Synchronizing Automata – I

M. V. Volkov

Ural State University, Ekaterinburg, Russia

Synchronizing automata

We consider DFA: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

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

Synchronizing automata

We consider DFA: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

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

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

Synchronizing automata

We consider DFA: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

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

$\mathscr{A}$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.
$|Q \,.\, w| = 1$. Here $Q \,.\, v = \{\delta(q, v) \mid q \in Q\}$.

Synchronizing automata

We consider DFA: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

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

$\mathscr{A}$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.
$|Q \,.\, w| = 1$. Here $Q \,.\, v = \{\delta(q, v) \mid q \in Q\}$.

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

Synchronizing automata

We consider DFA: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

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

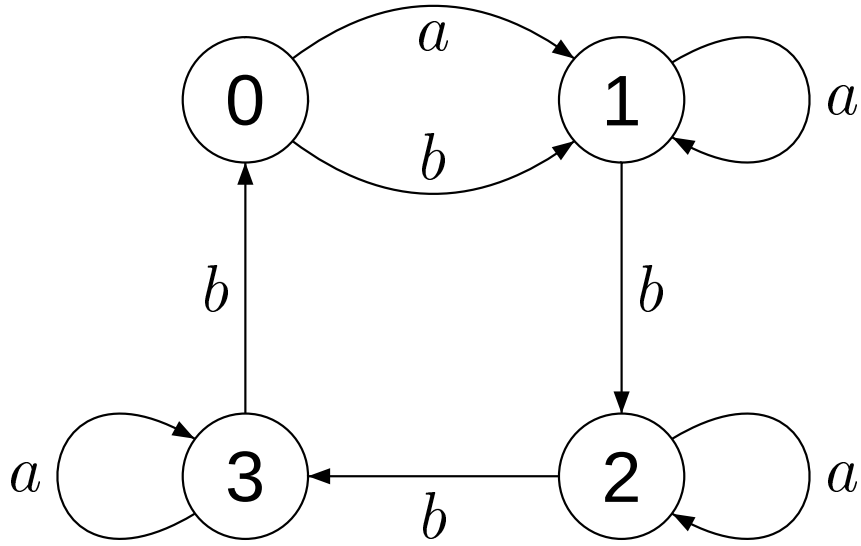
$\mathscr{A}$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.
$|Q \,.\, w| = 1$. Here $Q \,.\, v = \{\delta(q, v) \mid q \in Q\}$.

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

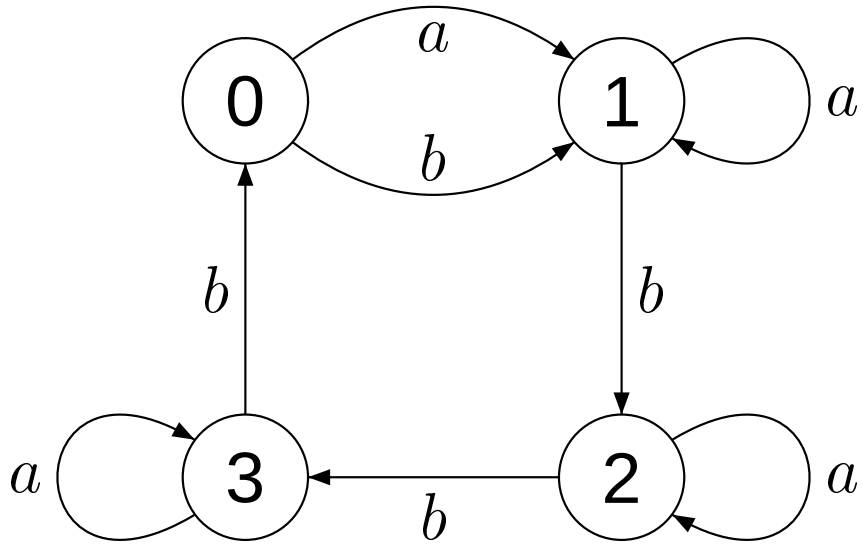
Other names:

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

Synchronizing automata



Synchronizing automata



A reset word is $abbbabbba$. Applying it at any state brings the automaton to the state 1.

Synchronizing Automata

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

Synchronizing Automata

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

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

Synchronizing Automata

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

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

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

Synchronizing Automata

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

Synchronizing Automata

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

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

Synchronizing Automata

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

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

Synchronizing Automata

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

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

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

Re-inventing by Engineers

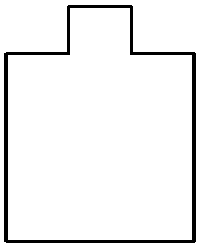
Since the 60s and till the 90s synchronizing automata were considered as a useful tool for testing of reactive systems (first circuits, later protocols).

Re-inventing by Engineers

Since the 60s and till the 90s synchronizing automata were considered as a useful tool for testing of reactive systems (first circuits, later protocols).
In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.

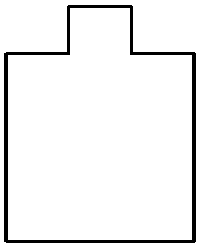
Re-inventing by Engineers

Since the 60s and till the 90s synchronizing automata were considered as a useful tool for testing of reactive systems (first circuits, later protocols).
In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.
Suppose that one of the parts of a certain device has the following shape:



Re-inventing by Engineers

Since the 60s and till the 90s synchronizing automata were considered as a useful tool for testing of reactive systems (first circuits, later protocols).
In the 80s, the notion was reinvented by engineers working in a branch of robotics which deals with part handling problems in industrial automation.
Suppose that one of the parts of a certain device has the following shape:



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

Re-inventing by Engineers

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

Re-inventing by Engineers

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

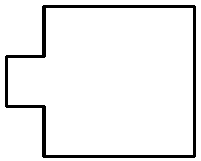
Suppose that prior the assembly the part should take the “bump-left” orientation (the second one in the picture). Thus, one has to construct an orienter which action will put the part in the prescribed position independently of its initial orientation.

Re-inventing by Engineers

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

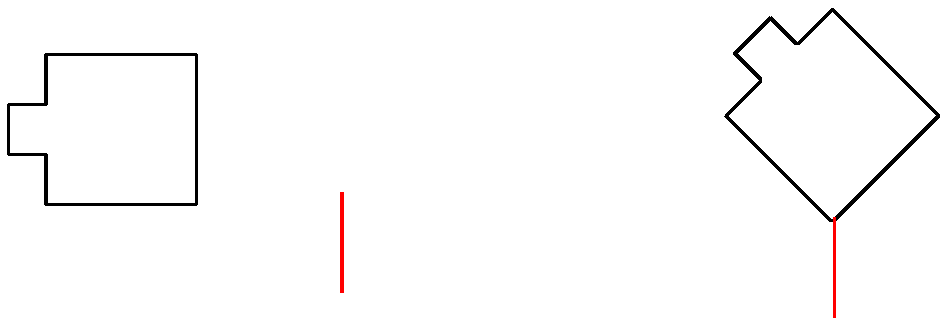
Re-inventing by Engineers

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



Re-inventing by Engineers

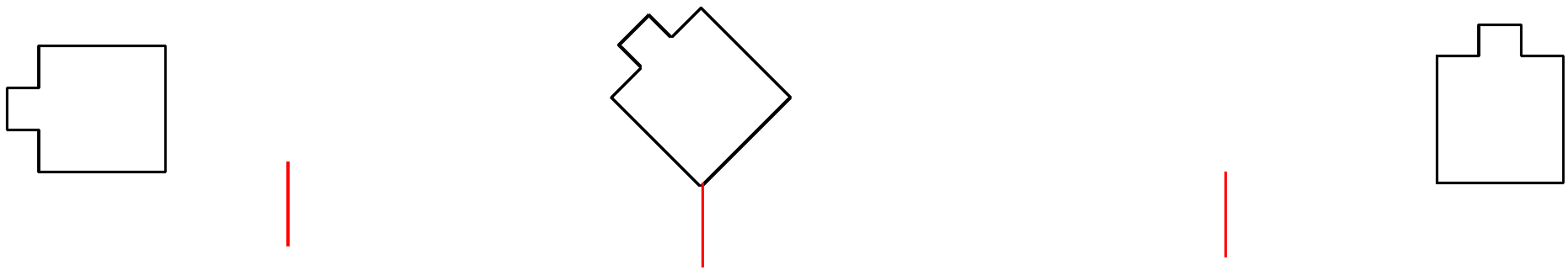
We put parts to be oriented on a conveyer belt which takes them to the assembly point and let the stream of the parts encounter a series of passive obstacles of two types (*high* and *low*) placed along the belt.
A high obstacle is high enough so that any part on the belt encounters this obstacle by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



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

Re-inventing by Engineers

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



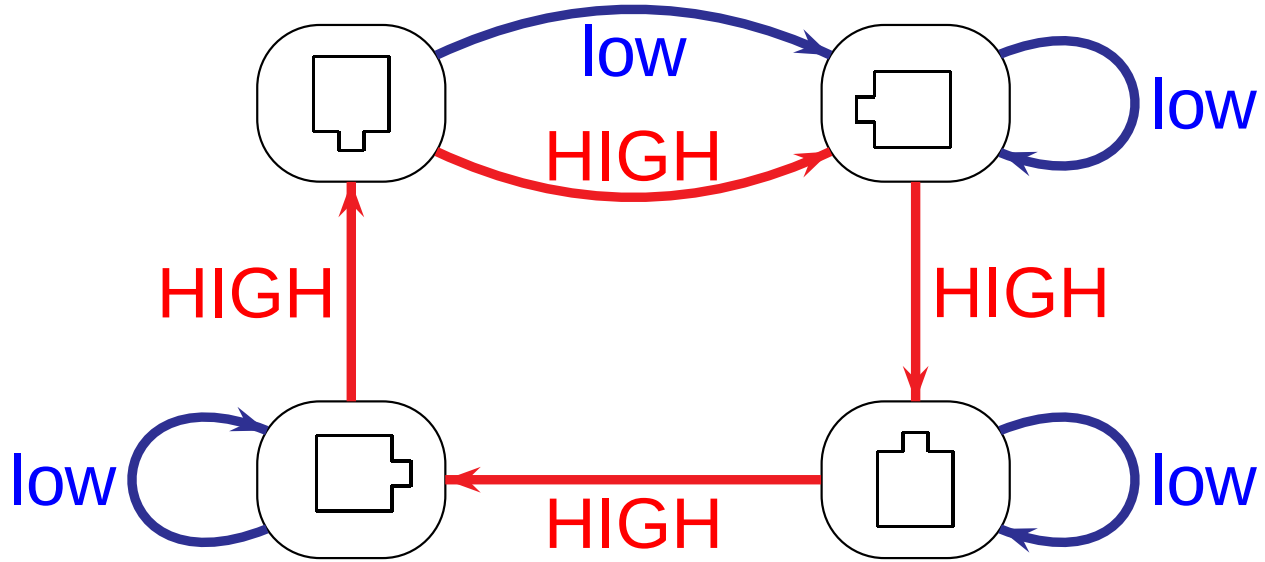
Being curried by the belt, the part then is forced to turn 90° clockwise.

Re-inventing by Engineers

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

Re-inventing by Engineers

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

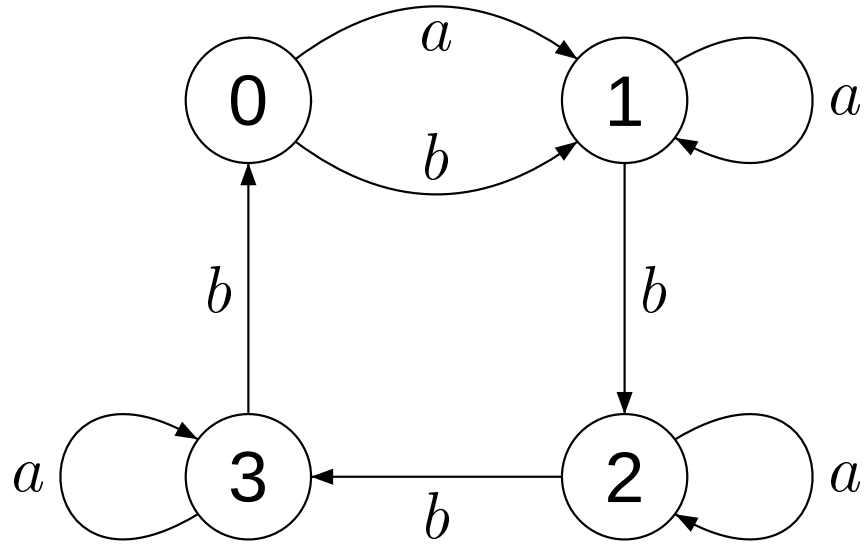


Re-inventing by Engineers

We met this picture a few slides ago:

Re-inventing by Engineers

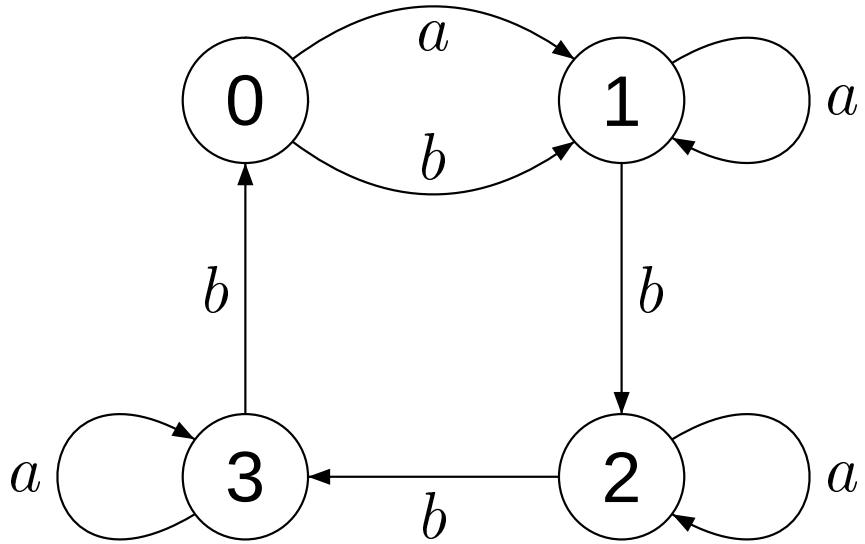
We met this picture a few slides ago:



– this was our example of a synchronizing automaton, and we saw that $abbbabbba$ is a reset sequence of actions.

Re-inventing by Engineers

We met this picture a few slides ago:



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

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

yields the desired sensorless orienter.

Possible Use in Biocomputing

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

Possible Use in Biocomputing

In DNA-computing, there is a fast progressing work by Ehud Shapiro's group on "*soup of automata*" (Programmable and autonomous computing machine made of biomolecules, Nature 414, no.1 (November 22, 2001) 430–434; DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003) 2191–2196, etc). They have produced a solution containing 3×10^{12} identical DNA-based automata per μl.

Possible Use in Biocomputing

In DNA-computing, there is a fast progressing work by Ehud Shapiro's group on "*soup of automata*" (Programmable and autonomous computing machine made of biomolecules, Nature 414, no.1 (November 22, 2001) 430–434; DNA molecule provides a computing machine with both data and fuel, Proc. National Acad. Sci. USA 100 (2003) 2191–2196, etc). They have produced a solution containing 3×10^{12} identical DNA-based automata per μl. These automata can work in parallel on different inputs (DNA strands), thus ending up in different and unpredictable states.

Possible Use in Biocomputing

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

Outline of the Tutorial

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance. We discuss them in Lecture I.

Outline of the Tutorial

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance. We discuss them in Lecture I.
- Synchronizing automata constitute an interesting combinatorial object. Their studies are mainly motivated by the Černý conjecture. We discuss the Černý conjecture in Lecture II.

Outline of the Tutorial

- From the viewpoint of applications, real or yet imaginary, algorithmic issues are of crucial importance. We discuss them in Lecture I.
- Synchronizing automata constitute an interesting combinatorial object. Their studies are mainly motivated by the Černý conjecture. We discuss the Černý conjecture in Lecture II.
- Yet another mathematical motivation for studying synchronizing automata comes from symbolic dynamics. In Lecture III we present a recent breakthrough in the area—a (positive) solution to the Road Coloring Problem found by Trahtman.

Checking Synchronizability

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?*

Checking Synchronizability

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

Checking Synchronizability

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

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

- states are the non-empty subsets of Q,
- $\delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

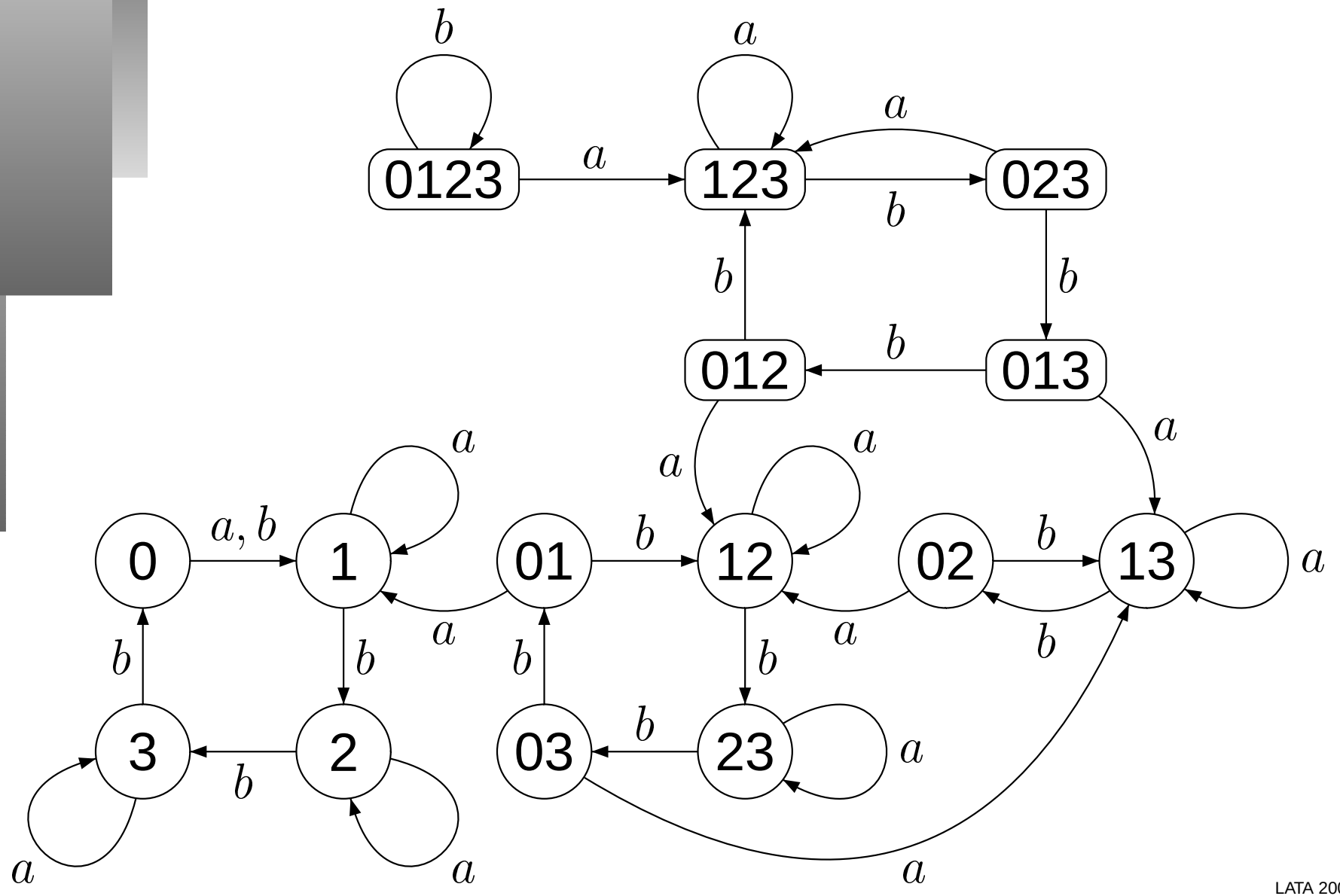
Checking Synchronizability

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

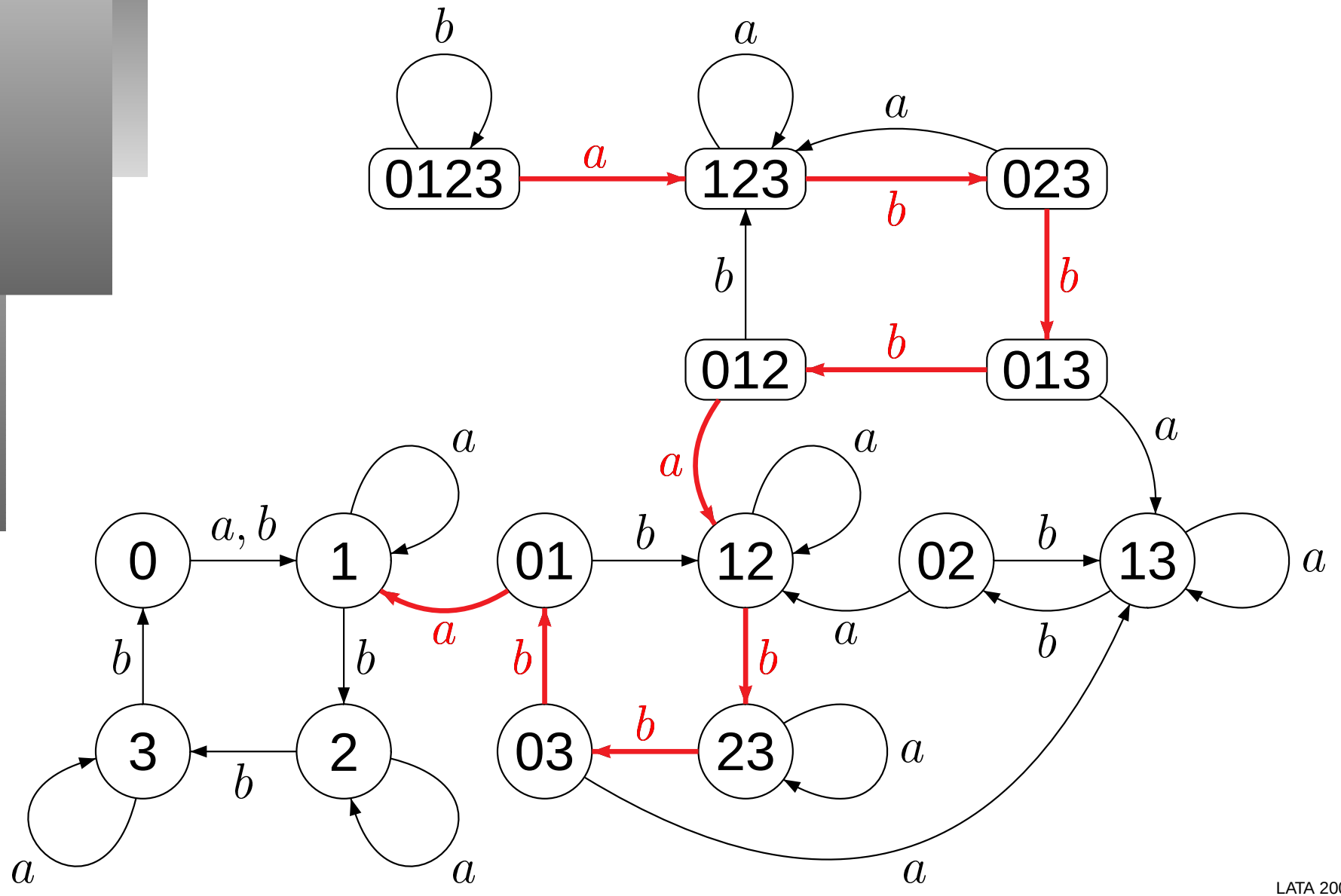
The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

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

Checking Synchronizability



Checking Synchronizability



Checking Synchronizability

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

Checking Synchronizability

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

Checking Synchronizability

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

The following result by Černý gives a polynomial algorithm:

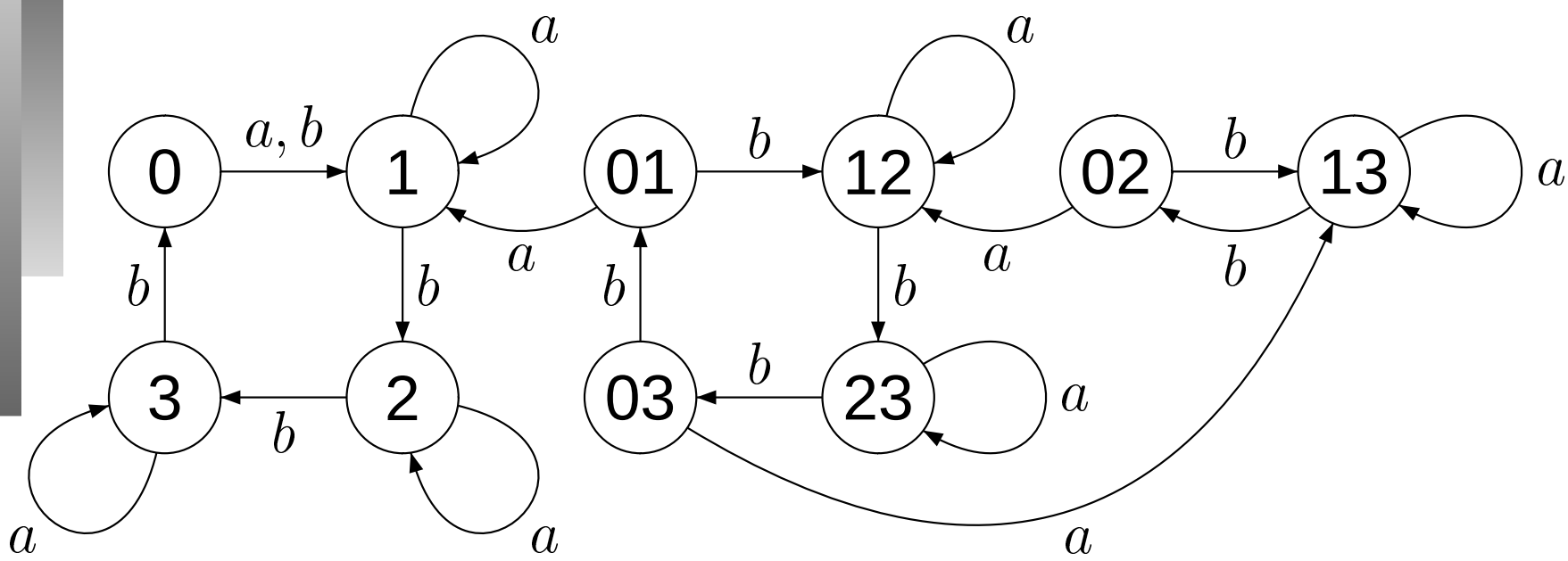
Checking Synchronizability

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

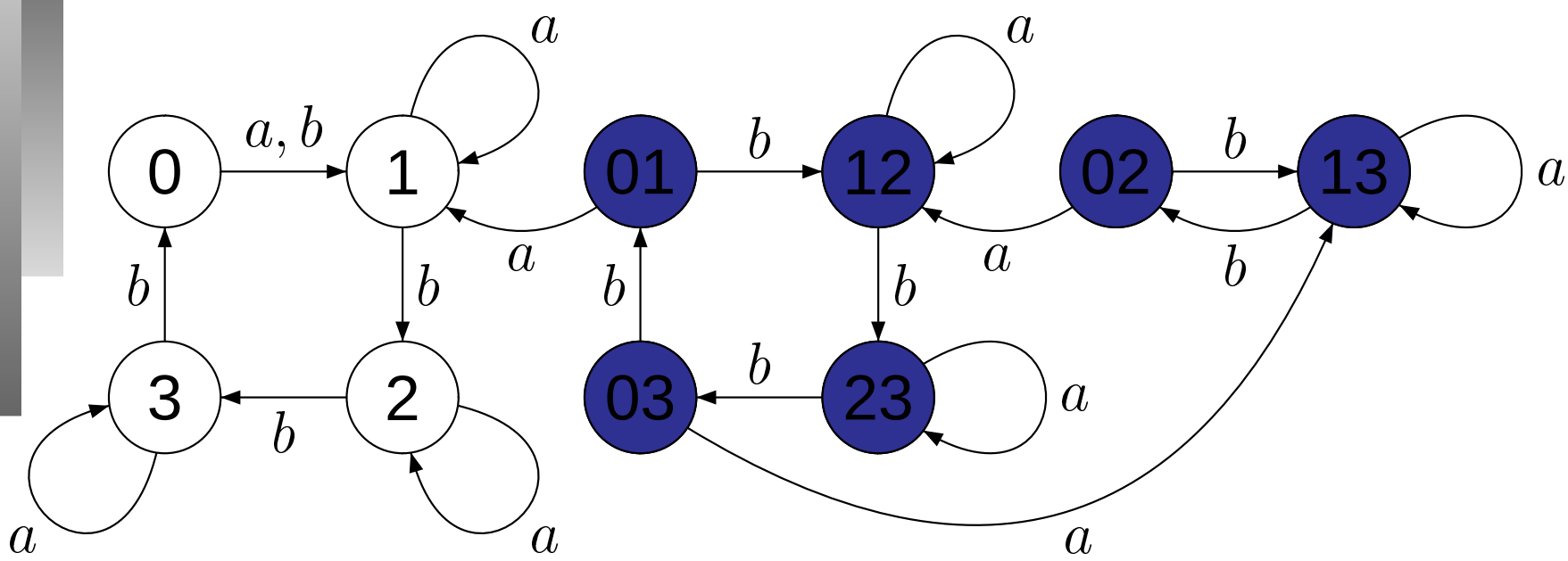
The following result by Černý gives a polynomial algorithm:

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

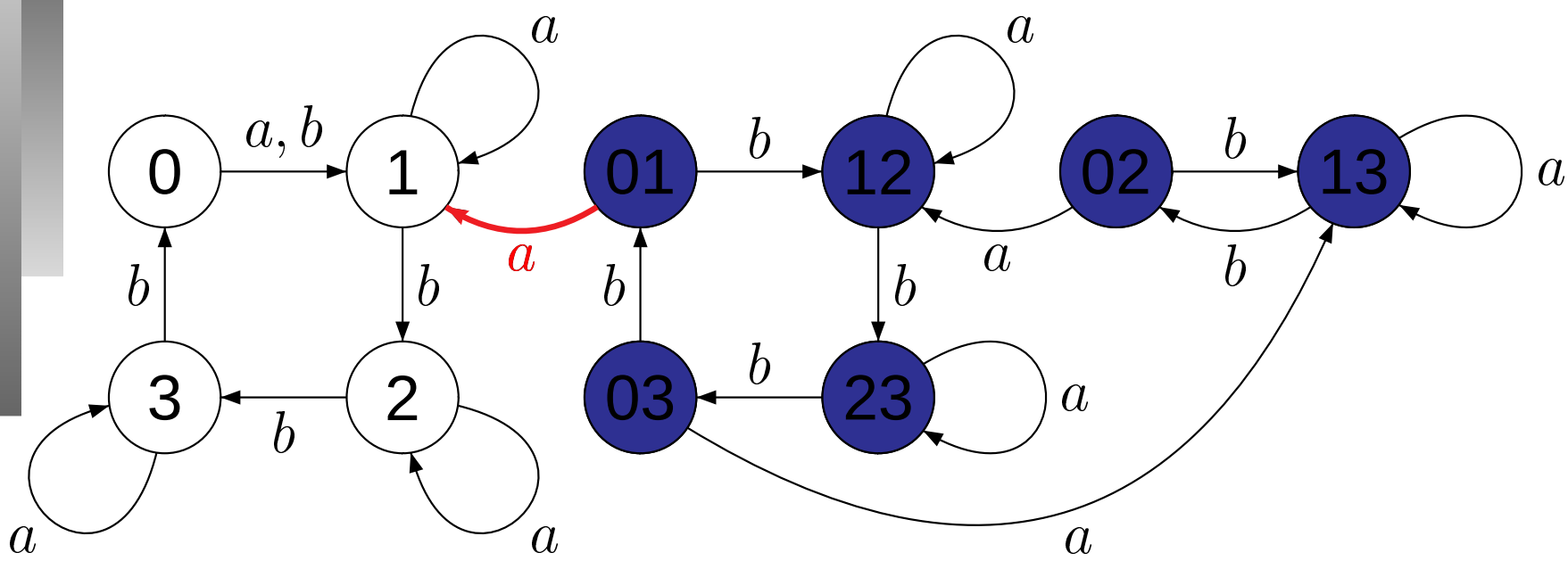
Checking Synchronizability



Checking Synchronizability

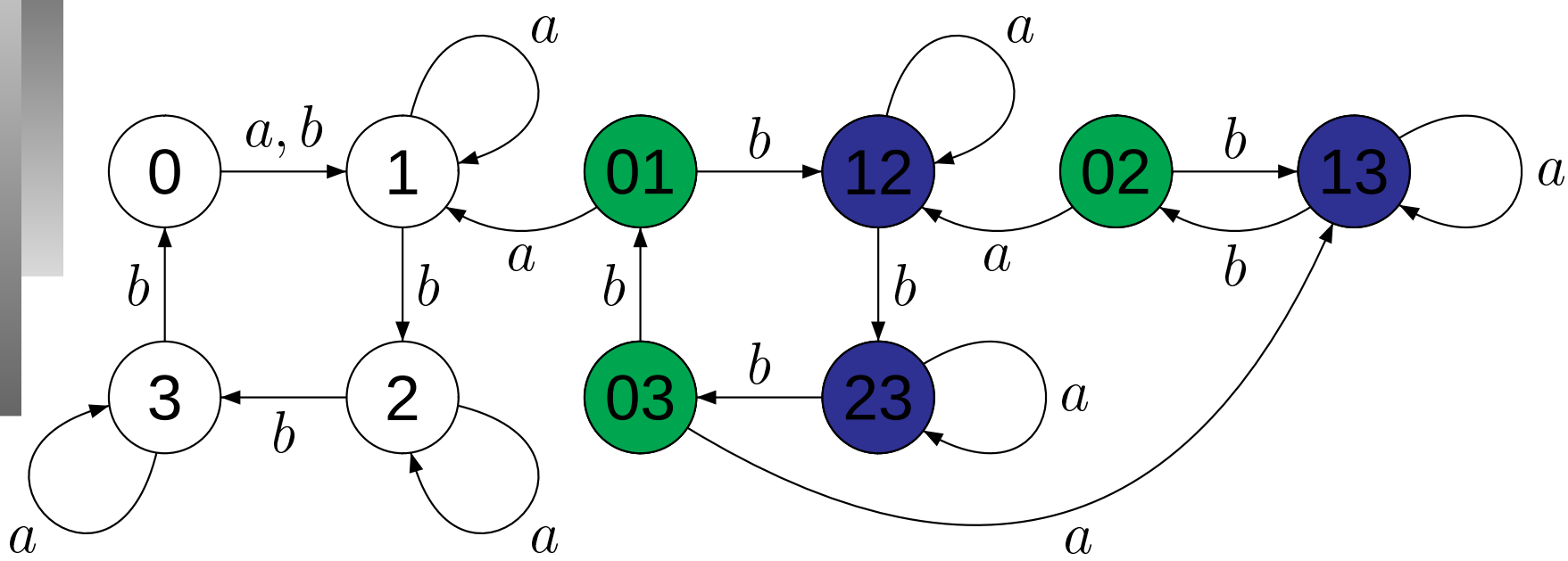


Checking Synchronizability

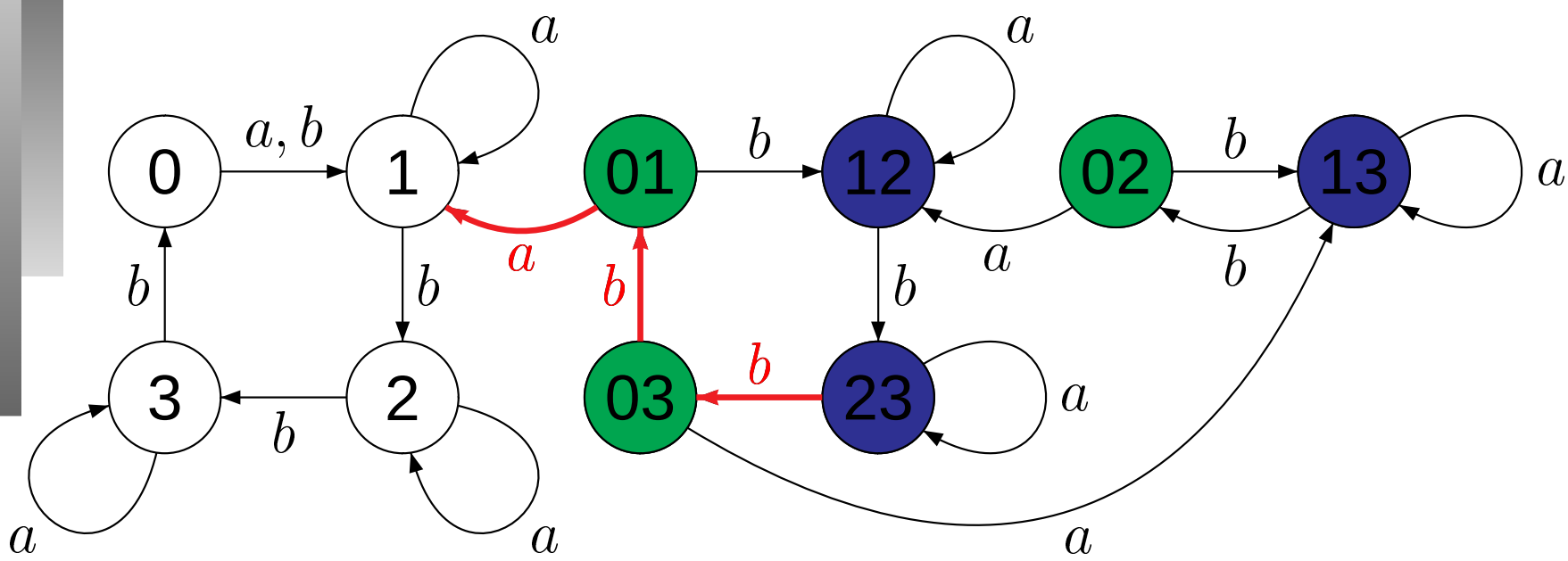


$$a,\ Q\,.\,a = \{1, 2, 3\}$$

Checking Synchronizability

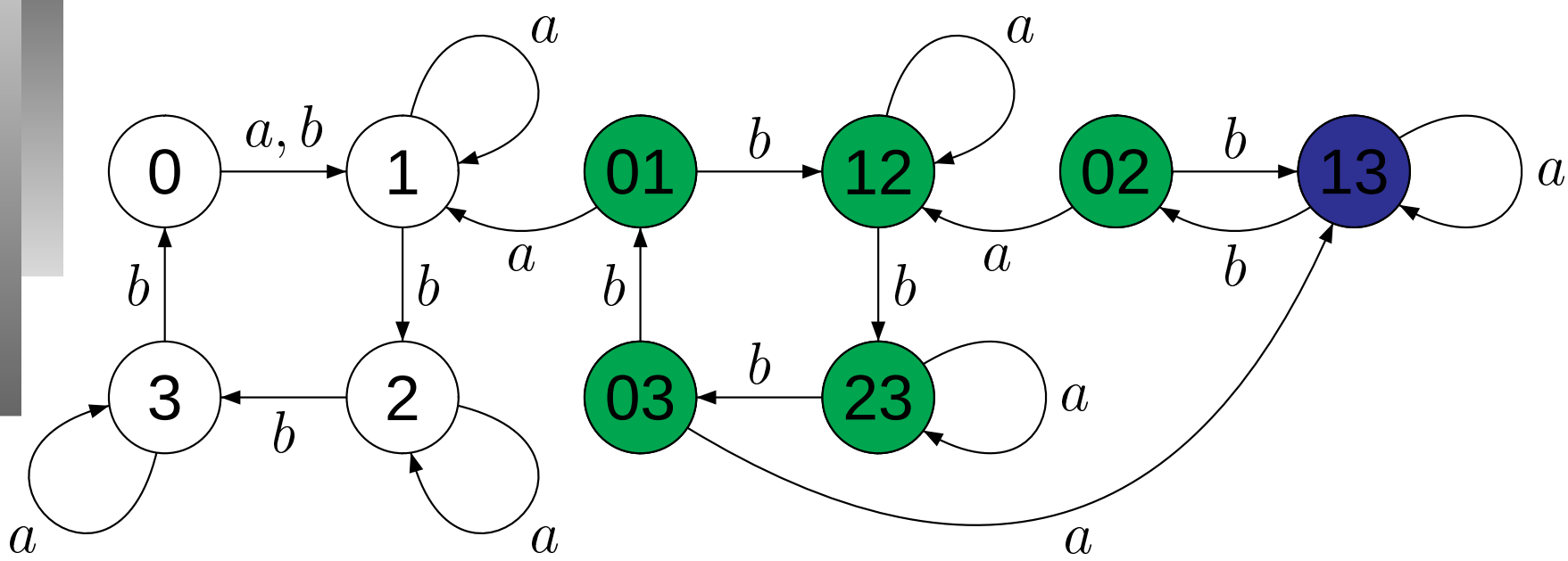


Checking Synchronizability

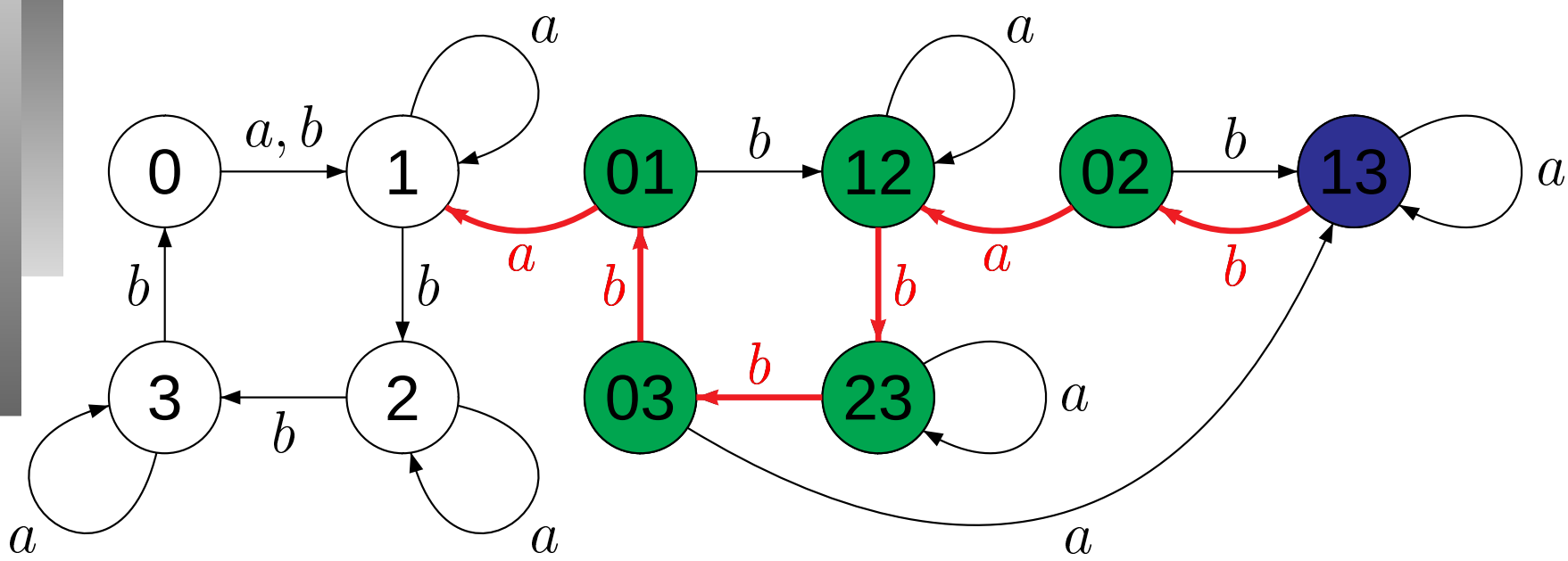


$$a \cdot bba,\ Q \,.\, abba = \{1, 3\}$$

Checking Synchronizability

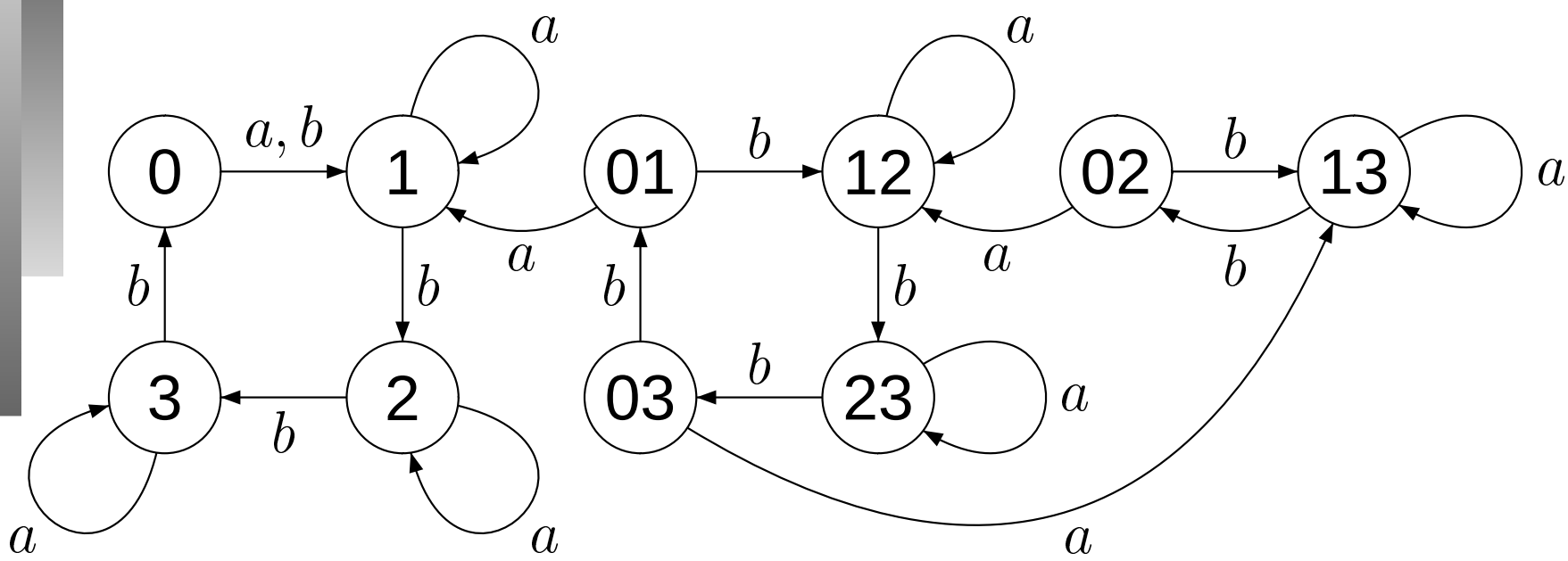


Checking Synchronizability



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

Checking Synchronizability



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

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

Checking Synchronizability

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q.

Checking Synchronizability

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

Checking Synchronizability

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

Checking Synchronizability

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

Clearly, the resulting reset word has length $O(n^3)$:

Checking Synchronizability

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

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of green 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$.

Checking Synchronizability

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

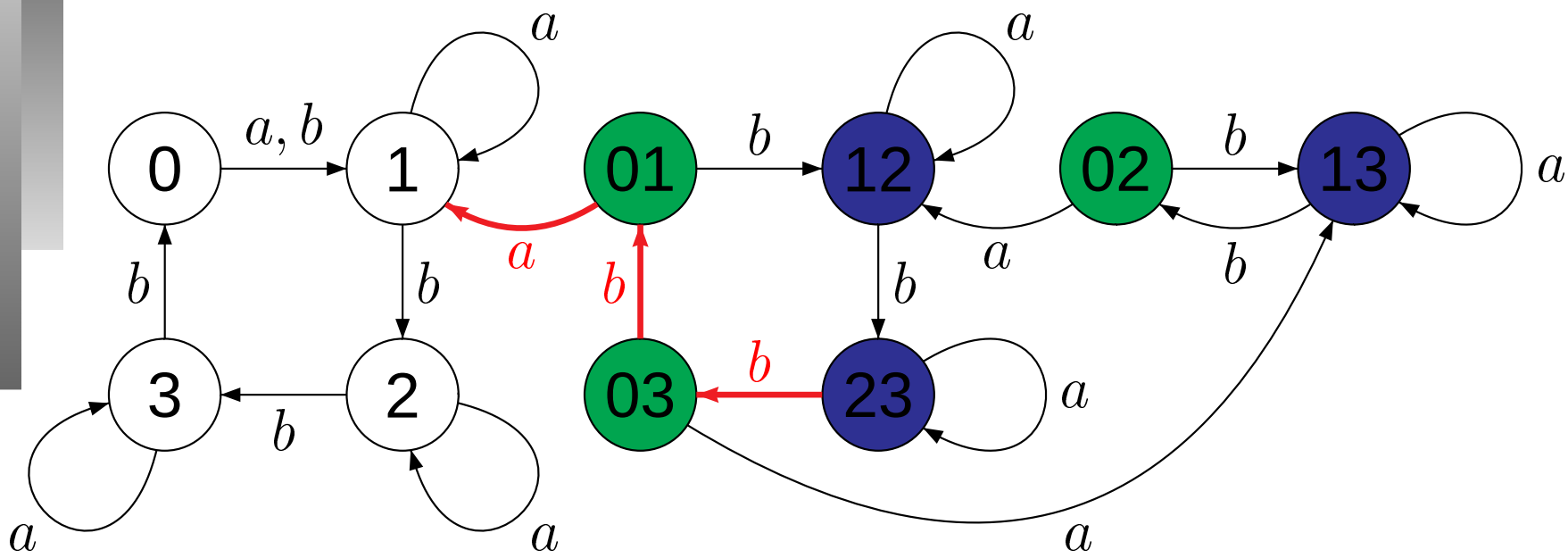
Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of green 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the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

Checking Synchronizability

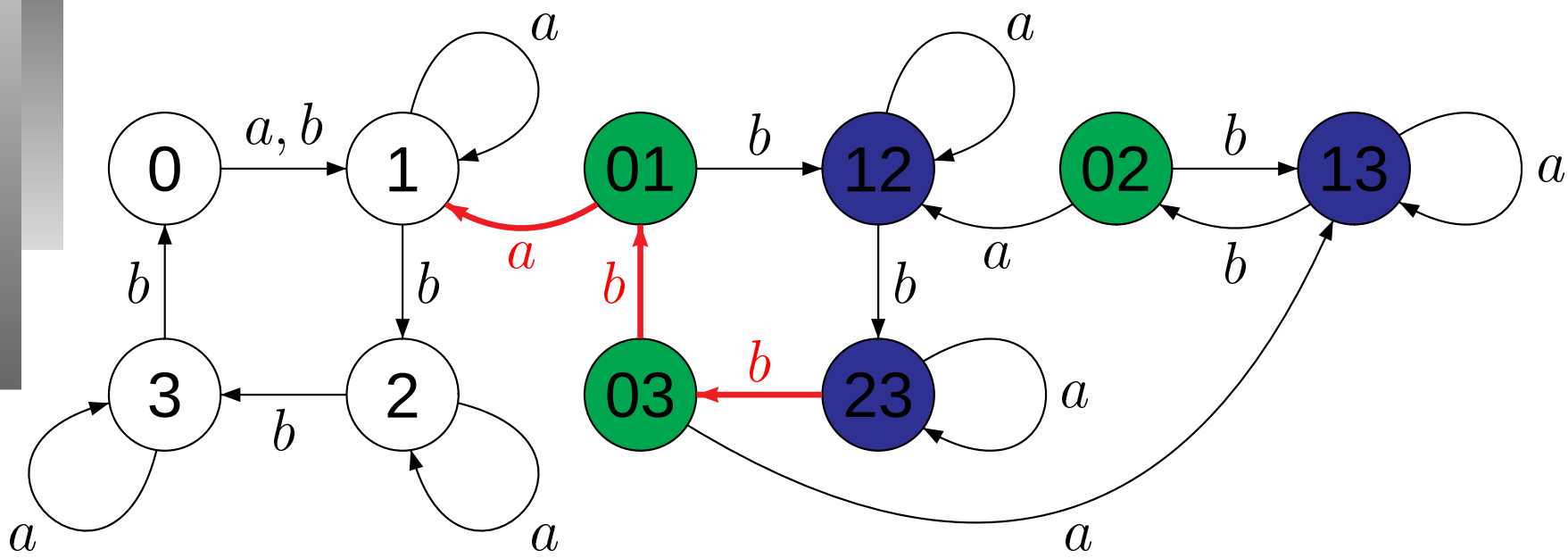
Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$. If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$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

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

Studying Greedy Algorithm

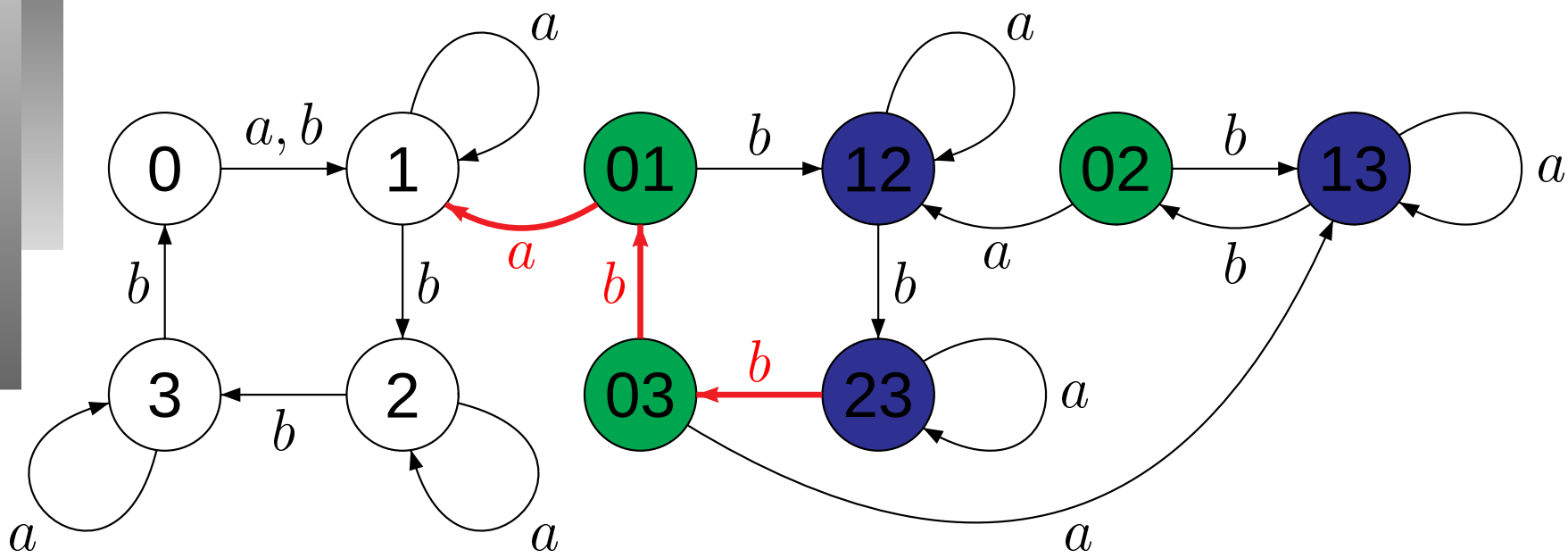


Studying Greedy Algorithm



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

Studying Greedy Algorithm



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

Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$ and let $v = a_1 \cdots a_\ell$ with $a_i \in \Sigma$, $i = 1, \ldots, \ell$, be a word of minimum length such that $|P \,.\, v| < k$.

Studying Greedy Algorithm

The sets $P_1 = P,\ P_2 = P_1 \,.\, a_1,\ \ldots,\ P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q.

Studying Greedy Algorithm

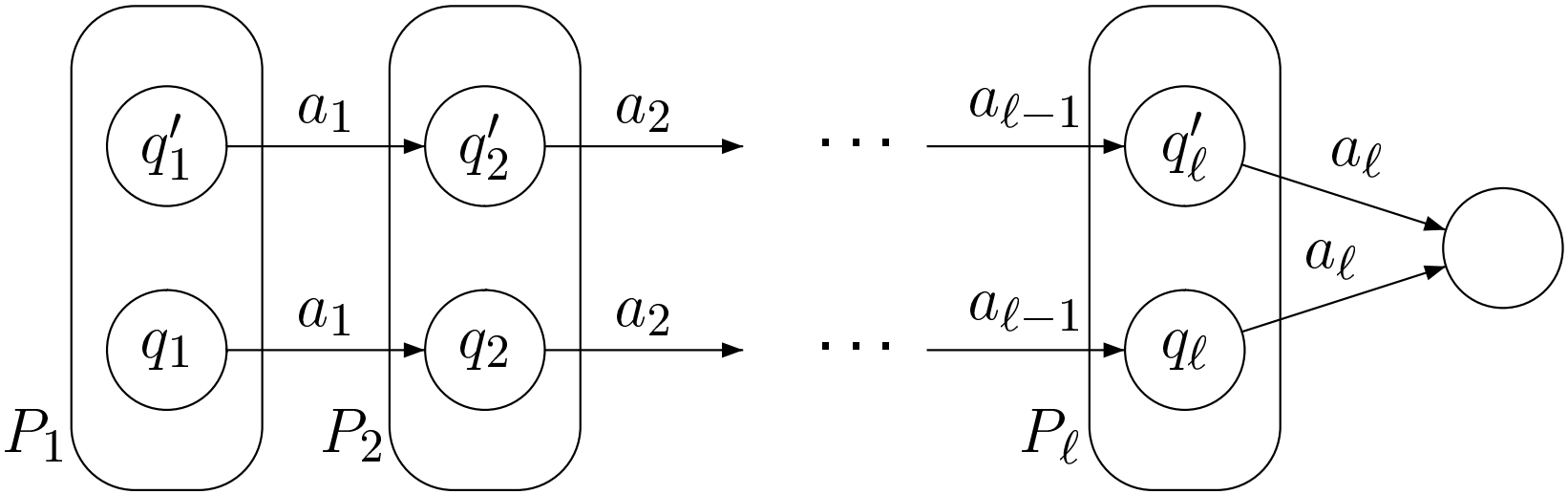
The sets $P_1 = P,\ P_2 = P_1 \,.\, a_1,\ \ldots,\ P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \,.\, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$.

Studying Greedy Algorithm

The sets $P_1 = P,\ P_2 = P_1 \,.\, a_1,\ \ldots,\ P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \,.\, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \ldots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \ldots, \ell - 1$.

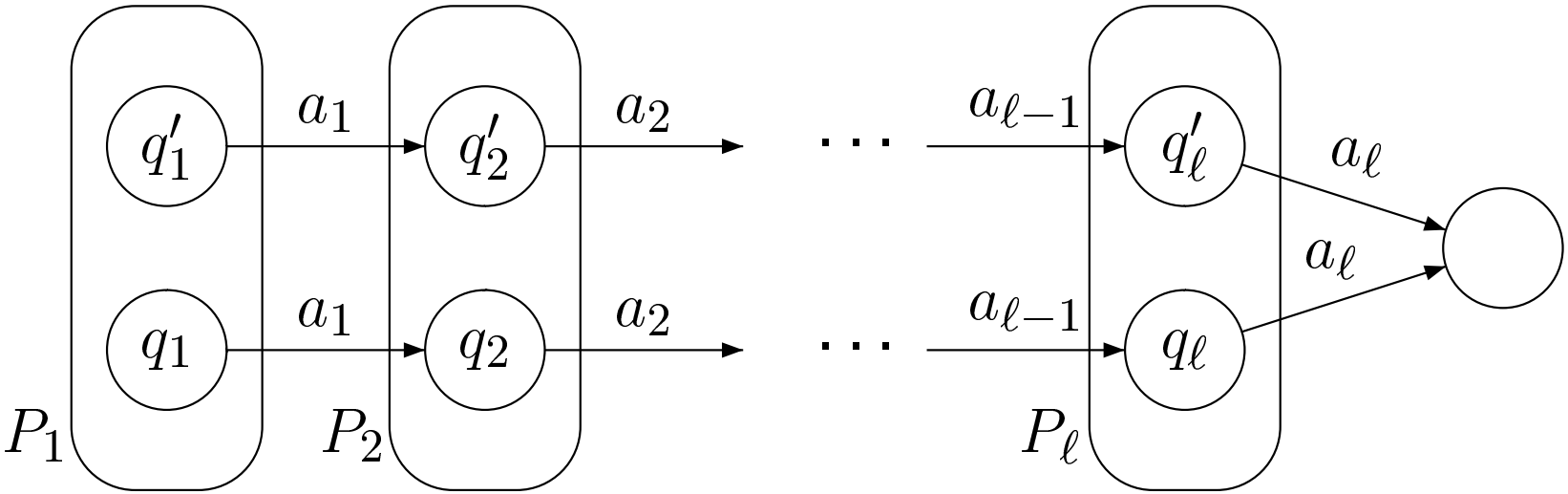
Studying Greedy Algorithm

The sets $P_1 = P,\ P_2 = P_1 \,.\, a_1,\ \ldots,\ P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \,.\, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \ldots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \ldots, \ell - 1$.



Studying Greedy Algorithm

The sets $P_1 = P,\ P_2 = P_1 \,.\, a_1,\ \ldots,\ P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \,.\, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \ldots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \ldots, \ell - 1$.



The condition that v is a word of minimum length with $|P \,.\, v| < |P|$ implies $R_i \not\subseteq P_j$ for $1 \leq j < i \leq \ell$.

Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets $(k > 1)$ such that each P_i, $1 < i \leq \ell$, includes a “fresh” 2-subset that does not occur in any pervious P_j $(1 \leq j < i)$.

Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets $(k > 1)$ such that each P_i, $1 < i \leq \ell$, includes a “fresh” 2-subset that does not occur in any pervious P_j $(1 \leq j < i)$. How long can such *refreshing* sequences be?

Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets $(k > 1)$ such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any pervious P_j $(1 \leq j < i)$. How long can such *refreshing* sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list.

Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets $(k > 1)$ such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any pervious P_j $(1 \leq j < i)$. How long can such *refreshing* sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \ldots, T_s$ of length $s = \binom{n-k+2}{2}$.

Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets $(k > 1)$ such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any pervious P_j $(1 \leq j < i)$. How long can such *refreshing* sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \ldots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

Combinatorial Configuration

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127). See pre-proceedings for a detailed history.

Combinatorial Configuration

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127). See pre-proceedings for a detailed history.

The proof uses linearization techniques which is quite common in combinatorics of finite sets.

Combinatorial Configuration

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127). See pre-proceedings for a detailed history.

The proof uses linearization techniques which is quite common in combinatorics of finite sets. One reformulates the problem in linear algebra terms and then uses the corresponding machinery.

Combinatorial Configuration

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127). See pre-proceedings for a detailed history.

The proof uses linearization techniques which is quite common in combinatorics of finite sets. One reformulates the problem in linear algebra terms and then uses the corresponding machinery.

We identify Q with $\{1, 2, \ldots, n\}$ and assign to each k-subset $I = \{i_1, \ldots, i_k\}$ the following polynomial $D(I)$ in variables $x_{i_1}, \ldots, x_{i_k}$ over the field of rationals.

Combinatorial Configuration

$$I = \{i_1, \ldots, i_k\} \mapsto D(I) = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^{k-3} & x_{i_1} & x_{i_1}^2 \\ 1 & i_2 & i_2^2 & \cdots & i_2^{k-3} & x_{i_2} & x_{i_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^{k-3} & x_{i_k} & x_{i_k}^2 \end{vmatrix}_{k \times k}$$

Combinatorial Configuration

$$I = \{i_1, \ldots, i_k\} \mapsto D(I) = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^{k-3} & x_{i_1} & x_{i_1}^2 \\ 1 & i_2 & i_2^2 & \cdots & i_2^{k-3} & x_{i_2} & x_{i_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^{k-3} & x_{i_k} & x_{i_k}^2 \end{vmatrix}_{k \times k}$$

Then one proves that:

• the polynomials $D(P_1), \ldots, D(P_\ell)$ are linearly independent whenever the k-subsets $P_1, \ldots, P_\ell$ form a refreshing sequence;

Combinatorial Configuration

$$I = \{i_1, \ldots, i_k\} \mapsto D(I) = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^{k-3} & x_{i_1} & x_{i_1}^2 \\ 1 & i_2 & i_2^2 & \cdots & i_2^{k-3} & x_{i_2} & x_{i_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^{k-3} & x_{i_k} & x_{i_k}^2 \end{vmatrix}_{k \times k}$$

Then one proves that:

- the polynomials $D(P_1), \ldots, D(P_\ell)$ are linearly independent whenever the k-subsets $P_1, \ldots, P_\ell$ form a refreshing sequence;
- the polynomials $D(T_1), \ldots, D(T_s)$ (derived from the "standard" sequence) generate the linear space spanned by all polynomials of the form $D(I)$.

Studying Greedy Algorithm

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.

Studying Greedy Algorithm

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$. Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

Studying Greedy Algorithm

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$. Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

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

Studying Greedy Algorithm

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$. Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\begin{matrix} 2 \\ 2 \end{matrix} + \begin{matrix} 3 \\ 2 \end{matrix} + \begin{matrix} 4 \\ 2 \end{matrix} + \cdots + \begin{matrix} n-1 \\ 2 \end{matrix} + \begin{matrix} n \\ 2 \end{matrix} =$$

$$\begin{matrix} 3 \\ 3 \end{matrix} + \begin{matrix} 3 \\ 2 \end{matrix} + \begin{matrix} 4 \\ 2 \end{matrix} + \cdots + \begin{matrix} n-1 \\ 2 \end{matrix} + \begin{matrix} n \\ 2 \end{matrix} =$$

Studying Greedy Algorithm

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$. Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

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

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

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

Studying Greedy Algorithm

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.
Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

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

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

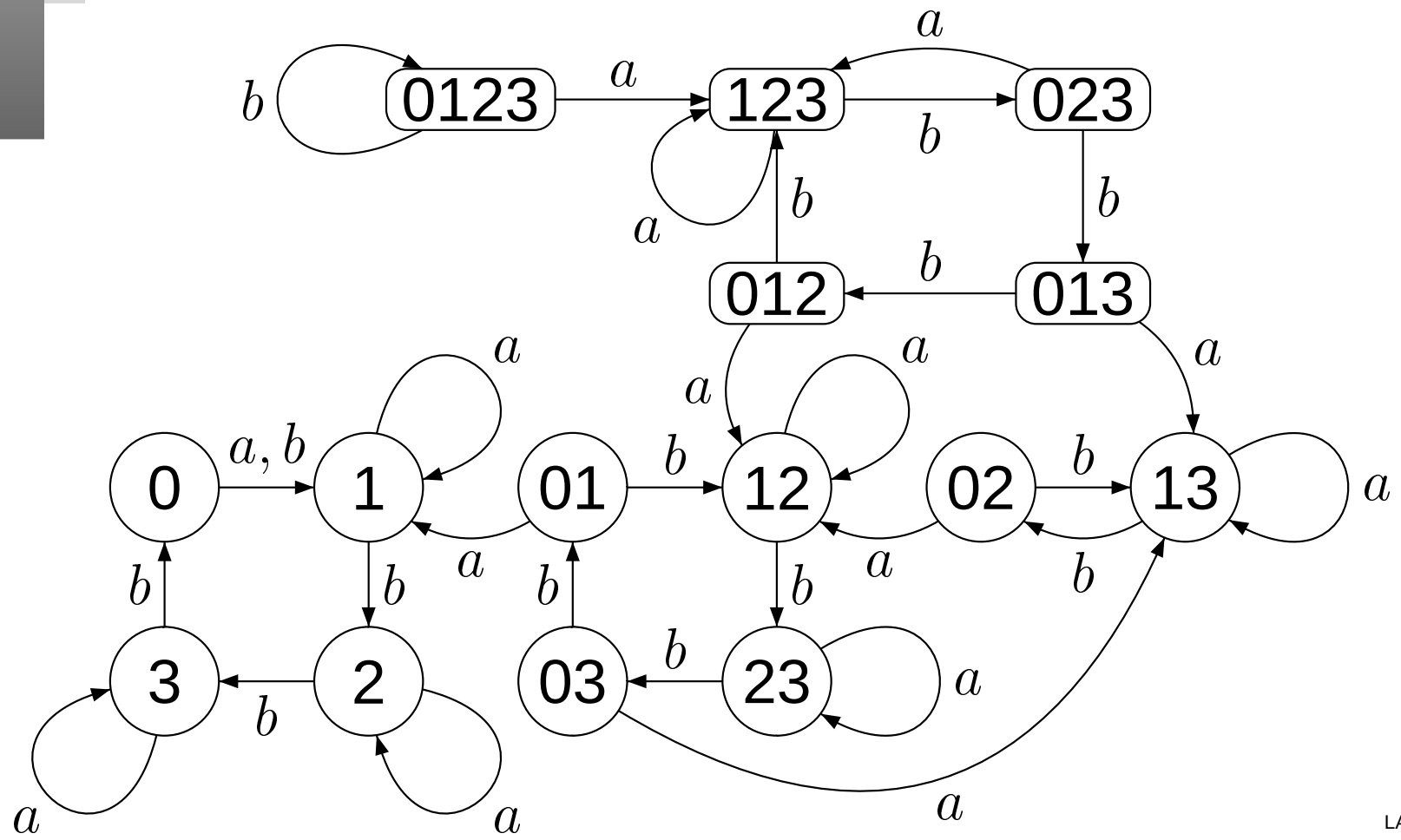
$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}$$

Studying Greedy Algorithm

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

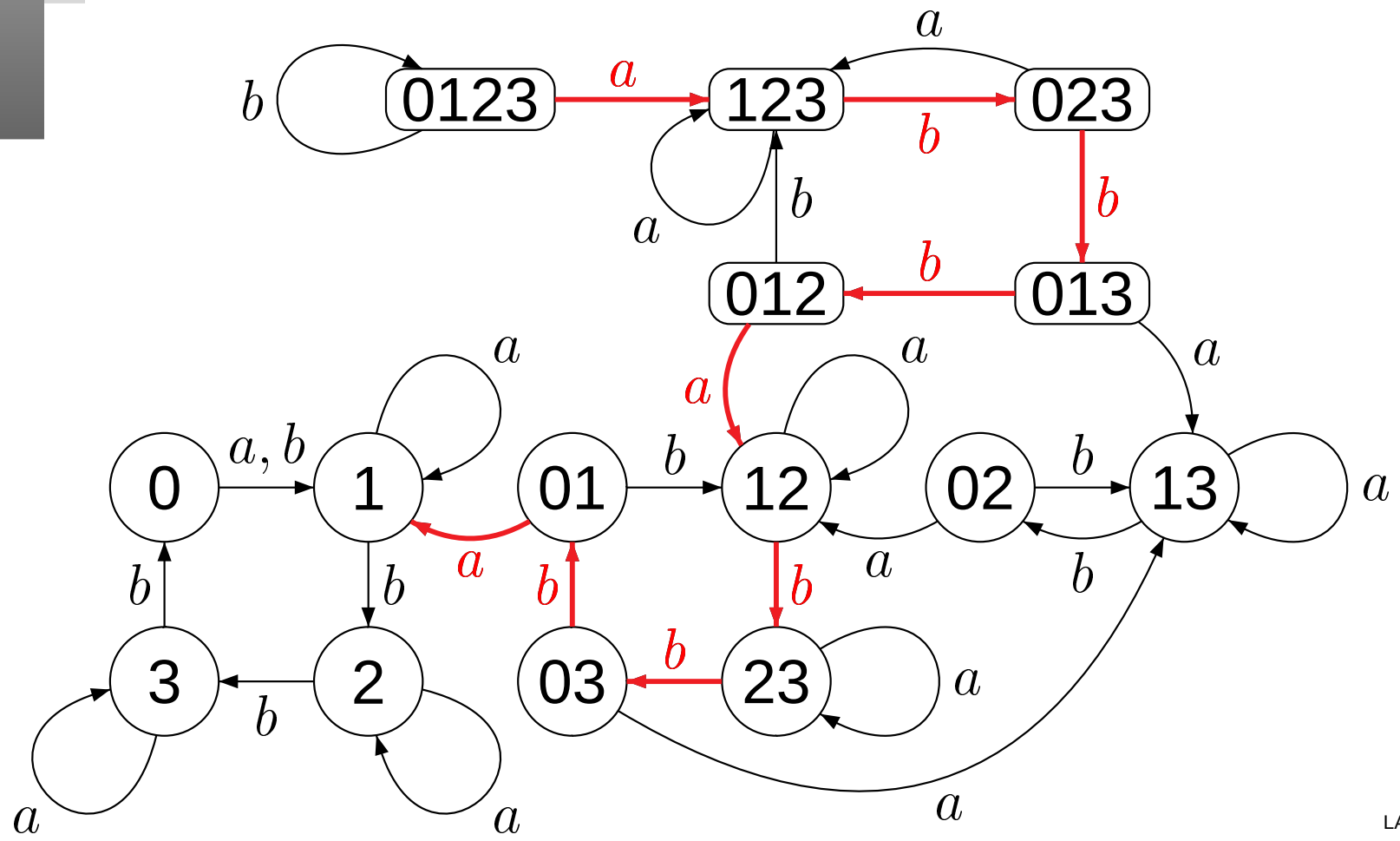
Studying Greedy Algorithm

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



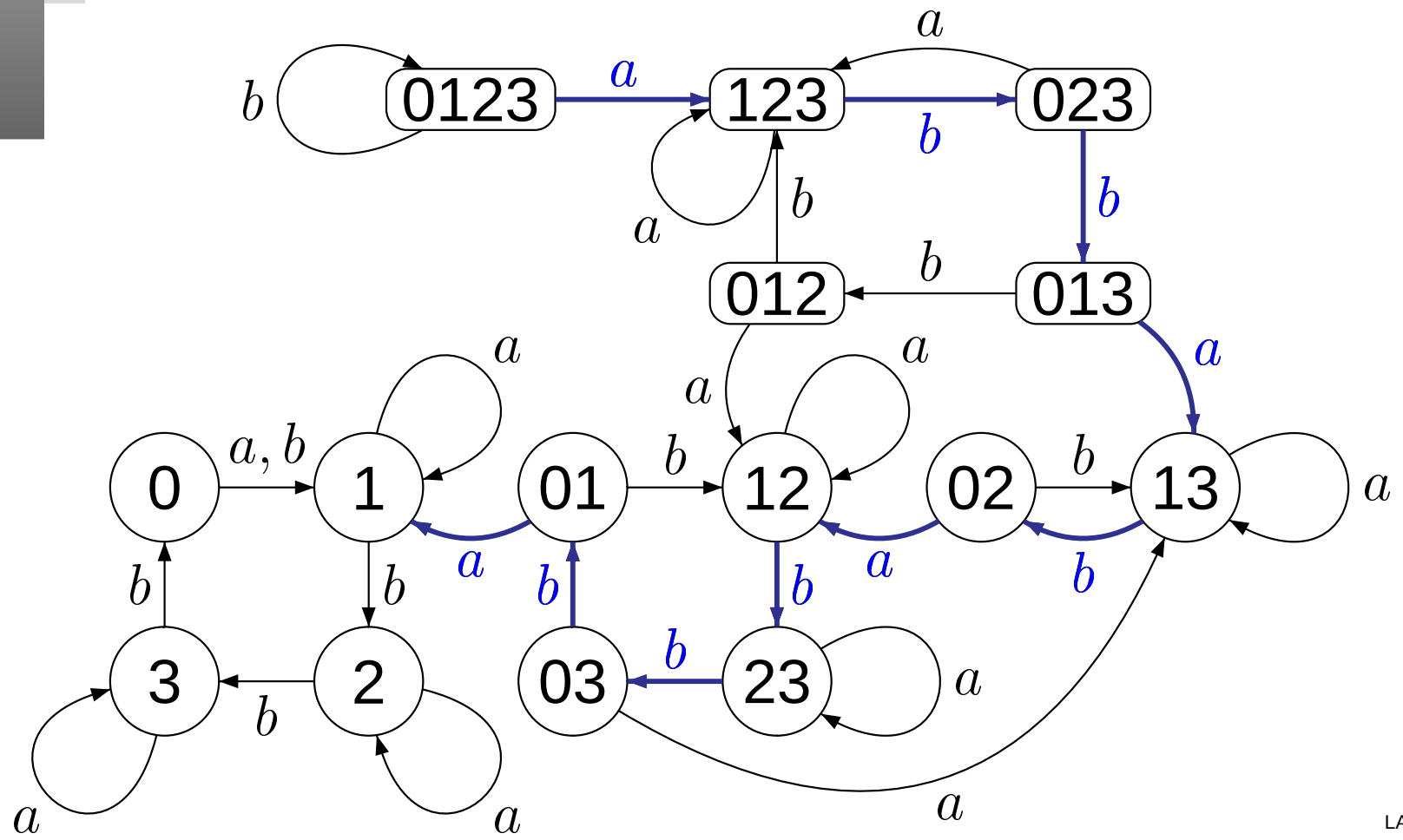
Studying Greedy Algorithm

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



Studying Greedy Algorithm

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



Studying Greedy Algorithm

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large:

Studying Greedy Algorithm

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\frac{n^3-n}{6}$; see Lecture II.

Studying Greedy Algorithm

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\frac{n^3-n}{6}$; see Lecture II.

The behaviour of the greedy algorithm on average is not yet understood;

Studying Greedy Algorithm

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\frac{n^3-n}{6}$; see Lecture II.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Studying Greedy Algorithm

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\frac{n^3-n}{6}$; see Lecture II.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata, see the pre-proceedings.

Studying Greedy Algorithm

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

Studying Greedy Algorithm

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

For instance (though it is not very likely), it may exists a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length either is minimum possible or exceeds the minimum possible exactly by one!