1. Recap

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

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

$\mathcal{A}$ is called synchronizing if there exists a word $w \in \Sigma^*$ whose action resets $\mathcal{A}$, 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 $q, q' \in Q$.

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

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

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

- \( Q \) the state set
- \( \Sigma \) the input alphabet
- \( \delta : Q \times \Sigma \rightarrow Q \) the transition function

\( \mathcal{A} \) is called synchronizing if there exists a word \( w \in \Sigma^* \) whose action resets \( \mathcal{A} \), 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 \( q, q' \in Q \).

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

Any \( w \) with this property is a reset word for \( \mathcal{A} \).
1. Recap

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

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

$\mathcal{A}$ is called **synchronizing** if there exists a word $w \in \Sigma^*$ whose action resets $\mathcal{A}$, 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 $q, q' \in Q$.

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

Any $w$ with this property is a **reset word** for $\mathcal{A}$. 
Deterministic finite automata (DFA): \( \mathcal{A} = \langle Q, \Sigma, \delta \rangle \).

- \( Q \) the state set
- \( \Sigma \) the input alphabet
- \( \delta : Q \times \Sigma \to Q \) the transition function

\( \mathcal{A} \) is called **synchronizing** if there exists a word \( w \in \Sigma^* \) whose action resets \( \mathcal{A} \), 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 \( q, q' \in Q \).

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

Any \( w \) with this property is a **reset word** for \( \mathcal{A} \).
A reset word is $abbbabbbba$. In fact, this is the shortest reset word for this automaton.
A reset word is \( a b b b a b b b b a \). In fact, this is the shortest reset word for this automaton.
3. Greedy Algorithm

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

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

**GreedyCompression**($A$)

1: $w \leftarrow \varepsilon$\hspace{1cm}▷ Initializing the current word
2: $P \leftarrow Q$\hspace{1cm}▷ Initializing the current set
3: while $|P| > 1$ do
4: \hspace{1cm}if $|P \cdot u| = |P|$ for all $u \in \Sigma^*$ then
5: \hspace{2cm}return Failure
6: \hspace{1cm}else
7: \hspace{2cm}take a word $v \in \Sigma^*$ of minimum length with $|P \cdot v| < |P|$
8: \hspace{2cm}$w \leftarrow wv$\hspace{1cm}▷ Updating the current word
9: \hspace{2cm}$P \leftarrow P \cdot v$\hspace{1cm}▷ Updating the current set
10: 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.
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 \( \text{NP} \neq \text{coNP} \)) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.
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 $\text{NP} \neq \text{coNP}$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.
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 $\text{NP} \neq \text{coNP}$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.
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 $\text{NP} \neq \text{coNP}$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.
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 \( \text{NP} \neq \text{coNP} \)) 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, \ldots). 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 \textit{combinatorial decision problems}, i.e., problems whose input is a \textit{finite object} (graph, formula, automaton, \ldots). and whose question is whether or not a given object possesses a certain \textit{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.
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.
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.
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.
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.
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.
8. Classes P, NP, and coNP

Arthur, an ordinary man
8. Classes P, NP, and coNP

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).
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).
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).
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).
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 $P \subseteq coNP$.

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.
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.
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.
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.
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 $P \subseteq \text{coNP}$.
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 $P \neq NP \neq \text{coNP}$.
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.
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.
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.
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 \( \mathcal{A} = \langle Q, \Sigma, \delta \rangle \) and a positive integer \( \ell \), is it true that \( \mathcal{A} \) has a reset word of length \( \ell \)?

Clearly, **Short-Reset-Word** belongs to NP: Merlin can non-deterministically guess a word \( w \in \Sigma^* \) of length \( \ell \) and then Arthur can check if \( w \) is a reset word for \( \mathcal{A} \) 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 \( \mathcal{A} = \langle Q, \Sigma, \delta \rangle \) and a positive integer \( \ell \), is it true that \( \mathcal{A} \) has a reset word of length \( \ell \)?

Clearly, **Short-Reset-Word** belongs to NP: Merlin can non-deterministically guess a word \( w \in \Sigma^* \) of length \( \ell \) and then Arthur can check if \( w \) is a reset word for \( \mathcal{A} \) 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 $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ and a positive integer $\ell$, is it true that $\mathcal{A}$ has a reset word of length $\ell$?

Clearly, **Short-Reset-Word** belongs to NP: Merlin can non-deterministically guess a word $w \in \Sigma^*$ of length $\ell$ and then Arthur can check if $w$ is a reset word for $\mathcal{A}$ 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.
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: $C = \{x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$

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

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

A truth assignment $\varphi$ 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.
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: $C = \{x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$

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

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

A truth assignment $\varphi$ 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.
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: $C = \{x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$

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

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

A truth assignment $\varphi$ 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.
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: $C = \{ x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3 \}$

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

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

A truth assignment $\varphi$ 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.
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: $C = \{ x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3 \}$

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

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

A truth assignment $\varphi$ 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.
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: $C = \{x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$

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

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

A truth assignment $\phi$ satisfies $C$ if $\phi(c) = 1$ 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.
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: $C = \{x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$

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

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

A truth assignment $\varphi$ 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.
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: $C = \{ x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3 \}$

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

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

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

$$q_{i,j} \cdot a = \begin{cases} z & \text{if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} & \text{otherwise} \end{cases} \quad \text{for } 1 \leq i \leq m, 1 \leq j \leq n;$$
Given an instance $C$ of SAT with $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $c_1, \ldots, c_m$, one constructs $A(C)$ 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:

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

$$q_{i,j} \cdot b = \begin{cases} z \text{ if } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \quad \text{for } 1 \leq i \leq m, \ 1 \leq j \leq n;$$
Given an instance $C$ of SAT with $n$ variables $x_1, \ldots, x_n$ and $m$ clauses $c_1, \ldots, c_m$, one constructs $A(C)$ 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:

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

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

$$q_{i,n+1} \cdot a = q_{i,n+1} \cdot b = z \quad \text{for } 1 \leq i \leq m;$$

$$z \cdot a = z \cdot b = z.$$
14. Reduction from SAT

For $C = \{ x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3 \}$:
14. Reduction from SAT

For $C = \{x_1 \lor x_2 \lor x_3, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$:
15. Reduction from SAT

It is easy to see that $\mathcal{A}(C)$ 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 $(\mathcal{A}(C), n)$ of \textsc{Short-Reset-Word} to an arbitrary $n$-variable instance $C$ of \textsc{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 $\mathcal{A}(C)$. 
It is easy to see that $A(C)$ 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 $A(C)$.

Thus, assigning the instance $(A(C), n)$ 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 $A(C)$.
It is easy to see that $A(C)$ 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 $A(C)$.

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

Thus, assigning the instance $(A(C), n)$ 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 $A(C)$. 
It is easy to see that $A(C)$ 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 $(A(C), n)$ of \textsc{Short-Reset-Word} to an arbitrary $n$-variable instance $C$ of \textsc{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 $A(C)$.
It is easy to see that $A(C)$ 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 $(A(C), n)$ 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 $A(C)$. 
For $C = \{x_1 \lor x_2, \neg x_1 \lor x_2, \neg x_2 \lor x_3, \neg x_2 \lor \neg x_3\}$:
16. Reduction from SAT

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

- $c_4$: $q_{4,1} \rightarrow q_{4,2} \rightarrow q_{4,3} \rightarrow q_{4,4}$
  - $x_1$ with transitions $a, b$
  - $x_2$ with transition $a$
  - $x_3$ with transition $a$

- $c_3$: $q_{3,1} \rightarrow q_{3,2} \rightarrow q_{3,3} \rightarrow q_{3,4}$
  - $a, b$ with transition $a$
  - $b$ with transition $b$

- $c_2$: $q_{2,1} \rightarrow q_{2,2} \rightarrow q_{2,3} \rightarrow q_{2,4}$
  - $a$ with transition $b$
  - $b$ with transition $a, b$

- $c_1$: $q_{1,1} \rightarrow q_{1,2} \rightarrow q_{1,3} \rightarrow q_{1,4}$
  - $b$ with transition $b$
  - $a, b$ with transition $a, b$

$z$
Now consider the following decision problem:

**Shortest-Reset-Word**: Given a synchronizing automaton $A$ and a positive integer $\ell$, is it true that the minimum length of a reset word for $A$ is equal to $\ell$?

Assigning the instance $(A(C), n + 1)$ 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.
Now consider the following decision problem:

**Shortest-Reset-Word**: *Given a synchronizing automaton $A$ and a positive integer $\ell$, is it true that the minimum length of a reset word for $A$ is equal to $\ell$?*

Assigning the instance $(A(C), n + 1)$ 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.
Now consider the following decision problem:

**Shortest-Reset-Word:** Given a synchronize automaton $A$ and a positive integer $\ell$, is it true that the minimum length of a reset word for $A$ is equal to $\ell$?

Assigning the instance $(A(C), n + 1)$ 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.
Now consider the following decision problem:

**Shortest-Reset-Word:** \textit{Given a synchronizing automaton }$A$\textit{ and a positive integer }$\ell$\textit{, is it true that the minimum length of a reset word for }$A$\textit{ is equal to }$\ell$\textit{?}

Assigning the instance $(A(C), n + 1)$ 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.
Now consider the following decision problem:

**Shortest-Reset-Word:** Given a synchronizing automaton $A$ and a positive integer $\ell$, is it true that the minimum length of a reset word for $A$ is equal to $\ell$?

Assigning the instance $(A(C), n + 1)$ 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 $= \text{coNP}$.

**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.
\[ P^{NP[\log]} \] 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.
$\text{P}^{\text{NP}[\log]}$ 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 $\text{P}^{\text{NP}[\log]}$ (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 $\text{FP}^{\text{NP}[\log]}$ of $\text{P}^{\text{NP}[\log]}$ — 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.
$P^{NP[\log]}$ 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.
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!

Mikhail Berlinkov has shown that under $\text{NP} \neq \text{P}$, 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 $k \times \text{minimum possible length of a reset word}$ (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?
However, all these results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Mikhail Berlinkov has shown that under $\text{NP} \neq \text{P}$, 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 $k \times \text{minimum possible length of a reset word}$ (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?
However, all these results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Mikhail Berlinkov has shown that under \( \text{NP} \neq \text{P} \), 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 \( k \times \text{minimum possible length of a reset word} \) (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?
Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) have observed that \textsc{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 \textsc{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 \textsc{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).
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).
Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) have observed that \textsc{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 \textsc{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 \textsc{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).
Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) have observed that \textsc{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 \textsc{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 \textsc{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 \textbf{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).
Michael Gerbush and Brent Heeringa (Approximating minimum reset sequence, CIAA 2010, LNCS 6482: 154–162, 2010) have observed that \textsc{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 \textsc{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 \textsc{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).
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).