P(l)aying for Synchronization

Mikhail Volkov

Ural Federal University, Ekaterinburg, Russia

LMRC, February 2nd, 2012

Definitions and Terminology

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

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

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

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

Definitions and Terminology

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

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

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

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

Definitions and Terminology

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

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

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

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

Definitions and Terminology

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

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

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

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

Definitions and Terminology

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

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

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

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

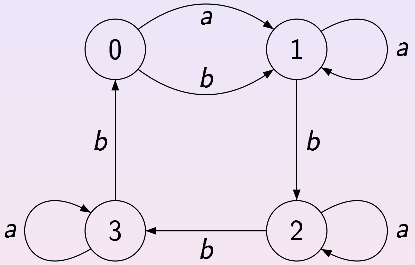
We consider complete deterministic finite automata (DFA) $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ where Q stands for the state set, Σ is the input alphabet, and $\delta : Q \times \Sigma \to Q$ is a (total) transition function.

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

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

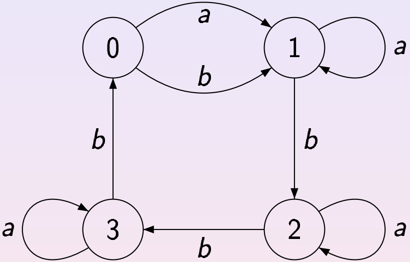
Any w with this property is a reset word for $\mathscr{A}$.

Example



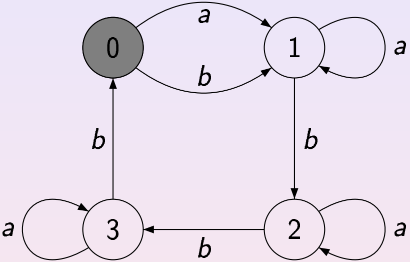
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



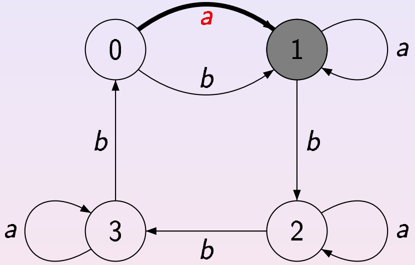
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



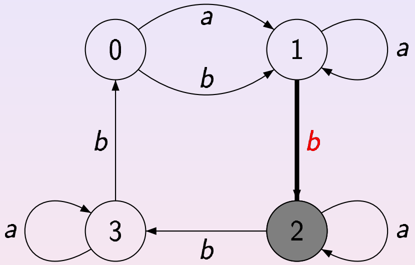
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



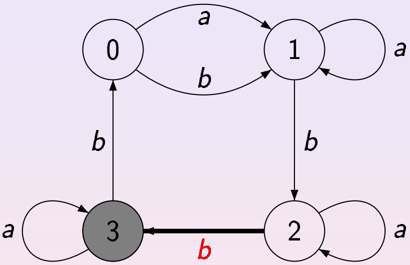
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



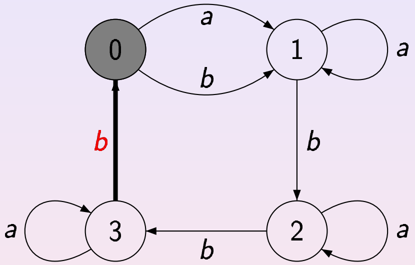
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



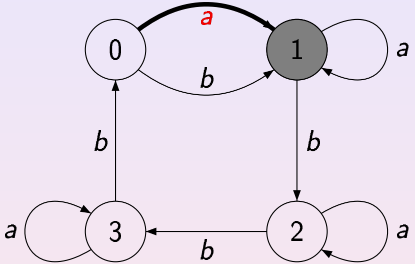
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



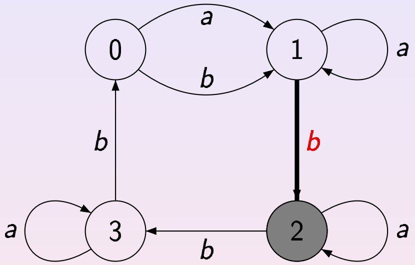
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



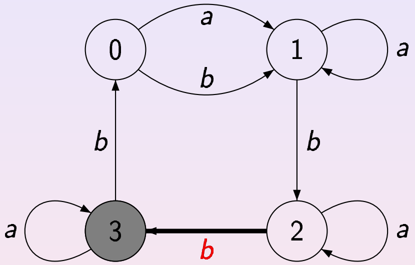
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



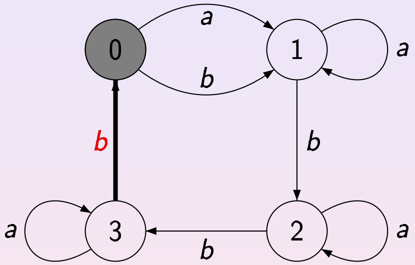
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



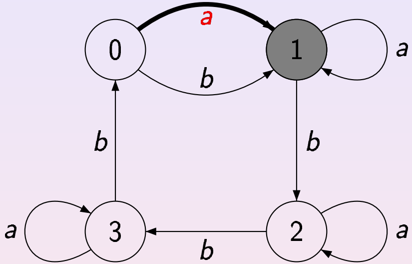
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



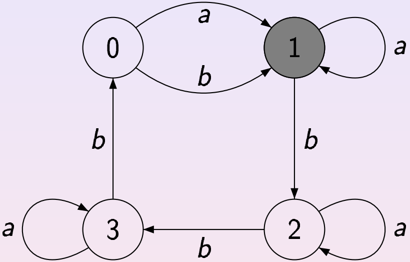
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



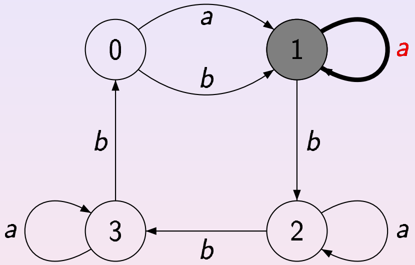
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



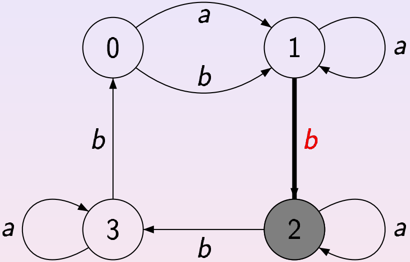
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



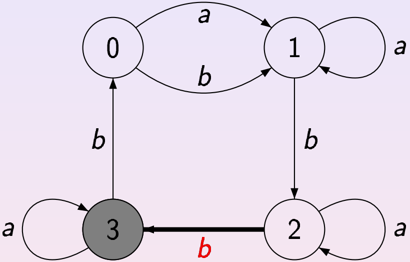
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



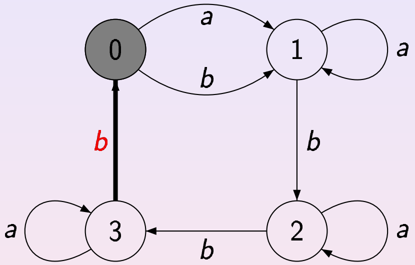
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



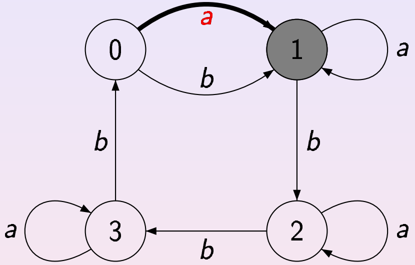
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



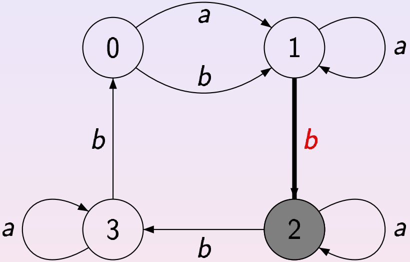
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



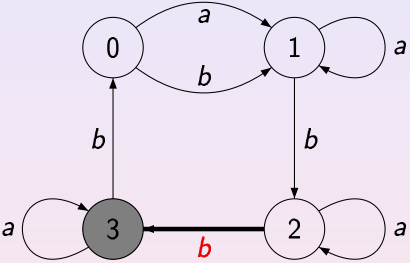
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



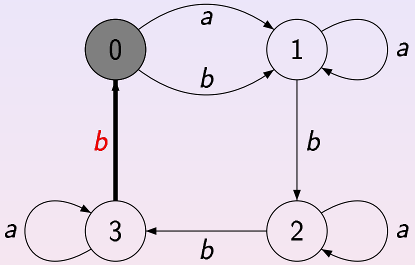
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



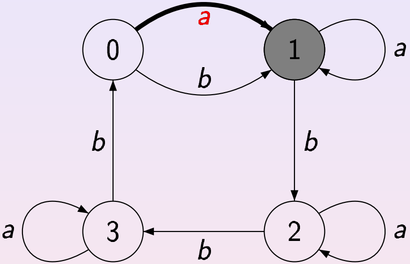
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



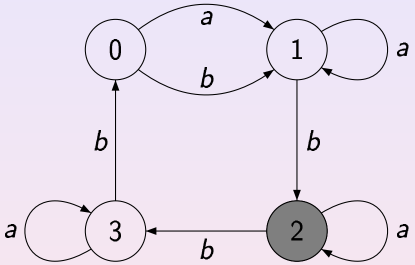
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



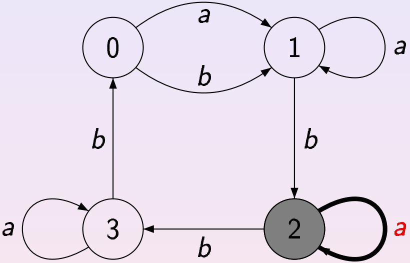
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



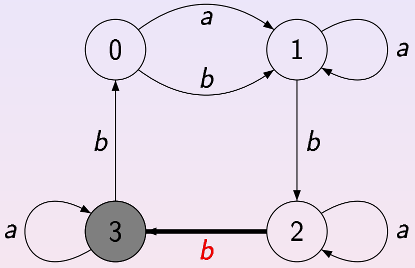
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



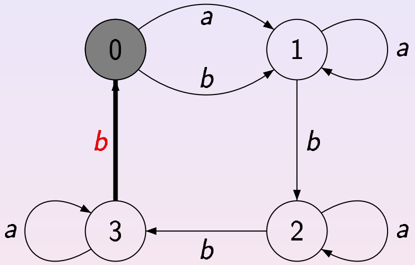
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



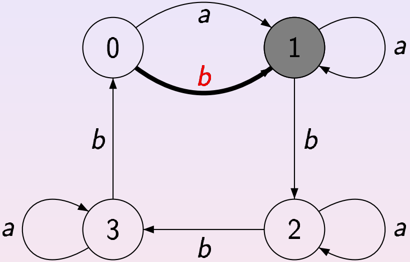
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



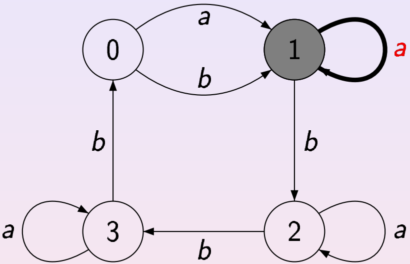
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



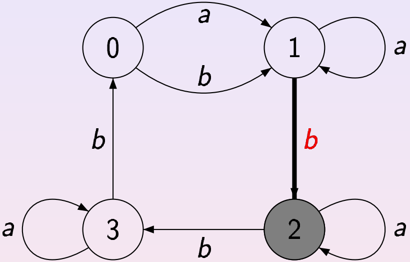
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



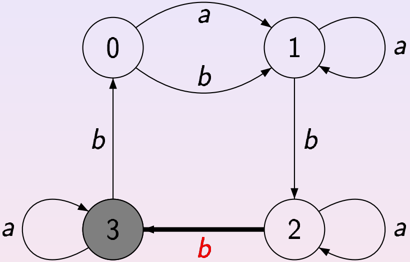
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



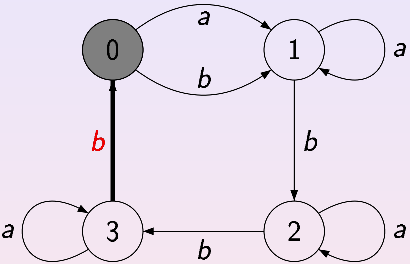
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



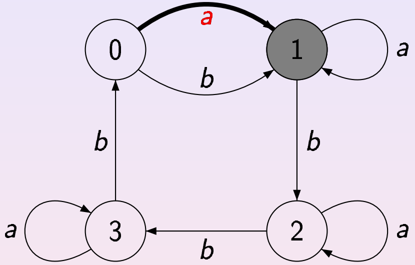
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



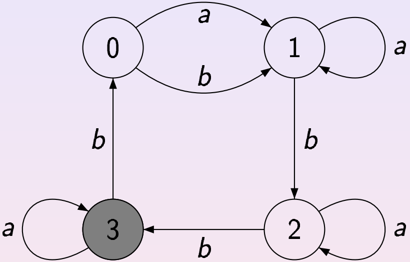
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



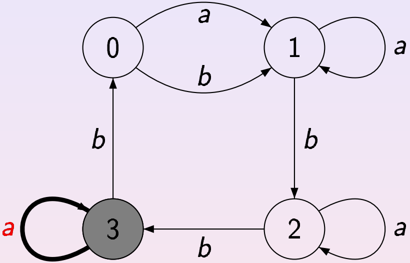
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



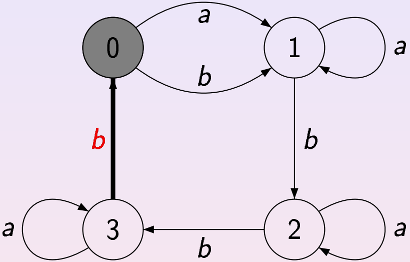
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



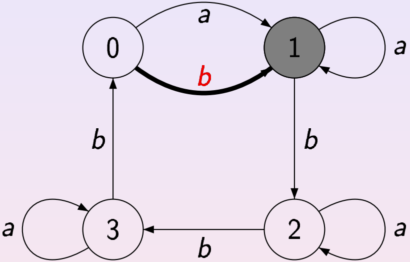
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



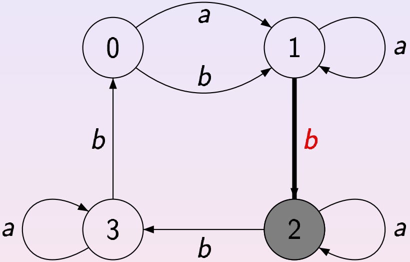
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



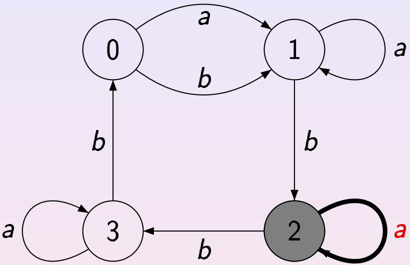
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



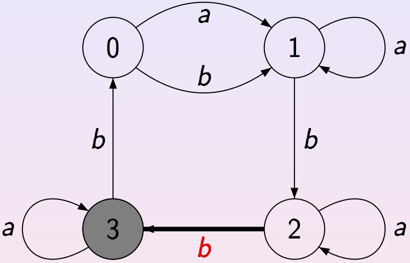
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



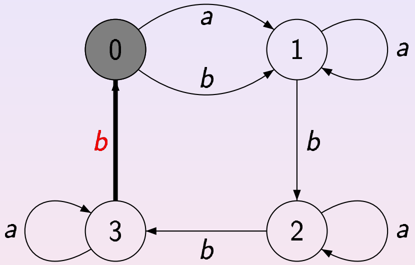
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



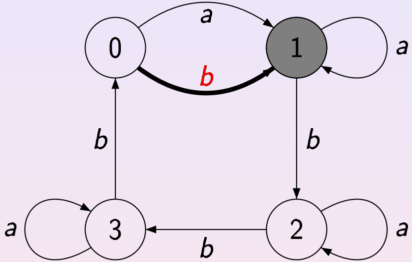
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



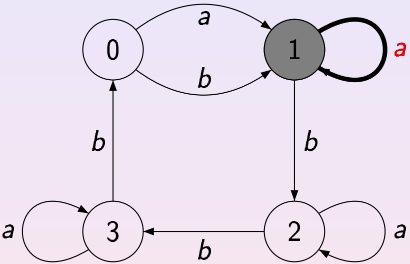
A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Example



A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Our Game

In many situations, it is convenient to think of synchronization of a given automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ in terms of the following solitaire-like game.

- The digraph of $\mathscr{A}$ — the game-board.
- The initial position — each state holds a coin.
- Each letter $c \in \Sigma$ defines a move — coins slide along the arrows labelled c and, whenever two coins meet at some state, one of them is removed.
- The goal — to free all but one states.

Our Game

In many situations, it is convenient to think of synchronization of a given automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ in terms of the following solitaire-like game.

- The digraph of $\mathscr{A}$ — the game-board.
- The initial position — each state holds a coin.
- Each letter $c \in \Sigma$ defines a move — coins slide along the arrows labelled c and, whenever two coins meet at some state, one of them is removed.
- The goal — to free all but one states.

Our Game

In many situations, it is convenient to think of synchronization of a given automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ in terms of the following solitaire-like game.

- The digraph of $\mathscr{A}$ — the game-board.
- The initial position — each state holds a coin.
- Each letter $c \in \Sigma$ defines a move — coins slide along the arrows labelled c and, whenever two coins meet at some state, one of them is removed.
- The goal — to free all but one states.

Our Game

In many situations, it is convenient to think of synchronization of a given automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ in terms of the following solitaire-like game.

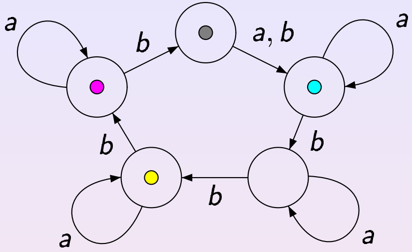
- The digraph of $\mathscr{A}$ — the game-board.
- The initial position — each state holds a coin.
- Each letter $c \in \Sigma$ defines a move — coins slide along the arrows labelled c and, whenever two coins meet at some state, one of them is removed.
- The goal — to free all but one states.

Our Game

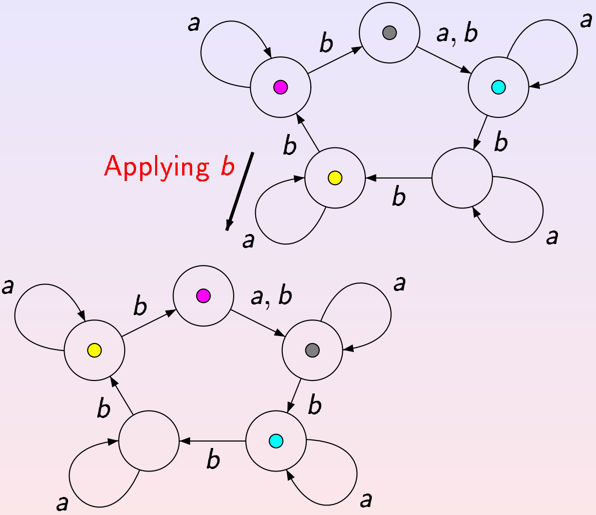
In many situations, it is convenient to think of synchronization of a given automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ in terms of the following solitaire-like game.

- The digraph of $\mathscr{A}$ — the game-board.
- The initial position — each state holds a coin.
- Each letter $c \in \Sigma$ defines a move — coins slide along the arrows labelled c and, whenever two coins meet at some state, one of them is removed.
- The goal — to free all but one states.

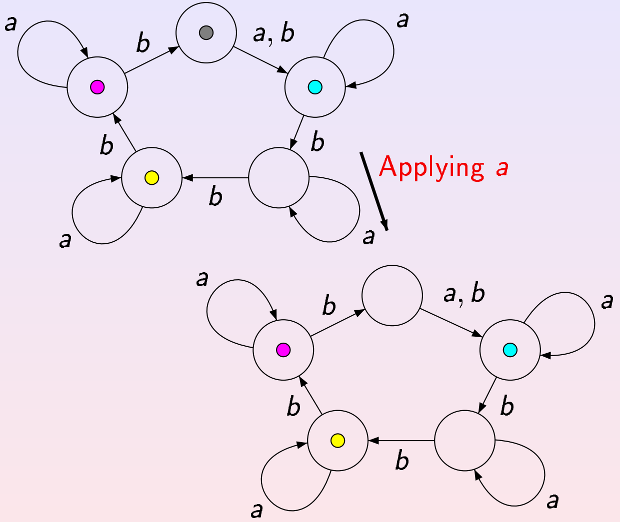
An Illustration



An Illustration

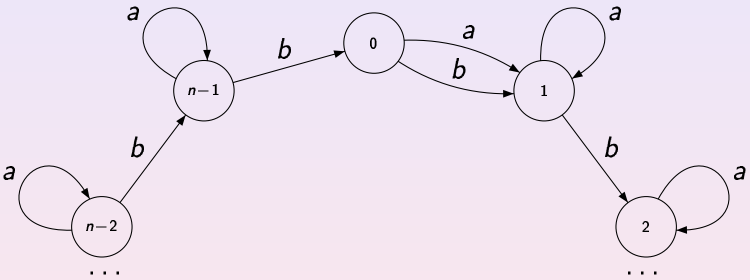


An Illustration



The Černý Series

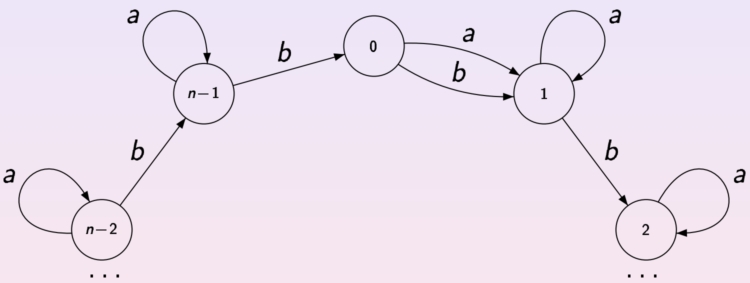
In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \ldots,$ of synchronizing automata over 2 letters:



Černý has proved that the shortest reset word for $\mathscr{C}_n$ is $(ab^{n-1})^{n-2}a$ of length $(n-1)^2$. We present a proof of this result using a synchronization game.

The Černý Series

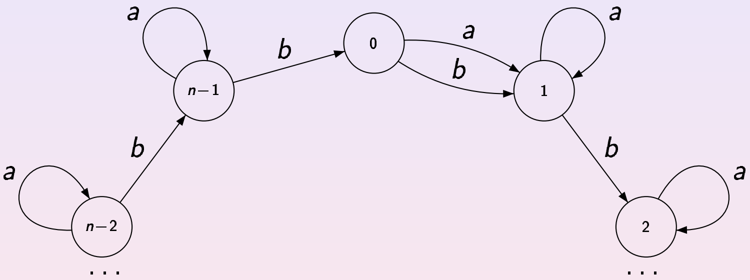
In his 1964 paper Jan Černý constructed a series $\mathscr{C}_n$, $n = 2, 3, \ldots$, of synchronizing automata over 2 letters:



Černý has proved that the shortest reset word for $\mathscr{C}_n$ is $(ab^{n-1})^{n-2}a$ of length $(n-1)^2$. We present a proof of this result using a synchronization game.

The Černý Series

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



Černý has proved that the shortest reset word for $\mathscr{C}_n$ is $(ab^{n-1})^{n-2}a$ of length $(n-1)^2$. We present a proof of this result using a synchronization game.

Extra Conventions

In the game on $\mathscr{C}_n$ we adopt a few extra conventions:

- The coins are pairwise distinct.
- Whenever two coins meet at the state 1, the coin arriving from 0 is removed.
- The only coin that remains at the end of the game is called the golden coin G.

LMRC, February 2nd, 2012

Extra Conventions

In the game on $\mathscr{C}_n$ we adopt a few extra conventions:

- The coins are pairwise distinct.
- Whenever two coins meet at the state 1, the coin arriving from 0 is removed.
- The only coin that remains at the end of the game is called the golden coin G.

Extra Conventions

In the game on $\mathscr{C}_n$ we adopt a few extra conventions:

- The coins are pairwise distinct.
- Whenever two coins meet at the state 1, the coin arriving from 0 is removed.
- The only coin that remains at the end of the game is called the golden coin G.

Extra Conventions

In the game on $\mathscr{C}_n$ we adopt a few extra conventions:

- The coins are pairwise distinct.
- Whenever two coins meet at the state 1, the coin arriving from 0 is removed.
- The only coin that remains at the end of the game is called the golden coin G.

Key Idea

Let P_0 be an initial distribution of coins, w a reset word. Denote by P_i the position that arises when we apply the prefix of w of length i to the position P_0. We want to define the weight $\mathrm{wg}(P_i)$ of the position such that

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) for each $i = 1, \ldots, |w|$, the action of the i^{th} letter of w decreases the weight by 1 at most, that is, $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$.

Then $$|w| = \sum_{i=1}^{|w|} 1 \geq \sum_{i=1}^{|w|} (\mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)) =$$

$$\mathrm{wg}(P_0) - \mathrm{wg}(P_{|w|}) \geq n(n-1) - (n-1) = (n-1)^2.$$

Key Idea

Let P_0 be an initial distribution of coins, w a reset word. Denote by P_i the position that arises when we apply the prefix of w of length i to the position P_0. We want to define the weight $\mathrm{wg}(P_i)$ of the position such that

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) for each $i = 1, \ldots, |w|$, the action of the i^{th} letter of w decreases the weight by 1 at most, that is, $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$.

$$\text{Then} \quad |w| = \sum_{i=1}^{|w|} 1 \geq \sum_{i=1}^{|w|} (\mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)) =$$
$$\mathrm{wg}(P_0) - \mathrm{wg}(P_{|w|}) \geq n(n-1) - (n-1) = (n-1)^2.$$

Key Idea

Let P_0 be an initial distribution of coins, w a reset word. Denote by P_i the position that arises when we apply the prefix of w of length i to the position P_0. We want to define the weight $\mathrm{wg}(P_i)$ of the position such that

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) for each $i = 1, \ldots, |w|$, the action of the i^{th} letter of w decreases the weight by 1 at most, that is, $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$.

Then $$|w| = \sum_{i=1}^{|w|} 1 \geq \sum_{i=1}^{|w|} (\mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)) =$$
$$\mathrm{wg}(P_0) - \mathrm{wg}(P_{|w|}) \geq n(n-1) - (n-1) = (n-1)^2.$$

Key Idea

Let P_0 be an initial distribution of coins, w a reset word. Denote by P_i the position that arises when we apply the prefix of w of length i to the position P_0. We want to define the weight $\mathrm{wg}(P_i)$ of the position such that

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) for each $i = 1, \ldots, |w|$, the action of the i^{th} letter of w decreases the weight by 1 at most, that is, $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$.

Then $$|w| = \sum_{i=1}^{|w|} 1 \geq \sum_{i=1}^{|w|} (\mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)) = \mathrm{wg}(P_0) - \mathrm{wg}(P_{|w|}) \geq n(n-1) - (n-1) = (n-1)^2.$$

Key Idea

Let P_0 be an initial distribution of coins, w a reset word. Denote by P_i the position that arises when we apply the prefix of w of length i to the position P_0. We want to define the weight $\mathrm{wg}(P_i)$ of the position such that

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) for each $i = 1, \ldots, |w|$, the action of the i^{th} letter of w decreases the weight by 1 at most, that is, $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$.

Then $|w| = \sum_{i=1}^{|w|} 1 \geq \sum_{i=1}^{|w|} (\mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)) =$

$$\mathrm{wg}(P_0) - \mathrm{wg}(P_{|w|}) \geq n(n-1) - (n-1) = (n-1)^2.$$

Constructing the Weight Function

The trick consists in letting the weight of each coin depend on its relative location w.r.t. the golden coin.

If a coin C is present in a position P_i, define the *weight of C in P_i* as

$$\mathrm{wg}(C, P_i) = n \cdot d_i(C) + m_i(C)$$

where $m_i(C)$ is the number of moves from the state holding C to 0 and $d_i(C)$ is the number of moves from the state holding C to one holding G. (Recall that G stands for the golden coin G which is present in all positions.)

The weight of P_i is the maximum weight of the coins present in this position.

Constructing the Weight Function

The trick consists in letting the weight of each coin depend on its relative location w.r.t. the golden coin.

If a coin C is present in a position P_i, define the *weight of C in P_i* as

$$\mathrm{wg}(C, P_i) = n \cdot d_i(C) + m_i(C)$$

where $m_i(C)$ is the number of moves from the state holding C to 0 and $d_i(C)$ is the number of moves from the state holding C to one holding G. (Recall that G stands for the golden coin G which is present in all positions.)

The weight of P_i is the maximum weight of the coins present in this position.

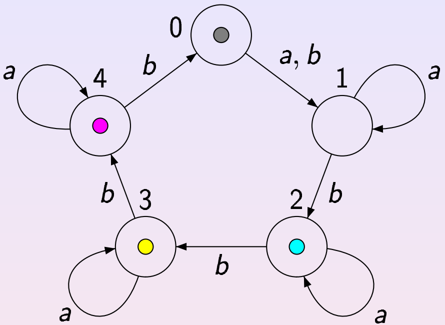
Constructing the Weight Function

The trick consists in letting the weight of each coin depend on its relative location w.r.t. the golden coin.
If a coin C is present in a position P_i, define the *weight of C in P_i* as

$$\mathrm{wg}(C, P_i) = n \cdot d_i(C) + m_i(C)$$

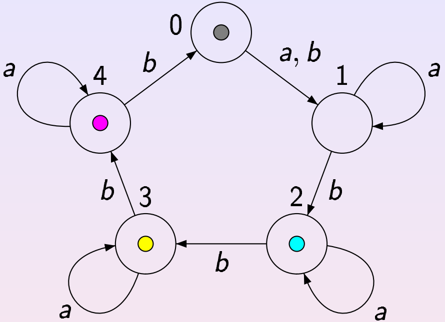
where $m_i(C)$ is the number of moves from the state holding C to 0 and $d_i(C)$ is the number of moves from the state holding C to one holding G. (Recall that G stands for the golden coin G which is present in all positions.)
The weight of P_i is the maximum weight of the coins present in this position.

Example



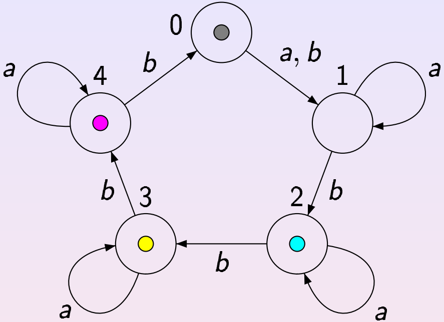
Assume that the yellow coin is the golden one. Then its weight is 2. The weight of the cyan coin is $5 \cdot 1 + 3 = 8$. The weight of the gray coin is $5 \cdot 3 + 0 = 15$. The weight of the magenta coin is $5 \cdot 4 + 1 = 21$, and this is the weight of the position.

Example



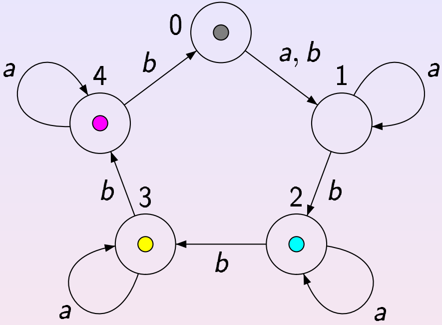
Assume that the yellow coin is the golden one. Then its weight is 2. The weight of the cyan coin is $5 \cdot 1 + 3 = 8$. The weight of the gray coin is $5 \cdot 3 + 0 = 15$. The weight of the magenta coin is $5 \cdot 4 + 1 = 21$, and this is the weight of the position.

Example



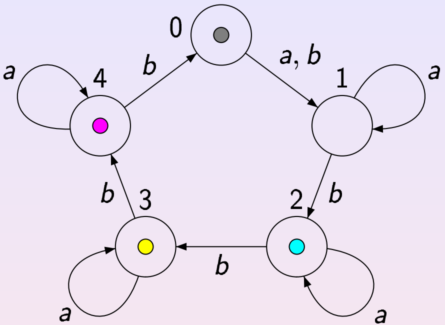
Assume that the yellow coin is the golden one. Then its weight is 2. The weight of the cyan coin is $5 \cdot 1 + 3 = 8$. The weight of the gray coin is $5 \cdot 3 + 0 = 15$. The weight of the magenta coin is $5 \cdot 4 + 1 = 21$, and this is the weight of the position.

Example



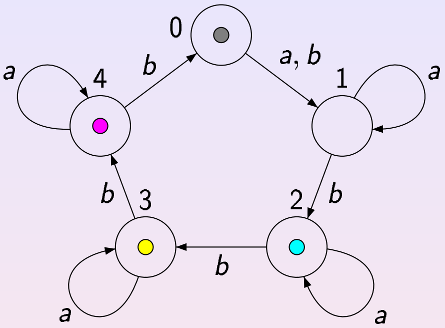
Assume that the yellow coin is the golden one. Then its weight is 2. The weight of the cyan coin is $5 \cdot 1 + 3 = 8$. The weight of the gray coin is $5 \cdot 3 + 0 = 15$. The weight of the magenta coin is $5 \cdot 4 + 1 = 21$, and this is the weight of the position.

Example



Assume that the yellow coin is the golden one. Then its weight is 2. The weight of the cyan coin is $5 \cdot 1 + 3 = 8$. The weight of the gray coin is $5 \cdot 3 + 0 = 15$. The weight of the magenta coin is $5 \cdot 4 + 1 = 21$, and this is the weight of the position.

Example



Assume that the yellow coin is the golden one. Then its weight is 2. The weight of the cyan coin is $5 \cdot 1 + 3 = 8$. The weight of the gray coin is $5 \cdot 3 + 0 = 15$. The weight of the magenta coin is $5 \cdot 4 + 1 = 21$, and this is the weight of the position.

Properties of the Weight Function

We have to check that our weight function satisfies the conditions

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$ for each $i = 1, \dots, |w|$.

In the initial position all states are covered with coins. Consider the coin C that covers the state in one step clockwise after the state covered with the golden coin. Then $d_0(C) = n-1$ whence

$$\mathrm{wg}(C, P_0) = n \cdot (n-1) + m_0(C) \geq n(n-1).$$

Since the weight of a position is not less than the weight of any coin in this position, we have $\mathrm{wg}(P_0) \geq n(n-1)$.

We have to check that our weight function satisfies the conditions

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$ for each $i = 1, \dots, |w|$.

In the initial position all states are covered with coins. Consider the coin C that covers the state in one step clockwise after the state covered with the golden coin. Then $d_0(C) = n-1$ whence

$$\mathrm{wg}(C, P_0) = n \cdot (n-1) + m_0(C) \geq n(n-1).$$

Since the weight of a position is not less than the weight of any coin in this position, we have $\mathrm{wg}(P_0) \geq n(n-1)$.

Properties of the Weight Function

We have to check that our weight function satisfies the conditions

(i) $\mathrm{wg}(P_0) \geq n(n-1)$ and $\mathrm{wg}(P_{|w|}) \leq n-1$;

(ii) $1 \geq \mathrm{wg}(P_{i-1}) - \mathrm{wg}(P_i)$ for each $i = 1, \dots, |w|$.

In the initial position all states are covered with coins. Consider the coin C that covers the state in one step clockwise after the state covered with the golden coin. Then $d_0(C) = n-1$ whence

$$\mathrm{wg}(C, P_0) = n \cdot (n-1) + m_0(C) \geq n(n-1).$$

Since the weight of a position is not less than the weight of any coin in this position, we have $\mathrm{wg}(P_0) \geq n(n-1)$.

Properties of the Weight Function

In the final position only the golden coin G remains, whence the weight of $P_{|w|}$ is the weight of G. Clearly, $\mathrm{wg}(G, P_i) = m_i(G) \leq n-1$ for any position P_i.

Let C be a coin of maximum weight in P_{i-1}. If the transition from P_{i-1} to P_i is caused by b, then $d_i(C) = d_{i-1}(C)$ (because the relative location of the coins does not change) and $m_i(C) = m_{i-1}(C) - 1$ if $m_{i-1}(C) > 0$, otherwise $m_i(C) = n-1$. We see that

$$\mathrm{wg}(P_i) \geq \mathrm{wg}(C, P_i) = n \cdot d_i(C) + m_i(C) \geq$$
$$n \cdot d_{i-1}(C) + m_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1.$$

Properties of the Weight Function

In the final position only the golden coin G remains, whence the weight of $P_{|w|}$ is the weight of G. Clearly, $\mathrm{wg}(G, P_i) = m_i(G) \leq n - 1$ for any position P_i.

Let C be a coin of maximum weight in P_{i-1}. If the transition from P_{i-1} to P_i is caused by b, then $d_i(C) = d_{i-1}(C)$ (because the relative location of the coins does not change) and $m_i(C) = m_{i-1}(C) - 1$ if $m_{i-1}(C) > 0$, otherwise $m_i(C) = n - 1$.

We see that

$$\mathrm{wg}(P_i) \geq \mathrm{wg}(C, P_i) = n \cdot d_i(C) + m_i(C) \geq$$
$$n \cdot d_{i-1}(C) + m_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1.$$

Properties of the Weight Function

In the final position only the golden coin G remains, whence the weight of $P_{|w|}$ is the weight of G. Clearly, $\mathrm{wg}(G, P_i) = m_i(G) \leq n - 1$ for any position P_i.

Let C be a coin of maximum weight in P_{i-1}. If the transition from P_{i-1} to P_i is caused by b, then $d_i(C) = d_{i-1}(C)$ (because the relative location of the coins does not change) and $m_i(C) = m_{i-1}(C) - 1$ if $m_{i-1}(C) > 0$, otherwise $m_i(C) = n - 1$. We see that

$$\mathrm{wg}(P_i) \geq \mathrm{wg}(C, P_i) = n \cdot d_i(C) + m_i(C) \geq$$
$$n \cdot d_{i-1}(C) + m_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1.$$

Properties of the Weight Function

Suppose the transition from P_{i-1} to P_i is caused by a. If C does not cover 0 in P_{i-1}, then $m_i(C) = m_{i-1}(C)$ and $d_i(C) = d_{i-1}(C)$ if G does not cover 0 in P_{i-1}, otherwise $d_i(C) = d_{i-1}(C) + 1$. Thus, the transition from P_{i-1} to P_i cannot decrease the weight.

Assume that C covers 0 in P_{i-1}. Then in P_i the state 1 holds a coin C' (which may or may not coincide with C). In P_{i-1} the golden coin G does not cover 0 whence it does not move and $d_i(C') = d_{i-1}(C) - 1$. Therefore

$$\begin{aligned}\mathrm{wg}(P_i) \geq \mathrm{wg}(C', P_i) &= n \cdot d_i(C') + n - 1 = n \cdot (d_{i-1}(C) - 1) + n - 1 \\ &= n \cdot d_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1.\end{aligned}$$

Properties of the Weight Function

Suppose the transition from P_{i-1} to P_i is caused by a. If C does not cover 0 in P_{i-1}, then $m_i(C) = m_{i-1}(C)$ and $d_i(C) = d_{i-1}(C)$ if G does not cover 0 in P_{i-1}, otherwise $d_i(C) = d_{i-1}(C) + 1$. Thus, the transition from P_{i-1} to P_i cannot decrease the weight.

Assume that C covers 0 in P_{i-1}. Then in P_i the state 1 holds a coin C' (which may or may not coincide with C). In P_{i-1} the golden coin G does not cover 0 whence it does not move and $d_i(C') = d_{i-1}(C) - 1$. Therefore

$$\begin{aligned} \mathrm{wg}(P_i) \geq \mathrm{wg}(C', P_i) &= n \cdot d_i(C') + n - 1 = n \cdot (d_{i-1}(C) - 1) + n - 1 \\ &= n \cdot d_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1. \end{aligned}$$

Properties of the Weight Function

Suppose the transition from P_{i-1} to P_i is caused by a. If C does not cover 0 in P_{i-1}, then $m_i(C) = m_{i-1}(C)$ and $d_i(C) = d_{i-1}(C)$ if G does not cover 0 in P_{i-1}, otherwise $d_i(C) = d_{i-1}(C) + 1$. Thus, the transition from P_{i-1} to P_i cannot decrease the weight.

Assume that C covers 0 in P_{i-1}. Then in P_i the state 1 holds a coin C' (which may or may not coincide with C). In P_{i-1} the golden coin G does not cover 0 whence it does not move and $d_i(C') = d_{i-1}(C) - 1$. Therefore

$$\begin{aligned}\mathrm{wg}(P_i) \geq \mathrm{wg}(C', P_i) &= n \cdot d_i(C') + n - 1 = n \cdot (d_{i-1}(C) - 1) + n - 1 \\ &= n \cdot d_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1.\end{aligned}$$

Properties of the Weight Function

Suppose the transition from P_{i-1} to P_i is caused by a. If C does not cover 0 in P_{i-1}, then $m_i(C) = m_{i-1}(C)$ and $d_i(C) = d_{i-1}(C)$ if G does not cover 0 in P_{i-1}, otherwise $d_i(C) = d_{i-1}(C) + 1$. Thus, the transition from P_{i-1} to P_i cannot decrease the weight.

Assume that C covers 0 in P_{i-1}. Then in P_i the state 1 holds a coin C' (which may or may not coincide with C). In P_{i-1} the golden coin G does not cover 0 whence it does not move and $d_i(C') = d_{i-1}(C) - 1$. Therefore

$$\begin{aligned}\mathrm{wg}(P_i) \geq \mathrm{wg}(C', P_i) &= n \cdot d_i(C') + n - 1 = n \cdot (d_{i-1}(C) - 1) + n - 1 \\ &= n \cdot d_{i-1}(C) - 1 = \mathrm{wg}(C, P_{i-1}) - 1 = \mathrm{wg}(P_{i-1}) - 1.\end{aligned}$$

A Two-Players Game

Assume that there are two players: Alice (Synchronizer) and Bob (Desynchronizer) whose moves alternate. Alice (who plays first) wants to synchronize the given automaton, Bob aims to make her task as hard as possible.

Provided that both players play optimally, the outcome of such a game depends only on the underlying automaton so studying synchronization games is a way to study synchronizing automata. The most natural questions here are the following:

- Clearly, Bob wins on non-synchronizing automata. May he win on a synchronizing automaton?
- Given a DFA $\mathscr{A}$, how and how fast can one decide who wins on $\mathscr{A}$?
- If Alice wins, how long can be the game?

A Two-Players Game

Assume that there are two players: Alice (Synchronizer) and Bob (Desynchronizer) whose moves alternate. Alice (who plays first) wants to synchronize the given automaton, Bob aims to make her task as hard as possible.

Provided that both players play optimally, the outcome of such a game depends only on the underlying automaton so studying synchronization games is a way to study synchronizing automata. The most natural questions here are the following:

- Clearly, Bob wins on non-synchronizing automata. May he win on a synchronizing automaton?
- Given a DFA $\mathscr{A}$, how and how fast can one decide who wins on $\mathscr{A}$?
- If Alice wins, how long can be the game?

A Two-Players Game

Assume that there are two players: Alice (Synchronizer) and Bob (Desynchronizer) whose moves alternate. Alice (who plays first) wants to synchronize the given automaton, Bob aims to make her task as hard as possible.

Provided that both players play optimally, the outcome of such a game depends only on the underlying automaton so studying synchronization games is a way to study synchronizing automata. The most natural questions here are the following:

- Clearly, Bob wins on non-synchronizing automata. May he win on a synchronizing automaton?
- Given a DFA $\mathscr{A}$, how and how fast can one decide who wins on $\mathscr{A}$?
- If Alice wins, how long can be the game?

A Two-Players Game

Assume that there are two players: Alice (Synchronizer) and Bob (Desynchronizer) whose moves alternate. Alice (who plays first) wants to synchronize the given automaton, Bob aims to make her task as hard as possible.

Provided that both players play optimally, the outcome of such a game depends only on the underlying automaton so studying synchronization games is a way to study synchronizing automata. The most natural questions here are the following:

- Clearly, Bob wins on non-synchronizing automata. May he win on a synchronizing automaton?
- Given a DFA $\mathscr{A}$, how and how fast can one decide who wins on $\mathscr{A}$?
- If Alice wins, how long can be the game?

A Two-Players Game

Assume that there are two players: Alice (Synchronizer) and Bob (Desynchronizer) whose moves alternate. Alice (who plays first) wants to synchronize the given automaton, Bob aims to make her task as hard as possible.

Provided that both players play optimally, the outcome of such a game depends only on the underlying automaton so studying synchronization games is a way to study synchronizing automata. The most natural questions here are the following:

- Clearly, Bob wins on non-synchronizing automata. May he win on a synchronizing automaton?
- Given a DFA $\mathscr{A}$, how and how fast can one decide who wins on $\mathscr{A}$?
- If Alice wins, how long can be the game?

A Two-Players Game

Assume that there are two players: Alice (Synchronizer) and Bob (Desynchronizer) whose moves alternate. Alice (who plays first) wants to synchronize the given automaton, Bob aims to make her task as hard as possible.

Provided that both players play optimally, the outcome of such a game depends only on the underlying automaton so studying synchronization games is a way to study synchronizing automata. The most natural questions here are the following:

- Clearly, Bob wins on non-synchronizing automata. May he win on a synchronizing automaton?
- Given a DFA $\mathscr{A}$, how and how fast can one decide who wins on $\mathscr{A}$?
- If Alice wins, how long can be the game?

The answers to the above questions have been found by Födor Fominykh.

- Bob can win on a synchronizing automaton (for instance, he wins on $\mathscr{C}_n$).
- Given $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can decide who wins on $\mathscr{A}$ in $O(|Q|^2 \cdot |\Sigma|)$ time.
- If Alice wins, then she can win in $O(|Q|^3|)$ moves.
- For every synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can construct an automaton $\mathscr{A}'$ with $2|Q|$ states such that Alice wins on $\mathscr{A}'$ but the minimum number of moves she needs to win is no less than the minimum length of reset words for $\mathscr{A}$.

A Two-Players Game: Answers

The answers to the above questions have been found by Födor Fominykh.

- Bob can win on a synchronizing automaton (for instance, he wins on $\mathscr{C}_n$).
- Given $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can decide who wins on $\mathscr{A}$ in $O(|Q|^2 \cdot |\Sigma|)$ time.
- If Alice wins, then she can win in $O(|Q|^3|)$ moves.
- For every synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can construct an automaton $\mathscr{A}'$ with $2|Q|$ states such that Alice wins on $\mathscr{A}'$ but the minimum number of moves she needs to win is no less than the minimum length of reset words for $\mathscr{A}$.

A Two-Players Game: Answers

The answers to the above questions have been found by Födor Fominykh.

- Bob can win on a synchronizing automaton (for instance, he wins on $\mathscr{C}_n$).
- Given $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$, one can decide who wins on $\mathscr{A}$ in $O(|Q|^2 \cdot |\Sigma|)$ time.
- If Alice wins, then she can win in $O(|Q|^3|)$ moves.
- For every synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$, one can construct an automaton $\mathscr{A}'$ with $2|Q|$ states such that Alice wins on $\mathscr{A}'$ but the minimum number of moves she needs to win is no less than the minimum length of reset words for $\mathscr{A}$.

The answers to the above questions have been found by Födor Fominykh.

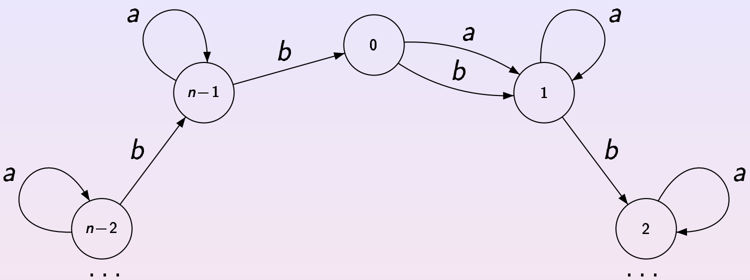
- Bob can win on a synchronizing automaton (for instance, he wins on $\mathscr{C}_n$).
- Given $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can decide who wins on $\mathscr{A}$ in $O(|Q|^2 \cdot |\Sigma|)$ time.
- If Alice wins, then she can win in $O(|Q|^3|)$ moves.
- For every synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can construct an automaton $\mathscr{A}'$ with $2|Q|$ states such that Alice wins on $\mathscr{A}'$ but the minimum number of moves she needs to win is no less than the minimum length of reset words for $\mathscr{A}$.

A Two-Players Game: Answers

The answers to the above questions have been found by Födor Fominykh.

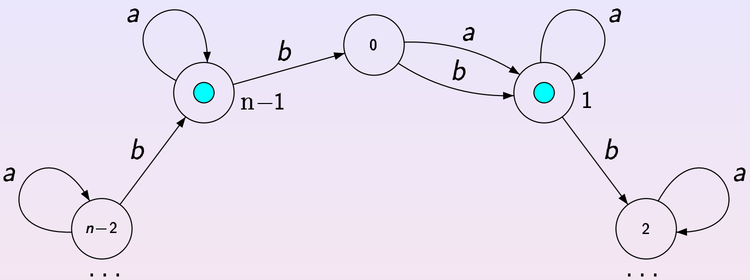
- Bob can win on a synchronizing automaton (for instance, he wins on $\mathscr{C}_n$).
- Given $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can decide who wins on $\mathscr{A}$ in $O(|Q|^2 \cdot |\Sigma|)$ time.
- If Alice wins, then she can win in $O(|Q|^3|)$ moves.
- For every synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, one can construct an automaton $\mathscr{A}'$ with $2|Q|$ states such that Alice wins on $\mathscr{A}'$ but the minimum number of moves she needs to win is no less than the minimum length of reset words for $\mathscr{A}$.

Bob Wins on $\mathscr{C}_n$

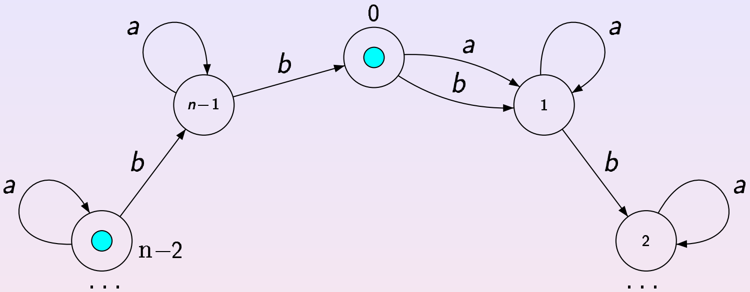


Bob only has to trace the coins that cover the states $n-1$ and 1 in the initial position. He can always say a except two cases: when the chosen coins cover either $n-2$ and 0 or 0 and 2 in which cases Bob says b. This way Bob can always keep the coins two steps apart from each other whence Alice can remove none of them.

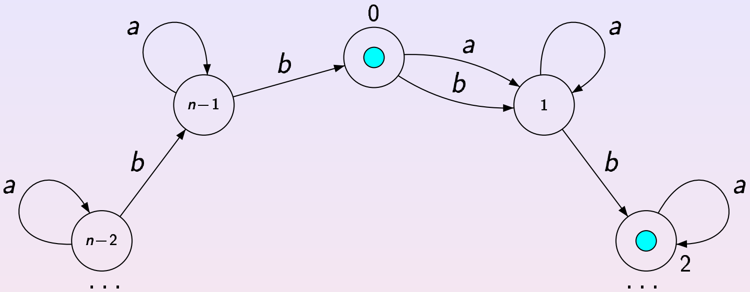
Bob Wins on $\mathscr{C}_n$



Bob only has to trace the coins that cover the states $n-1$ and 1 in the initial position. He can always say a except two cases: when the chosen coins cover either $n-2$ and 0 or 0 and 2 in which cases Bob says b. This way Bob can always keep the coins two steps apart from each other whence Alice can remove none of them.

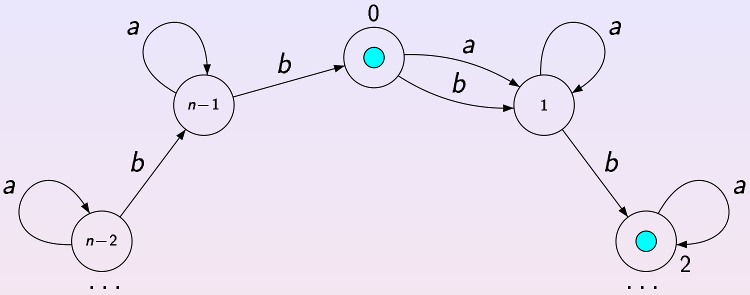


Bob only has to trace the coins that cover the states $n-1$ and 1 in the initial position. He can always say a except two cases: when the chosen coins cover either $n-2$ and 0 or 0 and 2 in which cases Bob says b. This way Bob can always keep the coins two steps apart from each other whence Alice can remove none of them.



Bob only has to trace the coins that cover the states $n-1$ and 1 in the initial position. He can always say a except two cases: when the chosen coins cover either $n-2$ and 0 or 0 and 2 in which cases Bob says b. This way Bob can always keep the coins two steps apart from each other whence Alice can remove none of them.

Bob Wins on $\mathscr{C}_n$



Bob only has to trace the coins that cover the states $n-1$ and 1 in the initial position. He can always say a except two cases: when the chosen coins cover either $n-2$ and 0 or 0 and 2 in which cases Bob says b. This way Bob can always keep the coins two steps apart from each other whence Alice can remove none of them.

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

The claim that one can decide who wins on $\mathscr{A}\langle Q, \Sigma, \delta \rangle$ in $O(|Q|^2 \cdot |\Sigma|)$ time and the claim that if Alice wins, then she can do this in $O(|Q|^3|)$ moves both follow from the next easy fact:

Proposition. *Alice wins the game on a DFA $\mathscr{A}$ iff she wins in every position with only two coins.*

Then we construct a new automaton whose states are all $O(|Q|^2)$ positions with two coins plus a sink state (corresponding to all positions with one coin) and mark its states in a standard way: A state p is an Alice state if either Alice can reach the sink state from p by a single move or she has a move leading to a position in which every Bob's reply leads to an Alice state. The marking is basically a BFS on the reverse graph, and Alice wins iff all states will eventually be marked as Alice states.

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

The claim that one can decide who wins on $\mathscr{A}\langle Q, \Sigma, \delta \rangle$ in $O(|Q|^2 \cdot |\Sigma|)$ time and the claim that if Alice wins, then she can do this in $O(|Q|^3|)$ moves both follow from the next easy fact:

Proposition. *Alice wins the game on a DFA $\mathscr{A}$ iff she wins in every position with only two coins.*

Then we construct a new automaton whose states are all $O(|Q|^2)$ positions with two coins plus a sink state (corresponding to all positions with one coin) and mark its states in a standard way: A state p is an Alice state if either Alice can reach the sink state from p by a single move or she has a move leading to a position in which every Bob's reply leads to an Alice state. The marking is basically a BFS on the reverse graph, and Alice wins iff all states will eventually be marked as Alice states.

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

The claim that one can decide who wins on $\mathscr{A}\langle Q, \Sigma, \delta \rangle$ in $O(|Q|^2 \cdot |\Sigma|)$ time and the claim that if Alice wins, then she can do this in $O(|Q|^3|)$ moves both follow from the next easy fact:

Proposition. *Alice wins the game on a DFA $\mathscr{A}$ iff she wins in every position with only two coins.*

Then we construct a new automaton whose states are all $O(|Q|^2)$ positions with two coins plus a sink state (corresponding to all positions with one coin) and mark its states in a standard way: A state p is an Alice state if either Alice can reach the sink state from p by a single move or she has a move leading to a position in which every Bob's reply leads to an Alice state. The marking is basically a BFS on the reverse graph, and Alice wins iff all states will eventually be marked as Alice states.

Upper Bounds for Decision Time and Game Length

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

The claim that one can decide who wins on $\mathscr{A}\langle Q, \Sigma, \delta \rangle$ in $O(|Q|^2 \cdot |\Sigma|)$ time and the claim that if Alice wins, then she can do this in $O(|Q|^3|)$ moves both follow from the next easy fact:

Proposition. *Alice wins the game on a DFA $\mathscr{A}$ iff she wins in every position with only two coins.*

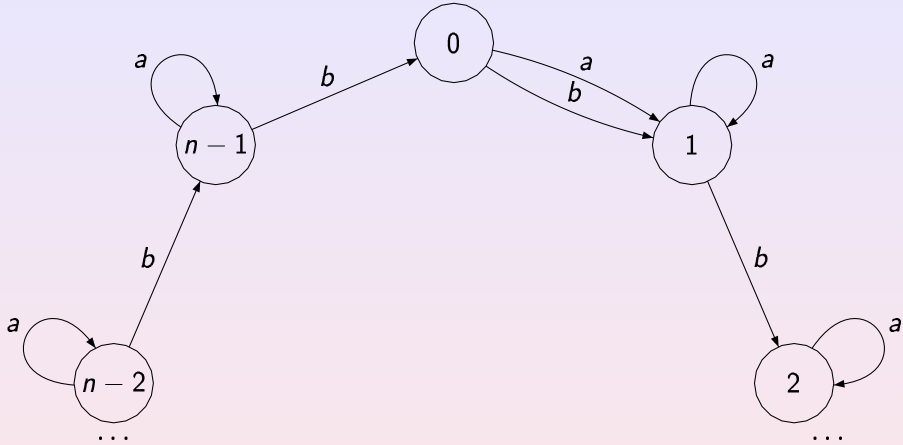
Then we construct a new automaton whose states are all $O(|Q|^2)$ positions with two coins plus a sink state (corresponding to all positions with one coin) and mark its states in a standard way: A state p is an Alice state if either Alice can reach the sink state from p by a single move or she has a move leading to a position in which every Bob's reply leads to an Alice state. The marking is basically a BFS on the reverse graph, and Alice wins iff all states will eventually be marked as Alice states.

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

The claim that one can decide who wins on $\mathscr{A}\langle Q, \Sigma, \delta \rangle$ in $O(|Q|^2 \cdot |\Sigma|)$ time and the claim that if Alice wins, then she can do this in $O(|Q|^3|)$ moves both follow from the next easy fact:

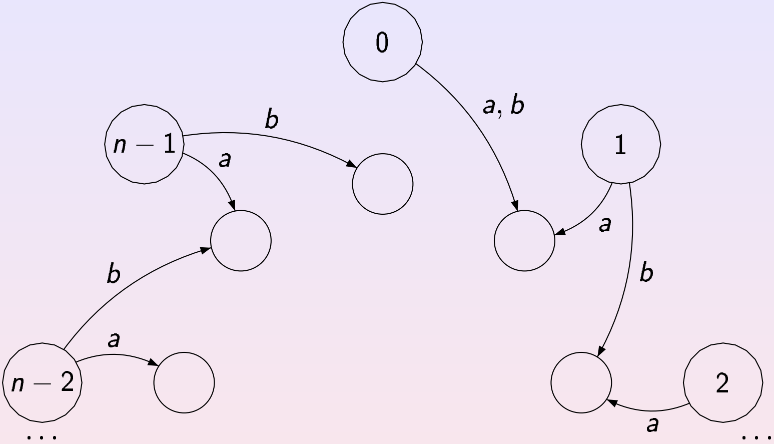
Proposition. *Alice wins the game on a DFA $\mathscr{A}$ iff she wins in every position with only two coins.*

Then we construct a new automaton whose states are all $O(|Q|^2)$ positions with two coins plus a sink state (corresponding to all positions with one coin) and mark its states in a standard way: A state p is an Alice state if either Alice can reach the sink state from p by a single move or she has a move leading to a position in which every Bob's reply leads to an Alice state. The marking is basically a BFS on the reverse graph, and Alice wins iff all states will eventually be marked as Alice states.



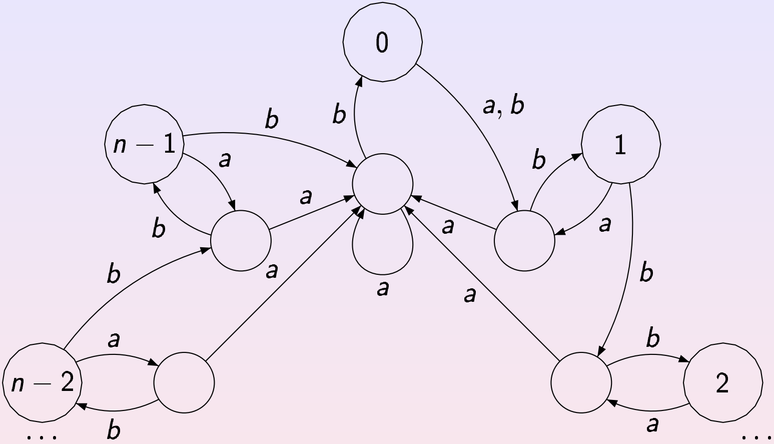
We start with the Černý automaton $\mathscr{C}_n$

Long Games

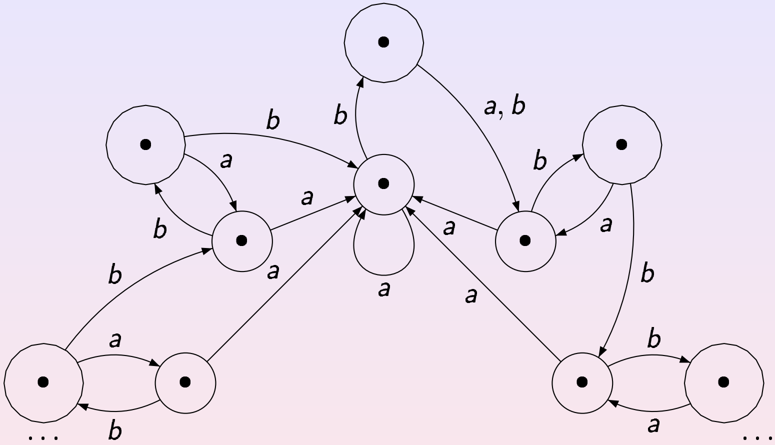


We start with the Černý automaton $\mathscr{C}_n$ and modify it as shown by doubling the states and mimicking the transitions.

Long Games

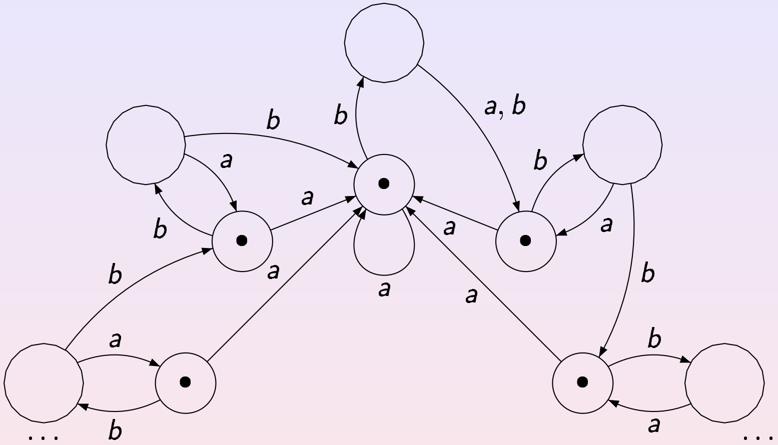


We start with the Černý automaton $\mathscr{C}_n$ and modify it as shown by doubling the states and mimicking the transitions.

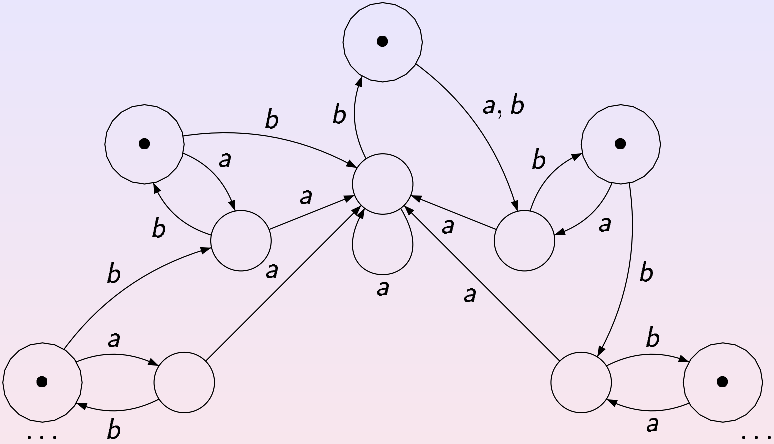


The initial position.

Long Games

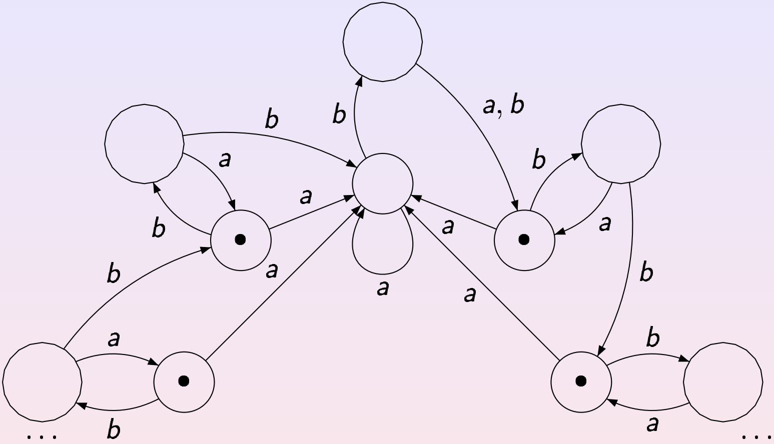


Alice says a.



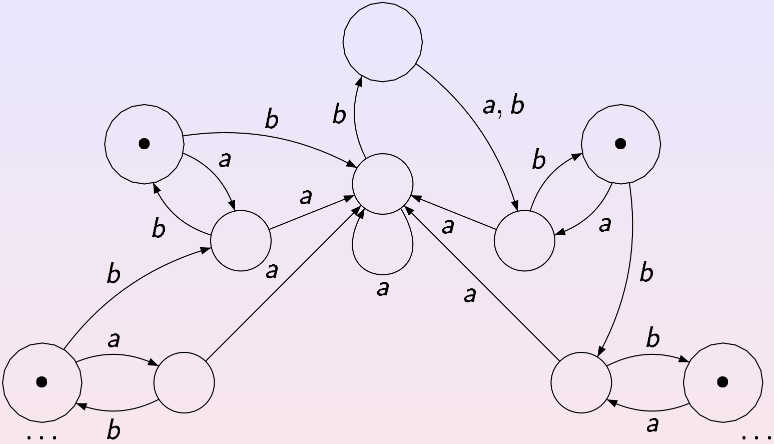
Bob must reply b otherwise he loses immediately. Now the position imitates the initial position in the game on $\mathscr{C}_n$.

Long Games



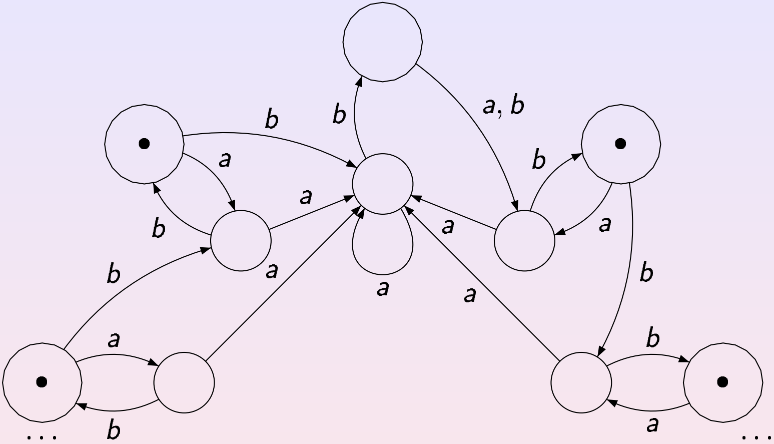
Alice says a again.

Long Games



Bob must reply b otherwise he loses immediately. Now the position imitates the one after the first move in the one-player game on $\mathscr{C}_n$.

Long Games



Continuing, we see that Alice wins by spelling out the reset word for $\mathscr{C}_n$ but cannot win faster if Bob replies b on each move.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

LMRC, February 2nd, 2012

A Corollary and Further Games

We can register the following rather unexpected corollary:

If Alice has an $O(n^2)$-strategy for each n-state automaton with a reset word of length 2 on which she can win, then there is a quadratic upper bound in the Černý problem.

Other possible synchronization games include:

- Games against the Nature (Nature replies by random moves) — see Andreas Blass, Yuri Gurevich, Lev Nachmanson, Margus Veanes: Play to Test, in: Formal Approaches to Software Testing, 5th International Workshop, FATES 2005 (Lect. Notes Comp. Sci., vol. 3997), Springer, 2006, 32–46.
- Games on non-deterministic automata.
- Road Coloring games: Alice and Bob alternatively color edges of a given primitive digraph aiming at a synchronizing / non-synchronizing automaton.

Synchronization Costs

Now let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a synchronizing automaton in which every transition has a cost (a positive integer). More formally, $\mathscr{A}$ is equipped with an extra function $\gamma : Q \times \Sigma \to \mathbb{Z}_+$.

For $w = a_1 \cdots a_k \in \Sigma^*$ and $q \in Q$, the cost of applying w at q is

$$\gamma(q, w) = \sum_{i=0}^{k-1} \gamma(\delta(q, a_1 \cdots a_i), a_{i+1}).$$

If w is a reset word for $\mathscr{A}$, the cost of synchronizing $\mathscr{A}$ by w is

$$\gamma(w) = \max_{q \in Q} \gamma(q, w).$$

(Since we do not know the state of $\mathscr{A}$ prior to synchronization, we are forced to take the most costly case into account.)

Synchronization Costs

Now let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a synchronizing automaton in which every transition has a cost (a positive integer). More formally, $\mathscr{A}$ is equipped with an extra function $\gamma : Q \times \Sigma \to \mathbb{Z}_+$.

For $w = a_1 \cdots a_k \in \Sigma^*$ and $q \in Q$, the cost of applying w at q is

$$\gamma(q, w) = \sum_{i=0}^{k-1} \gamma\big(\delta(q, a_1 \cdots a_i), a_{i+1}\big).$$

If w is a reset word for $\mathscr{A}$, the cost of synchronizing $\mathscr{A}$ by w is

$$\gamma(w) = \max_{q \in Q} \gamma(q, w).$$

(Since we do not know the state of $\mathscr{A}$ prior to synchronization, we are forced to take the most costly case into account.)

Now let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a synchronizing automaton in which every transition has a cost (a positive integer). More formally, $\mathscr{A}$ is equipped with an extra function $\gamma : Q \times \Sigma \to \mathbb{Z}_+$.

For $w = a_1 \cdots a_k \in \Sigma^*$ and $q \in Q$, the cost of applying w at q is

$$\gamma(q, w) = \sum_{i=0}^{k-1} \gamma\big(\delta(q, a_1 \cdots a_i), a_{i+1}\big).$$

If w is a reset word for $\mathscr{A}$, the cost of synchronizing $\mathscr{A}$ by w is

$$\gamma(w) = \max_{q \in Q} \gamma(q, w).$$

(Since we do not know the state of $\mathscr{A}$ prior to synchronization, we are forced to take the most costly case into account.)

Synchronization Costs

Now let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a synchronizing automaton in which every transition has a cost (a positive integer). More formally, $\mathscr{A}$ is equipped with an extra function $\gamma : Q \times \Sigma \to \mathbb{Z}_+$.

For $w = a_1 \cdots a_k \in \Sigma^*$ and $q \in Q$, the cost of applying w at q is

$$\gamma(q, w) = \sum_{i=0}^{k-1} \gamma\big(\delta(q, a_1 \cdots a_i), a_{i+1}\big).$$

If w is a reset word for $\mathscr{A}$, the cost of synchronizing $\mathscr{A}$ by w is

$$\gamma(w) = \max_{q \in Q} \gamma(q, w).$$

(Since we do not know the state of $\mathscr{A}$ prior to synchronization, we are forced to take the most costly case into account.)

Now let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a synchronizing automaton in which every transition has a cost (a positive integer). More formally, $\mathscr{A}$ is equipped with an extra function $\gamma : Q \times \Sigma \to \mathbb{Z}_+$.

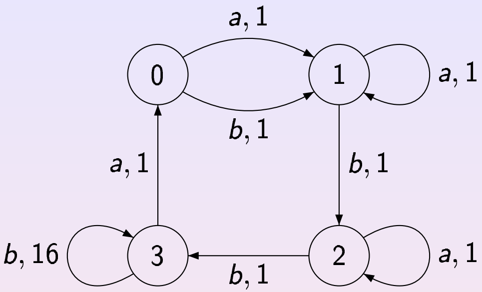
For $w = a_1 \cdots a_k \in \Sigma^*$ and $q \in Q$, the cost of applying w at q is

$$\gamma(q, w) = \sum_{i=0}^{k-1} \gamma\big(\delta(q, a_1 \cdots a_i), a_{i+1}\big).$$

If w is a reset word for $\mathscr{A}$, the cost of synchronizing $\mathscr{A}$ by w is

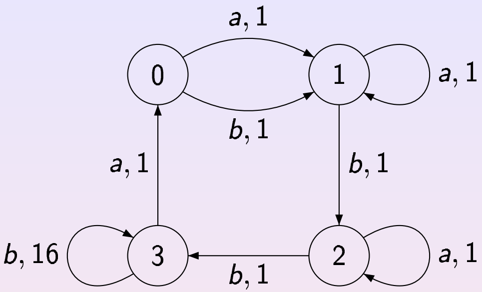
$$\gamma(w) = \max_{q \in Q} \gamma(q, w).$$

(Since we do not know the state of $\mathscr{A}$ prior to synchronization, we are forced to take the most costly case into account.)



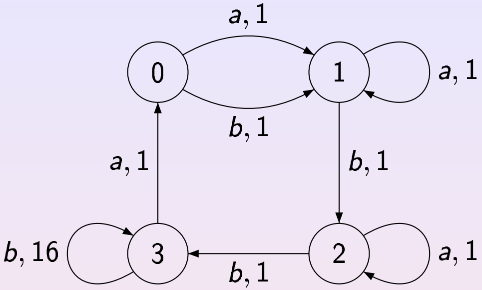
One sees that the shortest reset word for this automaton is b^3 but the cost of synchronizing by b^3 is 48. On the other hand, the word a^2baba^2 which is much longer manages to completely avoid the 'bad' loop at the state 3 whence the cost of synchronizing by a^2baba^2 is only 7.

An Example



One sees that the shortest reset word for this automaton is b^3 but the cost of synchronizing by b^3 is 48. On the other hand, the word a^2baba^2 which is much longer manages to completely avoid the 'bad' loop at the state 3 whence the cost of synchronizing by a^2baba^2 is only 7.

An Example



One sees that the shortest reset word for this automaton is b^3 but the cost of synchronizing by b^3 is 48. On the other hand, the word a^2baba^2 which is much longer manages to completely avoid the 'bad' loop at the state 3 whence the cost of synchronizing by a^2baba^2 is only 7.

Consider the following decision problem:

SYNCHRONIZING ON BUDGET: *Given a synchronizing automaton* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *with the cost function* γ *and a positive integer* B, *is it true that* $\mathscr{A}$ *has a reset word with* $\gamma(w) \leq B$*?*

Recall that the following problem SHORT-RESET-WORD is known to be NP-complete:

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

Clearly, SHORT-RESET-WORD is a special case of SYNCHRONIZING ON BUDGET when $\gamma(q, a) = 1$ for all $q \in Q$ and $a \in \Sigma$. Thus, SYNCHRONIZING ON BUDGET is NP-hard.

Consider the following decision problem:

SYNCHRONIZING ON BUDGET: *Given a synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ with the cost function γ and a positive integer B, is it true that $\mathscr{A}$ has a reset word with $\gamma(w) \leq B$?*

Recall that the following problem SHORT-RESET-WORD is known to be NP-complete:

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

Clearly, SHORT-RESET-WORD is a special case of SYNCHRONIZING ON BUDGET when $\gamma(q, a) = 1$ for all $q \in Q$ and $a \in \Sigma$. Thus, SYNCHRONIZING ON BUDGET is NP-hard.

Consider the following decision problem:

SYNCHRONIZING ON BUDGET: *Given a synchronizing automaton* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *with the cost function* γ *and a positive integer* B*, is it true that* $\mathscr{A}$ *has a reset word with* $\gamma(w) \leq B$*?*

Recall that the following problem SHORT-RESET-WORD is known to be NP-complete:

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

Clearly, SHORT-RESET-WORD is a special case of SYNCHRONIZING ON BUDGET when $\gamma(q, a) = 1$ for all $q \in Q$ and $a \in \Sigma$. Thus, SYNCHRONIZING ON BUDGET is NP-hard.

Consider the following decision problem:

Synchronizing on Budget: *Given a synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ with the cost function γ and a positive integer B, is it true that $\mathscr{A}$ has a reset word with $\gamma(w) \leq B$?*

Recall that the following problem Short-Reset-Word is known to be NP-complete:

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

Clearly, Short-Reset-Word is a special case of Synchronizing on Budget when $\gamma(q, a) = 1$ for all $q \in Q$ and $a \in \Sigma$. Thus, Synchronizing on Budget is NP-hard.

Easy Case

If the transition costs and the budget are given in unary, then SYNCHRONIZING ON BUDGET is in NP.

One can non-deterministically guess a word $w \in \Sigma^*$ of length $\ell \leq B$ and then check if w is a reset word for $\mathscr{A}$ with $\gamma(w) \leq B$. This can be done in $\ell B|Q|$ time.

Thus, in this case SYNCHRONIZING ON BUDGET is NP-complete.

Easy Case

If the transition costs and the budget are given in unary, then SYNCHRONIZING ON BUDGET is in NP.
One can non-deterministically guess a word $w \in \Sigma^*$ of length $\ell \leq B$ and then check if w is a reset word for $\mathscr{A}$ with $\gamma(w) \leq B$. This can be done in $\ell B|Q|$ time.

Thus, in this case SYNCHRONIZING ON BUDGET is NP-complete.

Easy Case

If the transition costs and the budget are given in unary, then SYNCHRONIZING ON BUDGET is in NP.
One can non-deterministically guess a word $w \in \Sigma^*$ of length $\ell \leq B$ and then check if w is a reset word for $\mathscr{A}$ with $\gamma(w) \leq B$.
This can be done in $\ell B|Q|$ time.

Thus, in this case SYNCHRONIZING ON BUDGET is NP-complete.

Hard Case

Now assume that the transition costs $\gamma(q, a)$ and the budget B are given in binary. Then one can show that for some synchronizing automata any reset word satisfying $\gamma(w) \leq B$ is exponentially long in $|Q|$. Therefore the above non-deterministic algorithm is not polynomial in time.

However, it can be implemented in polynomial space if one guesses the word w letter by letter. One guesses the first letter of w (say, a), apply a at every state $q \in Q$ and save two arrays: $\{\delta(q, a)\}$ and $\{\gamma(q, a)\}$. Then one guesses the second letter of w and updates both arrays, etc.

Thus, in this case Synchronizing on Budget is in PSPACE.

Hard Case

Now assume that the transition costs $\gamma(q, a)$ and the budget B are given in binary. Then one can show that for some synchronizing automata any reset word satisfying $\gamma(w) \leq B$ is exponentially long in $|Q|$. Therefore the above non-deterministic algorithm is not polynomial in time.

However, it can be implemented in polynomial space if one guesses the word w letter by letter. One guesses the first letter of w (say, a), apply a at every state $q \in Q$ and save two arrays: $\{\delta(q, a)\}$ and $\{\gamma(q, a)\}$. Then one guesses the second letter of w and updates both arrays, etc.

Thus, in this case SYNCHRONIZING ON BUDGET is in PSPACE.

LMRC, February 2nd, 2012

Hard Case

Now assume that the transition costs $\gamma(q,a)$ and the budget B are given in binary. Then one can show that for some synchronizing automata any reset word satisfying $\gamma(w) \leq B$ is exponentially long in $|Q|$. Therefore the above non-deterministic algorithm is not polynomial in time.

However, it can be implemented in polynomial space if one guesses the word w letter by letter. One guesses the first letter of w (say, a), apply a at every state $q \in Q$ and save two arrays: $\{\delta(q,a)\}$ and $\{\gamma(q,a)\}$. Then one guesses the second letter of w and updates both arrays, etc.

Thus, in this case Synchronizing on Budget is in PSPACE.

Hard Case

Now assume that the transition costs $\gamma(q,a)$ and the budget B are given in binary. Then one can show that for some synchronizing automata any reset word satisfying $\gamma(w) \leq B$ is exponentially long in $|Q|$. Therefore the above non-deterministic algorithm is not polynomial in time.

However, it can be implemented in polynomial space if one guesses the word w letter by letter. One guesses the first letter of w (say, a), apply a at every state $q \in Q$ and save two arrays: $\{\delta(q,a)\}$ and $\{\gamma(q,a)\}$. Then one guesses the second letter of w and updates both arrays, etc.

Thus, in this case Synchronizing on Budget is in PSPACE.

Hard Case

Now assume that the transition costs $\gamma(q, a)$ and the budget B are given in binary. Then one can show that for some synchronizing automata any reset word satisfying $\gamma(w) \leq B$ is exponentially long in $|Q|$. Therefore the above non-deterministic algorithm is not polynomial in time.

However, it can be implemented in polynomial space if one guesses the word w letter by letter. One guesses the first letter of w (say, a), apply a at every state $q \in Q$ and save two arrays: $\{\delta(q, a)\}$ and $\{\gamma(q, a)\}$. Then one guesses the second letter of w and updates both arrays, etc.

Thus, in this case SYNCHRONIZING ON BUDGET is in PSPACE.

Hard Case

Now assume that the transition costs $\gamma(q, a)$ and the budget B are given in binary. Then one can show that for some synchronizing automata any reset word satisfying $\gamma(w) \leq B$ is exponentially long in $|Q|$. Therefore the above non-deterministic algorithm is not polynomial in time.

However, it can be implemented in polynomial space if one guesses the word w letter by letter. One guesses the first letter of w (say, a), apply a at every state $q \in Q$ and save two arrays: $\{\delta(q, a)\}$ and $\{\gamma(q, a)\}$. Then one guesses the second letter of w and updates both arrays, etc.

Thus, in this case Synchronizing on Budget is in PSPACE.

We show that Synchronizing on Budget is PSPACE-complete by a reduction from Careful Synchronization.

Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Sigma$ such that:

1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.

We show that SYNCHRONIZING ON BUDGET is PSPACE-complete by a reduction from Careful Synchronization.

Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Sigma$ such that:

1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.

PSPACE-Completeness

We show that SYNCHRONIZING ON BUDGET is PSPACE-complete by a reduction from Careful Synchronization.

Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Sigma$ such that:

1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.

We show that SYNCHRONIZING ON BUDGET is PSPACE-complete by a reduction from Careful Synchronization.

Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \Sigma$ such that:

1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.

PSPACE-Completeness

We show that SYNCHRONIZING ON BUDGET is PSPACE-complete by a reduction from Careful Synchronization.

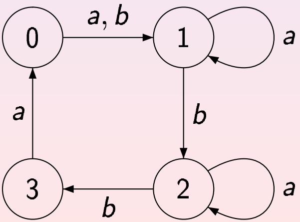
Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta\rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \Sigma$ such that:

1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.

We show that SYNCHRONIZING ON BUDGET is PSPACE-complete by a reduction from Careful Synchronization.

Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Sigma$ such that:

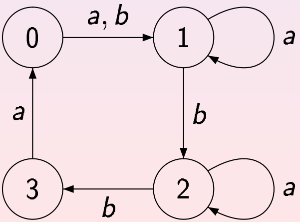
1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.



We show that SYNCHRONIZING ON BUDGET is PSPACE-complete by a reduction from Careful Synchronization.

Recall that an incomplete deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ is said to be carefully synchronizing if there exists $w = a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \Sigma$ such that:

1) $\delta(q, a_1)$ is defined for all $q \in Q$,
2) $\delta(q, a_i)$ with $1 < i \leq \ell$ is defined for all $q \in \delta(Q, a_1 \cdots a_{i-1})$,
3) $|\delta(Q, w)| = 1$.



A careful reset word is a^2baba^2.

Theorem (Martyugin, 2010)

Checking if a given incomplete DFA is carefully synchronizing is PSPACE-complete.

There is also an obvious upper bound $2^n - n - 1$ on the minimum length of the shortest careful reset word for carefully synchronizing automata with n states.

It comes from the power automaton $\mathcal{P}'(\mathscr{A})$ of a given incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

- states are the non-empty subsets of Q,
- $\Delta(P, a) = \{\delta(p, a) \mid p \in P\}$ provided $\delta(p, a)$ is defined for each $p \in P$; otherwise $\delta(P, a)$ is undefined.

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

Theorem (Martyugin, 2010)

Checking if a given incomplete DFA is carefully synchronizing is PSPACE-complete.

There is also an obvious upper bound $2^n - n - 1$ on the minimum length of the shortest careful reset word for carefully synchronizing automata with n states.

It comes from the power automaton $\mathcal{P}'(\mathscr{A})$ of a given incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

- states are the non-empty subsets of Q,
- $\Delta(P, a) = \{\delta(p, a) \mid p \in P\}$ provided $\delta(p, a)$ is defined for each $p \in P$; otherwise $\delta(P, a)$ is undefined.

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

PSPACE-Completeness

Theorem (Martyugin, 2010)

Checking if a given incomplete DFA is carefully synchronizing is PSPACE-complete.

There is also an obvious upper bound $2^n - n - 1$ on the minimum length of the shortest careful reset word for carefully synchronizing automata with n states.

It comes from the power automaton $\mathcal{P}'(\mathscr{A})$ of a given incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

- states are the non-empty subsets of Q,
- $\Delta(P, a) = \{\delta(p, a) \mid p \in P\}$ provided $\delta(p, a)$ is defined for each $p \in P$; otherwise $\delta(P, a)$ is undefined.

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

Theorem (Martyugin, 2010)

Checking if a given incomplete DFA is carefully synchronizing is PSPACE-complete.

There is also an obvious upper bound $2^n - n - 1$ on the minimum length of the shortest careful reset word for carefully synchronizing automata with n states.

It comes from the power automaton $\mathcal{P}'(\mathscr{A})$ of a given incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

- states are the non-empty subsets of Q,
- $\Delta(P, a) = \{\delta(p, a) \mid p \in P\}$ provided $\delta(p, a)$ is defined for each $p \in P$; otherwise $\delta(P, a)$ is undefined.

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

PSPACE-Completeness

Theorem (Martyugin, 2010)

Checking if a given incomplete DFA is carefully synchronizing is PSPACE-complete.

There is also an obvious upper bound $2^n - n - 1$ on the minimum length of the shortest careful reset word for carefully synchronizing automata with n states.

It comes from the power automaton $\mathcal{P}'(\mathscr{A})$ of a given incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

- states are the non-empty subsets of Q,
- $\Delta(P, a) = \{\delta(p, a) \mid p \in P\}$ provided $\delta(p, a)$ is defined for each $p \in P$; otherwise $\delta(P, a)$ is undefined.

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

Theorem (Martyugin, 2010)

Checking if a given incomplete DFA is carefully synchronizing is PSPACE-complete.

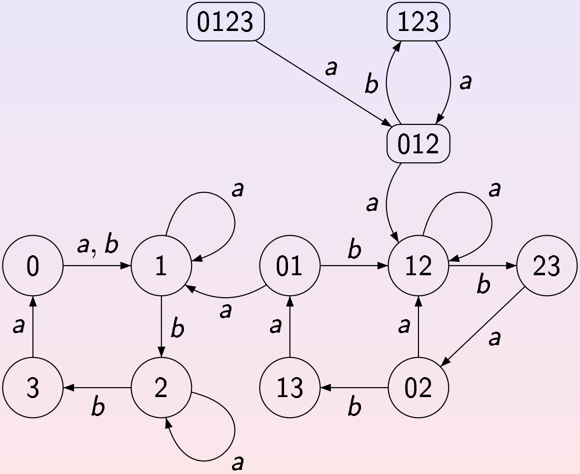
There is also an obvious upper bound $2^n - n - 1$ on the minimum length of the shortest careful reset word for carefully synchronizing automata with n states.

It comes from the power automaton $\mathcal{P}'(\mathscr{A})$ of a given incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:

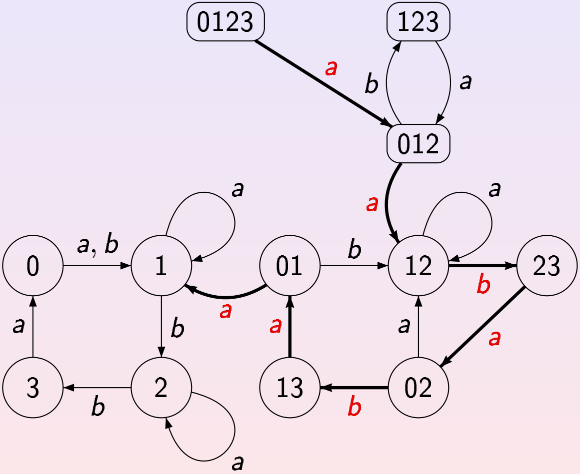
- states are the non-empty subsets of Q,
- $\Delta(P, a) = \{\delta(p, a) \mid p \in P\}$ provided $\delta(p, a)$ is defined for each $p \in P$; otherwise $\delta(P, a)$ is undefined.

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

An Example



An Example



A Reduction

Now, given an incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, we construct an instance $(\mathscr{A}', B)$ of Synchronizing on Budget as follows:

- $B = 2^n - 1$ where $n = |Q|$.
- $\mathscr{A}' = \langle Q, \Sigma, \delta' \rangle$ is $\mathscr{A}$ completed by letting

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{whenever } \delta(q,a) \text{ is defined,} \\ q & \text{otherwise.} \end{cases}$$

- Transition costs are defined by:

$$\gamma(q,a) = \begin{cases} 1 & \text{whenever } \delta(q,a) \text{ is defined,} \\ 2^n & \text{otherwise.} \end{cases}$$

A Reduction

Now, given an incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, we construct an instance $(\mathscr{A}', B)$ of Synchronizing on Budget as follows:

- $B = 2^n - 1$ where $n = |Q|$.
- $\mathscr{A}' = \langle Q, \Sigma, \delta' \rangle$ is $\mathscr{A}$ completed by letting

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{whenever } \delta(q,a) \text{ is defined,} \\ q & \text{otherwise.} \end{cases}$$

- Transition costs are defined by:

$$\gamma(q,a) = \begin{cases} 1 & \text{whenever } \delta(q,a) \text{ is defined,} \\ 2^n & \text{otherwise.} \end{cases}$$

A Reduction

Now, given an incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, we construct an instance $(\mathscr{A}', B)$ of Synchronizing on Budget as follows:

- $B = 2^n - 1$ where $n = |Q|$.
- $\mathscr{A}' = \langle Q, \Sigma, \delta' \rangle$ is $\mathscr{A}$ completed by letting

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{whenever } \delta(q,a) \text{ is defined,} \\ q & \text{otherwise.} \end{cases}$$

- Transition costs are defined by:

$$\gamma(q,a) = \begin{cases} 1 & \text{whenever } \delta(q,a) \text{ is defined,} \\ 2^n & \text{otherwise.} \end{cases}$$

A Reduction

Now, given an incomplete DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, we construct an instance $(\mathscr{A}', B)$ of SYNCHRONIZING ON BUDGET as follows:

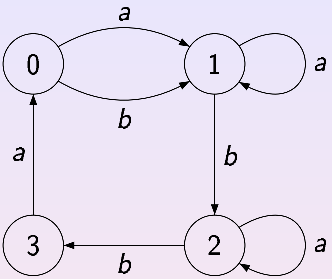
- $B = 2^n - 1$ where $n = |Q|$.
- $\mathscr{A}' = \langle Q, \Sigma, \delta' \rangle$ is $\mathscr{A}$ completed by letting

$$\delta'(q, a) = \begin{cases} \delta(q, a) & \text{whenever } \delta(q, a) \text{ is defined,} \\ q & \text{otherwise.} \end{cases}$$

- Transition costs are defined by:

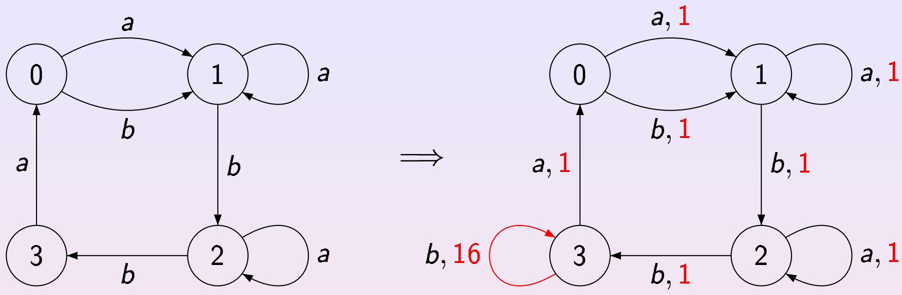
$$\gamma(q, a) = \begin{cases} 1 & \text{whenever } \delta(q, a) \text{ is defined,} \\ 2^n & \text{otherwise.} \end{cases}$$

An Example



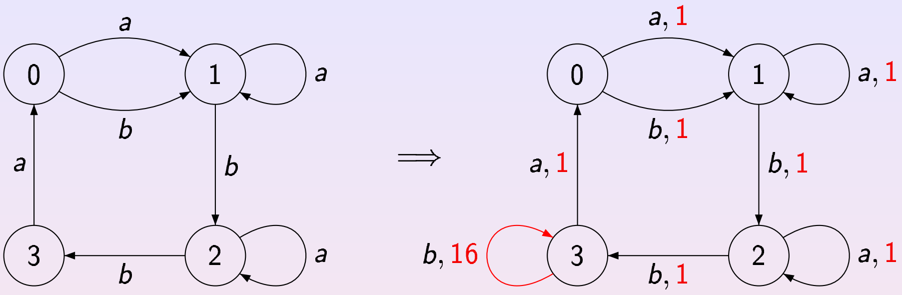
It is easy to see that $\mathscr{A}$ is carefully synchronizing iff $\mathscr{A}'$ can be synchronized on budget B. Hence, by Martyugin's theorem SYNCHRONIZING ON BUDGET is PSPACE-complete.

An Example



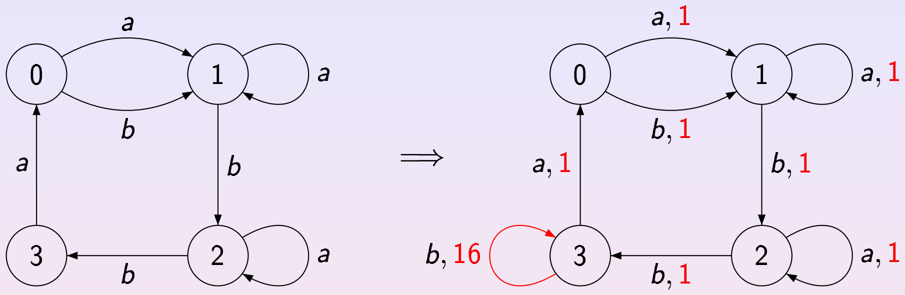
It is easy to see that $\mathscr{A}$ is carefully synchronizing iff $\mathscr{A}'$ can be synchronized on budget B. Hence, by Martyugin's theorem Synchronizing on Budget is PSPACE-complete.

An Example



It is easy to see that $\mathscr{A}$ is carefully synchronizing iff $\mathscr{A}'$ can be synchronized on budget B. Hence, by Martyugin's theorem SYNCHRONIZING ON BUDGET is PSPACE-complete.

An Example



It is easy to see that $\mathscr{A}$ is carefully synchronizing iff $\mathscr{A}'$ can be synchronized on budget B. Hence, by Martyugin's theorem SYNCHRONIZING ON BUDGET is PSPACE-complete.

Conclusion and Future Work

- We have answered the most immediate questions concerning synchronization games and synchronization costs.
- We have demonstrated an interesting application of careful synchronization.

Many natural open questions remain, including a synthesis of synchronization games and synchronization costs. We mean a game of two players on a synchronizing automaton equipped with a cost function where the aim of Alice is to minimize synchronization costs while Bob aims to prevent synchronization or at least to maximize synchronization costs.

Conclusion and Future Work

- We have answered the most immediate questions concerning synchronization games and synchronization costs.
- We have demonstrated an interesting application of careful synchronization.

Many natural open questions remain, including a synthesis of synchronization games and synchronization costs. We mean a game of two players on a synchronizing automaton equipped with a cost function where the aim of Alice is to minimize synchronization costs while Bob aims to prevent synchronization or at least to maximize synchronization costs.

Conclusion and Future Work

- We have answered the most immediate questions concerning synchronization games and synchronization costs.
- We have demonstrated an interesting application of careful synchronization.

Many natural open questions remain, including a synthesis of synchronization games and synchronization costs. We mean a game of two players on a synchronizing automaton equipped with a cost function where the aim of Alice is to minimize synchronization costs while Bob aims to prevent synchronization or at least to maximize synchronization costs.

LMRC, February 2nd, 2012