Limited Automata: Power and Complexity

Giovanni Pighizzini

Dipartimento di Informatica
Università degli Studi di Milano, Italy

ICTCS 2019 – Como, Italy
September 11, 2019

UNIVERSITÀ DEGLI STUDI DI MILANO

Introduction

We are interested in...

- Computational models and their computational power

We are interested in...

- Computational models and their computational power, e.g., classical models
  - Finite automata
  - Pushdown automata
  - Turing machines

We are interested in...

- Computational models and their computational power
- Computational models operating with restricted resources

We are interested in...

- Computational models and their computational power
- Computational models operating with restricted resources, e.g.,
  - Turing machines using *linear space* characterize *context-sensitive languages* [Kuroda '64]
  - Single-tape Turing machines working in *linear time* characterize *regular languages*, i.e., are equivalent to finite automata [Hennie '65]

We are interested in...

- Computational models and their computational power
- Computational models operating with restricted resources, e.g.,
  - Turing machines using *linear space* characterize *context-sensitive languages* [Kuroda '64]
  - Single-tape Turing machines working in *linear time* characterize *regular languages*, i.e., are equivalent to finite automata [Hennie '65]

We are interested in...

- Computational models and their computational power
- Computational models operating with restricted resources
- Descriptional complexity

We are interested in...

- Computational models and their computational power
- Computational models operating with restricted resources
- Descriptional complexity
  - Investigation of computational models with respect to the sizes of their descriptions (roughly, number of symbols used to write down the description)

Formal language (or computability) point of view:

- The class of languages recognized by NFAs coincides with the class of languages recognized by DFAs

Formal language (or computability) point of view:

- The class of languages recognized by NFAs coincides with the class of languages recognized by DFAs

Formal language (or computability) point of view:

- The class of languages recognized by NFAs coincides with the class of languages recognized by DFAs

Descriptional complexity point of view:

- Each n-state NFA can be simulated by a DFA *with* 2^n *states*
  [Rabin&Scott '59]
- For each integer n there exists a language L_n s.t.:
  - L_n is accepted by an n-state NFA
  - the minimum DFA for L_n requires 2^n states

  [Meyer&Fischer '71]
- Hence:

  The exact cost, in terms of states, of the simulation of NFAs by DFAs is 2^n

Formal language (or computability) point of view:

- The class of languages recognized by NFAs coincides with the class of languages recognized by DFAs

Descriptional complexity point of view:

- Each n-state NFA can be simulated by a DFA *with* 2^n *states* [Rabin&Scott '59]
- For each integer n there exists a language L_n s.t.:
  - L_n is accepted by an n-state NFA
  - the minimum DFA for L_n requires 2^n states

  [Meyer&Fischer '71]
- Hence:

  The exact cost, in terms of states, of the simulation of NFAs by DFAs is 2^n

Formal language (or computability) point of view:

- The class of languages recognized by NFAs coincides with the class of languages recognized by DFAs

Descriptional complexity point of view:

- Each n-state NFA can be simulated by a DFA *with* 2^n *states* [Rabin&Scott '59]
- For each integer n there exists a language L_n s.t.:
  - L_n is accepted by an n-state NFA
  - the minimum DFA for L_n requires 2^n states

  [Meyer&Fischer '71]
- Hence:

  The exact cost, in terms of states, of the simulation of NFAs by DFAs is 2^n

Formal language (or computability) point of view:

- The class of languages recognized by NFAs coincides with the class of languages recognized by DFAs

Descriptional complexity point of view:

- Each n-state NFA can be simulated by a DFA *with* 2^n *states*
  [Rabin&Scott '59]
- For each integer n there exists a language L_n s.t.:
  - L_n is accepted by an n-state NFA
  - the minimum DFA for L_n requires 2^n states

  [Meyer&Fischer '71]
- Hence:

  The exact cost, in terms of states, of the simulation
  of NFAs by DFAs is 2^n

Descriptional Complexity

Given

- $\mathcal{C}$ a class of languages
- $\mathcal{S}$ a formal system (e.g., class of devices, class of grammars,...) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Descriptional complexity compares different descriptions of a same class of languages:

- $\mathcal{S}'$ another formal system able to represent all the languages in $\mathcal{C}$:

Question

Find the relationships between the sizes of the representations in the system $\mathcal{S}$ and in the system $\mathcal{S}'$ of the languages of $\mathcal{C}$

Descriptional Complexity

Given

- $\mathcal{C}$ a class of languages
- $\mathcal{S}$ a formal system (e.g., class of devices, class of grammars,...) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Descriptional complexity compares different descriptions of a same class of languages:

- $\mathcal{S}'$ another formal system able to represent all the languages in $\mathcal{C}$:

Question

Find the relationships between the sizes of the representations in the system $\mathcal{S}$ and in the system $\mathcal{S}'$ of the languages of $\mathcal{C}$

Descriptional Complexity

Given

- $\mathcal{C}$ a class of languages
- $\mathcal{S}$ a formal system (e.g., class of devices, class of grammars,...) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Descriptional complexity compares different descriptions of a same class of languages:

- $\mathcal{S}'$ another formal system able to represent all the languages in $\mathcal{C}$:

Question

Find the relationships between the sizes of the representations in the system $\mathcal{S}$ and in the system $\mathcal{S}'$ of the languages of $\mathcal{C}$

Descriptional Complexity

Given

- $\mathcal{C}$ a class of languages
- $\mathcal{S}$ a formal system (e.g., class of devices, class of grammars,...) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Descriptional complexity compares different descriptions of a same class of languages:

- $\mathcal{S}'$ another formal system able to represent all the languages in $\mathcal{C}$:

Question

Find the relationships between the sizes of the representations in the system $\mathcal{S}$ and in the system $\mathcal{S}'$ of the languages of $\mathcal{C}$

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a "small" 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a “small” 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a “small” 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a "small" 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA

- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)

- How much it costs (in terms of states)?

- Can we obtain a "small" 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a "small" 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a "small" 2DFA from a 1NFA or a 2NFA?

A Long-standing Open Problem in Descriptional Complexity

The Question of Sakoda and Sipser (1978)

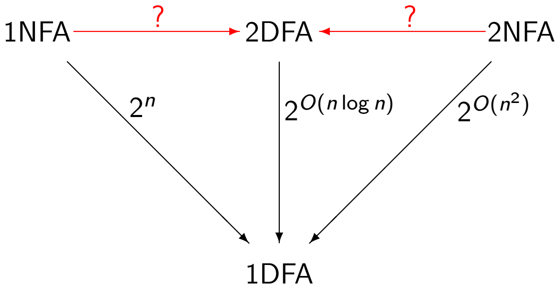
- Two-way finite automata
  - input head can be moved to the left or to the right
  - computational power does not increase
  - exponential simulation by one-way DFA
- Can we use two-way motion to remove nondeterminism from finite automata?
  - YES (same class of languages)
- How much it costs (in terms of states)?
- Can we obtain a "small" 2DFA from a 1NFA or a 2NFA?

The Question of Sakoda and Sipser

$$\begin{tikzcd} \text{1NFA} \arrow[dr, "2^n"] & \text{2DFA} \arrow[d, "2^{O(n \log n)}"] & \text{2NFA} \arrow[dl, "2^{O(n^2)}"] \\ & \text{1DFA} & \end{tikzcd}$$

[Rabin&Scott '59, Shepherdson '59, Meyer&Fischer '71, . . .]

The Question of Sakoda and Sipser



Problem ([Sakoda&Sipser '78])

Do there exist polynomial simulations of

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

The Question of Sakoda and Sipser

$$\begin{tikzcd}
\text{1NFA} \arrow[r, "?"] \arrow[dr, "2^n"] & \text{2DFA} \arrow[d, "2^{O(n \log n)}"] & \text{2NFA} \arrow[l, "?"'] \arrow[dl, "2^{O(n^2)}"'] \\
& \text{1DFA} &
\end{tikzcd}$$

Problem ([Sakoda&Sipser '78])

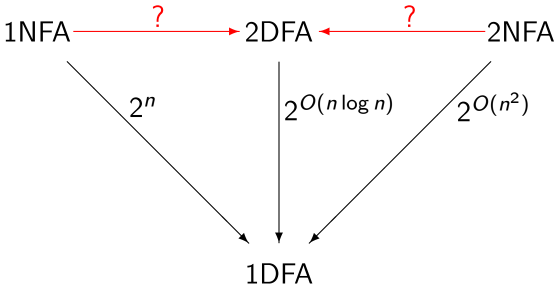
Do there exist polynomial simulations of

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

Conjecture

These simulations are not polynomial

The Question of Sakoda and Sipser



- 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]

The Question of Sakoda and Sipser

- Open since 1978
- It seems to be very difficult in the general case
- Results and exponential separations for restricted versions
- Connections with fundamental structural complexity questions as P vs NP and L vs NL

The Question of Sakoda and Sipser

- Open since 1978
- It seems to be very difficult in the general case
- Results and exponential separations for restricted versions
- Connections with fundamental structural complexity questions as P vs NP and L vs NL

The Question of Sakoda and Sipser

- Open since 1978
- It seems to be very difficult in the general case
- Results and exponential separations for restricted versions
- Connections with fundamental structural complexity questions as P vs NP and L vs NL

The Question of Sakoda and Sipser

- Open since 1978
- It seems to be very difficult in the general case
- Results and exponential separations for restricted versions
- Connections with fundamental structural complexity questions as P vs NP and L vs NL

Introduction to Limited Automata

Limited automata

- Model proposed by Thomas N. Hibbard in 1967
  (scan limited automata)
- One-tape Turing machines with rewriting restrictions
- Variants characterizing regular, context-free, deterministic context-free languages

Limited automata

- Model proposed by Thomas N. Hibbard in 1967
  (scan limited automata)
- One-tape Turing machines with rewriting restrictions
- Variants characterizing regular, context-free, deterministic context-free languages

Limited automata

- Model proposed by Thomas N. Hibbard in 1967
  (scan limited automata)
- One-tape Turing machines with rewriting restrictions
- Variants characterizing regular, context-free, deterministic context-free languages

A Classical Example: Balanced Brackets

$$(\quad (\quad) \quad (\quad (\quad) \quad) \quad)$$

How to recognize if a sequence of brackets is correctly balanced?

A Classical Example: Balanced Brackets

(( ) (( )))

How to recognize if a sequence of brackets is correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

A Classical Example: Balanced Brackets

$$(\quad (\quad) \quad (\quad (\quad) \quad) \quad)$$

How to recognize if a sequence of brackets is correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

A Classical Example: Balanced Brackets

$$(_1 \quad (_2 \quad)_2 \quad (_2 \quad (_3 \quad)_3 \quad)_2 \quad)_1$$

How to recognize if a sequence of brackets is correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

A Classical Example: Balanced Brackets

(( ) (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(( ) (( )))

$\longrightarrow$ $\uparrow$

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(( ) (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(( ) (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

$$\begin{array}{ccccccccc} (& (& x & (& (&) &) &) \end{array}$$

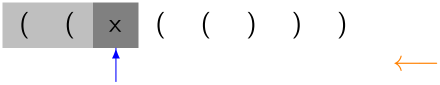
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(	(	x	(	(	)	)	)

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(	x	x	(	(	)	)	)

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (( )))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (( )))

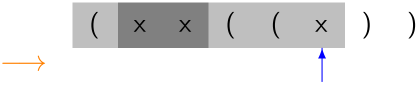
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x ((x))

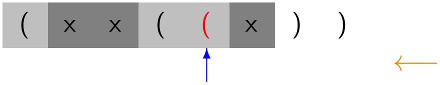
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



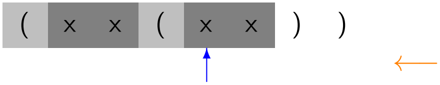
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



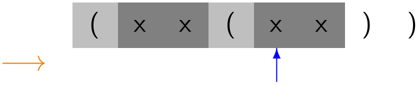
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (x x))

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x (x x))

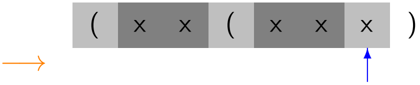
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



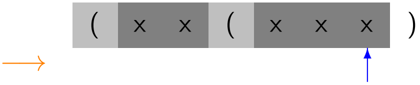
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



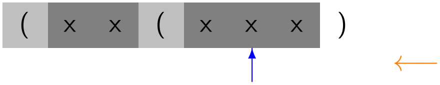
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



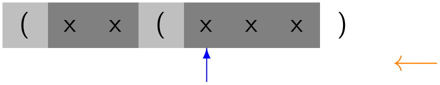
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

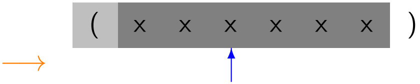
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



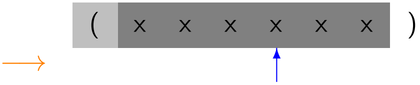
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



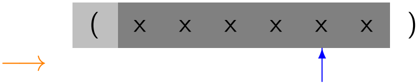
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

(x x x x x x)

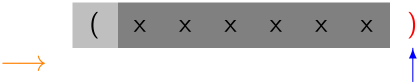
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



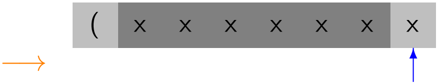
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

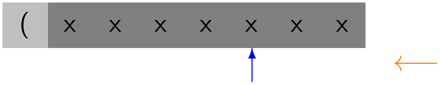
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



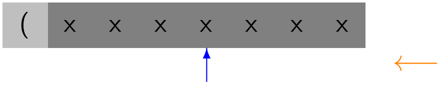
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

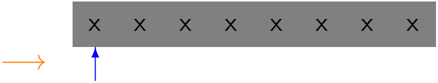
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



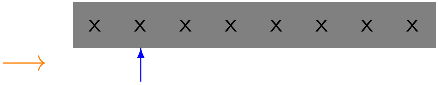
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



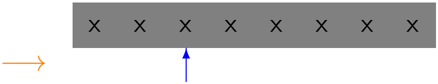
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



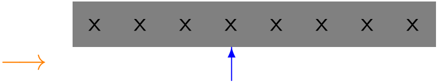
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



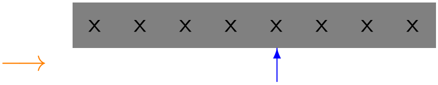
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



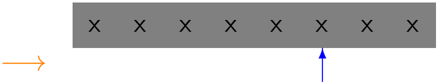
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



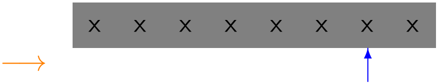
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



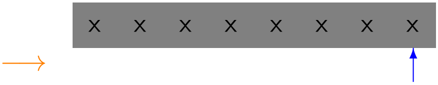
How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets



How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

A Classical Example: Balanced Brackets

x	x	x	x	x	x	x	x

How to recognize if a sequence of brackets its correctly balanced?

- *For each opening bracket*
  locate its corresponding closing bracket

Use counters!

- *For each closing bracket*
  locate its corresponding opening bracket

Limited automata!

Limited Automata [Hibbard '67]

One-tape Turing machines with restricted rewritings

Definition

Fixed an integer $d \geq 1$, a *d-limited automaton* is

- a one-tape Turing machine
- which is allowed to overwrite the content of each tape cell *only in the first d visits*

Limited Automata [Hibbard '67]

One-tape Turing machines with restricted rewritings

Definition

Fixed an integer $d \geq 1$, a *d-limited automaton* is

- a one-tape Turing machine
- which is allowed to overwrite the content of each tape cell *only in the first d visits*

Technical details:

- Input surrounded by two end-markers
- End-markers are never overwritten
- The head cannot exceed the end-markers

Limited Automata [Hibbard '67]

One-tape Turing machines with restricted rewritings

Definition

Fixed an integer $d \geq 1$, a *d-limited automaton* is

- a one-tape Turing machine
- which is allowed to overwrite the content of each tape cell *only in the first d visits*

Computational power

- For each $d \geq 2$, d-limited automata characterize context-free languages [Hibbard '67]

Limited Automata [Hibbard '67]

One-tape Turing machines with restricted rewritings

Definition

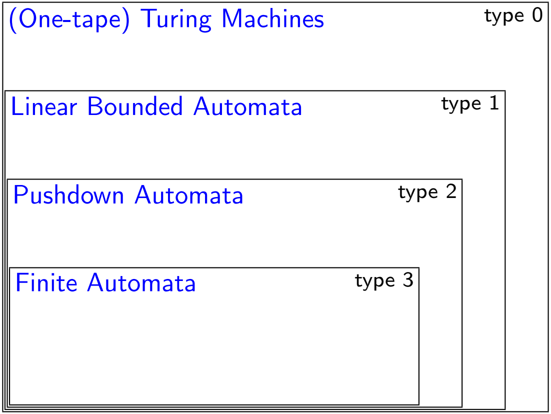
Fixed an integer $d \geq 1$, a *d-limited automaton* is

- a one-tape Turing machine
- which is allowed to overwrite the content of each tape cell *only in the first d visits*

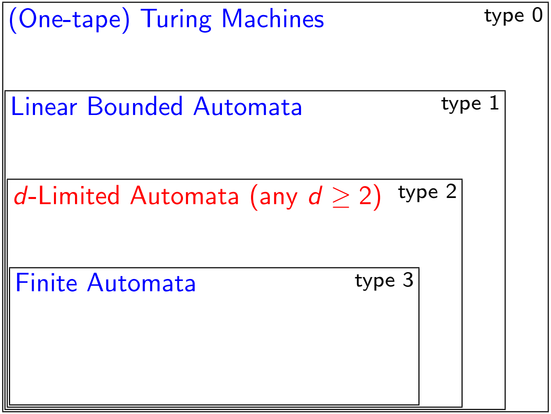
Computational power

- For each $d \geq 2$, d-limited automata characterize context-free languages [Hibbard '67]
- 1-limited automata characterize regular languages [Wagner&Wechsung '86]

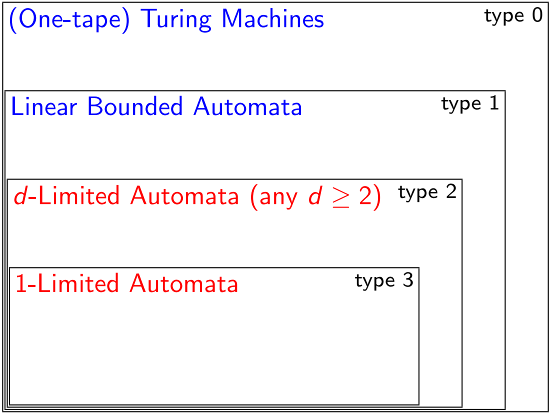
The Chomsky Hierarchy



The Chomsky Hierarchy



The Chomsky Hierarchy



Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

Theorem ([Chomsky&Schützenberger '63])

Each CFL $L \subseteq \Sigma^*$ *can be expressed as* $L = h(D_k \cap R)$ *where:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language (i.e., balanced brackets) over* $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$
- $R \subseteq \Omega_k^*$ *is a regular language*
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism*

Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

Theorem ([Chomsky&Schützenberger '63])

Each CFL $L \subseteq \Sigma^*$ *can be expressed as* $L = h(D_k \cap R)$ *where:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language (i.e., balanced brackets) over* $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$
- $R \subseteq \Omega_k^*$ *is a regular language*
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism*

Transducer T *for* h^{-1}

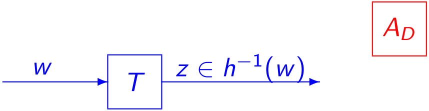
$$\xrightarrow{\;w\;} \boxed{T} \xrightarrow{\;z \in h^{-1}(w)\;}$$

Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

Theorem ([Chomsky&Schützenberger '63])

Each CFL $L \subseteq \Sigma^*$ *can be expressed as* $L = h(D_k \cap R)$ *where:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language (i.e., balanced brackets) over* $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ — 2-LA A_D
- $R \subseteq \Omega_k^*$ *is a regular language*
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism* — *Transducer* T *for* h^{-1}

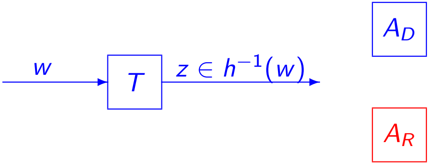


Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

Theorem ([Chomsky&Schützenberger '63])

Each CFL $L \subseteq \Sigma^*$ *can be expressed as* $L = h(D_k \cap R)$ *where:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language (i.e., balanced brackets) over* $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ — 2-LA A_D
- $R \subseteq \Omega_k^*$ *is a regular language* — *Finite automaton* A_R
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism* — *Transducer* T *for* h^{-1}

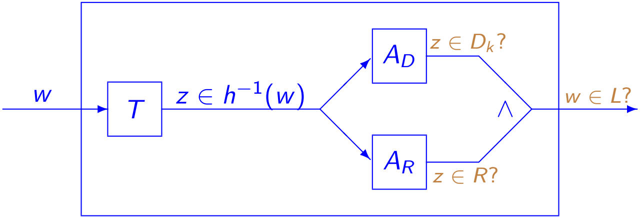


Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

Theorem ([Chomsky&Schützenberger '63])

Each CFL $L \subseteq \Sigma^*$ *can be expressed as* $L = h(D_k \cap R)$ *where:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language (i.e., balanced brackets) over* $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ — 2-LA A_D
- $R \subseteq \Omega_k^*$ *is a regular language* — *Finite automaton* A_R
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism* — *Transducer* T *for* h^{-1}

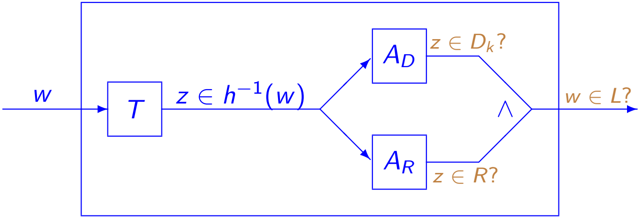


Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

Theorem ([Chomsky&Schützenberger '63])

Each CFL $L \subseteq \Sigma^*$ *can be expressed as* $L = h(D_k \cap R)$ *where:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language (i.e., balanced brackets) over* $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ — 2-LA A_D
- $R \subseteq \Omega_k^*$ *is a regular language* — *Finite automaton* A_R
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism* — *Transducer* T *for* h^{-1}



Suitably simulating this combination of T, A_D and A_R we obtain a 2-LA

Determinism vs Nondeterminism

- Simulations in [Hibbard '67]:
  Determinism is preserved by the simulation PDAs by 2-LAs,
  but not by the converse simulation
- A different simulation of 2-LAs by PDAs,
  which *preserves determinism*, is given in [P.&Pisoni '15]

Deterministic 2-Limited Automata $\equiv$ DCFLs

Determinism vs Nondeterminism

- Simulations in [Hibbard '67]:
  Determinism is preserved by the simulation PDAs by 2-LAs,
  but not by the converse simulation

- A different simulation of 2-LAs by PDAs,
  which *preserves determinism*, is given in [P.&Pisoni '15]

Deterministic 2-Limited Automata $\equiv$ DCFLs

Determinism vs Nondeterminism

- Simulations in [Hibbard '67]:
  Determinism is preserved by the simulation PDAs by 2-LAs,
  but not by the converse simulation

- A different simulation of 2-LAs by PDAs,
  which *preserves determinism*, is given in [P.&Pisoni '15]

Deterministic 2-Limited Automata $\equiv$ DCFLs

Determinism vs Nondeterminism

What about *deterministic* d-Limited Automata, $d > 2$?

- $L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$
  is accepted by a *deterministic* 3-LA, but is not a DCFL

- Infinite hierarchy [Hibbard '67]

  For each $d \geq 2$ there is a language which is accepted by a deterministic d-limited automaton and that cannot be accepted by any deterministic $(d-1)$-limited automaton

Determinism vs Nondeterminism

What about *deterministic* d-Limited Automata, $d > 2$?

- $L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$
  is accepted by a *deterministic* 3-LA, but is not a DCFL

- Infinite hierarchy [Hibbard '67]

  For each $d \geq 2$ there is a language which is accepted by a deterministic d-limited automaton and that cannot be accepted by any deterministic $(d-1)$-limited automaton

Determinism vs Nondeterminism

What about *deterministic* d-Limited Automata, $d > 2$?

- $L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$
  is accepted by a *deterministic* 3-LA, but is not a DCFL

- Infinite hierarchy [Hibbard '67]

  For each $d \geq 2$ there is a language which is accepted by a deterministic d-limited automaton and that cannot be accepted by any deterministic $(d-1)$-limited automaton

Determinism vs Nondeterminism

What about *deterministic* d-Limited Automata, $d > 2$?

- $L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$
  is accepted by a *deterministic* 3-LA, but is not a DCFL
- Infinite hierarchy [Hibbard '67]

  For each $d \geq 2$ there is a language which is accepted by a deterministic d-limited automaton and that cannot be accepted by any deterministic $(d-1)$-limited automaton

Determinism vs Nondeterminism

What about *deterministic* d-Limited Automata, $d > 2$?

- $L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$
  is accepted by a *deterministic* 3-LA, but is not a DCFL
- Infinite hierarchy [Hibbard '67]

  For each $d \geq 2$ there is a language which is accepted by a deterministic d-limited automaton and that cannot be accepted by any deterministic $(d-1)$-limited automaton

Claim [Hibbard '67]

For any $d > 0$, the set of Palindromes cannot be accepted by any *deterministic* d-LA

Hence $\bigcup_{d>0}$ det-d-LA $\subset$ CFL properly

Determinism vs Nondeterminism

What about *deterministic* d-Limited Automata, $d > 2$?

- $L = \{a^n b^n c \mid n \geq 0\} \cup \{a^n b^{2n} d \mid n \geq 0\}$
  is accepted by a *deterministic* 3-LA, but is not a DCFL

- Infinite hierarchy [Hibbard '67]

  For each $d \geq 2$ there is a language which is accepted by a deterministic d-limited automaton and that cannot be accepted by any deterministic $(d-1)$-limited automaton

Claim [Hibbard '67]

For any $d > 0$, the set of Palindromes cannot be accepted by any *deterministic* d-LA

Hence $\bigcup_{d>0}$ det-d-LA $\subset$ CFL properly

Open Problem

Any proof?

Descriptional Complexity of Limited Automata

The Language B_n $(n > 0)$

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ \ k > 0,$$

The Language B_n $(n > 0)$

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ \ k > 0, \\ \text{and } x_j = x,\ \text{ for some } 1 \leq j \leq k\,\}$$

The Language B_n $(n > 0)$

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ k > 0, \\ \text{and } x_j = x,\ \text{for some } 1 \le j \le k\,\}$$

Example $(n = 3)$:

0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 1 0

The Language B_n $(n > 0)$

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ \ k > 0, \\ \text{and } x_j = x,\ \text{ for some } 1 \le j \le k\,\}$$

Example $(n = 3)$:

$$0\ 0\ 1 | 0\ 1\ 0 | \color{red}{1\ 1\ 0} | 0\ 1\ 0 | 1\ 0\ 0 | 1\ 1\ 1 | \color{blue}{1\ 1\ 0}$$

A Deterministic 2-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 $\hat{1}$ $\hat{1}$ $\hat{0}$ $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ \hat{1}\ \hat{1}\ \hat{0}\ \triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 $\hat{1}$ $\hat{1}$ $\hat{0}$ $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright\ 0\,0\,1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,1\,0\,0\,\mathtt{x}\,\mathtt{x}\,\mathtt{x}\,\hat{1}\,\hat{1}\,\hat{0} \triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right), symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright\ 0\,0\,1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,\mathtt{x}\,\mathtt{x}\,\mathtt{x}\,\mathtt{x}\,\mathtt{x}\,\mathtt{x}\,\hat{1}\,\hat{1}\,\hat{0}\ \triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$$\triangleright\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \mathtt{x}\ \hat{1}\ \hat{1}\ \hat{0}\ \triangleleft \qquad (n = 3)$$

1. Scan all the tape from left to right and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right), symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 x x x x x x x x x x x x $\hat{1}$ $\hat{1}$ $\hat{0}$ $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

A Deterministic 2-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 x x x x x x x x x x x x x $\hat{1}$ $\hat{1}$ $\hat{0}$ $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and check if the input length is a multiple of n
2. Move to the left and mark the rightmost block of n symbols
3. Compare the other blocks of length n (from the right),
   symbol by symbol, with the last block
4. When the matching block is found, accept

Complexity:

- $O(n)$ states
- Fixed working alphabet

$\Rightarrow$ det-2-LA of size $O(n)$

A Nondeterministic 1-Limited Automaton for B_n

$\triangleright$ 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 $\triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and mark two nondeterministically chosen cells
2. Check that:
   - the input length is a multiple of n,
   - the last marked cell is the leftmost one of the last block, and
   - the other marked cell is the leftmost one of another block
3. Compare symbol by symbol the two blocks that start from the marked cells
4. Accept if the two blocks are equal

A Nondeterministic 1-Limited Automaton for B_n

$\rhd$ 0 0 1 0 1 0 $\hat{1}$ 1 0 0 1 0 1 0 0 1 1 1 $\hat{1}$ 1 0 $\lhd$ $(n = 3)$

1. Scan all the tape from left to right
   and mark two nondeterministically chosen cells
2. Check that:
   - the input length is a multiple of n,
   - the last marked cell is the leftmost one of the last block, and
   - the other marked cell is the leftmost one of another block
3. Compare symbol by symbol the two blocks that start from the marked cells
4. Accept if the two blocks are equal

A Nondeterministic 1-Limited Automaton for B_n

$\triangleright\ 0\ 0\ 1\ 0\ 1\ 0\ \hat{1}\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ \hat{1}\ 1\ 0\ \triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and mark two nondeterministically chosen cells
2. Check that:
   - the input length is a multiple of n,
   - the last marked cell is the leftmost one of the last block, and
   - the other marked cell is the leftmost one of another block
3. Compare symbol by symbol the two blocks that start from the marked cells
4. Accept if the two blocks are equal

A Nondeterministic 1-Limited Automaton for B_n

$$\triangleright\ 0\ 0\ 1\ 0\ 1\ 0\ \hat{1}\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ \hat{1}\ 1\ 0\ \triangleleft \qquad (n=3)$$

1. Scan all the tape from left to right
   and mark two nondeterministically chosen cells
2. Check that:
   - the input length is a multiple of n,
   - the last marked cell is the leftmost one of the last block, and
   - the other marked cell is the leftmost one of another block
3. Compare symbol by symbol the two blocks that start from the marked cells
4. Accept if the two blocks are equal

A Nondeterministic 1-Limited Automaton for B_n

$\triangleright\ 0\ 0\ 1\ 0\ 1\ 0\ \hat{1}\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ \hat{1}\ 1\ 0\ \triangleleft$ $(n = 3)$

1. Scan all the tape from left to right
   and mark two nondeterministically chosen cells
2. Check that:
   - the input length is a multiple of n,
   - the last marked cell is the leftmost one of the last block, and
   - the other marked cell is the leftmost one of another block
3. Compare symbol by symbol the two blocks that start from the marked cells
4. Accept if the two blocks are equal

Complexity:

- $O(n)$ states
- Fixed working alphabet

$\Rightarrow$ 1-LA of size $O(n)$

Lower bounds for B_n

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ k > 0, \text{ and } x_j = x,\ \text{for some } 1 \leq j \leq k\,\}$$

Lower bounds for B_n

$$B_n = \{x_1 \, x_2 \cdots x_k \, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ k > 0, \text{ and } x_j = x,\ \text{for some } 1 \leq j \leq k\,\}$$

Finite automata

Each 1DFA accepting B_n requires a number of states at least *double exponential* in n

Proof: standard distinguishability arguments

1-LAs $\rightarrow$ 1DFAs

At least double exponential gap!

Lower bounds for B_n

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ \ k > 0,\ \text{and } x_j = x,\ \text{for some } 1 \le j \le k\,\}$$

Finite automata

Each 1DFA accepting B_n requires a number of states at least *double exponential* in n

Proof: standard distinguishability arguments

1-LAs $\to$ 1DFAs

At least double exponential gap!

Lower bounds for B_n

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \mid |x_1| = \cdots = |x_k| = |x| = n,\ k > 0, \text{ and } x_j = x, \text{ for some } 1 \le j \le k\}$$

Finite automata

Each 1DFA accepting B_n requires a number of states at least *double exponential* in n

Proof: standard distinguishability arguments

1-LAs $\rightarrow$ 1DFAs

At least double exponential gap!

CFGs and PDAs

Each CFG generating B_n (PDA recognizing B_n) has size at least *exponential* in n

Proof: "interchange" lemma for CFLs

det-2-LAs $\rightarrow$ PDAs

At least exponential gap!

Lower bounds for B_n

$$B_n = \{x_1\, x_2 \cdots x_k\, x \in \{0,1\}^* \quad | \quad |x_1| = \cdots = |x_k| = |x| = n,\ \ k > 0, \\ \text{and } x_j = x,\ \text{ for some } 1 \leq j \leq k\,\}$$

Finite automata

Each 1DFA accepting B_n requires a number of states at least *double exponential* in n

Proof: standard distinguishability arguments

1-LAs $\rightarrow$ 1DFAs

At least double exponential gap!

CFGs and PDAs

Each CFG generating B_n (PDA recognizing B_n) has size at least *exponential* in n

Proof: "interchange" lemma for CFLs

det-2-LAs $\rightarrow$ PDAs

At least exponential gap!

Size Costs of Simulations

d-LAs versus PDAs (or CFGs), $d \geq 2$

- 2-LAs $\rightarrow$ PDAs [P.&Pisoni '15]
  d-LAs $\rightarrow$ PDAs, $d > 2$ [Kutrib&P.&Wendlandt '18]
  exponential
- det-2-LAs $\rightarrow$ DPDAs [P.&Pisoni '15]
  double exponential upper bound (optimal?)
  exponential if the input for the simulating DPDA is end-marked
- PDAs $\rightarrow$ 2-LAs,
  DPDAs $\rightarrow$ det-2-LAs [P.&Pisoni '15]
  polynomial

Size Costs of Simulations

d-LAs versus PDAs (or CFGs), $d \geq 2$

- 2-LAs $\rightarrow$ PDAs [P.&Pisoni '15]
  d-LAs $\rightarrow$ PDAs, $d > 2$ [Kutrib&P.&Wendlandt '18]
  exponential
- det-2-LAs $\rightarrow$ DPDAs [P.&Pisoni '15]
  double exponential upper bound (optimal?)
  exponential if the input for the simulating DPDA is end-marked
- PDAs $\rightarrow$ 2-LAs,
  DPDAs $\rightarrow$ det-2-LAs [P.&Pisoni '15]
  polynomial

Size Costs of Simulations

d-LAs versus PDAs (or CFGs), $d \geq 2$

- 2-LAs $\rightarrow$ PDAs [P.&Pisoni '15]
  d-LAs $\rightarrow$ PDAs, $d > 2$ [Kutrib&P.&Wendlandt '18]
  exponential
- det-2-LAs $\rightarrow$ DPDAs [P.&Pisoni '15]
  double exponential upper bound (optimal?)
  exponential if the input for the simulating DPDA is end-marked
- PDAs $\rightarrow$ 2-LAs,
  DPDAs $\rightarrow$ det-2-LAs [P.&Pisoni '15]
  polynomial

Size Costs of Simulations

1-LAs versus Finite Automata [Wagner&Wechsung '86, P.&Pisoni '14]

- 1-LAs $\rightarrow$ 1NFA
  exponential
- 1-LAs $\rightarrow$ 1DFA
  double exponential
- det-1-LAs $\rightarrow$ 1DFA
  exponential

Size Costs of Simulations

1-LAs versus Finite Automata [Wagner&Wechsung '86, P.&Pisoni '14]

- 1-LAs $\to$ 1NFA
  exponential
- 1-LAs $\to$ 1DFA
  double exponential

- det-1-LAs $\to$ 1DFA
  exponential

Size Costs of Simulations

1-LAs versus Finite Automata

[Wagner&Wechsung '86, P.&Pisoni '14]

- 1-LAs $\rightarrow$ 1NFA
  exponential
- 1-LAs $\rightarrow$ 1DFA
  double exponential
- det-1-LAs $\rightarrow$ 1DFA
  exponential

Size Costs of Simulations

1-LAs versus Finite Automata [Wagner&Wechsung '86, P.&Pisoni '14]

- 1-LA*s* $\rightarrow$ 1NFA
  exponential
- 1-LA*s* $\rightarrow$ 1DFA
  double exponential
- det-1-LA*s* $\rightarrow$ 1DFA
  exponential

Double role of nondeterminism in 1-LA*s*

On a tape cell:

First visit: To overwrite the content
by a nondeterministically chosen symbol σ

Next visits: To select a transition

the set of available transitions depends on σ!

The Unary Case

Previous gaps are witnessed using languages B_n, defined over a *two letter alphabet*

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- An exponential gap [P.&Prigioniero '19]
  Languages $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs of size $O(n)$
  - Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P. '00]

The Unary Case

Previous gaps are witnessed using languages B_n, defined over a *two letter alphabet*

What happens in the unary case?

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- An exponential gap [P.&Prigioniero '19]
  Languages $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs of size $O(n)$
  - Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P. '00]

The Unary Case

Previous gaps are witnessed using languages B_n, defined over a *two letter alphabet*

What happens in the unary case?

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- An exponential gap [P.&Prigioniero '19]
  Languages $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs of size $O(n)$
  - Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P. '00]

The Unary Case

Previous gaps are witnessed using languages B_n, defined over a *two letter alphabet*

What happens in the unary case?

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- An exponential gap [P.&Prigioniero '19]
  Languages $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs of size $O(n)$
  - Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P.'00]

The Unary Case

Previous gaps are witnessed using languages B_n, defined over a *two letter alphabet*

What happens in the unary case?

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- An exponential gap [P.&Prigioniero '19]
  Languages $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs of size $O(n)$
  - Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P. '00]

The Unary Case

Previous gaps are witnessed using languages B_n, defined over a *two letter alphabet*

What happens in the unary case?

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- An exponential gap [P.&Prigioniero '19]
  Languages $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs of size $O(n)$
  - Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P.'00]

Size of Limited Automata vs Finite Automata

1-LA

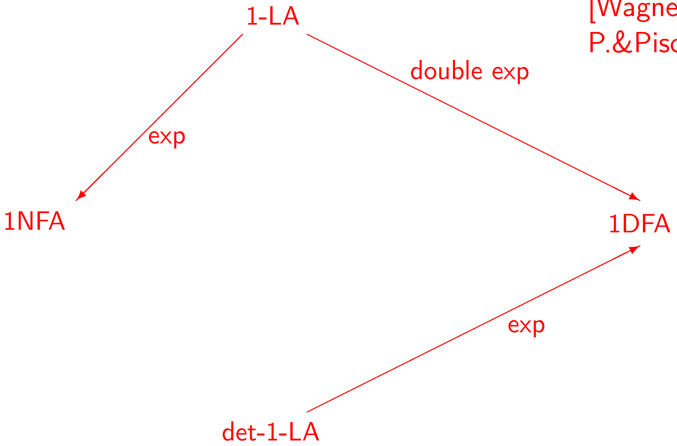
1NFA

1DFA

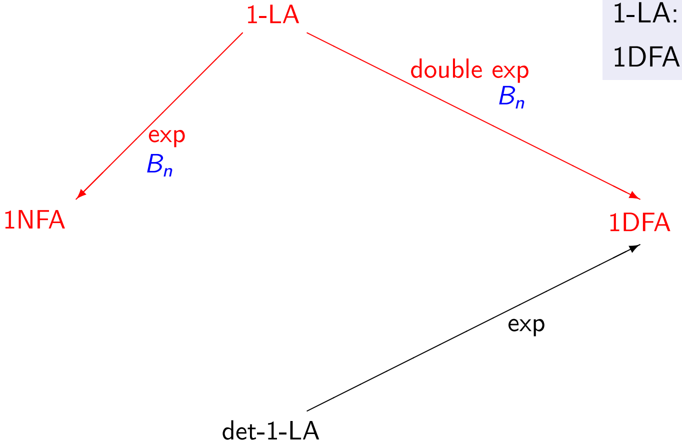
det-1-LA

Size of Limited Automata vs Finite Automata

Simulations
[Wagner&Wechsung '86, P.&Pisoni '14]

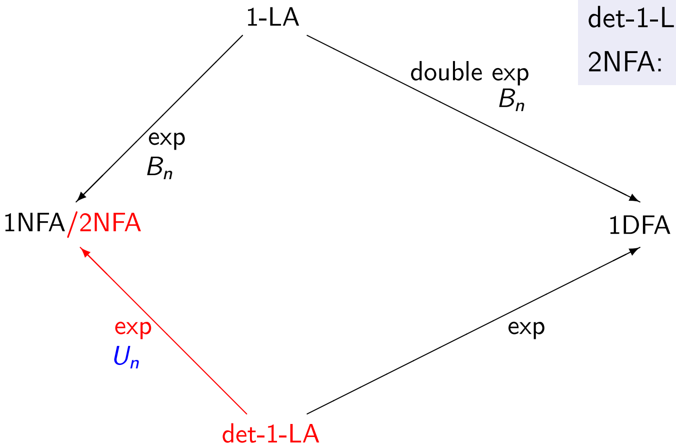


Size of Limited Automata vs Finite Automata



$B_n \subseteq \{0,1\}^*$	
1-LA:	size $O(n)$
1DFA:	$\geq 2^{2^n}$ states

Size of Limited Automata vs Finite Automata



$U_n = \{a^{2^n}\}^*$

det-1-LA: size $O(n)$

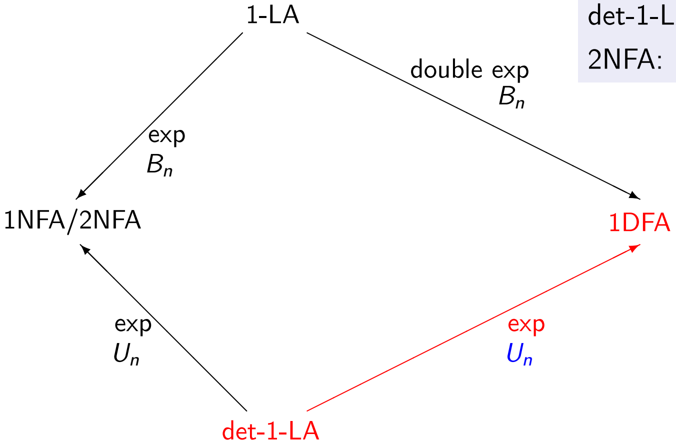
2NFA: $\geq 2^n$ states

Size of Limited Automata vs Finite Automata

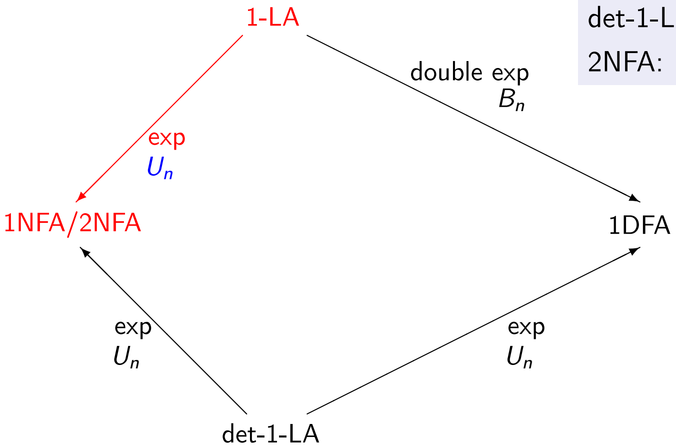
$U_n = \{a^{2^n}\}^*$

det-1-LA: size $O(n)$

2NFA: $\geq 2^n$ states



Size of Limited Automata vs Finite Automata

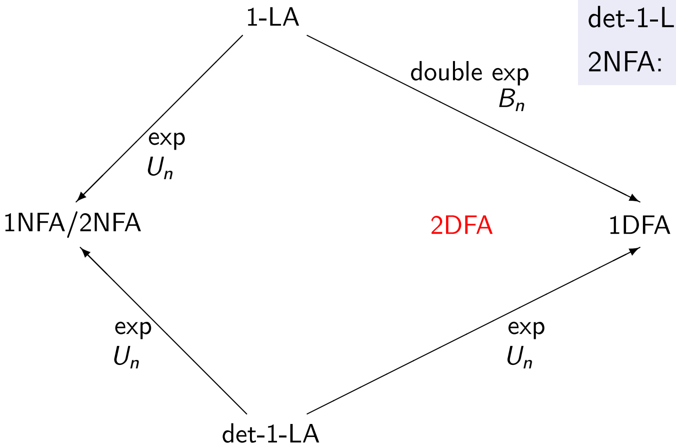


$U_n = \{a^{2^n}\}^*$

det-1-LA: size $O(n)$

2NFA: $\geq 2^n$ states

Size of Limited Automata vs Finite Automata

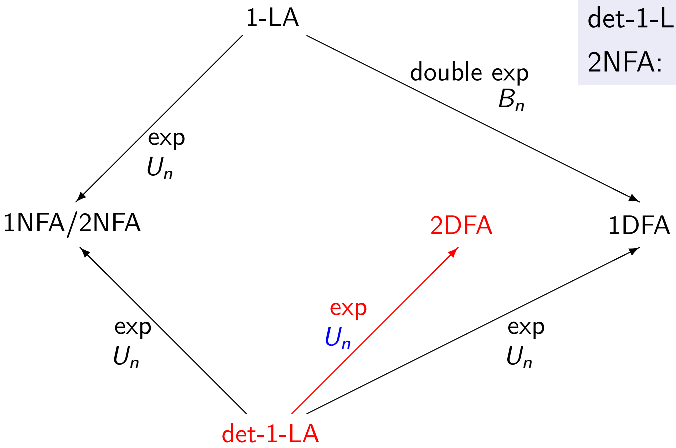


$U_n = \{a^{2^n}\}^*$

det-1-LA: size $O(n)$

2NFA: $\geq 2^n$ states

Size of Limited Automata vs Finite Automata



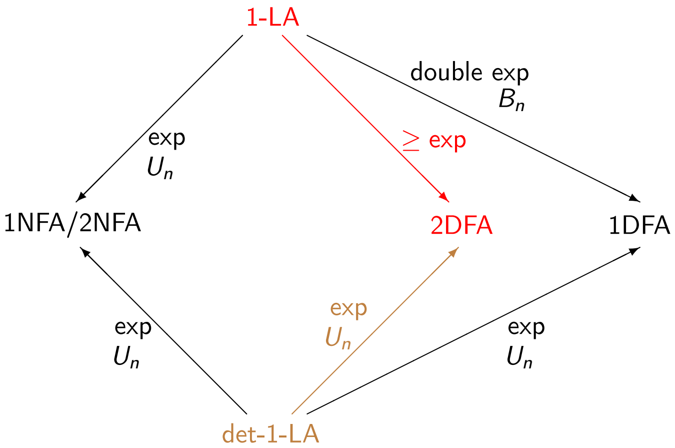
$U_n = \{a^{2^n}\}^*$

det-1-LA: size $O(n)$

2NFA: $\geq 2^n$ states

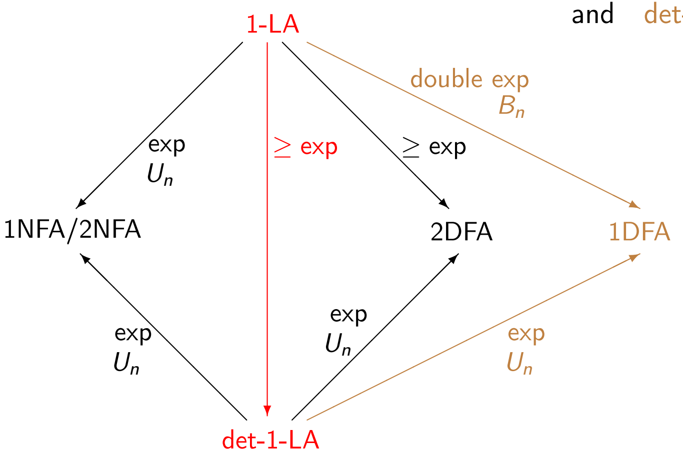
Size of Limited Automata vs Finite Automata

From det-1-LA→2DFA

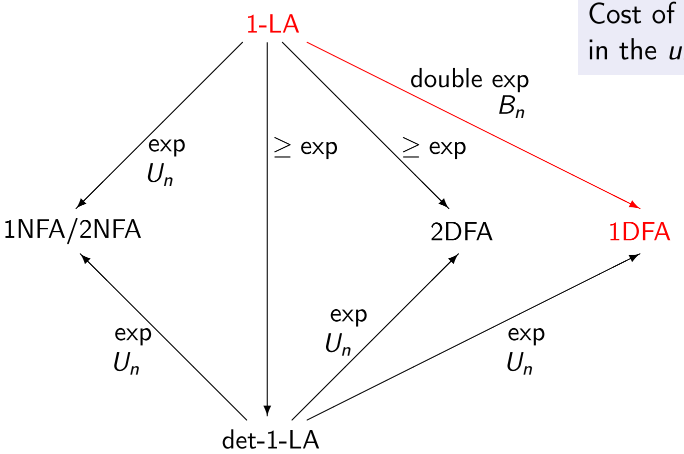


Size of Limited Automata vs Finite Automata

From 1-LA→1DFA
and det-1-LA→1DFA



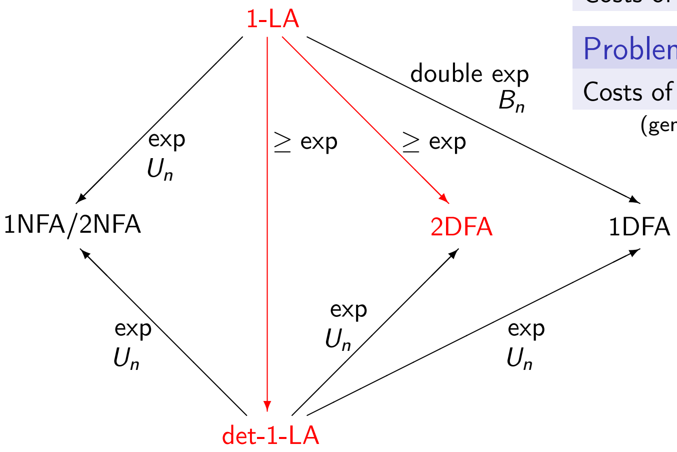
Size of Limited Automata vs Finite Automata



Problem 1

Cost of 1-LA $\rightarrow$ 1DFA in the *unary case*

Size of Limited Automata vs Finite Automata



Problem 2

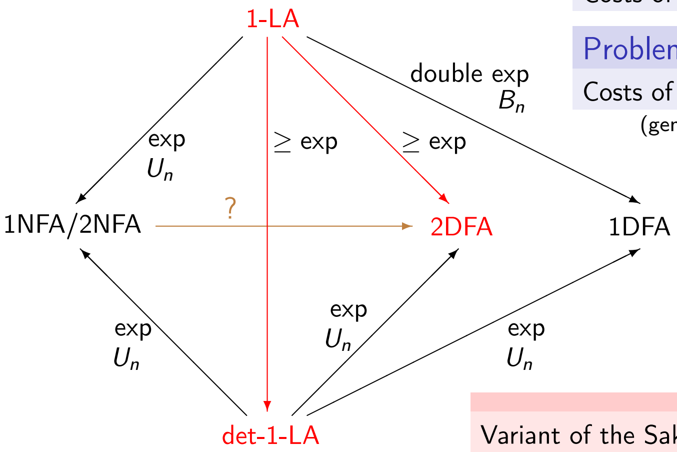
Costs of 1-LA $\to$ det-1-LA

Problem 3

Costs of 1-LA $\to$ 2DFA

(general and unary case)

Size of Limited Automata vs Finite Automata



Problem 2

Costs of 1-LA $\to$ det-1-LA

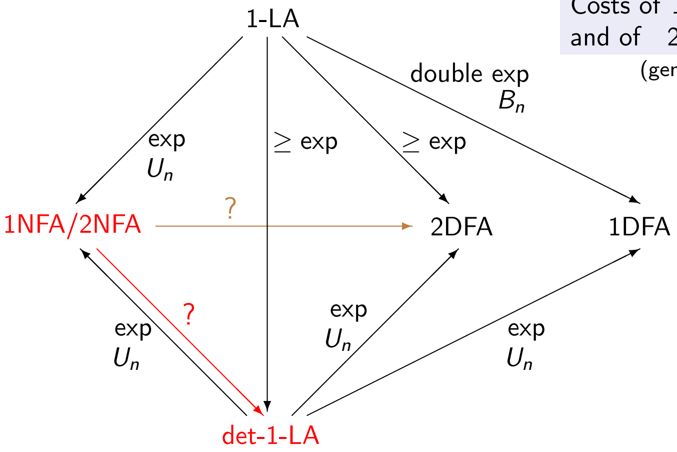
Problem 3

Costs of 1-LA $\to$ 2DFA

(general and unary case)

Variant of the Sakoda and Sipser question 1NFA/2NFA $\to$ 2DFA

Size of Limited Automata vs Finite Automata

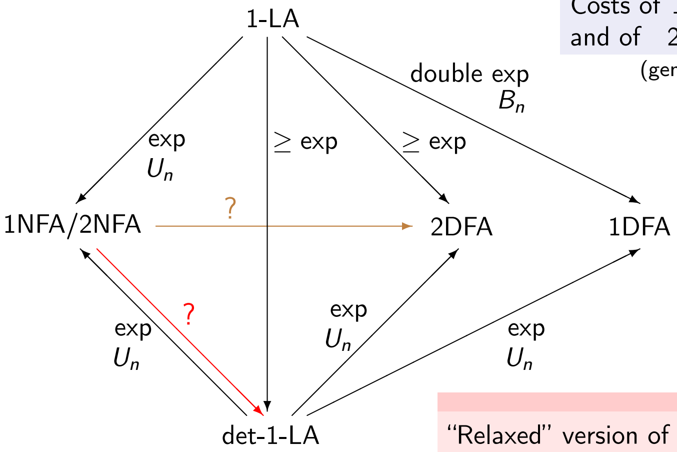


Problem 4

Costs of 1NFA $\rightarrow$ det-1-LA
and of 2NFA $\rightarrow$ det-1-LA

(general and unary case)

Size of Limited Automata vs Finite Automata



Problem 4

Costs of 1NFA $\rightarrow$ det-1-LA
and of 2NFA $\rightarrow$ det-1-LA
(general and unary case)

"Relaxed" version of the Sakoda and Sipser question 1NFA/2NFA $\rightarrow$ 2DFA

Variants of Limited Automata

Further Restrictions

Restrictions of 2-limited automata which still characterize CFLs:

- Forgetting automata [Jancar&Mráz&Plátek '96]
- Strongly limited automata [P.'15]
- ....

Active Visits and Return Complexity [Wechsung '75]

Active visit to a tape cell: any visit overwriting the content

Active Visits and Return Complexity [Wechsung '75]

Active visit to a tape cell: any visit overwriting the content

d-limited automata (dual d-return complexity)
Only *the first d visits* to a tape cell can be active

Active Visits and Return Complexity [Wechsung '75]

Active visit to a tape cell: any visit overwriting the content

d-limited automata (dual d-return complexity)
Only *the first d visits* to a tape cell can be active

d-return complexity (ret-c(d))
Only *the last d visits* to a tape cell can be active

Active Visits and Return Complexity [Wechsung '75]

Active visit to a tape cell: any visit overwriting the content

d-limited automata (dual d-return complexity)
Only *the first d visits* to a tape cell can be active

d-return complexity (ret-c(d))
Only *the last d visits* to a tape cell can be active

- ret-c(1): regular languages
- ret-c(d), $d \geq 2$: context-free languages [Wechsung '75]
- det-ret-c(2): not comparable with DCFL [Peckel '77]
  - PAL $\in$ det-ret-c(2) $\setminus$ DCFL
  - $\{a^n b^{n+m} a^m \mid n, m > 0\} \in$ DCFL$\setminus$det-ret-c(2)

Active Visits and Return Complexity [Wechsung '75]

Active visit to a tape cell: any visit overwriting the content

d-limited automata (dual d-return complexity)
Only *the first d visits* to a tape cell can be active

d-return complexity (ret-c(d))
Only *the last d visits* to a tape cell can be active

- ret-c(1): regular languages
- ret-c(d), $d \geq 2$: context-free languages [Wechsung '75]
- det-ret-c(2): not comparable with DCFL [Peckel '77]
  - PAL $\in$ det-ret-c(2) $\setminus$ DCFL
  - $\{a^n b^{n+m} a^m \mid n, m > 0\} \in$ DCFL$\setminus$det-ret-c(2)

Active Visits and Return Complexity [Wechsung '75]

Active visit to a tape cell: any visit overwriting the content

d-limited automata (dual d-return complexity)
Only *the first d visits* to a tape cell can be active

d-return complexity (ret-c(d))
Only *the last d visits* to a tape cell can be active

- ret-c(1): regular languages
- ret-c(d), $d \geq 2$: context-free languages [Wechsung '75]
- det-ret-c(2): not comparable with DCFL [Peckel '77]
  - PAL $\in$ det-ret-c(2) $\setminus$ DCFL
  - $\{a^n b^{n+m} a^m \mid n, m > 0\} \in$ DCFL$\setminus$det-ret-c(2)

Conclusion

Final Remarks

- 2-limited automata:
  interesting machine characterization of CFL
- 1-limited automata:
  stimulating open problems in descriptional complexity,
  connections with the question of Sakoda and Sipser
- *Reversible limited automata*:
  computational and descriptional power
  [Kutrib&Wendlandt '17]
- *Probabilistic limited automata*:
  Probabilistic extensions [Yamakami '19]
- *Connections with nest word automata (input-driven* PDAs*)*:
  any investigation?

Final Remarks

- 2-limited automata:
  interesting machine characterization of CFL
- 1-limited automata:
  stimulating open problems in descriptional complexity,
  connections with the question of Sakoda and Sipser
- *Reversible limited automata*:
  computational and descriptional power
  [Kutrib&Wendlandt '17]
- *Probabilistic limited automata*:
  Probabilistic extensions [Yamakami '19]
- *Connections with nest word automata (input-driven* PDAs*)*:
  any investigation?

Final Remarks

- 2-limited automata:
  interesting machine characterization of CFL
- 1-limited automata:
  stimulating open problems in descriptional complexity,
  connections with the question of Sakoda and Sipser
- *Reversible limited automata*:
  computational and descriptional power
  [Kutrib&Wendlandt '17]
- *Probabilistic limited automata*:
  Probabilistic extensions [Yamakami '19]
- *Connections with nest word automata (input-driven* PDAs*)*:
  any investigation?

Final Remarks

- 2-limited automata:
  interesting machine characterization of CFL
- 1-limited automata:
  stimulating open problems in descriptional complexity,
  connections with the question of Sakoda and Sipser
- *Reversible limited automata*:
  computational and descriptional power
  [Kutrib&Wendlandt '17]
- *Probabilistic limited automata*:
  Probabilistic extensions [Yamakami '19]
- *Connections with nest word automata (input-driven* PDAs*)*:
  any investigation?

Final Remarks

- 2-limited automata:
  interesting machine characterization of CFL
- 1-limited automata:
  stimulating open problems in descriptional complexity,
  connections with the question of Sakoda and Sipser
- *Reversible limited automata*:
  computational and descriptional power
  [Kutrib&Wendlandt '17]
- *Probabilistic limited automata*:
  Probabilistic extensions [Yamakami '19]
- *Connections with nest word automata (input-driven* PDAs*)*:
  any investigation?

Thank you for your attention!