Two-Way Finite Automata
Old and Recent Results

Giovanni Pighizzini

Dipartimento di Informatica
Università degli Studi di Milano

Automata and JAC 2012
La Marana, Corsica, France
September 19-21, 2012
Finite State Automata

One-way version

At each step the input head is moved one position to the right

- **1DFA**: *deterministic* transitions
- **1NFA**: *nondeterministic* transitions
A Very Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$H_n = (a + b)^{n-1} a (a + b)^*$$
A Very Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ H_n = (a + b)^{n-1}a(a + b)^* \]

Check the \( n \)th symbol from the left!
\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ H_n = (a + b)^{n-1} a (a + b)^* \]

Check the \( n \)th symbol from the left!

Ex. \( n = 4 \)
A Very Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ H_n = (a + b)^{n-1} a (a + b)^* \]

Check the \( n \)th symbol from the left!

Ex. \( n = 4 \)
A Very Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$H_n = (a + b)^{n-1}a(a + b)^*$$

Check the $n$th symbol from the left!

Ex. $n = 4$

Ex.

\[
\begin{array}{c|c|c}
    a & b & b \\
\end{array}
\]

\[
\begin{array}{c|c|c}
    3 \\
\end{array}
\]
A Very Preliminary Example

\[ \Sigma = \{ a, b \}, \text{ fixed } n > 0: \]

\[ H_n = (a + b)^{n-1}a(a + b)^* \]

Check the \( n \)th symbol from the left!

Ex. \( n = 4 \)
A Very Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ H_n = (a + b)^{n-1}a(a + b)^* \]

Check the \( n \)th symbol from the left!

Ex. \( n = 4 \)

\[
\begin{array}{cccc}
a & b & b & a \\
\end{array}
\]

\[ q_\gamma \]
A Very Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$H_n = (a + b)^{n-1}a(a + b)^*$$

Check the $n$th symbol from the left!

Ex. $n = 4$

```
Input: a b b a b a
```

Diagram:

```
q_Y
```

DFA: 1DFA with 2 states
A Very Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$H_n = (a + b)^{n-1}a(a + b)^*$$

Check the $n$th symbol from the left!

Ex. $n = 4$

1DFA: $n + 2$ states
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^n a(a + b)^{n-1} \]
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a(a + b)^{n-1} \]

Check the \( n \)th symbol from the right!
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

How to locate it?
A Preliminary Example

Σ = \{a, b\}, fixed \(n > 0\):

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \(n\)th symbol from the right!

How to locate it?

Use nondeterminism!
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ I_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

How to locate it?

Use nondeterminism!

**Guess** Reading the symbol \( a \) the automaton can guess that it is the \( n \)th symbol from the right

**Verify** In the next steps the automaton verifies such a guess
A Preliminary Example

\[ \Sigma = \{ a, b \}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)
A Preliminary Example

$$\Sigma = \{a, b\}, \text{ fixed } n > 0:$$

$$l_n = (a + b)^* a (a + b)^{n-1}$$

Check the $n$th symbol from the right!

Ex. $n = 4$
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \(n\)th symbol from the right!

Ex. \(n = 4\)

\[ b \quad a \quad a \]

\[ q_0 \]

guess

4th symbol from the right
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^{n}a(a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

Ex. \( n = 4 \)

\[ b \ a \ a \ a \]

\[ 3 \]

verify
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

Ex.

```
| b | a | a | a | b |
```

\[ 2 \]

verify
A Preliminary Example

Σ = \{a, b\}, fixed \( n > 0 \):

\[ l_n = (a + b)^n a(a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

\[
\begin{array}{cccccc}
  b & a & a & a & b & a \\
\end{array}
\]

\[
\begin{array}{ccc}
  1 & \text{verify} \\
\end{array}
\]
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

\[ b \ a \ a \ a \ b \ a \]

1NFA: \( n + 1 \) states
A Preliminary Example

\[ \Sigma = \{ a, b \}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^n a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

![Diagram](image)
A Preliminary Example

$$\Sigma = \{a, b\}, \text{ fixed } n > 0:$$

$$l_n = (a + b)^* a (a + b)^{n-1}$$

Check the $n$th symbol from the right!

Very nice!

...but I need a deterministic automaton...
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$l_n = (a + b)^* a (a + b)^{n-1}$$

Check the $n$th symbol from the right!

Very nice!

...but I need a deterministic automaton...

Remember the previous $n$ input symbols!
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^n a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$l_n = (a + b)^n a (a + b)^{n-1}$

Check the $n$th symbol from the right!

Ex. $n = 4$
A Preliminary Example

\[ \Sigma = \{ a, b \}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

\[
\begin{array}{cccc}
 & b & a & a \\
\end{array}
\]

\[ \rightarrow \]

\[
\begin{array}{ccc}
 b & a & a \\
\end{array}
\]

YES!

1DFA: 2 states. . . but I need a smaller deterministic automaton...

This is the smallest one!

However...
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$l_n = (a + b)^* a (a + b)^{n-1}$$

Check the $n$th symbol from the right!

Ex. $n = 4$

```
  b a a a a b  
```

```
  b a a a a b  
  b a a a a b  
```

Yes!

1DFA: 2 states... but I need a smaller deterministic automaton...

This is the smallest one!

However...
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$l_n = (a + b)^* a (a + b)^{n-1}$$

Check the $n$th symbol from the right!

Ex. $n = 4$

```
input: b a a a a b
```

This is the smallest one!
A Preliminary Example

\( \Sigma = \{a, b\} \), fixed \( n > 0 \):

\[ l_n = (a + b)^n a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

```
Input: b a a a a b a
```

```
YES!
```

```
aba
```

\[ a a b a \]
A Preliminary Example

\[ \Sigma = \{a, b\} \text{, fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

![Input string](image)

YES!

1DFA: \( 2^n \) states
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0:\]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Ex. \( n = 4 \)

1DFA: \( 2^n \) states

...but I need a smaller deterministic automaton...
A Preliminary Example

\[
\Sigma = \{a, b\}, \text{ fixed } n > 0:
\]

\[
l_n = (a + b)^* a (a + b)^{n-1}
\]

Check the \(n\)th symbol from the right!

Ex. \(n = 4\)

\[
\begin{array}{ccccc}
& b & a & a & a & b \\
\end{array}
\]

YES!

1DFA: \(2^n\) states

...but I need a smaller deterministic automaton...

This is the smallest one!

However...
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a(a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$I_n = (a + b)^*a(a + b)^{n-1}$$

Check the $n$th symbol from the right!

...if the head can be moved back...

Ex. $n = 4$

![Diagram of a two-way deterministic automaton with states $q_0$, $b$, and transitions leading from $q_0$ to $b$.]
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

\[
\begin{array}{c}
\text{b} \quad \text{a} \\
\end{array}
\]

\[
\begin{array}{c}
q_0 \\
\end{array}
\]

\[ \text{YES!} \]

Two-way deterministic automaton (2DFA): \( n + ... \) states
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

---

Two-way deterministic automaton (2DFA):

\[ \begin{array}{c}
q_0 \\
\downarrow \\
b \quad a \\
\quad \quad a \\
\end{array} \]
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^n a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

\[
\begin{array}{cccc}
  b & a & a & a \\
\end{array}
\]

\[
\begin{array}{cccc}
  q_0 & \\
\end{array}
\]
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

Ex. \( n = 4 \)

\[
\begin{array}{cccc}
& a & a & a & b \\
q_0 & & & & \\
\end{array}
\]
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

\[
\begin{array}{cccccc}
  b & a & a & a & b & a \\
\end{array}
\]

YES!

Two-way deterministic automaton (2DFA): \( n + \)... \( n \) states
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$l_n = (a + b)^* a (a + b)^{n-1}$

Check the $n$th symbol from the right!

...if the head can be moved back...

Ex. $n = 4$

Ex. $n = 4$

```plaintext
| b | a | a | a | b | a | - |
```

$\text{right endmarker}$

$q_0$
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^n a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

Ex. \( n = 4 \)
A Preliminary Example

$\Sigma = \{a, b\}$, fixed $n > 0$:

$$l_n = (a + b)^* a (a + b)^{n-1}$$

Check the $n$th symbol from the right!

...if the head can be moved back...

Ex. $n = 4$

Ex. $n = 4$
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^*a(a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

```
| b | a | a | a | b | a | \n---|---|---|---|---|---|
```

![Diagram with state transitions]

YES!
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

\[
\begin{array}{ccccccc}
  b & a & a & a & b & a & \mid \\
\end{array}
\]

4  \quad \text{decision}

\text{if input symbol} = a \text{ then accept}
\text{else reject}
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^*a(a + b)^{n-1} \]

Check the \( n \text{th} \) symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

```
 b a a a b a ⊟
```

YES!

```
 4
```

decision
if input symbol = a then accept
ever reject
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^n a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

...if the head can be moved back...

Ex. \( n = 4 \)

\[ \begin{array}{cccccc}
  b & a & a & a & b & a \\
\end{array} \]

YES!

Two-way deterministic automaton (2DFA): \( n + ... \) states
A Preliminary Example

\[ \Sigma = \{ a, b \}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^*a(a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Summing up, \( l_n \) is accepted by

- a 1NFA and a 2DFA with approximatively the same number of states \( n + \ldots \)
- each 1DFA is exponentially larger (\( \geq 2^n \) states)
A Preliminary Example

\[ \Sigma = \{a, b\}, \text{ fixed } n > 0: \]

\[ l_n = (a + b)^* a (a + b)^{n-1} \]

Check the \( n \)th symbol from the right!

Summing up, \( l_n \) is accepted by

- a 1NFA and a 2DFA with approximatively the same number of states \( n + \ldots \)
- each 1DFA is exponentially larger (\( \geq 2^n \) states)

*In this example,*

nondeterminism can be removed using two-way motion keeping approximatively the same number of states
Two-Way Automata: Technical Details

- Input surrounded by the endmarkers $\vdash$ and $\dashv$
- Moves
  - to the left
  - to the right
  - stationary
- Initial configuration
- Accepting configuration
- Infinite computations are possible
- Deterministic (2DFA) and nondeterministic (2NFA) versions
What about the power of these models?
What about the power of these models?

They share the same computational power, namely they characterize the class of *regular languages*,
What about the power of these models?

They share the same computational power, namely they characterize the class of *regular languages*, however...

...some of them are more succinct
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

1NFA: \( n + 2 \) states
Main Example: $L_n = (a + b)^* a(a + b)^{n-1} a(a + b)^*$

1NFA: $n + 2$ states
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

Minimum 1DFA: \( 2^n + 1 \) states
Main Example: \( L_n = (a + b)^* a(a + b)^{n-1} a(a + b)^* \)

2DFA?

Even scanning from the right it seems that we need to remember a “window” of \( n \) symbols

We use a different technique!
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

Even scanning from the right it seems that we need to remember a “window” of \( n \) symbols.

2DFA?

We use a different technique!
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

2DFA ?

Even scanning from the right it seems that we need to remember a “window” of \( n \) symbols

We use a different technique!
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

\[
\begin{array}{cccccccccc}
\hline
\text{ } & b & b & a & b & a & a & b & a & a \\
\hline
\end{array}
\]

\( n = 4 \)

while input symbol \( \not= a \) do move to the right

Exception: if input symbol = \( \sqsubset \) then reject

2DFA: 2 \( n + ... \) states
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

\[ \begin{array}{cccccccccc}
| & b & b & a & b & a & a & b & a & a & - |
\end{array} \quad n = 4
\]

**while** input symbol \( \neq a \) **do** move to the right
Main Example: $L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$

```
- b b a b a a b a a a -
```

$n = 4$

while input symbol $\neq a$ do move to the right
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

\[
\vdash \quad b \quad b \quad a \quad b \quad a \quad a \quad b \quad a \quad a \quad a \quad \vdash \\
\]

\( n = 4 \)

\textbf{while} input symbol \( \neq a \) do move to the right

move \( n \) squares to the right
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

\[
\begin{array}{ccccccccc}
- & b & b & a & b & a & a & b & a & a & a & - \\
\end{array}
\]

\( n = 4 \)

\textbf{while} input symbol \( \neq a \) \textbf{do} move to the right

move \( n \) squares to the right
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

\[
\begin{array}{cccccc}
| & b & b & a & b & a & a & b & a & a & a & | \\
\end{array}
\]

\( n = 4 \)

while input symbol \( \neq a \) do move to the right
move \( n \) squares to the right
Main Example: \( L_n = (a + b)^n a (a + b)^{n-1} a (a + b)^* \)

\[
\begin{array}{cccccccccc}
\rightarrow & b & b & a & b & a & a & b & a & a & \rightarrow \\
\end{array}
\quad n = 4
\]

while input symbol \( \neq a \) do move to the right
move \( n \) squares to the right
Main Example: $L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^*$

while input symbol $\neq a$ do move to the right
move $n$ squares to the right
if input symbol $= a$ then accept
else move $n - 1$ cells to the left
repeat from the first step
Main Example: $L_n = (a + b)^n a (a + b)^{n-1} a (a + b)^*$

While input symbol $\neq a$ do move to the right
move $n$ squares to the right
if input symbol $= a$ then accept
else move $n - 1$ cells to the left
repeat from the first step
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

\[
\begin{array}{cccccccc}
\vdash & b & b & a & b & a & a & b & a & a & a & \vdash \\
\end{array}
\quad n = 4
\]

\begin{enumerate}
\item \textbf{while} input symbol \( \neq a \) \textbf{do} move to the right
\item move \( n \) squares to the right
\item \textbf{if} input symbol = \( a \) \textbf{then} accept
\item else move \( n - 1 \) cells to the left
\item repeat from the first step
\end{enumerate}
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

\[
\begin{array}{cccccccc}
| & b & b & a & b & a & a & b & a & a & a & |- \\
\end{array}
\]

\( n = 4 \)

**while** input symbol \( \neq a \) **do** move to the right
move \( n \) squares to the right
**if** input symbol = \( a \) **then** accept
**else** move \( n - 1 \) cells to the left
**repeat** from the first step

Exception: **if** input symbol = \( \vdash \) **then** reject

2DFA: \( 2^n + \ldots \) states
Main Example: $L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$

while input symbol $\neq a$ do move to the right
move $n$ squares to the right
if input symbol $= a$ then accept
else move $n - 1$ cells to the left
repeat from the first step

Exception: if input symbol $= \dashv$ then reject

2DFA: $2^n + \ldots$ states

| b b a b a a b a a a | $n = 4$
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

\[ \begin{array}{cccccccccc}
\text{ while input symbol } \neq a & \text{ do move to the right} \\
\text{ move } n \text{ squares to the right} \\
\text{ if input symbol } = a & \text{ then accept} \\
\text{ else move } n - 1 \text{ cells to the left} \\
\text{ repeat from the first step} \\
\end{array} \]
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

\[
\begin{array}{ccccccccccc}
\_ & b & b & a & b & a & a & b & a & a & a & \_
\end{array}
\]

\( n = 4 \)

\( \text{while input symbol} \neq a \text{ do move to the right} \)

\( \text{move } n \text{ squares to the right} \)

\( \text{if input symbol} = a \text{ then accept} \)

\( \text{else move } n - 1 \text{ cells to the left} \)

\( \text{repeat from the first step} \)
Main Example: $L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$

```
| b | b | a | b | a | a | b | a | a | a |
```

$n = 4$

**while** input symbol $\neq a$ **do** move to the right

move $n$ squares to the right

**if** input symbol $= a$ **then** accept

**else** move $n - 1$ cells to the left

**repeat** from the first step
Main Example: $L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$

```
- b b a b a a b a a a -
```

$n = 4$

while input symbol $\neq a$ do move to the right
move $n$ squares to the right
if input symbol $= a$ then accept
else move $n - 1$ cells to the left
repeat from the first step
Main Example: $L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^*$

$$\begin{array}{cccccccc}
\vdash & b & b & a & b & a & a & b & a & a & a & \vdash
\end{array} \quad n = 4$$

while input symbol $\neq a$ do move to the right
move $n$ squares to the right
if input symbol $= a$ then accept
else move $n - 1$ cells to the left
repeat from the first step
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

\[
\begin{array}{cccccccc}
\leftarrow & b & b & a & b & a & a & b & a & a & a & \rightarrow \\
\end{array}
\]

\( n = 4 \)

while input symbol \( \neq a \) do move to the right
move \( n \) squares to the right
if input symbol = a then accept
else move \( n - 1 \) cells to the left
repeat from the first step

Exception: if input symbol = \( \rightarrow \) then reject
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

\[
\begin{array}{cccccccc}
\vdash b & b & a & b & a & b & a & a & a & \vdash \\
\end{array}
\quad n = 4
\]

while input symbol \( \neq a \) do move to the right
move \( n \) squares to the right
if input symbol = a then accept
else move \( n - 1 \) cells to the left
repeat from the first step

Exception: if input symbol = \( \vdash \) then reject

2DFA: \( 2n + \ldots \) states
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

A different algorithm

\[
\begin{array}{cccccccccc}
\_ & b & b & a & a & a & b & a & b & b & b & \_ & n = 4
\end{array}
\]
Main Example: $L_n = (a + b)^* a(a + b)^{n-1} a(a + b)^*$

A different algorithm

```
├── b ├── b ├── a ─── a ─── a ─── a ─── b ─── a ─── b ─── b ─── b ─── n = 4
```

Check positions $k$ s.t. $k \equiv 1 \pmod{n}$
Main Example: $L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^*$

A different algorithm

\[
\begin{array}{cccccccccc}
| & b & b & a & a & a & b & a & b & b & b & |
\end{array}
\quad n = 4
\]

Check positions $k$ s.t. $k \equiv 1 \pmod{n}$
Check positions $k$ s.t. $k \equiv 2 \pmod{n}$
Even this strategy can be implemented using $O(n)$ states!

Sweeping automata:
- Deterministic transitions
- Head reversals only at the endmarkers
Main Example: $L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$

A different algorithm

Check positions $k$ s.t. $k \equiv 1 \pmod{n}$
Check positions $k$ s.t. $k \equiv 2 \pmod{n}$

...
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

A different algorithm

\[ \overline{\text{ b } b \text{ a } a \text{ a } a \text{ b } a \text{ b } b \text{ a } b \text{ b } \text{ } n = 4} \]

Check positions \( k \) s.t. \( k \equiv 1 \pmod{n} \)
Check positions \( k \) s.t. \( k \equiv 2 \pmod{n} \)

\[ \ldots \]
Check positions \( k \) s.t. \( k \equiv n \pmod{n} \)
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

A different algorithm

\[
\begin{array}{cccccccccc}
\downarrow & b & b & a & a & a & a & b & a & b & b & a & b & b & \downarrow \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}
\]

\( n = 4 \)

Check positions \( k \) s.t. \( k \equiv 1 \pmod{n} \)
Check positions \( k \) s.t. \( k \equiv 2 \pmod{n} \)
\ldots
Check positions \( k \) s.t. \( k \equiv n \pmod{n} \)

Even this strategy can be implemented using \( O(n) \) states!
Main Example: \( L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \)

A different algorithm

```
| b | b | a | a | a | b | a | b | b | a | b | b | b |  
```

\( n = 4 \)

Check positions \( k \) s.t. \( k \equiv 1 \pmod{n} \)
Check positions \( k \) s.t. \( k \equiv 2 \pmod{n} \)
\ldots
Check positions \( k \) s.t. \( k \equiv n \pmod{n} \)

Even this strategy can be implemented using \( O(n) \) states!

**Sweeping automata:**
- Deterministic transitions
- Head reversals *only at the endmarkers*
Main Example: \( L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^* \)

Summing up,

- \( L_n \) is accepted by
  - a 1NFA
  - a 2DFA
  - a sweeping automaton
  with \( O(n) \) states
- Each 1DFA is exponentially larger

Also for this example, nondeterminism can be removed using two-way motion keeping a linear number of states

Is it always possible to replace nondeterminism by two-way motion without increasing too much the size?
Main Example: $L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^*$

Summing up,
- $L_n$ is accepted by
  - a 1NFA
  - a 2DFA
  - a sweeping automaton

  with $O(n)$ states
- Each 1DFA is exponentially larger

Also for this example,

nondeterminism can be removed using two-way motion
keeping a linear number of states

Is it always possible
to replace nondeterminism by two-way motion
without increasing too much the size?
Costs of the Optimal Simulations Between Automata

1NFA 2DFA 2NFA

\[ 2^n \quad O(2^n \log n) \quad O(2^{n^2}) \]

1DFA

[Rabin & Scott ’59, Shepardson ’59, Meyer & Fischer ’71, …]
Costs of the Optimal Simulations Between Automata

1NFA \rightarrow ? \rightarrow 2DFA \rightarrow ? \rightarrow 2NFA

\begin{align*}
1DFA & \quad 2^n \\
2DFA & \quad O(2^n \log n) \\
2NFA & \quad O(2^{n^2})
\end{align*}

[Rabin&Scott ’59, Shepardson ’59, Meyer&Fischer ’71, …]

**Question**

*How much the possibility of moving the input head forth and back is useful to eliminate the nondeterminism?*
Costs of the Optimal Simulations Between Automata

Problem ([Sakoda&Sipser ’78])

Do there exist polynomial simulations of

- 1NFAs by 2DFAs
- 2NFAs by 2DFAs?
Problem ([Sakoda&Sipser ’78])

Do there exist polynomial simulations of
- 1NFAs by 2DFAs
- 2NFAs by 2DFAs?

Conjecture

These simulations are not polynomial.
Costs of the Optimal Simulations Between Automata

1NFA ➔ ? ➔ 2DFA ➔ ? ➔ 2NFA

1DFA

2^n
O(2^n \cdot \log n)
O(2^{n^2})

- Exponential upper bounds deriving from the simulations of 1NFAs and 2NFAs by 1DFAs
- Polynomial lower bound \( \Omega(n^2) \) for the cost of the simulation of 1NFAs by 2DFAs

[Chrobak ’86]
Sakoda and Sipser Question

- Very difficult in its general form
- Not very encouraging obtained results:
  
  Lower and upper bounds too far
  (Polynomial vs exponential)

- Hence:

  Try to attack restricted versions of the problem!
NFAs vs 2DFAs: Restricted Versions

(i) Restrictions on the resulting machines (2DFAs)
   ▶ sweeping automata [Sipser ’80]
   ▶ oblivious automata [Hromkovič&Schnitger ’03]
   ▶ “few reversal” automata [Kapoutsis ’11]

(ii) Restrictions on the languages
   ▶ unary regular languages [Geffert Mereghetti&P ’03]

(iii) Restrictions on the starting machines (2NFAs)
   ▶ outer nondeterministic automata [Guillon Geffert&P ’12]
NFAs vs 2DFAs: Restricted Versions

(i) Restrictions on the resulting machines (2DFAs)
- sweeping automata [Sipser ’80]
- oblivious automata [Hromkovič&Schnitger ’03]
- “few reversal” automata [Kapoutsis ’11]

(ii) Restrictions on the languages
- unary regular languages [Geffert Mereghetti&P ’03]

(iii) Restrictions on the starting machines (2NFAs)
- outer nondeterministic automata [Guillon Geffert&P ’12]
(i) Restrictions on the resulting machines (2DFAs)
   - sweeping automata [Sipser ’80]
   - oblivious automata [Hromkovič&Schnitger ’03]
   - “few reversal” automata [Kapoutsis ’11]

(ii) Restrictions on the languages
   - unary regular languages [Geffert Mereghetti&P ’03]

(iii) Restrictions on the starting machines (2NFAs)
   - outer nondeterministic automata [Guillon Geffert&P ’12]
\[ L_n = (a + b)^*(a + b)^{n-1}a(a + b)^* \] Again!

Naïf algorithm: compare input positions \( i \) and \( i + n, \ i = 1, 2, \ldots \)

\[ \leftarrow b \ b \ a \ b \ a \ a \ b \ a \ a \ a \ \leftarrow \]

\[ n = 4 \]
\[ L_n = (a + b)^n a (a + b)^{n-1} a (a + b)^* \]

Naïf algorithm: compare input positions \( i \) and \( i + n \), \( i = 1, 2, \ldots \)
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$  Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

| $\leftarrow b$ $b$ $a$ $b$ $a$ $a$ $b$ $a$ $a$ $a$ $\rightarrow$ |

$b$

$n = 4$
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

\[\vdash b b a b a a b a a a \vdash\]  \[n = 4\]
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$  Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

\[ \vdash b b a b a a b a a a \vdash \]

\[ n = 4 \]
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \]

Again!

Naïf algorithm: compare input positions \( i \) and \( i + n \), \( i = 1, 2, \ldots \)

\[ \vdash b b a b a a b a a a \]

\[ n = 4 \]
$L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^*$

Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

├── b b a b a a b a a a ┤  $n = 4$
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

$\vdash b b a b a a b a a a \vdash$

$n = 4$
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \] Again!

Naïf algorithm: compare input positions \( i \) and \( i + n \), \( i = 1, 2, \ldots \)

\[ \vdash \overline{b b a b a a b a a a} \]

\[ n = 4 \]
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

The string can be accepted!
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \] Again!

Naïf algorithm: compare input positions \( i \) and \( i + n \), \( i = 1, 2, \ldots \)

\[ \begin{array}{cccccccc}
| & b & b & a & b & a & b & a & a & a & | \\
\end{array} \]

\( n = 4 \)

The string can be accepted!

...but our automaton continues to scan
$L_n = (a + b)^n a (a + b)^{n-1} a (a + b)^n$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

$\vdash b b a b a a b a a a \vdash$  

$n = 4$
$L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^*$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

\[ \begin{array}{cccccc}
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\vdots & b & b & a & b & a \\
\end{array} \]

$n = 4$
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \]

Again!

Naïf algorithm: compare input positions \( i \) and \( i + n \), \( i = 1, 2, \ldots \)

\[ \vdash b b a b a a b a a a \vdash \]

\( n = 4 \)
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \] Again!

Naïf algorithm: compare input positions \( i \) and \( i + n \), \( i = 1, 2, \ldots \)

```
|   | b | b | a | b | a | a | b | a | a | a |
```

\( n = 4 \)

Even in this case \( O(n) \) states!
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \] Again!

Naïf algorithm: compare input positions \( i \) and \( i + n, \ i = 1, 2, \ldots \)

\[ \text{⊢ } b \ b \ a \ b \ a \ a \ b \ a \ a \ a \ ] \quad n = 4

Even in this case \( O(n) \) states!

**Oblivious Automata:**
- Deterministic transitions
- Same “trajectory” on all inputs of the same length
$L_n = (a + b)^*a(a + b)^{n-1}a(a + b)^*$ Again!

Naïf algorithm: compare input positions $i$ and $i + n$, $i = 1, 2, \ldots$

```
|   b | b | a | b | a | b | a | a | a | --|
```

$n = 4$

**Number of head reversals:**
On input of length $m$:

- This technique uses about $2m$ reversals, a *linear number* in the input length
- The “sweeping” algorithm uses about $2n$ reversals, a *constant number* in the input length
\[ L_n = (a + b)^* a (a + b)^{n-1} a (a + b)^* \] Again!

Naïf algorithm: compare input positions \( i \) and \( i + n, \ i = 1, 2, \ldots \)

\[
\begin{array}{ccccccccc}
| & b & b & a & b & a & a & b & a & a & a & | \\
\end{array}
\]

\( n = 4 \)

Number of head reversals:
On input of length \( m \):

- This technique uses about \( 2m \) reversals, a **linear number** in the input length
- The “sweeping” algorithm uses about \( 2n \) reversals, a **constant number** in the input length
Another Restricted Model

“Few Reversal” Automata [Kapoutsis ’11]:

- On input of length $m$ the number of reversals is $o(m)$, i.e., sublinear
- We consider only the deterministic case

Theorem ([Kapoutsis&P ’12])

Each 2DFA using $o(m)$ reversals actually uses $O(1)$ reversals
Another Restricted Model

“Few Reversal” Automata [Kapoutsis ’11]:

- On input of length \( m \) the number of reversals is \( o(m) \), i.e., sublinear
- We consider only the deterministic case

Theorem ([Kapoutsis & P ’12])

Each 2DFA using \( o(m) \) reversals actually uses \( O(1) \) reversals
Restricted Models: Separations

oblivious  sweeping  few reversals
Restricted Models: Separations

oblivious sweeping \( O(n^2) \) few reversals
Restricted Models: Separations

oblivious

sweeping

few reversals

$O(n^2)$
Restricted Models: Separations

1NFA

[Sipser '80]

oblivious ← ------------ sweeping ------------- → few reversals

\[ O(n^2) \]

exp separation
Restricted Models: Separations

1NFA

[Sipser '80]

oblivious \textarrow{sweeping} \textarrow{few reversals}

[Berman '80, Micali '81]

2DFA

\textarrow{O(n^2)}

exp separation
Restricted Models: Separations

1NFA

[Hromkovič&Schnitger ’03]

oblivious ←------------ sweeping ------------→ few reversals

2DFA

\( O(n^2) \)

exp separation
Restricted Models: Separations

1NFA

[Hromkovič&Schnitger ’03]

oblivious ←— sweeping ——→ few reversals

2DFA

[Kutrib Malcher&F ’12]

$O(n^2)$

exp separation
Restricted Models: Separations

1NFA

[Hromkovič & Schnitger ’03]

[Kutrib Malcher & P ’12]

oblivious ←-----→ sweeping ←-----→ few reversals

2DFA

[O(n^2)]

exp separation

[Kutrib Malcher & P ’12]
Restricted Models: Separations

1NFA

oblivious  sweeping  few reversals

2DFA

\[ O(n^2) \]

exp separation

[Kapoutsis '11]
Restricted Models: Separations

1NFA

[Oblivious sweeping few reversals]

[Kapoutsis'11]

2DFA

[Oblivious sweeping few reversals]

[Kapoutsis'11]

\[ O(n^2) \]

\[ \exp \text{separation} \]
Restricted Models: Separations

1NFA

[Oblivious sweeping few reversals]

2DFA

[Kapoutsis'11]

[O(\(n^2\)) exp separation]
Restricted Models: Separations

2NFA

\[ O(n^2) \]

exp

separation

2DFA

oblivious

sweeping

few reversals
Restricted Models: Separations

1NFA

[O(n^2)]

exp separation

2DFA

[46, 56, 57] Katherine Malcher & P'12

oblivious sweeping few reversals
Problem ([Sakoda&Sipser ’78])

Do there exist polynomial simulations of

- 1NFAs by 2DFAs
- 2NFAs by 2DFAs?

Another possible restriction:

The unary case $\#\Sigma = 1$
The costs of the optimal simulations between automata are different in the unary and in the general case.
Optimal Simulation Between Unary Automata

The costs of the optimal simulations between automata are different in the unary and in the general case.

\[ e^{\Theta(\sqrt{n \ln n})} \]

[Chrobak '86]
The costs of the optimal simulations between automata are different in the unary and in the general case.

\[ e^{\Theta(\sqrt{n \ln n})} \]

[Chrobak ’86]
The costs of the optimal simulations between automata are different in the unary and in the general case.

\[ e^{\Theta(\sqrt{n \ln n})} \]

[Moreghetti & P '01]
The costs of the optimal simulations between automata are different in the unary and in the general case.
The costs of the optimal simulations between automata are different in the unary and in the general case.

$2\text{NFA} \rightarrow 1\text{DFA}$ follows from $2\text{NFA} \rightarrow 1\text{NFA}$.
The costs of the optimal simulations between automata are different in the unary and in the general case.

\[ e^{\Theta(\sqrt{n \ln n})} \]

In the unary case this question is solved!
(polynomial conversion)
Optimal Simulation Between Unary Automata

The costs of the optimal simulations between automata are different in the unary and in the general case.

2NFA $\rightarrow$ 2DFA

Even in the unary case this question is open!

- $e^{\Theta(\sqrt{n \ln n})}$ upper bound (from 2NFA $\rightarrow$ 1DFA)
- $\Omega(n^2)$ lower bound (from 1NFA $\rightarrow$ 2DFA)
The costs of the optimal simulations between automata are different in the unary and in the general case.

2NFA $\rightarrow$ 2DFA

Even in the unary case this question is open!

- $e^{\Theta(\sqrt{n \ln n})}$ upper bound (from 2NFA $\rightarrow$ 1DFA)
- $\Omega(n^2)$ lower bound (from 1NFA $\rightarrow$ 2DFA)
The costs of the optimal simulations between automata are different in the unary and in the general case.

\[ e^{\Theta(\sqrt{n \ln n})} \]

Even in the unary case, this question is open!

\[ e^{\Theta(\sqrt{n \ln n})} \] upper bound
\[ \Omega(n^2) \] lower bound

A better upper bound \( e^{O(\ln^2 n)} \) has been proved!
Quasi Sweeping Automata (qsNFA):

- nondeterministic choices and
- head reversals

are possible only when the head is visiting the endmarkers

Theorem (Quasi Sweeping Simulation)

Each n-state unary 2NFA A can be transformed into a 2NFA M s.t.

- M is quasi sweeping
- M has at most \( N \leq 2n + 2 \) states
- M and A are “almost equivalent” (possible differences only for inputs of length \( \leq 5n^2 \))
A Normal Form for Unary 2NFAs
[Geffert Mereghetti&P ’03]

Quasi Sweeping Automata (qsNFA):
- nondeterministic choices and
- head reversals

are possible only when the head is visiting the endmarkers

Theorem (Quasi Sweeping Simulation)

Each n-state unary 2NFA A can be transformed into a 2NFA M s.t.
- M is quasi sweeping
- M has at most $N \leq 2n + 2$ states
- M and A are “almost equivalent”
  (possible differences only for inputs of length $\leq 5n^2$)
From Unary qsNFAs to 2DFAs

[Geffert Mereghetti&P ’03]

- *M* a fixed qsNFA with *N* states
- An input *w* is accepted iff there is an accepting computation visiting the left endmarker ≤ *N* times
- For *p*, *q* ∈ *Q*, *k* ≥ 1, we define the predicate

\[
\text{reachable}(p, q, k) \equiv \exists \text{computation path on } w \text{ which}
\]

- starts in the state *p* on the left endmarker
- ends in the state *q* on the left endmarker
- visits the left endmarker ≤ *k* more times

- Assuming acceptance on the left endmarker in state *q_f*:

\[
w \in L(M) \iff \text{reachable}(q_0, q_f, N) \text{ is true}\]
From Unary qsNFAs to 2DFAs
[Geoffert Mereghetti&P ’03]

- $M$ a fixed qsNFA with $N$ states
- An input $w$ is accepted iff there is an accepting computation visiting the left endmarker $\leq N$ times
- For $p, q \in Q$, $k \geq 1$, we define the predicate
  \[ \text{reachable}(p, q, k) \equiv \exists \text{computation path on } w \text{ which} \]
  \[ \begin{align*}
  &\text{starts in the state } p \text{ on the left endmarker} \\
  &\text{ends in the state } q \text{ on the left endmarker} \\
  &\text{visits the left endmarker } \leq k \text{ more times}
  \end{align*} \]
- Assuming acceptance on the left endmarker in state $q_f$:
  \[ w \in L(M) \text{ iff } \text{reachable}(q_0, q_f, N) \text{ is true} \]
From Unary qsNFAs to 2DFAs
[Geffert Mereghetti & P’03]

- $M$ a fixed qsNFA with $N$ states
- An input $w$ is accepted iff there is an accepting computation visiting the left endmarker $\leq N$ times
- For $p, q \in Q, k \geq 1$, we define the predicate
  \[\text{reachable}(p, q, k) \equiv \exists \text{ computation path on } w \text{ which}\]
  - starts in the state $p$ on the left endmarker
  - ends in the state $q$ on the left endmarker
  - visits the left endmarker $\leq k$ more times

Assuming acceptance on the left endmarker in state $q_f$:
\[w \in L(M) \iff \text{reachable}(q_0, q_f, N) \text{ is true}\]
From Unary qsNFAs to 2DFAs
[Geffert Mereghetti&P'03]

- $M$ a fixed qsNFA with $N$ states
- An input $w$ is accepted iff there is an accepting computation visiting the left endmarker $\leq N$ times
- For $p, q \in Q$, $k \geq 1$, we define the predicate
  \[
  \text{reachable}(p, q, k) \equiv \exists \text{ computation path on } w \text{ which }
  \begin{align*}
  &\text{starts in the state } p \text{ on the left endmarker} \\
  &\text{ends in the state } q \text{ on the left endmarker} \\
  &\text{visits the left endmarker } \leq k \text{ more times}
  \end{align*}
  \]
- Assuming acceptance on the left endmarker in state $q_f$:
  \[
  w \in L(M) \text{ iff } \text{reachable}(q_0, q_f, N) \text{ is true}
  \]
How to Evaluate \textit{reachable}

\textit{Divide-and-conquer technique}

\textbf{function} \textit{reachable}(p, q, k)
\begin{enumerate}
    \item if \( k = 1 \) then return \textit{reach1}(p, q) \quad \text{///<direct simulation}
    \item else begin
        \begin{enumerate}
            \item for each state \( r \in Q \) do
                \begin{enumerate}
                    \item if \textit{reachable}(p, r, \lfloor k/2 \rfloor) \text{ and } \textit{reachable}(r, q, \lceil k/2 \rceil)
                        then return true \quad \text{///<recursion}
                \end{enumerate}
        \end{enumerate}
    \end{enumerate}
    \item return false
\end{enumerate}
end

This strategy can be implemented by a 2DFA with \( e^{O(\ln 2 N)} \) states in order to compute \textit{reachable}(q_0, q_f, N), i.e., to decide if the input \( w \in L(M) \).
How to Evaluate \textit{reachable}?

\textit{Divide–and–conquer} technique

\begin{verbatim}
function reachable(p, q, k)
if \( k = 1 \) then return reach1(p, q)  //direct simulation
else begin
  for each state \( r \in Q \) do
    if reachable(p, r, ⌊k/2⌋) and reachable(r, q, ⌈k/2⌉) then return true  //recursion
  return false
end
\end{verbatim}

This strategy can be implemented by a 2DFA with \( eO(\ln 2N) \) states in order to compute \( \text{reachable}(q_0, q_f, N) \), i.e., to decide if the input \( w \in L(M) \).
How to Evaluate *reachable*?

*Divide–and–conquer* technique

```
function reachable(p, q, k)
if k = 1 then return reach1(p, q)       //direct simulation
else begin
    for each state r ∈ Q do
        if reachable(p, r, ⌊k/2⌋) and reachable(r, q, ⌈k/2⌉)
            then return true                    //recursion
    return false
end
```

This strategy can be implemented by a 2DFA with $e^{O(\ln 2 N)}$ states in order to compute $reachable(q_0, q_f, N)$, i.e., to decide if the input $w \in L(M)$. 
How to Evaluate \( \textit{reachable} \)?

\begin{center}
\textit{Divide–and–conquer} technique
\end{center}

\textbf{function} \( \textit{reachable}(p, q, k) \)
\textbf{if} \( k = 1 \) \textbf{then} \textbf{return} \( \textit{reach1}(p, q) \) \hspace{1cm} \text{//direct simulation}
\textbf{else} \textbf{begin}
\hspace{1cm} \textbf{for each state} \( r \in Q \) \textbf{do}
\hspace{2cm} \textbf{if} \( \textit{reachable}(p, r, \lfloor k/2 \rfloor) \) \textbf{and} \( \textit{reachable}(r, q, \lceil k/2 \rceil) \)
\hspace{2cm} \textbf{then} \textbf{return} \textit{true} \hspace{1cm} \text{//recursion}
\hspace{1cm} \textbf{return} \textit{false}
\textbf{end}
How to Evaluate \textit{reachable}?

\textit{Divide–and–conquer} technique

\textbf{function} \textit{reachable}(p, q, k)
\begin{algorithmic}
\Function{reachable}{p, q, k}
\If{$k = 1$} \With{\Return \textit{reach1}(p, q) \texttt{//direct simulation}}\EndIf
\Else\Begin\For{\text{each state } r \in Q} \If{\textit{reachable}(p, r, \lfloor k/2 \rfloor) \text{ and } \textit{reachable}(r, q, \lceil k/2 \rceil)} \With{\Return true \texttt{//recursion}} \EndIf\EndFor\Return false\EndElse\EndFunction
\end{algorithmic}

This strategy can be implemented by a 2DFA with $e^{O(\ln^2 N)}$ states in order to compute \textit{reachable}(q_0, q_f, N), i.e., to decide if the input $w \in L(M)$.
From Unary 2NFAs by 2DFAs

\[ A \quad \text{given unary 2NFA} \quad n \text{ states} \]

\[ \downarrow \]

\[ M \quad \text{almost equivalent qsNFA} \quad N \leq 2n + 2 \text{ states} \]

\[ \downarrow \]

\[ B \quad \text{2DFA equivalent to } M \quad e^{O(\ln^2 N)} \text{ states} \]

\[ \downarrow \]

\[ C \quad \text{2DFA equivalent to } A \quad e^{O(\ln^2 n)} \text{ states} \]

Theorem ([Geffert Mereghetti&P '03])

Each unary n-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states
From Unary 2NFAs by 2DFAs

\[ A \quad \text{given unary 2NFA} \quad n \text{ states} \]
\[ \downarrow \]
\[ M \quad \text{almost equivalent qsNFA} \quad N \leq 2n + 2 \text{ states} \]
\[ \downarrow \]
\[ B \quad \text{2DFA equivalent to } M \quad e^{O(\ln^2 N)} \text{ states} \]
\[ \downarrow \]
\[ C \quad \text{2DFA equivalent to } A \quad e^{O(\ln^2 n)} \text{ states} \]

Theorem ([Geffert Mereghetti&P ’03])

Each unary \( n \)-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states
Theorem ([Geffert Mereghetti&P '03])

Each unary $n$-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states.
From Unary 2NFAs by 2DFAs

- **A** given unary 2NFA \( n \) states
  \[ \downarrow \]
- **M** almost equivalent qsNFA \( N \leq 2n + 2 \) states
  \[ \downarrow \]
  **Quasi Sweeping Simulation**
  **Subexponential Deterministic Simulation**
- **B** 2DFA equivalent to **M** \( e^{O(\ln^2 N)} \) states
  \[ \downarrow \]
- **C** 2DFA equivalent to **A** \( e^{O(\ln^2 n)} \) states

**Theorem ([Geffert Mereghetti&P ’03])**

*Each unary \( n \)-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states*
From Unary 2NFAs by 2DFAs

\[ A \quad \text{given unary 2NFA} \quad n \text{ states} \]
\[ \Downarrow \quad \text{Quasi Sweeping Simulation} \]
\[ M \quad \text{almost equivalent qsNFA} \quad N \leq 2n + 2 \text{ states} \]
\[ \Downarrow \quad \text{Subexponential Deterministic Simulation} \]
\[ B \quad \text{2DFA equivalent to } M \quad e^{O(\ln^2 N)} \text{ states} \]
\[ \Downarrow \]

\[ C \quad \text{2DFA equivalent to } A \quad e^{O(\ln^2 n)} \text{ states} \]

Theorem ([Geffert Mereghetti&P ’03])

Each unary \( n \)-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states.
From Unary 2NFAs by 2DFAs

\[ A \]
given unary 2NFA \( n \) states

\[ \Downarrow \]
Quasi Sweeping Simulation

\[ M \]
almost equivalent qsNFA \( N \leq 2n + 2 \) states

\[ \Downarrow \]
Subexponential Deterministic Simulation

\[ B \]
2DFA equivalent to \( M \) \( e^{O(\ln^2 N)} \) states

\[ \Downarrow \]
Preliminary scan to accept/reject inputs of length \( \leq 5n^2 \)
then simulation of \( B \) for longer inputs

\[ C \]
2DFA equivalent to \( A \) \( e^{O(\ln^2 n)} \) states

Theorem ([Geffert Mereghetti&P '03])

Each unary \( n \)-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states
From Unary 2NFAs by 2DFAs

\[ A \quad \text{given unary 2NFA} \quad n \text{ states} \]
\[ \Downarrow \quad \text{Quasi Sweeping Simulation} \]
\[ M \quad \text{almost equivalent qsNFA} \quad N \leq 2n + 2 \text{ states} \]
\[ \Downarrow \quad \text{Subexponential Deterministic Simulation} \]
\[ B \quad \text{2DFA equivalent to } M \quad e^{O(\ln^2 N)} \text{ states} \]
\[ \Downarrow \quad \text{Preliminary scan to accept/reject inputs of length } \leq 5n^2 \]
\[ \text{then simulation of } B \text{ for longer inputs} \]
\[ C \quad \text{2DFA equivalent to } A \quad e^{O(\ln^2 n)} \text{ states} \]

Theorem ([Geffert Mereghetti&P '03])

Each unary \( n \)-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states.
From Unary 2NFAs by 2DFAs

A: given unary 2NFA
\[ n \text{ states} \]

\[ \quad \quad \downarrow \quad \quad \text{Quasi Sweeping Simulation} \]

M: almost equivalent qsNFA
\[ N \leq 2n + 2 \text{ states} \]

\[ \quad \quad \downarrow \quad \quad \text{Subexponential Deterministic Simulation} \]

B: 2DFA equivalent to M
\[ e^{O(ln^2 N)} \text{ states} \]

\[ \quad \quad \downarrow \quad \quad \text{Preliminary scan to accept/reject inputs of length} \leq 5n^2 \]

\[ \quad \quad \downarrow \quad \quad \text{then simulation of B for longer inputs} \]

C: 2DFA equivalent to A
\[ e^{O(ln^2 n)} \text{ states} \]

---

Theorem ([Geffert Mereghetti&P ’03])

Each unary n-state 2NFA can be simulated by a 2DFA with \( e^{O(ln^2 n)} \) states
Quasi Sweeping Simulation: Consequences

Using quasi sweeping simulation of unary 2NFAs several results have been discovered:

(i) Subexponential simulation of unary 2NFAs by 2DFAs
   Each unary $n$-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states [Geffert Mereghetti&P '03]

(ii) Polynomial complementation of unary 2NFAs
     Inductive counting argument for qsNFAs [Geffert Mereghetti&P '07]

(iii) Polynomial simulation of unary 2NFAs by 2DFAs
     under the condition $L = NL$ [Geffert&P '11]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs
     (unconditional) [Geffert&P '11]
Using quasi sweeping simulation of unary 2NFAs several results have been discovered:

(i) Subexponential simulation of unary 2NFAs by 2DFAs
Each unary $n$-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states [Geffert Mereghetti&P '03]

(ii) Polynomial complementation of unary 2NFAs
Inductive counting argument for qsNFAs [Geffert Mereghetti&P '07]

(iii) Polynomial simulation of unary 2NFAs by 2DFAs
under the condition $L = NL$ [Geffert&P '11]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs (unconditional) [Geffert&P '11]
Using quasi sweeping simulation of unary 2NFAs several results have been discovered:

(i) Subexponential simulation of unary 2NFAs by 2DFAs
   Each unary $n$-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states [Geffert Mereghetti&P '03]

(ii) Polynomial complementation of unary 2NFAs
     Inductive counting argument for qsNFAs [Geffert Mereghetti&P '07]

(iii) Polynomial simulation of unary 2NFAs by 2DFAs under the condition $L = NL$ [Geffert&P '11]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs (unconditional) [Geffert&P '11]
Using quasi sweeping simulation of unary 2NFAs several results have been discovered:

(i) Subexponential simulation of unary 2NFAs by 2DFAs
Each unary $n$-state 2NFA can be simulated by a 2DFA with $e^{O(\ln^2 n)}$ states [Geffert Mereghetti&P ’03]

(ii) Polynomial complementation of unary 2NFAs
Inductive counting argument for qsNFAs [Geffert Mereghetti&P ’07]

(iii) Polynomial simulation of unary 2NFAs by 2DFAs
\textit{under the condition} $L = NL$ [Geffert&P ’11]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs (unconditional) [Geffert&P ’11]
Using quasi sweeping simulation of unary 2NFAs several results have been discovered:

(i) Subexponential simulation of unary 2NFAs by 2DFAs
Each unary \( n \)-state 2NFA can be simulated by a 2DFA with \( e^{O(\ln^2 n)} \) states [Geffert Mereghetti&P ’03]

(ii) Polynomial complementation of unary 2NFAs
Inductive counting argument for qsNFAs [Geffert Mereghetti&P ’07]

(iii) Polynomial simulation of unary 2NFAs by 2DFAs
under the condition \( L = NL \) [Geffert&P ’11]

(iv) Polynomial simulation of unary 2NFAs by unambiguous 2NFAs (unconditional) [Geffert&P ’11]
Restricted 2NFAs

Outer Nondeterministic Automata (OFAs) [Guillon Geffert&P ’12]:

- nondeterministic choices

are possible only when the head is visiting the endmarkers
Restricted 2NFAs

*Outer Nondeterministic Automata (OFAs)* [Guillon Geffert&P ’12]:

- *nondeterministic choices*

are *possible only* when the head is visiting the *endmarkers*

Hence:

- No restrictions on the *input alphabet*
- No restrictions on *head reversals*
- *Deterministic transitions* on “real” input symbols
The results we obtained for the unary case can be extended to 2OFA: [Guillon Geffert&P ’12]

(i) Subexponential simulation of 2OFA by 2DFA
(ii) Polynomial complementation of 2OFA
(iii) Polynomial simulation of 2OFA by 2DFA
   under the condition $L = NL$
(iv) Polynomial simulation of 2OFA by unambiguous 2OFA

While in the unary case all the proofs rely on the \textit{quasi sweeping simulation}, for 2OFA we do not have a similar tool!
The results we obtained for the unary case can be extended to 2OFAs: [Guillon Geffert&P ’12]

(i) Subexponential simulation of 2OFAs by 2DFAs
(ii) Polynomial complementation of 2OFAs
(iii) Polynomial simulation of 2OFAs by 2DFAs under the condition $L = NL$
(iv) Polynomial simulation of 2OFAs by unambiguous 2OFAs

While in the unary case all the proofs rely on the quasi sweeping simulation, for 2OFAs we do not have a similar tool!
The results we obtained for the unary case can be extended to 2OFAs: [Guillon Geffert&P ’12]

(i) Subexponential simulation of 2OFAs by 2DFAs
(ii) Polynomial complementation of 2OFAs
(iii) Polynomial simulation of 2OFAs by 2DFAs under the condition $L = NL$
(iv) Polynomial simulation of 2OFAs by unambiguous 2OFAs

While in the unary case all the proofs rely on the quasi sweeping simulation, for 2OFAs we do not have a similar tool!
Procedure \textit{reach}(p, q)

- Checks the existence of a computation segment
  - from the left endmarker in the state \(p\)
  - to the left endmarker in the state \(q\)
  - not visiting the left endmarker in between

- Critical point: infinite loops
  - Modification of a technique for the complementation
    of 2DFAs [Geffert Mereghetti&P’07],
    which refines a construction for space bounded TM [Sipser’80]
Procedure \textit{reach}(p, q)

- Checks the existence of a computation segment
  - from the left endmarker in the state \( p \)
  - to the left endmarker in the state \( q \)
  - not visiting the left endmarker in between

- Critical point: infinite loops
  - Modification of a technique for the complementation of 2DFAs [Geffert Mereghetti P’07], which refines a construction for space bounded TM [Sipser’80]
Outer Nondeterministic Automata (OFAs)

Procedure $\text{reach}(p, q)$

- Checks the existence of a computation segment
  - from the left endmarker in the state $p$
  - to the left endmarker in the state $q$
  - not visiting the left endmarker in between

- Critical point: infinite loops
  - Modification of a technique for the complementation of 2DFAs [Geffert Mereghetti&P ’07],
    which refines a construction for space bounded TM [Sipser ’80]
Outer Nondeterministic Automata (OFAs)

Procedure \textit{reach}(p, q)

- Checks the existence of a computation segment
  - from the left endmarker in the state $p$
  - to the left endmarker in the state $q$
  - not visiting the left endmarker in between

- Critical point: infinite loops
  - Modification of a technique for the complementation of 2DFAs [Geffert Mereghetti&P ’07], which refines a construction for space bounded TM [Sipser ’80]

Loops involving endmarkers are also possible

- They can be avoided by observing that for each accepting computation visiting one endmarkers more than $|Q|$ times there exists a shorter accepting computation
Sakoda&Sipser Question: Current Knowledge

- **Upper bounds**

<table>
<thead>
<tr>
<th></th>
<th>1NFA→2DFA</th>
<th>2NFA→2DFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>unary case</td>
<td>$O(n^2)$</td>
<td>$e^{O(\ln^2 n)}$</td>
</tr>
<tr>
<td>OFAs</td>
<td>optimal</td>
<td>exponential</td>
</tr>
<tr>
<td>general case</td>
<td>exponential</td>
<td>exponential</td>
</tr>
</tbody>
</table>

Unary case [Chrobak ’86, Geffert Mereghetti&P ’03]
OFAs [Guillon Geffert&P ’12]

- **Lower Bounds**

In all the cases, the best known lower bound is $\Omega(n^2)$ [Chrobak ’86]
Final Remarks

Speaking about...

...Finite automata

usually we mean

One-way finite automata

Why this difference?

In both cases:

▶ Computability aspects
▶ Complexity aspects

Minicomplexity

▶ Complexity theory of two-way finite automata

[Kapoutsis, DCFS 2012]
### Final Remarks

Speaking about...

<table>
<thead>
<tr>
<th>...Finite automata</th>
<th>...Turing machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>usually we mean</td>
<td>usually we mean</td>
</tr>
<tr>
<td><em>One-way finite automata</em></td>
<td><em>Two-way Turing machines</em></td>
</tr>
</tbody>
</table>

Why this difference?

- Computability aspects
- Complexity aspects

---

\[\text{Kapoutsis, DCFS 2012}\]
Final Remarks

Speaking about...

...Finite automata
usually we mean
One-way finite automata

...Turing machines
usually we mean
Two-way Turing machines

Why this difference?
Final Remarks

Speaking about...

...Finite automata

usually we mean

One-way finite automata

...Turing machines

usually we mean

Two-way Turing machines

Why this difference?

In both cases:

- Computability aspects
- Complexity aspects
Final Remarks

Speaking about...

Finite automata
usually we mean

One-way finite automata

Turing machines
usually we mean

Two-way Turing machines

Why this difference?

In both cases:

- Computability aspects
- Complexity aspects

Minicomplexity

- Complexity theory of two-way finite automata

[Kapoutsis, DCFS 2012]
Final Remarks

- The question of Sakoda and Sipser is very challenging
- In the investigation of restricted versions many interesting and not artificial models have been considered
- The results obtained under restrictions, even if not solving the full problem, are not trivial and, in many cases, very deep
- Connections with space and structural complexity
  - questions
  - techniques
- Connections with number theory (unary automata)
Final Remarks

- The question of Sakoda and Sipser is very challenging.
- In the investigation of restricted versions many interesting and not artificial models have been considered.
- The results obtained under restrictions, even if not solving the full problem, are not trivial and, in many cases, very deep.
- Connections with space and structural complexity questions.
- Connections with number theory (unary automata).
Final Remarks

- The question of Sakoda and Sipser is very challenging.
- In the investigation of restricted versions many interesting and not artificial models have been considered.
- The results obtained under restrictions, even if not solving the full problem, are not trivial and, in many cases, very deep.
- Connections with space and structural complexity questions.
- Connections with number theory (unary automata).
Final Remarks

- The question of Sakoda and Sipser is very challenging
- In the investigation of restricted versions many interesting and not artificial models have been considered
- The results obtained under restrictions, even if not solving the full problem, are not trivial and, in many cases, very deep
- Connections with space and structural complexity
  - questions
  - techniques
- Connections with number theory (unary automata)
Final Remarks

- The question of Sakoda and Sipser is very challenging
- In the investigation of restricted versions many interesting and not artificial models have been considered
- The results obtained under restrictions, even if not solving the full problem, are not trivial and, in many cases, very deep
- Connections with space and structural complexity
  - questions
  - techniques
- Connections with number theory (unary automata)
Thank you for your attention!