

# The Complexity of Finding Reset Words in Finite Automata

Michael Ummels  
ummels@lsv.ens-cachan.fr

LSV, CNRS & ENS Cachan

Joint work with Jörg Olschewski

# Reset Words

**Setup:** Deterministic finite automata  $\mathcal{A} = (Q, \Sigma, \delta)$ .

## Definition

A word  $w$  is a *reset word* for  $\mathcal{A}$  if  $|\delta^*(Q, w)| = 1$ .

An automaton  $\mathcal{A}$  is *synchronizing* if it has a reset word.

**Note:** If  $w$  is a reset word, then so is  $xwy$  for all  $x, y \in \Sigma^*$ .

**Question:** How long can a shortest reset word be?

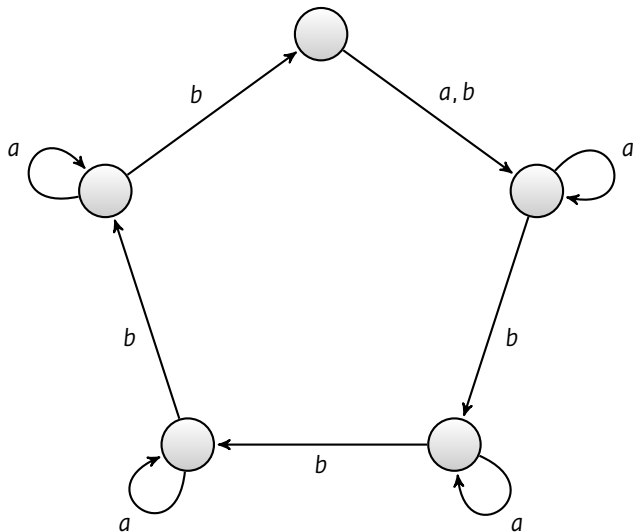
- ▶ Lower bound:  $(n - 1)^2$  (Černý)
- ▶ Upper bound:  $(n^3 - n)/6$  (Pin)

## Conjecture (Černý)

Every synchronizing automaton has a reset word of length  $(n - 1)^2$ .

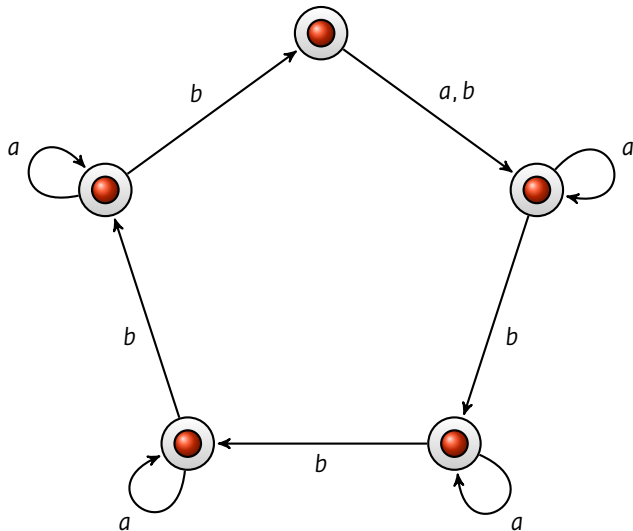
# Example

Černý's automaton  $\mathcal{C}_5$ :



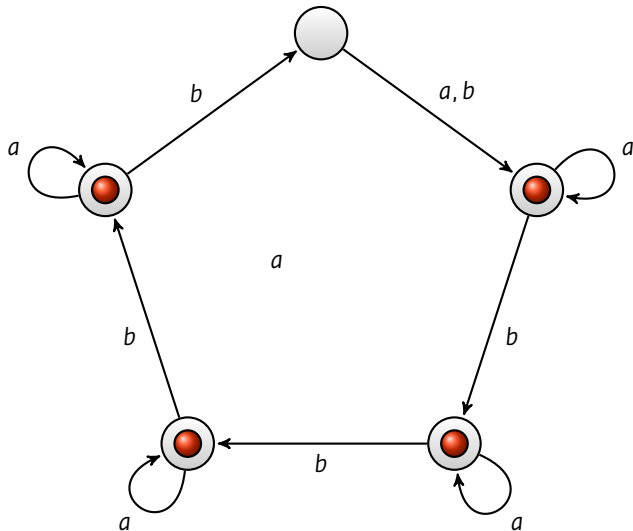
# Example

Černý's automaton  $\mathcal{C}_5$ :



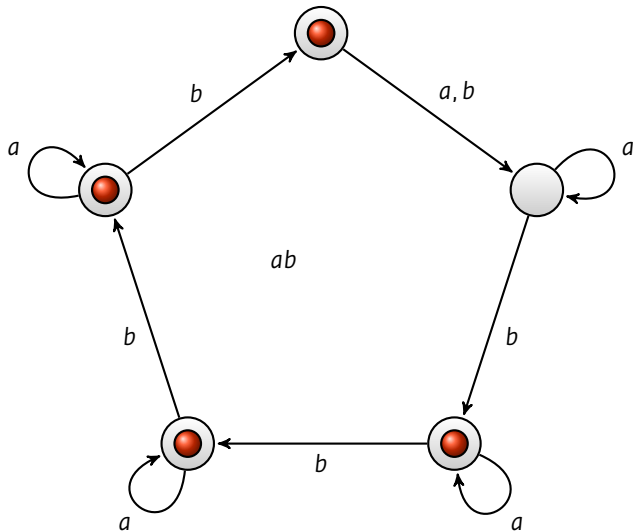
# Example

Černý's automaton  $\mathcal{C}_5$ :



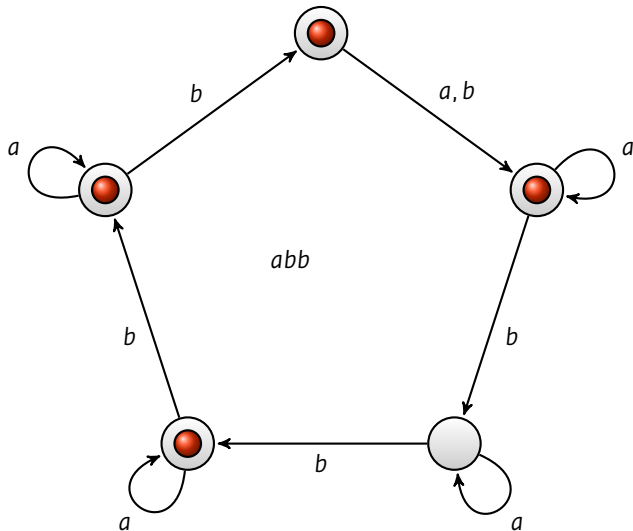
# Example

Černý's automaton  $\mathcal{C}_5$ :



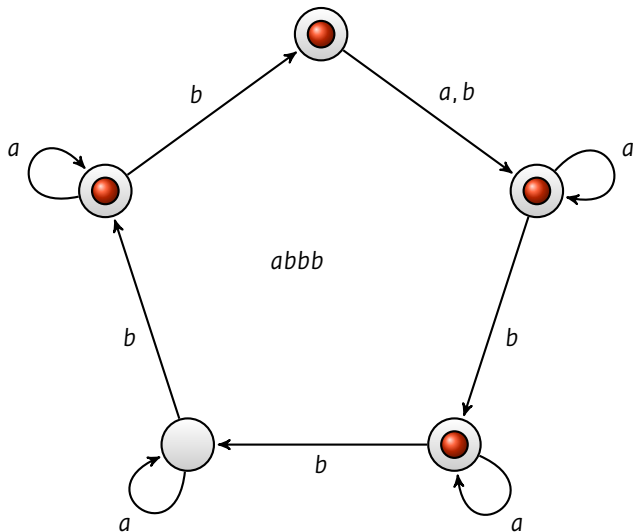
# Example

Černý's automaton  $\mathcal{C}_5$ :



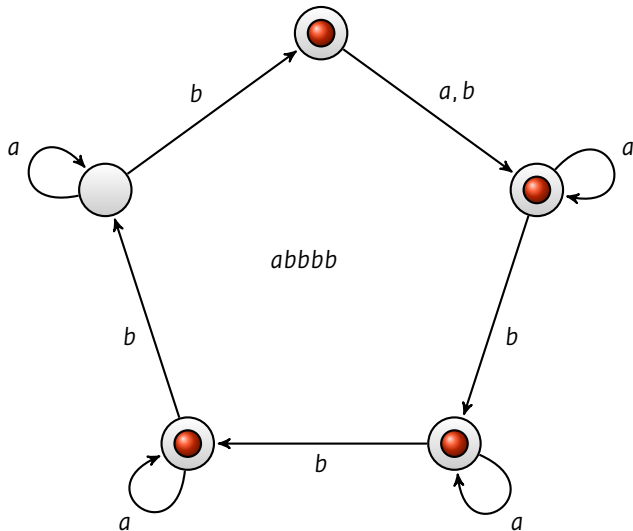
# Example

Černý's automaton  $\mathcal{C}_5$ :



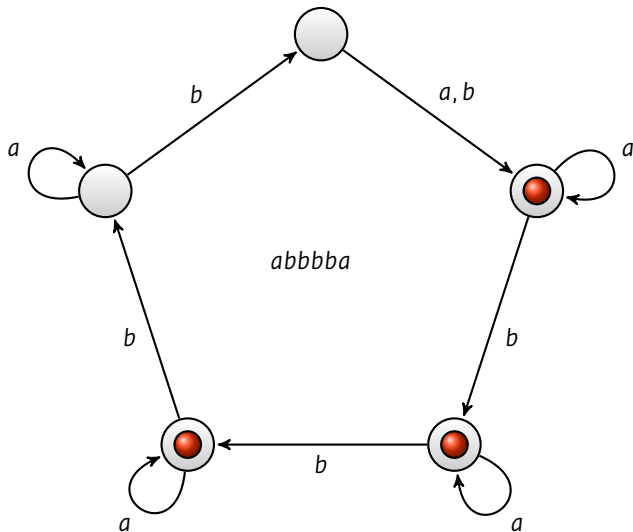
# Example

Černý's automaton  $\mathcal{C}_5$ :



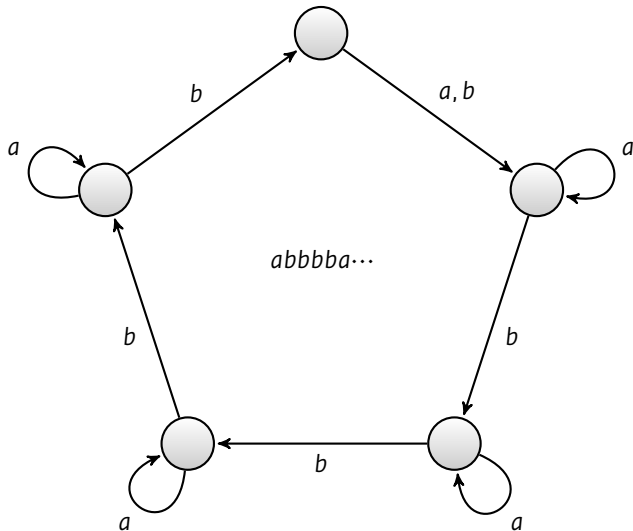
# Example

Černý's automaton  $\mathcal{C}_5$ :



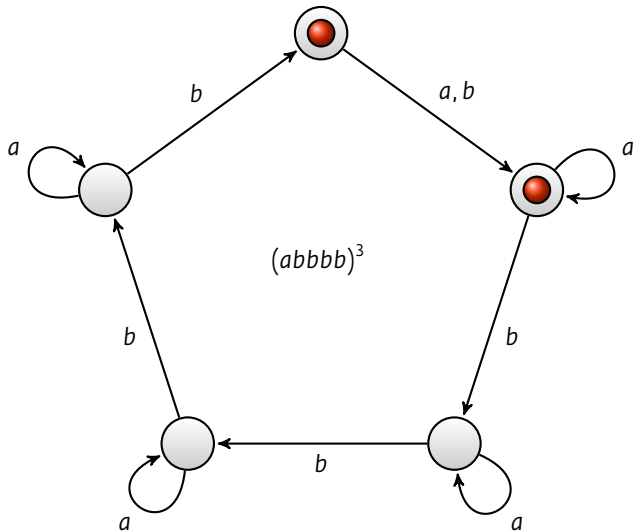
# Example

Černý's automaton  $\mathcal{C}_5$ :



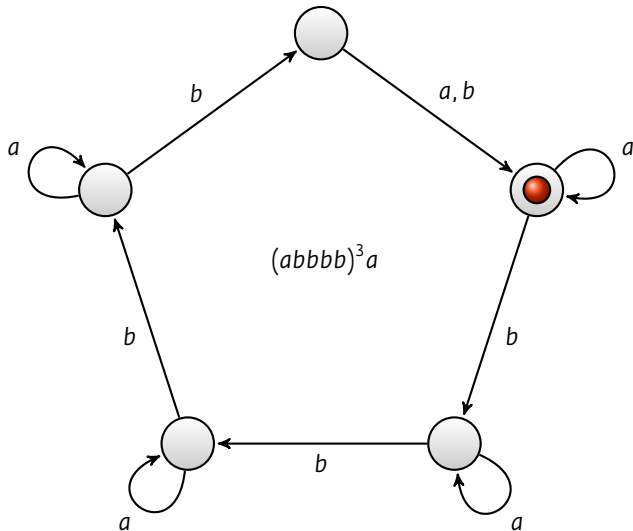
# Example

Černý's automaton  $\mathcal{C}_5$ :



# Example

Černý's automaton  $\mathcal{C}_5$ :



# Deciding Synchronicity

**Question:** How to decide whether an automaton is synchronizing?

**Observation:** An automaton is synchronizing iff there exists a path from  $Q$  to a singleton state in the powerset automaton.

## Lemma (Černý)

An automaton is synchronizing iff for each  $p, q \in Q$  there exists  $w \in \Sigma^*$  with  $\delta^*(p, w) = \delta^*(q, w)$ .

The latter property can be checked by examining a small part of the powerset automaton.

## Theorem

Whether an automaton is synchronizing can be decided in polynomial time.

**Next Question:** How to find a reset word?

# Computing a Reset Word

## Theorem

Computing a reset word can be done in polynomial time.

Polynomial-time procedure to find a reset word:

$r := \varepsilon; P := Q$

**while**  $|P| > 1$  **do**

**pick**  $p \neq q \in P$

**if** exists  $w \in \Sigma^*$  with  $\delta^*(p, w) = \delta^*(q, w)$  **then**

$r := r \cdot w; P := \delta^*(P, w)$

**else**

**output** “ $\mathcal{A}$  not synchronizing”

**end while**

**output**  $r$

**Note:** The resulting reset word might not be a shortest.

# Finding Short Reset Words

Question: What if we want to find short(est) reset words?

Several Problems:

- ▶ Given  $\mathcal{A}$  and  $k \in \mathbb{N}$ , decide whether there exists a reset word of length  $k$ . NP-complete
- ▶ Given  $\mathcal{A}$  and  $k \in \mathbb{N}$ , decide whether a shortest reset word has length  $k$ . DP-complete
- ▶ Given  $\mathcal{A}$  and  $k \in \mathbb{N}$ , compute a reset word of length  $k$ . FNP-complete
- ▶ Given  $\mathcal{A}$ , compute the length of a shortest reset word.  $\text{FP}^{\text{NP}[\log]}$ -complete
- ▶ Given  $\mathcal{A}$ , compute a shortest reset word.  $\text{FP}^{\text{NP}[\log]}$ -hard,  $\in \text{FP}^{\text{NP}}$

## Theorem (Eppstein)

Deciding whether an automaton has a reset word of a given length is NP-complete.

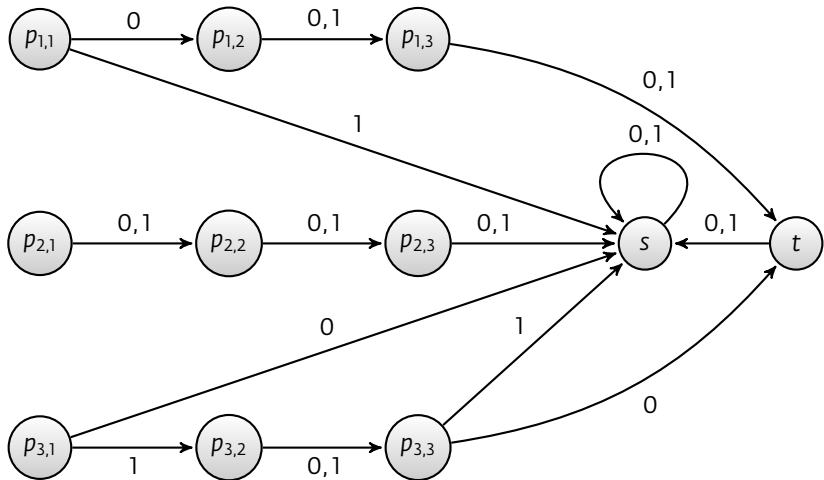
Membership in NP: Guess a reset word!

NP-hardness: Reduction from SAT.

# Epstein's Proof

$$\varphi = X_1 \wedge (X_3 \vee \neg X_3) \wedge (\neg X_1 \vee X_3)$$

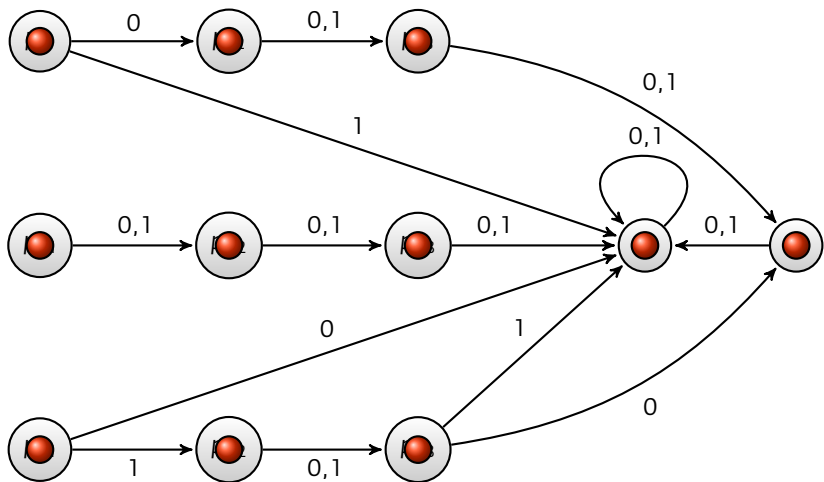
$$w = 101$$



# Eppstein's Proof

$$\varphi = X_1 \wedge (X_3 \vee \neg X_3) \wedge (\neg X_1 \vee X_3)$$

$$w = 101$$

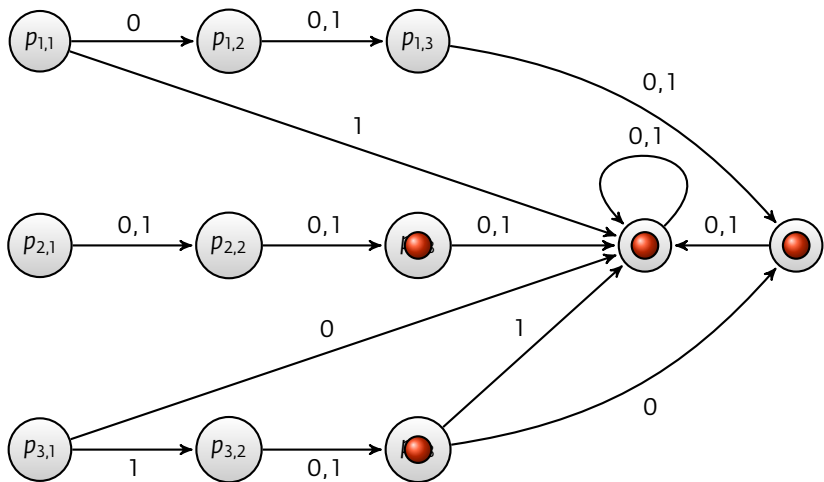




# Eppstein's Proof

$$\varphi = X_1 \wedge (X_3 \vee \neg X_3) \wedge (\neg X_1 \vee X_3)$$

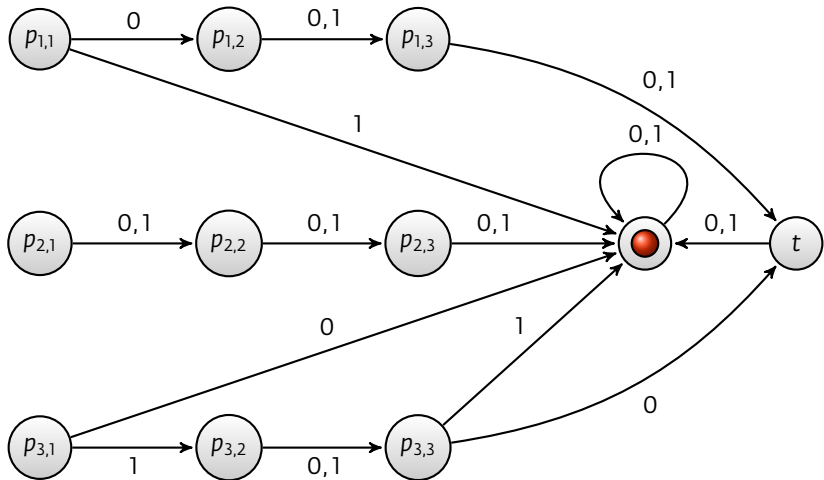
$$w = 101$$



# Epstein's Proof

$$\varphi = X_1 \wedge (X_3 \vee \neg X_3) \wedge (\neg X_1 \vee X_3)$$

$$w = 101$$



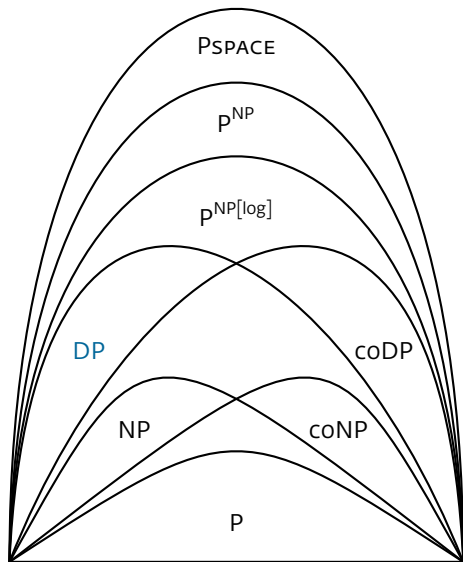
# The class DP

DP = closure of  $NP \cup coNP$  under intersection.

Characterization:  $L \in DP$  iff  
 $L = L_1 \cap L_2$  with  $L_1 \in NP$ ,  $L_2 \in coNP$ .

DP-complete problems:

- ▶ SAT-UNSAT
- ▶ EXACT-TSP
- ▶ CRITICAL-SAT
- ▶ ...



# DP-Completeness

## Theorem

Deciding, given  $\mathcal{A}$  and  $k \in \mathbb{N}$ , whether  $k$  is the length of a shortest reset word for  $\mathcal{A}$  is DP-complete.

Membership in DP: Easy!

$k$  is length of shortest reset word

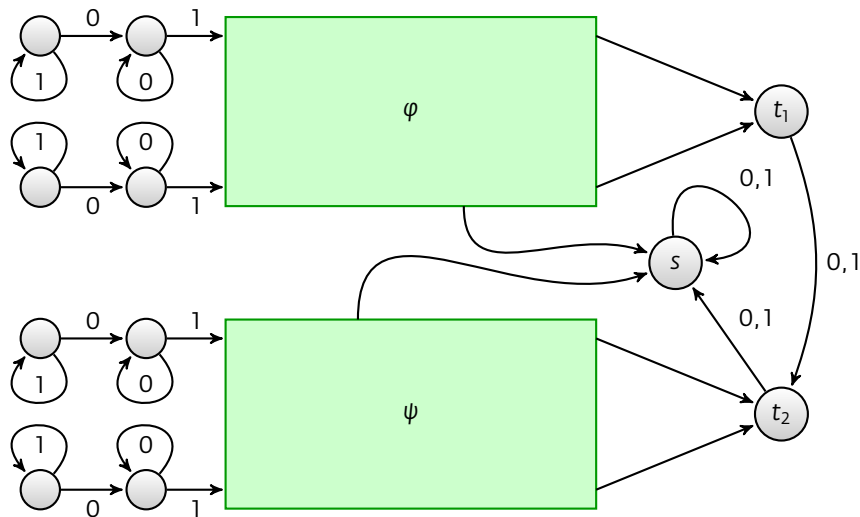
iff  $\mathcal{A}$  has reset word of length  $k$  (NP)

and  $\mathcal{A}$  has no reset word of length  $k - 1$  (coNP)

DP-hardness: Reduction from

$\text{SAT-UNSAT} = \{(\varphi, \psi) \mid \varphi \text{ is satisfiable and } \psi \text{ is unsatisfiable}\}.$

# DP-hardness



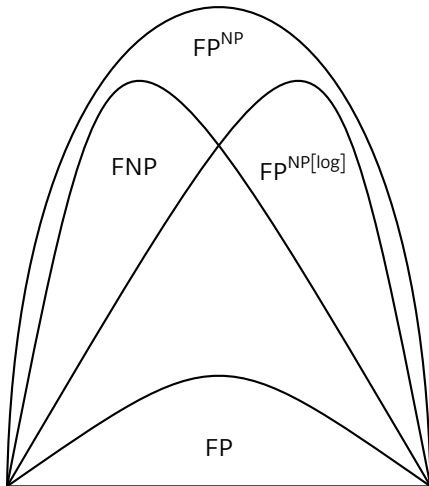
$\varphi$  satisfiable and  $\psi$  unsatisfiable  $\Leftrightarrow$  shortest reset word has length  $k + 3$ .

# Search Problems

**Search Problem:** described by a relation  $R \subseteq \Sigma^* \times \Sigma^*$

**Classes:**

- ▶ FP: polynomial time
- ▶ FNP: (polynomially balanced) polynomial-time relation
- ▶  $FP^{NP[\log]}$ : polynomial-time with oracle for NP ( $O(\log n)$  queries)
- ▶  $FP^{NP}$ : polynomial-time with oracle for NP



# Computing the Length of a Shortest Reset Word

## Theorem

Computing the length of a shortest reset word is  $\text{FP}^{\text{NP}^{\lceil \log \rceil}}$ -complete.

## Algorithm:

**if**  $\mathcal{A}$  is not synchronizing **then reject**

$low := -1$

$high := (n^3 - n)/6$

**while**  $high - low > 1$  **do**

$k := \lceil (low + high)/2 \rceil$

**if**  $\mathcal{A}$  has a reset word of length  $k$  **then**  $high := k$  **else**  $low := k$

**end while**

**return**  $high$

**Hardness:** Reduction from MAX-SAT-SIZE.

# More Results

## Theorem

Computing a reset word of length  $k$  is FNP-complete.

**Membership:** Immediate.

**Hardness:** Reduction from FSAT (Eppstein's reduction).

## Theorem

Computing a shortest reset word is in  $FP^{NP}$ .

**Membership:** Binary search through prefix tree.

Hardness for  $FP^{NP[\log]}$  follows from hardness result for computing length.

# Summary

