Completely Reachable Automata

Mikhail Volkov
(joint with Evgenija Bondar)

Ural Federal University, Ekaterinburg, Russia

DCFS'16, July 6, 2016

Where I am from?

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs).

$\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs). $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs). $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs). $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs). $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs). $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

Definitions, Terminology, and Disclaimer

We consider complete deterministic finite automata (DFAs). $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ 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 introduce a DFA as $\langle Q, \Sigma \rangle$ and write $q \,.\, w$ for $\delta(q, w)$ and $P \,.\, w$ for $\{\delta(q, w) \mid q \in P\}$.

Given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, a non-empty subset $P \subseteq Q$ is reachable in $\mathscr{A}$ if $P = Q \,.\, w$ for some word $w \in \Sigma^*$. A DFA is completely reachable if every non-empty set of its states is reachable.

The present talk is in fact a work-in-progress report—we do have some results but many, many natural questions still remain open. Some of these open questions will be mentioned here but many more can be found in the proceedings paper or in its expanded version at arXiv.org (arXiv:1607.00554).

First Motivation: Synchronizing Automata

A $\mathscr{A} = \langle Q, \Sigma \rangle$ 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 it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$.

In short, $|Q \,.\, w| = 1$; that is a singleton is reachable in $\mathscr{A}$.

Hence, a completely reachable automaton is synchronizing.

Any w with $|Q \,.\, w| = 1$ is a reset word for $\mathscr{A}$. The minimum length of reset words for $\mathscr{A}$ is called the reset threshold of $\mathscr{A}$.

First Motivation: Synchronizing Automata

A $\mathscr{A} = \langle Q, \Sigma \rangle$ 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 it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$. In short, $|Q \,.\, w| = 1$; that is a singleton is reachable in $\mathscr{A}$.

Hence, a completely reachable automaton is synchronizing.

Any w with $|Q \,.\, w| = 1$ is a reset word for $\mathscr{A}$. The minimum length of reset words for $\mathscr{A}$ is called the reset threshold of $\mathscr{A}$.

First Motivation: Synchronizing Automata

A $\mathscr{A} = \langle Q, \Sigma \rangle$ 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 it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$. In short, $|Q \,.\, w| = 1$; that is a singleton is reachable in $\mathscr{A}$.

Hence, a completely reachable automaton is synchronizing.

Any w with $|Q \,.\, w| = 1$ is a reset word for $\mathscr{A}$. The minimum length of reset words for $\mathscr{A}$ is called the reset threshold of $\mathscr{A}$.

First Motivation: Synchronizing Automata

A $\mathscr{A} = \langle Q, \Sigma \rangle$ 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 it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$. In short, $|Q \,.\, w| = 1$; that is a singleton is reachable in $\mathscr{A}$.

Hence, a completely reachable automaton is synchronizing.

Any w with $|Q \,.\, w| = 1$ is a reset word for $\mathscr{A}$. The minimum length of reset words for $\mathscr{A}$ is called the reset threshold of $\mathscr{A}$.

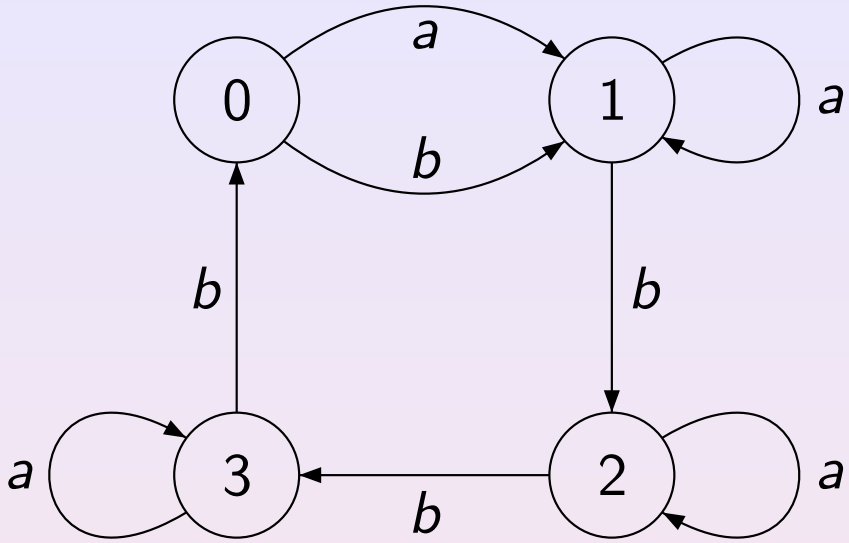
First Motivation: Synchronizing Automata

A $\mathscr{A} = \langle Q, \Sigma \rangle$ 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 it started: $q \,.\, w = q' \,.\, w$ for all $q, q' \in Q$. In short, $|Q \,.\, w| = 1$; that is a singleton is reachable in $\mathscr{A}$.

Hence, a completely reachable automaton is synchronizing.

Any w with $|Q \,.\, w| = 1$ is a reset word for $\mathscr{A}$. The minimum length of reset words for $\mathscr{A}$ is called the reset threshold of $\mathscr{A}$.

An Example

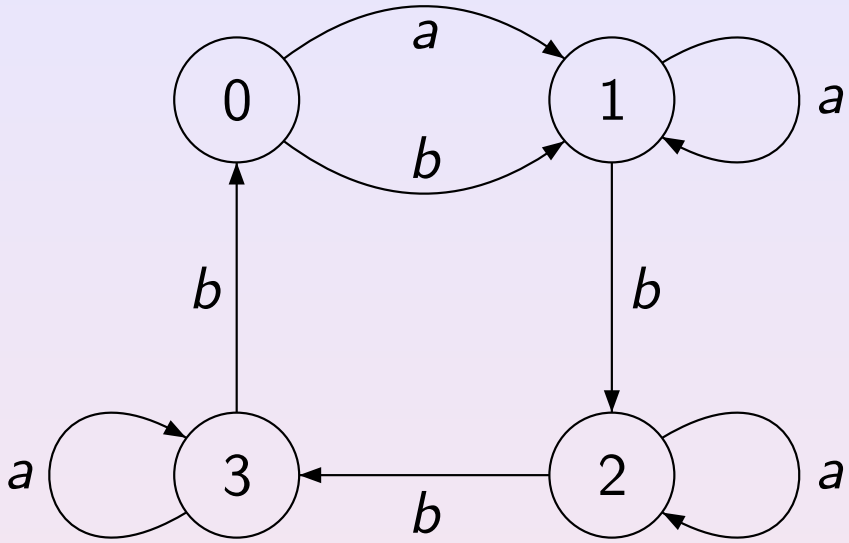


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

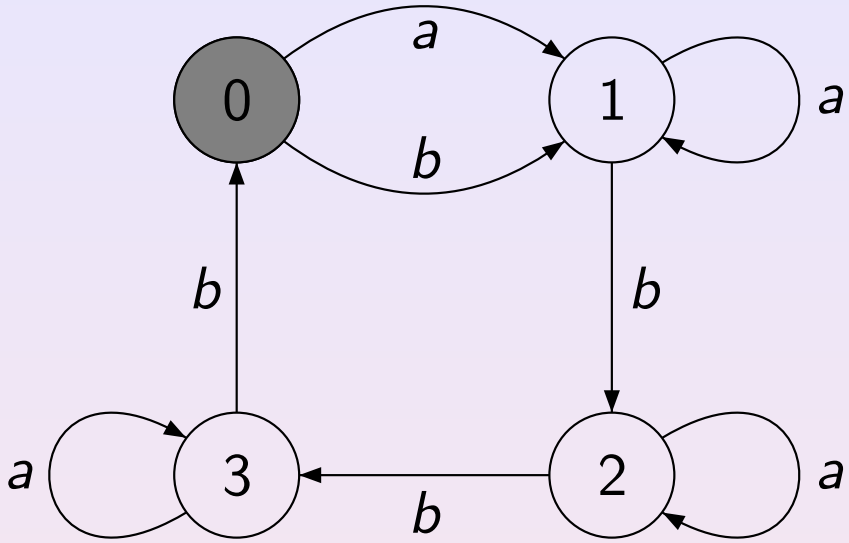


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

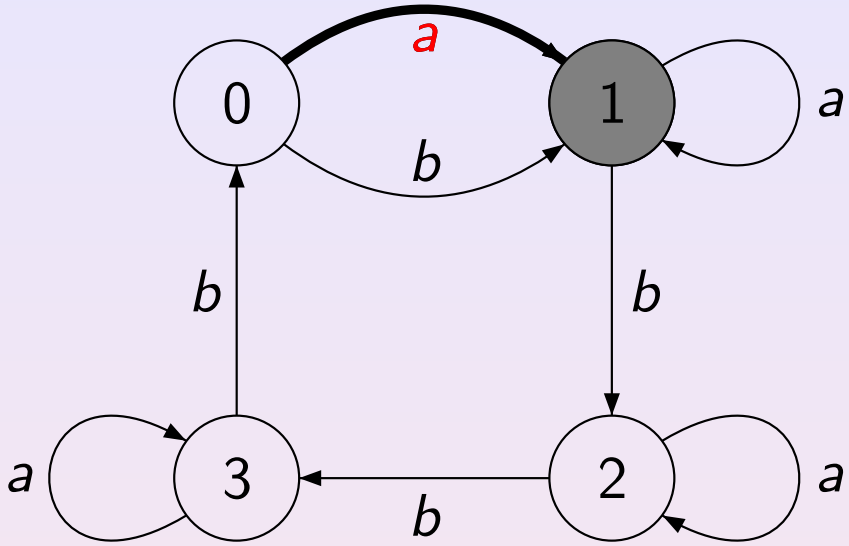


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

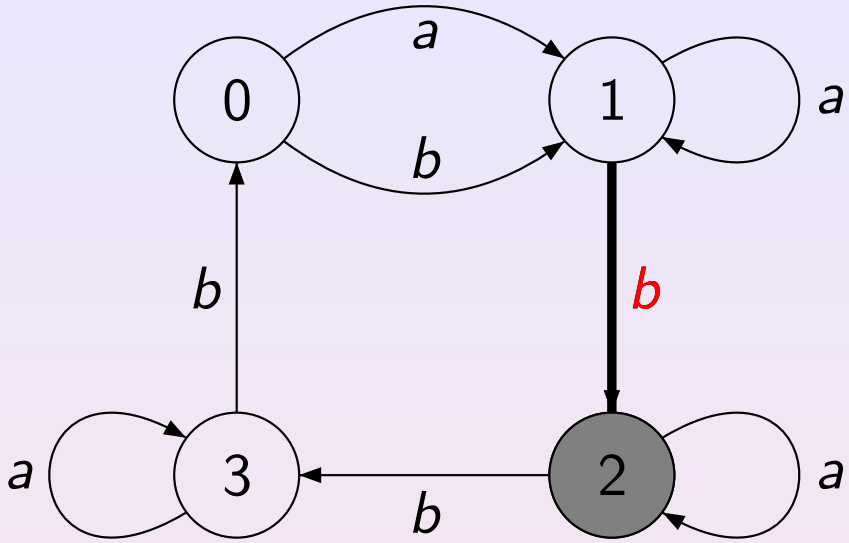


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

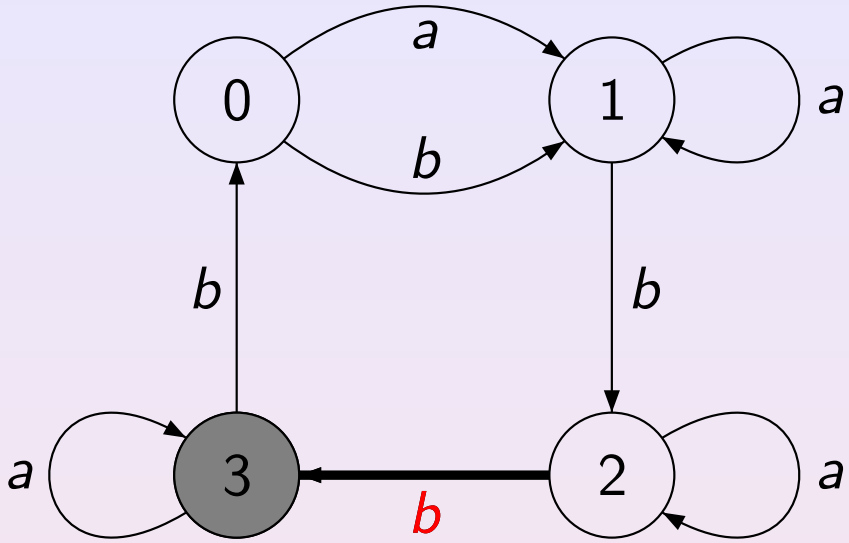


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

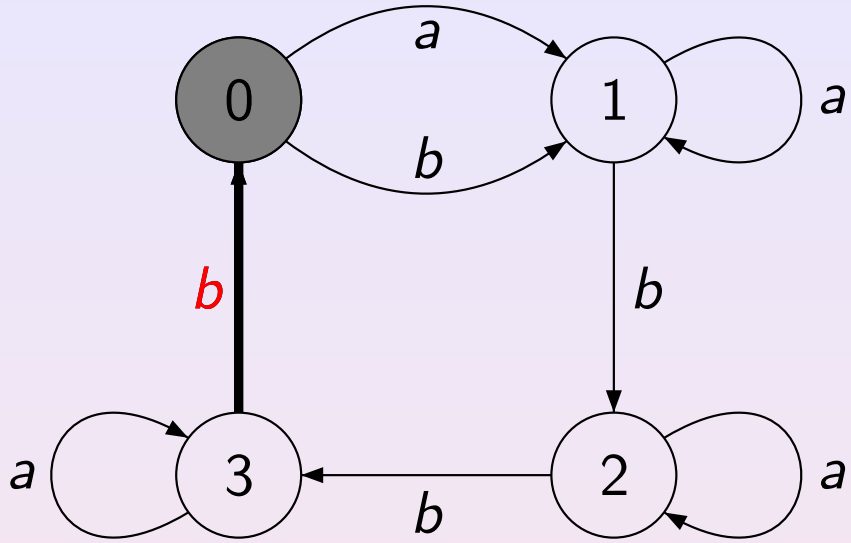


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

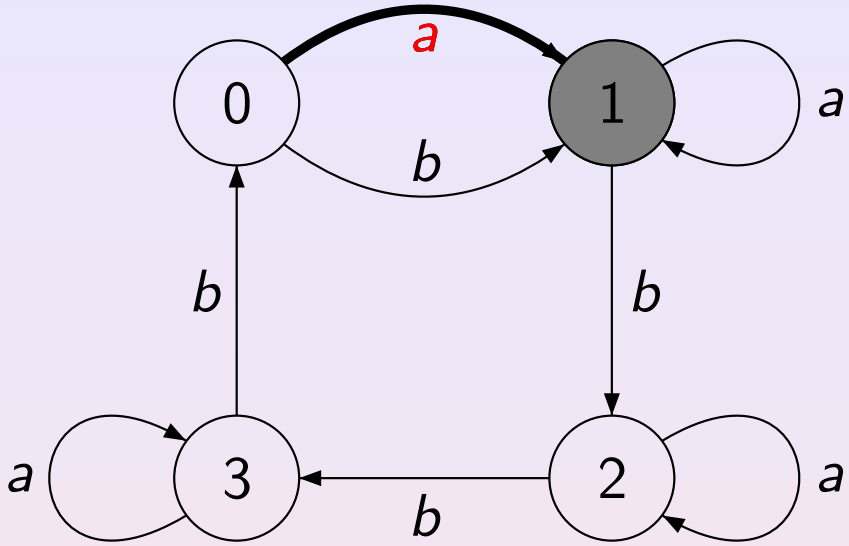


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

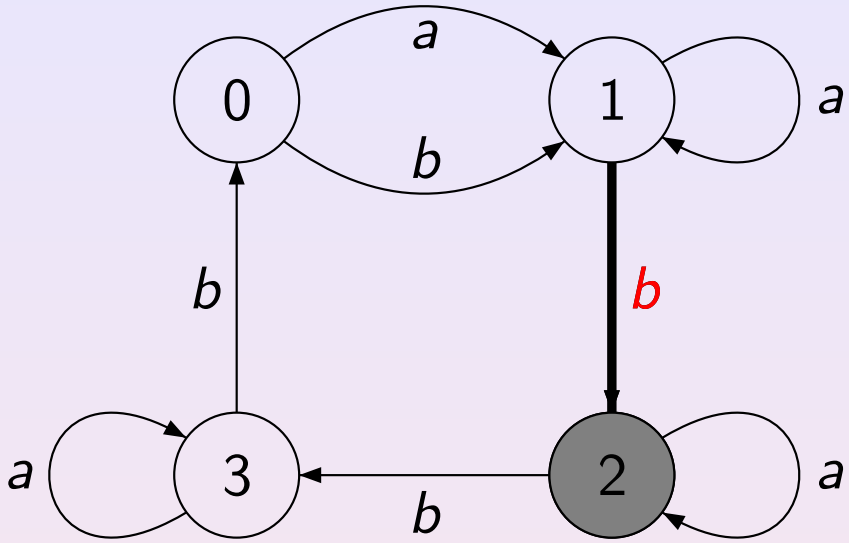


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

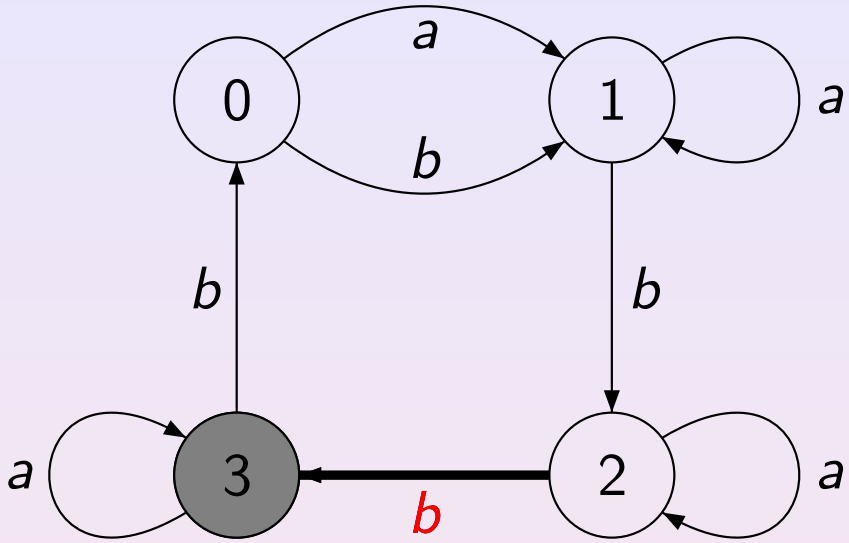


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

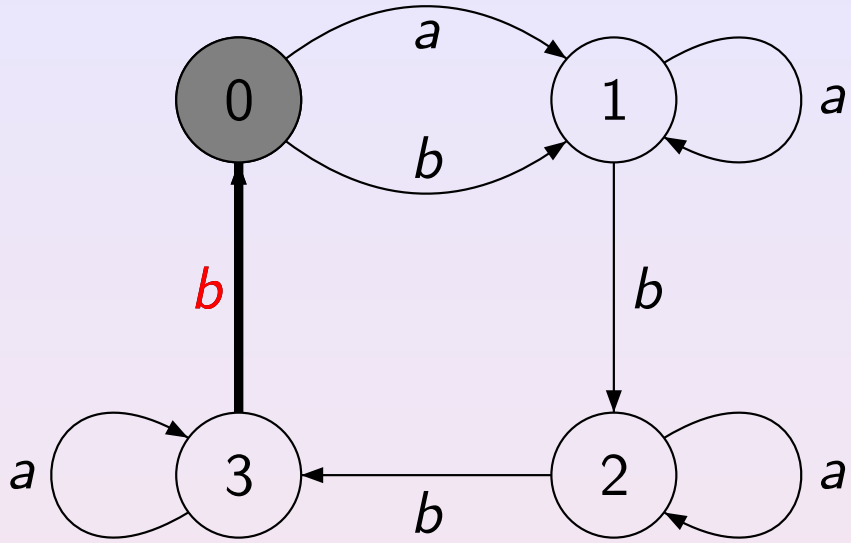


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

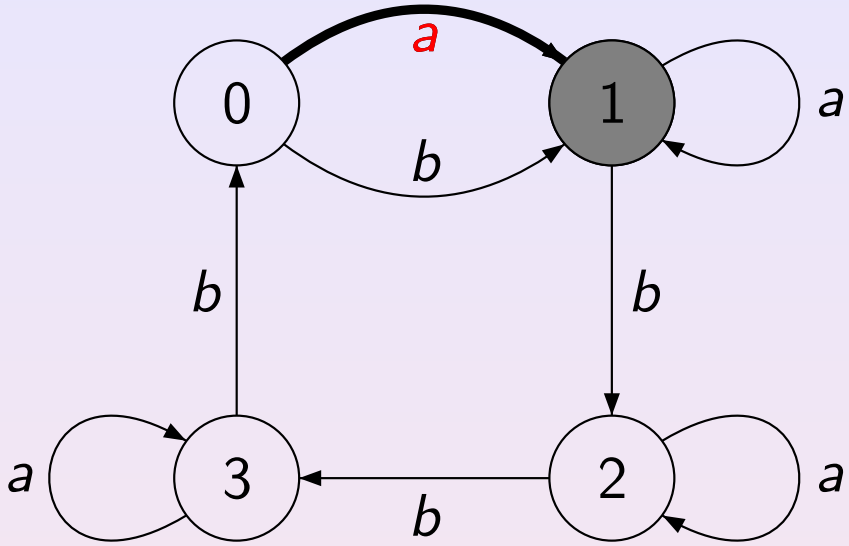


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

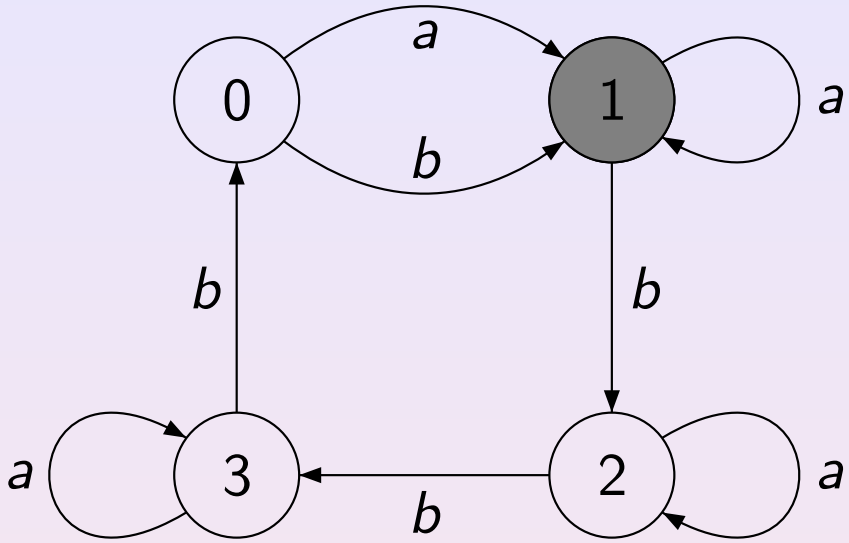


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

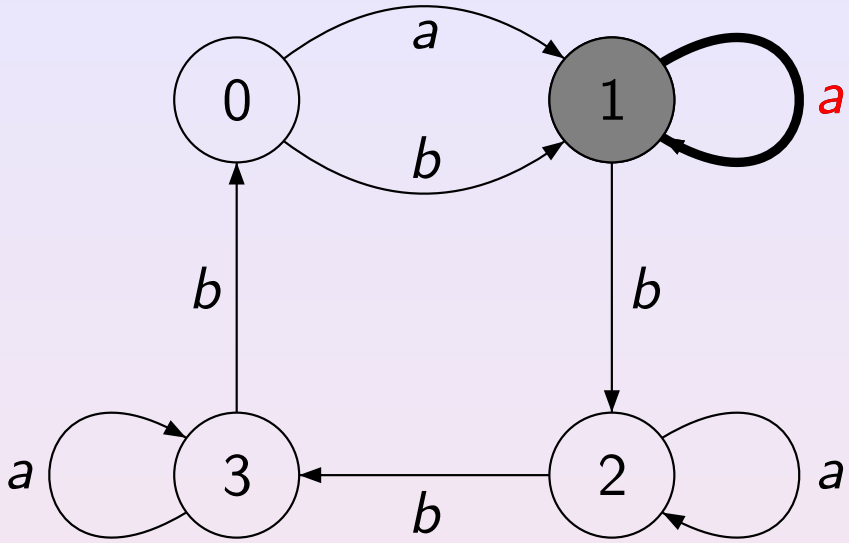


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

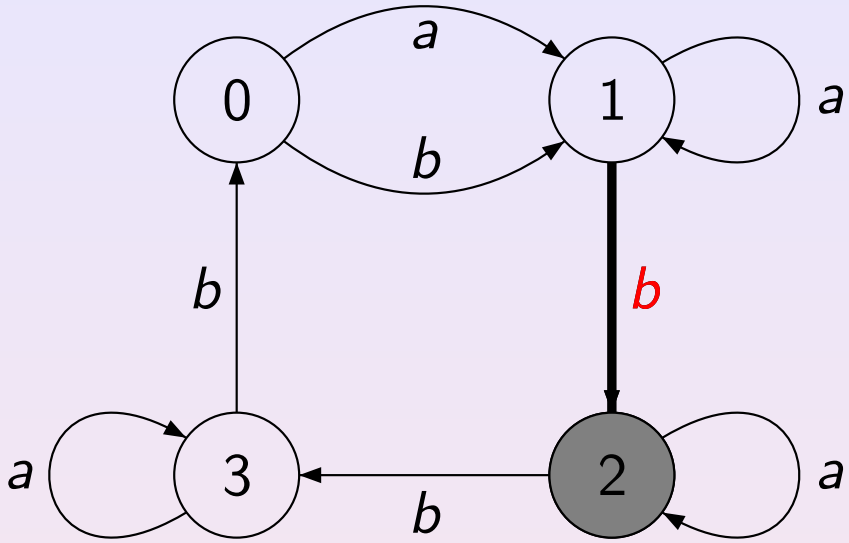


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

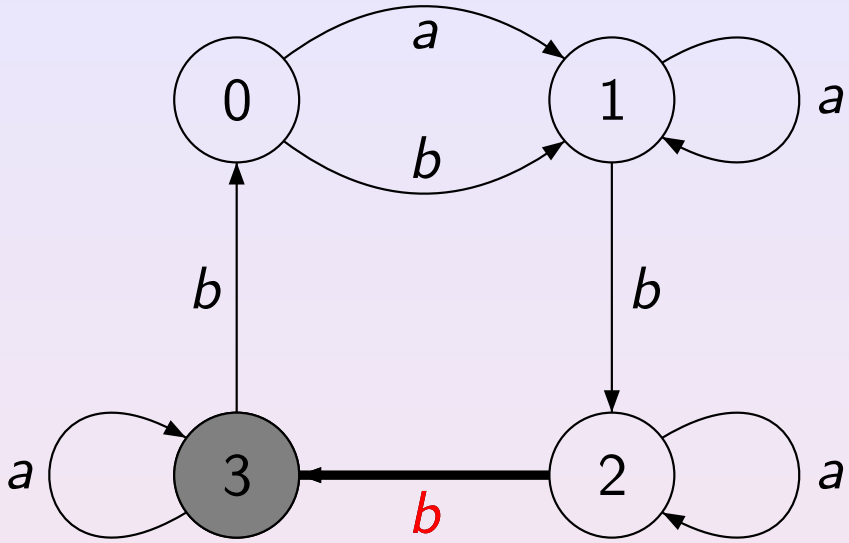


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

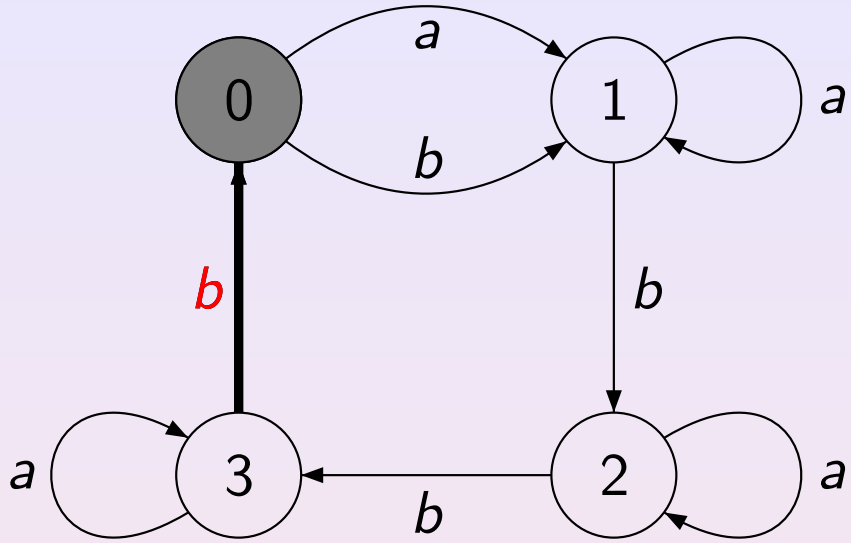


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

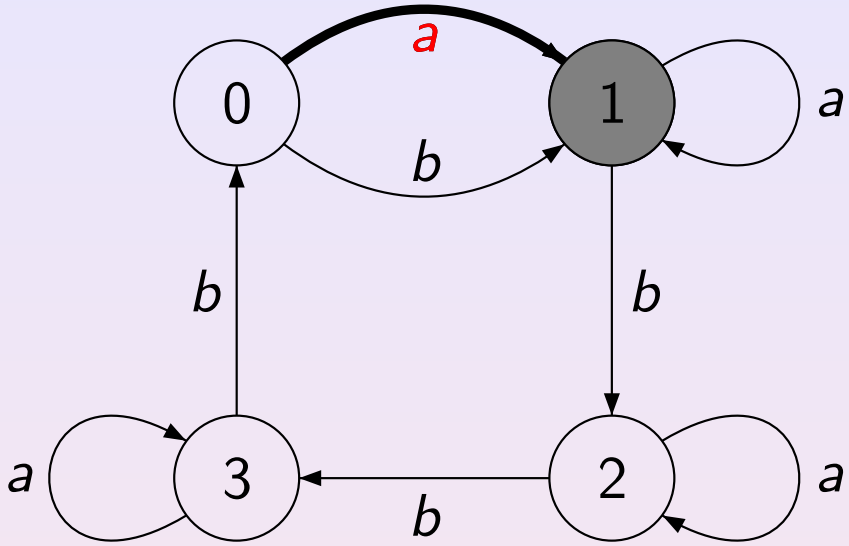


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

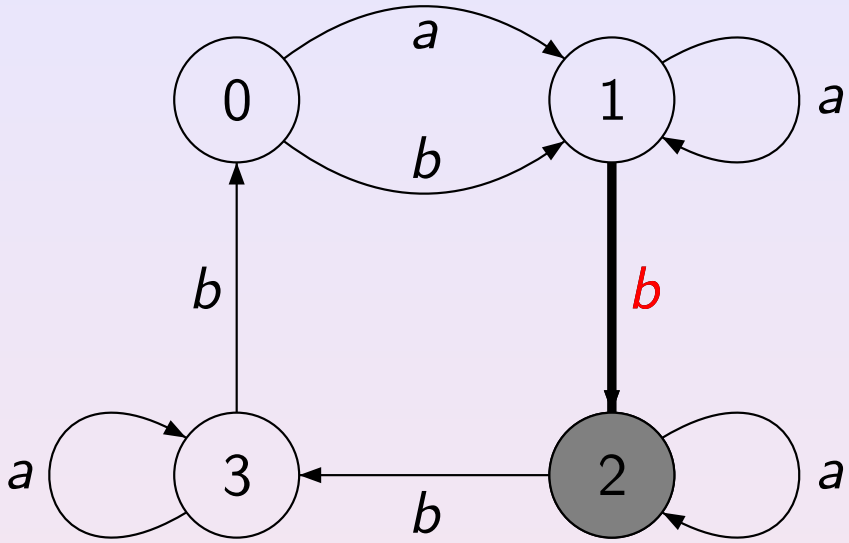


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

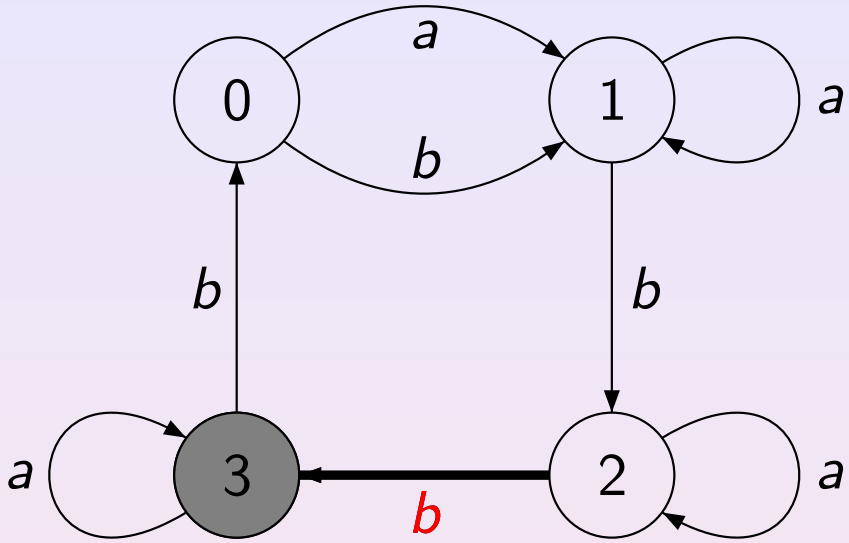


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

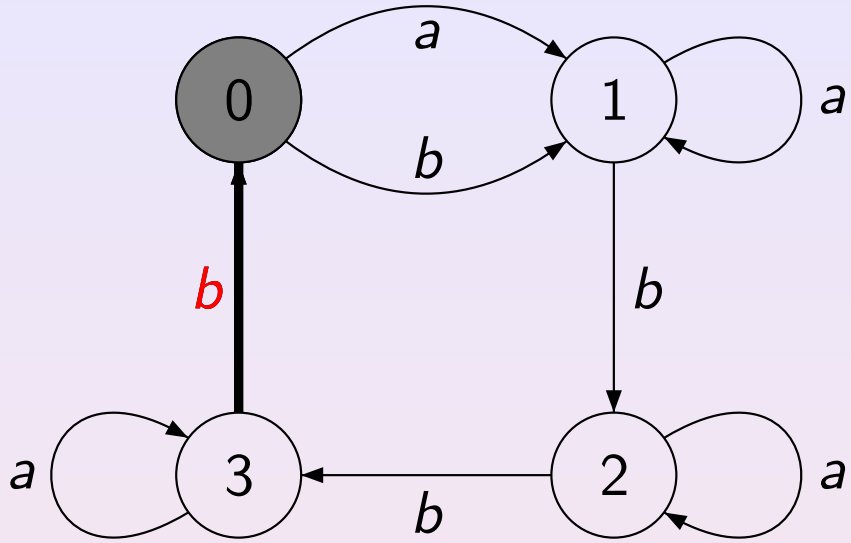


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

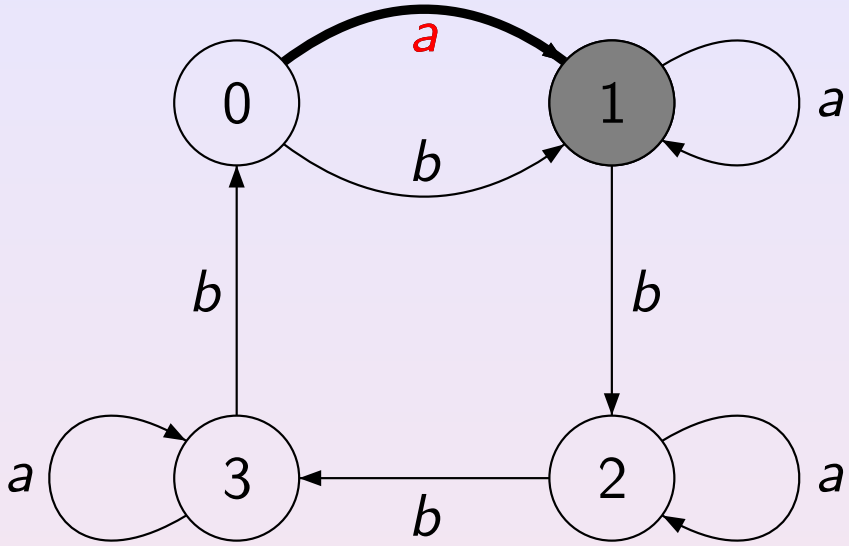


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

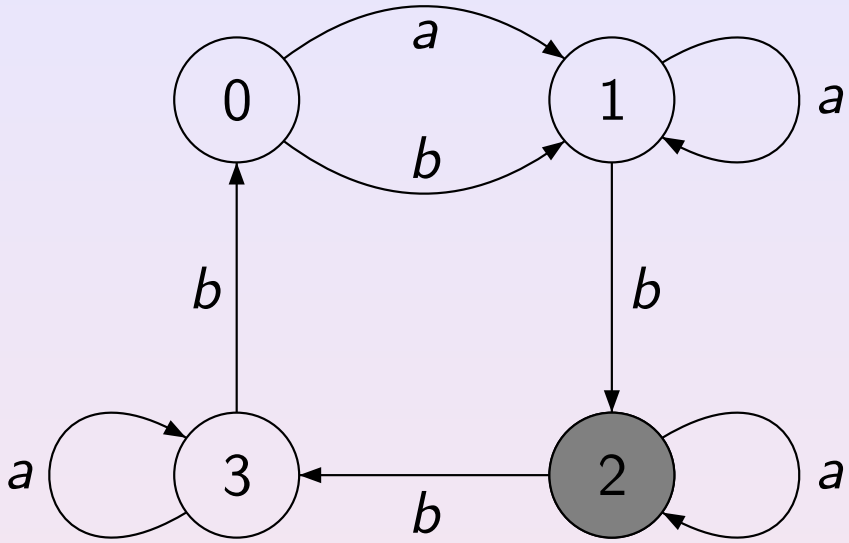


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

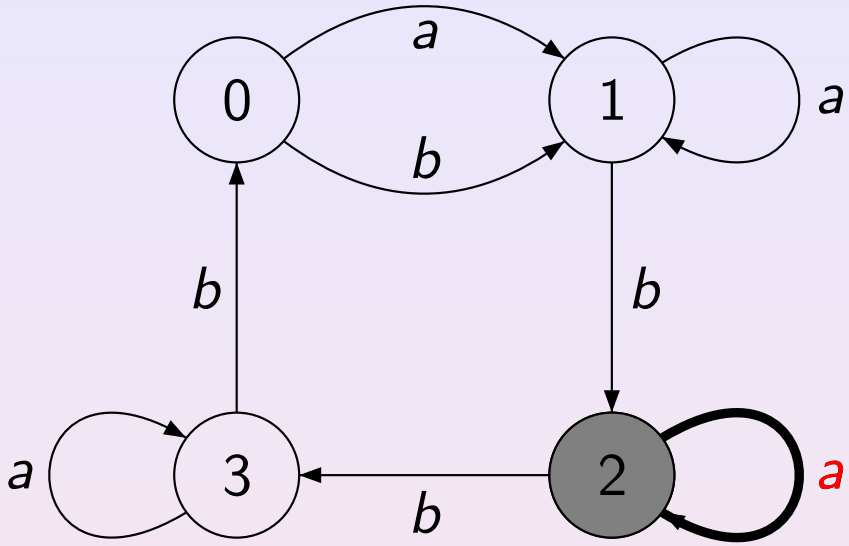


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

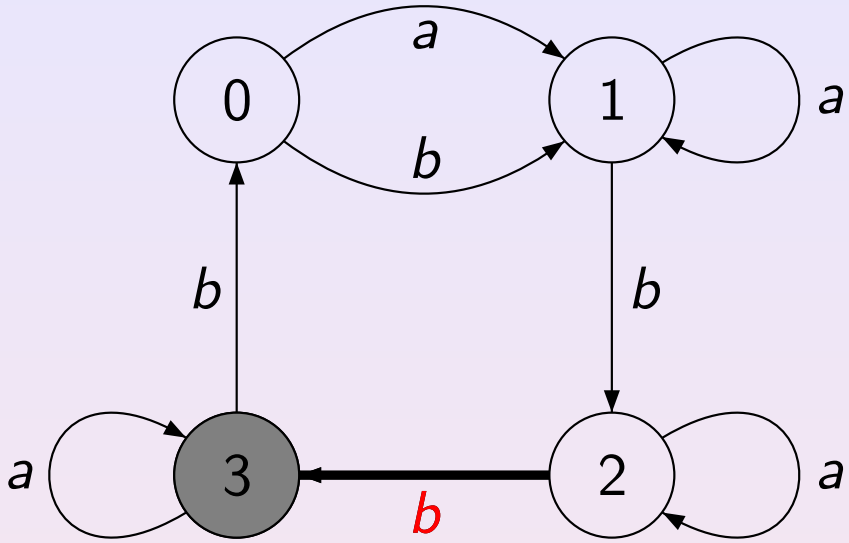


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

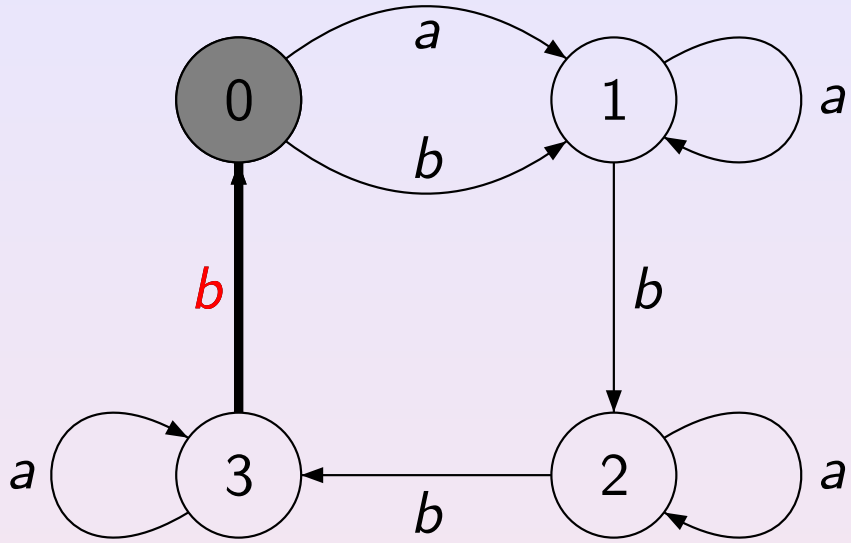


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

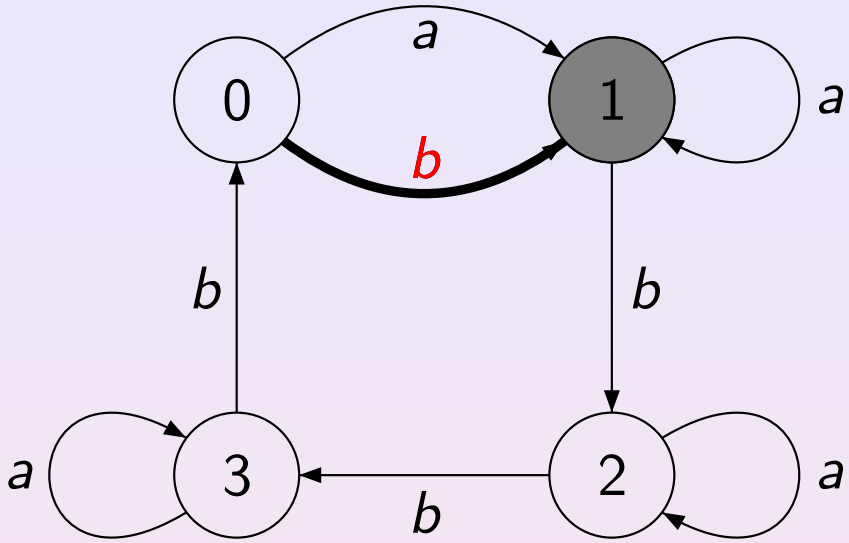


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

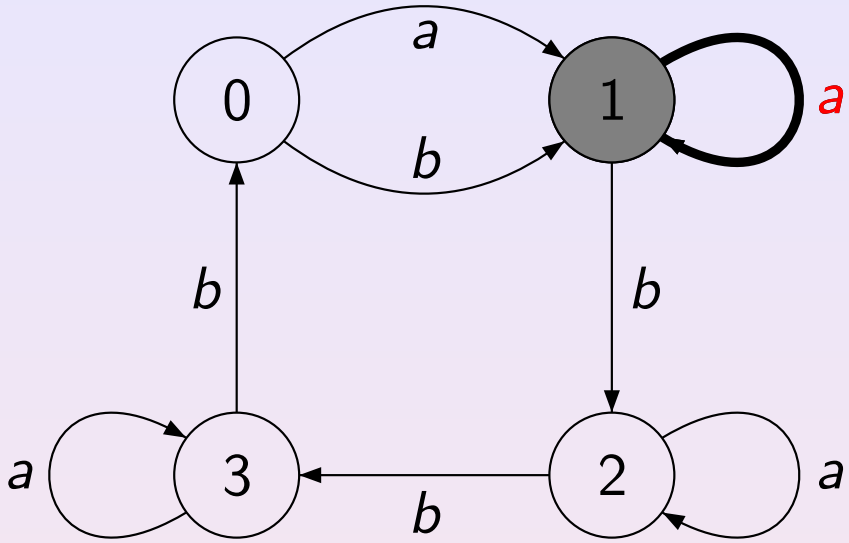


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

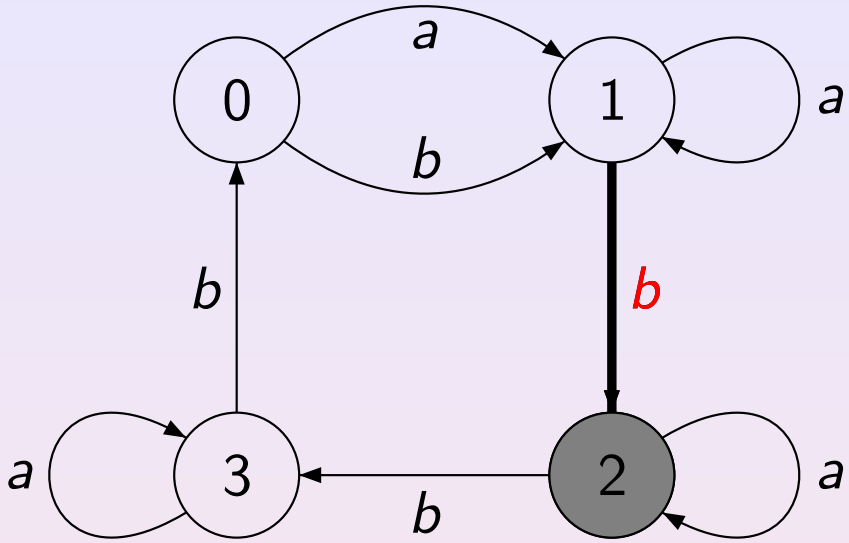


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

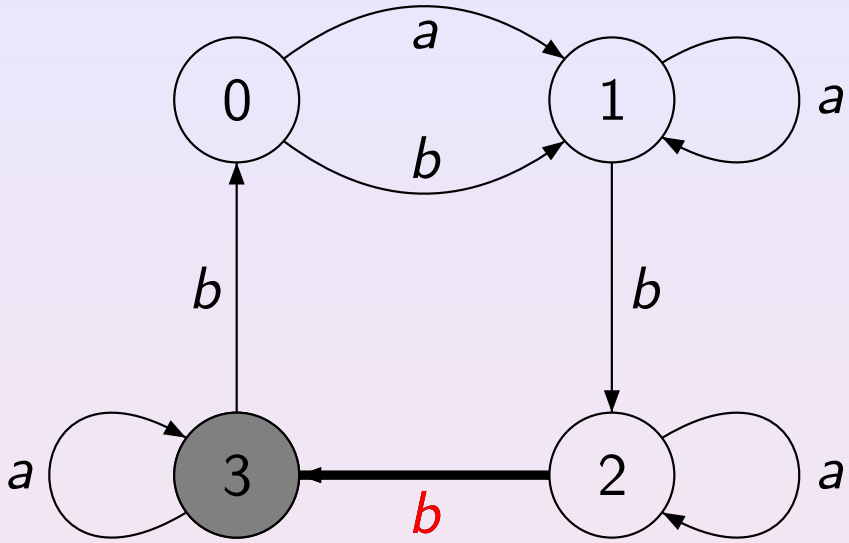


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

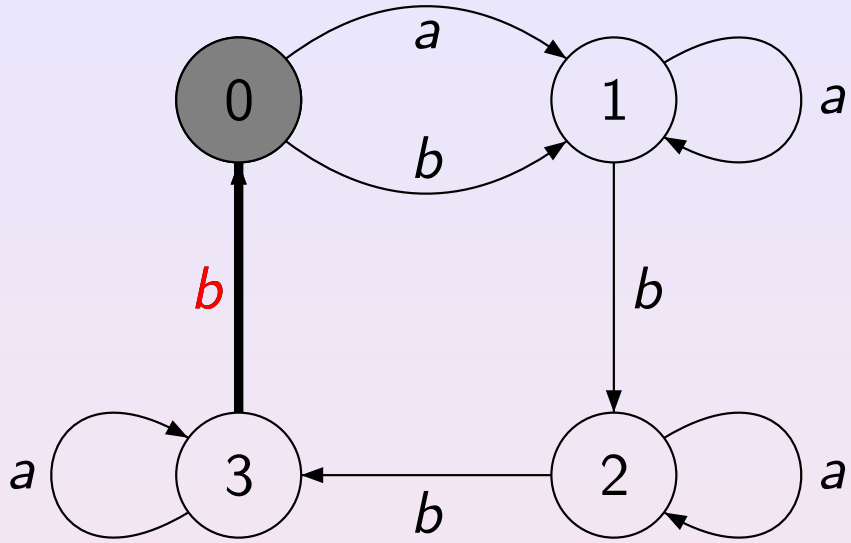


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

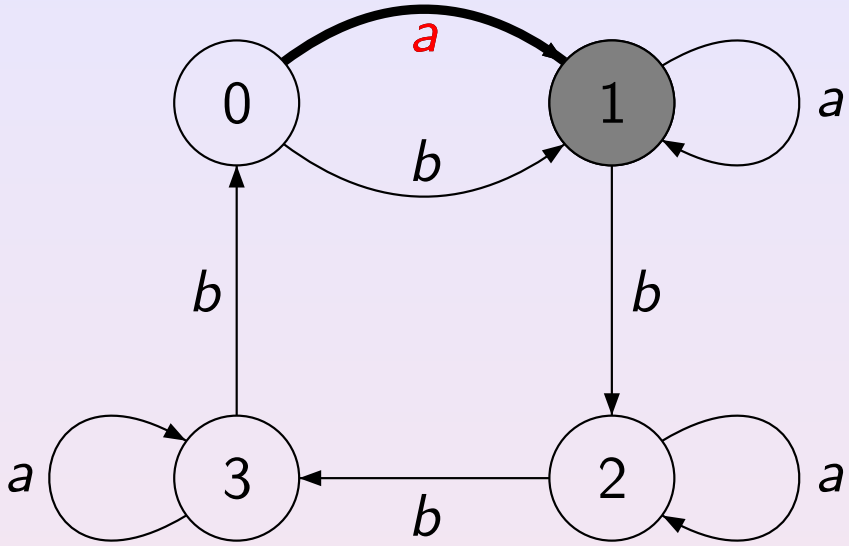


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

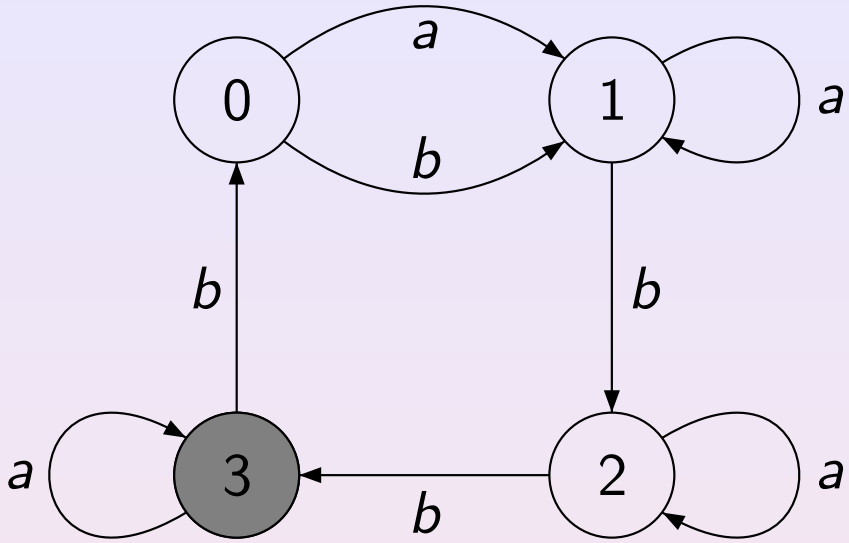


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

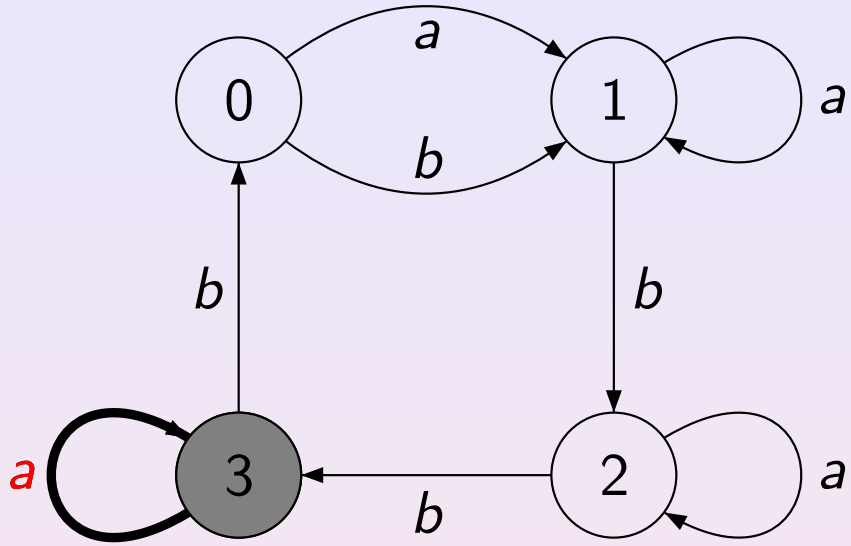


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

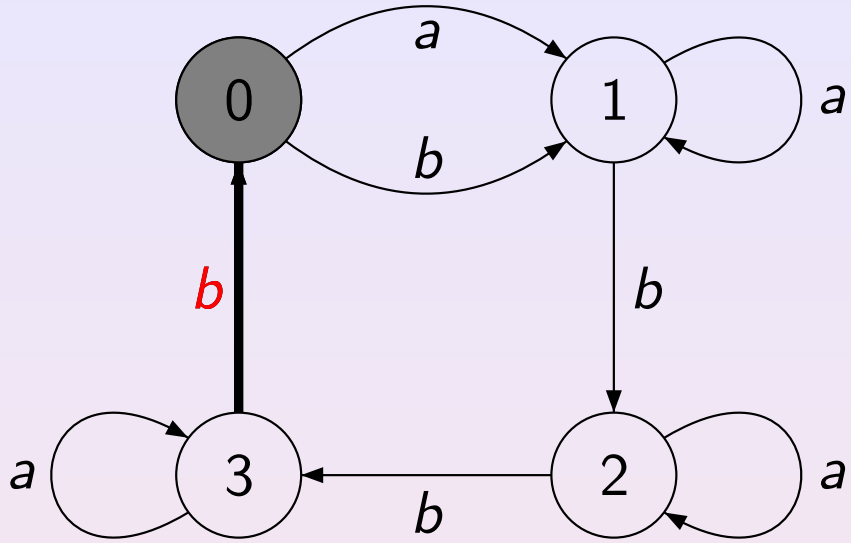


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

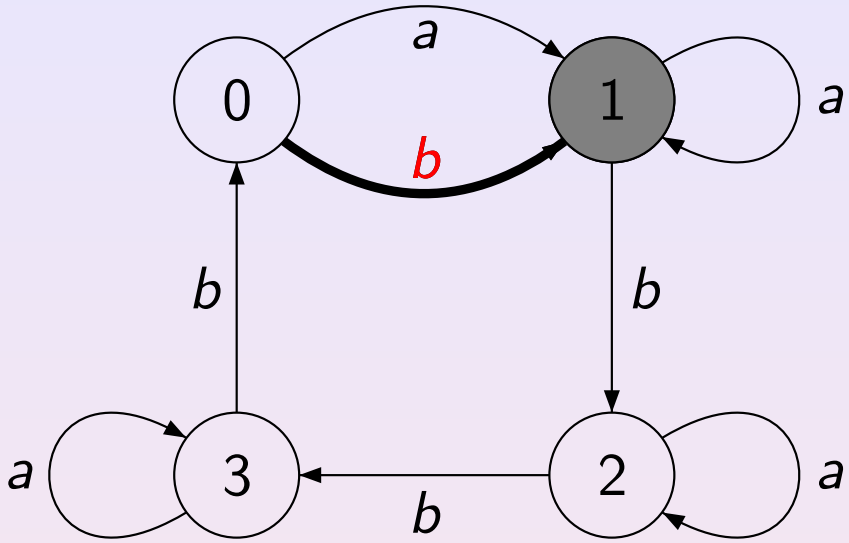


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

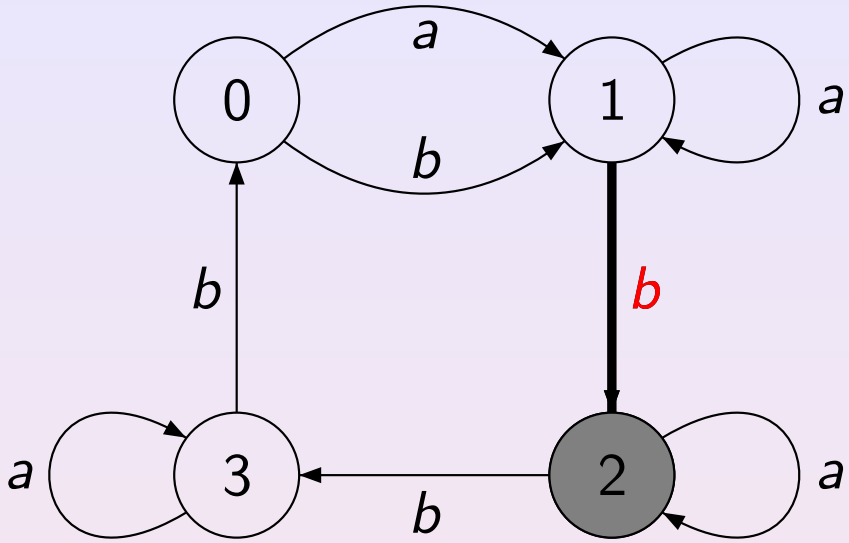


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

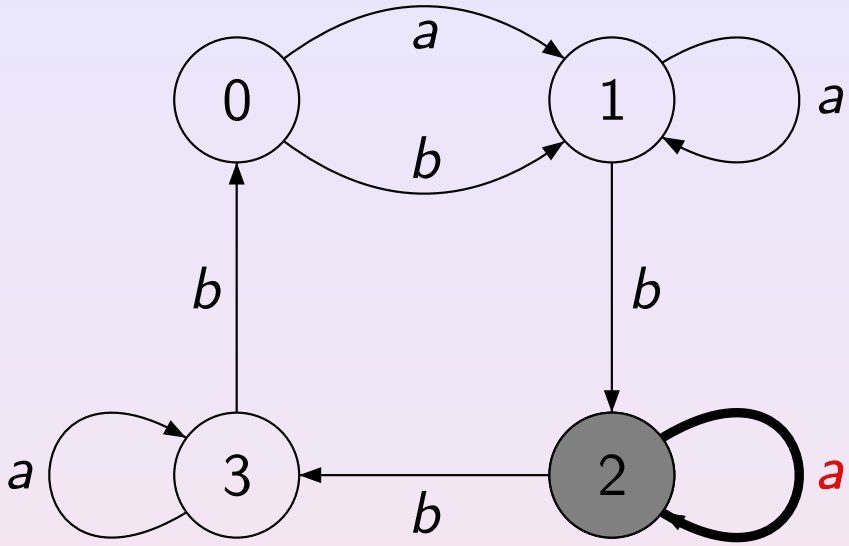


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

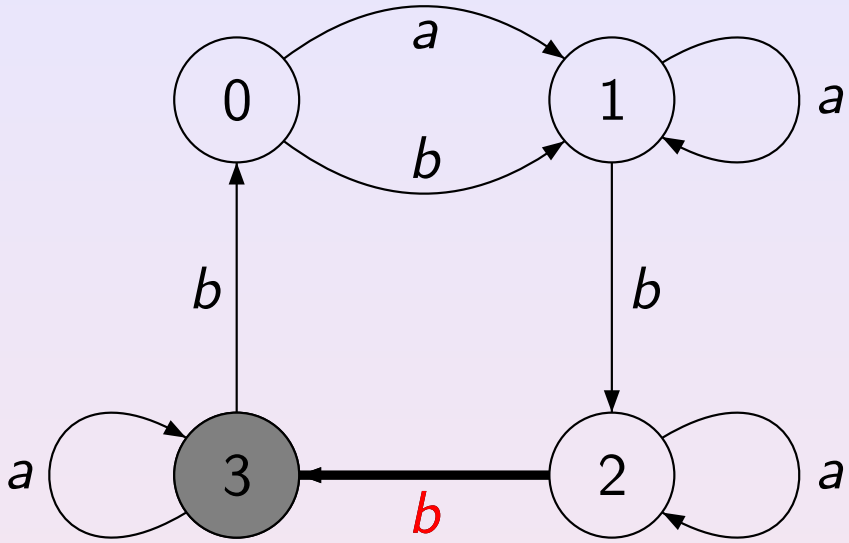


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

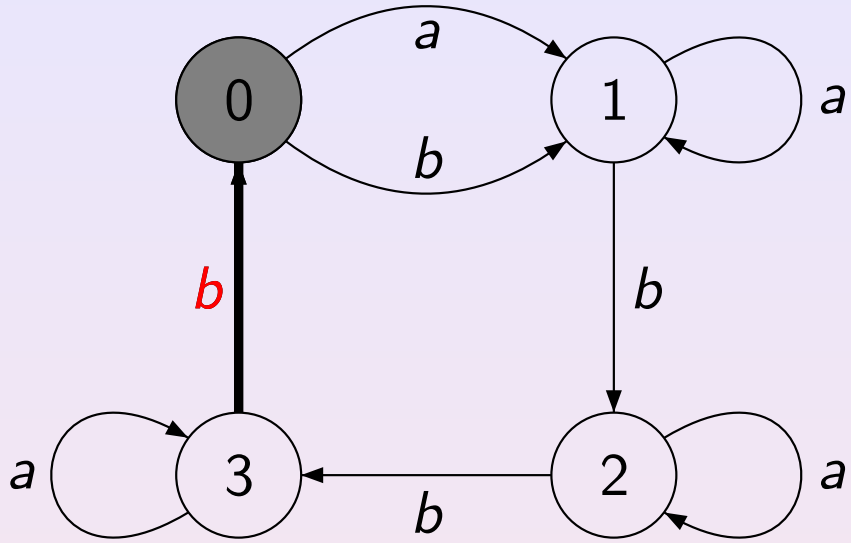


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

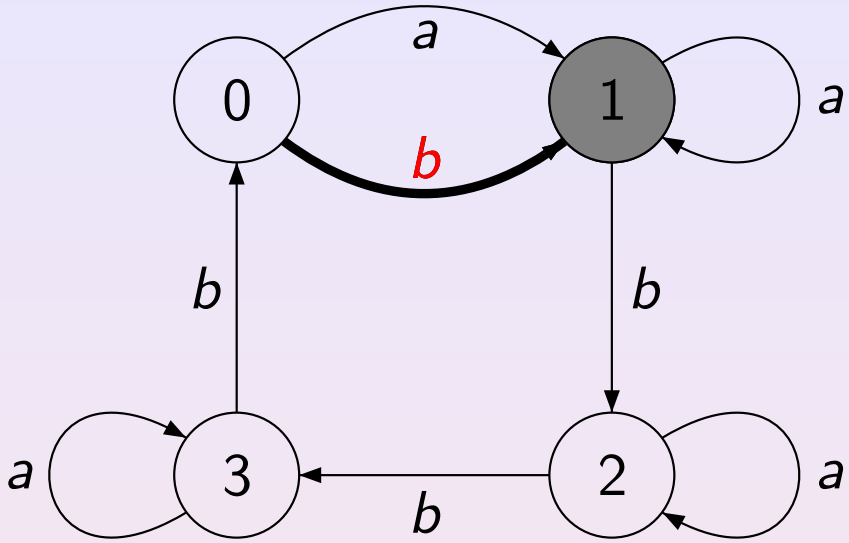


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

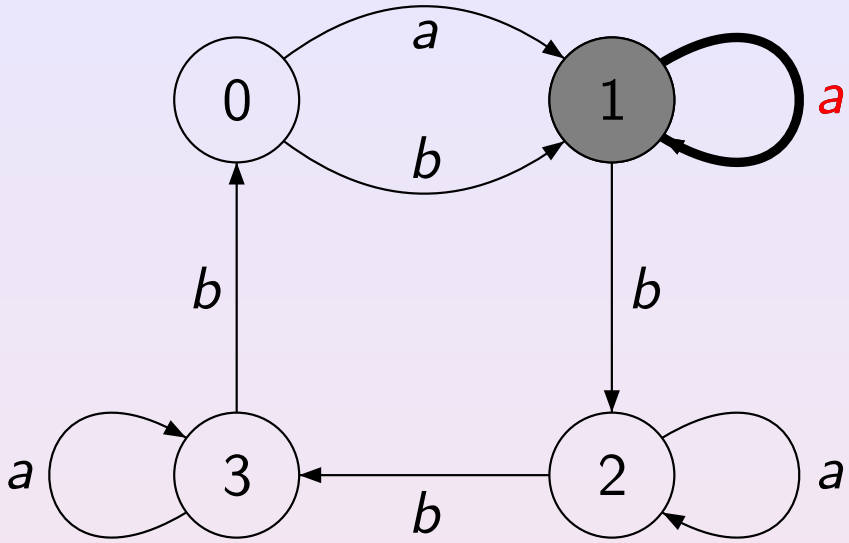


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example



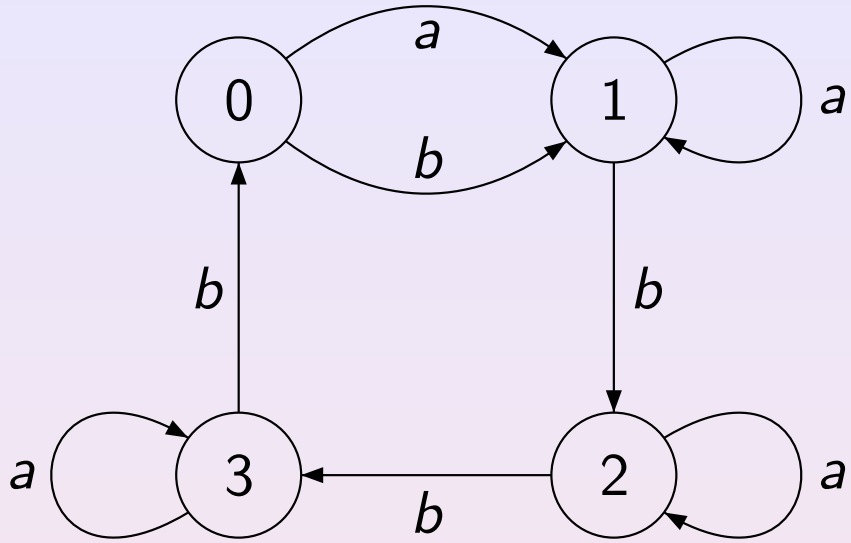
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964.

For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example

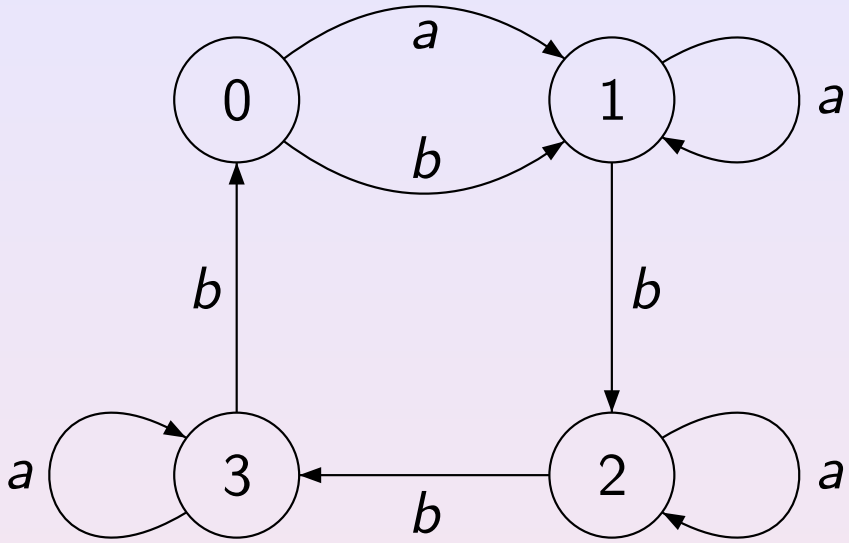


A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964. For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example



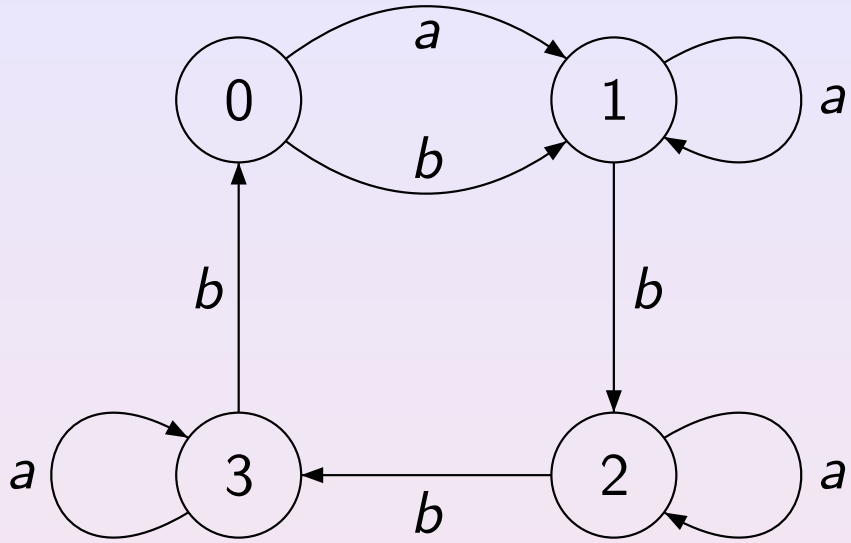
A reset word is *abbbabbba*: applying it at any state brings the automaton to the state 1.

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964.

For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

An Example



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

In fact, this is the reset word of minimum length for the automaton whence its reset threshold is 9.

The automaton belongs to the series $\mathscr{C}_n$ found by Jan Černý in 1964.

For each $n > 1$, the automaton $\mathscr{C}_n$ has n states, 2 input letters and reset threshold $(n-1)^2$.

The Černý Series

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

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

The automaton in the previous slide is $\mathscr{C}_4$.

The Černý Series

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

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

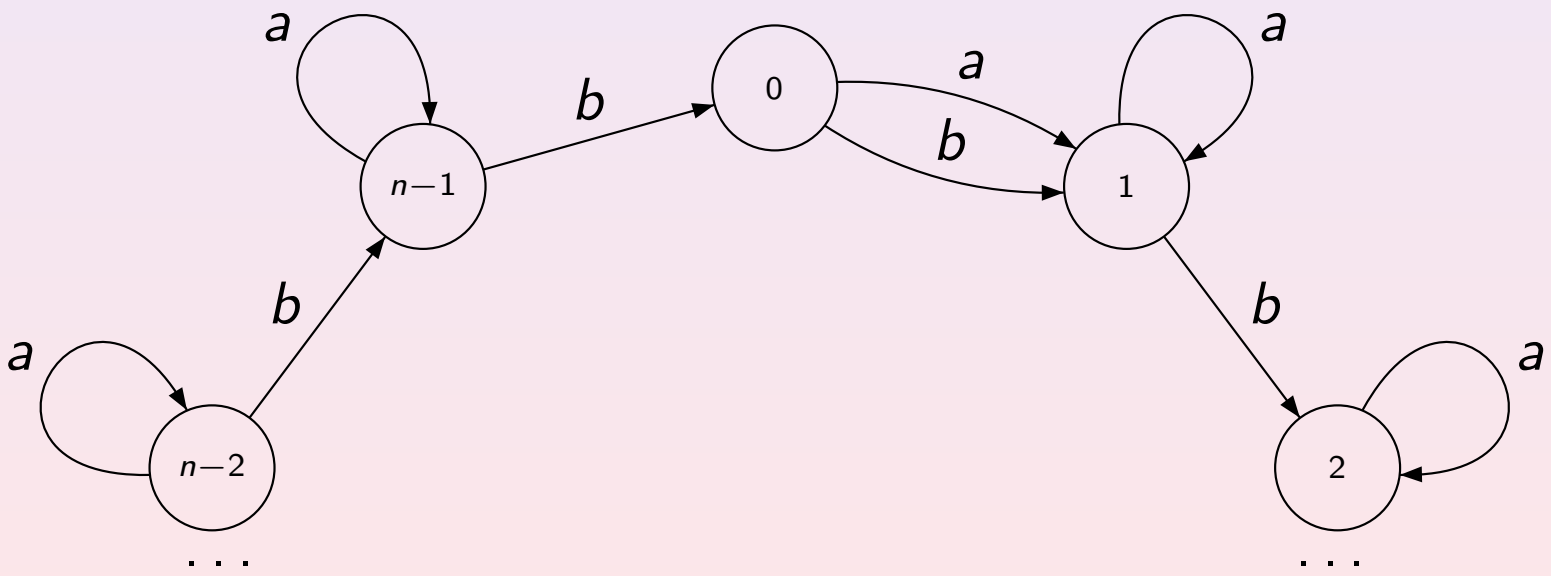
The automaton in the previous slide is $\mathscr{C}_4$.

The Černý Series

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

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

The automaton in the previous slide is $\mathscr{C}_4$.
Here is a generic automaton from the Černý series:

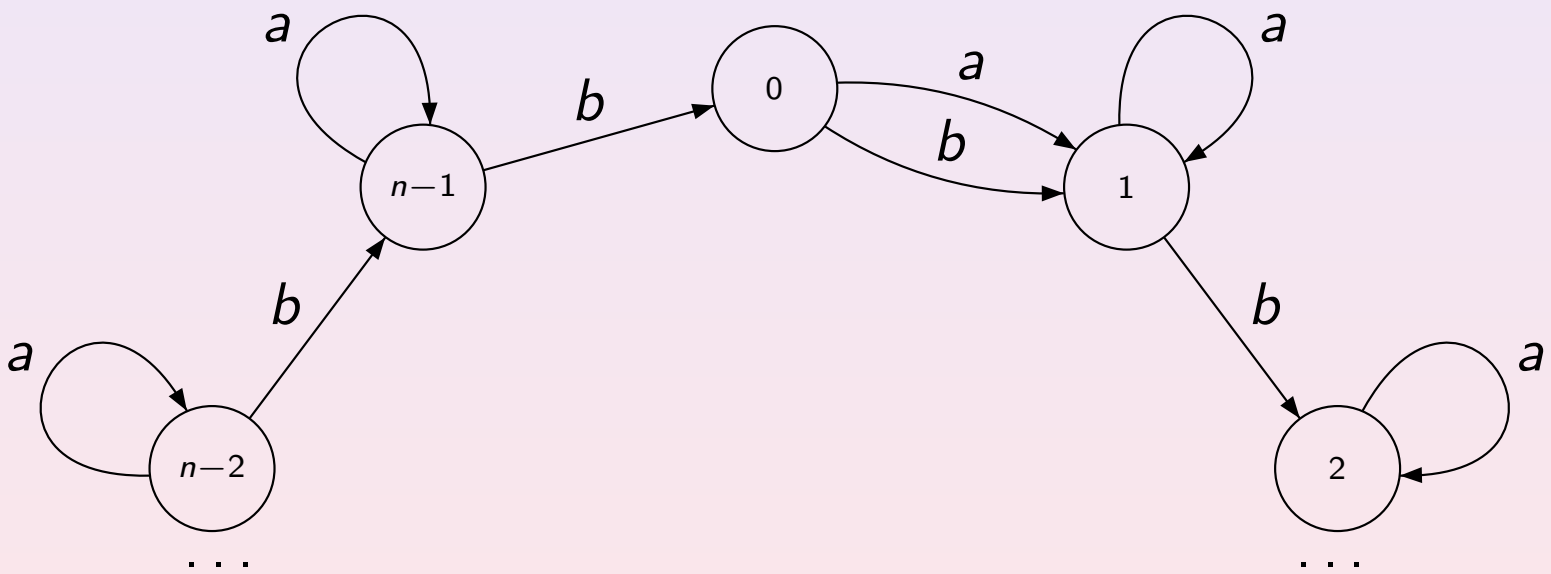


The Černý Series

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

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

The automaton in the previous slide is $\mathscr{C}_4$.
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 $n(n-2)+1 = (n-1)^2$.

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$.

The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$. The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$. The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$. The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$. The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

$$\text{(Černý, 1964)} \quad (n-1)^2 \leq C(n)$$

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$. The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

$$\text{(Černý, 1964)}\quad (n-1)^2 \leq C(n) \leq \frac{n^3-n}{6}\quad \text{(Pin–Frankl, 1983)}.$$

The Černý Conjecture

The Černý conjecture is the claim that every synchronizing automaton with n states possesses a reset word of length $(n-1)^2$. The validity of the conjecture is main open problem of the area and arguably one of the most long-standing open problems in combinatorial theory of finite automata.

Define the Černý function $C(n)$ as the maximum reset threshold for synchronizing automata with n states. In terms of this function, our current knowledge can be summarized in one line:

$$\text{(Černý, 1964)} \quad (n-1)^2 \leq C(n) \leq \frac{n^3-n}{6} \quad \text{(Pin–Frankl, 1983).}$$

The Černý conjecture thus claims that in fact $C(n) = (n-1)^2$.

Approaching the Černý Conjecture

Since the Černý Conjecture has proved to be hard in general, a natural strategy consists in considering its restriction to some special classes of DFAs.

The conjecture has been proved for many important special cases. This includes for instance:

- Louis Dubuc's result for automata in which a letter acts on the state set Q as a cyclic permutation of order $|Q|$ (Sur le automates circulaires et la conjecture de Černý, RAIRO Inform. Theor. Appl., 32 (1998) 21–34 [in French]).
- Jarkko Kari's result for automata with Eulerian digraphs (Synchronizing finite automata on Eulerian digraphs, Theoret. Comput. Sci., 295 (2003) 223–232).
- Benjamin Steinberg's result for automata in which a letter labels only one cycle (one-cluster automata) and this cycle is of prime length (The Černý conjecture for one-cluster automata with prime length cycle. Theoret. Comput. Sci. 412 (2011) 5487–5491).

Approaching the Černý Conjecture

Since the Černý Conjecture has proved to be hard in general, a natural strategy consists in considering its restriction to some special classes of DFAs.

The conjecture has been proved for many important special cases.

This includes for instance:

- Louis Dubuc's result for automata in which a letter acts on the state set Q as a cyclic permutation of order $|Q|$ (Sur le automates circulaires et la conjecture de Černý, RAIRO Inform. Theor. Appl., 32 (1998) 21–34 [in French]).
- Jarkko Kari's result for automata with Eulerian digraphs (Synchronizing finite automata on Eulerian digraphs, Theoret. Comput. Sci., 295 (2003) 223–232).
- Benjamin Steinberg's result for automata in which a letter labels only one cycle (one-cluster automata) and this cycle is of prime length (The Černý conjecture for one-cluster automata with prime length cycle. Theoret. Comput. Sci. 412 (2011) 5487–5491).

Approaching the Černý Conjecture

Since the Černý Conjecture has proved to be hard in general, a natural strategy consists in considering its restriction to some special classes of DFAs.
The conjecture has been proved for many important special cases. This includes for instance:

• Louis Dubuc's result for automata in which a letter acts on the state set Q as a cyclic permutation of order $|Q|$ (Sur le automates circulaires et la conjecture de Černý, RAIRO Inform. Theor. Appl., 32 (1998) 21–34 [in French]).

• Jarkko Kari's result for automata with Eulerian digraphs (Synchronizing finite automata on Eulerian digraphs, Theoret. Comput. Sci., 295 (2003) 223–232).

• Benjamin Steinberg's result for automata in which a letter labels only one cycle (one-cluster automata) and this cycle is of prime length (The Černý conjecture for one-cluster automata with prime length cycle. Theoret. Comput. Sci. 412 (2011) 5487–5491).

Approaching the Černý Conjecture

Since the Černý Conjecture has proved to be hard in general, a natural strategy consists in considering its restriction to some special classes of DFAs.
The conjecture has been proved for many important special cases. This includes for instance:

- Louis Dubuc's result for automata in which a letter acts on the state set Q as a cyclic permutation of order $|Q|$ (Sur le automates circulaires et la conjecture de Černý, RAIRO Inform. Theor. Appl., 32 (1998) 21–34 [in French]).
- Jarkko Kari's result for automata with Eulerian digraphs (Synchronizing finite automata on Eulerian digraphs, Theoret. Comput. Sci., 295 (2003) 223–232).
- Benjamin Steinberg's result for automata in which a letter labels only one cycle (one-cluster automata) and this cycle is of prime length (The Černý conjecture for one-cluster automata with prime length cycle. Theoret. Comput. Sci. 412 (2011) 5487–5491).

Approaching the Černý Conjecture

Since the Černý Conjecture has proved to be hard in general, a natural strategy consists in considering its restriction to some special classes of DFAs.
The conjecture has been proved for many important special cases. This includes for instance:

• Louis Dubuc's result for automata in which a letter acts on the state set Q as a cyclic permutation of order $|Q|$ (Sur le automates circulaires et la conjecture de Černý, RAIRO Inform. Theor. Appl., 32 (1998) 21–34 [in French]).

• Jarkko Kari's result for automata with Eulerian digraphs (Synchronizing finite automata on Eulerian digraphs, Theoret. Comput. Sci., 295 (2003) 223–232).

• Benjamin Steinberg's result for automata in which a letter labels only one cycle (one-cluster automata) and this cycle is of prime length (The Černý conjecture for one-cluster automata with prime length cycle. Theoret. Comput. Sci. 412 (2011) 5487–5491).

An Observation

Observation (Marina Maslennikova, arXiv:1404.2816; Henk Don, arXiv:1507.06070)

The Černý automata $\mathscr{C}_n$ are completely reachable.

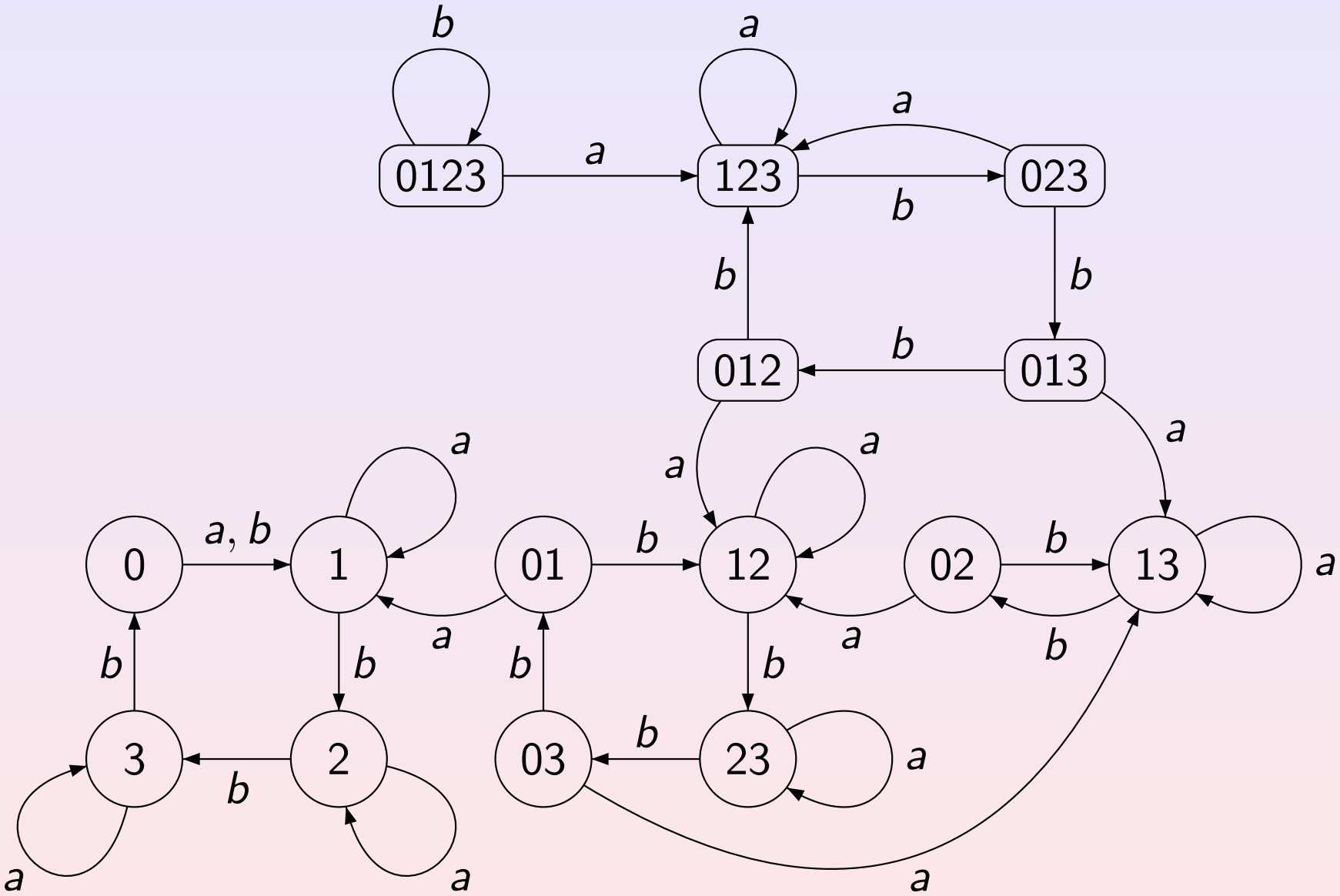
An Observation

Observation (Marina Maslennikova, arXiv:1404.2816; Henk Don, arXiv:1507.06070)

The Černý automata $\mathscr{C}_n$ are completely reachable.

For an illustration, consider the power-set automaton of the Černý automaton $\mathscr{C}_4$.

An Observation



Restricting to Completely Reachable Automata

Recall that every completely reachable automaton is synchronizing.

On the other hand, the above observation ensures that the lower bound $(n-1)^2$ for the Černý function $C(n)$ is attained by a family of completely reachable automata.

Therefore completely reachable automata forms quite a natural class to study from the viewpoint of the Černý conjecture.

It is relatively easy to show that the conjecture holds true for completely reachable automata with two input letters; however, the general case remains open.

Some partial results in this direction have been obtained by Don.

Restricting to Completely Reachable Automata

Recall that every completely reachable automaton is synchronizing. On the other hand, the above observation ensures that the lower bound $(n-1)^2$ for the Černý function $C(n)$ is attained by a family of completely reachable automata.

Therefore completely reachable automata forms quite a natural class to study from the viewpoint of the Černý conjecture.

It is relatively easy to show that the conjecture holds true for completely reachable automata with two input letters; however, the general case remains open.

Some partial results in this direction have been obtained by Don.

Restricting to Completely Reachable Automata

Recall that every completely reachable automaton is synchronizing. On the other hand, the above observation ensures that the lower bound $(n-1)^2$ for the Černý function $C(n)$ is attained by a family of completely reachable automata.

Therefore completely reachable automata forms quite a natural class to study from the viewpoint of the Černý conjecture.

It is relatively easy to show that the conjecture holds true for completely reachable automata with two input letters; however, the general case remains open.

Some partial results in this direction have been obtained by Don.

Restricting to Completely Reachable Automata

Recall that every completely reachable automaton is synchronizing. On the other hand, the above observation ensures that the lower bound $(n-1)^2$ for the Černý function $C(n)$ is attained by a family of completely reachable automata.

Therefore completely reachable automata forms quite a natural class to study from the viewpoint of the Černý conjecture.

It is relatively easy to show that the conjecture holds true for completely reachable automata with two input letters; however, the general case remains open.

Some partial results in this direction have been obtained by Don.

Restricting to Completely Reachable Automata

Recall that every completely reachable automaton is synchronizing. On the other hand, the above observation ensures that the lower bound $(n-1)^2$ for the Černý function $C(n)$ is attained by a family of completely reachable automata.

Therefore completely reachable automata forms quite a natural class to study from the viewpoint of the Černý conjecture.

It is relatively easy to show that the conjecture holds true for completely reachable automata with two input letters; however, the general case remains open.

Some partial results in this direction have been obtained by Don.

Restricting to Completely Reachable Automata

Recall that every completely reachable automaton is synchronizing. On the other hand, the above observation ensures that the lower bound $(n-1)^2$ for the Černý function $C(n)$ is attained by a family of completely reachable automata.

Therefore completely reachable automata forms quite a natural class to study from the viewpoint of the Černý conjecture.

It is relatively easy to show that the conjecture holds true for completely reachable automata with two input letters; however, the general case remains open.

Some partial results in this direction have been obtained by Don.

Second Motivation: Syntactic Complexity

Syntactic complexity of a regular language is a well established concept that has attracted much attention lately, in particular, in a nice series of recent papers by Janusz Brzozowski and his collaborators. It is defined as the size of the transition monoid of the minimal DFA recognizing the language. The transition monoid $M(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is the monoid of all transformations of the set Q induced by the words in Σ^*.

It appears to be worthwhile to extend this concept to automata by defining the syntactic complexity of an arbitrary DFA $\mathscr{A}$ as the size of its transition monoid $M(\mathscr{A})$. If one thinks of a DFA as a computational device rather than acceptor, its transition monoid can be thought of as the device's 'software library', and measuring the complexity of an automaton by the size of its 'software library' is fairly natural.

Second Motivation: Syntactic Complexity

Syntactic complexity of a regular language is a well established concept that has attracted much attention lately, in particular, in a nice series of recent papers by Janusz Brzozowski and his collaborators. It is defined as the size of the transition monoid of the minimal DFA recognizing the language. The transition monoid $M(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is the monoid of all transformations of the set Q induced by the words in Σ^*.

It appears to be worthwhile to extend this concept to automata by defining the syntactic complexity of an arbitrary DFA $\mathscr{A}$ as the size of its transition monoid $M(\mathscr{A})$. If one thinks of a DFA as a computational device rather than acceptor, its transition monoid can be thought of as the device's 'software library', and measuring the complexity of an automaton by the size of its 'software library' is fairly natural.

Second Motivation: Syntactic Complexity

Syntactic complexity of a regular language is a well established concept that has attracted much attention lately, in particular, in a nice series of recent papers by Janusz Brzozowski and his collaborators. It is defined as the size of the transition monoid of the minimal DFA recognizing the language. The transition monoid $M(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is the monoid of all transformations of the set Q induced by the words in Σ^*.

It appears to be worthwhile to extend this concept to automata by defining the syntactic complexity of an arbitrary DFA $\mathscr{A}$ as the size of its transition monoid $M(\mathscr{A})$. If one thinks of a DFA as a computational device rather than acceptor, its transition monoid can be thought of as the device's 'software library', and measuring the complexity of an automaton by the size of its 'software library' is fairly natural.

Second Motivation: Syntactic Complexity

Syntactic complexity of a regular language is a well established concept that has attracted much attention lately, in particular, in a nice series of recent papers by Janusz Brzozowski and his collaborators. It is defined as the size of the transition monoid of the minimal DFA recognizing the language. The transition monoid $M(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is the monoid of all transformations of the set Q induced by the words in Σ^*.

It appears to be worthwhile to extend this concept to automata by defining the syntactic complexity of an arbitrary DFA $\mathscr{A}$ as the size of its transition monoid $M(\mathscr{A})$. If one thinks of a DFA as a computational device rather than acceptor, its transition monoid can be thought of as the device's 'software library', and measuring the complexity of an automaton by the size of its 'software library' is fairly natural.

Second Motivation: Syntactic Complexity

Syntactic complexity of a regular language is a well established concept that has attracted much attention lately, in particular, in a nice series of recent papers by Janusz Brzozowski and his collaborators. It is defined as the size of the transition monoid of the minimal DFA recognizing the language. The transition monoid $M(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is the monoid of all transformations of the set Q induced by the words in Σ^*.

It appears to be worthwhile to extend this concept to automata by defining the syntactic complexity of an arbitrary DFA $\mathscr{A}$ as the size of its transition monoid $M(\mathscr{A})$. If one thinks of a DFA as a computational device rather than acceptor, its transition monoid can be thought of as the device's 'software library', and measuring the complexity of an automaton by the size of its 'software library' is fairly natural.

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R}\, y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L}\, y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H}\, y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J}\, y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R} y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L} y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H} y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J} y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R} y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L} y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H} y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J} y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R}\, y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L}\, y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H}\, y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J}\, y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R}\, y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L}\, y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H}\, y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J}\, y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R}\, y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L}\, y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H}\, y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J}\, y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R}\, y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L}\, y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H}\, y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J}\, y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R} y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L} y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H} y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J} y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

Green's Relations

The following four equivalence relations can be defined on every monoid M:

$x \mathcal{R}\, y \Leftrightarrow xM = yM$, i.e., x and y are prefixes of each other;

$x \mathcal{L}\, y \Leftrightarrow Mx = My$, i.e., x and y are suffixes of each other;

$x \mathcal{H}\, y \Leftrightarrow xM = yM \wedge Mx = My$, i.e., $\mathcal{H} = \mathcal{R} \cap \mathcal{L}$;

$x \mathcal{J}\, y \Leftrightarrow MxM = MyM$, i.e., x and y are factors of each other.

They were introduced by James Alexander Green in 1951 and since then they have played a crucial role in monoid theory and its applications to formal languages.

A language is star-free iff its syntactic monoid is $\mathcal{H}$-trivial (Marcel-Paul Schützenberger, 1964).

A language is piecewise testable iff its syntactic monoid is $\mathcal{J}$-trivial (Imre Simon, 1972).

$\mathcal{L}$-Trivial Case

Brzozowski and his collaborators have studied in depth the syntactic complexity of star-free ($\mathcal{H}$-trivial), piecewise testable ($\mathcal{J}$-trivial), and $\mathcal{R}$-trivial languages. In each case, tight (or close to tight) bounds on the syntactic complexity taken as a function of the state complexity n of corresponding languages have been determined. For instance, $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial languages, respectively (J. A. Brzozowski, Baiyu Li: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. Int. J. Found. Comput. Sci. 25 (2014) 807–822).

One may think that the $\mathcal{L}$-trivial languages are left-right symmetric to the $\mathcal{R}$-trivial ones, but this is not the case! While the definitions of $\mathcal{L}$ and $\mathcal{R}$ are symmetric, the classes of $\mathcal{L}$-trivial and $\mathcal{L}$-trivial languages are not; in fact, the two classes are very different! Thus, the problem of determining the syntactic complexity of $\mathcal{L}$-trivial languages remains open.

$\mathcal{L}$-Trivial Case

Brzozowski and his collaborators have studied in depth the syntactic complexity of star-free ($\mathcal{H}$-trivial), piecewise testable ($\mathcal{J}$-trivial), and $\mathcal{R}$-trivial languages. In each case, tight (or close to tight) bounds on the syntactic complexity taken as a function of the state complexity n of corresponding languages have been determined. For instance, $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial languages, respectively (J. A. Brzozowski, Baiyu Li: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. Int. J. Found. Comput. Sci. 25 (2014) 807–822).

One may think that the $\mathcal{L}$-trivial languages are left-right symmetric to the $\mathcal{R}$-trivial ones, but this is not the case! While the definitions of $\mathcal{L}$ and $\mathcal{R}$ are symmetric, the classes of $\mathcal{L}$-trivial and $\mathcal{L}$-trivial languages are not; in fact, the two classes are very different! Thus, the problem of determining the syntactic complexity of $\mathcal{L}$-trivial languages remains open.

$\mathcal{L}$-Trivial Case

Brzozowski and his collaborators have studied in depth the syntactic complexity of star-free ($\mathcal{H}$-trivial), piecewise testable ($\mathcal{J}$-trivial), and $\mathcal{R}$-trivial languages. In each case, tight (or close to tight) bounds on the syntactic complexity taken as a function of the state complexity n of corresponding languages have been determined. For instance, $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial languages, respectively (J. A. Brzozowski, Baiyu Li: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. Int. J. Found. Comput. Sci. 25 (2014) 807–822).

One may think that the $\mathcal{L}$-trivial languages are left-right symmetric to the $\mathcal{R}$-trivial ones, but this is not the case! While the definitions of $\mathcal{L}$ and $\mathcal{R}$ are symmetric, the classes of $\mathcal{L}$-trivial and $\mathcal{L}$-trivial languages are not; in fact, the two classes are very different! Thus, the problem of determining the syntactic complexity of $\mathcal{L}$-trivial languages remains open.

$\mathcal{L}$-Trivial Case

Brzozowski and his collaborators have studied in depth the syntactic complexity of star-free ($\mathcal{H}$-trivial), piecewise testable ($\mathcal{J}$-trivial), and $\mathcal{R}$-trivial languages. In each case, tight (or close to tight) bounds on the syntactic complexity taken as a function of the state complexity n of corresponding languages have been determined. For instance, $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial languages, respectively (J. A. Brzozowski, Baiyu Li: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. Int. J. Found. Comput. Sci. 25 (2014) 807–822).

One may think that the $\mathcal{L}$-trivial languages are left-right symmetric to the $\mathcal{R}$-trivial ones, but this is not the case! While the definitions of $\mathcal{L}$ and $\mathcal{R}$ are symmetric, the classes of $\mathcal{L}$-trivial and $\mathcal{L}$-trivial languages are not; in fact, the two classes are very different! Thus, the problem of determining the syntactic complexity of $\mathcal{L}$-trivial languages remains open.

$\mathcal{L}$-Trivial Case

Brzozowski and his collaborators have studied in depth the syntactic complexity of star-free ($\mathcal{H}$-trivial), piecewise testable ($\mathcal{J}$-trivial), and $\mathcal{R}$-trivial languages. In each case, tight (or close to tight) bounds on the syntactic complexity taken as a function of the state complexity n of corresponding languages have been determined. For instance, $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial languages, respectively (J. A. Brzozowski, Baiyu Li: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. Int. J. Found. Comput. Sci. 25 (2014) 807–822).

One may think that the $\mathcal{L}$-trivial languages are left-right symmetric to the $\mathcal{R}$-trivial ones, but this is not the case! While the definitions of $\mathcal{L}$ and $\mathcal{R}$ are symmetric, the classes of $\mathcal{L}$-trivial and $\mathcal{L}$-trivial languages are not; in fact, the two classes are very different! Thus, the problem of determining the syntactic complexity of $\mathcal{L}$-trivial languages remains open.

$\mathcal{L}$-Trivial Case

Brzozowski and his collaborators have studied in depth the syntactic complexity of star-free ($\mathcal{H}$-trivial), piecewise testable ($\mathcal{J}$-trivial), and $\mathcal{R}$-trivial languages. In each case, tight (or close to tight) bounds on the syntactic complexity taken as a function of the state complexity n of corresponding languages have been determined. For instance, $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial languages, respectively (J. A. Brzozowski, Baiyu Li: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. Int. J. Found. Comput. Sci. 25 (2014) 807–822).

One may think that the $\mathcal{L}$-trivial languages are left-right symmetric to the $\mathcal{R}$-trivial ones, but this is not the case! While the definitions of $\mathcal{L}$ and $\mathcal{R}$ are symmetric, the classes of $\mathcal{L}$-trivial and $\mathcal{L}$-trivial languages are not; in fact, the two classes are very different! Thus, the problem of determining the syntactic complexity of $\mathcal{L}$-trivial languages remains open.

Connections to Completely Reachable Automata

It is well known (and easy) that if two transformations from the transition monoid of a DFA are $\mathcal{L}$-related, they must have the same image. Hence, if a DFA $\mathscr{A}$ is such that different transformations from the monoid $M(\mathscr{A})$ have different images, then the monoid must be $\mathcal{L}$-trivial. How big can be the monoid with the latter property? Can the transition monoid of a completely reachable automaton have this property?

These questions are far from being obvious and are of interest also from the viewpoint of algebra, where the latter question has been stated in the literature as the question about the existence of $\mathcal{L}$-cross-sections in the full transformation monoid, see O. Ganyushkin, V. Mazorchuk, Classical Finite Transformation Semigroups: An Introduction. Springer, 2009.

Connections to Completely Reachable Automata

It is well known (and easy) that if two transformations from the transition monoid of a DFA are $\mathcal{L}$-related, they must have the same image. Hence, if a DFA $\mathscr{A}$ is such that different transformations from the monoid $M(\mathscr{A})$ have different images, then the monoid must be $\mathcal{L}$-trivial. How big can be the monoid with the latter property? Can the transition monoid of a completely reachable automaton have this property?

These questions are far from being obvious and are of interest also from the viewpoint of algebra, where the latter question has been stated in the literature as the question about the existence of $\mathcal{L}$-cross-sections in the full transformation monoid, see O. Ganyushkin, V. Mazorchuk, Classical Finite Transformation Semigroups: An Introduction. Springer, 2009.

Connections to Completely Reachable Automata

It is well known (and easy) that if two transformations from the transition monoid of a DFA are $\mathcal{L}$-related, they must have the same image. Hence, if a DFA $\mathscr{A}$ is such that different transformations from the monoid $M(\mathscr{A})$ have different images, then the monoid must be $\mathcal{L}$-trivial. How big can be the monoid with the latter property? Can the transition monoid of a completely reachable automaton have this property?

These questions are far from being obvious and are of interest also from the viewpoint of algebra, where the latter question has been stated in the literature as the question about the existence of $\mathcal{L}$-cross-sections in the full transformation monoid, see O. Ganyushkin, V. Mazorchuk, Classical Finite Transformation Semigroups: An Introduction. Springer, 2009.

Connections to Completely Reachable Automata

It is well known (and easy) that if two transformations from the transition monoid of a DFA are $\mathcal{L}$-related, they must have the same image. Hence, if a DFA $\mathscr{A}$ is such that different transformations from the monoid $M(\mathscr{A})$ have different images, then the monoid must be $\mathcal{L}$-trivial. How big can be the monoid with the latter property? Can the transition monoid of a completely reachable automaton have this property?

These questions are far from being obvious and are of interest also from the viewpoint of algebra, where the latter question has been stated in the literature as the question about the existence of $\mathcal{L}$-cross-sections in the full transformation monoid, see O. Ganyushkin, V. Mazorchuk, Classical Finite Transformation Semigroups: An Introduction. Springer, 2009.

Connections to Completely Reachable Automata

It is well known (and easy) that if two transformations from the transition monoid of a DFA are $\mathcal{L}$-related, they must have the same image. Hence, if a DFA $\mathscr{A}$ is such that different transformations from the monoid $M(\mathscr{A})$ have different images, then the monoid must be $\mathcal{L}$-trivial. How big can be the monoid with the latter property? Can the transition monoid of a completely reachable automaton have this property?

These questions are far from being obvious and are of interest also from the viewpoint of algebra, where the latter question has been stated in the literature as the question about the existence of $\mathcal{L}$-cross-sections in the full transformation monoid, see O. Ganyushkin, V. Mazorchuk, Classical Finite Transformation Semigroups: An Introduction. Springer, 2009.

Minimal Completely Reachable Automata

For every $n \geq 1$, we construct all completely reachable automata $\mathscr{A}$ with n states such that different transformations from $M(\mathscr{A})$ have different images. Clearly, the size of $M(\mathscr{A})$ is then $2^n - 1$, and this is the minimum possible syntactic complexity of a completely reachable automaton with n states. Therefore, we refer to these automata as to minimal completely reachable automata.

From the viewpoint of the syntactic complexity of $\mathcal{L}$-trivial languages, our result shows that there are $\mathcal{L}$-trivial languages with state complexity n whose syntactic complexity is $2^n - 1$. We do not know whether or not this bound is tight.

Minimal Completely Reachable Automata

For every $n \geq 1$, we construct all completely reachable automata $\mathscr{A}$ with n states such that different transformations from $M(\mathscr{A})$ have different images. Clearly, the size of $M(\mathscr{A})$ is then $2^n - 1$, and this is the minimum possible syntactic complexity of a completely reachable automaton with n states. Therefore, we refer to these automata as to minimal completely reachable automata.

From the viewpoint of the syntactic complexity of $\mathcal{L}$-trivial languages, our result shows that there are $\mathcal{L}$-trivial languages with state complexity n whose syntactic complexity is $2^n - 1$. We do not know whether or not this bound is tight.

Minimal Completely Reachable Automata

For every $n \geq 1$, we construct all completely reachable automata $\mathscr{A}$ with n states such that different transformations from $M(\mathscr{A})$ have different images. Clearly, the size of $M(\mathscr{A})$ is then $2^n - 1$, and this is the minimum possible syntactic complexity of a completely reachable automaton with n states. Therefore, we refer to these automata as to minimal completely reachable automata.

From the viewpoint of the syntactic complexity of $\mathcal{L}$-trivial languages, our result shows that there are $\mathcal{L}$-trivial languages with state complexity n whose syntactic complexity is $2^n - 1$. We do not know whether or not this bound is tight.

Minimal Completely Reachable Automata

For every $n \geq 1$, we construct all completely reachable automata $\mathscr{A}$ with n states such that different transformations from $M(\mathscr{A})$ have different images. Clearly, the size of $M(\mathscr{A})$ is then $2^n - 1$, and this is the minimum possible syntactic complexity of a completely reachable automaton with n states. Therefore, we refer to these automata as to minimal completely reachable automata.

From the viewpoint of the syntactic complexity of $\mathcal{L}$-trivial languages, our result shows that there are $\mathcal{L}$-trivial languages with state complexity n whose syntactic complexity is $2^n - 1$. We do not know whether or not this bound is tight.

Minimal Completely Reachable Automata

For every $n \geq 1$, we construct all completely reachable automata $\mathscr{A}$ with n states such that different transformations from $M(\mathscr{A})$ have different images. Clearly, the size of $M(\mathscr{A})$ is then $2^n - 1$, and this is the minimum possible syntactic complexity of a completely reachable automaton with n states. Therefore, we refer to these automata as to minimal completely reachable automata.

From the viewpoint of the syntactic complexity of $\mathcal{L}$-trivial languages, our result shows that there are $\mathcal{L}$-trivial languages with state complexity n whose syntactic complexity is $2^n - 1$. We do not know whether or not this bound is tight.

Trees

Our construction produces minimal completely reachable automata from full binary trees satisfying certain subordination conditions.

Trees

Our construction produces minimal completely reachable automata from full binary trees satisfying certain subordination conditions.
A binary tree is full if each its vertex v either is a leaf or has exactly two children.

Trees

Our construction produces minimal completely reachable automata from full binary trees satisfying certain subordination conditions. A binary tree is full if each its vertex v either is a leaf or has exactly two children. We refer to the left/right child of v as to the son/daughter of v.

Trees

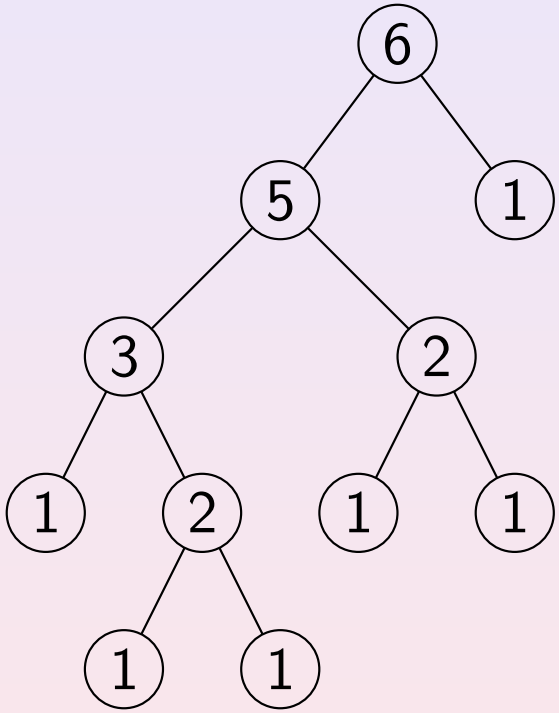
Our construction produces minimal completely reachable automata from full binary trees satisfying certain subordination conditions.
A binary tree is full if each its vertex v either is a leaf or has exactly two children. We refer to the left/right child of v as to the son/daughter of v.
If Γ is a tree and v is its vertex, Γ_v is the subtree of Γ rooted at v.

Trees

Our construction produces minimal completely reachable automata from full binary trees satisfying certain subordination conditions.
A binary tree is full if each its vertex v either is a leaf or has exactly two children. We refer to the left/right child of v as to the son/daughter of v.
If Γ is a tree and v is its vertex, Γ_v is the subtree of Γ rooted at v.
The span of v is the number of leaves in Γ_v.

Trees

This is a tree with vertices labelled by their spans:



Respectful Trees

A homomorphism between trees is a map between their vertex sets that preserves the roots, the parent–child relation and the genders of non-root vertices.

Respectful Trees

A homomorphism between trees is a map between their vertex sets that preserves the roots, the parent–child relation and the genders of non-root vertices. If u and v are vertices of a tree Γ, we say that u subordinates v if there is a 1-1 homomorphism $\Gamma_u \to \Gamma_v$.

Respectful Trees

A homomorphism between trees is a map between their vertex sets that preserves the roots, the parent–child relation and the genders of non-root vertices. If u and v are vertices of a tree Γ, we say that u subordinates v if there is a 1-1 homomorphism $\Gamma_u \to \Gamma_v$.
A respectful tree is such that:

(S1) if a male vertex has a nephew, he subordinates his uncle;

(S2) if a female vertex has a niece, she subordinates her aunt.

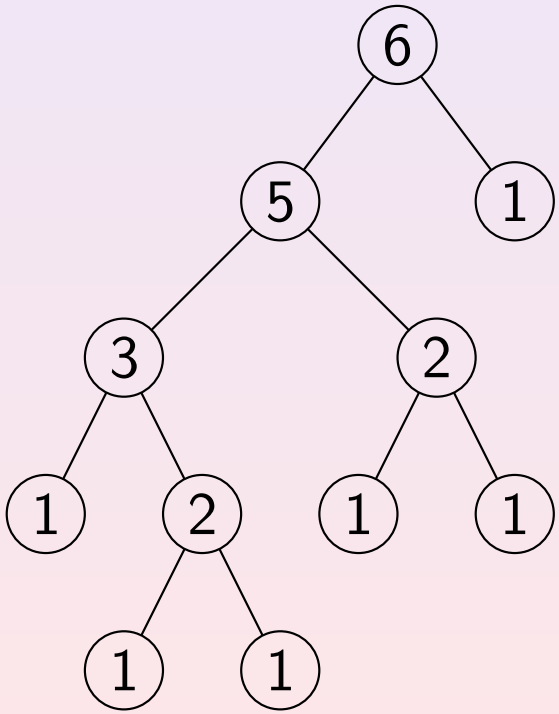
Respectful Trees

A respectful tree is such that:

(S1) if a male vertex has a nephew, he subordinates his uncle;

(S2) if a female vertex has a niece, she subordinates her aunt.

This tree is not respectful as it fails to satisfy (S2):



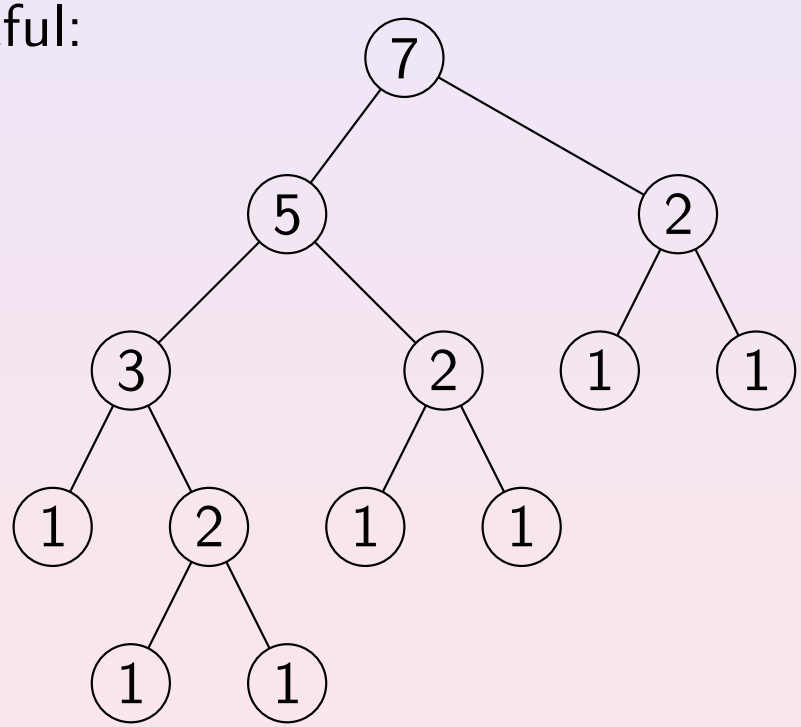
Respectful Trees

A respectful tree is such that:

(S1) if a male vertex has a nephew, he subordinates his uncle;

(S2) if a female vertex has a niece, she subordinates her aunt.

This tree is respectful:



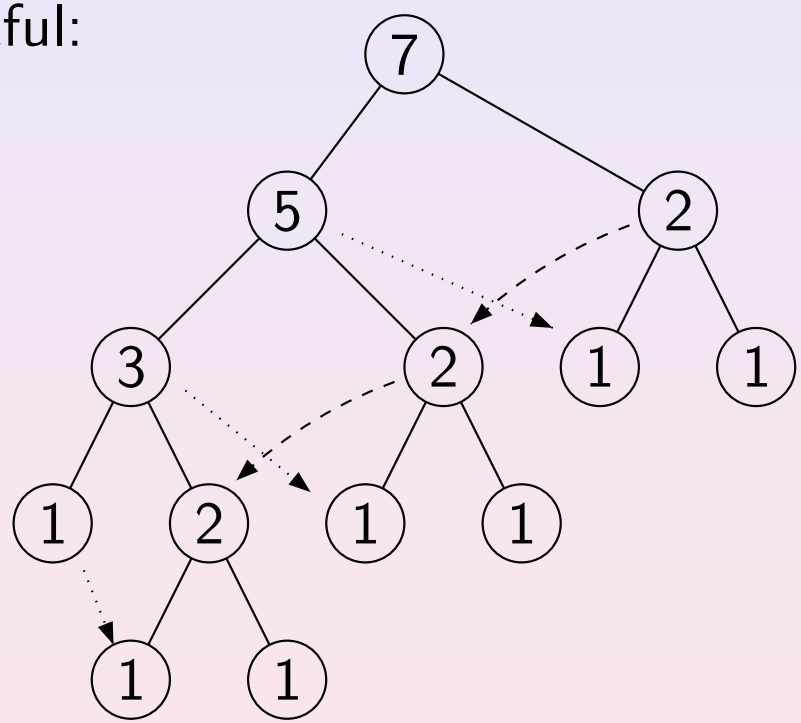
Respectful Trees

A respectful tree is such that:

(S1) if a male vertex has a nephew, he subordinates his uncle;

(S2) if a female vertex has a niece, she subordinates her aunt.

This tree is respectful:



Here dotted and dashed arrows show the uncle–nephew and the aunt–niece relations respectively.

Interval Markings

It is easy to show that there exist respectful trees with any number of leaves.

Interval Markings

It is easy to show that there exist respectful trees with any number of leaves. The number of respectful trees quickly grows with the number of leaves, but we are not aware of any closed formula for the former number nor of its growth rate.

Interval Markings

We use certain markings of trees by intervals of the set $\mathbb{N}$ of positive integers.

Interval Markings

We use certain markings of trees by intervals of the set $\mathbb{N}$ of positive integers. A faithful interval marking of a tree is a map μ from its vertex set into the set of all intervals in $\mathbb{N}$ such that for each vertex v,

- the size of the interval $v\mu$ is equal to the span of v;
- if $v\mu = [i, j]$ and s and d are the son and the daughter of v respectively, then $s\mu = [i, k]$ and $d\mu = [k + 1, j]$ for some k such that $i \leq k < j$.

Interval Markings

We use certain markings of trees by intervals of the set $\mathbb{N}$ of positive integers. A faithful interval marking of a tree is a map μ from its vertex set into the set of all intervals in $\mathbb{N}$ such that for each vertex v,

- the size of the interval $v\mu$ is equal to the span of v;
- if $v\mu = [i, j]$ and s and d are the son and the daughter of v respectively, then $s\mu = [i, k]$ and $d\mu = [k + 1, j]$ for some k such that $i \leq k < j$.

Every tree Γ admits a faithful interval marking which is unique up to an additive translation: given any two markings μ, μ' of Γ, there is an integer m such that $v\mu = v\mu' + m$ for every vertex v.

Interval Markings

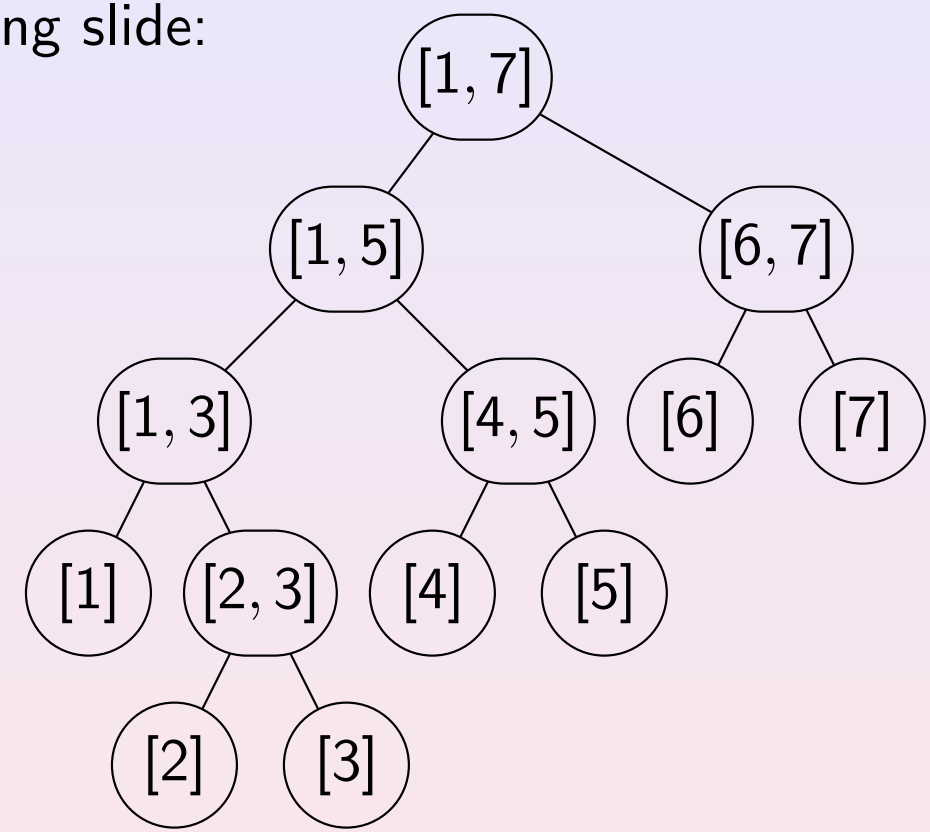
We use certain markings of trees by intervals of the set $\mathbb{N}$ of positive integers. A faithful interval marking of a tree is a map μ from its vertex set into the set of all intervals in $\mathbb{N}$ such that for each vertex v,

- the size of the interval $v\mu$ is equal to the span of v;
- if $v\mu = [i, j]$ and s and d are the son and the daughter of v respectively, then $s\mu = [i, k]$ and $d\mu = [k + 1, j]$ for some k such that $i \leq k < j$.

Every tree Γ admits a faithful interval marking which is unique up to an additive translation: given any two markings μ, μ' of Γ, there is an integer m such that $v\mu = v\mu' + m$ for every vertex v.
If μ is a faithful interval marking of Γ and v is a vertex of Γ, then the restriction of μ to the subtree Γ_v is a faithful interval marking of the latter.

Interval Markings

Here is a faithful interval marking of the respectful tree from the preceding slide:



Here we write $[i]$ instead of $[i, i]$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.

The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.

$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

We define the action of the letters by induction on n.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

We define the action of the letters by induction on n. For $n=1$, Γ is the trivial tree with one vertex r and no edges and $\mathscr{A}_\mu(\Gamma)$ is the trivial automaton with one state and no transitions.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

Now suppose that $n > 1$ and take any non-root vertex v of Γ; we have to define the action of the letter a_v on of the interval $r\mu$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

Now suppose that $n > 1$ and take any non-root vertex v of Γ; we have to define the action of the letter a_v on of the interval $r\mu$. If s and d are respectively the son and the daughter of r, the interval $r\mu$ is the disjoint union of $s\mu$ and $d\mu$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

Now suppose that $n > 1$ and take any non-root vertex v of Γ; we have to define the action of the letter a_v on of the interval $r\mu$. If s and d are respectively the son and the daughter of r, the interval $r\mu$ is the disjoint union of $s\mu$ and $d\mu$. If $v \neq s$ and $v \neq d$, then v is a non-root vertex in one of the subtrees Γ_s or Γ_d; WLOG assume that v belongs to Γ_s.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

Now suppose that $n > 1$ and take any non-root vertex v of Γ; we have to define the action of the letter a_v on of the interval $r\mu$. If s and d are respectively the son and the daughter of r, the interval $r\mu$ is the disjoint union of $s\mu$ and $d\mu$. If $v \neq s$ and $v \neq d$, then v is a non-root vertex in one of the subtrees Γ_s or Γ_d; WLOG assume that v belongs to Γ_s. By the induction assumption applied to Γ_s and its marking induced by μ, the action of a_v on the interval $s\mu$ is already defined; we extend this action to the whole interval $r\mu$ by setting $y \,.\, a_v := y$ for each $y \in d\mu$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$. Otherwise let t be the nephew of s.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$. Otherwise let t be the nephew of s. By (S1) there is a 1-1 homomorphism $\xi\colon \Gamma_t \to \Gamma_s$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$. Otherwise let t be the nephew of s. By (S1) there is a 1-1 homomorphism $\xi\colon \Gamma_t \to \Gamma_s$. The intervals $(\ell\xi)\mu$, where ℓ runs over the set of all leaves of Γ_t, partition the interval $s\mu$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$. Otherwise let t be the nephew of s. By (S1) there is a 1-1 homomorphism $\xi\colon \Gamma_t \to \Gamma_s$. The intervals $(\ell\xi)\mu$, where ℓ runs over the set of all leaves of Γ_t, partition the interval $s\mu$. Now if a number $x \in s\mu$ belongs to $(\ell\xi)\mu$ for some leaf ℓ of Γ_t and $\ell\mu = [y]$ for some $y \in \mathbb{N}$, we let $x \,.\, a_s := y$.

From Trees to Automata

For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

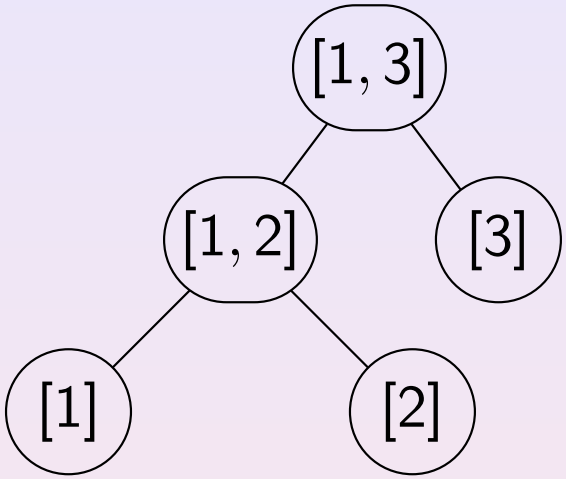
It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$. Otherwise let t be the nephew of s. By (S1) there is a 1-1 homomorphism $\xi\colon \Gamma_t \to \Gamma_s$. The intervals $(\ell\xi)\mu$, where ℓ runs over the set of all leaves of Γ_t, partition the interval $s\mu$. Now if a number $x \in s\mu$ belongs to $(\ell\xi)\mu$ for some leaf ℓ of Γ_t and $\ell\mu = [y]$ for some $y \in \mathbb{N}$, we let $x \,.\, a_s := y$. By the induction assumption applied to Γ_d and its marking induced by μ, the action of a_t on the interval $d\mu$ is already defined and we extend the action of a_s to $d\mu$ by setting $y \,.\, a_s := y \,.\, a_t$ for all $y \in d\mu$.

From Trees to Automata

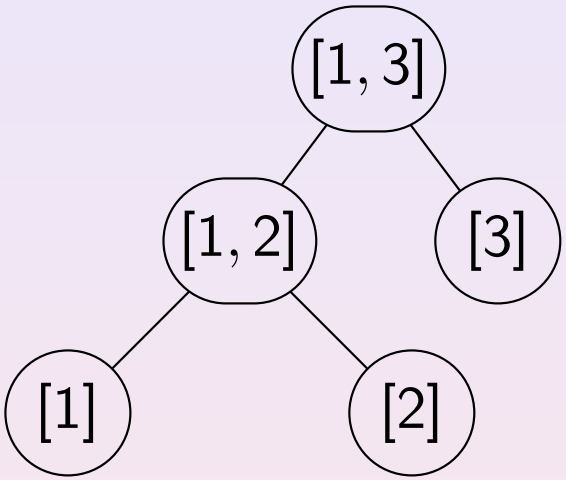
For each respectful tree Γ with n leaves and each its faithful interval marking μ, we construct an automaton $\mathscr{A}_\mu(\Gamma)$.
The state set of $\mathscr{A}_\mu(\Gamma)$ is the interval $r\mu$, where r is the root of Γ.
$\mathscr{A}_\mu(\Gamma)$ has $2n-2$ input letters a_v, one for each non-root vertex v.

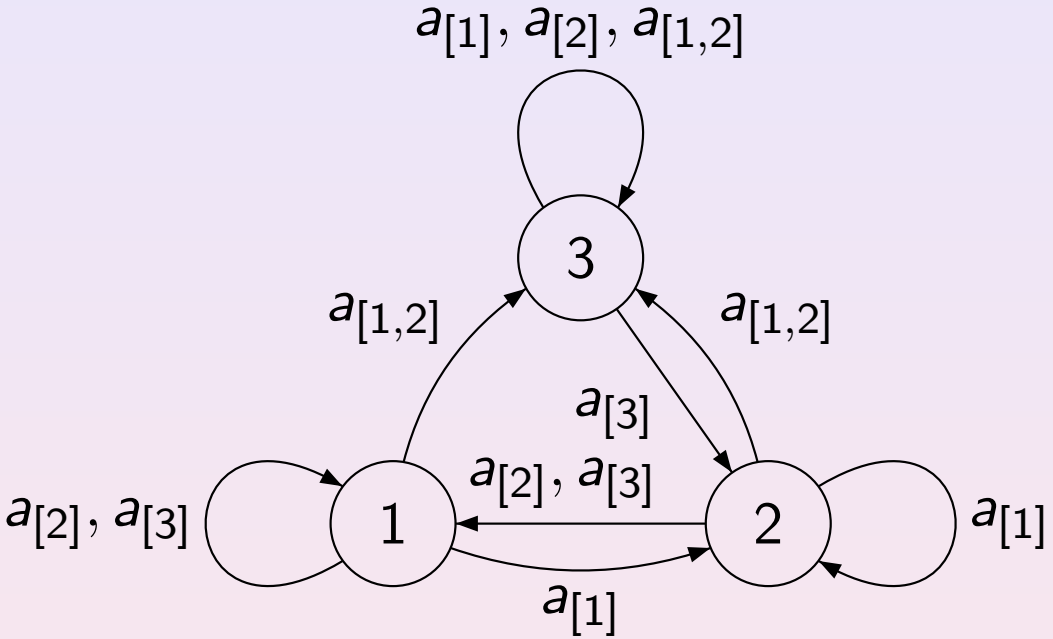
It remains to define the actions of a_s and a_d. By symmetry, it suffices to define the action of a_s. If s has no nephew in Γ, then d is a leaf and $d\mu = [y]$ for some $y \in \mathbb{N}$. Then let $x \,.\, a_s := y$ for each $x \in r\mu$. Otherwise let t be the nephew of s. By (S1) there is a 1-1 homomorphism $\xi\colon \Gamma_t \to \Gamma_s$. The intervals $(\ell\xi)\mu$, where ℓ runs over the set of all leaves of Γ_t, partition the interval $s\mu$. Now if a number $x \in s\mu$ belongs to $(\ell\xi)\mu$ for some leaf ℓ of Γ_t and $\ell\mu = [y]$ for some $y \in \mathbb{N}$, we let $x \,.\, a_s := y$. By the induction assumption applied to Γ_d and its marking induced by μ, the action of a_t on the interval $d\mu$ is already defined and we extend the action of a_s to $d\mu$ by setting $y \,.\, a_s := y \,.\, a_t$ for all $y \in d\mu$.
This completes our construction.

An Example



An Example





All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$. We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$.

We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$. We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$. We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$. We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$. We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

All Minimal Completely Reachable Automata

Automata constructed from different markings of the same respectful tree are isomorphic. Therefore we denote the DFA derived from any marking of a respectful tree Γ simply by $\mathscr{A}(\Gamma)$. We say that two DFAs $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ are syntactically equivalent if their transition monoids coincide.

1. For each respectful tree Γ, the automaton $\mathscr{A}(\Gamma)$ is a minimal completely reachable automaton.
2. Non-isomorphic respectful trees give rise to non-isomorphic automata.
3. Every minimal completely reachable automaton is syntactically equivalent to an automaton of the form $\mathscr{A}(\Gamma)$ for a suitable respectful tree Γ.
4. Every minimal completely reachable automaton with n states has at least $2n - 2$ input letters.

Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.

Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.

Three Open Questions

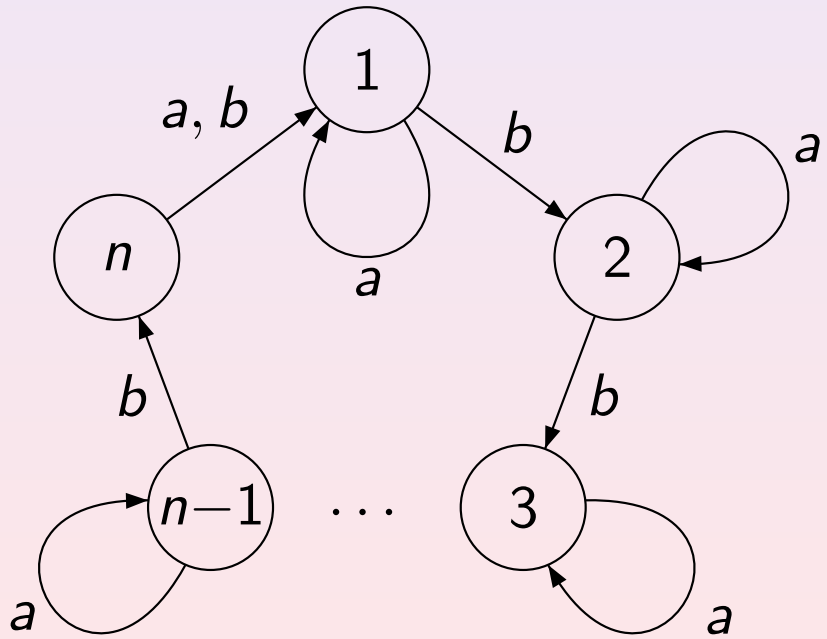
What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.

Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

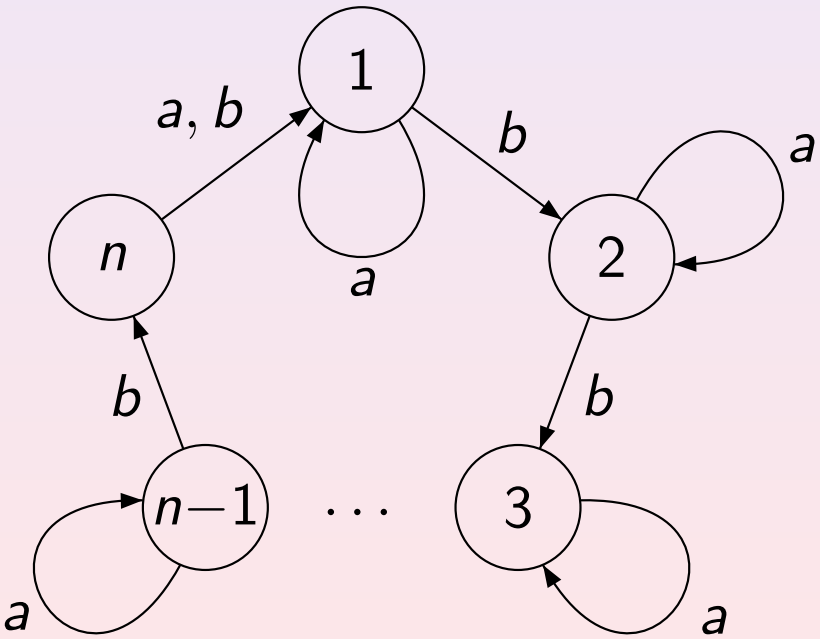
A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.

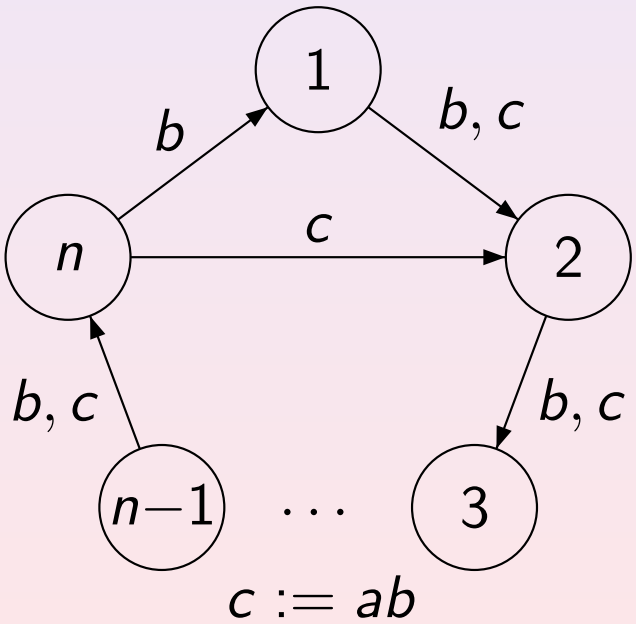


Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.





Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.
Is it true that every completely reachable automaton induces a minimal completely reachable automaton?

Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.
Is it true that every completely reachable automaton induces a minimal completely reachable automaton? In other words, is it true that an automaton of the form $\mathscr{A}(\Gamma)$ 'hides' within every completely reachable automaton?

Three Open Questions

What are lower bounds for the syntactic complexity of completely reachable automata with restricted alphabet?

A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ induces a DFA $\mathscr{B} = \langle Q, \Delta, \zeta \rangle$ on the same state set if the transition monoid of $\mathscr{A}$ contains that of $\mathscr{B}$. This means that for every letter $b \in \Delta$, there exists a word $w \in \Sigma^*$ such that $\zeta(q, b) = \delta(q, w)$ for every $q \in Q$.
Is it true that every completely reachable automaton induces a minimal completely reachable automaton? In other words, is it true that an automaton of the form $\mathscr{A}(\Gamma)$ 'hides' within every completely reachable automaton?

What is the computational complexity of recognizing complete reachability?