Limited Automata: Properties, Complexity, Variants

Giovanni Pighizzini

Dipartimento di Informatica
Università degli Studi di Milano, Italy

DCFS 2019 – Košice, Slovakia
July 17, 2019

UNIVERSITÀ DEGLI STUDI
DI MILANO

Introduction

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

Introduction

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

Introduction

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

Outline

- An Introductory Example
- Definition of Limited Automata
- Computational Power
- Descriptional Complexity (Part I)
- Limited Automata and Unary Languages
- Descriptional Complexity (Part II)
  ...open problems...
- Variants and related models
- Conclusion

Outline

Outline

- An Introductory Example
- Definition of Limited Automata
- Computational Power
- Descriptional Complexity (Part I)
- Limited Automata and Unary Languages
- Descriptional Complexity (Part II)
  ...open problems...
- Variants and related models
- Conclusion

Outline

Introduction

Outline

- An Introductory Example
- Definition of Limited Automata
- Computational Power
- Descriptional Complexity (Part I)
- Limited Automata and Unary Languages
- Descriptional Complexity (Part II)
  ...open problems...
- Variants and related models
- Conclusion

Outline

- An Introductory Example
- Definition of Limited Automata
- Computational Power
- Descriptional Complexity (Part I)
- Limited Automata and Unary Languages
- Descriptional Complexity (Part II)
  ...open problems...
- Variants and related models
- Conclusion

Introduction

Outline

- An Introductory Example
- Definition of Limited Automata
- Computational Power
- Descriptional Complexity (Part I)
- Limited Automata and Unary Languages
- Descriptional Complexity (Part II)
  ...open problems...
- Variants and related models
- Conclusion

Introduction

Outline

- An Introductory Example
- Definition of Limited Automata
- Computational Power
- Descriptional Complexity (Part I)
- Limited Automata and Unary Languages
- Descriptional Complexity (Part II)
  ...open problems...
- Variants and related models
- Conclusion

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

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

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

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

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

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

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 (( )))

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 (( )))

$\longleftarrow$

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 (( )))

$\longleftarrow$

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 (( )))

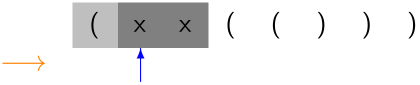
How to recognize if a sequence of brackets its correctly balanced?

- *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



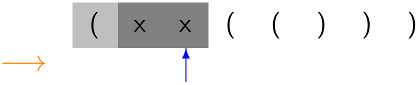
How to recognize if a sequence of brackets its correctly balanced?

- *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



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 ((x))

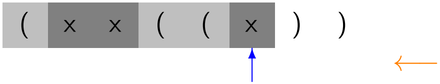
How to recognize if a sequence of brackets its correctly balanced?

- *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



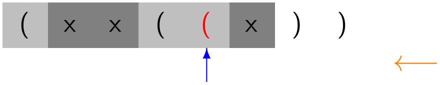
How to recognize if a sequence of brackets its correctly balanced?

- *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

(x x (x x))

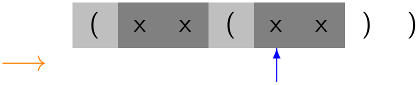
How to recognize if a sequence of brackets its correctly balanced?

- *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



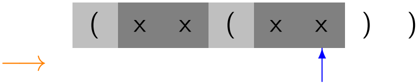
How to recognize if a sequence of brackets its correctly balanced?

- *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



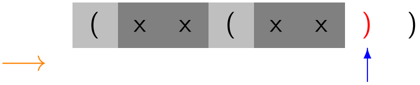
How to recognize if a sequence of brackets its correctly balanced?

- *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



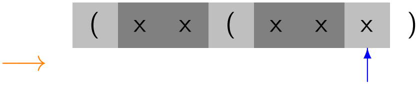
How to recognize if a sequence of brackets its correctly balanced?

- *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



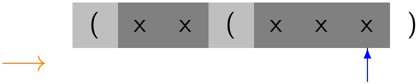
How to recognize if a sequence of brackets its correctly balanced?

- *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

(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

(	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

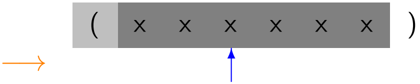
How to recognize if a sequence of brackets its correctly balanced?

- *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



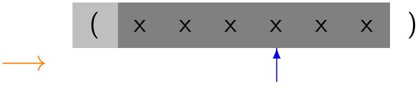
How to recognize if a sequence of brackets its correctly balanced?

- *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



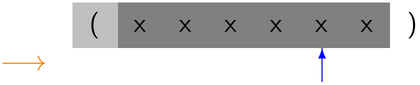
How to recognize if a sequence of brackets its correctly balanced?

- *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

(x x x x x x)

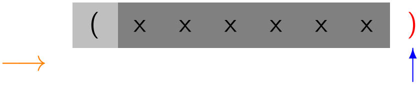
How to recognize if a sequence of brackets its correctly balanced?

- *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



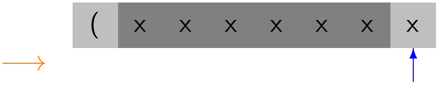
How to recognize if a sequence of brackets its correctly balanced?

- *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

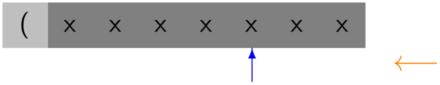
How to recognize if a sequence of brackets its correctly balanced?

- *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



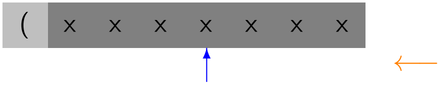
How to recognize if a sequence of brackets its correctly balanced?

- *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

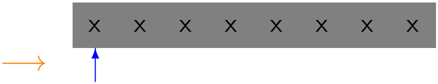
How to recognize if a sequence of brackets its correctly balanced?

- *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



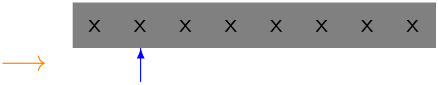
How to recognize if a sequence of brackets its correctly balanced?

- *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



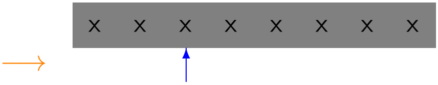
How to recognize if a sequence of brackets its correctly balanced?

- *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



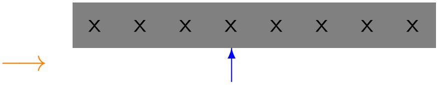
How to recognize if a sequence of brackets its correctly balanced?

- *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



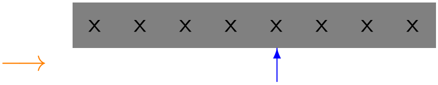
How to recognize if a sequence of brackets its correctly balanced?

- *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



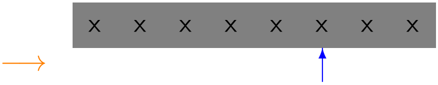
How to recognize if a sequence of brackets its correctly balanced?

- *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



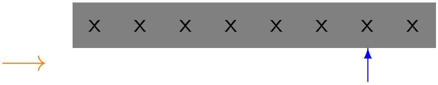
How to recognize if a sequence of brackets its correctly balanced?

- *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



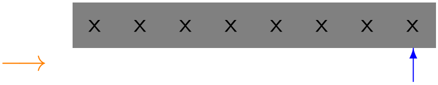
How to recognize if a sequence of brackets its correctly balanced?

- *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

Definition and Computational Power

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

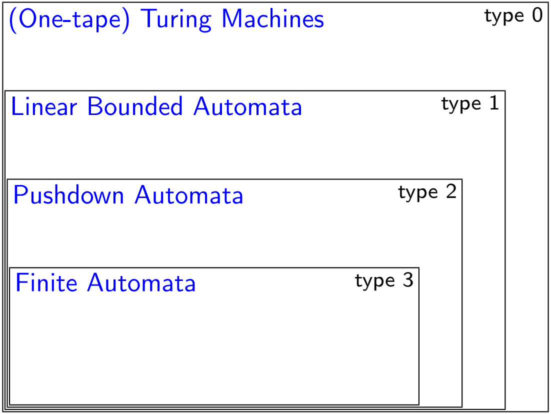
Fixed an integer $d \geq 1$, a *d-limited automaton* is

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

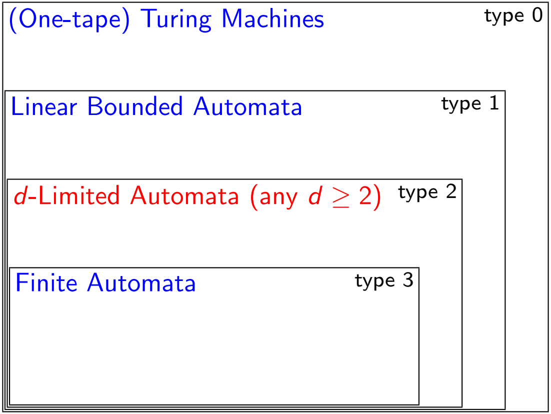
Computational power

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

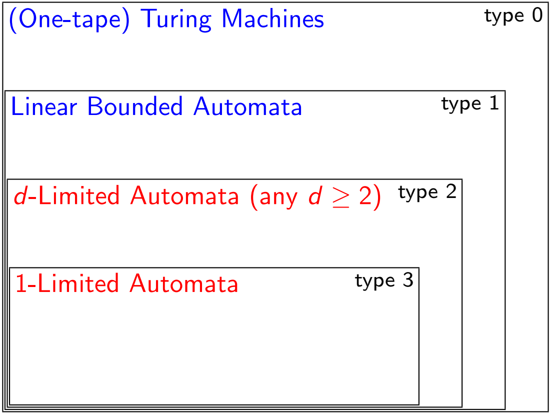
The Chomsky Hierarchy



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}

$$\boxed{A_D}$$

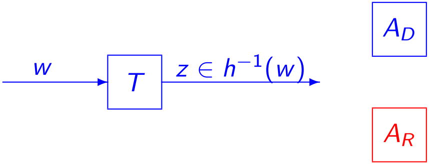
$$\xrightarrow{w} \boxed{T} \xrightarrow{z \in h^{-1}(w)}$$

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* — *Finite automaton* A_R
- $h : \Omega_k \to \Sigma^*$ *is a homomorphism* — *Transducer* T *for* h^{-1}

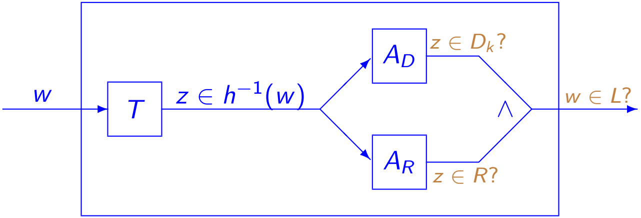


Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

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}

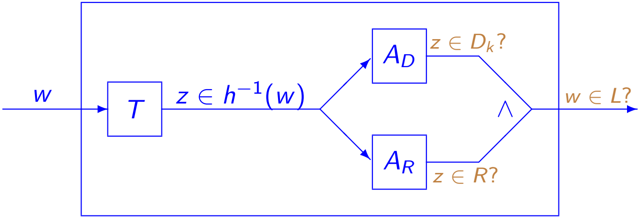


Why Each CFL is Accepted by a 2-LA [P.&Pisoni '14]

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?

Non-constant Number of Rewritings

$f(n)$*-limited automaton* ($f : \mathbf{N} \to \mathbf{N}$):
one-tape Turing machine s.t. for each accepted input w there is an accepting computation in which each tape cell is rewritten at most in the first $f(|w|)$ visits

Theorem [Wechsung&Brandstädt '79]

$$f(n)\text{-LA}s \quad \equiv \quad \text{1AuxPDA}s\text{-space}(f(n))$$

i.e., class of languages by one-way PDAs extended with an auxiliary worktape, where $O(f(n))$ space is used

Non-constant Number of Rewritings

$f(n)$*-limited automaton* ($f : \mathbf{N} \to \mathbf{N}$):
one-tape Turing machine s.t. for each accepted input w there is an accepting computation in which each tape cell is rewritten at most in the first $f(|w|)$ visits

Theorem [Wechsung&Brandstädt '79]

$$f(n)\text{-LA}s \quad \equiv \quad 1\text{AuxPDA}s\text{-space}(f(n))$$

i.e., class of languages by one-way PDAs extended with an auxiliary worktape, where $O(f(n))$ space is used

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 $(n > 0)$

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

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 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

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

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

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 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 starts 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 starts 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 starts 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 starts from the marked cells
4. Accept if the two blocks are equal

A Nondeterministic 1-Limited Automaton for B_n

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

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 starts 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

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

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!

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 $\rightarrow$ 1DFAs

At least double exponential gap!

Lower bounds for B_n

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

Finite automata

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

Proof: standard distinguishability arguments

1-LAs $\to$ 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 $\to$ PDAs

At least exponential gap!

Lower bounds for B_n

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

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!

Size Costs of Simulations

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

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

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!

Descriptional Complexity - TO BE CONTINUED...

Limited Automata and Unary Languages

Limited Automata and Unary Languages

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- Improvements in [P.&Prigioniero '19]:

Limited Automata and Unary Languages

- Preliminary observations in [P.&Pisoni '14]
- Several results in [Kutrib&Wendlandt '15]
  (including superpolynomial gaps 1-LAs $\rightarrow$ finite automata)
- Improvements in [P.&Prigioniero '19]:
  - Languages $L_n = \{a^{2^n}\}$ and $U_n = \{a^{2^n}\}^*$
  - Recognition by "small" deterministic 1-LAs
  - Exponential gaps

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	$\triangleleft$

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	$\triangleleft$

$n = 4$

- Make n sweeps of the tape
- At each sweep overwrite each "odd" a

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	X	a	X	a	X	a	X	a	X	a	X	a	X	a	X	a	$\triangleleft$

$n = 4$

- Make n sweeps of the tape
- At each sweep overwrite each "odd" a

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: “divide” n times the input length by 2

$\triangleright$	X	X	X	a	X	X	X	a	X	X	X	a	X	X	X	a	$\triangleleft$

$n = 4$

- Make n sweeps of the tape
- At each sweep overwrite each “odd” a

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	X	X	X	X	X	X	X	a	X	X	X	X	X	X	X	a	$\triangleleft$

$n = 4$

- Make n sweeps of the tape
- At each sweep overwrite each "odd" a

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	a	$\triangleleft$

$n = 4$

- Make n sweeps of the tape
- At each sweep overwrite each "odd" a
- Accept if only exactly one a is left on the tape

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	a

$\triangleleft$

$n = 4$

- Make n sweeps of the tape
- At each sweep overwrite each "odd" a
- Accept if only exactly one a is left on the tape
- $O(n)$ states

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	$\triangleleft$

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	0	a	0	a	0	a	0	a	0	a	0	a	0	a	0	a	$\triangleleft$

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

▷	0	1	0	a	0	1	0	a	0	1	0	a	0	1	0	a	◁

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	0	1	0	2	0	1	0	a	0	1	0	2	0	1	0	a	$\triangleleft$

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	a	$\triangleleft$

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

▷	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	◁

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

A Linear Bounded Automaton for $L_n = \{a^{2^n}\}$

Idea: "divide" n times the input length by 2

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

Possible variation:

- Overwrite using the number of current sweep (counting from 0)

We can build a 1-LA that, for each tape cell,
guesses the number of the sweep
in which this linear bounded automaton rewrites the cell

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\rhd$ | a | a | a | a | a | a | a | a | a | a | a | a | a | a | a | a | $\lhd$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$

0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \dots, n\}$
- *$(i+2)$th sweep, $i = 0, \dots, n$:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \dots, n\}$
- *$(i+2)$th sweep, $i = 0, \dots, n$:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$
- *$(i + 2)$th sweep, $i = 0, \ldots, n$:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$
- *$(i+2)$th sweep, $i = 0, \ldots, n$:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$
- $(i + 2)$*th sweep,* $i = 0, \ldots, n$*:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$
- $(i + 2)$*th sweep,* $i = 0, \ldots, n$*:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$
- $(i+2)$*th sweep,* $i = 0, \ldots, n$*:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$
- Size $O(n)$

A 1-Limited Automaton for $L_n = \{a^{2^n}\}$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

$n = 4$

- *1st sweep:*
  For each cell, guess and write a symbol in $\{0, 1, \ldots, n\}$
- $(i+2)$*th sweep,* $i = 0, \ldots, n$*:*
  Verify that the symbol i occurs in all odd positions,
  where positions are counted ignoring cells containing $j < i$
- Size $O(n)$

We can do better!

Size $O(n)$, only *deterministic* transitions

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

- First element: 0
- Next elements: $w \to w w'$
  - w part already constructed,
  - w' copy of w, where the last symbol replaced by its successor

0

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

- First element: 0
- Next elements: $w \to w w'$
  - w part already constructed,
  - w' copy of w, where the last symbol replaced by its successor

0 1

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

- First element: 0
- Next elements: $w \to ww'$
  - w part already constructed,
  - w' copy of w, where the last symbol replaced by its successor

0 1 0 2

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

- First element: 0
- Next elements: $w \rightarrow w w'$
  - w part already constructed,
  - w' copy of w, where the last symbol replaced by its successor

0 1 0 2 0 1 0 3

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

- First element: 0
- Next elements: $w \to w w'$
  - w part already constructed,
  - w' copy of w, where the last symbol replaced by its successor

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

The String on The Tape

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Prefix of the infinite sequence produced as follows:

- First element: 0
- Next elements: $w \to w w'$
  - w part already constructed,
  - w' copy of w, where the last symbol replaced by its successor

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4

Binary Carry Sequence

The Binary Carry Sequence: Definition

$$0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 4 \quad \cdots$$

The Binary Carry Sequence: Definition

0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\cdots$
σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}	σ_{16}	$\cdots$

Infinite sequence of integers $\sigma_1, \sigma_2, \ldots$ with:

$\sigma_j :=$ exponent of the *highest power of* 2 which divides j

The Binary Carry Sequence: Properties

- $w_j := \sigma_1\sigma_2\cdots\sigma_j$

  i.e., prefix of length j of the binary carry sequence

$$w_{11} = \begin{array}{ccccccccccc} 0 & 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 & \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \end{array}$$

The Binary Carry Sequence: Properties

- $w_j := \sigma_1\sigma_2\cdots\sigma_j$

  i.e., prefix of length j of the binary carry sequence

- $BIS(w_j)$:= *Backward Increasing Sequence* of w_j

  longest increasing sequence obtained with a greedy method by inspecting w_j from the end

$$w_{11} = \begin{array}{ccccccccccc} 0 & 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 & \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \end{array}$$

The Binary Carry Sequence: Properties

- $w_j := \sigma_1 \sigma_2 \cdots \sigma_j$

  i.e., prefix of length j of the binary carry sequence

- $BIS(w_j)$:= *Backward Increasing Sequence* of w_j

  longest increasing sequence obtained with a greedy method by inspecting w_j from the end

$$w_{11} = \begin{array}{ccccccccccc} 0 & 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 & \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \end{array}$$

$$BIS(w_{11})^R = \quad \ldots \quad 0$$

The Binary Carry Sequence: Properties

- $w_j := \sigma_1 \sigma_2 \cdots \sigma_j$

  i.e., prefix of length j of the binary carry sequence

- $BIS(w_j)$:= *Backward Increasing Sequence* of w_j

  longest increasing sequence obtained with a greedy method by inspecting w_j from the end

$w_{11} =$	0	1	0	2	0	1	0	3	0	1	0
	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}

$$BIS(w_{11})^R = \quad \ldots \quad 1 \qquad 0$$

The Binary Carry Sequence: Properties

- $w_j := \sigma_1 \sigma_2 \cdots \sigma_j$

  i.e., prefix of length j of the binary carry sequence

- $BIS(w_j) :=$ *Backward Increasing Sequence* of w_j

  longest increasing sequence obtained with a greedy method by inspecting w_j from the end

$$w_{11} = \begin{array}{ccccccccccc} 0 & 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 & \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \end{array}$$

$$BIS(w_{11})^R = 3 \quad 1 \quad 0$$

The Binary Carry Sequence: Properties

- $w_j := \sigma_1\sigma_2\cdots\sigma_j$

  i.e., prefix of length j of the binary carry sequence

- $BIS(w_j)$:= *Backward Increasing Sequence* of w_j

  longest increasing sequence obtained with a greedy method by inspecting w_j from the end

$$w_{11} = \begin{array}{ccccccccccc} 0 & 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 & \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \end{array}$$

$$BIS(w_{11})^R = \quad 3 \quad 1 \quad 0$$

$$11 = 2^3 + 2^1 + 2^0$$

$$= (1011)_2$$

The Binary Carry Sequence: Properties

- $w_j := \sigma_1\sigma_2\cdots\sigma_j$

  i.e., prefix of length j of the binary carry sequence

- $BIS(w_j)$:= *Backward Increasing Sequence* of w_j

  longest increasing sequence obtained with a greedy method by inspecting w_j from the end

$$w_{11} = \begin{array}{ccccccccccc} 0 & 1 & 0 & 2 & 0 & 1 & 0 & 3 & 0 & 1 & 0 \\ \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 & \sigma_7 & \sigma_8 & \sigma_9 & \sigma_{10} & \sigma_{11} \end{array}$$

$$BIS(w_{11})^R = 3 \quad 1 \quad 0$$

$$11 = 2^3 + 2^1 + 2^0$$

$$= (1011)_2$$

Property 1

$BIS(w_j)$ = positions of 1s in the binary representation of j

The Binary Carry Sequence: Properties

$$w_{11} = \quad 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0$$

$$BIS(w_{11})^R = \quad 3 \qquad 1 \qquad 0$$

$$11 = 2^3 + 2^1 + 2^0 = (1011)_2$$

The Binary Carry Sequence: Properties

$w_{11} = \quad 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0$

$BIS(w_{11})^R = \quad 3 \quad\quad 1 \quad\quad 0$

$$11 = 2^3 + 2^1 + 2^0 = (1011)_2$$

$$12 = 2^3 + 2^2 \quad\quad = (1100)_2$$

The Binary Carry Sequence: Properties

$$w_{11} = \quad 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0$$

$$BIS(w_{11})^R = \quad 3 \qquad 1 \qquad 0$$

$$11 = 2^3 + 2^1 + 2^0 = (1011)_2$$

$$12 = 2^3 + 2^2 = (1100)_2$$

$$BIS(w_{12})^R = \quad 3 \qquad 2$$

The Binary Carry Sequence: Properties

$$w_{11} = \; 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0$$

$$BIS(w_{11})^R = \; 3 \quad 1 \quad 0$$

$$11 = 2^3 + 2^1 + 2^0 = (1011)_2$$

$$12 = 2^3 + 2^2 = (1100)_2$$

$$BIS(w_{12})^R = \; 3 \quad 2$$

$$w_{12} = \; 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0 \quad 2$$

The Binary Carry Sequence: Properties

$$w_{11} = 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0$$

$$BIS(w_{11})^R = 3 \quad 1 \quad 0$$

$$11 = 2^3 + 2^1 + 2^0 = (1011)_2$$

$$12 = 2^3 + 2^2 = (1100)_2$$

$$BIS(w_{12})^R = 3 \quad 2$$

$$w_{12} = 0 \quad 1 \quad 0 \quad 2 \quad 0 \quad 1 \quad 0 \quad 3 \quad 0 \quad 1 \quad 0 \quad 2$$

Property 2

The symbol of the binary carry sequence in position $j+1$ is the smallest nonnegative integer that does not occur in $BIS(w_j)$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\rhd$	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	$\lhd$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\rhd$	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	$\lhd$
								$\uparrow i = 2$									

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	$\triangleleft$

$\uparrow i = 2$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

▷	0	1	0	2	0	1	0	3	0	1	0	a	a	a	a	a	◁

$i = 2$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

▷	0	1	0	2	0	1	0	3	0	1	0	2	a	a	a	a	◁

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	a	a	a	a	$\triangleleft$

...

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	a	a	a	$\triangleleft$
													$\uparrow$				

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	a	a	a	$\triangleleft$

$\cdots$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	a	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$ | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 3 | 0 | 1 | 0 | 2 | 0 | 1 | a | a | $\triangleleft$

$\ldots$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	a	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	a	$\triangleleft$

$\cdots$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i
- When n is written on a cell:
  - Move one position to the right
  - Accept iff the current cell contains the right endmarker

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

- 0 is written on the first cell
- For $j > 0$, with w_j on the first j cells, head on cell j:
  - Move to the left to compute the smallest $i \notin BIS(w_j)$
  - Move to the right to search the first cell containing a
  - Write i
- When n is written on a cell:
  - Move one position to the right
  - Accept iff the current cell contains the right endmarker

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$

0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

$\triangleleft$

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain
  a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$

0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

$\triangleleft$

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

▷	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	◁

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain
  a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4	$\triangleleft$

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$
- Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P.'00]

det-1-LAs $\longrightarrow$ 2NFAs

Exponential gap!

A Deterministic 1-LA for $L_n = \{a^{2^n}\}$

Idea: Write on the tape a prefix of the binary carry sequence

$n = 4$

$\triangleright$

0	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4

$\triangleleft$

- Each cell is rewritten *only* in the first visit
- Tape alphabet $\{0, \ldots, n\}$
- Finite state control with $O(n)$ states
- Total size of the description $O(n)$
- With a minor modification we can obtain a deterministic 1-LA of size $O(n)$ accepting $U_n = \{a^{2^n}\}^*$
- Each 2NFA accepting U_n should have at least 2^n states [Mereghetti&P.'00]

det-1-LAs $\rightarrow$ 2NFAs

Exponential gap!

Descriptional Complexity

1-Limited Automata vs Finite Automata

Size of Limited Automata vs Finite Automata

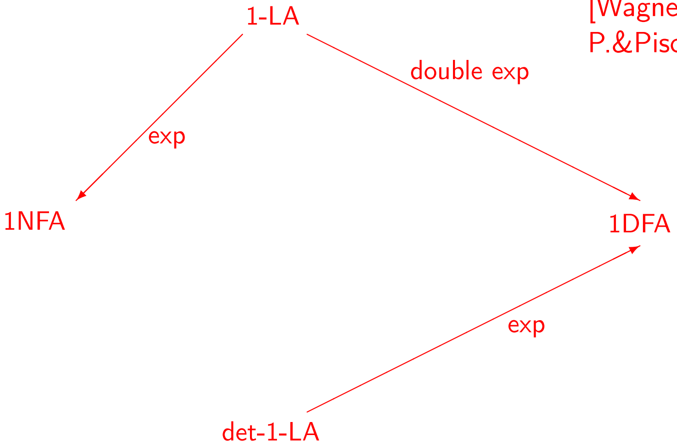
1-LA

1NFA

1DFA

det-1-LA

Size of Limited Automata vs Finite Automata



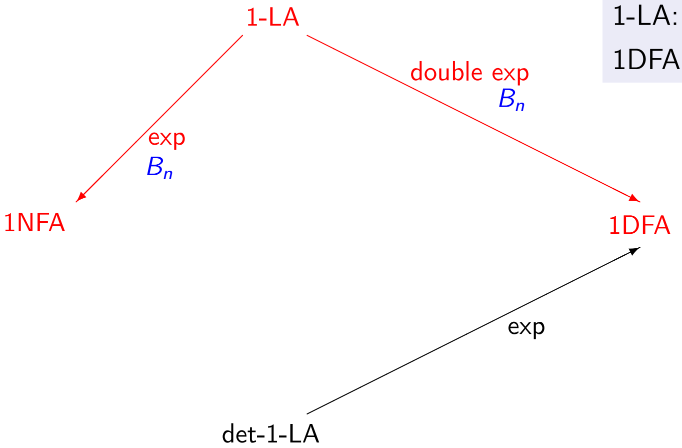
Simulations
[Wagner&Wechsung '86, P.&Pisoni '14]

Size of Limited Automata vs Finite Automata

$B_n \subseteq \{0,1\}^*$

1-LA: size $O(n)$

1DFA: $\geq 2^{2^n}$ states

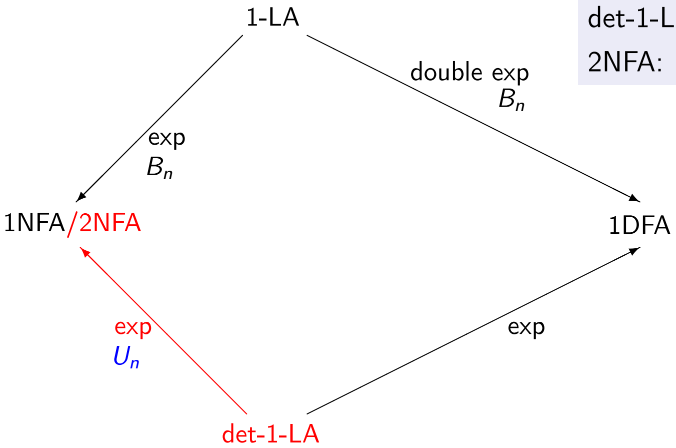


Size of Limited Automata vs Finite Automata

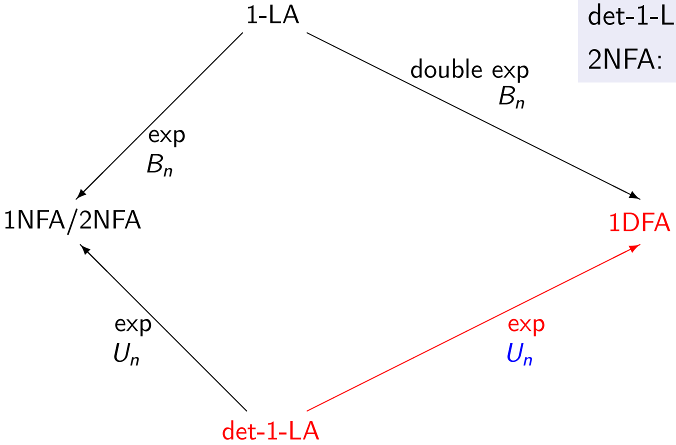
$U_n = \{a^{2^n}\}^*$

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

2NFA: $\geq 2^n$ states



Size of Limited Automata vs Finite Automata

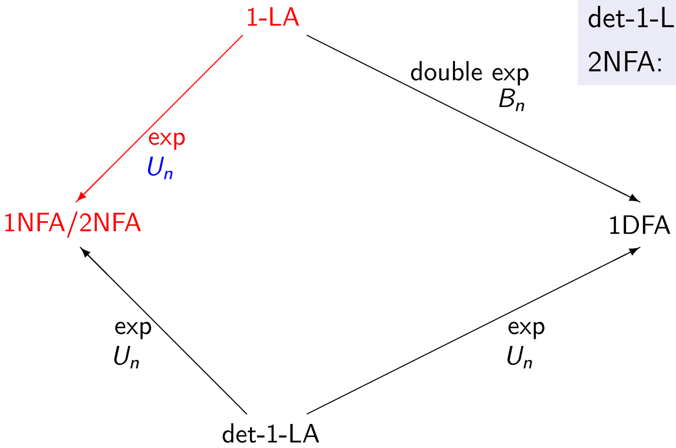


$U_n = \{a^{2^n}\}^*$

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 $O(n)$

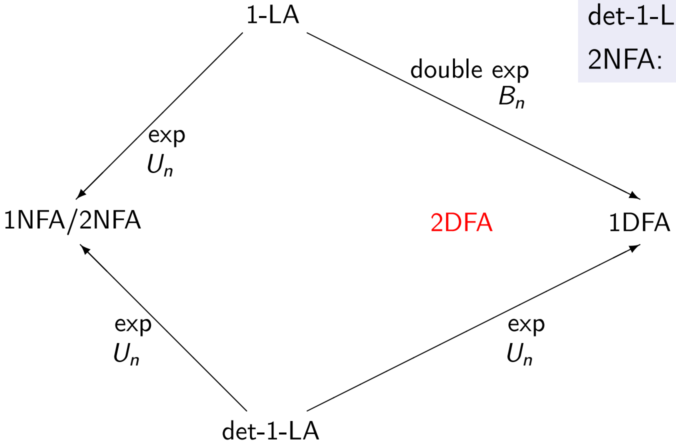
2NFA: $\geq 2^n$ states

Size of Limited Automata vs Finite Automata

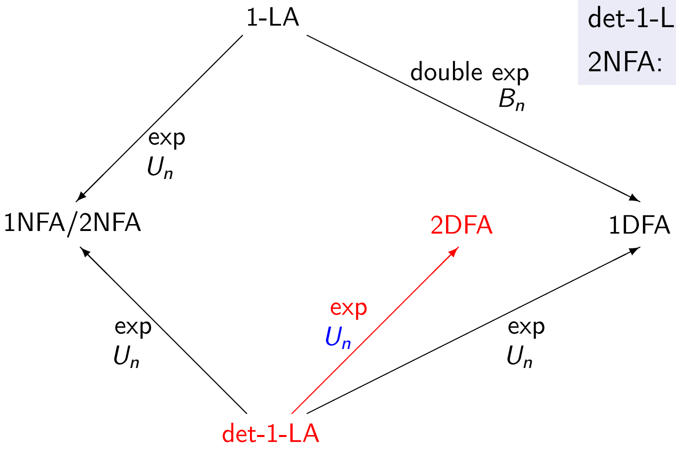
$U_n = \{a^{2^n}\}^*$

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

2NFA: $\geq 2^n$ states



Size of Limited Automata vs Finite Automata



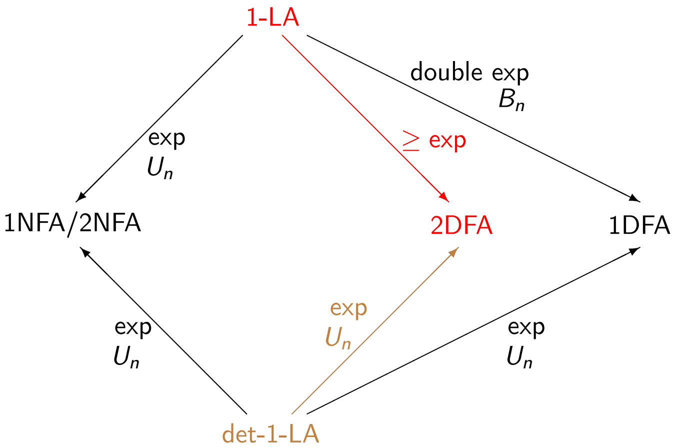
$U_n = \{a^{2^n}\}^*$

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

2NFA: $\geq 2^n$ states

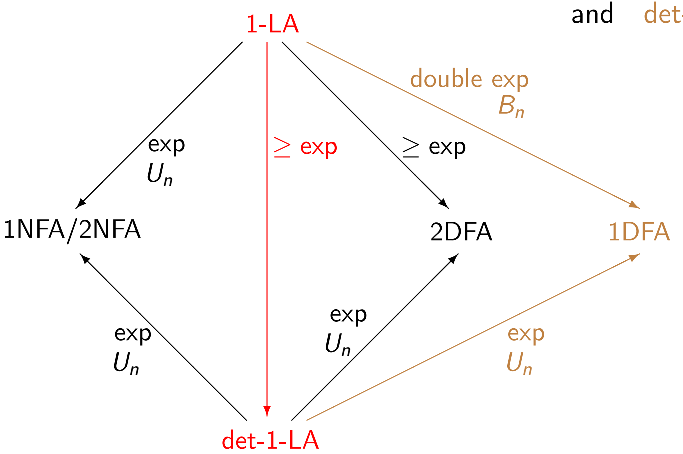
Size of Limited Automata vs Finite Automata

From det-1-LA→2DFA

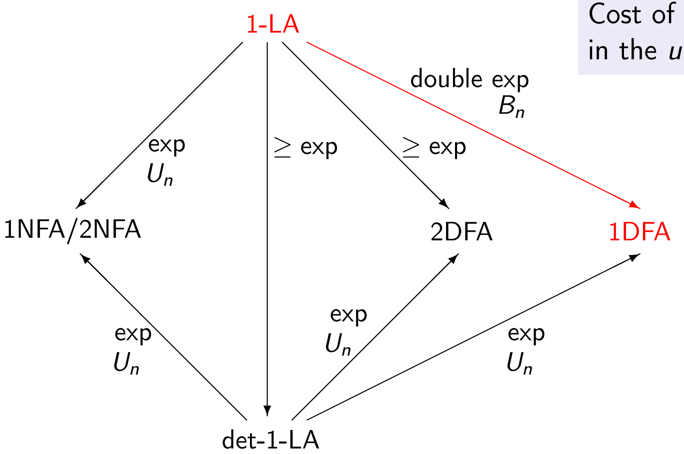


Size of Limited Automata vs Finite Automata

From 1-LA$\rightarrow$1DFA
and det-1-LA$\rightarrow$1DFA



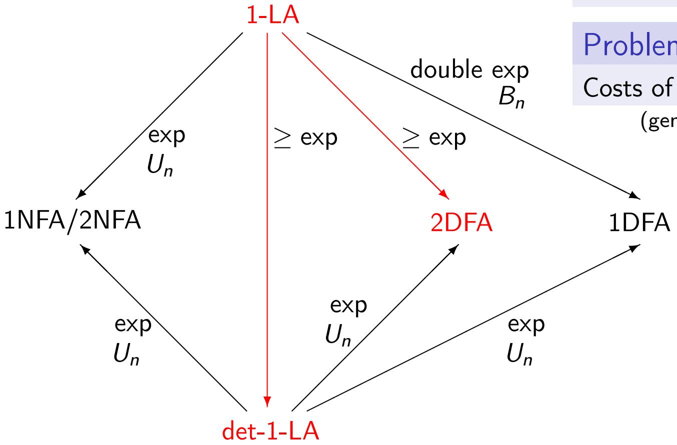
Size of Limited Automata vs Finite Automata



Problem 1

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

Size of Limited Automata vs Finite Automata



Problem 2

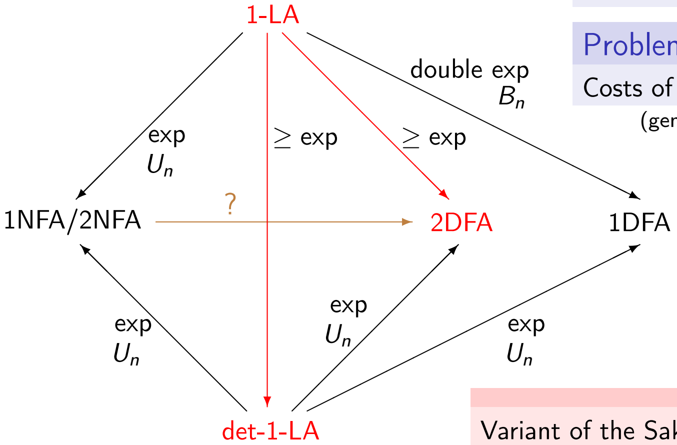
Costs of 1-LA $\rightarrow$ det-1-LA

Problem 3

Costs of 1-LA $\rightarrow$ 2DFA

(general and unary case)

Size of Limited Automata vs Finite Automata



Problem 2

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

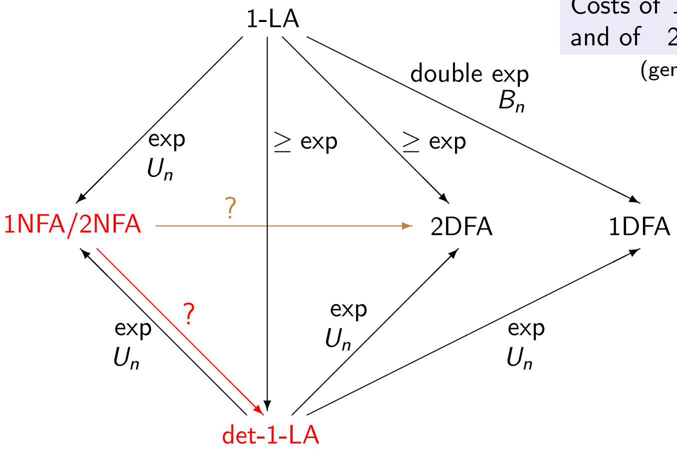
Problem 3

Costs of 1-LA $\rightarrow$ 2DFA

(general and unary case)

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

Size of Limited Automata vs Finite Automata

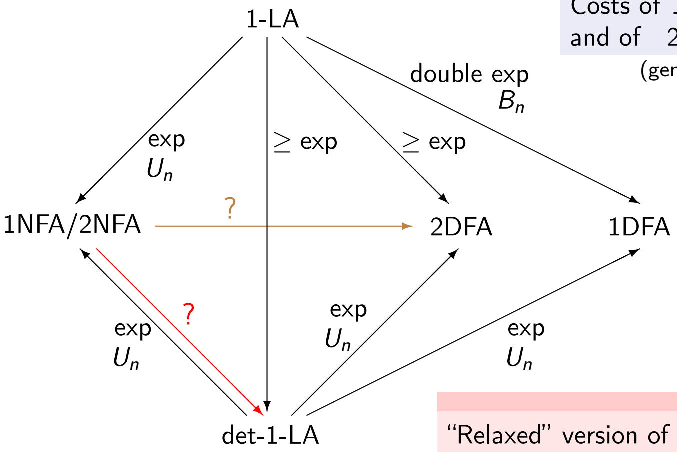


Problem 4

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

(general and unary case)

Size of Limited Automata vs Finite Automata



Problem 4

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

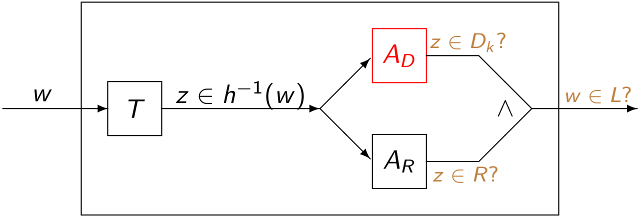
(general and unary case)

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

Variants of Limited Automata

Different Restrictions

Chomsky-Schützenberger Theorem:
Recognition of CFLs “reduced” to recognition of Dyck languages



Different Restrictions

Dyck languages are accepted without fully using capabilities of 2-limited automata

Different Restrictions

Dyck languages are accepted without fully using capabilities of 2-limited automata, e.g.,

- Moving to the right: only state q_0 is used, rewritings only when the head direction is reversed
- Moving to the left: no state change until the head direction is reversed
- Only one symbol used for rewriting
- ...

Different Restrictions

Question

Is it possible to restrict 2-limited automata without affecting their computational power?

Different Restrictions

Question

Is it possible to restrict 2-limited automata without affecting their computational power?

YES!

Forgetting Automata [Jancar&Mráz&Plátek '96]

- The content of any cell can be erased in the 1st or 2nd visit (using one fixed symbol)
- No other changes of the tape are allowed

Different Restrictions

Question

Is it possible to restrict 2-limited automata without affecting their computational power?

YES!

Forgetting Automata [Jancar&Mráz&Plátek '96]

- The content of any cell can be erased in the 1st or 2nd visit (using one fixed symbol)
- No other changes of the tape are allowed

Strongly Limited Automata [P.'15]

- Cells rewritten only while moving to the left
- Only one state is used while moving to the right

Strongly Limited Automata

- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related (while 2-limited automata can be exponentially smaller than PDAs)
- CFLs $\rightarrow$ strongly limited automata:

  conversion from CFGs which heavily uses nondeterminism
  - The class of languages accepted by *deterministic* strongly limited automata is a *proper* subclass of DCFLs.

Strongly Limited Automata

- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related (while 2-limited automata can be exponentially smaller than PDAs)
- CFLs $\to$ strongly limited automata:
  conversion from CFGs which heavily uses nondeterminism
  - The class of languages accepted by *deterministic* strongly limited automata is a *proper* subclass of DCFLs.

Strongly Limited Automata

- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related (while 2-limited automata can be exponentially smaller than PDAs)
- CFLs $\rightarrow$ strongly limited automata:
  conversion from CFGs which heavily uses nondeterminism
  - The class of languages accepted by *deterministic* strongly limited automata is a *proper* subclass of DCFLs.

Strongly Limited Automata

- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related (while 2-limited automata can be exponentially smaller than PDAs)
- CFLs $\rightarrow$ strongly limited automata:

  conversion from CFGs which heavily uses nondeterminism
  - The class of languages accepted by *deterministic* strongly limited automata is a *proper* subclass of DCFLs.

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-drive* 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-drive* 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-drive* 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-drive* 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-drive* PDAs*)*:
  any investigation?

Thank you for your attention!