Synchronizing Finite Automata

Lecture III: Complexity Issues

Mikhail Volkov

Ural Federal University

Spring of 2021

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

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 $q, q' \in Q$.

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

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

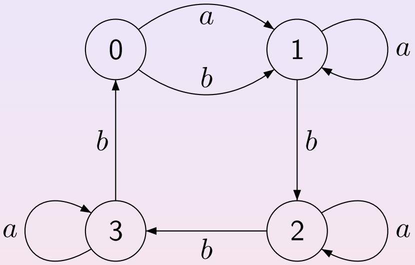
- Q the state set
- Σ the input alphabet
- $\delta : Q \times \Sigma \to Q$ the transition function

$\mathscr{A}$ 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 $q, q' \in Q$.

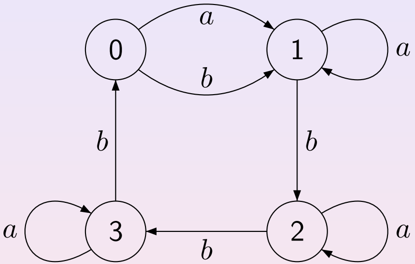
$|Q\,.\,w| = 1$. Here $Q\,.\,v = \{\delta(q, v) \mid q \in Q\}$.

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

2. Example



2. Example



A reset word is $abbbabbba$. In fact, this is the shortest reset word for this automaton.

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 $O(n^3)$).

3. Greedy Algorithm

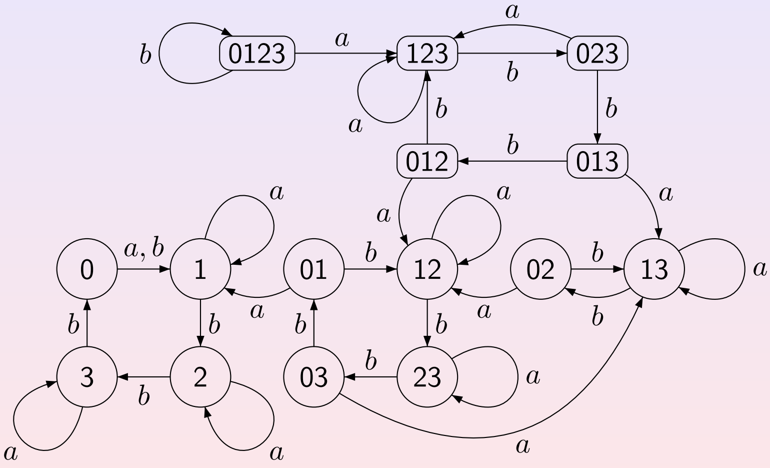
There is a algorithm that uses a natural greedy strategy and, when given a synchronizing automaton $\mathscr{A}$ with n states, finds a reset word of length at most $\frac{n^3-n}{6}$ for $\mathscr{A}$ spending polynomial time as a function of n. (In fact, time is $O(n^3)$).

GreedyCompression($\mathscr{A}$)

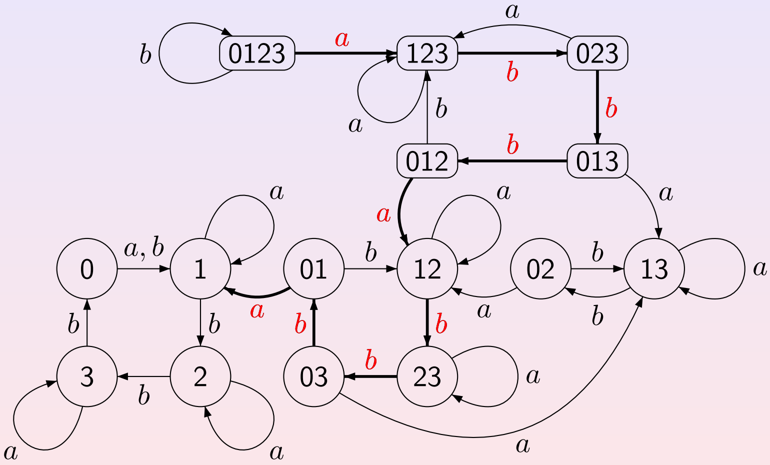
```
w ← ε                                          ▷ Initializing the current word
P ← Q                                          ▷ Initializing the current set
while |P| > 1 do
    if |P . u| = |P| for all u ∈ Σ* then
        return Failure
    else
        take a word v ∈ Σ* of minimum length with |P . v| < |P|
    w ← wv                                     ▷ Updating the current word
    P ← P . v                                  ▷ Updating the current set
return w
```

4. Example

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

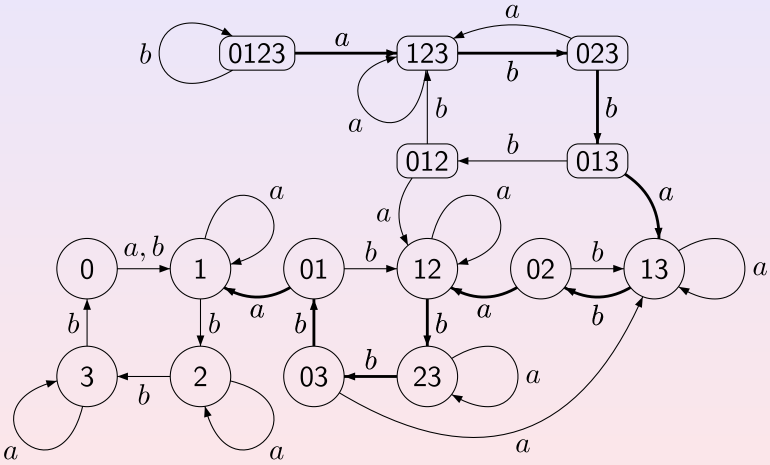


We have already seen that the greedy algorithm fails to find a reset word of minimum length.



4. Example

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



5. Short Reset Words are Hard to Find

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

5. Short Reset Words are Hard to Find

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

5. Short Reset Words are Hard to Find

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

Dmitry Ananichev and Vladimir Gusev (Approximation of reset thresholds with greedy algorithms, Fundam. Inform. 145:3, 221–227, 2016) have analysed the worst case behaviour of all natural variants of the greedy algorithm.

5. Short Reset Words are Hard to Find

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

Dmitry Ananichev and Vladimir Gusev (Approximation of reset thresholds with greedy algorithms, Fundam. Inform. 145:3, 221–227, 2016) have analysed the worst case behaviour of all natural variants of the greedy algorithm. They have shown that the gap between the sizes of the solution found by any of these variants and of the optimal solution can be arbitrarily large.

5. Short Reset Words are Hard to Find

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

Dmitry Ananichev and Vladimir Gusev (Approximation of reset thresholds with greedy algorithms, Fundam. Inform. 145:3, 221–227, 2016) have analysed the worst case behaviour of all natural variants of the greedy algorithm. They have shown that the gap between the sizes of the solution found by any of these variants and of the optimal solution 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.

6. 5-minute Tour in Complexity Theory

Recall what are P, NP, coNP, etc.

6. 5-minute Tour in Complexity Theory

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, . . .).

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, ...).

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).

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).

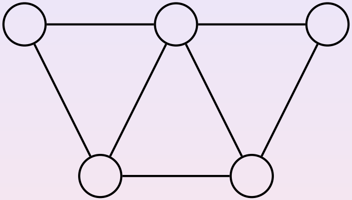
The answer to each concrete instance of such a problem is either YES or NO.

7. Example: Graph Coloring

The input of k-COLOR is a graph G.

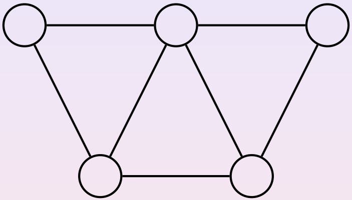
7. Example: Graph Coloring

The input of k-COLOR is a graph G.



7. Example: Graph Coloring

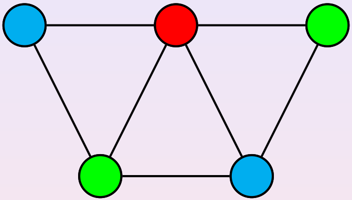
The input of k-COLOR is a graph G.



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

7. Example: Graph Coloring

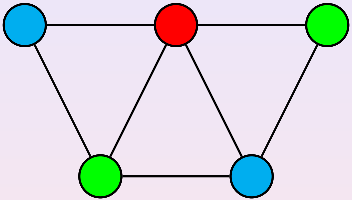
The input of k-COLOR is a graph G.



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

7. 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.

Arthur, an ordinary man

Arthur, an ordinary man

Merlin, a wizard

9. Classes P, NP, and coNP

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

9. 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.

9. 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).

9. 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 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).

10. Classes P, NP, and coNP

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

10. 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 $P \neq NP$?

10. Classes P, NP, and coNP

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

10. Classes P, NP, and coNP

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).

10. 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.

10. 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.

10. 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.

10. 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).

10. 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?

10. 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?

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 $\ell|Q|$.

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.

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables.

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables. A clause over V is a disjunction of literals and a literal is either a variable in V or the negation of a variable.

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables. A clause over V is a disjunction of literals and a literal is either a variable in V or the negation of a variable.
Example: $C = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables. A clause over V is a disjunction of literals and a literal is either a variable in V or the negation of a variable.
Example: $C = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$

A truth assignment on V is any map $\varphi\colon V \to \{0, 1\}$.

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables. A clause over V is a disjunction of literals and a literal is either a variable in V or the negation of a variable. Example: $C = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$

A truth assignment on V is any map $\varphi\colon V \to \{0,1\}$. It extends to a map $C \to \{0,1\}$ (still denoted by φ) via the usual rules:

$$\varphi(\neg x) = 1 - \varphi(x), \quad \varphi(x \vee y) = \max\{\varphi(x), \varphi(y)\}.$$

12. SAT

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables. A clause over V is a disjunction of literals and a literal is either a variable in V or the negation of a variable. Example: $C = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$

A truth assignment on V is any map $\varphi\colon V \to \{0,1\}$. It extends to a map $C \to \{0,1\}$ (still denoted by φ) via the usual rules:

$$\varphi(\neg x) = 1 - \varphi(x), \quad \varphi(x \vee y) = \max\{\varphi(x), \varphi(y)\}.$$

A truth assignment φ satisfies C if $\varphi(c) = 1$ for all $c \in C$.

Recall what the Boolean satisfiability problem (SAT) is.

An instance C of SAT is a collection of clauses over a set V of Boolean variables. A clause over V is a disjunction of literals and a literal is either a variable in V or the negation of a variable. Example: $C = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$

A truth assignment on V is any map $\varphi\colon V \to \{0,1\}$. It extends to a map $C \to \{0,1\}$ (still denoted by φ) via the usual rules:

$$\varphi(\neg x) = 1 - \varphi(x), \quad \varphi(x \vee y) = \max\{\varphi(x), \varphi(y)\}.$$

A truth assignment φ satisfies C if $\varphi(c) = 1$ for all $c \in C$.

The answer to an instance C is YES if C has a satisfying assignment (i.e., a truth assignment on V that satisfies C) and NO otherwise.

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

Given an instance C of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(C)$ 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 C of SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, one constructs $\mathscr{A}(C)$ 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:

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

13. Reduction from SAT

Given an instance C of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(C)$ 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:

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

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

13. Reduction from SAT

Given an instance C of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(C)$ 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:

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

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

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

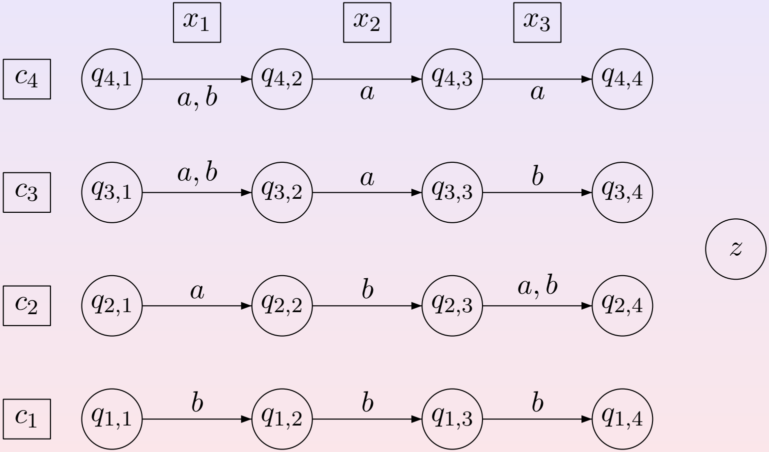
$$z \,.\, a = z \,.\, b = z.$$

14. Reduction from SAT

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

14. Reduction from SAT

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



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

15. Reduction from SAT

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

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

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

If we change C to $C' = \{x_1 \vee x_2, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_2 \vee \neg x_3\}$, it becomes unsatisfiable and $\mathscr{A}(C')$ is reset by no word of length 3.

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

Thus, assigning the instance $(\mathscr{A}(C), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance C of SAT, one gets a polynomial reduction

15. Reduction from SAT

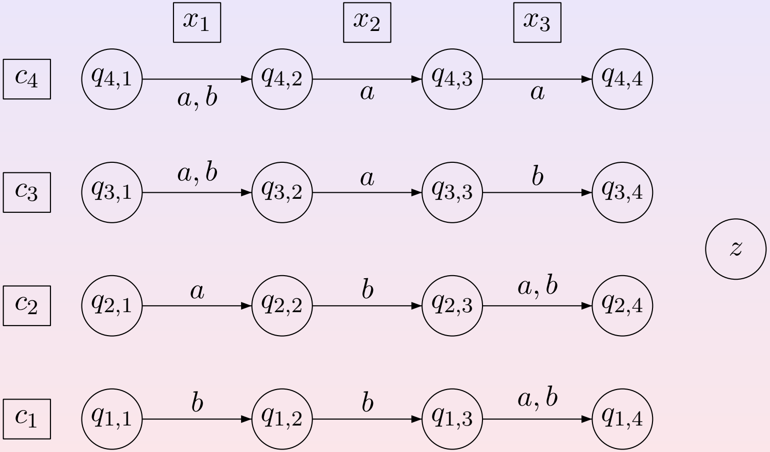
It is easy to see that $\mathscr{A}(C)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if C is satisfiable.

Thus, assigning the instance $(\mathscr{A}(C), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance C of SAT, one gets a polynomial reduction which is in fact parsimonious, i.e., there is a 1-1 correspondence between the satisfying assignments for C and reset words of length n for $\mathscr{A}(C)$.

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

16. Reduction from SAT

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



17. 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?

17. 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 $(\mathcal{A}(C), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system C of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if C is not satisfiable.

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

17. 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 $(\mathcal{A}(C), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system C of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if C 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.

17. 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 $(\mathcal{A}(C), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system C of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if C 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.

SHORTEST-RESET-WORD has been shown to be complete for DP (Difference Polynomial-Time) by Jörg Olschewski and Michael Ummels, The complexity of finding reset words in finite automata, MFCS 2010, LNCS 6281: 568–579, 2010.

18. 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.

18. 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.

$P^{NP[\log]}$ 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, loc. cit.

$P^{NP[\log]}$ 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, loc. cit.

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

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

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 (Approximating the minimum length of synchronizing words is hard, Theory Comput. Syst. 54:2, 211–223, 2014).

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 (Approximating the minimum length of synchronizing words is hard, Theory Comput. Syst. 54:2, 211–223, 2014).

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

20. Non-approximability: Logarithmic Factor

Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) 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.

Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) 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.

20. Non-approximability: Logarithmic Factor

Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) 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 $\bigcup_{j \in J} C_j = S$.

20. Non-approximability: Logarithmic Factor

Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) 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.

20. Non-approximability: Logarithmic Factor

Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) 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 (On two algorithmic problems about synchronizing automata, DLT 2014, LNCS 8633: 61–67, 2014).

21. Non-approximability: Sublinear Factor

Finally, Pawel Gawrychowski and Damian 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 P = NP (Strong inapproximability of the shortest reset word, MFCS 2015 Part 1, LNCS 9234: 243–255, 2015).

21. Non-approximability: Sublinear Factor

Finally, Pawel Gawrychowski and Damian 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 $\mathsf{P} = \mathsf{NP}$ (Strong inapproximability of the shortest reset word, MFCS 2015 Part 1, LNCS 9234: 243–255, 2015).

This is an ultimate result in a sense because polynomial algorithms that approximate the length of the shortest reset word within a linear factor are known.

21. Non-approximability: Sublinear Factor

Finally, Pawel Gawrychowski and Damian 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 P = NP (Strong inapproximability of the shortest reset word, MFCS 2015 Part 1, LNCS 9234: 243–255, 2015).

This is an ultimate result in a sense because polynomial algorithms that approximate the length of the shortest reset word within a linear factor are known.
For instance, Gerbush and Heeringa (loc. cit.) constructed an algorithm that, given a synchronizing automaton with n states and m input letters, finds its reset word with length $\leq \lceil \frac{n-1}{k-1} \rceil \ell$, where ℓ is the length of its shortest reset word, in time $O(kmn^k + \frac{n^4}{k})$.

Finally, Pawel Gawrychowski and Damian 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 P = NP (Strong inapproximability of the shortest reset word, MFCS 2015 Part 1, LNCS 9234: 243–255, 2015).

This is an ultimate result in a sense because polynomial algorithms that approximate the length of the shortest reset word within a linear factor are known.

For instance, Gerbush and Heeringa (loc. cit.) constructed an algorithm that, given a synchronizing automaton with n states and m input letters, finds its reset word with length $\leq \lceil\frac{n-1}{k-1}\rceil\ell$, where ℓ is the length of its shortest reset word, in time $O(kmn^k + \frac{n^4}{k})$. Here $k > 1$ is a lookahead parameter.

22. Constrained Synchronization

So far we allow reset words to be arbitrary words over the input language of the corresponding DFA.

22. Constrained Synchronization

So far we allow reset words to be arbitrary words over the input language of the corresponding DFA. In reality, however, available commands might be subject to certain restrictions.

So far we allow reset words to be arbitrary words over the input language of the corresponding DFA. In reality, however, available commands might be subject to certain restrictions. For instance, it is quite natural to assume that a reset word should always start and end with a specific command that first switches the automaton to a 'directive' mode and then returns the automaton to its usual mode: compare with $ tag in TEX.

22. Constrained Synchronization

So far we allow reset words to be arbitrary words over the input language of the corresponding DFA. In reality, however, available commands might be subject to certain restrictions. For instance, it is quite natural to assume that a reset word should always start and end with a specific command that first switches the automaton to a 'directive' mode and then returns the automaton to its usual mode: compare with $ tag in TEX.

In symbols, given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, we fix a letter $a \in \Sigma$ and seek a reset word from the (regular) language $a(\Sigma \setminus \{a\})^* a$.

22. Constrained Synchronization

So far we allow reset words to be arbitrary words over the input language of the corresponding DFA. In reality, however, available commands might be subject to certain restrictions. For instance, it is quite natural to assume that a reset word should always start and end with a specific command that first switches the automaton to a 'directive' mode and then returns the automaton to its usual mode: compare with $ tag in TEX.

In symbols, given a DFA $\mathscr{A} = \langle Q, \Sigma \rangle$, we fix a letter $a \in \Sigma$ and seek a reset word from the (regular) language $a(\Sigma \setminus \{a\})^* a$. Does $\mathscr{A}$ admit synchronization under such a constraint?

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

Theorem (Henning Fernau, Vladimir Gusev, Stefan Hoffmann, Markus Holzer, Mikhail Volkov, and Petra Wolf, MFCS 2019)

The problem of deciding whether a given $\mathscr{A} = \langle Q, \Sigma \rangle$ has a reset word from the language $a(\Sigma \setminus \{a\})^*a$, for a fixed letter $a \in \Sigma$, is PSPACE-complete if $|\Sigma| \geq 3$ and NP-complete if $|\Sigma| = 2$.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

Theorem (Henning Fernau, Vladimir Gusev, Stefan Hoffmann, Markus Holzer, Mikhail Volkov, and Petra Wolf, MFCS 2019)

The problem of deciding whether a given $\mathscr{A} = \langle Q, \Sigma \rangle$ has a reset word from the language $a(\Sigma \setminus \{a\})^*a$, for a fixed letter $a \in \Sigma$, is PSPACE-complete if $|\Sigma| \geq 3$ and NP-complete if $|\Sigma| = 2$.

To prove hardness we reduce from DFA-Intersection Nonemptiness.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

Theorem (Henning Fernau, Vladimir Gusev, Stefan Hoffmann, Markus Holzer, Mikhail Volkov, and Petra Wolf, MFCS 2019)

The problem of deciding whether a given $\mathscr{A} = \langle Q, \Sigma \rangle$ has a reset word from the language $a(\Sigma \setminus \{a\})^*a$, for a fixed letter $a \in \Sigma$, is PSPACE-complete if $|\Sigma| \geq 3$ and NP-complete if $|\Sigma| = 2$.

To prove hardness we reduce from DFA-Intersection Nonemptiness. Membership in PSPACE is easy due to Savitch's Theorem.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

Theorem (Henning Fernau, Vladimir Gusev, Stefan Hoffmann, Markus Holzer, Mikhail Volkov, and Petra Wolf, MFCS 2019)

The problem of deciding whether a given $\mathscr{A} = \langle Q, \Sigma \rangle$ has a reset word from the language $a(\Sigma \setminus \{a\})^*a$, for a fixed letter $a \in \Sigma$, is PSPACE-complete if $|\Sigma| \geq 3$ and NP-complete if $|\Sigma| = 2$.

To prove hardness we reduce from DFA-Intersection Nonemptiness.
Membership in PSPACE is easy due to Savitch's Theorem.
A relatively nontrivial part is membership in NP for $|\Sigma| = 2$.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^*a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

Theorem (Henning Fernau, Vladimir Gusev, Stefan Hoffmann, Markus Holzer, Mikhail Volkov, and Petra Wolf, MFCS 2019)

The problem of deciding whether a given $\mathscr{A} = \langle Q, \Sigma \rangle$ has a reset word from the language $a(\Sigma \setminus \{a\})^*a$, for a fixed letter $a \in \Sigma$, is PSPACE-complete if $|\Sigma| \geq 3$ and NP-complete if $|\Sigma| = 2$.

To prove hardness we reduce from DFA-Intersection Nonemptiness.
Membership in PSPACE is easy due to Savitch's Theorem.
A relatively nontrivial part is membership in NP for $|\Sigma| = 2$.
There are DFAs $\mathscr{A} = \langle Q, \{a, b\} \rangle$ for which the least N such that ab^Na is a reset word for $\mathscr{A}$ is exponentially big with respect to $|Q|$.

23. Constrained Synchronization: An Example

The constraint language $a(\Sigma \setminus \{a\})^* a$ is very simple and looks innocent. Therefore the following result was somewhat surprising.

Theorem (Henning Fernau, Vladimir Gusev, Stefan Hoffmann, Markus Holzer, Mikhail Volkov, and Petra Wolf, MFCS 2019)

The problem of deciding whether a given $\mathscr{A} = \langle Q, \Sigma \rangle$ has a reset word from the language $a(\Sigma \setminus \{a\})^* a$, for a fixed letter $a \in \Sigma$, is PSPACE-complete if $|\Sigma| \geq 3$ and NP-complete if $|\Sigma| = 2$.

To prove hardness we reduce from DFA-Intersection Nonemptiness. Membership in PSPACE is easy due to Savitch's Theorem. A relatively nontrivial part is membership in NP for $|\Sigma| = 2$. There are DFAs $\mathscr{A} = \langle Q, \{a, b\} \rangle$ for which the least N such that $ab^N a$ is a reset word for $\mathscr{A}$ is exponentially big with respect to $|Q|$. One can guess N in binary, but a direct verification that $ab^N a$ resets $\mathscr{A}$ may be unfeasible.

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language.

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L.

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L. It is easy to show that this problem is in PSPACE for each regular L.

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L. It is easy to show that this problem is in PSPACE for each regular L.

The ultimate goal is to classify regular languages L according to the computational complexity of L-Synchronization.

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L. It is easy to show that this problem is in PSPACE for each regular L.

The ultimate goal is to classify regular languages L according to the computational complexity of L-Synchronization.

For which complexity classes $\mathbf{C}$ can L-Synchronization be $\mathbf{C}$-complete?

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L. It is easy to show that this problem is in PSPACE for each regular L.

The ultimate goal is to classify regular languages L according to the computational complexity of L-Synchronization.

For which complexity classes $\mathbf{C}$ can L-Synchronization be $\mathbf{C}$-complete?

Recall that $L = ab^*a$ provides NP-completeness while for $L = a(b+c)^*a$ we get PSPACE-completeness.

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L. It is easy to show that this problem is in PSPACE for each regular L.

The ultimate goal is to classify regular languages L according to the computational complexity of L-Synchronization.

For which complexity classes $\mathbf{C}$ can L-Synchronization be $\mathbf{C}$-complete?

Recall that $L = ab^*a$ provides NP-completeness while for $L = a(b+c)^*a$ we get PSPACE-completeness. Of course, L-Synchronization can be in P.

24. Regular Constraints

Let $L \subseteq \Sigma^*$ be a regular language. The L-Synchronization problem asks whether a given DFA whose input alphabet is Σ admits a reset word in L. It is easy to show that this problem is in PSPACE for each regular L.

The ultimate goal is to classify regular languages L according to the computational complexity of L-Synchronization.

For which complexity classes $\mathbf{C}$ can L-Synchronization be $\mathbf{C}$-complete?

Recall that $L = ab^*a$ provides NP-completeness while for $L = a(b+c)^*a$ we get PSPACE-completeness. Of course, L-Synchronization can be in P.

So far we have not found representatives for any other complexity class so that one can state a trichotomy conjecture (but I do not believe in it).