# Synchronizing Finite Automata: an Overview

Mikhail Volkov

Ural Federal University, Ekaterinburg, Russia

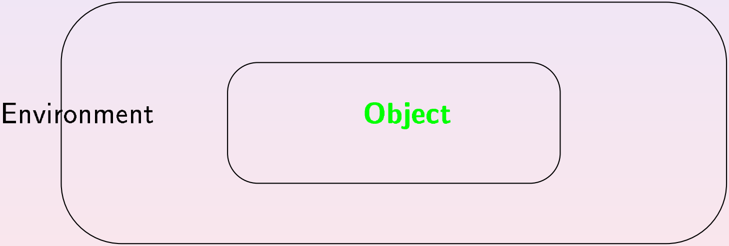
ICADM 2014, March 5th

# Where I am from?

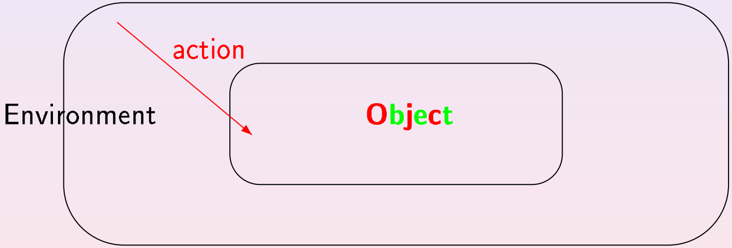
# Finite Automata

A finite automaton is a simple but extremely productive concept that captures the idea of an object interacting with an environment.

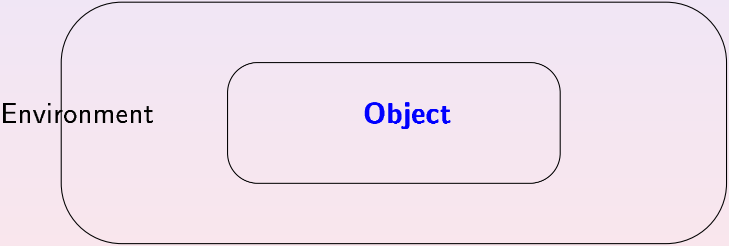
A finite automaton is a simple but extremely productive concept that captures the idea of an object interacting with an environment.



A finite automaton is a simple but extremely productive concept that captures the idea of an object interacting with an environment.



A finite automaton is a simple but extremely productive concept that captures the idea of an object interacting with an environment.



This notion originates in the seminal work by Alan Turing ("On Computable Numbers, With an Application to the Entscheidungsproblem", Proc. London Math. Soc., Ser. 2, 42 (1936), 230–265).

*"The behavior of the computer at any moment is determined by the symbols which he is observing, and his state of mind at that moment"*.

Another important source is the work by neurobiologists Warren McCulloch and Walter Pitts ("A Logical Calculus of the Ideas Immanent in Nervous Activity", Bull. Math. Biophys. 5 (1943), 115–133).

This notion originates in the seminal work by Alan Turing ("On Computable Numbers, With an Application to the Entscheidungsproblem", Proc. London Math. Soc., Ser. 2, 42 (1936), 230–265).

*"The behavior of the computer at any moment is determined by the symbols which he is observing, and his state of mind at that moment"*.

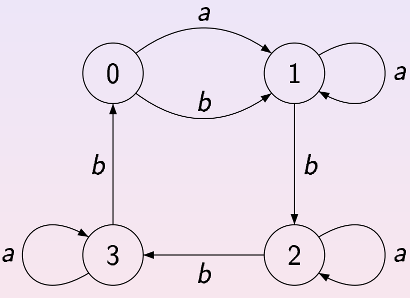
Another important source is the work by neurobiologists Warren McCulloch and Walter Pitts ("A Logical Calculus of the Ideas Immanent in Nervous Activity", Bull. Math. Biophys. 5 (1943), 115–133).

# Visualization

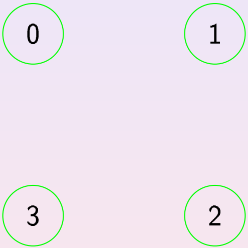
Finite automata admit a convenient visual representation.

# Visualization

Finite automata admit a convenient visual representation.

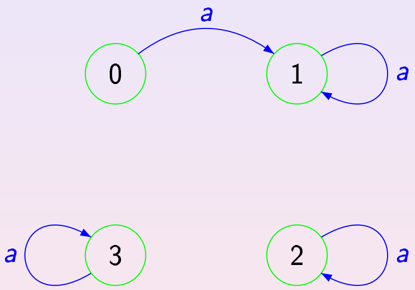


Finite automata admit a convenient visual representation.




Here one sees 4 **states** called 0,1,2,3,

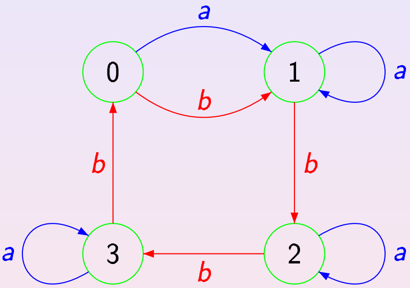
Finite automata admit a convenient visual representation.




Here one sees 4 **states** called 0,1,2,3, an action called *a*

# Visualization

Finite automata admit a convenient visual representation.




Here one sees 4 **states** called 0,1,2,3, an action called $a$ and another action called $b$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the finite state set;
- $\Sigma$ is the input finite alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the totally defined transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.

$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.

The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.

To simplify notation we often write $q \, . \, w$ for $\delta(q, w)$

and $P \, . \, w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and Terminology

We consider complete deterministic finite automata:

$$\mathscr{A} = \langle Q, \Sigma, \delta \rangle.$$

Here

- $Q$ is the state set;
- $\Sigma$ is the input alphabet;
- $\delta : Q \times \Sigma \to Q$ is the transition function.

We need neither initial nor final states.
$\Sigma^*$ stands for the set of all words over $\Sigma$ including the empty word.
The function $\delta$ uniquely extends to a function $Q \times \Sigma^* \to Q$ still denoted by $\delta$.
To simplify notation we often write $q\,.\,w$ for $\delta(q, w)$
and $P\,.\,w$ for $\{\delta(q, w) \mid q \in P\}$.

# Definitions and terminology

An automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is called synchronizing if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in $Q$ it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

We can also write this as $|Q \, . \, w| = 1$.

Any word $w$ with this property is a reset word for $\mathscr{A}$.

Other names:

- for automata: directable, cofinal, collapsible, etc;
- for words: directing, recurrent, synchronizing, etc.

# Definitions and terminology

An automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is called synchronizing if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in $Q$ it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

We can also write this as $|Q \,.\, w| = 1$.

Any word $w$ with this property is a reset word for $\mathscr{A}$.

Other names:

- for automata: directable, cofinal, collapsible, etc;
- for words: directing, recurrent, synchronizing, etc.

# Definitions and terminology

An automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is called synchronizing if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in $Q$ it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

We can also write this as $|Q \,.\, w| = 1$.

Any word $w$ with this property is a reset word for $\mathscr{A}$.

Other names:

- for automata: directable, cofinal, collapsible, etc;
- for words: directing, recurrent, synchronizing, etc.

# Definitions and terminology

An automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is called synchronizing if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in $Q$ it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

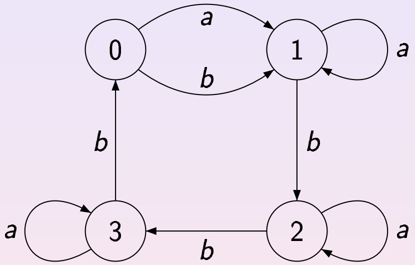
We can also write this as $|Q \,.\, w| = 1$.

Any word $w$ with this property is a reset word for $\mathscr{A}$.

Other names:

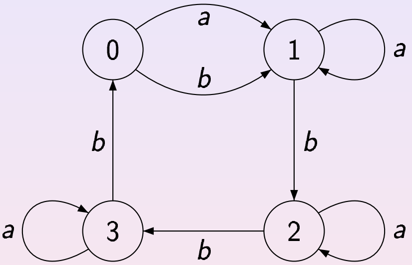
- for automata: directable, cofinal, collapsible, etc;
- for words: directing, recurrent, synchronizing, etc.

# An Example



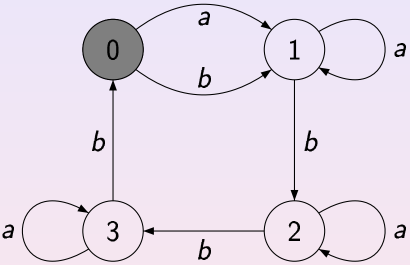

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



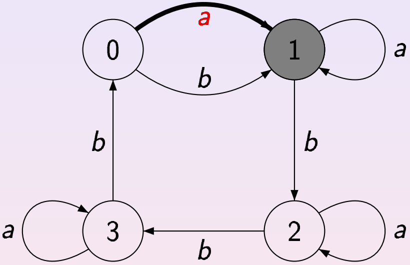

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



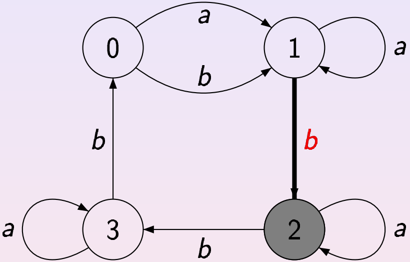

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



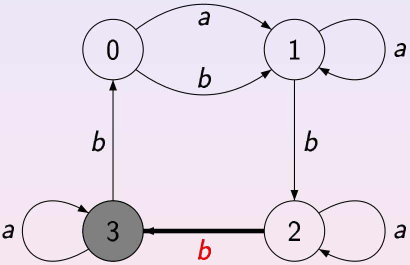

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



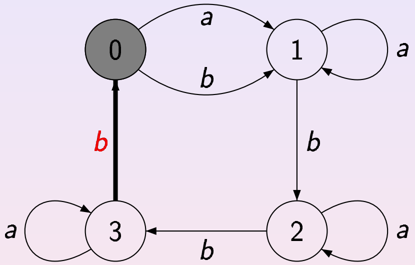

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



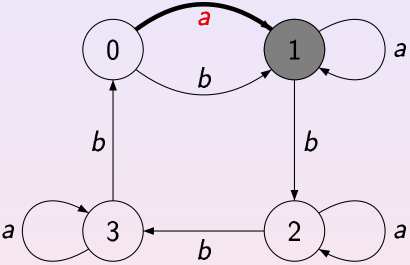

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



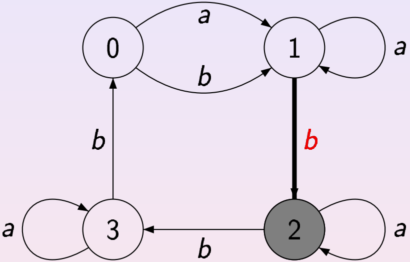

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



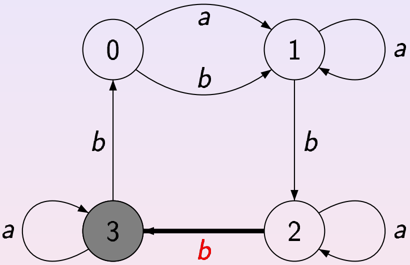

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



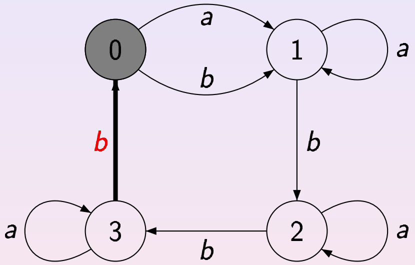

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example

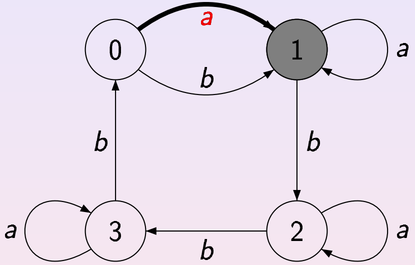



A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example

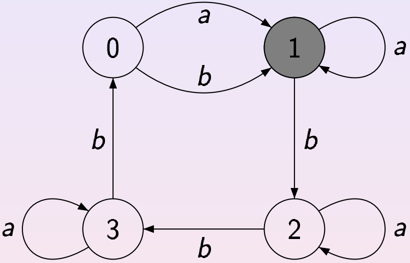



A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.



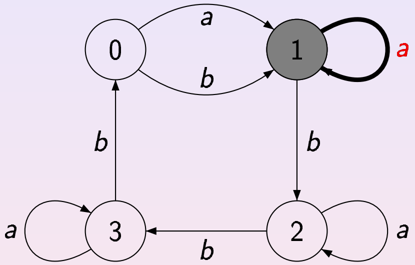

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



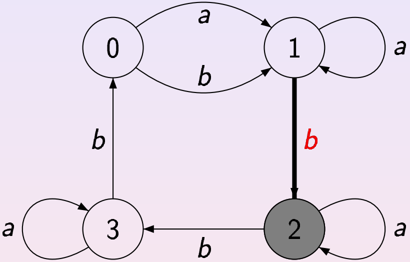

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



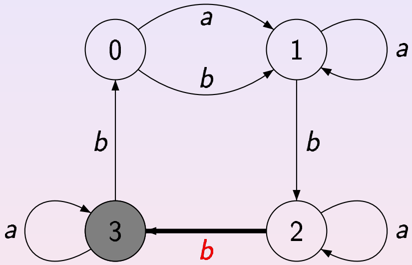

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



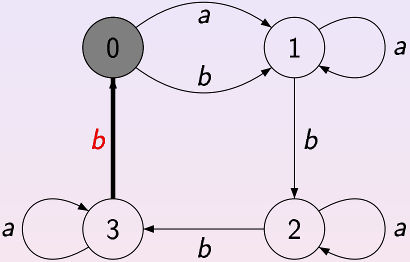

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



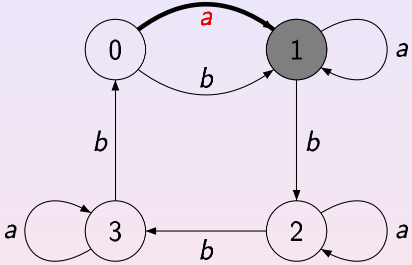

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



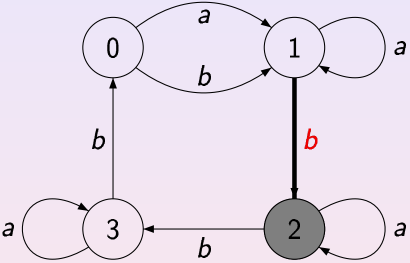

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



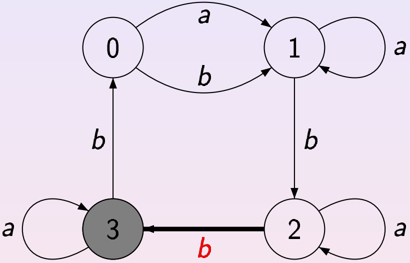

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



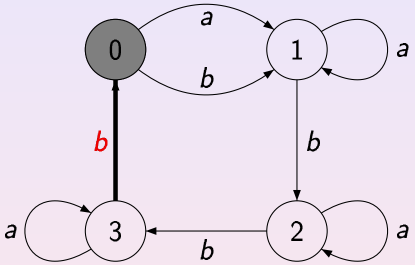

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



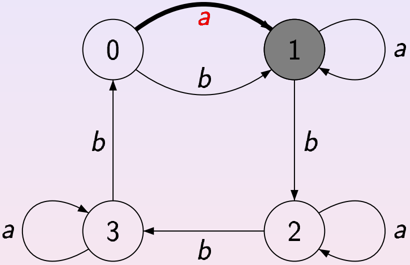

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



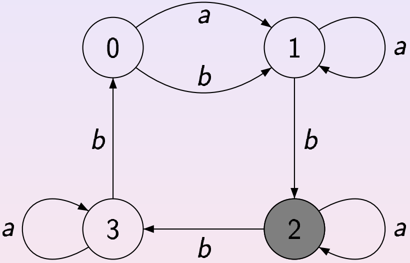

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



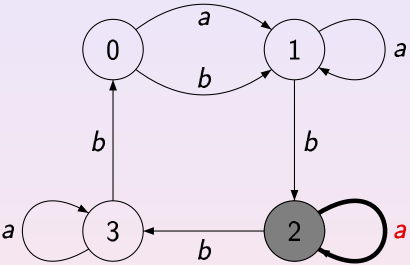

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



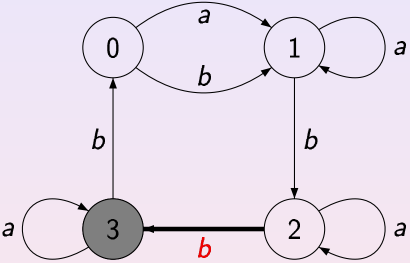

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



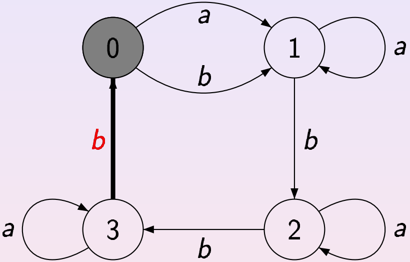

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



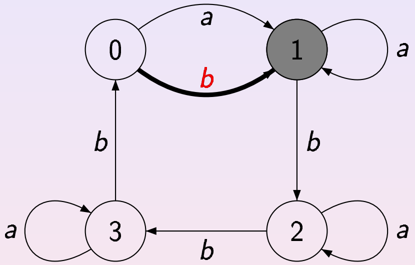

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



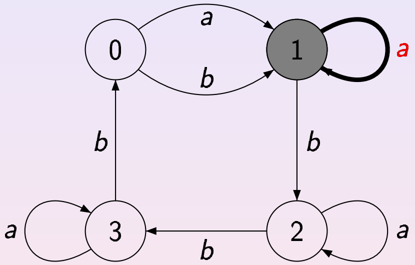

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



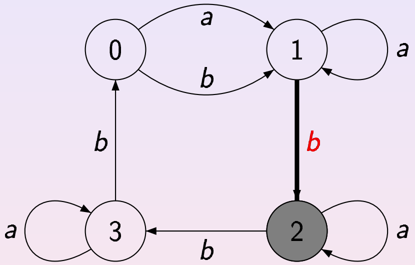

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



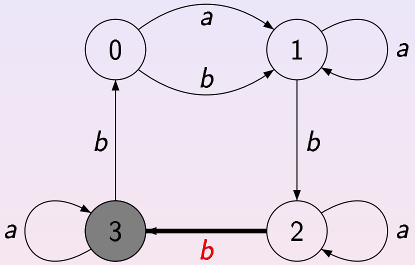

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



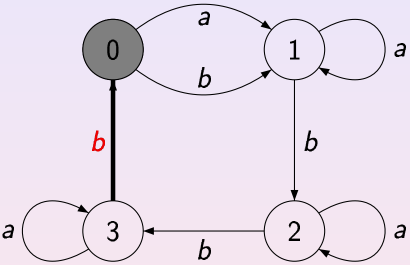

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



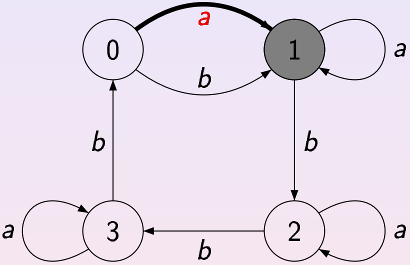

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



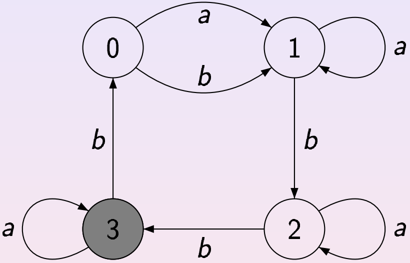

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



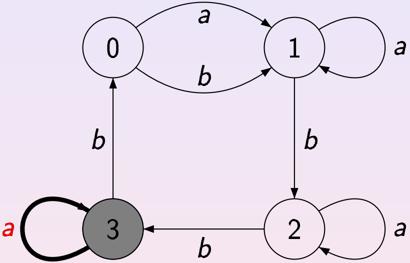

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



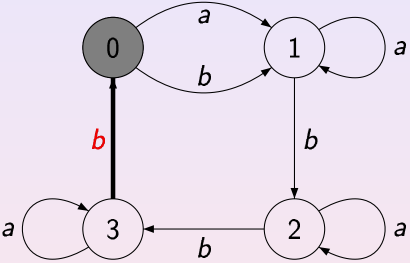

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



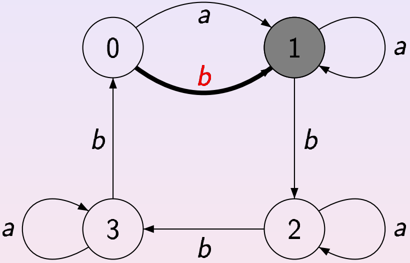

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



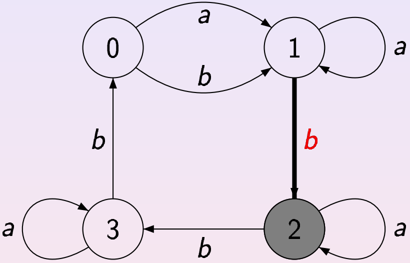

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



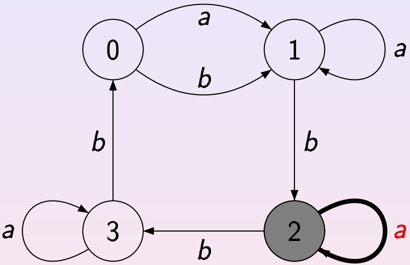

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



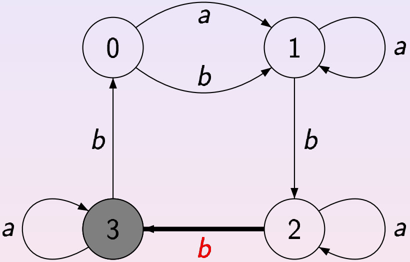

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example



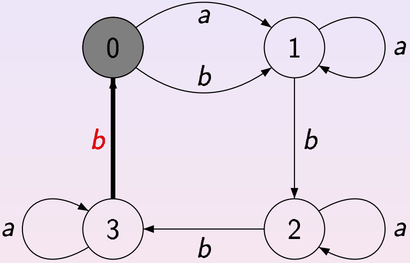

A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example

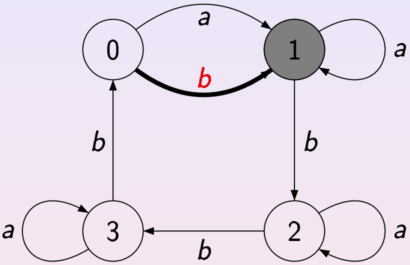



A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example

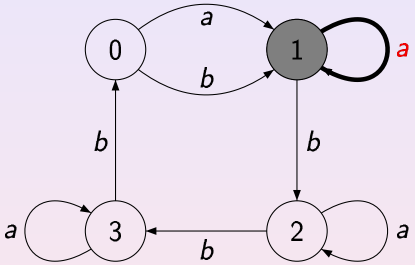



A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.




A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

# An Example




A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

The notion was formalized in 1964 in a paper by Jan Černý (Poznámka k homogénnym eksperimentom s konečnými automatami, Matematicko-fyzikalny Časopis Slovensk. Akad. Vied, 14, no.3, 208–216 [in Slovak]) though implicitly it had been around since at least 1956.

The idea of synchronization is pretty natural and of obvious importance: we aim to restore control over a device whose current state is not known.

Think of a satellite which loops around the Moon and cannot be controlled from the Earth while "behind" the Moon (Černý's original motivation).

The notion was formalized in 1964 in a paper by Jan Černý (Poznámka k homogénnym eksperimentom s konečnými automatami, Matematicko-fyzikalny Časopis Slovensk. Akad. Vied, 14, no.3, 208–216 [in Slovak]) though implicitly it had been around since at least 1956.

The idea of synchronization is pretty natural and of obvious importance: we aim to restore control over a device whose current state is not known.

Think of a satellite which loops around the Moon and cannot be controlled from the Earth while "behind" the Moon (Černý's original motivation).

# Cerný's Paper

The notion was formalized in 1964 in a paper by Jan Černý (Poznámka k homogénnym eksperimentom s konečnými automatami, Matematicko-fyzikalny Časopis Slovensk. Akad. Vied, 14, no.3, 208–216 [in Slovak]) though implicitly it had been around since at least 1956.

The idea of synchronization is pretty natural and of obvious importance: we aim to restore control over a device whose current state is not known.

Think of a satellite which loops around the Moon and cannot be controlled from the Earth while "behind" the Moon (Černý's original motivation).

# Other Sources

It is not surprising that synchronizing automata were re-invented a number of times:

- The notion was very natural by itself and fitted fairly well in what was considered as the mainstream of automata theory in the 1960s.
- Černý's paper published in Slovak language remained unknown in the English-speaking world for quite a long time.

Example: A. E. Laemmel, B. Rudner, Study of the application of coding theory, Report PIBEP-69-034, Polytechnic Inst. Brooklyn, Dept. Electrophysics, Farmingdale, N.Y., 94 pp.

# Other Sources

It is not surprising that synchronizing automata were re-invented a number of times:

- The notion was very natural by itself and fitted fairly well in what was considered as the mainstream of automata theory in the 1960s.
- Černý's paper published in Slovak language remained unknown in the English-speaking world for quite a long time.

Example: A. E. Laemmel, B. Rudner, Study of the application of coding theory, Report PIBEP-69-034, Polytechnic Inst. Brooklyn, Dept. Electrophysics, Farmingdale, N.Y., 94 pp.

# Other Sources

It is not surprising that synchronizing automata were re-invented a number of times:

• The notion was very natural by itself and fitted fairly well in what was considered as the mainstream of automata theory in the 1960s.

• Černý's paper published in Slovak language remained unknown in the English-speaking world for quite a long time.

Example: A. E. Laemmel, B. Rudner, Study of the application of coding theory, Report PIBEP-69-034, Polytechnic Inst. Brooklyn, Dept. Electrophysics, Farmingdale, N.Y., 94 pp.

# Other Sources

It is not surprising that synchronizing automata were re-invented a number of times:

- The notion was very natural by itself and fitted fairly well in what was considered as the mainstream of automata theory in the 1960s.
- Černý's paper published in Slovak language remained unknown in the English-speaking world for quite a long time.

Example: A. E. Laemmel, B. Rudner, Study of the application of coding theory, Report PIBEP-69-034, Polytechnic Inst. Brooklyn, Dept. Electrophysics, Farmingdale, N.Y., 94 pp.

A prefix code over a finite alphabet $\Sigma$ is a set $X$ of words in $\Sigma^*$ such that no word of $X$ is a prefix of another word of $X$. A prefix code is maximal if it is not contained in another prefix code over the same alphabet. A maximal prefix code $X$ over $\Sigma$ is synchronized if there is a word $x \in X^*$ such that for any word $w \in \Sigma^*$, one has $wx \in X^*$. Such a word $x$ is called a synchronizing word for $X$.

The advantage of synchronized codes is that they are able to recover after a loss of synchronization between the decoder and the coder caused by channel errors.

A prefix code over a finite alphabet $\Sigma$ is a set $X$ of words in $\Sigma^*$ such that no word of $X$ is a prefix of another word of $X$. A prefix code is maximal if it is not contained in another prefix code over the same alphabet. A maximal prefix code $X$ over $\Sigma$ is synchronized if there is a word $x \in X^*$ such that for any word $w \in \Sigma^*$, one has $wx \in X^*$. Such a word $x$ is called a synchronizing word for $X$.

The advantage of synchronized codes is that they are able to recover after a loss of synchronization between the decoder and the coder caused by channel errors.

A prefix code over a finite alphabet $\Sigma$ is a set $X$ of words in $\Sigma^*$ such that no word of $X$ is a prefix of another word of $X$. A prefix code is maximal if it is not contained in another prefix code over the same alphabet. A maximal prefix code $X$ over $\Sigma$ is synchronized if there is a word $x \in X^*$ such that for any word $w \in \Sigma^*$, one has $wx \in X^*$. Such a word $x$ is called a synchronizing word for $X$.

The advantage of synchronized codes is that they are able to recover after a loss of synchronization between the decoder and the coder caused by channel errors.

A prefix code over a finite alphabet $\Sigma$ is a set $X$ of words in $\Sigma^*$ such that no word of $X$ is a prefix of another word of $X$. A prefix code is maximal if it is not contained in another prefix code over the same alphabet. A maximal prefix code $X$ over $\Sigma$ is synchronized if there is a word $x \in X^*$ such that for any word $w \in \Sigma^*$, one has $wx \in X^*$. Such a word $x$ is called a synchronizing word for $X$.

The advantage of synchronized codes is that they are able to recover after a loss of synchronization between the decoder and the coder caused by channel errors.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$.
Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

Sent 0 0 0

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, . . . is a synchronizing word for $X$.

Sent 0 0 0 | 0 0 1 0

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

Sent   0 0 0 | 0 0 1 0 | 0 1 1 1 | . . .

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

$\Sigma = \{0,1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| 0 1 1 1 \| . . . |
| Received | 1 0 0  0 0 1 0  0 1 1 1 . . . |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| 0 1 1 1 \| . . . |
| Received | 1 0 0 0 **0 1 0** 0 1 1 1 . . . |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

$\Sigma = \{0,1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| 0 1 1 1 \| . . . |
| Received | 1 0 0 0 0 1 0 0 1 1 1 . . . |
| Decoded | 1 0 |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$.
Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, . . . is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| 0 1 1 1 \| . . . |
| Received | 1 0 0 0 0 1 0 0 1 1 1 . . . |
| Decoded | 1 0 \| 0 0 0 |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0,1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| 0 1 1 1 \| . . . |
| Received | 1 0 0 0 0 1 0 0 1 1 1 . . . |
| Decoded | 1 0 \| 0 0 0 \| 1 0 |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0,1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| **0 1 1 1** \| . . . |
| Received | 1 0 0 0 0 1 0 0 1 1 1 . . . |
| Decoded | 1 0 \| 0 0 0 \| 1 0 \| **0 1 1 1** \| . . . |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

# Synchronized Codes

$\Sigma = \{0, 1\}$, $X = \{000, 0010, 0011, 010, 0110, 0111, 10, 110, 111\}$. Then $X$ is a maximal prefix code and one can easily check that each of the words 010, 011110, 011111110, ... is a synchronizing word for $X$.

| | |
|---|---|
| Sent | 0 0 0 \| 0 0 1 0 \| **0 1 1 1** \| . . . |
| Received | 1 0 0 0 0 1 0 0 1 1 1 . . . |
| Decoded | 1 0 \| 0 0 0 \| 1 0 \| **0 1 1 1** \| . . . |

The vertical lines show the partition of each stream into code words and the boldfaced code words indicate the position at which the decoder resynchronizes.

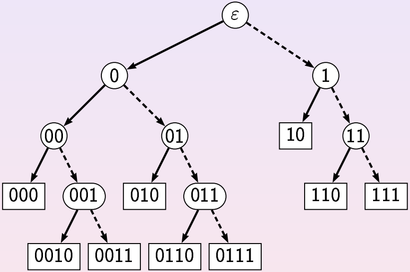
# Codes vs Automata

If $X$ is a finite maximal prefix code, then its decoding can be implemented by a DFA.

Synchronized codes precisely correspond to synchronizing automata!

# Codes vs Automata

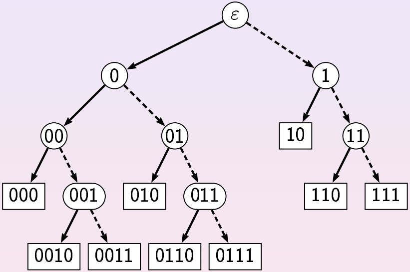
If $X$ is a finite maximal prefix code, then its decoding can be implemented by a DFA.

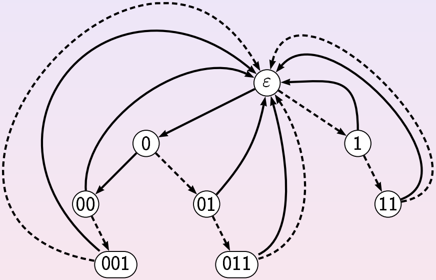



Synchronized codes precisely correspond to synchronizing automata!

# Codes vs Automata

If $X$ is a finite maximal prefix code, then its decoding can be implemented by a DFA.

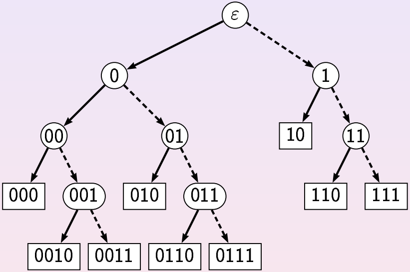


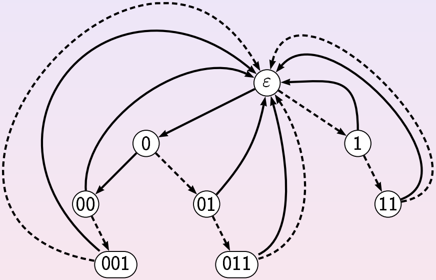




Synchronized codes precisely correspond to synchronizing automata!

# Codes vs Automata

If $X$ is a finite maximal prefix code, then its decoding can be implemented by a DFA.




Synchronized codes precisely correspond to synchronizing automata!

# Re-inventing by Engineers

Since the 60s synchronizing automata have been considered as a useful tool for testing of reactive systems (first circuits, later protocols) and have been also applied in coding theory.

In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.

Suppose that one of the parts of a certain device has the following shape:



Such parts arrive at manufacturing sites in boxes and they need to be sorted and oriented before assembly.

# Re-inventing by Engineers

Since the 60s synchronizing automata have been considered as a useful tool for testing of reactive systems (first circuits, later protocols) and have been also applied in coding theory.

In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.

Suppose that one of the parts of a certain device has the following shape:



Such parts arrive at manufacturing sites in boxes and they need to be sorted and oriented before assembly.

# Re-inventing by Engineers

Since the 60s synchronizing automata have been considered as a useful tool for testing of reactive systems (first circuits, later protocols) and have been also applied in coding theory.

In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.

Suppose that one of the parts of a certain device has the following shape:

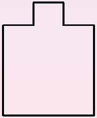


Such parts arrive at manufacturing sites in boxes and they need to be sorted and oriented before assembly.

Since the 60s synchronizing automata have been considered as a useful tool for testing of reactive systems (first circuits, later protocols) and have been also applied in coding theory.

In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.

Suppose that one of the parts of a certain device has the following shape:



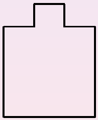
Such parts arrive at manufacturing sites in boxes and they need to be sorted and oriented before assembly.

# Re-inventing by Engineers

Since the 60s synchronizing automata have been considered as a useful tool for testing of reactive systems (first circuits, later protocols) and have been also applied in coding theory.

In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.

Suppose that one of the parts of a certain device has the following shape:



Such parts arrive at manufacturing sites in boxes and they need to be sorted and oriented before assembly.

# Re-inventing by Engineers

Assume that only four initial orientations of the part shown above are possible, namely, the following ones:

Suppose that prior the assembly the part should take the 'bump-left' orientation (the second one in the picture). Thus, one has to construct an orienter which action will put the part in the prescribed position independently of its initial orientation.

# Re-inventing by Engineers

Assume that only four initial orientations of the part shown above are possible, namely, the following ones:

Suppose that prior the assembly the part should take the 'bump-left' orientation (the second one in the picture). Thus, one has to construct an orienter which action will put the part in the prescribed position independently of its initial orientation.

# Re-inventing by Engineers

We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles of two types (*high* and *low*) placed along the belt.

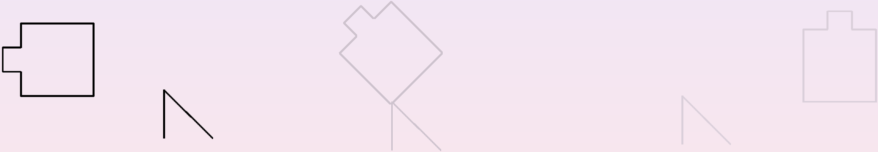
A high obstacle is high enough so that any part on the belt encounters this obstacle by its rightmost low angle.



Being carried by the belt, the part then is forced to turn 90° clockwise.

# Re-inventing by Engineers

We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles of two types (*high* and *low*) placed along the belt.
A high obstacle is high enough so that any part on the belt encounters this obstacle by its rightmost low angle.

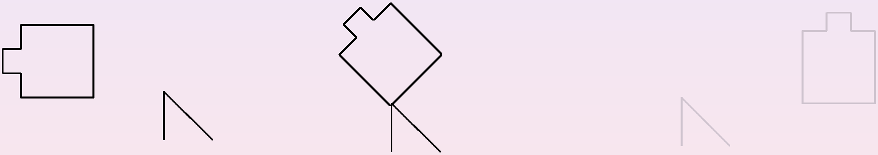


Being carried by the belt, the part then is forced to turn 90° clockwise.

# Re-inventing by Engineers

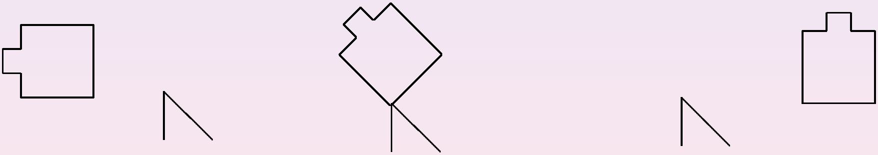
We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles of two types (*high* and *low*) placed along the belt.

A high obstacle is high enough so that any part on the belt encounters this obstacle by its rightmost low angle.



Being carried by the belt, the part then is forced to turn 90° clockwise.

We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles of two types (*high* and *low*) placed along the belt.
A high obstacle is high enough so that any part on the belt encounters this obstacle by its rightmost low angle.



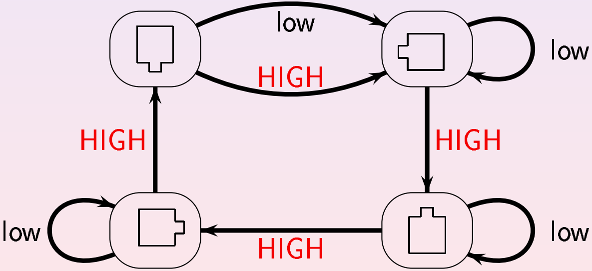
Being carried by the belt, the part then is forced to turn 90° clockwise.

# Re-inventing by Engineers

A low obstacle has the same effect whenever the part is in the "bump-down" orientation; otherwise it does not touch the part which therefore passes by without changing the orientation.

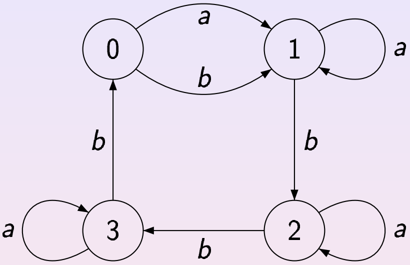
# Re-inventing by Engineers

A low obstacle has the same effect whenever the part is in the "bump-down" orientation; otherwise it does not touch the part which therefore passes by without changing the orientation. The following schema summarizes how the obstacles effect the orientation of the part in question:



# Re-inventing by Engineers

We met this picture a few slides ago:



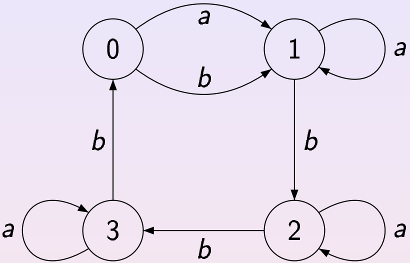

– this was our example of a synchronizing automaton, and we saw that *abbbabbba* is a reset sequence of actions. Hence the series of obstacles

low-HIGH-HIGH-HIGH-low-HIGH-HIGH-HIGH-low

yields the desired sensorless orienter.

# Re-inventing by Engineers

We met this picture a few slides ago:




– this was our example of a synchronizing automaton, and we saw that *abbbabbba* is a reset sequence of actions. Hence the series of obstacles

low-HIGH-HIGH-HIGH-low-HIGH-HIGH-HIGH-low

yields the desired sensorless orienter.

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20.$ Calculate the iterations of $\tau$ up to $\tau^4$:

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

$$\begin{array}{ccc} 0 & \mapsto & 11 \\ 1 & \mapsto & 12 \\ 2 & \mapsto & 20 \end{array}$$

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

$$\begin{array}{ccccc} 0 & \mapsto & 11 & \mapsto & 1212 \\ 1 & \mapsto & 12 & \mapsto & 1220 \\ 2 & \mapsto & 20 & \mapsto & 2011 \end{array}$$

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

$$\begin{array}{ccccccc} 0 & \mapsto & 11 & \mapsto & 1212 & \mapsto & 12201220 \\ 1 & \mapsto & 12 & \mapsto & 1220 & \mapsto & 12202011 \\ 2 & \mapsto & 20 & \mapsto & 2011 & \mapsto & 20111212 \end{array}$$

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0,1,2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

$$\begin{array}{ccccccccc}
0 & \mapsto & 11 & \mapsto & 1212 & \mapsto & 12201220 & \mapsto & 1220201112202011 \\
1 & \mapsto & 12 & \mapsto & 1220 & \mapsto & 12202011 & \mapsto & 1220201120111212 \\
2 & \mapsto & 20 & \mapsto & 2011 & \mapsto & 20111212 & \mapsto & 2011121212201220
\end{array}$$

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

$$\begin{array}{ccccccccc} 0 & \mapsto & 11 & \mapsto & 1212 & \mapsto & 12201220 & \mapsto & 122020\mathbf{1}112202011 \\ 1 & \mapsto & 12 & \mapsto & 1220 & \mapsto & 12202011 & \mapsto & 122020\mathbf{1}120111212 \\ 2 & \mapsto & 20 & \mapsto & 2011 & \mapsto & 20111212 & \mapsto & 201112\mathbf{1}212201220 \end{array}$$

Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$).
The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

A substitution on a finite alphabet $X$ is a map $\sigma : X \to X^+$; the substitution is said to be of constant length if all words $\sigma(x)$, $x \in X$, have the same length. One says that $\sigma$ satisfies the coincidence condition if there exist positive integers $m$ and $k$ such that all words $\sigma^k(x)$ have the same letter in the $m$-th position. For an example, consider the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11,\ 1 \mapsto 12,\ 2 \mapsto 20$. Calculate the iterations of $\tau$ up to $\tau^4$:

$$\begin{array}{ccccccccc} 0 & \mapsto & 11 & \mapsto & 1212 & \mapsto & 12201220 & \mapsto & 122020\mathbf{1}112202011 \\ 1 & \mapsto & 12 & \mapsto & 1220 & \mapsto & 12202011 & \mapsto & 122020\mathbf{1}120111212 \\ 2 & \mapsto & 20 & \mapsto & 2011 & \mapsto & 20111212 & \mapsto & 201112\mathbf{1}212201220 \end{array}$$

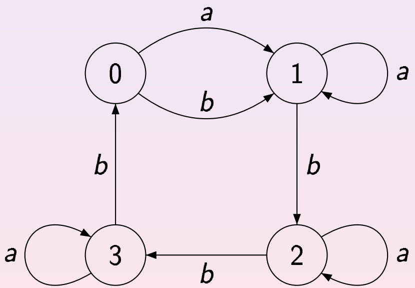
Thus, $\tau$ satisfies the coincidence condition (with $k = 4$, $m = 7$). The coincidence condition completely characterizes the constant length substitutions that give rise to dynamical systems measure-theoretically isomorphic to a translation on a compact Abelian group (Dekking, 1978).

# Re-inventing by Dynamics Theorists

There is a straightforward bijection between DFAs and constant length substitutions. Each DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ with $\Sigma = \{a_1, \dots, a_\ell\}$ defines a length $\ell$ substitution on $Q$ that maps every $q \in Q$ to the word $(q \,.\, a_1) \dots (q \,.\, a_\ell) \in Q^+$.

# Re-inventing by Dynamics Theorists

There is a straightforward bijection between DFAs and constant length substitutions. Each DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ with $\Sigma = \{a_1, \dots, a_\ell\}$ defines a length $\ell$ substitution on $Q$ that maps every $q \in Q$ to the word $(q \,.\, a_1) \dots (q \,.\, a_\ell) \in Q^+$. For instance, the automaton




induces the substitution $0 \mapsto 11, \ 1 \mapsto 12, \ 2 \mapsto 23, \ 3 \mapsto 30$.

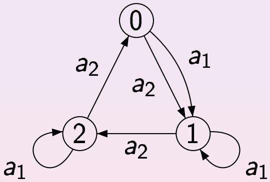
# Re-inventing by Dynamics Theorists

Conversely, each substitution $\sigma : X \to X^+$ such that all words $\sigma(x)$, $x \in X$, have the same length $\ell$ gives rise to a DFA for which $X$ is the state set and which has $\ell$ input letters $a_1, \dots, a_\ell$ acting on $X$ as follows: $x\,.\,a_i$ is the symbol in the $i$-th position of the word $\sigma(x)$.

Under this bijection substitutions satisfying the coincidence condition correspond precisely to synchronizing automata, and moreover, given a substitution, the number of iterations at which the coincidence first occurs is equal to the minimum length of reset word for the corresponding automaton.

# Re-inventing by Dynamics Theorists

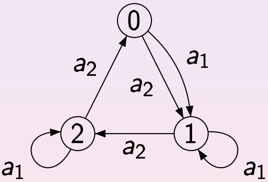
Conversely, each substitution $\sigma : X \to X^+$ such that all words $\sigma(x)$, $x \in X$, have the same length $\ell$ gives rise to a DFA for which $X$ is the state set and which has $\ell$ input letters $a_1, \dots, a_\ell$ acting on $X$ as follows: $x \,.\, a_i$ is the symbol in the $i$-th position of the word $\sigma(x)$. For instance, the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11$, $1 \mapsto 12$, $2 \mapsto 20$ induces the automaton:




Under this bijection substitutions satisfying the coincidence condition correspond precisely to synchronizing automata, and moreover, given a substitution, the number of iterations at which the coincidence first occurs is equal to the minimum length of reset word for the corresponding automaton.

# Re-inventing by Dynamics Theorists

Conversely, each substitution $\sigma : X \to X^+$ such that all words $\sigma(x)$, $x \in X$, have the same length $\ell$ gives rise to a DFA for which $X$ is the state set and which has $\ell$ input letters $a_1, \dots, a_\ell$ acting on $X$ as follows: $x \,.\, a_i$ is the symbol in the $i$-th position of the word $\sigma(x)$. For instance, the substitution $\tau$ on $X = \{0, 1, 2\}$ defined by $0 \mapsto 11$, $1 \mapsto 12$, $2 \mapsto 20$ induces the automaton:




Under this bijection substitutions satisfying the coincidence condition correspond precisely to synchronizing automata, and moreover, given a substitution, the number of iterations at which the coincidence first occurs is equal to the minimum length of reset word for the corresponding automaton.

# Possible Use in Biocomputing

In DNA-computing, there is fast progressing work by Ehud Shapiro's group on "*soup of automata*" (Programmable and autonomous computing machine made of biomolecules, Nature 414, no.1 (November 22, 2001) 430–434; DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003) 2191–2196, etc).

They have produced a solution containing $3 \times 10^{12}$ identical DNA-based automata per $\mu$l. These automata can work in parallel on different inputs (DNA strands), thus ending up in different and unpredictable states. One has to feed the automata with a reset sequence (again encoded by a DNA-strand) in order to get them ready for a new use.

# Possible Use in Biocomputing

In DNA-computing, there is fast progressing work by Ehud Shapiro's group on "*soup of automata*" (Programmable and autonomous computing machine made of biomolecules, Nature 414, no.1 (November 22, 2001) 430–434; DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003) 2191–2196, etc).

They have produced a solution containing $3 \times 10^{12}$ identical DNA-based automata per $\mu$l. These automata can work in parallel on different inputs (DNA strands), thus ending up in different and unpredictable states. One has to feed the automata with a reset sequence (again encoded by a DNA-strand) in order to get them ready for a new use.

# Possible Use in Biocomputing

In DNA-computing, there is fast progressing work by Ehud Shapiro's group on "*soup of automata*" (Programmable and autonomous computing machine made of biomolecules, Nature 414, no.1 (November 22, 2001) 430–434; DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003) 2191–2196, etc).
They have produced a solution containing $3 \times 10^{12}$ identical DNA-based automata per $\mu$l. These automata can work in parallel on different inputs (DNA strands), thus ending up in different and unpredictable states. One has to feed the automata with a reset sequence (again encoded by a DNA-strand) in order to get them ready for a new use.

# Possible Use in Biocomputing

In DNA-computing, there is fast progressing work by Ehud Shapiro's group on "*soup of automata*" (Programmable and autonomous computing machine made of biomolecules, Nature 414, no.1 (November 22, 2001) 430–434; DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003) 2191–2196, etc).

They have produced a solution containing $3 \times 10^{12}$ identical DNA-based automata per $\mu$l. These automata can work in parallel on different inputs (DNA strands), thus ending up in different and unpredictable states. One has to feed the automata with a reset sequence (again encoded by a DNA-strand) in order to get them ready for a new use.

# Outline of Theory

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance.
- Synchronizing automata constitute an interesting combinatorial object. Their studies from a combinatorial viewpoint are mainly motivated by the Černý Conjecture: every synchronizing automaton with $n$ states has a reset word of length $(n-1)^2$.
- Connections to symbolic dynamics have led to the Road Coloring Problem which has been recently solved by Trahtman.
- There are also interesting connections with the Perron–Frobenius theory of non-negative matrices and with the theory of Markov chains.

# Outline of Theory

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance.
- Synchronizing automata constitute an interesting combinatorial object. Their studies from a combinatorial viewpoint are mainly motivated by the Černý Conjecture: every synchronizing automaton with $n$ states has a reset word of length $(n-1)^2$.
- Connections to symbolic dynamics have led to the Road Coloring Problem which has been recently solved by Trahtman.
- There are also interesting connections with the Perron–Frobenius theory of non-negative matrices and with the theory of Markov chains.

# Outline of Theory

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance.
- Synchronizing automata constitute an interesting combinatorial object. Their studies from a combinatorial viewpoint are mainly motivated by the Černý Conjecture: every synchronizing automaton with $n$ states has a reset word of length $(n-1)^2$.
- Connections to symbolic dynamics have led to the Road Coloring Problem which has been recently solved by Trahtman.
- There are also interesting connections with the Perron–Frobenius theory of non-negative matrices and with the theory of Markov chains.

# Outline of Theory

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance.
- Synchronizing automata constitute an interesting combinatorial object. Their studies from a combinatorial viewpoint are mainly motivated by the Černý Conjecture: every synchronizing automaton with $n$ states has a reset word of length $(n-1)^2$.
- Connections to symbolic dynamics have led to the Road Coloring Problem which has been recently solved by Trahtman.
- There are also interesting connections with the Perron–Frobenius theory of non-negative matrices and with the theory of Markov chains.