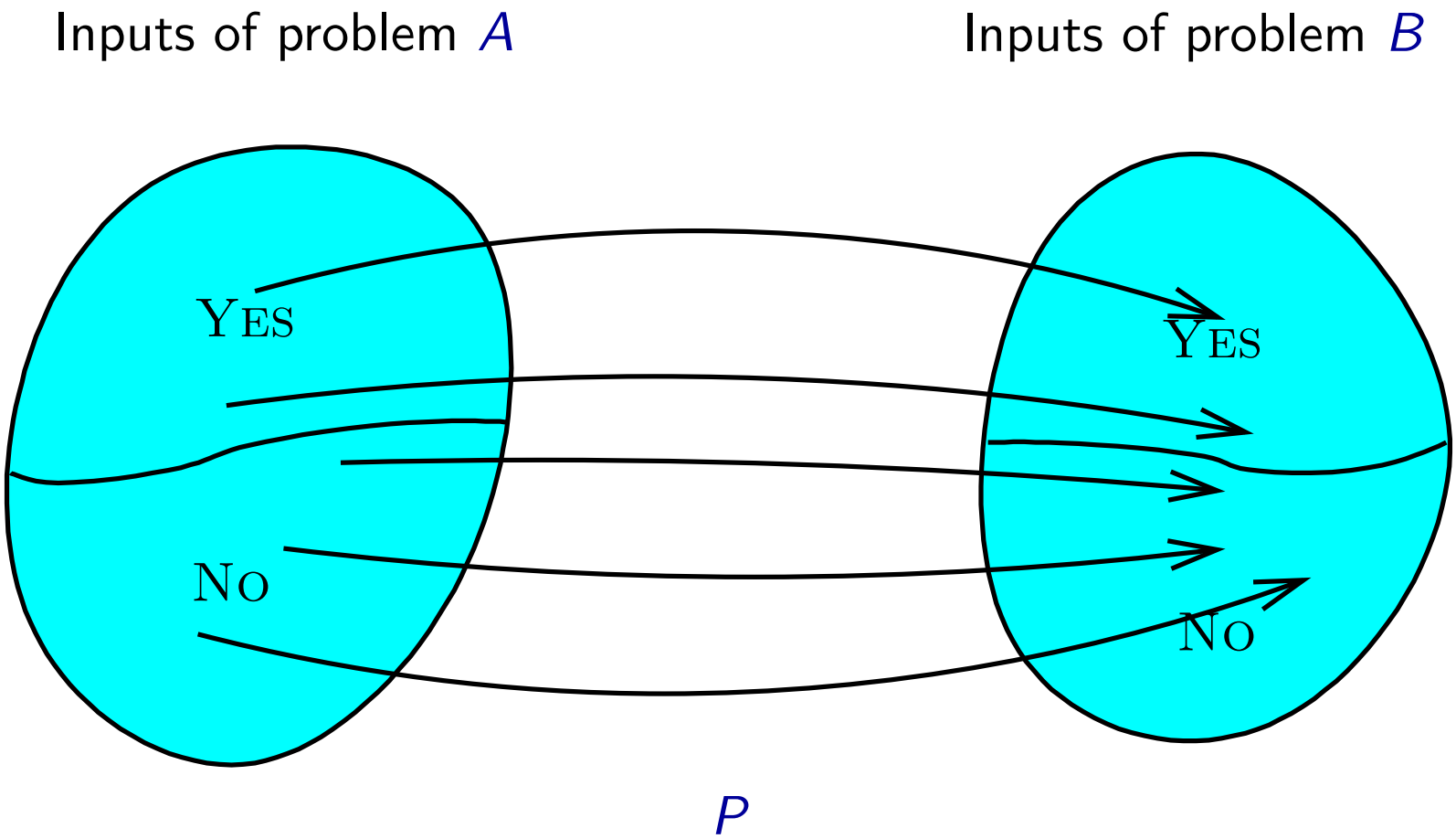
Inputs of problem *A*



Inputs of problem *B*



Polynomial Reduction between Problems



Polynomial Reduction 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.

Problem SAT

Definition of problem SAT:

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 ν where $[x_1]_\nu = 1$, $[x_2]_\nu = 0$, $[x_3]_\nu = 1$, it holds that $[\varphi_1]_\nu = 1$.

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

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

Some notions:

- A **literal** is a formula of the form x or $\neg x$ where x is a boolean variable.
- A **clause** is a disjuction of literals.

  Examples: $x_1 \vee \neg x_2$ $\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}$ 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:

$$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

For example for valuation ν where

$$\begin{aligned} [x_1]_\nu &= 0 \\ [x_2]_\nu &= 1 \\ [x_3]_\nu &= 0 \\ [x_4]_\nu &= 1 \end{aligned}$$

is $[\varphi_1]_\nu = 1$.

On the other hand, the following formula is not satisfiable:

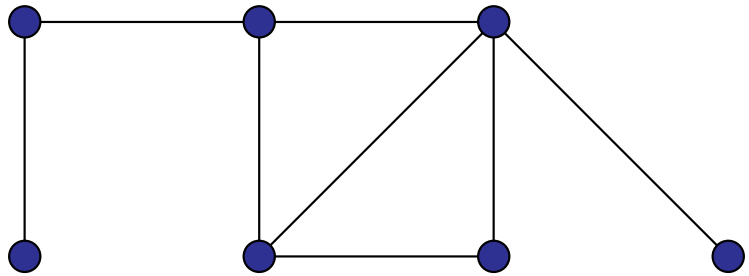
$$(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$$

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?



$k = 4$

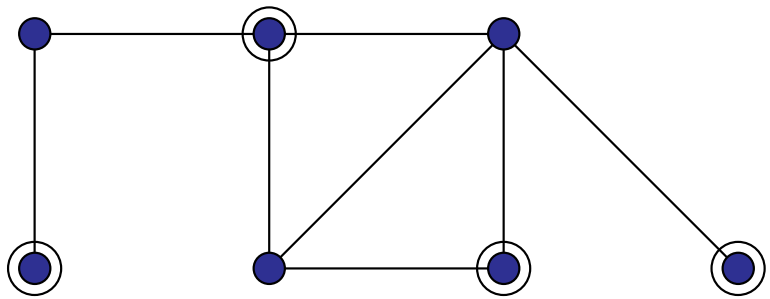
Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

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?

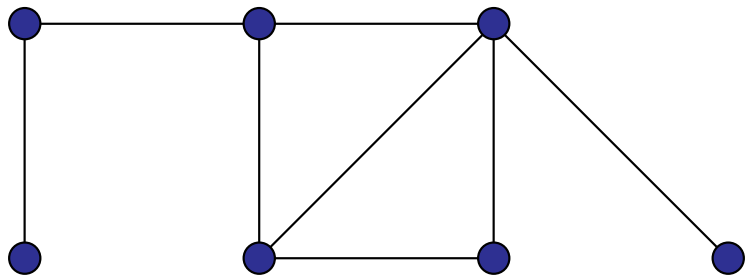


$k = 4$

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.

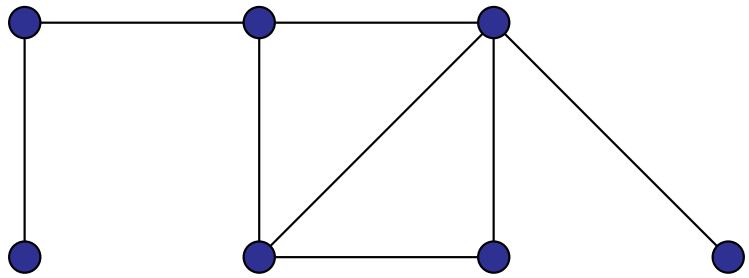
Independent Set (IS) Problem

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):

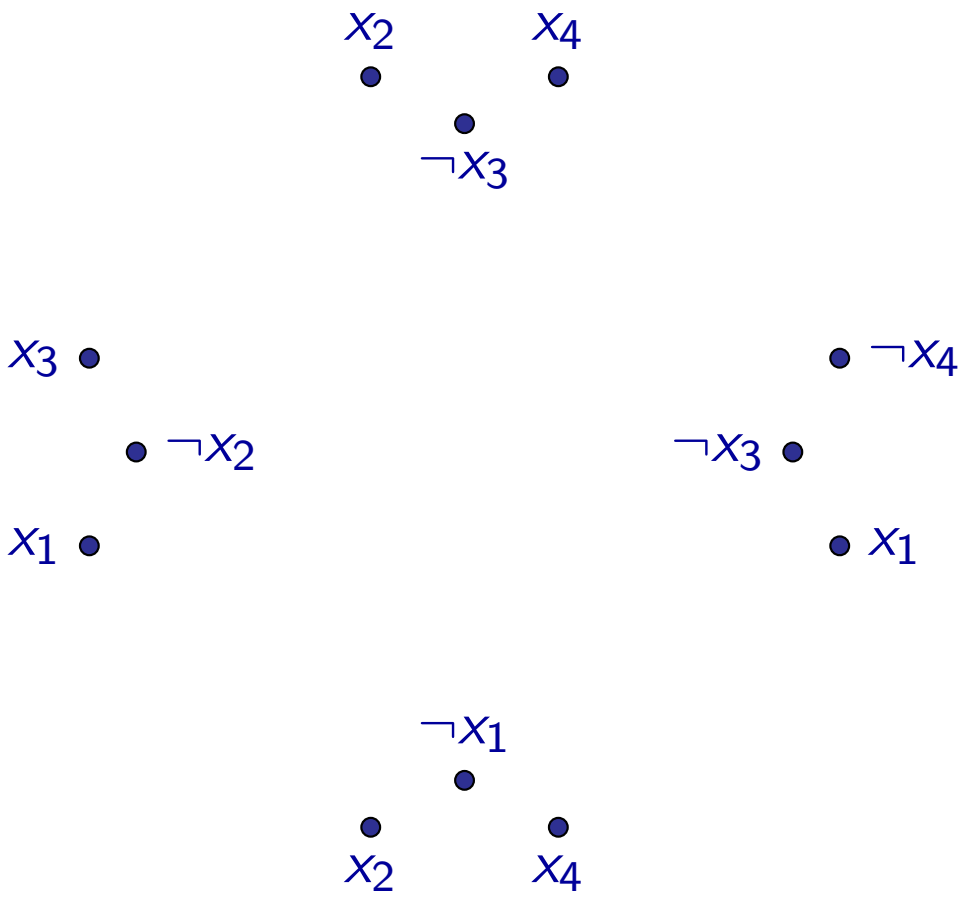
  There will be an independent set of size k in graph G iff formula φ will be satisfiable.

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

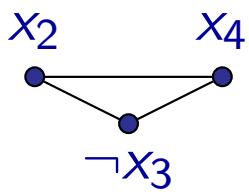
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

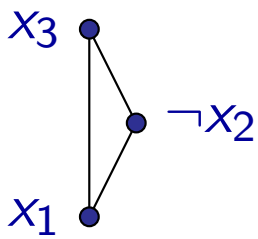


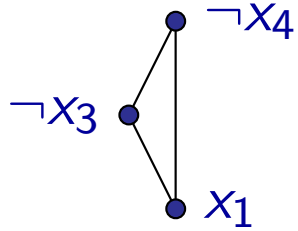
For each literal we add a node to the graph.

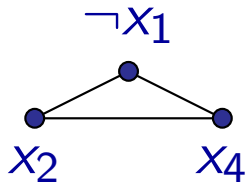
A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$





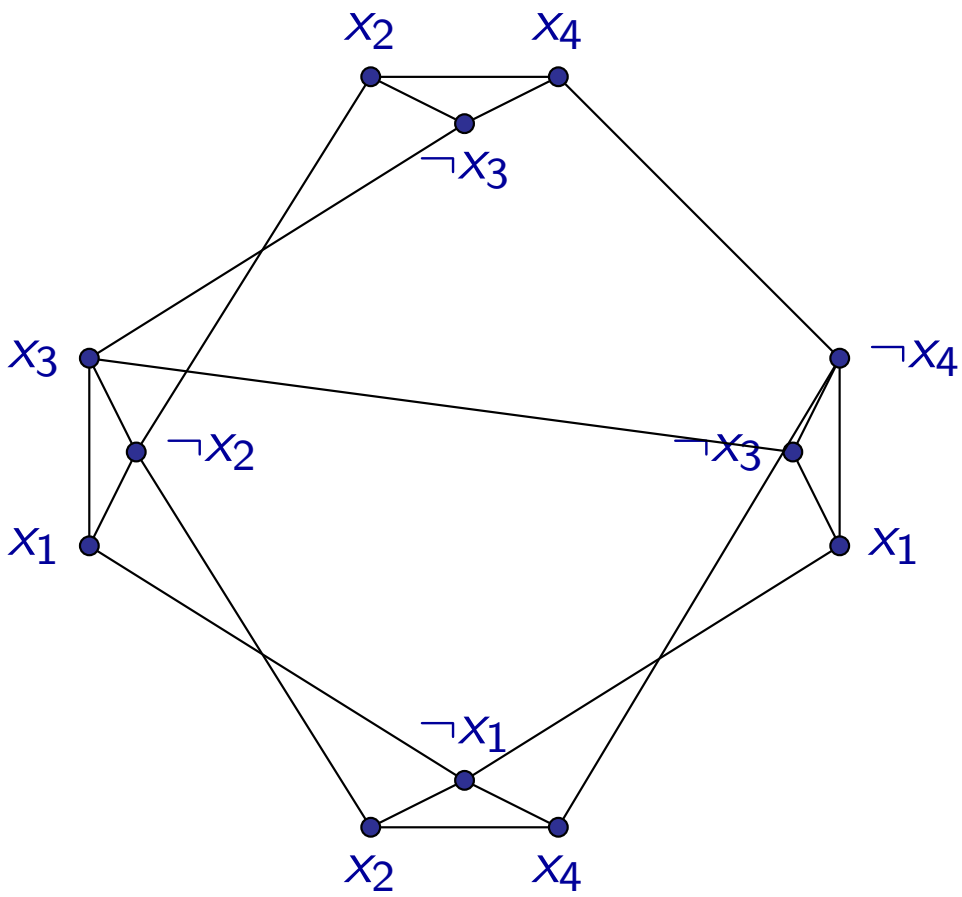




We connect with edges the nodes corresponding to literals belonging to the same clause.

A Reduction from 3-SAT to IS

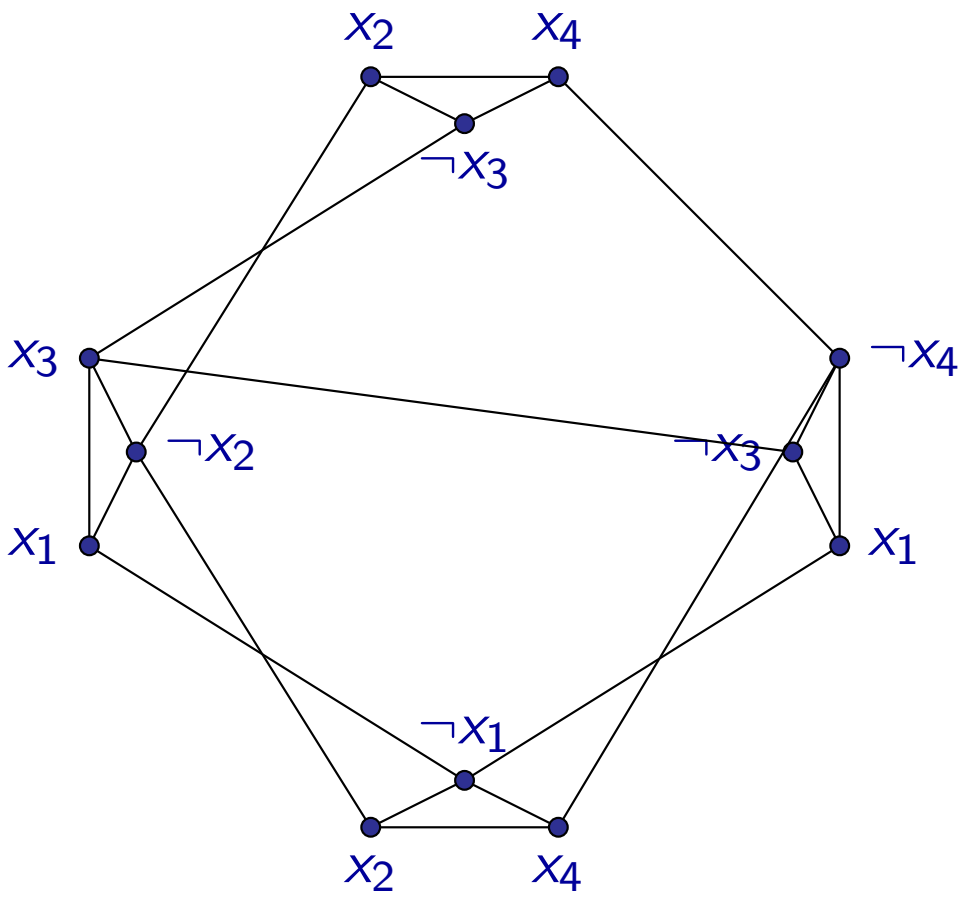
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



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

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

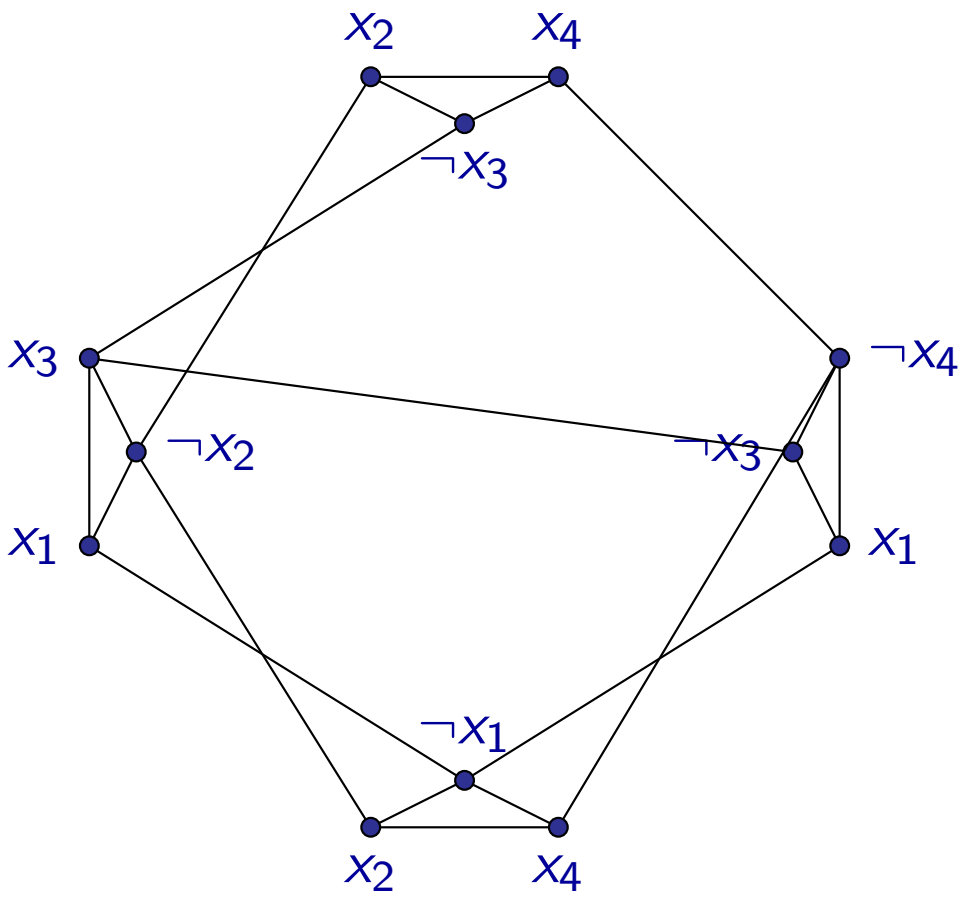


$k = 4$

We put k to be equal to the number of clauses.

A Reduction from 3-SAT to IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



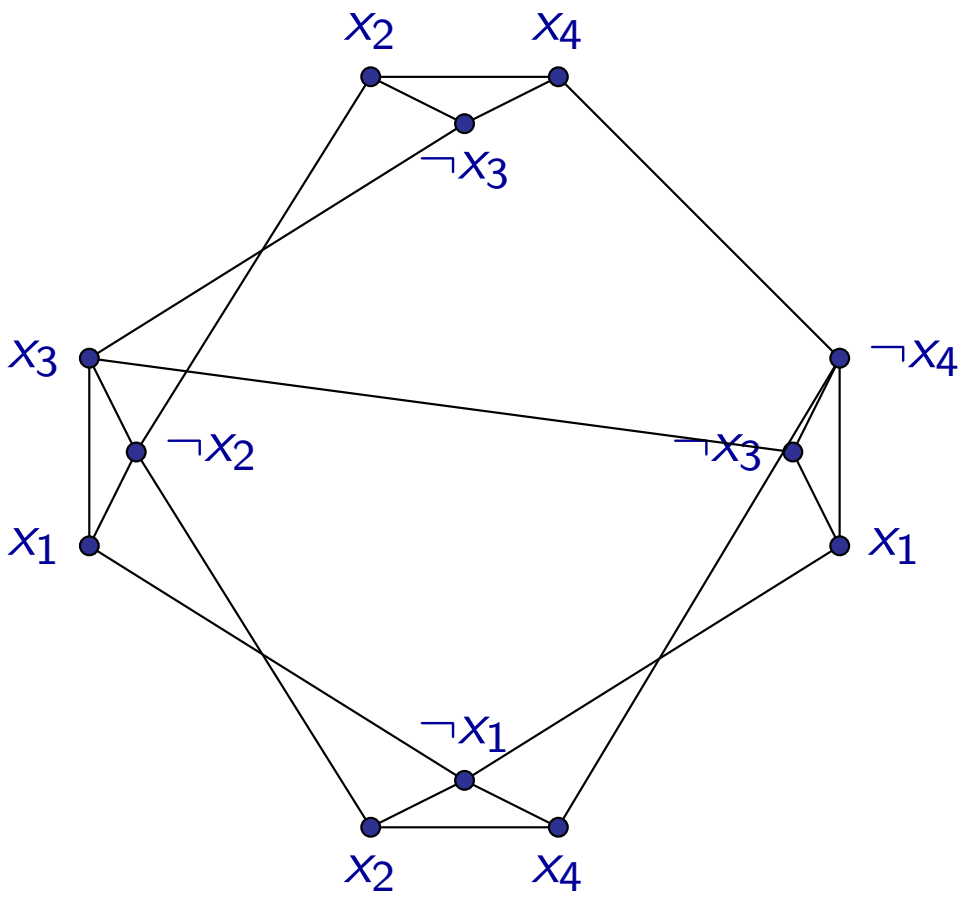
$k = 4$

The constructed graph and number k are the output of the algorithm.

A Reduction from 3-SAT to IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$\nu(x_1) = 1$
$\nu(x_2) = 1$
$\nu(x_3) = 0$
$\nu(x_4) = 1$



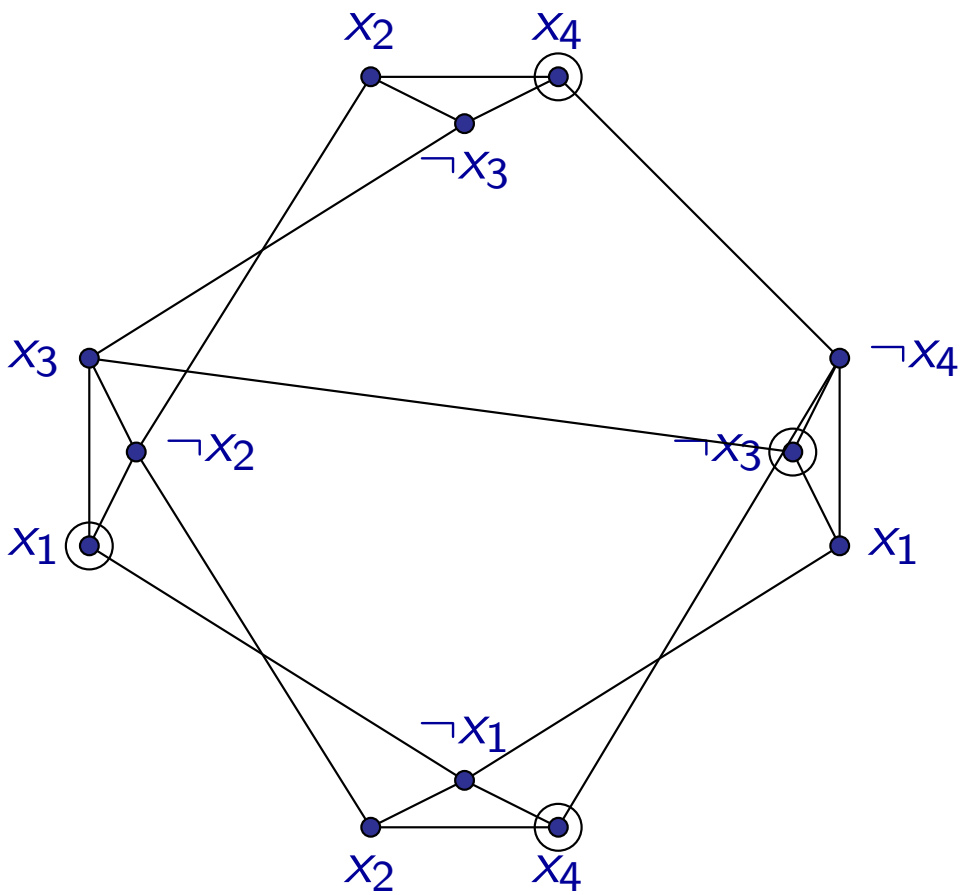
$k = 4$

If the formula φ is satisfiable then there is a valuation ν where every clause contains at least one literal with value 1.

A Reduction from 3-SAT to IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$\nu(x_1) = 1$
$\nu(x_2) = 1$
$\nu(x_3) = 0$
$\nu(x_4) = 1$



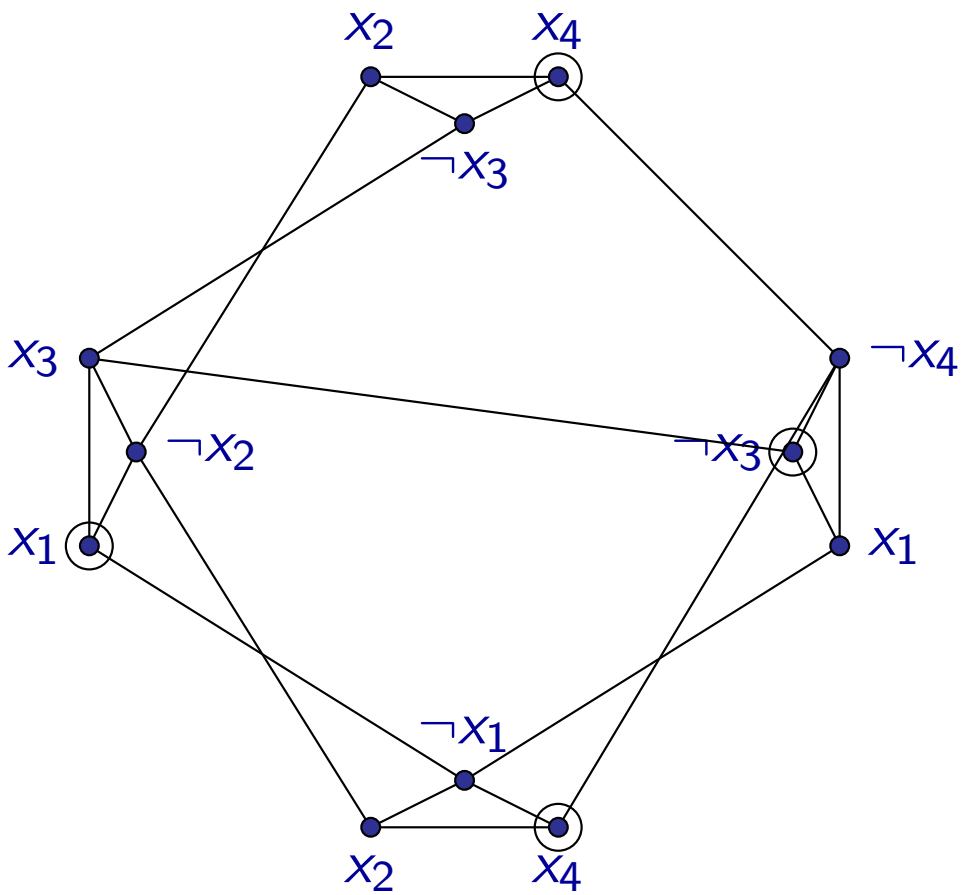
$k = 4$

We select one literal that has a value 1 in the valuation ν, and we put the corresponding node into the independent set.

A Reduction from 3-SAT to IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$\nu(x_1) = 1$
$\nu(x_2) = 1$
$\nu(x_3) = 0$
$\nu(x_4) = 1$



$k = 4$

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 ν.)

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:

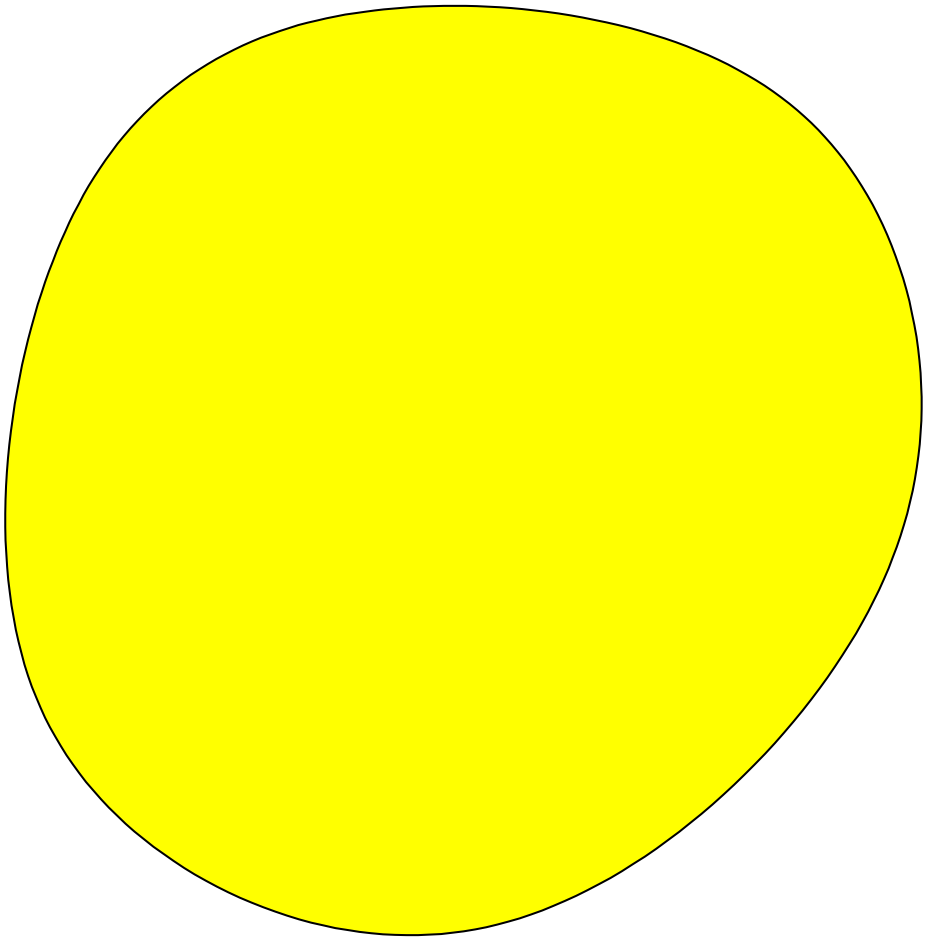
Graph G and number k can be constructed for a formula φ in time $O(n^2)$, where n is the size of formula φ.

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.

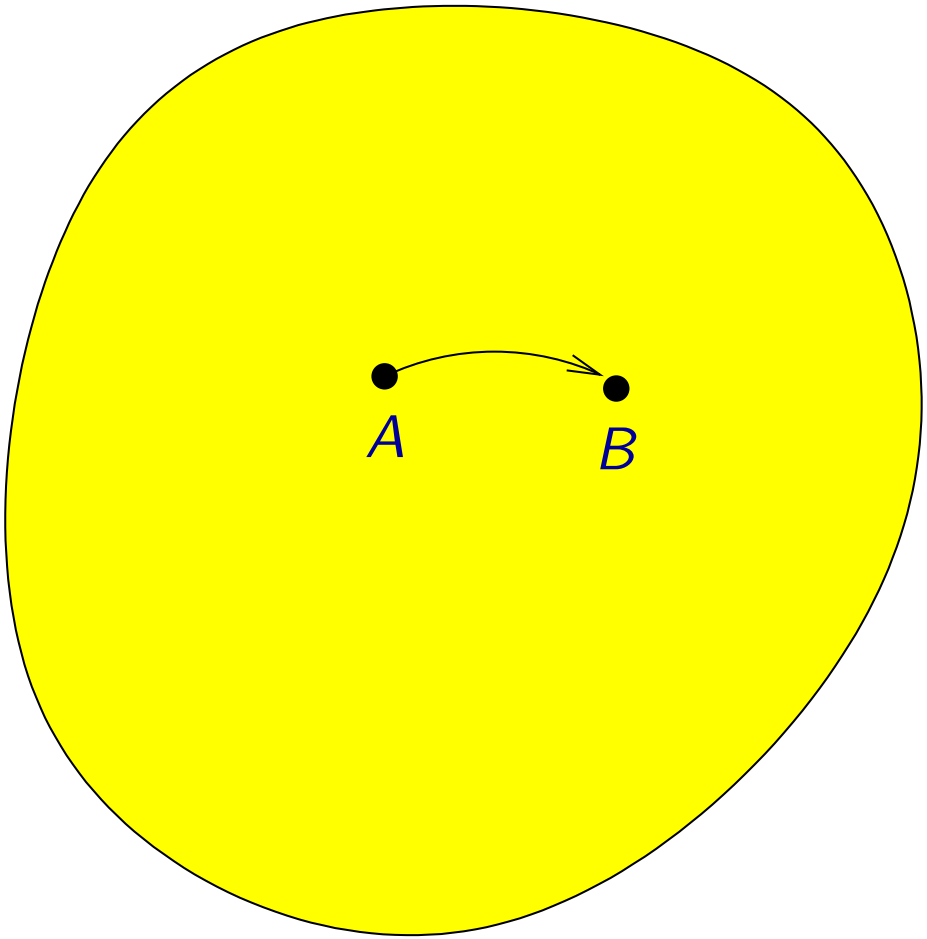
NP-Complete Problems

Let us consider a set of all decision problems.



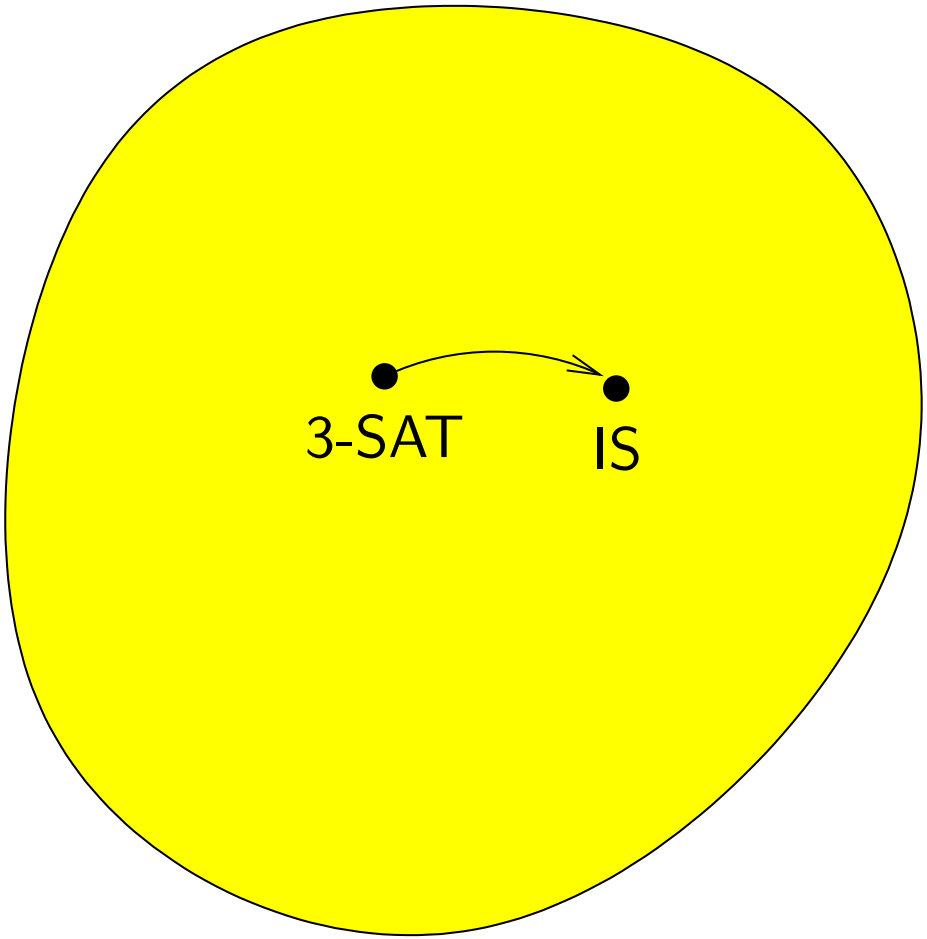
NP-Complete Problems

By an arrow we denote that a problem A can be reduced in polynomial time to a problem B.



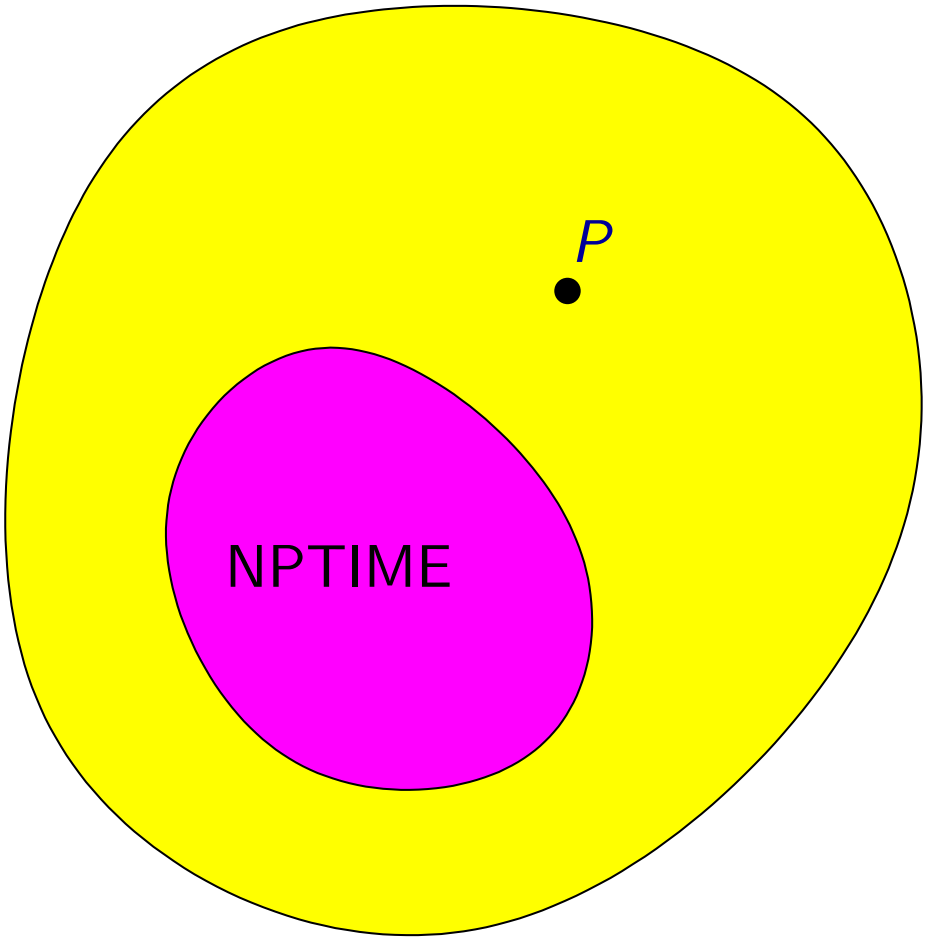
NP-Complete Problems

For example 3-SAT can be reduced in polynomial time to IS.



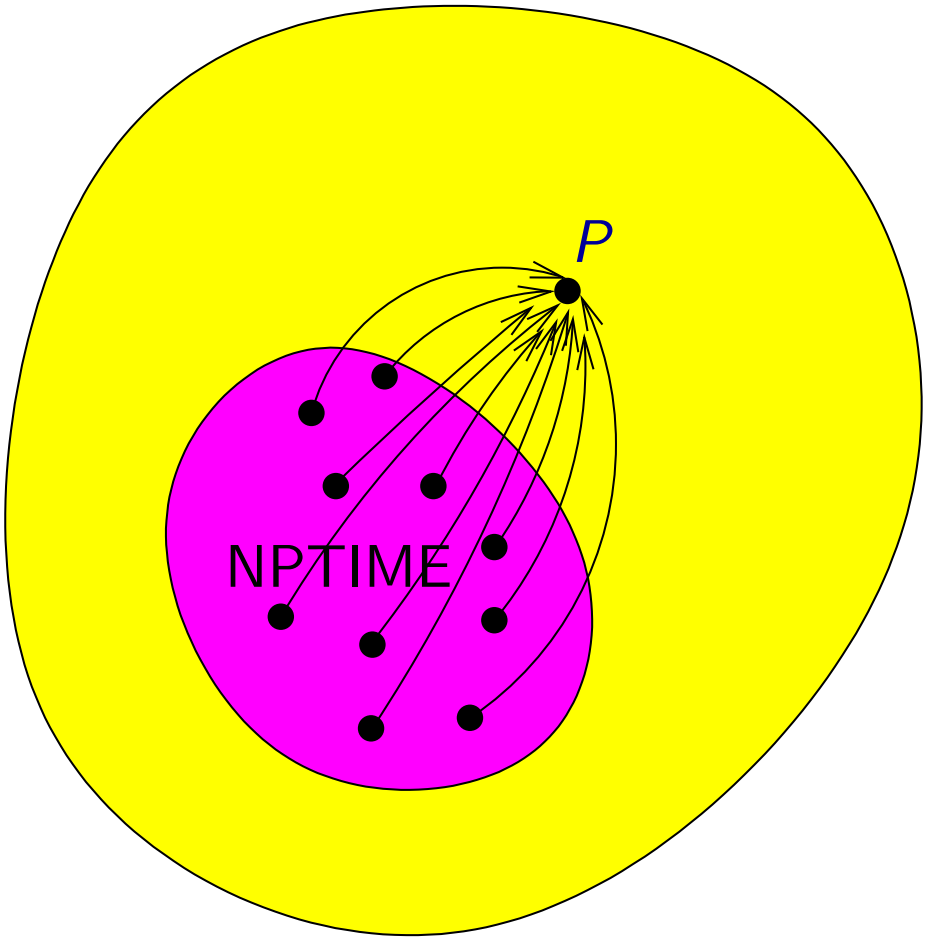
NP-Complete Problems

Let us consider now the class NPTIME and a problem P.



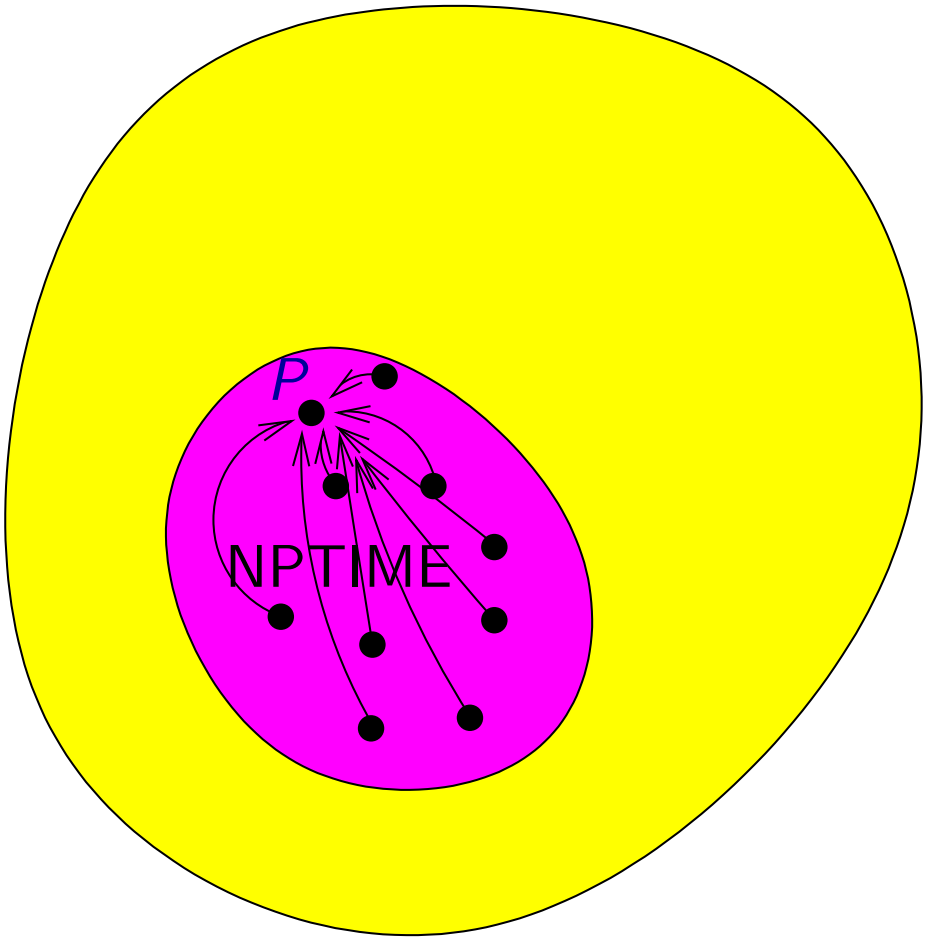
NP-Complete Problems

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

Theorem

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.

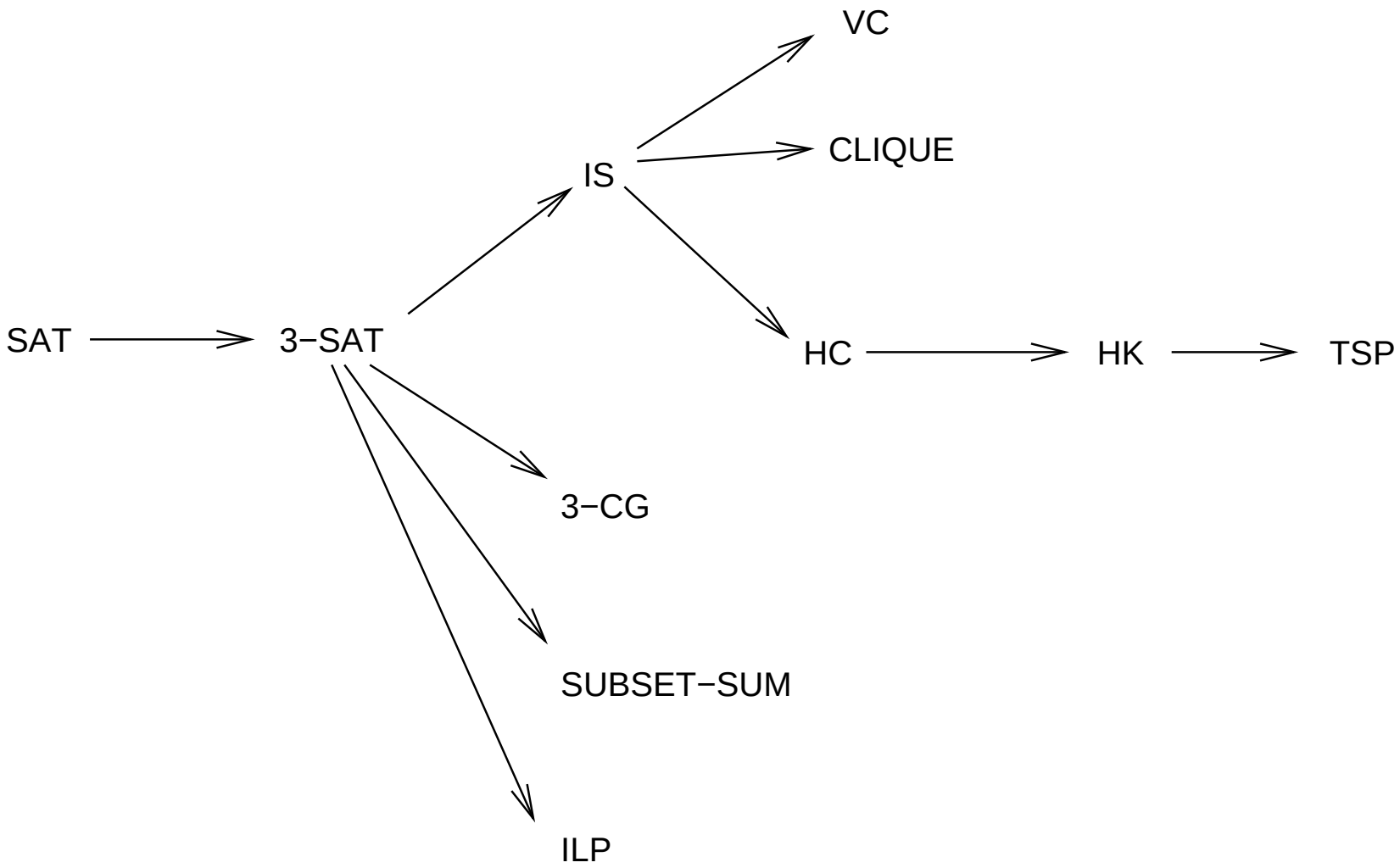
This means that problems 3-SAT and IS are NP-hard.

It is obvious that 3-SAT and IS are in 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:



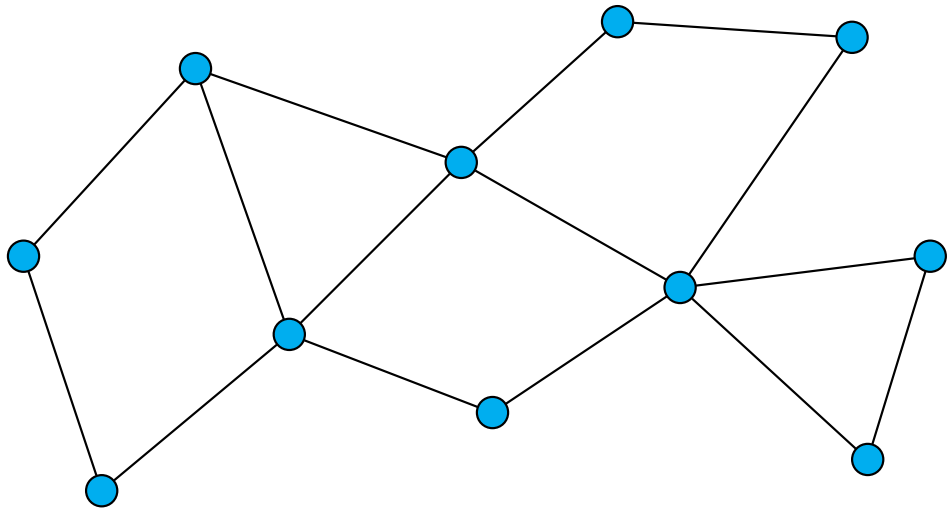
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

Input: An undirected graph G

Question: Is it possible to color nodes of the graph G using 3 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:



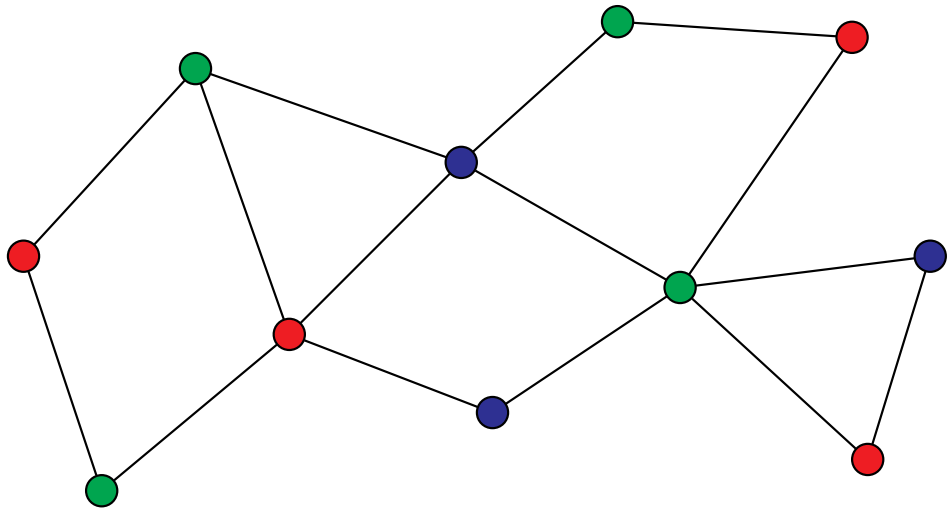
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

Input: An undirected graph G

Question: Is it possible to color nodes of the graph G using 3 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:



Answer: YES

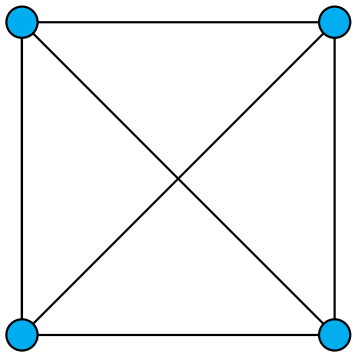
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

Input: An undirected graph G

Question: Is it possible to color nodes of the graph G using 3 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:



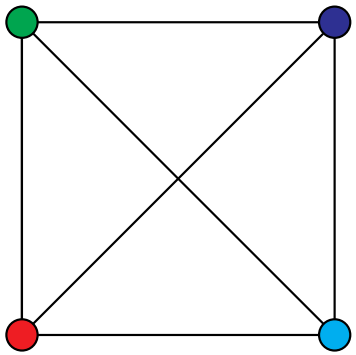
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

Input: An undirected graph G

Question: Is it possible to color nodes of the graph G using 3 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:



Answer: No

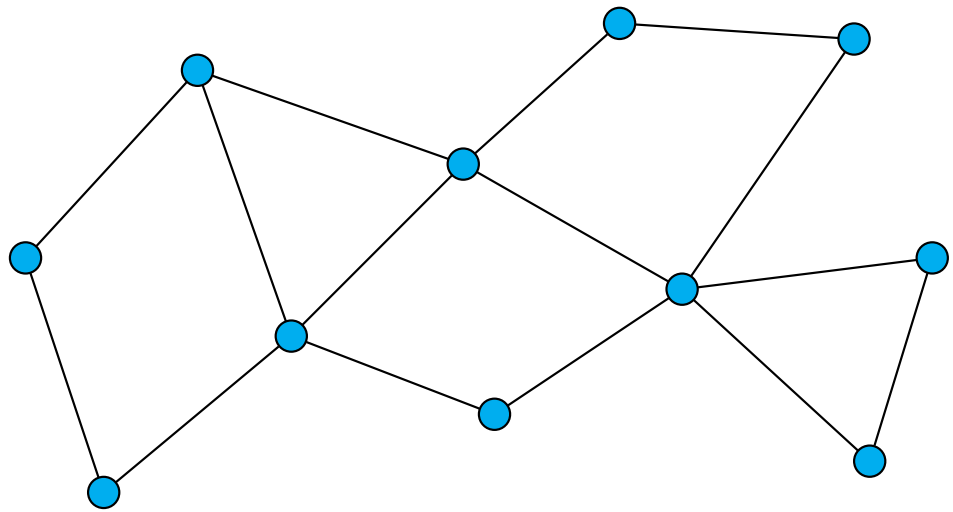
IS – Independent Set

IS - Independent Set

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

Question: Is there an independent set of size k in graph G (i.e., a set of k nodes where no pair of nodes from this set is connected by an edge)?

Example: $k = 5$



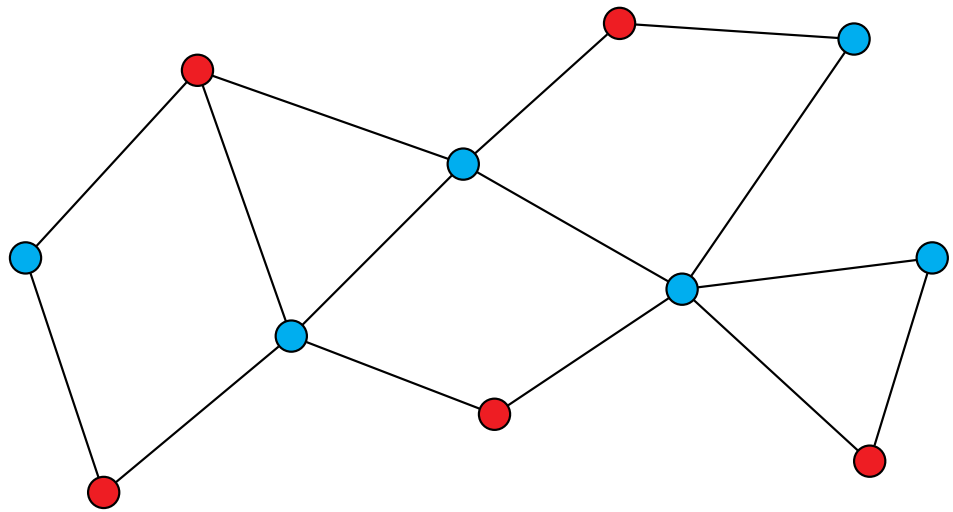
IS – Independent Set

IS - Independent Set

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

Question: Is there an independent set of size k in graph G (i.e., a set of k nodes where no pair of nodes from this set is connected by an edge)?

Example: $k = 5$



Answer: YES

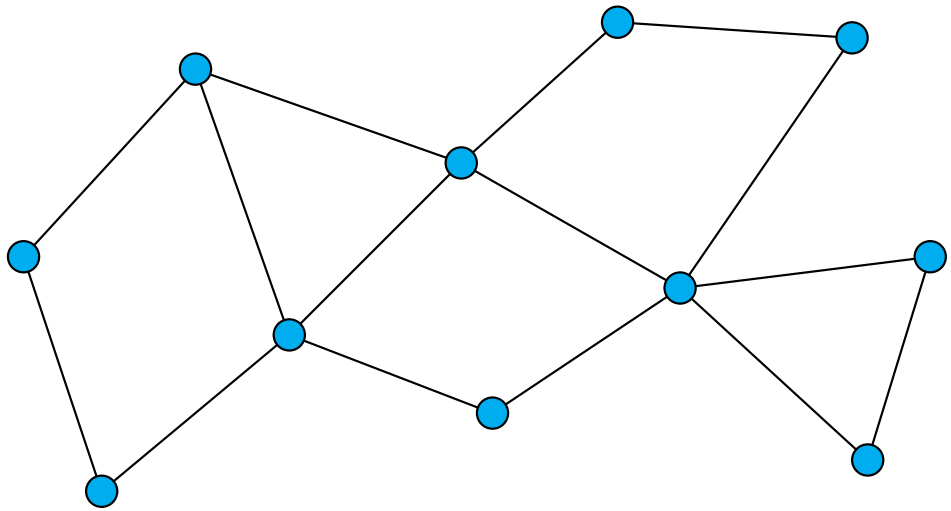
VC – Vertex Cover

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: $k = 6$



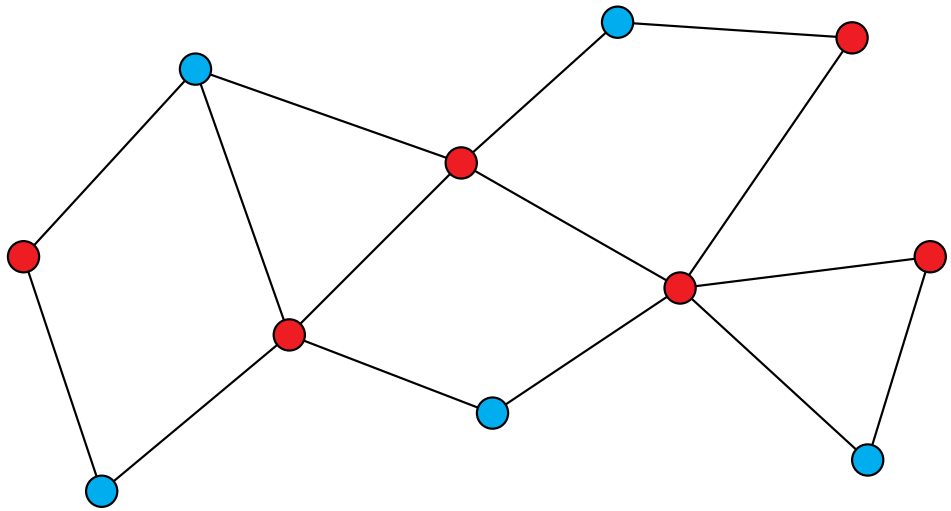
VC – Vertex Cover

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: $k = 6$



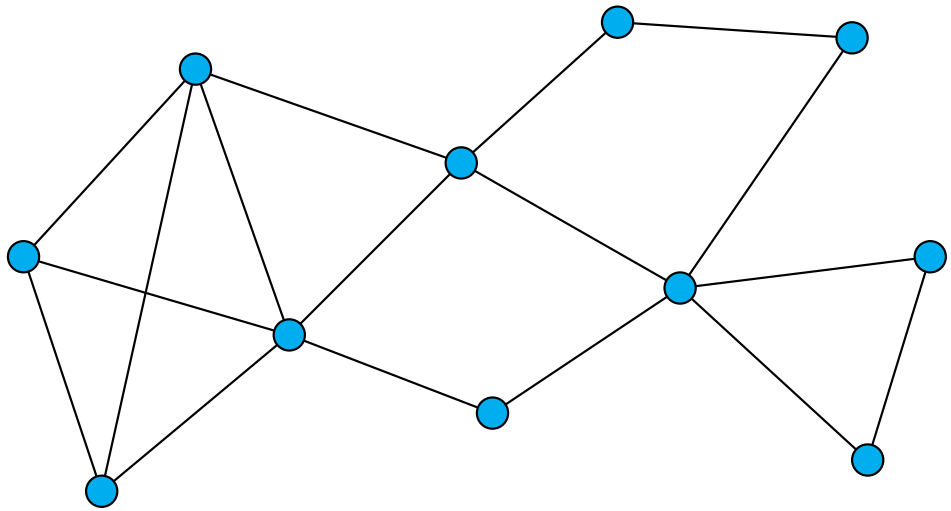
Answer: YES

CLIQUE

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: $k = 4$



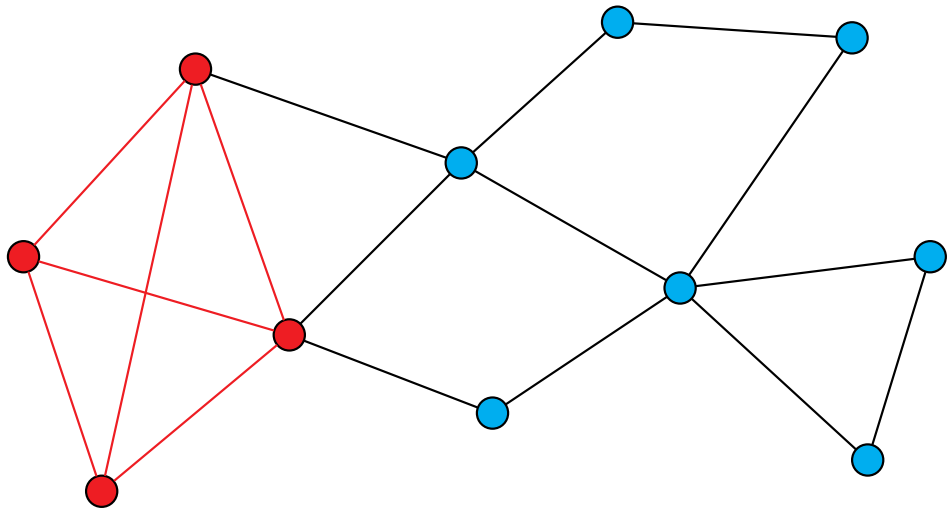
CLIQUE

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

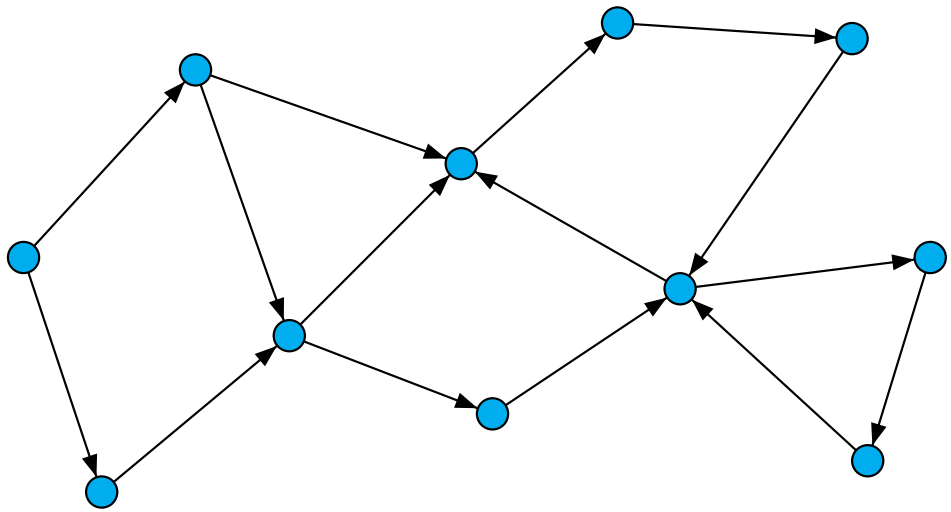
Hamiltonian Cycle

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:



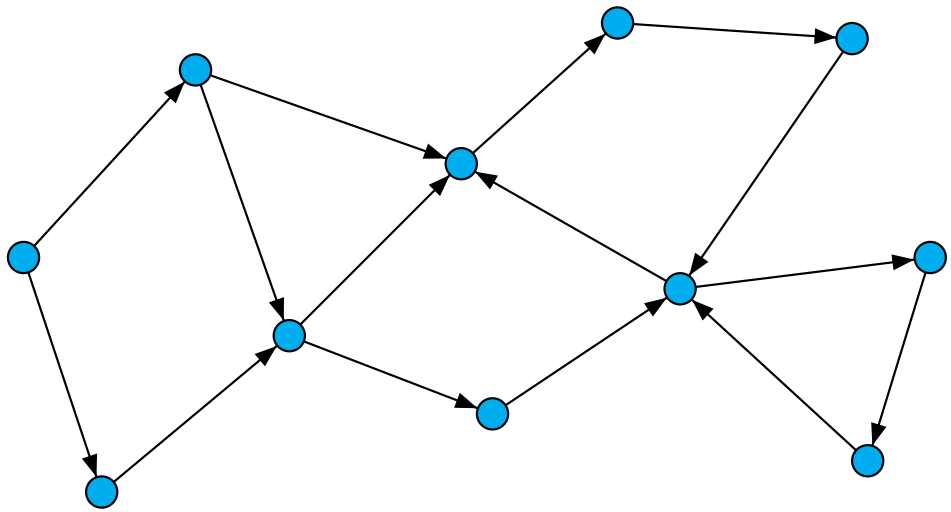
Hamiltonian Cycle

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

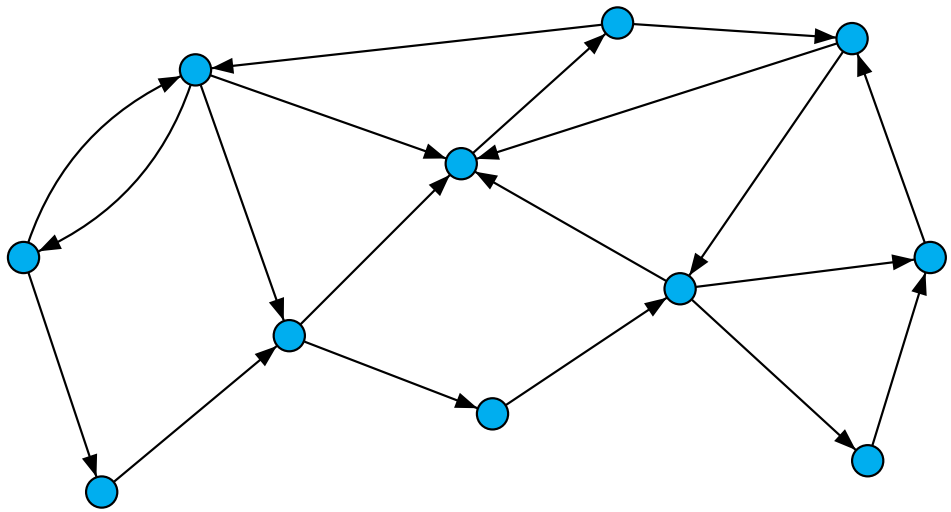
Hamiltonian Cycle

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:



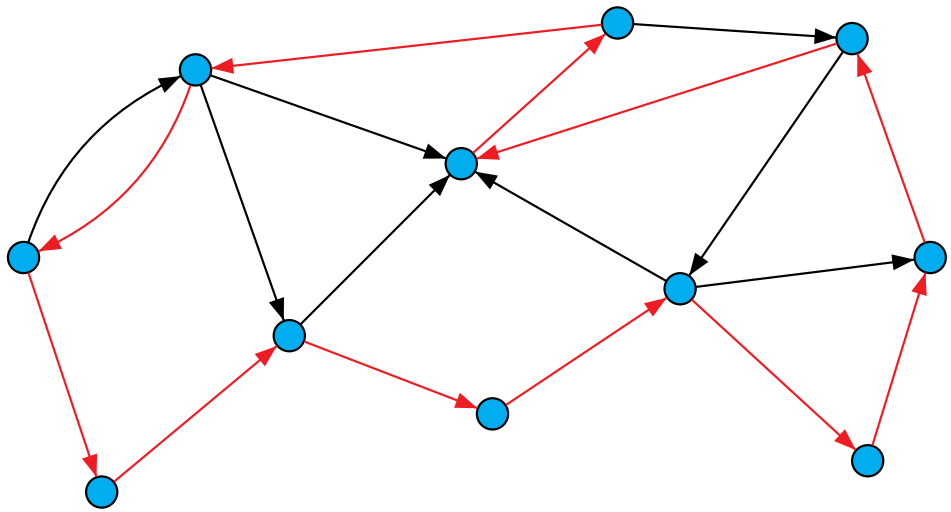
Hamiltonian Cycle

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

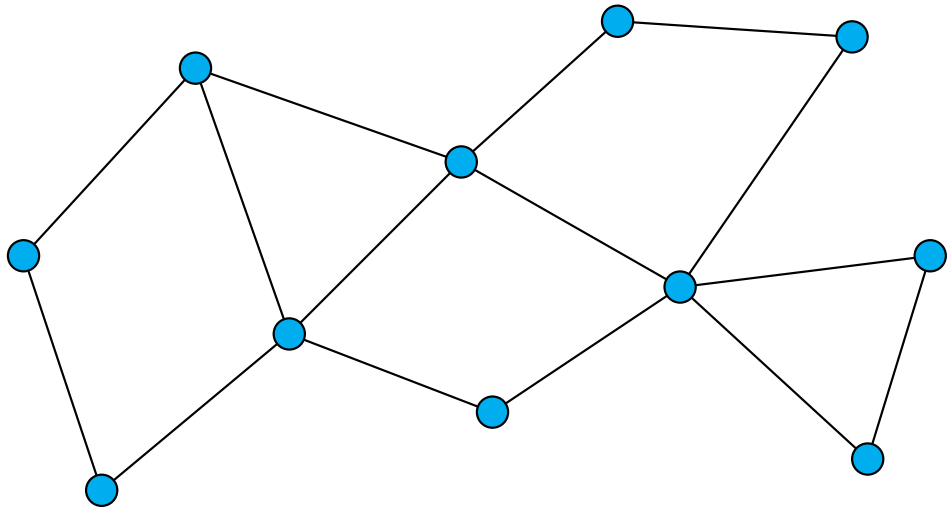
Hamiltonian Circuit

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

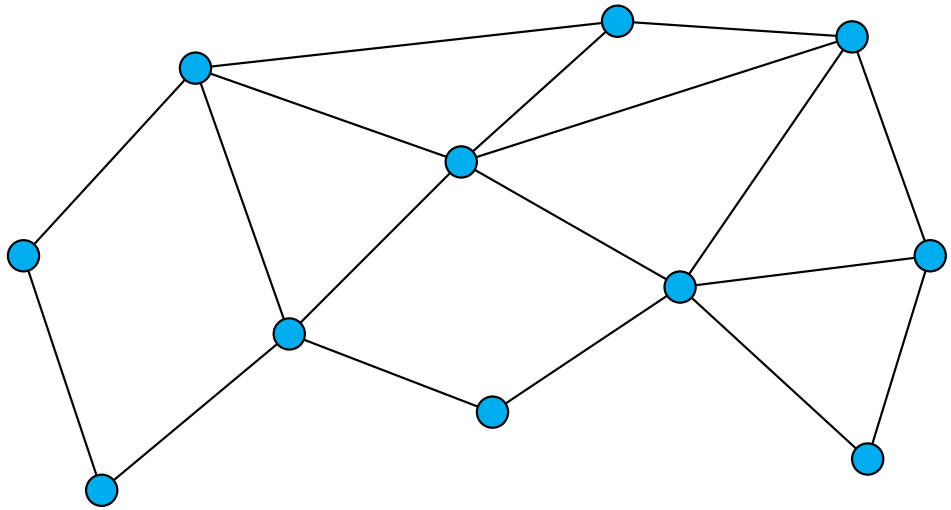
Hamiltonian Circuit

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:



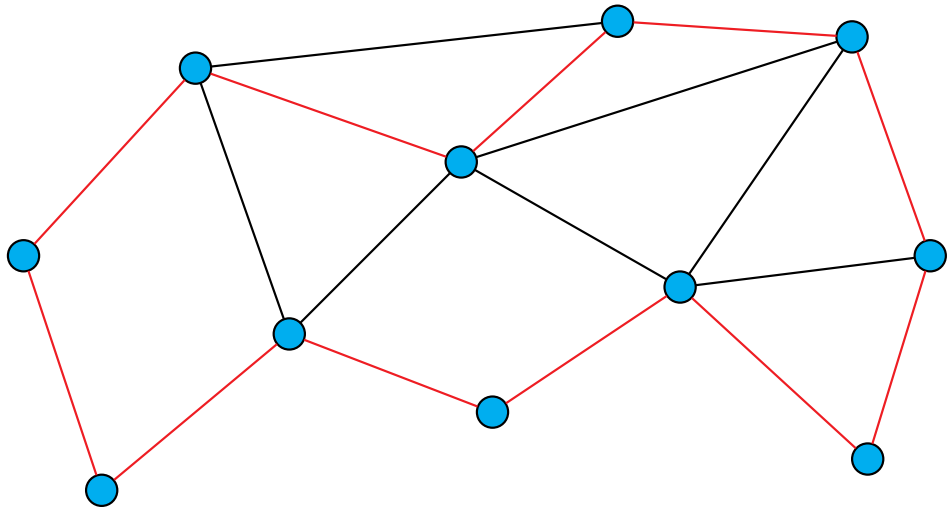
Hamiltonian Circuit

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

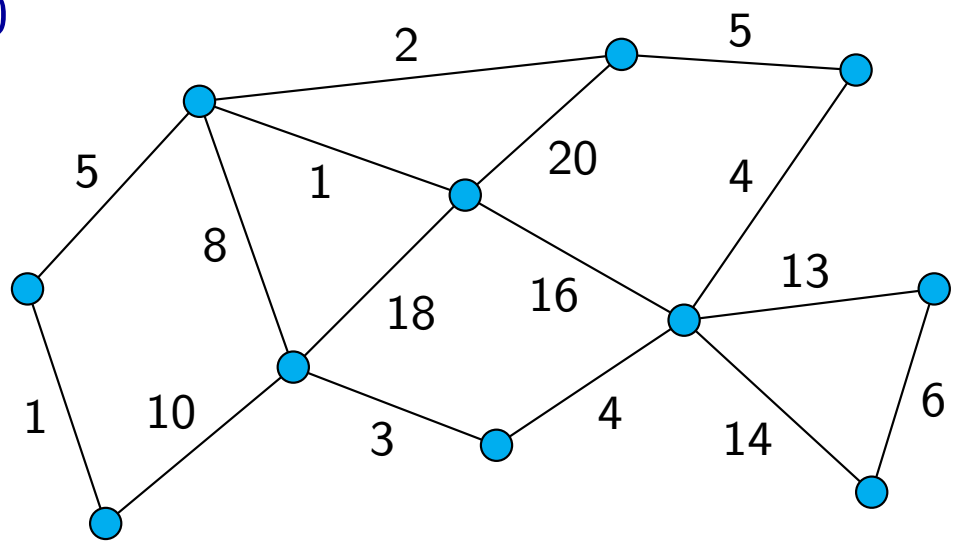
Traveling Salesman Problem

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: $k = 70$



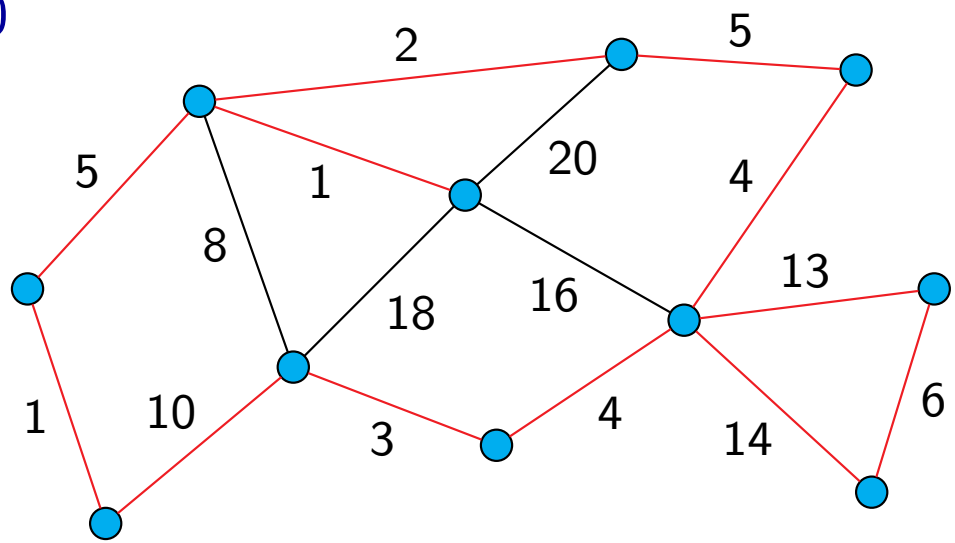
Traveling Salesman Problem

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 YES, since $3 + 5 + 7 = 15$.

For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 16$ the answer is NO, since no subset of these number 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-SET sum is a special case of the Knapsack problem can be seen from the following (almost trivial) reduction:

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 numbers in an input encoded in **binary**, i.e., the size of an input is proportional to the sum of lengths of binary representations of numbers in the input, then the problem SUBSET-SUM is NP-hard. (There is a polynomial time reduction from 3-SAT.)

SUBSET-SUM

It is not difficult to see that SUBSET-SUM and Knapsack problem (its decision variant) belong to the class NPTIME:

- A nondeterministic algorithm at first nondeterministically chooses a subset of elements of the sequence in the input and then (deterministically) verifies if it satisfies the given condition (resp. conditions).

  It is obvious that this verification can be done in time polynomial with respect to the size of the instance.

So the problems SUBSET-SUM and Knapsack problem are NP-complete.

ILP – Integer Linear Programming

Problem ILP (integer linear programming)

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

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

An example of an instance of the problem:

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

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

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

ILP – Integer Linear Programming

One of solutions of the system

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

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

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

because

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

So the answer for this instance is YES.

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.

P vs. NP

As already mentioned, the question if PTIME $=$ NPTIME is a longstanding open problem.

In general, it is supposed that most likely PTIME $\neq$ NPTIME but nobody has proved it yet.

Remark: For example it can be shown that PTIME $\neq$ NPTIME is a necessary (but not sufficient) condition for an existence of ciphers that are not easily breakable.

If somebody did find a polynomial algorithm for at least one NP-complete problem, we would have immediately obtained algorithms for fast breaking of all ciphers that are currently used.

Undecidable Problems

Algorithmically Undecidable Problems

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

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

Example: The following problem called the **Halting problem** is undecidable:

Halting problem

Input: A description of a Turing machine M and a word w.

Question: Does the computation of the machine M on the word w halt after some finite number of steps?

Halting Problem

Alternatively, we could define the problem as follows:

Halting problem

Input: A source code of a C 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

boolean 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.

Other Undecidable Problems

We have already seen an example of an undecidable problem:

Problem

Input: Context-free grammars G_1 and G_2.

Question: Is $L(G_1) = L(G_2)$?

respectively

Problem

Input: A context-free grammar generating a language over an alphabet Σ.

Question: Is $L(G) = \Sigma^*$?

Reduction between Problems

If we have already proved a (decision) problem to be undecidable, we can prove undecidability of other problems by reductions.

Let us say that A and B are decision problems.
A **reduction** of problem A to problem B is an algorithm P such that:

- It gets as an input an instance of problem A (denoted x).
- It produces an instance of problem B as an output (denoted $P(x)$).
- It holds for every instance x of problem A that the answer for x in problem A is YES iff the answer for $P(x)$ in problem B is YES.

Reduction between Problems

Let us say there is some reduction P from problem A to problem B.

If problem B is decidable then problem A is also decidable.

Solution of problem A for an input x:

- Call P with x as an input, it returns a value $P(x)$.
- Call the algorithm solving problem B with input $P(x)$.
- Write the returned value to the output as the result.

It is obvious that if A is undecidable then B cannot be decidable.

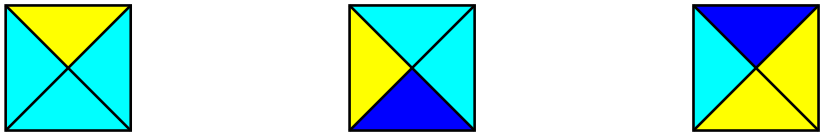
Other Undecidable Problems

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?
- ...

Other Undecidable Problems

An input is a set of types of tiles, such as:

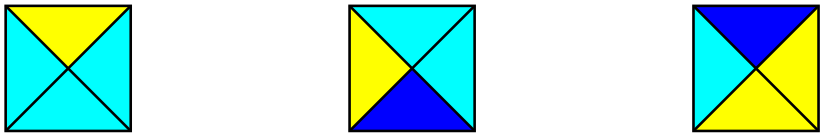


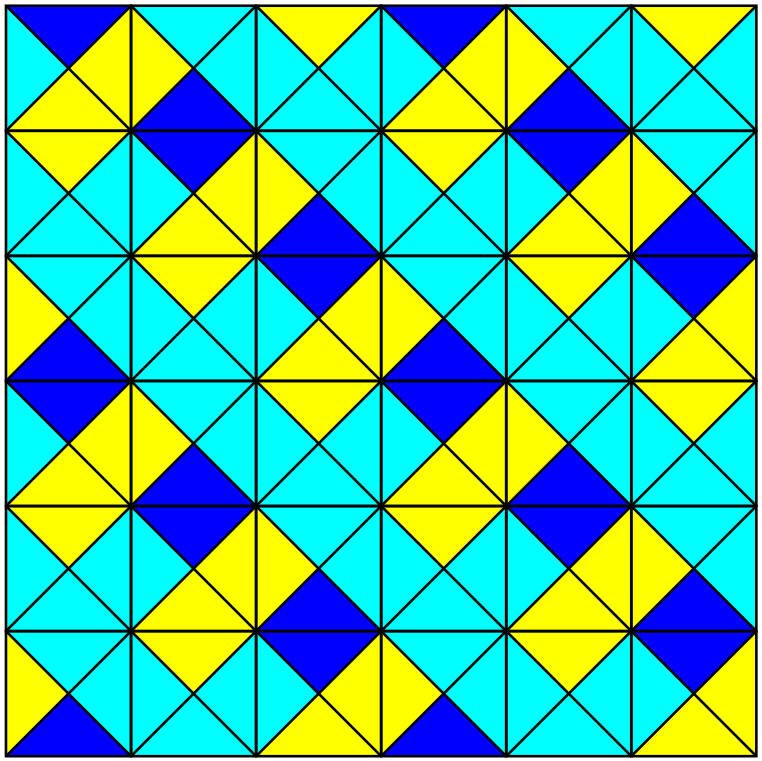
The question is whether it is possible to cover every finite area of an arbitrary size using the given types of tiles in such a way that the colors of neighboring tiles agree.

Remark: We can assume that we have an infinite number of tiles of all types.

The tiles cannot be rotated.

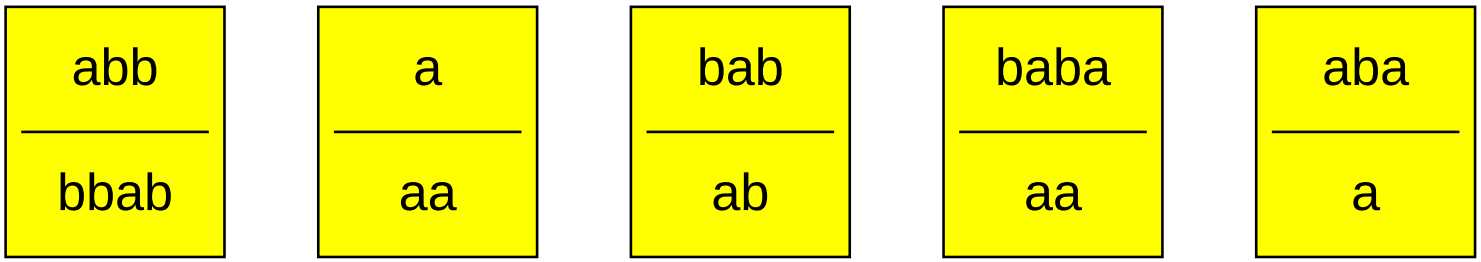
Other Undecidable Problems



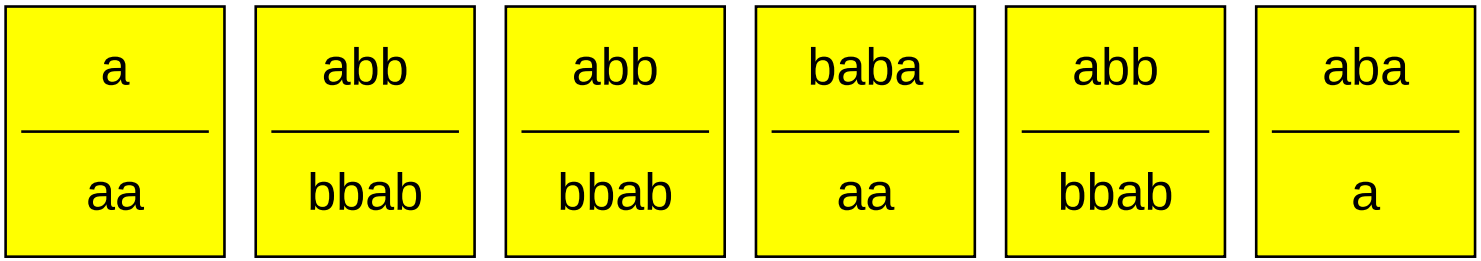


Other Undecidable Problems

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 G_1 and G_2.

Question: Is $L(G_1) \cap L(G_2) = \emptyset$?

Problem

Input: A context-free grammar G.

Question: Is G ambiguous?

Other Undecidable Problems

Problem

Input: A closed formula of FOL where $+$, $*$, and integer constants can be used as function symbols and $=$ and $<$ as predicate symbols.

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:

$$\forall x \exists y \forall z((x * y = z) \wedge (y + 5 = x))$$

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 FOL.

Question: Is $\models \varphi$?

Remark: Notation $\models \varphi$ denotes that formula φ is logically valid, i.e., it is true in all interpretations.

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.