Černý's Conjecture
and the Road Coloring Problem

Mikhail Volkov

Ural State University, Ekaterinburg, Russia

Vienna, November 24, 2010

Definitions and Terminology

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

$\mathscr{A}$ is called synchronizing if there is a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves $\mathscr{A}$ in one particular state no matter at which state in Q it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$. In short, $|Q \,.\, w| = 1$.

Any 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

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

$\mathscr{A}$ is called synchronizing if there is a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves $\mathscr{A}$ in one particular state no matter at which state in Q it started: $q\,.\,w = q'\,.\,w$ for all $q, q' \in Q$.
In short, $|Q\,.\,w| = 1$.

Any 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

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

$\mathscr{A}$ is called synchronizing if there is a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves $\mathscr{A}$ in one particular state no matter at which state in Q it started: $q \, . \, w = q' \, . \, w$ for all $q, q' \in Q$. In short, $|Q \, . \, w| = 1$.

Any 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

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

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

In short, $|Q\,.\,w| = 1$.

Any 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

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

$\mathscr{A}$ is called synchronizing if there is a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves $\mathscr{A}$ in one particular state no matter at which state in Q it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$.
In short, $|Q \,.\, w| = 1$.

Any 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

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

$\mathscr{A}$ is called synchronizing if there is a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves $\mathscr{A}$ in one particular state no matter at which state in Q it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$.
In short, $|Q \,.\, w| = 1$.

Any 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.

Vienna, November 24, 2010

Definitions and Terminology

We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

$\mathscr{A}$ is called synchronizing if there is a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves $\mathscr{A}$ in one particular state no matter at which state in Q it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$.
In short, $|Q \,.\, w| = 1$.

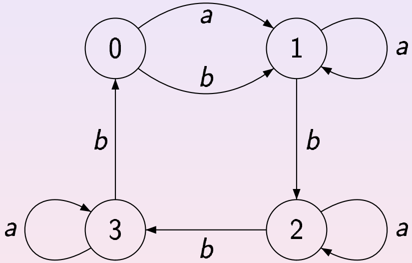
Any w with this property is a reset word for $\mathscr{A}$.

Other names:

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

Vienna, November 24, 2010

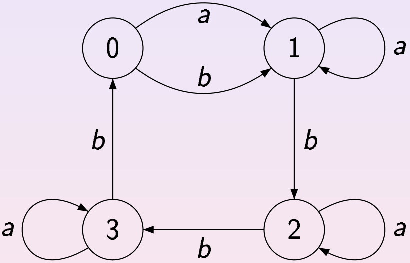
An Example



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

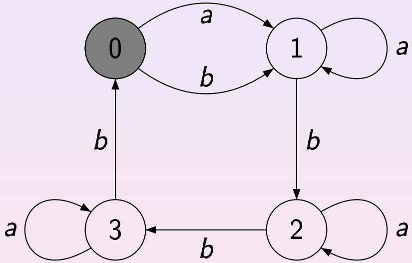
Vienna, November 24, 2010

An Example



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

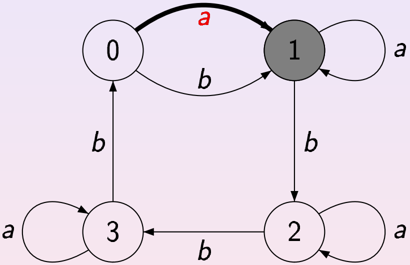
An Example



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

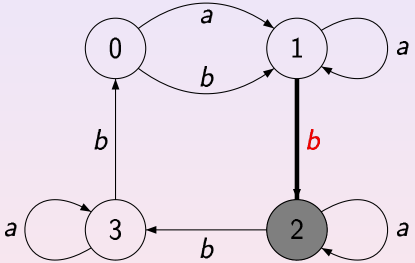
Vienna, November 24, 2010

An Example



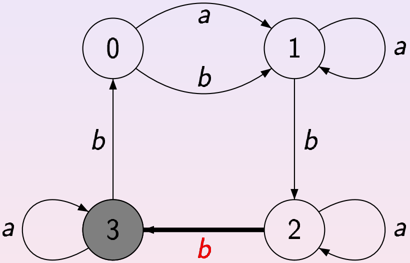
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



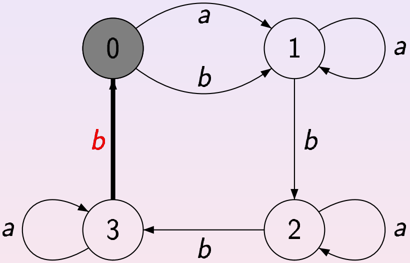
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



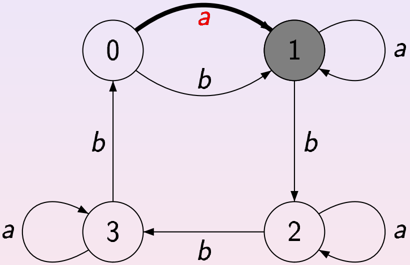
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



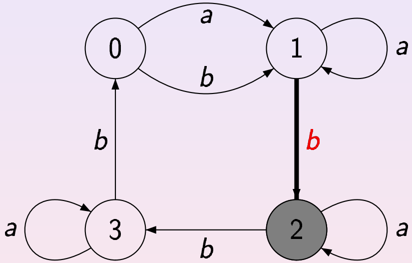
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



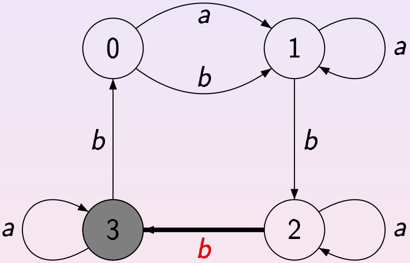
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



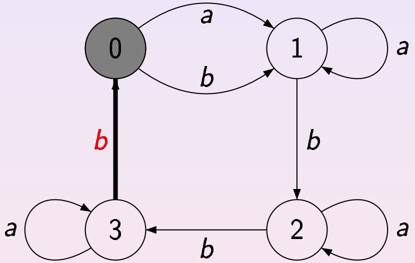
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



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

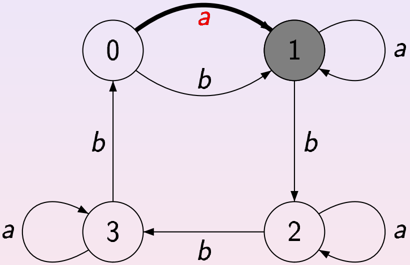
An Example



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

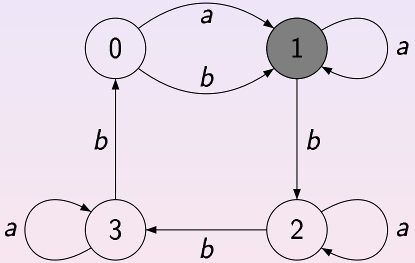
Vienna, November 24, 2010

An Example



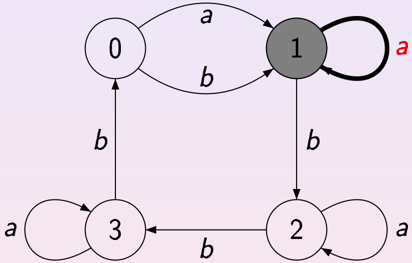
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



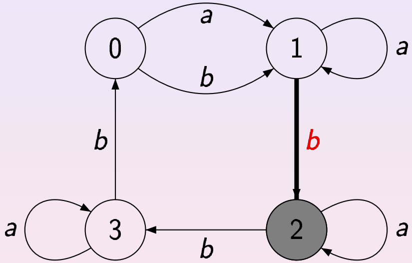
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



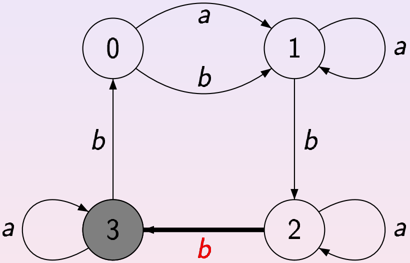
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



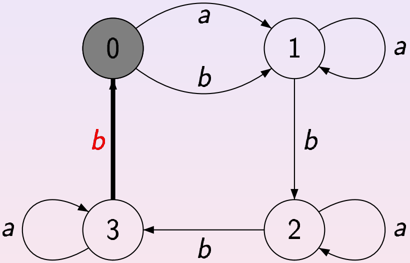
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



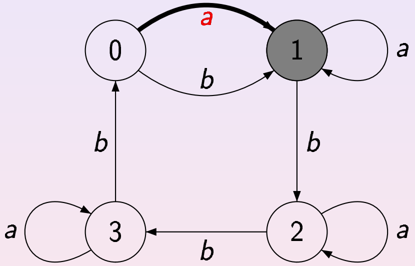
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



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

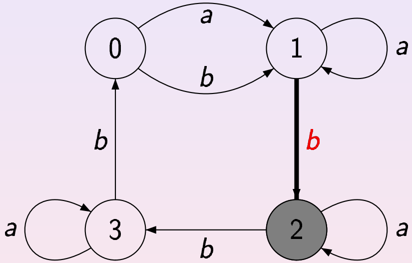
An Example



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

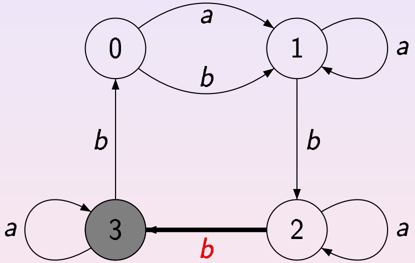
Vienna, November 24, 2010

An Example



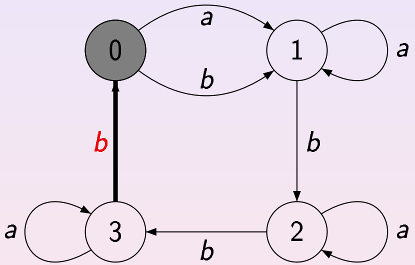
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



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

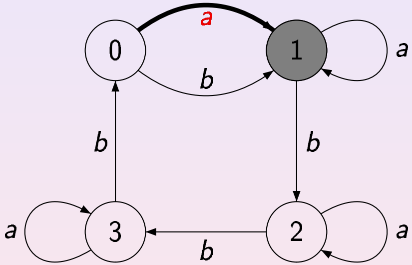
An Example



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

Vienna, November 24, 2010

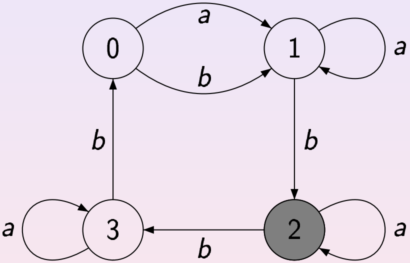
An Example



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

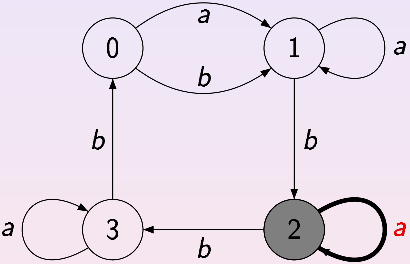
Vienna, November 24, 2010

An Example



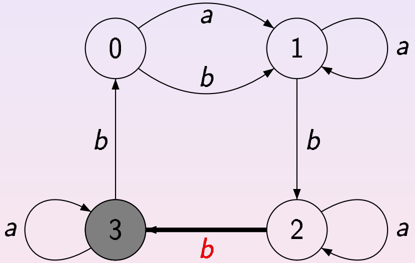
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



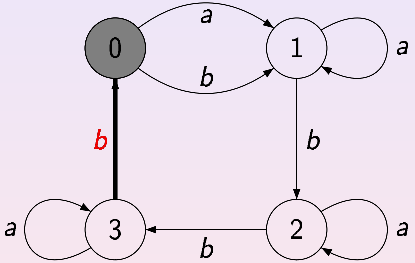
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



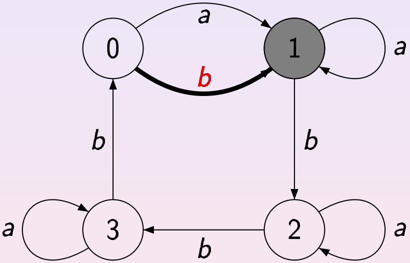
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



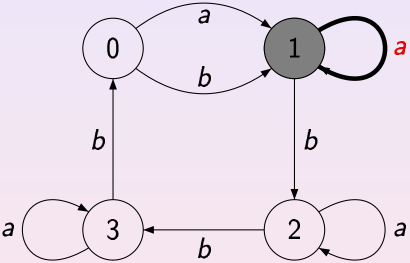
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



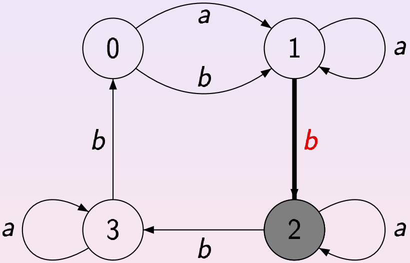
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



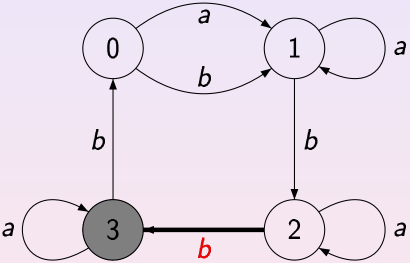
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



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

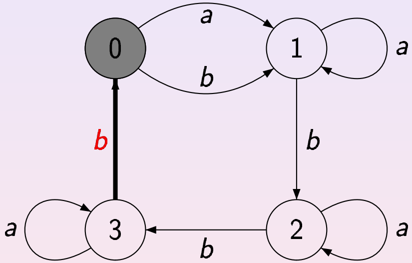
An Example



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

Vienna, November 24, 2010

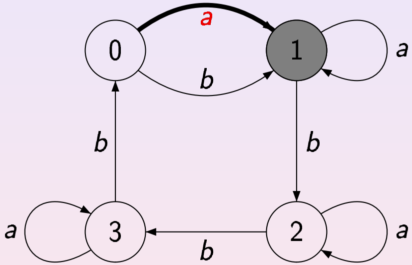
An Example



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

Vienna, November 24, 2010

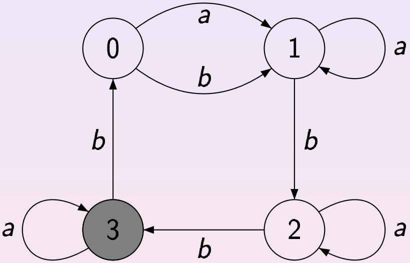
An Example



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

Vienna, November 24, 2010

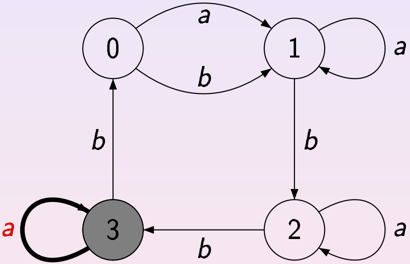
An Example



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

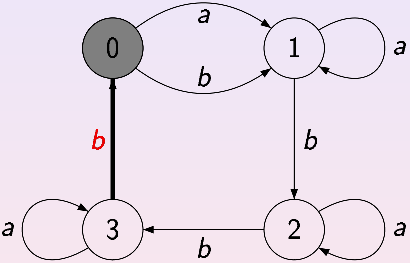
Vienna, November 24, 2010

An Example



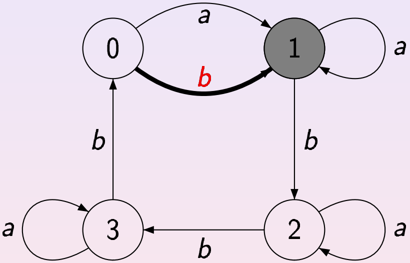
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



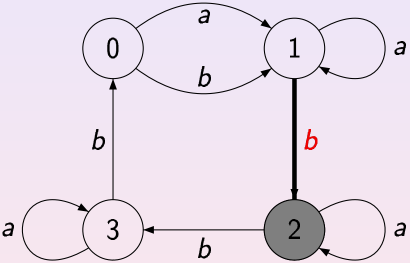
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



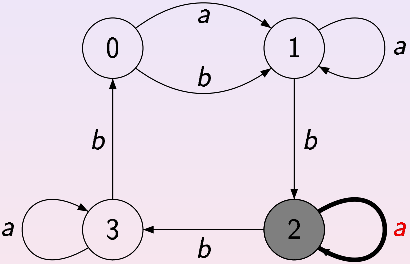
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



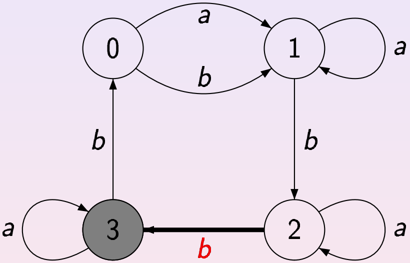
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



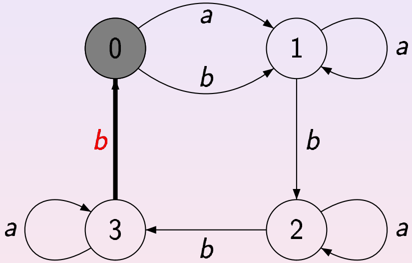
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



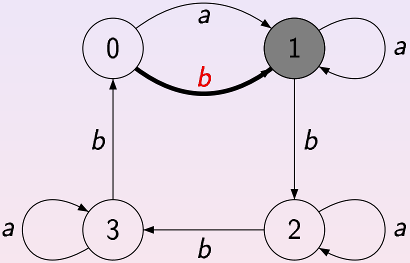
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

An Example



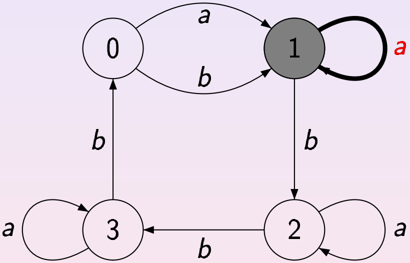
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

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.

Černý'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).

Černý'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).

Černý'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).

Ashby's Ghost Taming Automaton

The earliest synchronizing automaton that I was able to trace back in the literature appeared in Ross Ashby's 'An Introduction to Cybernetics' (1956), pp. 60–61.

Vienna, November 24, 2010

Ashby's Ghost Taming Automaton

The earliest synchronizing automaton that I was able to trace back in the literature appeared in Ross Ashby's 'An Introduction to Cybernetics' (1956), pp. 60–61.

'**4/15. Materiality**. *The reader may now like to test the methods of this chapter as an aid to solving the problem set by the following letter. It justifies the statement made in S.1/2 that cybernetics is not bound to the properties found in terrestrial matter, nor does it draw its laws from them. What is important in cybernetics is the extent to which the observed behaviour is regular and reproducible.*'

Ashby's Ghost Taming Automaton

The earliest synchronizing automaton that I was able to trace back in the literature appeared in Ross Ashby's 'An Introduction to Cybernetics' (1956), pp. 60–61.

The letter presents a puzzle about two ghostly noises, Singing and Laughter, in a haunted mansion. Each of the noises can be either on or off, and their behaviour depends on combinations of two possible actions, playing the organ or burning incense.

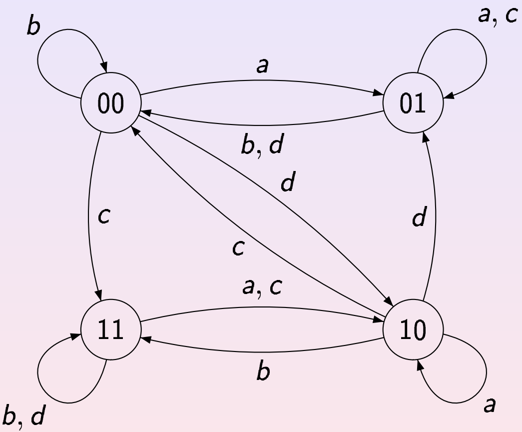
Vienna, November 24, 2010

Ashby's Ghost Taming Automaton

The earliest synchronizing automaton that I was able to trace back in the literature appeared in Ross Ashby's 'An Introduction to Cybernetics' (1956), pp. 60–61.

The letter presents a puzzle about two ghostly noises, Singing and Laughter, in a haunted mansion. Each of the noises can be either on or off, and their behaviour depends on combinations of two possible actions, playing the organ or burning incense.
Under a suitable encoding, this leads to an automaton with 4 states and 4 input letters shown in the next slide.

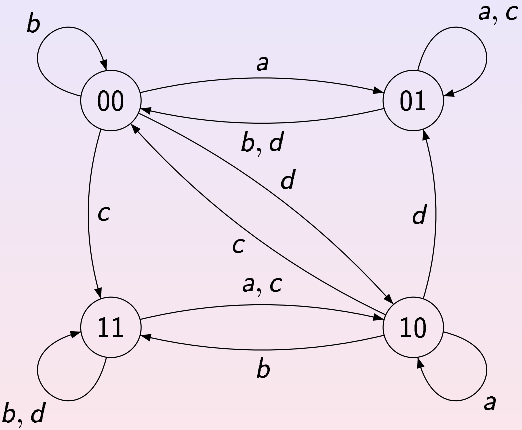
Ashby's Ghost Taming Automaton



It is easy to see that this is a synchronizing automaton and *acb* is its shortest reset word.

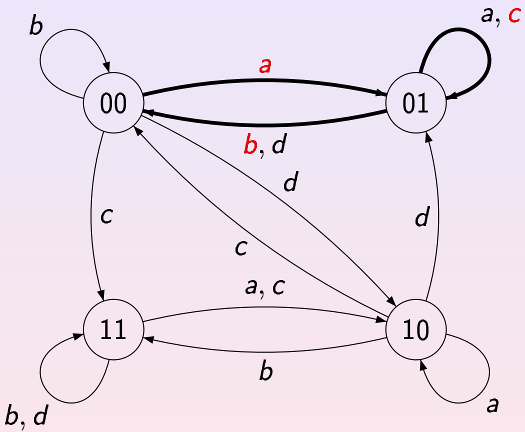
Vienna, November 24, 2010

Ashby's Ghost Taming Automaton



It is easy to see that this is a synchronizing automaton and *acb* is its shortest reset word.

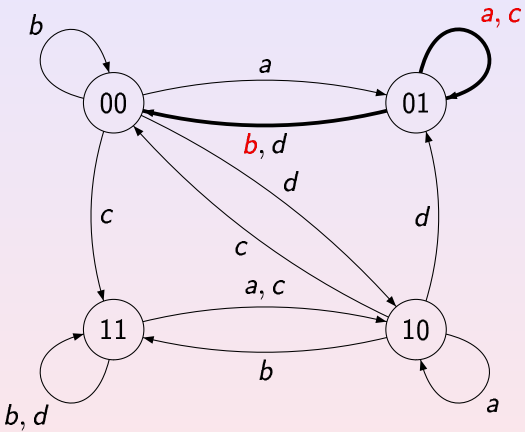
Ashby's Ghost Taming Automaton



It is easy to see that this is a synchronizing automaton and *acb* is its shortest reset word.

Vienna, November 24, 2010

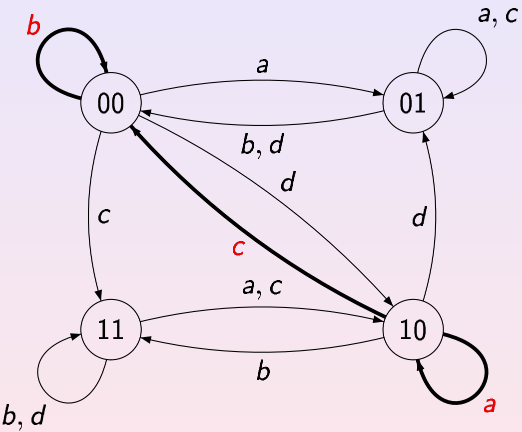
Ashby's Ghost Taming Automaton



It is easy to see that this is a synchronizing automaton and *acb* is its shortest reset word.

Vienna, November 24, 2010

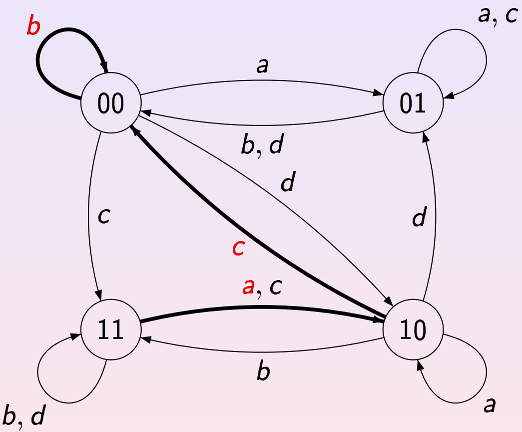
Ashby's Ghost Taming Automaton



It is easy to see that this is a synchronizing automaton and *acb* is its shortest reset word.

Vienna, November 24, 2010

Ashby's Ghost Taming Automaton



It is easy to see that this is a synchronizing automaton and *acb* is its shortest reset word.

A Frequently Discovered Notion

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 early 1960s: Moore, Ginsburg.
- 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 Frequently Discovered Notion

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 early 1960s: Moore, Ginsburg.
- 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 Frequently Discovered Notion

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 early 1960s: Moore, Ginsburg.
- 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 Frequently Discovered Notion

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 early 1960s: Moore, Ginsburg.

• 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.

Vienna, November 24, 2010

Crash Course in Coding Theory

A prefix code over a finite alphabet Σ is a set X of words in Σ^* 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 Σ 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.

Crash Course in Coding Theory

A prefix code over a finite alphabet Σ is a set X of words in Σ^* 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 Σ 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.

Crash Course in Coding Theory

A prefix code over a finite alphabet Σ is a set X of words in Σ^* 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 Σ 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.

Crash Course in Coding Theory

A prefix code over a finite alphabet Σ is a set X of words in Σ^* 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 Σ 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.

Vienna, November 24, 2010

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, $\dots$ 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.

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

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.

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.

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.

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

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.

Vienna, November 24, 2010

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.

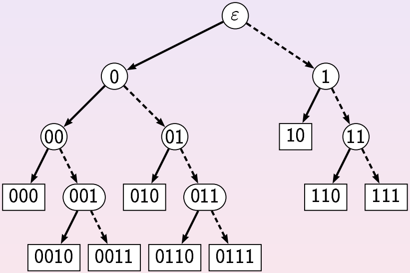
Codes vs Automata

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

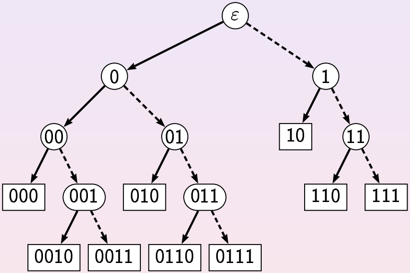
If X is a finite maximal prefix code, then its decoding can be implemented by a DFA.

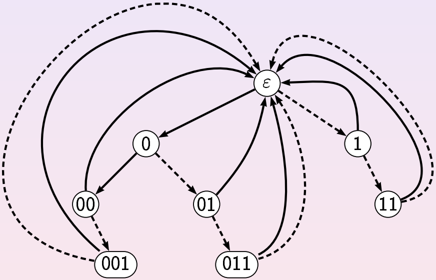


Synchronized codes precisely correspond to synchronizing automata!

Codes vs Automata

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

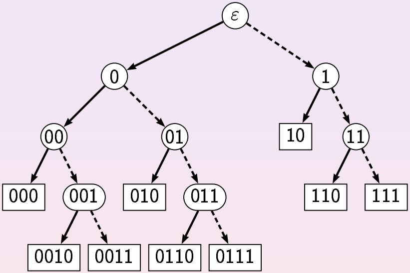


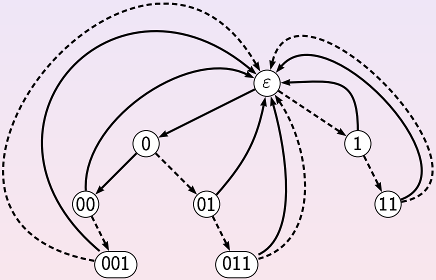


Synchronized codes precisely correspond to synchronizing automata!

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!

Other Motivations

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

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

And many further connections and applications . . .

Other Motivations

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

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

And many further connections and applications . . .

Other Motivations

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

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

And many further connections and applications . . .

Outline of the Talk

- From the viewpoint of applications 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.
- Interesting connections to symbolic dynamics led to the Road Coloring Problem but it will be mentioned only briefly.
- If time permits, we present some recent developments based on an interplay between slowly synchronizing automata and the Perron–Frobenius theory of non-negative matrices.

Vienna, November 24, 2010

Outline of the Talk

- From the viewpoint of applications 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.
- Interesting connections to symbolic dynamics led to the Road Coloring Problem but it will be mentioned only briefly.
- If time permits, we present some recent developments based on an interplay between slowly synchronizing automata and the Perron–Frobenius theory of non-negative matrices.

Vienna, November 24, 2010

Outline of the Talk

- From the viewpoint of applications 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.
- Interesting connections to symbolic dynamics led to the Road Coloring Problem but it will be mentioned only briefly.
- If time permits, we present some recent developments based on an interplay between slowly synchronizing automata and the Perron–Frobenius theory of non-negative matrices.

Vienna, November 24, 2010

Outline of the Talk

- From the viewpoint of applications 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.
- Interesting connections to symbolic dynamics led to the Road Coloring Problem but it will be mentioned only briefly.
- If time permits, we present some recent developments based on an interplay between slowly synchronizing automata and the Perron–Frobenius theory of non-negative matrices.

Vienna, November 24, 2010

Power Automaton

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\Delta(P, a) = P \, . \, a = \{\delta(p, a) \mid p \in P\}$

A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

Power Automaton

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\Delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

Power Automaton

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$:
- states are the non-empty subsets of Q,
- $\Delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

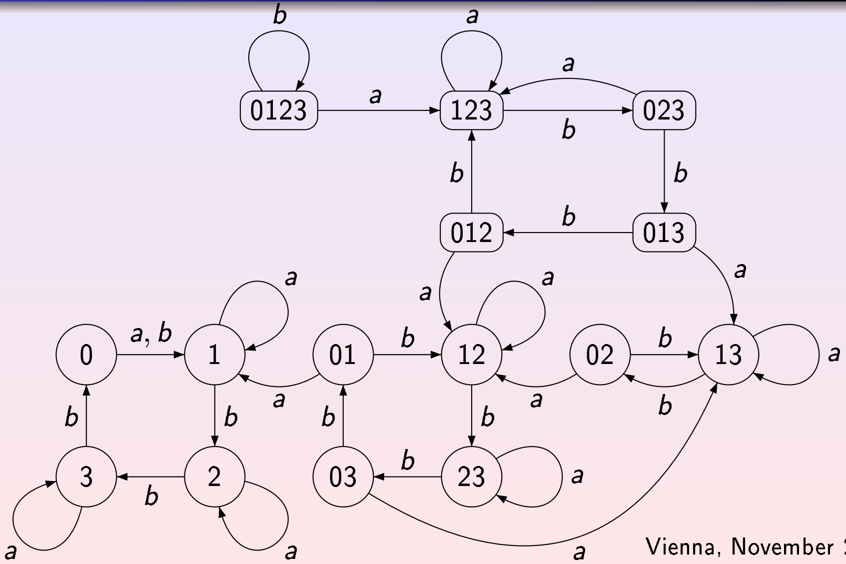
Power Automaton

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\Delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

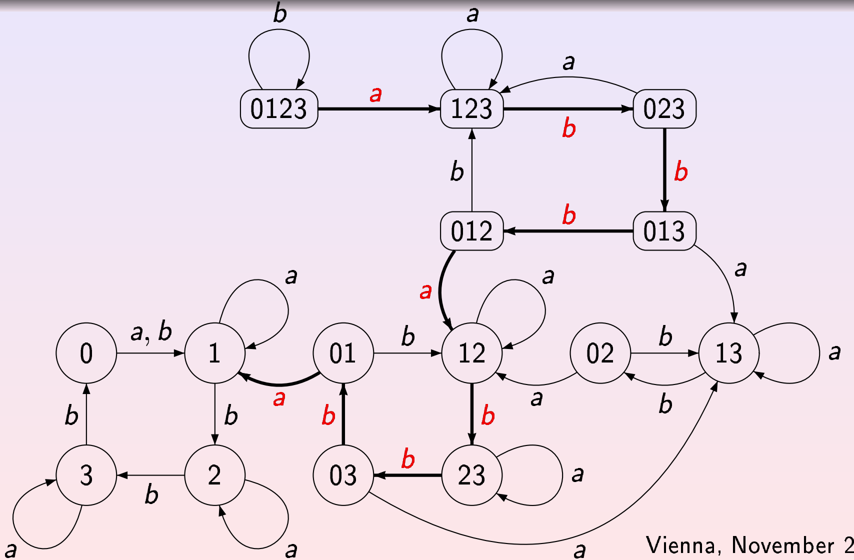
A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

An Example



Vienna, November 24, 2010

An Example



Vienna, November 24, 2010

A Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *is synchronizing iff for every* $q, q' \in Q$ *there exists a word* $w \in \Sigma^*$ *such that* $q \,.\, w = q' \,.\, w$.

A Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is synchronizing iff for every $q, q' \in Q$ there exists a word $w \in \Sigma^*$ such that $q \,.\, w = q' \,.\, w$.*

A Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is synchronizing iff for every $q, q' \in Q$ there exists a word $w \in \Sigma^*$ such that $q \,.\, w = q' \,.\, w$.*

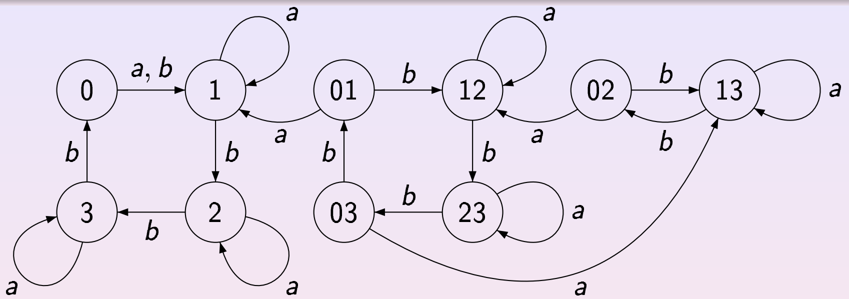
A Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *is synchronizing iff for every* $q, q' \in Q$ *there exists a word* $w \in \Sigma^*$ *such that* $q \,.\, w = q' \,.\, w$.

An Example

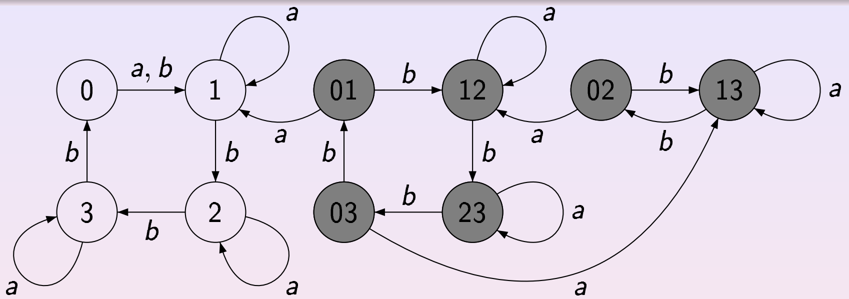


$a, \quad Q \,.\, a = \{1,2,3\}; \qquad a \cdot bba, \quad Q \,.\, abba = \{1,3\}$

$abba \cdot babbba, \quad Q \,.\, abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example

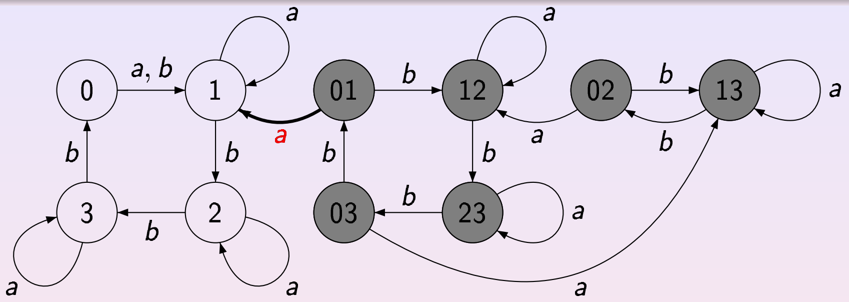


a, $Q\,.\,a = \{1, 2, 3\}$; $a \cdot bba$, $Q\,.\,abba = \{1, 3\}$

$abba \cdot babbba$, $Q\,.\,abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example

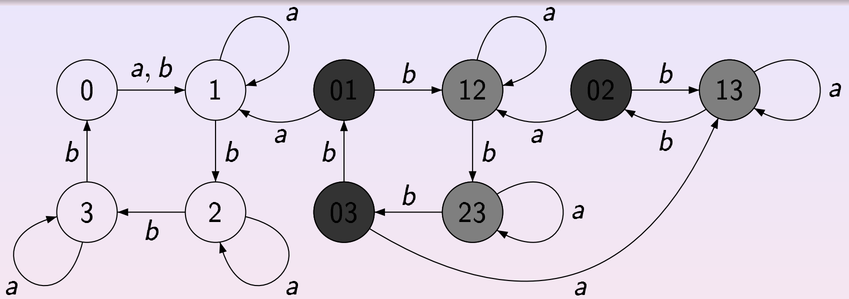


$a, \;\; Q\,.\,a = \{1,2,3\};$ $\quad a \cdot bba, \;\; Q\,.\,abba = \{1,3\}$

$abba \cdot babbba, \;\; Q\,.\,abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example

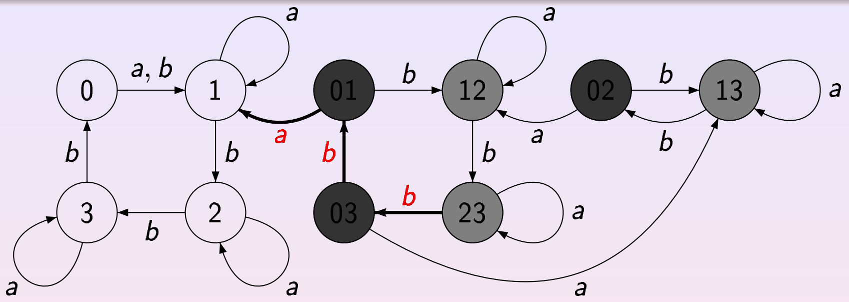


$a, \ Q \,.\, a = \{1,2,3\}; \quad a \cdot bba, \ Q \,.\, abba = \{1,3\}$

$abba \cdot babbba, \ Q \,.\, abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example

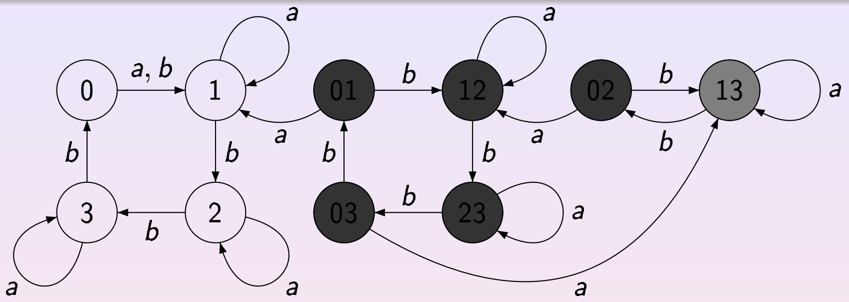


$$a, \; Q\,.\,a = \{1,2,3\}; \quad a \cdot bba, \; Q\,.\,abba = \{1,3\}$$

$$abba \cdot babbba, \; Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example

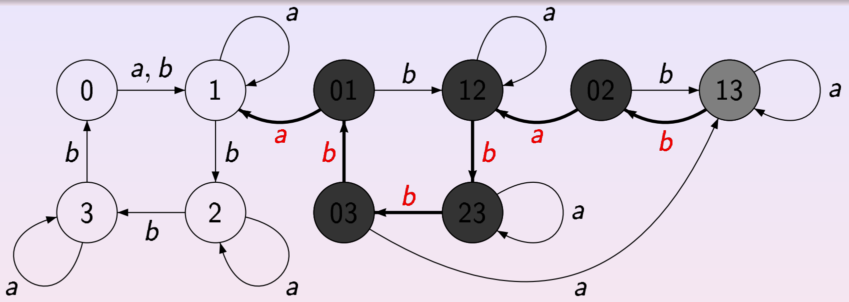


$$a, \ Q\,.\,a = \{1,2,3\}; \quad a \cdot bba, \ Q\,.\,abba = \{1,3\}$$

$$abba \cdot babbba, \ Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example

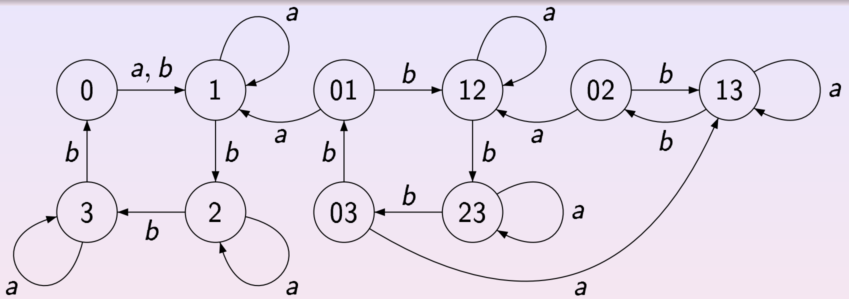


$$a, \;\; Q\,.\,a = \{1,2,3\}; \qquad a \cdot bba, \;\; Q\,.\,abba = \{1,3\}$$

$$abba \cdot babbba, \;\; Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

An Example



$$a, \ Q\,.\,a = \{1,2,3\}; \quad a \cdot bba, \ Q\,.\,abba = \{1,3\}$$

$$abba \cdot babbba, \ Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.

If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3-n}{3}$. Can we do better? What is the exact bound?

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.

If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.

If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3-n}{3}$. Can we do better? What is the exact bound?

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3-n}{3}$. Can we do better? What is the exact bound?

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

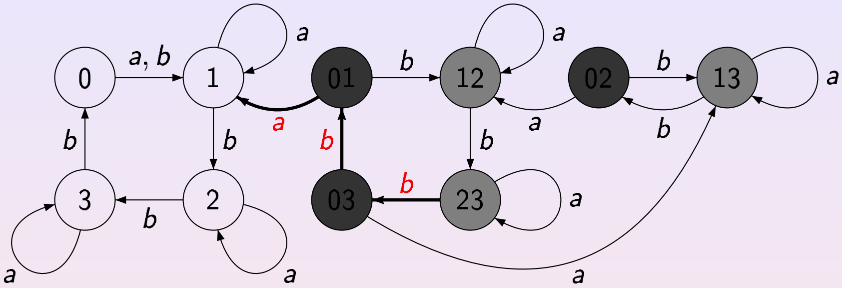
Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

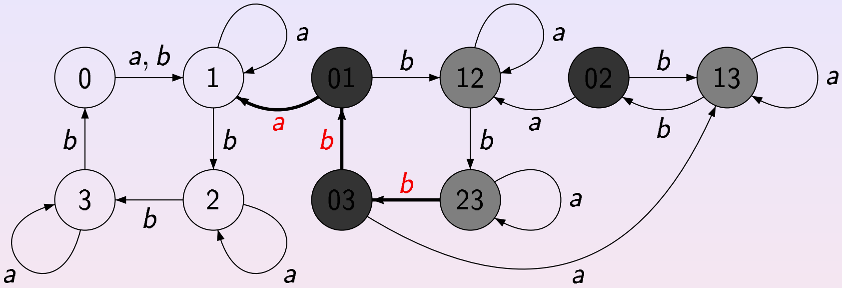
Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3-n}{3}$. Can we do better? What is the exact bound?

A Resource for Improvement



We see that the shortest path from a light-grey 2-subset to a singleton do not necessarily pass through all dark-grey 2-subsets. Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$. What is the minimum length of a word $v \in \Sigma^*$ such that $|P \,.\, v| < k$?

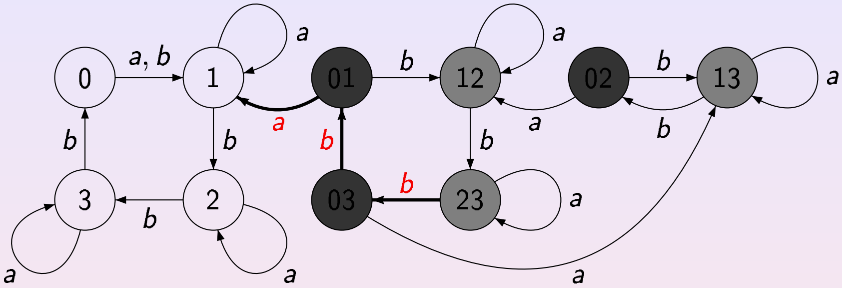
A Resource for Improvement



We see that the shortest path from a light-grey 2-subset to a singleton do not necessarily pass through all dark-grey 2-subsets.

Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$. What is the minimum length of a word $v \in \Sigma^*$ such that $|P\,.\,v| < k$?

A Resource for Improvement



We see that the shortest path from a light-grey 2-subset to a singleton do not necessarily pass through all dark-grey 2-subsets. Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$. What is the minimum length of a word $v \in \Sigma^*$ such that $|P\,.\,v| < k$?

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Vienna, November 24, 2010

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Results

In the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$— Jean-Eric Pin, 1983, based on a non-trivial combinatorial result by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

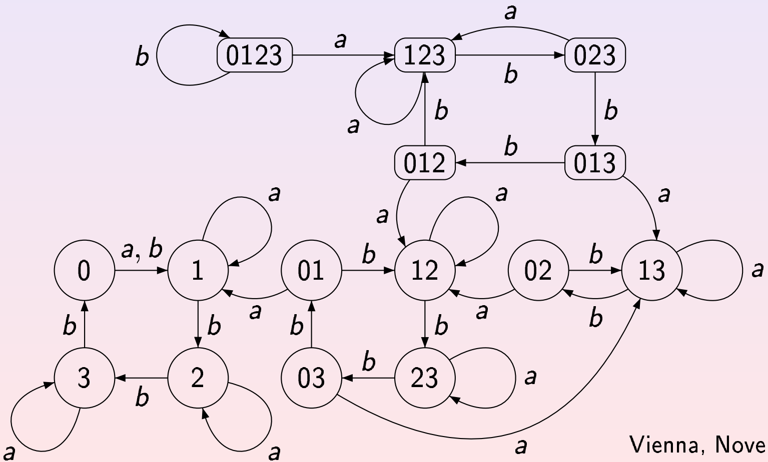
$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

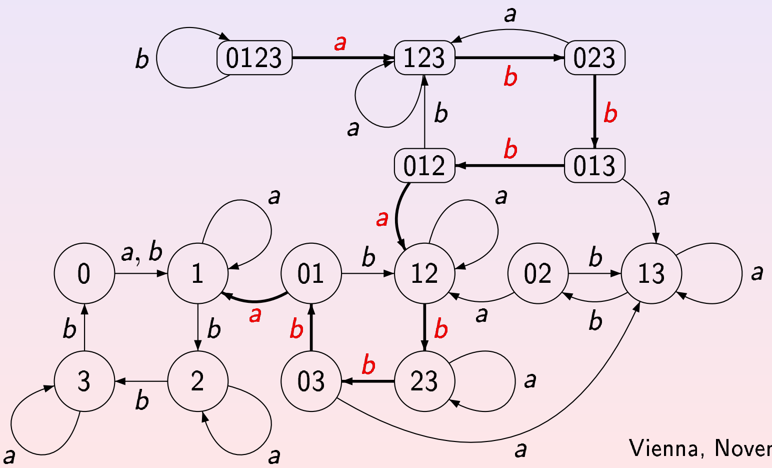
Example Revisited

We have already seen that the greedy algorithm fails to find a reset word of minimum length.



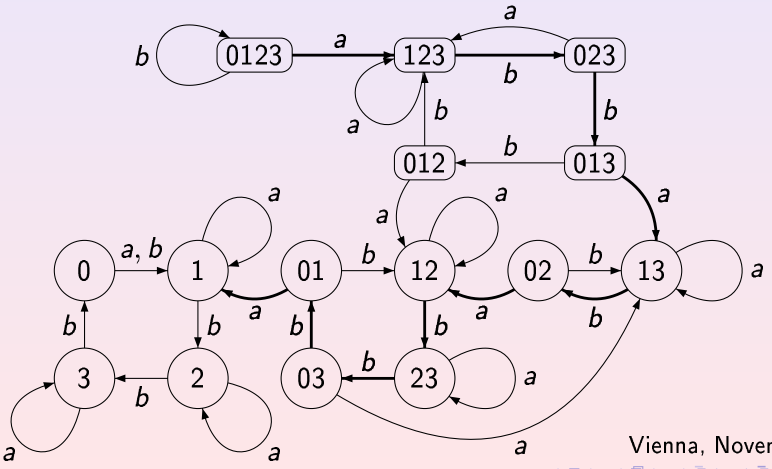
Example Revisited

We have already seen that the greedy algorithm fails to find a reset word of minimum length.



Example Revisited

We have already seen that the greedy algorithm fails to find a reset word of minimum length.



Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Find

Given a synchronizing automaton $\mathscr{A}$, we call the minimum length of its reset words the reset length of $\mathscr{A}$.

The gap between the reset length and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exist synchronizing automata with n states whose reset lengths are $\Omega(n^2)$ while the greedy algorithm produces reset words of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well. However, on slowly synchronizing automata a lavish algorithm turns out to perform much better.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the reset length.

Short Reset Words are Hard to Decide

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and a positive integer ℓ, is it true that $\mathscr{A}$ has a reset word of length ℓ?*

Clearly, SHORT-RESET-WORD belongs to NP: one can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that SHORT-RESET-WORD is NP-hard by a transparent reduction from SAT.

Short Reset Words are Hard to Decide

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *and a positive integer* ℓ*, is it true that* $\mathscr{A}$ *has a reset word of length* ℓ*?*

Clearly, SHORT-RESET-WORD belongs to NP: one can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that SHORT-RESET-WORD is NP-hard by a transparent reduction from SAT.

Short Reset Words are Hard to Decide

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *and a positive integer* ℓ*, is it true that* $\mathscr{A}$ *has a reset word of length* ℓ*?*

Clearly, SHORT-RESET-WORD belongs to NP: one can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that SHORT-RESET-WORD is NP-hard by a transparent reduction from SAT.

Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$.

The transitions are defined by:

Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$. The transitions are defined by:

Vienna, November 24, 2010

Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \le i \le m,\ 1 \le j \le n+1\}$. The transitions are defined by:

$$q_{i,j} \,.\, a = \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \le i \le m,\ 1 \le j \le n;$$

Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$. The transitions are defined by:

$$q_{i,j} \,.\, a = \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq n;$$

$$q_{i,j} \,.\, b = \begin{cases} z \text{ if } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq n;$$

Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$. The transitions are defined by:

$$q_{i,j} \,.\, a = \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq n;$$

$$q_{i,j} \,.\, b = \begin{cases} z \text{ if } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq n;$$

$$q_{i,n+1} \,.\, a = q_{i,n+1} \,.\, b = z \qquad \text{for } 1 \leq i \leq m;$$

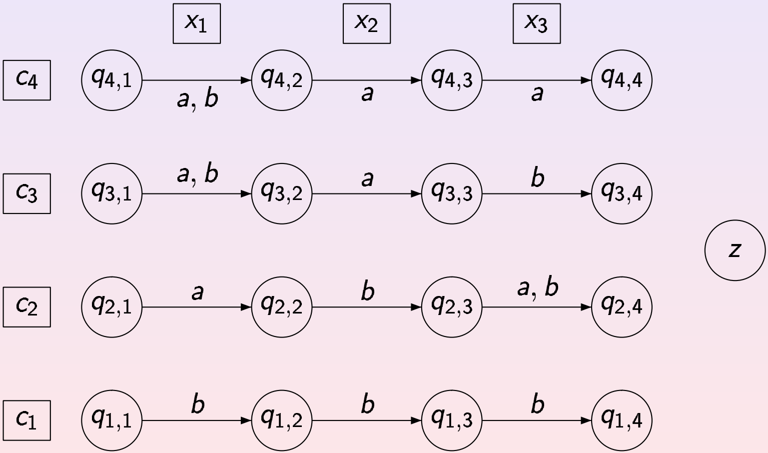
$$z \,.\, a = z \,.\, b = z.$$

Reduction from SAT

For $\psi = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$:

Reduction from SAT

For $\psi = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$:



Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable. In the above example the truth assignment $x_1 = x_2 = 0$, $x_3 = 1$ satisfies ψ and the word bba resets $\mathscr{A}(\psi)$.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable. In the above example the truth assignment $x_1 = x_2 = 0$, $x_3 = 1$ satisfies ψ and the word bba resets $\mathscr{A}(\psi)$.

If we change ψ to $\{x_1 \vee x_2, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_2 \vee \neg x_3\}$, it becomes unsatisfiable and $\mathscr{A}(\psi)$ is reset by no word of length 3. Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of $\textsc{Short-Reset-Word}$ to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

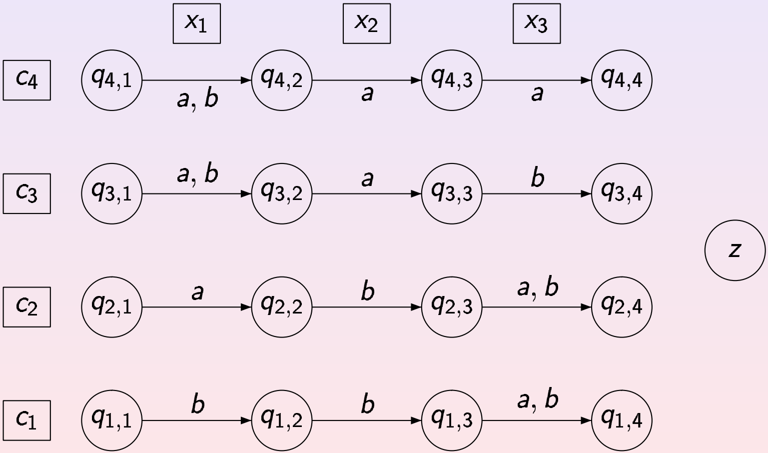
Reduction from SAT

For $\psi = \{x_1 \vee x_2, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_2 \vee \neg x_3\}$:

Vienna, November 24, 2010

Reduction from SAT

For $\psi = \{x_1 \vee x_2, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_2 \vee \neg x_3\}$:



Vienna, November 24, 2010

Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the reset length of $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has been shown to be complete for DP (Difference Polynomial-Time).

Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the reset length of $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has been shown to be complete for DP (Difference Polynomial-Time).

Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the reset length of $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP $=$ coNP.

Recently, SHORTEST-RESET-WORD has been shown to be complete for DP (Difference Polynomial-Time).

Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the reset length of $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has been shown to be complete for DP (Difference Polynomial-Time).

Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the reset length of $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has been shown to be complete for DP (Difference Polynomial-Time).

Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.

DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the reset length is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ — Jörg Olschewski & Michael Ummels, MFCS 2010.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.
DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the reset length is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ — Jörg Olschewski & Michael Ummels, MFCS 2010.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.
DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the reset length is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ — Jörg Olschewski & Michael Ummels, MFCS 2010.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.
DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the reset length is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ — Jörg Olschewski & Michael Ummels, MFCS 2010.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

Non-approximability

However, all known results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Recently, Mikhail Berlinkov (CSR 2010) has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length is less than $k\times$reset length.

Non-approximability

However, all known results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Recently, Mikhail Berlinkov (CSR 2010) has shown that under $\mathrm{NP} \neq \mathrm{P}$, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length is less than $k \times$reset length.

Vienna, November 24, 2010

The Černý Automata

Suppose a synchronizing automaton has n states. What is its reset length?

We know an upper bound: there always exists a reset word of length $\frac{n^3-n}{6}$. What about a lower bound?

In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \dots,$ of synchronizing automata over 2 letters.

The states of $\mathscr{C}_n$ are the residues modulo n, and the input letters a and b act as follows:

$$\delta(0, a) = 1, \ \delta(m, a) = m \text{ for } 0 < m < n, \delta(m, b) = m+1 \pmod{n}.$$

The automaton used as our 'standard' example is $\mathscr{C}_4$.

The Černý Automata

Suppose a synchronizing automaton has n states. What is its reset length?

We know an upper bound: there always exists a reset word of length $\frac{n^3-n}{6}$. What about a lower bound?

In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \dots,$ of synchronizing automata over 2 letters.

The states of $\mathscr{C}_n$ are the residues modulo n, and the input letters a and b act as follows:

$$\delta(0,a) = 1,\ \delta(m,a) = m \text{ for } 0 < m < n,\ \delta(m,b) = m+1 \pmod{n}.$$

The automaton used as our 'standard' example is $\mathscr{C}_4$.

The Černý Automata

Suppose a synchronizing automaton has n states. What is its reset length?

We know an upper bound: there always exists a reset word of length $\frac{n^3-n}{6}$. What about a lower bound?

In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \ldots,$ of synchronizing automata over 2 letters.

The states of $\mathscr{C}_n$ are the residues modulo n, and the input letters a and b act as follows:

$$\delta(0, a) = 1,\ \delta(m, a) = m \text{ for } 0 < m < n, \delta(m, b) = m{+}1 \pmod{n}.$$

The automaton used as our 'standard' example is $\mathscr{C}_4$.

The Černý Automata

Suppose a synchronizing automaton has n states. What is its reset length?

We know an upper bound: there always exists a reset word of length $\frac{n^3-n}{6}$. What about a lower bound?

In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \dots,$ of synchronizing automata over 2 letters.

The states of $\mathscr{C}_n$ are the residues modulo n, and the input letters a and b act as follows:

$$\delta(0, a) = 1,\ \delta(m, a) = m \text{ for } 0 < m < n, \delta(m, b) = m+1 \pmod{n}.$$

The automaton used as our 'standard' example is $\mathscr{C}_4$.

The Černý Automata

Suppose a synchronizing automaton has n states. What is its reset length?

We know an upper bound: there always exists a reset word of length $\frac{n^3-n}{6}$. What about a lower bound?

In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \dots,$ of synchronizing automata over 2 letters.

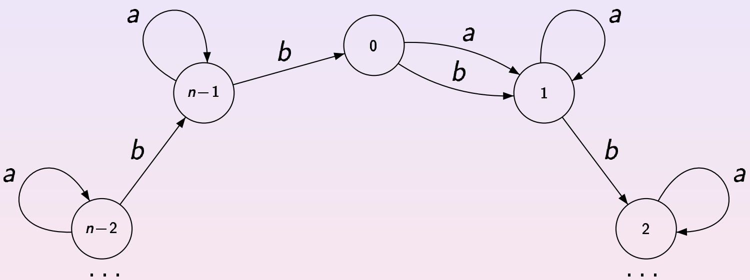
The states of $\mathscr{C}_n$ are the residues modulo n, and the input letters a and b act as follows:

$$\delta(0,a) = 1,\ \delta(m,a) = m \text{ for } 0 < m < n, \delta(m,b) = m+1 \pmod{n}.$$

The automaton used as our 'standard' example is $\mathscr{C}_4$.

The Černý Automata

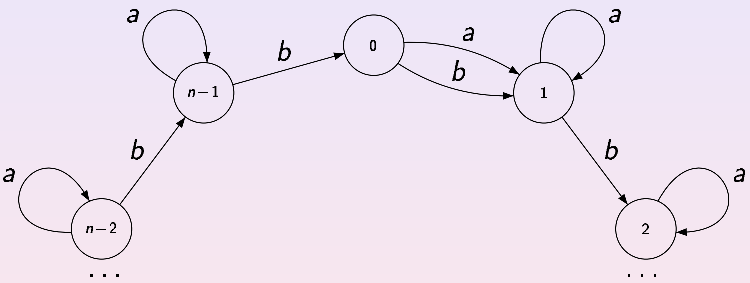
Here is a generic automaton from the Černý series:



Černý has proved that the shortest reset word for $\mathscr{C}_n$ is $(ab^{n-1})^{n-2}a$ of length $(n-1)^2$. As other results from Černý's paper of 1964, this nice series of automata has been rediscovered many times.

The Černý Automata

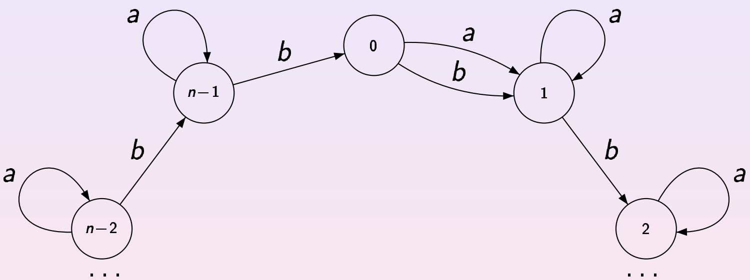
Here is a generic automaton from the Černý series:



Černý has proved that the shortest reset word for $\mathscr{C}_n$ is $(ab^{n-1})^{n-2}a$ of length $(n-1)^2$. As other results from Černý's paper of 1964, this nice series of automata has been rediscovered many times.

The Černý Automata

Here is a generic automaton from the Černý series:



Černý has proved that the shortest reset word for $\mathscr{C}_n$ is $(ab^{n-1})^{n-2}a$ of length $(n-1)^2$. As other results from Černý's paper of 1964, this nice series of automata has been rediscovered many times.

The Černý Function

Define the *Černý function* $C(n)$ as the maximum reset length for synchronizing automata with n states. The above property of the series $\{\mathscr{C}_n\}$, $n = 2, 3, \ldots$, yields the inequality $C(n) \geq (n-1)^2$.

The Černý Conjecture is the claim that in fact the equality $C(n) = (n-1)^2$ holds true. This simply looking conjecture is arguably the most longstanding open problem in the combinatorial theory of finite automata. Everything we know about the conjecture in general can be summarized in one line:

$$(n-1)^2 \leq C(n) \leq \frac{n^3 - n}{6}.$$

The Černý Function

Define the *Černý function* $C(n)$ as the maximum reset length for synchronizing automata with n states. The above property of the series $\{\mathscr{C}_n\}$, $n = 2, 3, \dots,$ yields the inequality $C(n) \geq (n-1)^2$. The Černý Conjecture is the claim that in fact the equality $C(n) = (n-1)^2$ holds true. This simply looking conjecture is arguably the most longstanding open problem in the combinatorial theory of finite automata. Everything we know about the conjecture in general can be summarized in one line:

$$(n-1)^2 \leq C(n) \leq \frac{n^3 - n}{6}.$$

Vienna, November 24, 2010

The Černý Function

Define the *Černý function* $C(n)$ as the maximum reset length for synchronizing automata with n states. The above property of the series $\{\mathscr{C}_n\}$, $n = 2, 3, \dots$, yields the inequality $C(n) \geq (n-1)^2$. The Černý Conjecture is the claim that in fact the equality $C(n) = (n-1)^2$ holds true. This simply looking conjecture is arguably the most longstanding open problem in the combinatorial theory of finite automata. Everything we know about the conjecture in general can be summarized in one line:

$$(n-1)^2 \leq C(n) \leq \frac{n^3 - n}{6}.$$

Vienna, November 24, 2010

The Černý Function

Define the *Černý function* $C(n)$ as the maximum reset length for synchronizing automata with n states. The above property of the series $\{\mathscr{C}_n\}$, $n = 2, 3, \dots,$ yields the inequality $C(n) \geq (n-1)^2$. The Černý Conjecture is the claim that in fact the equality $C(n) = (n-1)^2$ holds true. This simply looking conjecture is arguably the most longstanding open problem in the combinatorial theory of finite automata. Everything we know about the conjecture in general can be summarized in one line:

$$(n-1)^2 \leq C(n) \leq \frac{n^3 - n}{6}.$$

A Discussion

Why is the problem so surprisingly difficult?

- non-locality: prefixes of optimal solutions need not be optimal (that's why the greedy algorithm fails);
- combinatorics of finite sets is encoded in the problem.

Yet another reason: "slowly" synchronizing automata turn out to be extremely rare. The only known infinite series of n-state synchronizing automata with reset length $(n-1)^2$ is the Černý series $\mathscr{C}_n$, $n = 2, 3, \ldots$, with a few (actually, 8) sporadic examples for $n \leq 6$.

Vienna, November 24, 2010

A Discussion

Why is the problem so surprisingly difficult?

- non-locality: prefixes of optimal solutions need not be optimal (that's why the greedy algorithm fails);
- combinatorics of finite sets is encoded in the problem.

Yet another reason: "slowly" synchronizing automata turn out to be extremely rare. The only known infinite series of n-state synchronizing automata with reset length $(n-1)^2$ is the Černý series $\mathscr{C}_n$, $n = 2, 3, \dots$, with a few (actually, 8) sporadic examples for $n \leq 6$.

A Discussion

Why is the problem so surprisingly difficult?

- non-locality: prefixes of optimal solutions need not be optimal (that's why the greedy algorithm fails);
- combinatorics of finite sets is encoded in the problem.

Yet another reason: "slowly" synchronizing automata turn out to be extremely rare. The only known infinite series of n-state synchronizing automata with reset length $(n-1)^2$ is the Černý series $\mathscr{C}_n$, $n = 2, 3, \ldots$, with a few (actually, 8) sporadic examples for $n \leq 6$.

Vienna, November 24, 2010

A Discussion

Why is the problem so surprisingly difficult?

- non-locality: prefixes of optimal solutions need not be optimal (that's why the greedy algorithm fails);
- combinatorics of finite sets is encoded in the problem.

Yet another reason: "slowly" synchronizing automata turn out to be extremely rare. The only known infinite series of n-state synchronizing automata with reset length $(n-1)^2$ is the Černý series $\mathscr{C}_n$, $n = 2, 3, \ldots$, with a few (actually, 8) sporadic examples for $n \leq 6$.

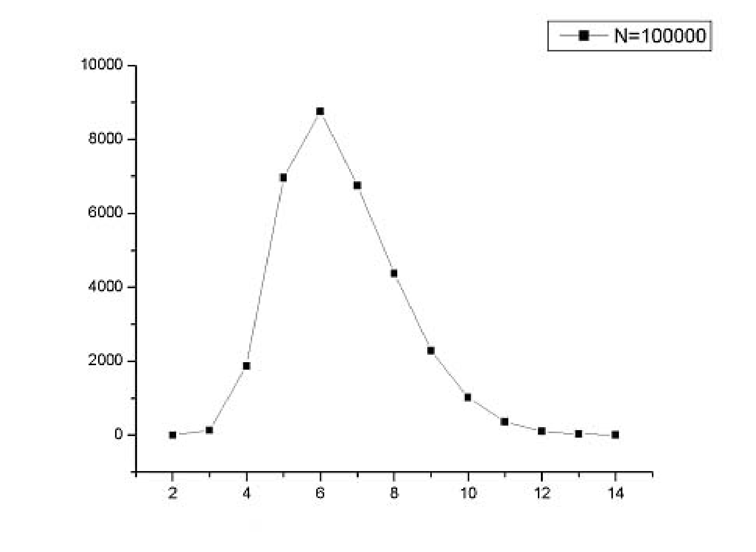
A Discussion

Why is the problem so surprisingly difficult?

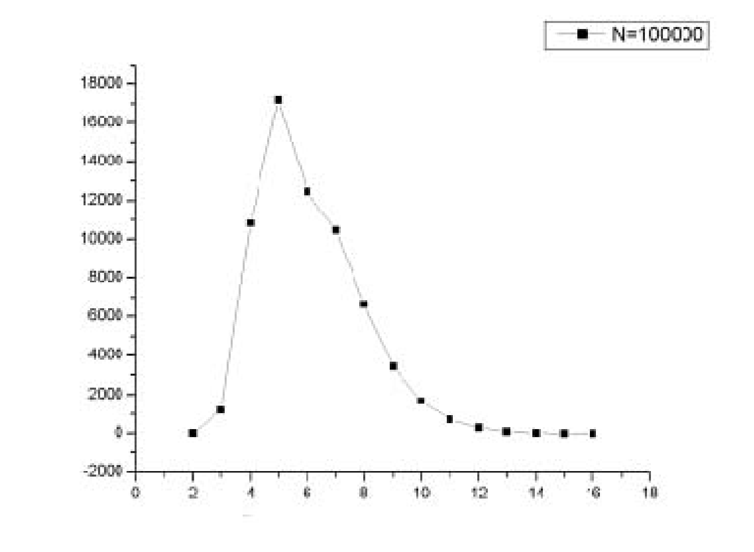
- non-locality: prefixes of optimal solutions need not be optimal (that's why the greedy algorithm fails);
- combinatorics of finite sets is encoded in the problem.

Yet another reason: "slowly" synchronizing automata turn out to be extremely rare. The only known infinite series of n-state synchronizing automata with reset length $(n-1)^2$ is the Černý series $\mathscr{C}_n$, $n = 2, 3, \dots$, with a few (actually, 8) sporadic examples for $n \leq 6$.

20-State Experiment



30-State Experiment



24, 2010

Random Automata

A (partial) explanation of these experimental observations: if Q is an n-set (with n large enough), then, on average, any product of $2n$ randomly chosen transformations of Q is a constant map (Peter Higgins, The range order of a product of i transformations from a finite full transformation semigroup, Semigroup Forum, 37 (1988) 31–36). In automata-theoretic terms, this fact means that a randomly chosen DFA with n states and a sufficiently large input alphabet tends to be synchronizing and is reset by any word of length $\geq 2n$.

Thus, "slowly" synchronizing automata cannot be discovered via random sampling.

Random Automata

A (partial) explanation of these experimental observations: if Q is an n-set (with n large enough), then, on average, any product of $2n$ randomly chosen transformations of Q is a constant map (Peter Higgins, The range order of a product of i transformations from a finite full transformation semigroup, Semigroup Forum, 37 (1988) 31–36). In automata-theoretic terms, this fact means that a randomly chosen DFA with n states and a sufficiently large input alphabet tends to be synchronizing and is reset by any word of length $\geq 2n$.

Thus, "slowly" synchronizing automata cannot be discovered via random sampling.

Random Automata

A (partial) explanation of these experimental observations: if Q is an n-set (with n large enough), then, on average, any product of $2n$ randomly chosen transformations of Q is a constant map (Peter Higgins, The range order of a product of i transformations from a finite full transformation semigroup, Semigroup Forum, 37 (1988) 31–36). In automata-theoretic terms, this fact means that a randomly chosen DFA with n states and a sufficiently large input alphabet tends to be synchronizing and is reset by any word of length $\geq 2n$.

Thus, "slowly" synchronizing automata cannot be discovered via random sampling.

Random Automata

A (partial) explanation of these experimental observations: if Q is an n-set (with n large enough), then, on average, any product of $2n$ randomly chosen transformations of Q is a constant map (Peter Higgins, The range order of a product of i transformations from a finite full transformation semigroup, Semigroup Forum, 37 (1988) 31–36). In automata-theoretic terms, this fact means that a randomly chosen DFA with n states and a sufficiently large input alphabet tends to be synchronizing and is reset by any word of length $\geq 2n$.

Thus, "slowly" synchronizing automata cannot be discovered via random sampling.

A Recent Discovery

Vladimir Gusev has performed a massive series of experiments searching exhaustively through automata with a modest number of states in order to find new examples of "slowly" synchronizing automata. He has used an approach due to Marco Almeida, Nelma Moreira and Rogério Reis, Enumeration and generation with a string automata representation, Theoret. Comput. Sci., 387 (2007) 93–102.

Vienna, November 24, 2010

A Recent Discovery

Vladimir Gusev has performed a massive series of experiments searching exhaustively through automata with a modest number of states in order to find new examples of “slowly” synchronizing automata. He has used an approach due to Marco Almeida, Nelma Moreira and Rogério Reis, Enumeration and generation with a string automata representation, Theoret. Comput. Sci., 387 (2007) 93–102.

The Second Gap

The next tables present the distribution of non-isomorphic synchronizing automata with 8 and 9 states and 2 letters with respect to their reset lengths.

The Second Gap

The next tables present the distribution of non-isomorphic synchronizing automata with 8 and 9 states and 2 letters with respect to their reset lengths.

8 states:

Reset length	49	48	47	46	45	44	43	42	41	40
# of automata	1	0	0	0	0	1	1	3	1	5

Vienna, November 24, 2010

The Second Gap

The next tables present the distribution of non-isomorphic synchronizing automata with 8 and 9 states and 2 letters with respect to their reset lengths.

8 states:

Reset length	49	48	47	46	45	44	43	42	41	40
# of automata	1	0	0	0	0	1	1	3	1	5

The Second Gap

The next tables present the distribution of non-isomorphic synchronizing automata with 8 and 9 states and 2 letters with respect to their reset lengths.

8 states:

Reset length	49	48	47	46	45	44	43	42	41	40
# of automata	1	0	0	0	0	1	1	3	1	5

9 states:

Reset length	64	63	62	61	60	59	58	57	56	55
# of automata	1	0	0	0	0	0	1	2	3	0

Reset length	54	53	52	51
# of automata	0	0	4	4

The Second Gap

The next tables present the distribution of non-isomorphic synchronizing automata with 8 and 9 states and 2 letters with respect to their reset lengths.

8 states:

Reset length	49	48	47	46	45	44	43	42	41	40
# of automata	1	0	0	0	0	1	1	3	1	5

9 states:

Reset length	64	63	62	61	60	59	58	57	56	55
# of automata	1	0	0	0	0	0	1	2	3	0

Reset length	54	53	52	51
# of automata	0	0	4	4

An Advantage of Being Old

Thus, the pattern is:

$(n-1)^2$ the first gap the "island" the second gap

The second gap first appears at 9 states and grows rather regularly with the number of states. The size of the island depends only on the parity of the number of states.

The very same pattern appears in the distribution of exponents of non-negative matrices.

An Advantage of Being Old

Thus, the pattern is:

$(n-1)^2$ the first gap the "island" the second gap

The second gap first appears at 9 states and grows rather regularly with the number of states. The size of the island depends only on the parity of the number of states.

The very same pattern appears in the distribution of exponents of non-negative matrices.

An Advantage of Being Old

Thus, the pattern is:

$(n-1)^2$ the first gap the "island" the second gap

The second gap first appears at 9 states and grows rather regularly with the number of states. The size of the island depends only on the parity of the number of states.
The very same pattern appears in the distribution of exponents of non-negative matrices.

Exponents of Non-negative Matrices

A non-negative matrix A is said to be primitive if some power A^k is positive. The minimum k with this property is called the exponent of A, denoted $\exp A$.

Helmut Wielandt proved in 1950 that for any primitive $n \times n$-matrix A, one has $\exp A \leq n^2 - 2n + 2 = (n-1)^2 + 1$, and this bound is tight. Possible exponents of $n \times n$-matrices were intensively studied in the 1960s, and it was discovered that two extreme values are each attained by a unique matrix, then there is a gap followed by an island followed by another gap. The sizes of the gaps and of the island perfectly match the sizes of the gaps and of the islands in possible lengths of shortest reset words for synchronizing automata with n states – basically one has the same picture shifted by 1. Clearly, this cannot be a mere coincidence.

Vienna, November 24, 2010

Exponents of Non-negative Matrices

A non-negative matrix A is said to be primitive if some power A^k is positive. The minimum k with this property is called the exponent of A, denoted $\exp A$.

Helmut Wielandt proved in 1950 that for any primitive $n \times n$-matrix A, one has $\exp A \leq n^2 - 2n + 2 = (n-1)^2 + 1$, and this bound is tight. Possible exponents of $n \times n$-matrices were intensively studied in the 1960s, and it was discovered that two extreme values are each attained by a unique matrix, then there is a gap followed by an island followed by another gap. The sizes of the gaps and of the island perfectly match the sizes of the gaps and of the islands in possible lengths of shortest reset words for synchronizing automata with n states – basically one has the same picture shifted by 1. Clearly, this cannot be a mere coincidence.

Exponents of Non-negative Matrices

A non-negative matrix A is said to be primitive if some power A^k is positive. The minimum k with this property is called the exponent of A, denoted $\exp A$.

Helmut Wielandt proved in 1950 that for any primitive $n \times n$-matrix A, one has $\exp A \leq n^2 - 2n + 2 = (n-1)^2 + 1$, and this bound is tight. Possible exponents of $n \times n$-matrices were intensively studied in the 1960s, and it was discovered that two extreme values are each attained by a unique matrix, then there is a gap followed by an island followed by another gap. The sizes of the gaps and of the island perfectly match the sizes of the gaps and of the islands in possible lengths of shortest reset words for synchronizing automata with n states – basically one has the same picture shifted by 1. Clearly, this cannot be a mere coincidence.

Exponents of Non-negative Matrices

A non-negative matrix A is said to be primitive if some power A^k is positive. The minimum k with this property is called the exponent of A, denoted $\exp A$.

Helmut Wielandt proved in 1950 that for any primitive $n \times n$-matrix A, one has $\exp A \leq n^2 - 2n + 2 = (n-1)^2 + 1$, and this bound is tight. Possible exponents of $n \times n$-matrices were intensively studied in the 1960s, and it was discovered that two extreme values are each attained by a unique matrix, then there is a gap followed by an island followed by another gap. The sizes of the gaps and of the island perfectly match the sizes of the gaps and of the islands in possible lengths of shortest reset words for synchronizing automata with n states – basically one has the same picture shifted by 1. Clearly, this cannot be a mere coincidence.

Exponents of Non-negative Matrices

A non-negative matrix A is said to be primitive if some power A^k is positive. The minimum k with this property is called the exponent of A, denoted $\exp A$.

Helmut Wielandt proved in 1950 that for any primitive $n \times n$-matrix A, one has $\exp A \leq n^2 - 2n + 2 = (n-1)^2 + 1$, and this bound is tight. Possible exponents of $n \times n$-matrices were intensively studied in the 1960s, and it was discovered that two extreme values are each attained by a unique matrix, then there is a gap followed by an island followed by another gap. The sizes of the gaps and of the island perfectly match the sizes of the gaps and of the islands in possible lengths of shortest reset words for synchronizing automata with n states – basically one has the same picture shifted by 1. Clearly, this cannot be a mere coincidence.

Digraphs and Matrices

A directed graph (digraph) is a pair $D = \langle V, E \rangle$.

- V set of vertices
- $E \subseteq V \times V$ set of edges

This definition allows loops but excludes multiple edges.

The matrix of a digraph $D = \langle V, E \rangle$ is just the matrix of the edge relation, that is, a $V \times V$-matrix whose entry in the row v and the column v' is 1 if $(v, v') \in E$ and 0 otherwise.

Vienna, November 24, 2010

Digraphs and Matrices

A directed graph (digraph) is a pair $D = \langle V, E \rangle$.

- V set of vertices
- $E \subseteq V \times V$ set of edges

This definition allows loops but excludes multiple edges.

The matrix of a digraph $D = \langle V, E \rangle$ is just the matrix of the edge relation, that is, a $V \times V$-matrix whose entry in the row v and the column v' is 1 if $(v, v') \in E$ and 0 otherwise.

Digraphs and Matrices

A directed graph (digraph) is a pair $D = \langle V, E\rangle$.

- V set of vertices
- $E \subseteq V \times V$ set of edges

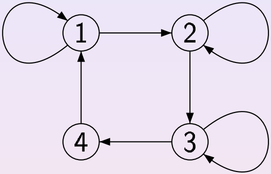
This definition allows loops but excludes multiple edges.

The matrix of a digraph $D = \langle V, E\rangle$ is just the matrix of the edge relation, that is, a $V \times V$-matrix whose entry in the row v and the column v' is 1 if $(v, v') \in E$ and 0 otherwise.

Vienna, November 24, 2010

Digraphs and Matrices

For instance, the matrix of the digraph



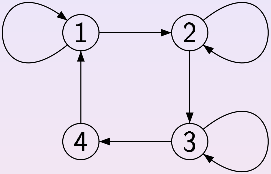
(with respect to the chosen numbering of its vertices) is $\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$.

Conversely, given an $n \times n$-matrix $P = (p_{ij})$ with non-negative real entries, we assign to it a digraph $D(P)$ on the set $\{1, 2, \ldots, n\}$ as follows: (i, j) is an edge of $D(P)$ if and only if $p_{ij} > 0$.

This 'two-way' correspondence allows us to reformulate in terms of digraphs several important notions and results which originated in the classical Perron–Frobenius theory of non-negative matrices.

Digraphs and Matrices

For instance, the matrix of the digraph



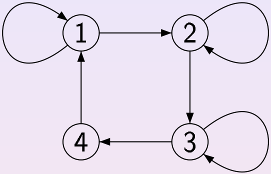
(with respect to the chosen numbering of its vertices) is $\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$.

Conversely, given an $n \times n$-matrix $P = (p_{ij})$ with non-negative real entries, we assign to it a digraph $D(P)$ on the set $\{1, 2, \ldots, n\}$ as follows: (i, j) is an edge of $D(P)$ if and only if $p_{ij} > 0$.

This 'two-way' correspondence allows us to reformulate in terms of digraphs several important notions and results which originated in the classical Perron–Frobenius theory of non-negative matrices.

Digraphs and Matrices

For instance, the matrix of the digraph



(with respect to the chosen numbering of its vertices) is $\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$.

Conversely, given an $n \times n$-matrix $P = (p_{ij})$ with non-negative real entries, we assign to it a digraph $D(P)$ on the set $\{1, 2, \dots, n\}$ as follows: (i, j) is an edge of $D(P)$ if and only if $p_{ij} > 0$.

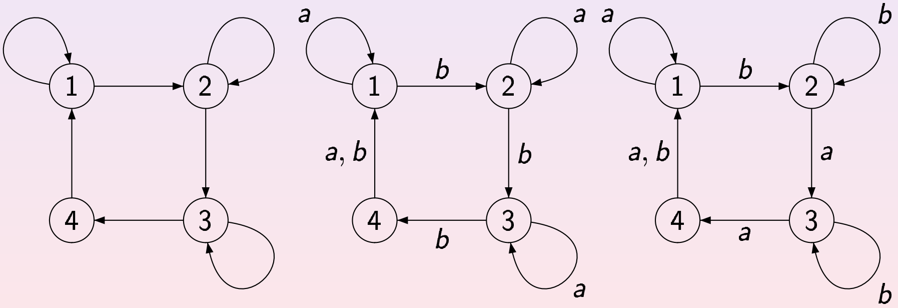
This 'two-way' correspondence allows us to reformulate in terms of digraphs several important notions and results which originated in the classical Perron–Frobenius theory of non-negative matrices.

Digraphs and Colorings

By a coloring of a digraph we mean assigning labels from an alphabet Σ to edges such that the digraph labelled this way becomes a DFA.

Digraphs and Colorings

By a coloring of a digraph we mean assigning labels from an alphabet Σ to edges such that the digraph labelled this way becomes a DFA.



Primitive Digraphs

A digraph D is primitive if D is strongly connected and the greatest common divisor of the lengths of all cycles in D is equal to 1.

Adler, Goodwyn, Weiss (Equivalence of topological Markov shifts, Israel J. Math., 27 (1977) 49–63):
Underlying digraphs of strongly connected synchronizing automata are primitive.

The Road Coloring Conjecture: Every primitive digraph admits a synchronizing coloring.

This was confirmed by Avraham Trahtman in 2007. The solution is published in: The Road Coloring Problem, Israel J. Math. 172 (2009) 51–60.

Vienna, November 24, 2010

Primitive Digraphs

A digraph D is primitive if D is strongly connected and the greatest common divisor of the lengths of all cycles in D is equal to 1.

Adler, Goodwyn, Weiss (Equivalence of topological Markov shifts, Israel J. Math., 27 (1977) 49–63):
Underlying digraphs of strongly connected synchronizing automata are primitive.

The Road Coloring Conjecture: Every primitive digraph admits a synchronizing coloring.

This was confirmed by Avraham Trahtman in 2007. The solution is published in: The Road Coloring Problem, Israel J. Math. 172 (2009) 51–60.

Primitive Digraphs

A digraph D is primitive if D is strongly connected and the greatest common divisor of the lengths of all cycles in D is equal to 1.

Adler, Goodwyn, Weiss (Equivalence of topological Markov shifts, Israel J. Math., 27 (1977) 49–63):
Underlying digraphs of strongly connected synchronizing automata are primitive.

The Road Coloring Conjecture: Every primitive digraph admits a synchronizing coloring.

This was confirmed by Avraham Trahtman in 2007. The solution is published in: The Road Coloring Problem, Israel J. Math. 172 (2009) 51–60.

Primitive Digraphs

A digraph D is primitive if D is strongly connected and the greatest common divisor of the lengths of all cycles in D is equal to 1.

Adler, Goodwyn, Weiss (Equivalence of topological Markov shifts, Israel J. Math., 27 (1977) 49–63):
Underlying digraphs of strongly connected synchronizing automata are primitive.

The Road Coloring Conjecture: Every primitive digraph admits a synchronizing coloring.

This was confirmed by Avraham Trahtman in 2007. The solution is published in: The Road Coloring Problem, Israel J. Math. 172 (2009) 51–60.

Exponents

A digraph D is primitive iff there exists t such that for each pair of vertices there exists a path between them of length exactly t.

(This is equivalent to saying that the t-th power of the matrix of D is positive.) The least t with this property is called the exponent of the digraph D and is denoted by $\gamma(D)$.

1950, Wielandt: The exponent of every primitive digraph on n vertices is not greater than $(n-1)^2+1$ and this bound is tight.

1964, Dulmage–Mendelsohn: There is exactly one primitive digraph on n vertices with exponent $(n-1)^2+1$ and exactly one primitive digraph on n vertices with exponent $(n-1)^2$.

If $n > 4$ is even, then there is no primitive digraph D on n vertices such that $n^2-4n+6 < \gamma(D) < (n-1)^2$.

If $n > 3$ is odd, then there is no primitive digraph D on n vertices such that $n^2-3n+4 < \gamma(D) < (n-1)^2$, or $n^2-4n+6 < \gamma(D) < n^2-3n+2$.

Exponents

A digraph D is primitive iff there exists t such that for each pair of vertices there exists a path between them of length exactly t. (This is equivalent to saying that the t-th power of the matrix of D is positive.) The least t with this property is called the exponent of the digraph D and is denoted by $\gamma(D)$.

1950, Wielandt: The exponent of every primitive digraph on n vertices is not greater than $(n-1)^2+1$ and this bound is tight.

1964, Dulmage–Mendelsohn: There is exactly one primitive digraph on n vertices with exponent $(n-1)^2+1$ and exactly one primitive digraph on n vertices with exponent $(n-1)^2$.

If $n > 4$ is even, then there is no primitive digraph D on n vertices such that $n^2-4n+6 < \gamma(D) < (n-1)^2$.

If $n > 3$ is odd, then there is no primitive digraph D on n vertices such that $n^2-3n+4 < \gamma(D) < (n-1)^2$, or $n^2-4n+6 < \gamma(D) < n^2-3n+2$.

Exponents

A digraph D is primitive iff there exists t such that for each pair of vertices there exists a path between them of length exactly t. (This is equivalent to saying that the t-th power of the matrix of D is positive.) The least t with this property is called the exponent of the digraph D and is denoted by $\gamma(D)$.

1950, Wielandt: The exponent of every primitive digraph on n vertices is not greater than $(n-1)^2+1$ and this bound is tight.

1964, Dulmage–Mendelsohn: There is exactly one primitive digraph on n vertices with exponent $(n-1)^2+1$ and exactly one primitive digraph on n vertices with exponent $(n-1)^2$.

If $n > 4$ is even, then there is no primitive digraph D on n vertices such that $n^2-4n+6 < \gamma(D) < (n-1)^2$.

If $n > 3$ is odd, then there is no primitive digraph D on n vertices such that $n^2-3n+4 < \gamma(D) < (n-1)^2$, or $n^2-4n+6 < \gamma(D) < n^2-3n+2$.

Exponents

A digraph D is primitive iff there exists t such that for each pair of vertices there exists a path between them of length exactly t. (This is equivalent to saying that the t-th power of the matrix of D is positive.) The least t with this property is called the exponent of the digraph D and is denoted by $\gamma(D)$.

1950, Wielandt: The exponent of every primitive digraph on n vertices is not greater than $(n-1)^2+1$ and this bound is tight.

1964, Dulmage–Mendelsohn: There is exactly one primitive digraph on n vertices with exponent $(n-1)^2+1$ and exactly one primitive digraph on n vertices with exponent $(n-1)^2$.

If $n>4$ is even, then there is no primitive digraph D on n vertices such that $n^2-4n+6<\gamma(D)<(n-1)^2$.

If $n>3$ is odd, then there is no primitive digraph D on n vertices such that $n^2-3n+4<\gamma(D)<(n-1)^2$, or $n^2-4n+6<\gamma(D)<n^2-3n+2$.

Exponents

A digraph D is primitive iff there exists t such that for each pair of vertices there exists a path between them of length exactly t. (This is equivalent to saying that the t-th power of the matrix of D is positive.) The least t with this property is called the exponent of the digraph D and is denoted by $\gamma(D)$.

1950, Wielandt: The exponent of every primitive digraph on n vertices is not greater than $(n-1)^2+1$ and this bound is tight.

1964, Dulmage–Mendelsohn: There is exactly one primitive digraph on n vertices with exponent $(n-1)^2+1$ and exactly one primitive digraph on n vertices with exponent $(n-1)^2$.

If $n>4$ is even, then there is no primitive digraph D on n vertices such that $n^2-4n+6<\gamma(D)<(n-1)^2$.

If $n>3$ is odd, then there is no primitive digraph D on n vertices such that $n^2-3n+4<\gamma(D)<(n-1)^2$, or $n^2-4n+6<\gamma(D)<n^2-3n+2$.

Exponents

A digraph D is primitive iff there exists t such that for each pair of vertices there exists a path between them of length exactly t. (This is equivalent to saying that the t-th power of the matrix of D is positive.) The least t with this property is called the exponent of the digraph D and is denoted by $\gamma(D)$.

1950, Wielandt: The exponent of every primitive digraph on n vertices is not greater than $(n-1)^2+1$ and this bound is tight.

1964, Dulmage–Mendelsohn: There is exactly one primitive digraph on n vertices with exponent $(n-1)^2+1$ and exactly one primitive digraph on n vertices with exponent $(n-1)^2$.
If $n>4$ is even, then there is no primitive digraph D on n vertices such that $n^2-4n+6<\gamma(D)<(n-1)^2$.
If $n>3$ is odd, then there is no primitive digraph D on n vertices such that $n^2-3n+4<\gamma(D)<(n-1)^2$, or $n^2-4n+6<\gamma(D)<n^2-3n+2$.

Exponents vs Reset Lengths

Exponents of primitive digraphs with 9 vertices vs reset lengths of 2-letter strongly connected synchronizing automata with 9 states

N	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51
# of primitive digraphs with exponent N	1	1	0	0	0	0	0	1	1	2	0	0	0	0	4
# of 2-letter synchronizing automata with reset length N	0	1	0	0	0	0	0	1	2	3	0	0	0	4	4

Vienna, November 24, 2010

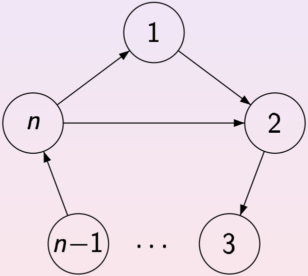
The Wielandt Automaton

The Wielandt automaton $\mathscr{W}_n$ is a (unique) coloring of the Wielandt digraph W_n with $\gamma(W_n) = (n-1)^2 + 1$.

Vienna, November 24, 2010

The Wielandt Automaton

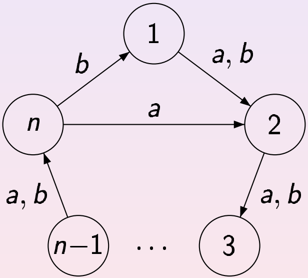
The Wielandt automaton $\mathscr{W}_n$ is a (unique) coloring of the Wielandt digraph W_n with $\gamma(W_n) = (n-1)^2 + 1$.



Vienna, November 24, 2010

The Wielandt Automaton

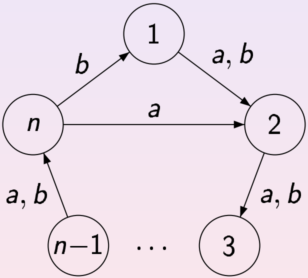
The Wielandt automaton $\mathscr{W}_n$ is a (unique) coloring of the Wielandt digraph W_n with $\gamma(W_n) = (n-1)^2 + 1$.



Vienna, November 24, 2010

The Wielandt Automaton

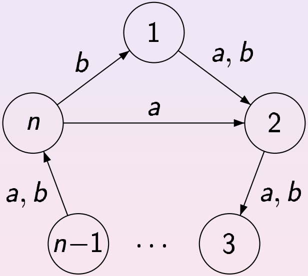
The Wielandt automaton $\mathscr{W}_n$ is a (unique) coloring of the Wielandt digraph W_n with $\gamma(W_n) = (n-1)^2 + 1$.



It is easy to show that the reset length of $\mathscr{W}_n$ is $n^2 - 3n + 3$.

The Wielandt Automaton

The Wielandt automaton $\mathscr{W}_n$ is a (unique) coloring of the Wielandt digraph W_n with $\gamma(W_n) = (n-1)^2 + 1$.



It is easy to show that the reset length of $\mathscr{W}_n$ is $n^2 - 3n + 3$.

In a similar way, every digraph with large exponent generates slowly synchronizing automata.

A Hybrid Conjecture

The Wielandt digraph admits an essentially unique coloring.
In general, a digraph can be colored in many ways.

The Hybrid Problem: What is the minimum reset length for synchronizing colorings of a primitive digraph with n vertices?

The Wielandt digraph provides a lower bound $n^2 - 3n + 3$.

We conjecture that this bound is tight:
The Hybrid Conjecture: Every primitive digraph with n vertices admits a synchronizing coloring with reset length at most $n^2 - 3n + 3$.

A Hybrid Conjecture

The Wielandt digraph admits an essentially unique coloring. In general, a digraph can be colored in many ways.

The Hybrid Problem: What is the minimum reset length for synchronizing colorings of a primitive digraph with n vertices?

The Wielandt digraph provides a lower bound $n^2 - 3n + 3$.

We conjecture that this bound is tight:
The Hybrid Conjecture: Every primitive digraph with n vertices admits a synchronizing coloring with reset length at most $n^2 - 3n + 3$.

A Hybrid Conjecture

The Wielandt digraph admits an essentially unique coloring. In general, a digraph can be colored in many ways.

The Hybrid Problem: What is the minimum reset length for synchronizing colorings of a primitive digraph with n vertices?

The Wielandt digraph provides a lower bound $n^2 - 3n + 3$.

We conjecture that this bound is tight:
The Hybrid Conjecture: Every primitive digraph with n vertices admits a synchronizing coloring with reset length at most $n^2 - 3n + 3$.

A Hybrid Conjecture

The Wielandt digraph admits an essentially unique coloring. In general, a digraph can be colored in many ways.

The Hybrid Problem: What is the minimum reset length for synchronizing colorings of a primitive digraph with n vertices?

The Wielandt digraph provides a lower bound $n^2 - 3n + 3$.

We conjecture that this bound is tight:
The Hybrid Conjecture: Every primitive digraph with n vertices admits a synchronizing coloring with reset length at most $n^2 - 3n + 3$.

A Hybrid Conjecture

The Wielandt digraph admits an essentially unique coloring. In general, a digraph can be colored in many ways.

The Hybrid Problem: What is the minimum reset length for synchronizing colorings of a primitive digraph with n vertices?

The Wielandt digraph provides a lower bound $n^2 - 3n + 3$.

We conjecture that this bound is tight:

The Hybrid Conjecture: Every primitive digraph with n vertices admits a synchronizing coloring with reset length at most $n^2 - 3n + 3$.

A Hybrid Conjecture

The Wielandt digraph admits an essentially unique coloring. In general, a digraph can be colored in many ways.

The Hybrid Problem: What is the minimum reset length for synchronizing colorings of a primitive digraph with n vertices?

The Wielandt digraph provides a lower bound $n^2 - 3n + 3$.

We conjecture that this bound is tight:
The Hybrid Conjecture: Every primitive digraph with n vertices admits a synchronizing coloring with reset length at most $n^2 - 3n + 3$.

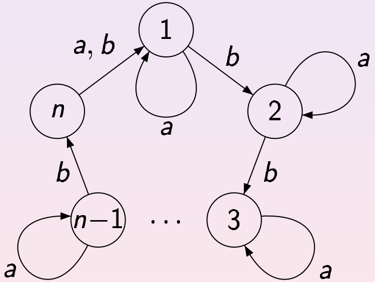
The Černý Automata Revisited

It turns out that the Černý automaton $\mathscr{C}_n$ is closely related to Wielandt automaton $\mathscr{W}_n$.

Vienna, November 24, 2010

The Černý Automata Revisited

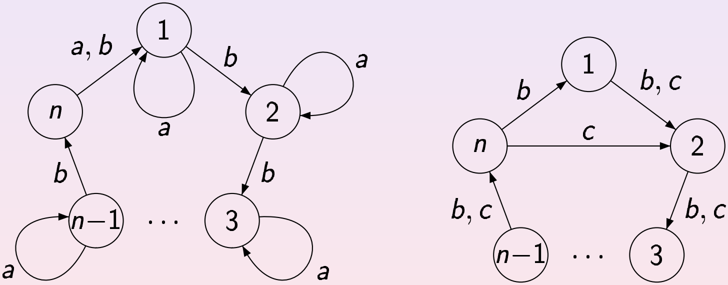
It turns out that the Černý automaton $\mathscr{C}_n$ is closely related to Wielandt automaton $\mathscr{W}_n$.



Vienna, November 24, 2010

The Černý Automata Revisited

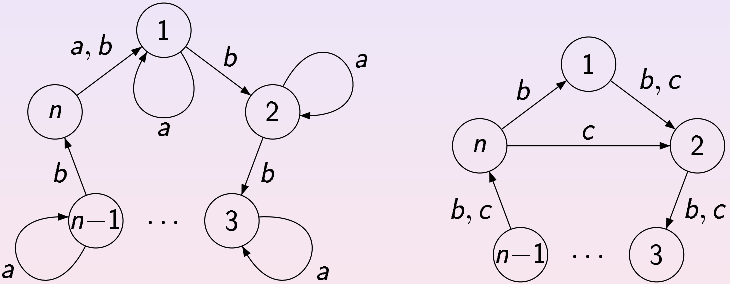
It turns out that the Černý automaton $\mathscr{C}_n$ is closely related to Wielandt automaton $\mathscr{W}_n$.



$\mathscr{C}_n$ becomes $\mathscr{W}_n$ under the action of b and $c = ab$.

The Černý Automata Revisited

It turns out that the Černý automaton $\mathscr{C}_n$ is closely related to Wielandt automaton $\mathscr{W}_n$.



$\mathscr{C}_n$ becomes $\mathscr{W}_n$ under the action of b and $c = ab$.

It is easy to see that every shortest reset word of $\mathscr{C}_n$ transforms into a reset word of $\mathscr{W}_n$, and this allows one to easily recover the Černý bound $(n-1)^2$.

Summary

The 5 years of the AutoMathA programme brought drastic changes to the theory of synchronizing automata. We may expect further progress in the nearest future.

Vienna, November 24, 2010