Synchronizing Finite Automata

Lecture III: Complexity Issues

Mikhail Volkov

Ural Federal University / Hunter College

1. Recap

Deterministic finite automata (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 $\mathscr{A}$.

1. Recap

Deterministic finite automata (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 $\mathscr{A}$.

1. Recap

Deterministic finite automata (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 $\mathscr{A}$.

1. Recap

Deterministic finite automata (DFA): $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

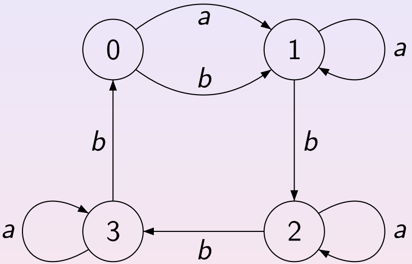
- Q the state set
- Σ the input alphabet
- $\delta : Q \times \Sigma \to Q$ the transition function

$\mathscr{A}$ 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 $Q\,.\,v = \{\delta(q, v) \mid q \in Q\}$.

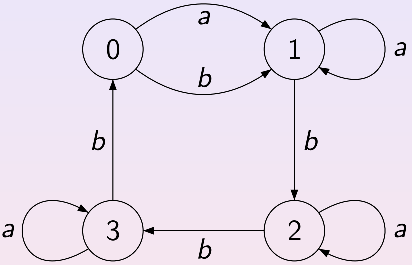
Any w with this property is a reset word for $\mathscr{A}$.

2. Example



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

2. Example



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

3. Short Reset Words are Hard to Find

There is a algorithm that uses a natural greedy strategy and, when given a synchronizing automaton $\mathscr{A}$ with n states, finds a reset word of length at most $\frac{n^3-n}{6}$ for $\mathscr{A}$ spending polynomial time as a function of n. (In fact, time is $O(n^3)$).

However, it is known that the gap between the size of the solution found by this greedy algorithm question or any of its versions and the size of the optimal solution (i.e., a reset word of minimum length) can be arbitrarily large.

Now we aim to prove that under standard assumptions (like $\mathrm{NP} \neq \mathrm{coNP}$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

3. Short Reset Words are Hard to Find

There is a algorithm that uses a natural greedy strategy and, when given a synchronizing automaton $\mathscr{A}$ with n states, finds a reset word of length at most $\frac{n^3-n}{6}$ for $\mathscr{A}$ spending polynomial time as a function of n. (In fact, time is $O(n^3)$).

However, it is known that the gap between the size of the solution found by this greedy algorithm question or any of its versions and the size of the optimal solution (i.e., a reset word of minimum length) can be arbitrarily large.

Now we aim to prove that under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

3. Short Reset Words are Hard to Find

There is a algorithm that uses a natural greedy strategy and, when given a synchronizing automaton $\mathscr{A}$ with n states, finds a reset word of length at most $\frac{n^3-n}{6}$ for $\mathscr{A}$ spending polynomial time as a function of n. (In fact, time is $O(n^3)$).

However, it is known that the gap between the size of the solution found by this greedy algorithm question or any of its versions and the size of the optimal solution (i.e., a reset word of minimum length) can be arbitrarily large.

Now we aim to prove that under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

Recall what are P, NP, coNP, etc.

These are classes of combinatorial decision problems, i.e., problems whose input is a finite object (graph, formula, automaton, ...). and whose question is whether or not a given object possesses a certain property (which usually gives the name to the problem). The answer to each concrete instance of such a problem is either YES or NO.

Recall what are P, NP, coNP, etc.

These are classes of combinatorial decision problems, i.e., problems whose input is a finite object (graph, formula, automaton, . . .).
and whose question is whether or not a given object possesses a certain property (which usually gives the name to the problem).
The answer to each concrete instance of such a problem is either YES or NO.

Recall what are P, NP, coNP, etc.

These are classes of combinatorial decision problems, i.e., problems whose input is a finite object (graph, formula, automaton, ...).

and whose question is whether or not a given object possesses a certain property (which usually gives the name to the problem).

The answer to each concrete instance of such a problem is either YES or NO.

Recall what are P, NP, coNP, etc.

These are classes of combinatorial decision problems, i.e., problems whose input is a finite object (graph, formula, automaton, ...). and whose question is whether or not a given object possesses a certain property (which usually gives the name to the problem).

The answer to each concrete instance of such a problem is either YES or NO.

Recall what are P, NP, coNP, etc.

These are classes of combinatorial decision problems, i.e., problems whose input is a finite object (graph, formula, automaton, ...). and whose question is whether or not a given object possesses a certain property (which usually gives the name to the problem). The answer to each concrete instance of such a problem is either YES or NO.

Recall what are P, NP, coNP, etc.

These are classes of combinatorial decision problems, i.e., problems whose input is a finite object (graph, formula, automaton, . . .). and whose question is whether or not a given object possesses a certain property (which usually gives the name to the problem). The answer to each concrete instance of such a problem is either YES or NO.

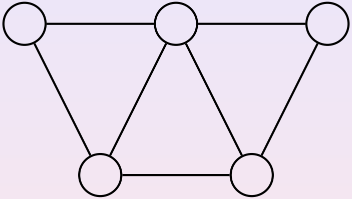
5. Example: Graph Coloring

The input of k-COLOR is a graph G.

The question is whether the vertices of G can be labeled with k colors so that adjacent vertices are assigned different colors.
For the above graph, the answer to 3-COLOR is YES
while the answer to 2-COLOR is NO.

5. Example: Graph Coloring

The input of k-COLOR is a graph G.

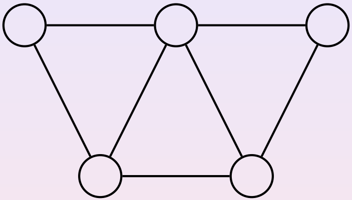


The question is whether the vertices of G can be labeled with k colors so that adjacent vertices are assigned different colors.

For the above graph, the answer to 3-COLOR is YES while the answer to 2-COLOR is NO.

5. Example: Graph Coloring

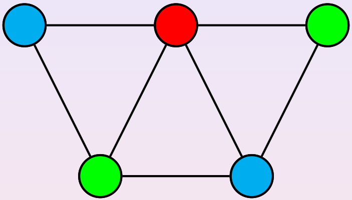
The input of k-COLOR is a graph G.



The question is whether the vertices of G can be labeled with k colors so that adjacent vertices are assigned different colors.
For the above graph, the answer to 3-COLOR is YES
while the answer to 2-COLOR is NO.

5. Example: Graph Coloring

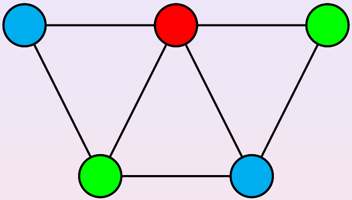
The input of k-COLOR is a graph G.



The question is whether the vertices of G can be labeled with k colors so that adjacent vertices are assigned different colors. For the above graph, the answer to 3-COLOR is YES while the answer to 2-COLOR is NO.

5. Example: Graph Coloring

The input of k-COLOR is a graph G.



The question is whether the vertices of G can be labeled with k colors so that adjacent vertices are assigned different colors.
For the above graph, the answer to 3-COLOR is YES
while the answer to 2-COLOR is NO.

6. Classes P, NP, and coNP

Arthur, an ordinary man

Arthur, an ordinary man

Merlin, a wizard

A problem is in P if Arthur can solve it in polynomial time (of the size of its input).

Example: 2-COLOR is in P since Arthur can check in polynomial time whether or not all simple cycles of a given graph are of even length.

A problem is in NP if, whenever the answer to its instance is YES, Merlin can convince Arthur that the answer is YES in polynomial time (of the size of the input).

Example: 3-COLOR is in NP since, given a 3-colorable graph, Merlin can exhibit its 3-coloring, and Arthur can check in polynomial time that this coloring is correct.

A problem is in coNP if, whenever the answer to its instance is NO, Merlin can convince Arthur that the answer is NO in polynomial time (of the size of the input).

7. Classes P, NP, and coNP

A problem is in P if Arthur can solve it in polynomial time (of the size of its input).
Example: 2-COLOR is in P since Arthur can check in polynomial time whether or not all simple cycles of a given graph are of even length.

A problem is in NP if, whenever the answer to its instance is YES, Merlin can convince Arthur that the answer is YES in polynomial time (of the size of the input).
Example: 3-COLOR is in NP since, given a 3-colorable graph, Merlin can exhibit its 3-coloring, and Arthur can check in polynomial time that this coloring is correct.

A problem is in coNP if, whenever the answer to its instance is NO, Merlin can convince Arthur that the answer is NO in polynomial time (of the size of the input).

7. Classes P, NP, and coNP

A problem is in P if Arthur can solve it in polynomial time (of the size of its input).
Example: 2-COLOR is in P since Arthur can check in polynomial time whether or not all simple cycles of a given graph are of even length.

A problem is in NP if, whenever the answer to its instance is YES, Merlin can convince Arthur that the answer is YES in polynomial time (of the size of the input).
Example: 3-COLOR is in NP since, given a 3-colorable graph, Merlin can exhibit its 3-coloring, and Arthur can check in polynomial time that this coloring is correct.

A problem is in coNP if, whenever the answer to its instance is NO, Merlin can convince Arthur that the answer is NO in polynomial time (of the size of the input).

7. Classes P, NP, and coNP

A problem is in P if Arthur can solve it in polynomial time (of the size of its input).
Example: 2-COLOR is in P since Arthur can check in polynomial time whether or not all simple cycles of a given graph are of even length.

A problem is in NP if, whenever the answer to its instance is YES, Merlin can convince Arthur that the answer is YES in polynomial time (of the size of the input).
Example: 3-COLOR is in NP since, given a 3-colorable graph, Merlin can exhibit its 3-coloring, and Arthur can check in polynomial time that this coloring is correct.

A problem is in coNP if, whenever the answer to its instance is NO, Merlin can convince Arthur that the answer is NO in polynomial time (of the size of the input).

7. Classes P, NP, and coNP

A problem is in P if Arthur can solve it in polynomial time (of the size of its input).
Example: 2-COLOR is in P since Arthur can check in polynomial time whether or not all simple cycles of a given graph are of even length.

A problem is in NP if, whenever the answer to its instance is YES, Merlin can convince Arthur that the answer is YES in polynomial time (of the size of the input).
Example: 3-COLOR is in NP since, given a 3-colorable graph, Merlin can exhibit its 3-coloring, and Arthur can check in polynomial time that this coloring is correct.

A problem is in coNP if, whenever the answer to its instance is NO, Merlin can convince Arthur that the answer is NO in polynomial time (of the size of the input).

8. Classes P, NP, and coNP

Clearly $P \subseteq NP$ and $P \subseteq coNP$.

Is any of the inclusions strict? In other words, is it true that $P \neq NP$?

This is a VERY BIG PROBLEM

which is worth \$1000000 (before tax).

According to the present paradigm, we assume that $P \neq NP \neq coNP$.

An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.

An NP-complete problem is a problem in NP that at the same time is NP-hard.

Example: 3-COLOR is NP-complete (Leonid Levin, 1973).

How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

8. Classes P, NP, and coNP

Clearly $P \subseteq NP$ and $P \subseteq coNP$.
Is any of the inclusions strict? In other words, is it true that $P \neq NP$?

This is a VERY BIG PROBLEM
which is worth \$1000000 (before tax).

According to the present paradigm, we assume that $P \neq NP \neq coNP$.

An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.

An NP-complete problem is a problem in NP that at the same time is NP-hard.

Example: 3-COLOR is NP-complete (Leonid Levin, 1973).

How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

8. Classes P, NP, and coNP

Clearly $P \subseteq NP$ and $P \subseteq coNP$.

Is any of the inclusions strict? In other words, is it true that $P \neq NP$?

This is a VERY BIG PROBLEM

which is worth \$1000000 (before tax).

According to the present paradigm, we assume that $P \neq NP \neq coNP$.

An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.

An NP-complete problem is a problem in NP that at the same time is NP-hard.

Example: 3-COLOR is NP-complete (Leonid Levin, 1973).

How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

Clearly $P \subseteq NP$ and $P \subseteq coNP$.
Is any of the inclusions strict? In other words, is it true that $P \neq NP$?
This is a VERY BIG PROBLEM
which is worth \$1000000 (before tax).
According to the present paradigm, we assume that
$P \neq NP \neq coNP$.
An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.
An NP-complete problem is a problem in NP that at the same time is NP-hard.
Example: 3-COLOR is NP-complete (Leonid Levin, 1973).
How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

Clearly $\mathrm{P} \subseteq \mathrm{NP}$ and $\mathrm{P} \subseteq \mathrm{coNP}$.

Is any of the inclusions strict? In other words, is it true that $\mathrm{P} \neq \mathrm{NP}$?

This is a VERY BIG PROBLEM

which is worth \$1000000 (before tax).

According to the present paradigm, we assume that $\mathrm{P} \neq \mathrm{NP} \neq \mathrm{coNP}$.

An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.

An NP-complete problem is a problem in NP that at the same time is NP-hard.

Example: 3-COLOR is NP-complete (Leonid Levin, 1973).

How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

8. Classes P, NP, and coNP

Clearly $P \subseteq NP$ and $P \subseteq coNP$.
Is any of the inclusions strict? In other words, is it true that $P \neq NP$?
This is a VERY BIG PROBLEM
which is worth $1000000 (before tax).
According to the present paradigm, we assume that $P \neq NP \neq coNP$.
An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.
An NP-complete problem is a problem in NP that at the same time is NP-hard.
Example: 3-COLOR is NP-complete (Leonid Levin, 1973).
How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

Clearly $P \subseteq NP$ and $P \subseteq coNP$.
Is any of the inclusions strict? In other words, is it true that $P \neq NP$?
This is a VERY BIG PROBLEM
which is worth \$1000000 (before tax).
According to the present paradigm, we assume that $P \neq NP \neq coNP$.
An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.
An NP-complete problem is a problem in NP that at the same time is NP-hard.
Example: 3-COLOR is NP-complete (Leonid Levin, 1973).
How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

Clearly $P \subseteq NP$ and $P \subseteq coNP$.
Is any of the inclusions strict? In other words, is it true that $P \neq NP$?
This is a VERY BIG PROBLEM
which is worth \$1000000 (before tax).
According to the present paradigm, we assume that $P \neq NP \neq coNP$.
An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.
An NP-complete problem is a problem in NP that at the same time is NP-hard.
Example: 3-COLOR is NP-complete (Leonid Levin, 1973).
How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

Clearly $P \subseteq NP$ and $P \subseteq coNP$.

Is any of the inclusions strict? In other words, is it true that $P \neq NP$?

This is a VERY BIG PROBLEM which is worth \$1000000 (before tax).

According to the present paradigm, we assume that $P \neq NP \neq coNP$.

An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.

An NP-complete problem is a problem in NP that at the same time is NP-hard.

Example: 3-COLOR is NP-complete (Leonid Levin, 1973).

How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

8. Classes P, NP, and coNP

Clearly $P \subseteq NP$ and $P \subseteq coNP$.

Is any of the inclusions strict? In other words, is it true that $P \neq NP$?

This is a VERY BIG PROBLEM

which is worth $1000000 (before tax).

According to the present paradigm, we assume that $P \neq NP \neq coNP$.

An NP-hard problem is a problem to which any problem from NP can be reduced in polynomial time.

An NP-complete problem is a problem in NP that at the same time is NP-hard.

Example: 3-COLOR is NP-complete (Leonid Levin, 1973).

How can one prove that a problem is NP-hard? Via a polynomial reduction from some problem known to be NP-complete.

Consider the following decision problem:

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

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

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

Consider the following decision problem:

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

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

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

Consider the following decision problem:

$\textsc{Short-Reset-Word}$: 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, $\textsc{Short-Reset-Word}$ belongs to NP: Merlin can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then Arthur can check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that $\textsc{Short-Reset-Word}$ is NP-hard by a transparent reduction from SAT which is a classical NP-complete problem.

10. Reduction from SAT

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

The transitions are defined by:

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

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

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

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

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

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

10. Reduction from SAT

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

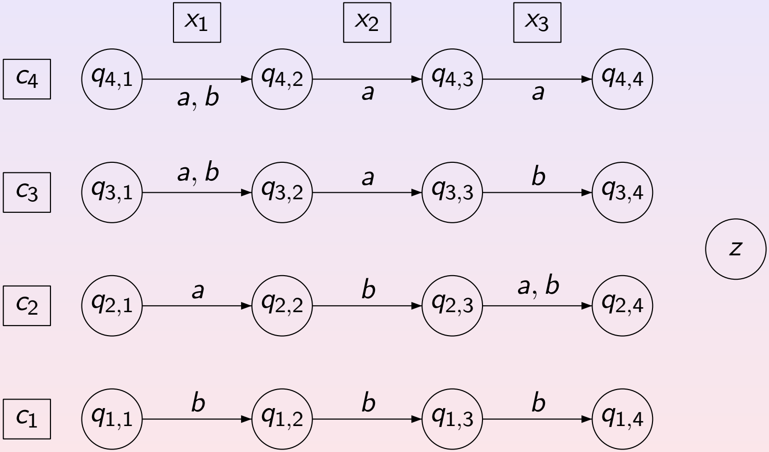
$$
\begin{aligned}
q_{i,j} \,.\, a &= \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} && \text{for } 1 \le i \le m,\ 1 \le j \le n; \\
q_{i,j} \,.\, b &= \begin{cases} z \text{ if } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases}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

11. Reduction from SAT

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

11. Reduction from SAT

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



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

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

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

In the above example the truth assignment $x_1 = x_2 = 0$, $x_3 = 1$ satisfies ψ and the word bba resets $\mathscr{A}(\psi)$.

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

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

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

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

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

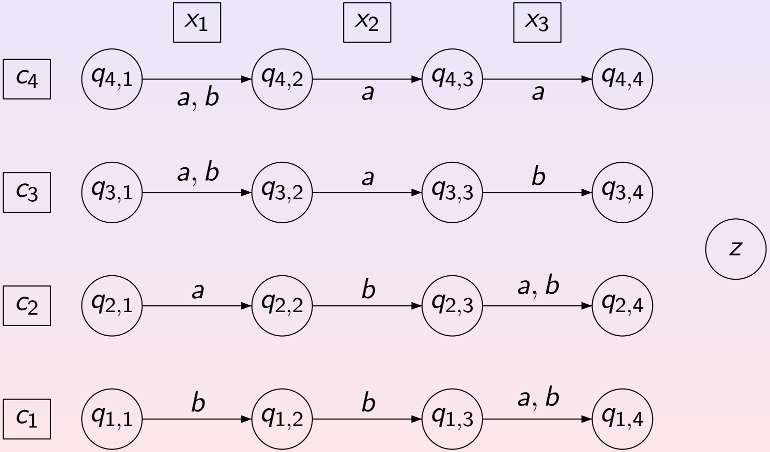
12. Reduction from SAT

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

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

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

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



14. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

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

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

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time) – Olschewski and Ummels, MFCS-2010.

Now consider the following decision problem:

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

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

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time) – Olschewski and Ummels, MFCS-2010.

Now consider the following decision problem:

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

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

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time) – Olschewski and Ummels, MFCS-2010.

Now consider the following decision problem:

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

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

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time) – Olschewski and Ummels, MFCS-2010.

14. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

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

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

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time) – Olschewski and Ummels, MFCS-2010.

15. Computing is Harder than Deciding

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

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

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ – Olschewski and Ummels, MFCS-2010.

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

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

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ – Olschewski and Ummels, MFCS-2010.

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

15. Computing is Harder than Deciding

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

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ – Olschewski and Ummels, MFCS-2010.

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

15. Computing is Harder than Deciding

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

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$ – Olschewski and Ummels, MFCS-2010.

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

16. Non-approximability: Constant Factor

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

Mikhail Berlinkov has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton with two input letters, produces a reset word whose length is less than $k \times$minimum possible length of a reset word.

The next question was: is approximating within a logarithmic factor possible?

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

Mikhail Berlinkov has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton with two input letters, produces a reset word whose length is less than $k \times$ minimum possible length of a reset word.

The next question was: is approximating within a logarithmic factor possible?

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

Mikhail Berlinkov has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton with two input letters, produces a reset word whose length is less than $k \times$minimum possible length of a reset word.

The next question was: is approximating within a logarithmic factor possible?

17. Non-approximability: Logarithmic Factor

Gerbush and Heeringa have observed that SET COVER admits a transparent reduction to the problem of finding a reset word of minimum length for a given synchronizing automaton. Recall that an instance of SET COVER consists of a set S, a family $\{C_i\}_{i\in I}$ of subsets of S and a positive integer N. The question is whether or not there exists a subset $J \subseteq I$ such that $|J| \leq N$ and $\bigcup_{j\in J} C_j = S$.

Using a difficult result on SET COVER by Alon, Moshkovitz and Safra, Gerbush and Heeringa have deduced that the minimum length of reset words for synchronizing automata with n states and unbounded alphabet cannot be approximated within the factor $c \log n$ for some constant $c > 0$ unless $\mathrm{P} = \mathrm{NP}$.

Berlinkov has obtained a similar result for synchronizing automata with only 2 input letters.

17. Non-approximability: Logarithmic Factor

Gerbush and Heeringa have observed that SET COVER admits a transparent reduction to the problem of finding a reset word of minimum length for a given synchronizing automaton. Recall that an instance of SET COVER consists of a set S, a family $\{C_i\}_{i\in I}$ of subsets of S and a positive integer N. The question is whether or not there exists a subset $J \subseteq I$ such that $|J| \leq N$ and $\bigcup_{j\in J} C_j = S$.

Using a difficult result on SET COVER by Alon, Moshkovitz and Safra, Gerbush and Heeringa have deduced that the minimum length of reset words for synchronizing automata with n states and unbounded alphabet cannot be approximated within the factor $c \log n$ for some constant $c > 0$ unless $\mathrm{P} = \mathrm{NP}$.

Berlinkov has obtained a similar result for synchronizing automata with only 2 input letters.

17. Non-approximability: Logarithmic Factor

Gerbush and Heeringa have observed that SET COVER admits a transparent reduction to the problem of finding a reset word of minimum length for a given synchronizing automaton. Recall that an instance of SET COVER consists of a set S, a family $\{C_i\}_{i\in I}$ of subsets of S and a positive integer N. The question is whether or not there exists a subset $J \subseteq I$ such that $|J| \leq N$ and $\bigcup_{j\in J} C_j = S$.

Using a difficult result on SET COVER by Alon, Moshkovitz and Safra, Gerbush and Heeringa have deduced that the minimum length of reset words for synchronizing automata with n states and unbounded alphabet cannot be approximated within the factor $c \log n$ for some constant $c > 0$ unless P = NP.

Berlinkov has obtained a similar result for synchronizing automata with only 2 input letters.

Gerbush and Heeringa have observed that SET COVER admits a transparent reduction to the problem of finding a reset word of minimum length for a given synchronizing automaton. Recall that an instance of SET COVER consists of a set S, a family $\{C_i\}_{i\in I}$ of subsets of S and a positive integer N. The question is whether or not there exists a subset $J \subseteq I$ such that $|J| \leq N$ and $\bigcup_{j\in J} C_j = S$.

Using a difficult result on SET COVER by Alon, Moshkovitz and Safra, Gerbush and Heeringa have deduced that the minimum length of reset words for synchronizing automata with n states and unbounded alphabet cannot be approximated within the factor $c \log n$ for some constant $c > 0$ unless $\mathrm{P} = \mathrm{NP}$.

Berlinkov has obtained a similar result for synchronizing automata with only 2 input letters.

17. Non-approximability: Logarithmic Factor

Gerbush and Heeringa have observed that SET COVER admits a transparent reduction to the problem of finding a reset word of minimum length for a given synchronizing automaton. Recall that an instance of SET COVER consists of a set S, a family $\{C_i\}_{i \in I}$ of subsets of S and a positive integer N. The question is whether or not there exists a subset $J \subseteq I$ such that $|J| \leq N$ and $\bigcup_{j \in J} C_j = S$.

Using a difficult result on SET COVER by Alon, Moshkovitz and Safra, Gerbush and Heeringa have deduced that the minimum length of reset words for synchronizing automata with n states and unbounded alphabet cannot be approximated within the factor $c \log n$ for some constant $c > 0$ unless $\mathrm{P} = \mathrm{NP}$.

Berlinkov has obtained a similar result for synchronizing automata with only 2 input letters.

18. Non-approximability: Sublinear Factor

Very recently, Gawrychowski and Straszak have shown that for every $\varepsilon > 0$ it is not possible to approximate the length of the shortest reset word for synchronizing automata with n states within a factor of $n^{1-\varepsilon}$ in polynomial time, unless $\mathrm{P} = \mathrm{NP}$.