Limited Automata and Regular Languages

Giovanni Pighizzini   Andrea Pisoni

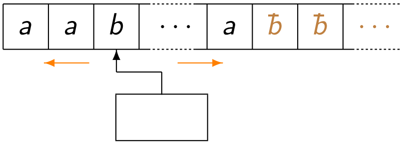
Dipartimento di Informatica
Università degli Studi di Milano, Italy

DCFS 2013

London, ON, Canada
July 22–25, 2013

UNIVERSITÀ DEGLI STUDI
DI MILANO

One-Tape Turing Machine

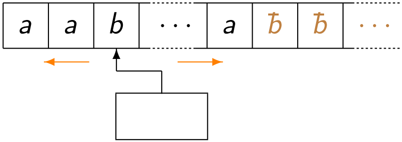


Very simple but powerful model!

Recursive enumerable languages

- No rewritings: *two-way finite automata*
  Regular languages
- Linear space:
  Context-sensitive languages [Kuroda'64]
- Linear time:
  Regular languages [Hennie'65]

One-Tape Turing Machine



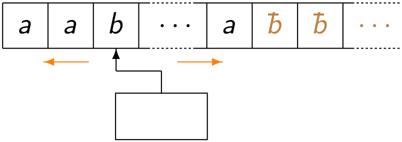
Very simple but powerful model!

Recursive enumerable languages

What about restricted versions?

- No rewritings: *two-way finite automata*
  Regular languages
- Linear space:
  Context-sensitive languages [Kuroda'64]
- Linear time:
  Regular languages [Hennie'65]

One-Tape Turing Machine



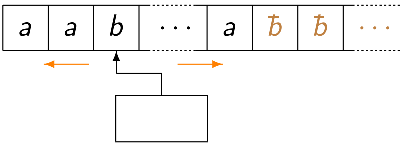
Very simple but powerful model!

Recursive enumerable languages

What about restricted versions?

- No rewritings: *two-way finite automata*
  Regular languages
- Linear space:
  Context-sensitive languages [Kuroda'64]
- Linear time:
  Regular languages [Hennie'65]

One-Tape Turing Machine



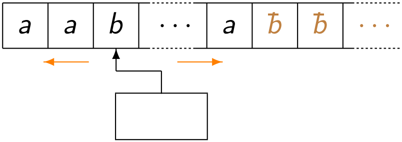
Very simple but powerful model!

Recursive enumerable languages

What about restricted versions?

- No rewritings: *two-way finite automata*
  Regular languages
- Linear space:
  Context-sensitive languages [Kuroda'64]
- Linear time:
  Regular languages [Hennie'65]

One-Tape Turing Machine



Very simple but powerful model!

Recursive enumerable languages

What about restricted versions?

- No rewritings: *two-way finite automata*
  Regular languages
- Linear space:
  Context-sensitive languages [Kuroda'64]
- Linear time:
  Regular languages [Hennie'65]

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 rewrite the content of each tape cell *only in the first d visits*

- End-marked tape
- The space is bounded by the input length
(this restriction can be removed without changing the computational power and the state upper bounds)

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 rewrite the content of each tape cell *only in the first d visits*

- End-marked tape
- The space is bounded by the input length
  (this restriction can be removed without changing the computational power and the state upper bounds)

Limited Automata [Hibbard'67]

One-tape Turing machines with restricted rewritings

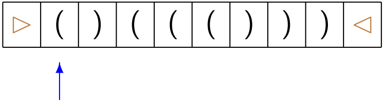
Definition

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

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

- End-marked tape
- The space is bounded by the input length
  (this restriction can be removed without changing the computational power and the state upper bounds)

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

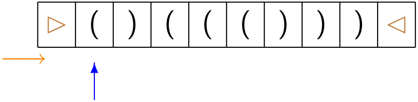
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

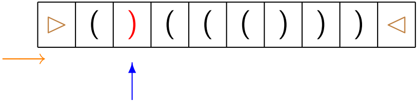
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

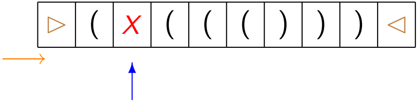
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

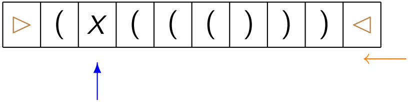
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

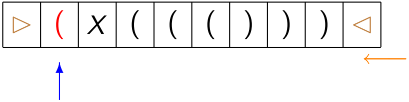
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

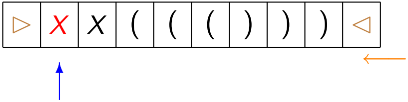
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

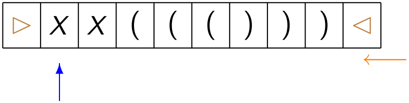
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

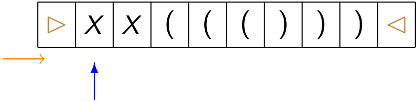
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

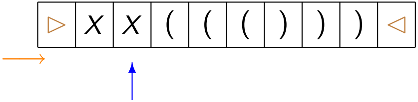
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

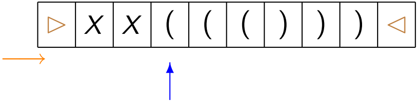
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

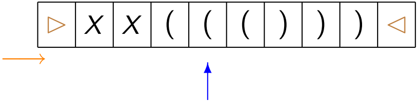
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

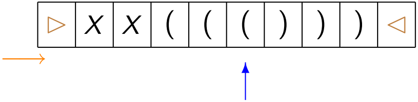
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

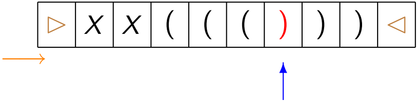
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

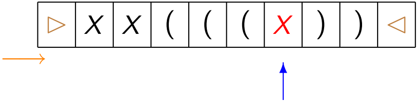
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

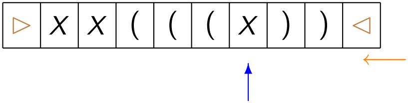
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

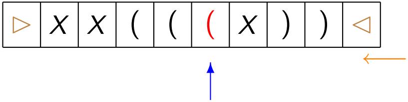
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

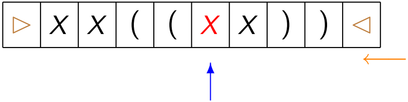
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

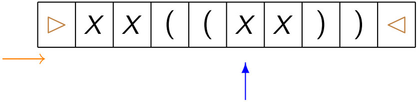
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

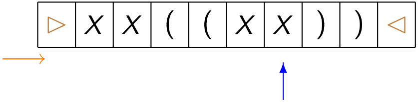
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

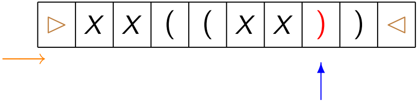
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

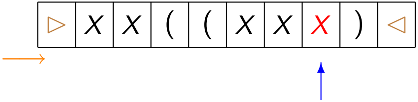
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

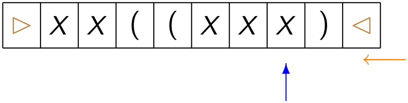
(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by X
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by X
(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

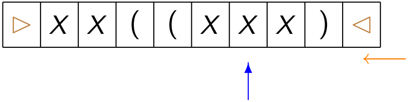
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

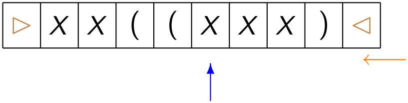
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

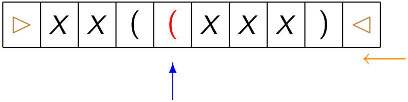
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

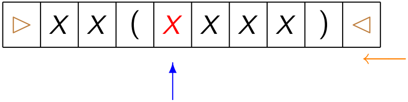
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

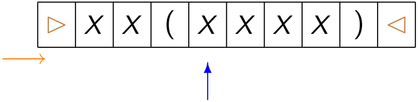
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

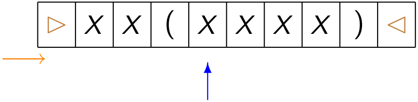
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

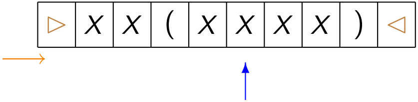
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

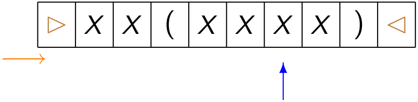
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

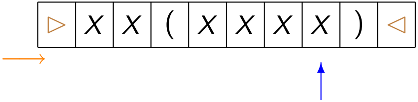
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

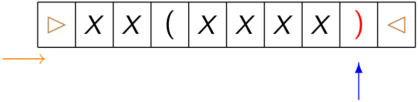
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

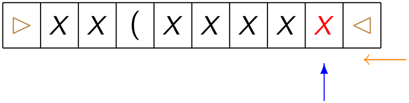
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

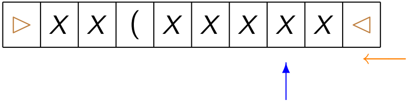
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

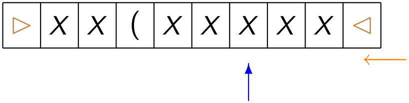
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

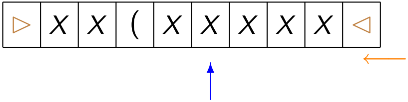
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

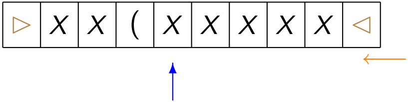
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

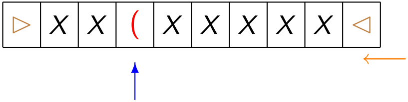
(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by X
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by X
(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

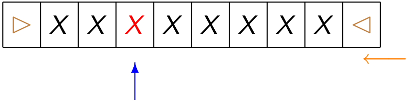
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

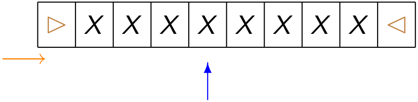
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

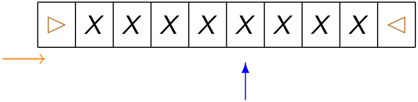
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

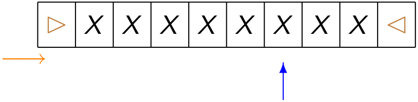
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

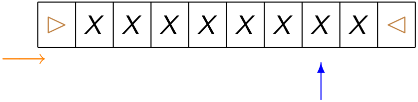
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

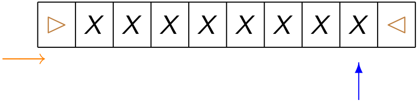
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

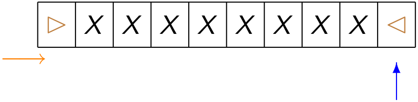
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

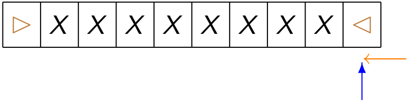
(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

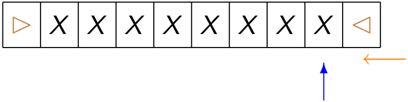
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



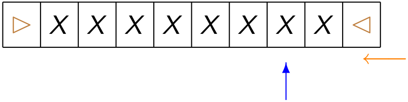
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by X
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by X
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

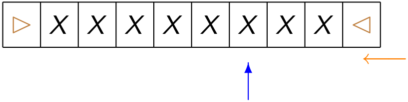
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

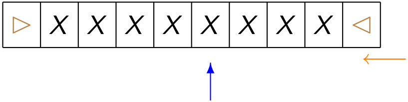
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

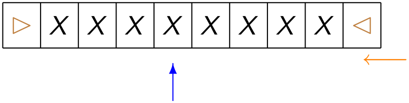
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



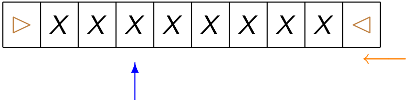
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by X
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by X
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

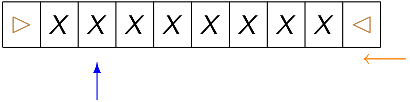
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

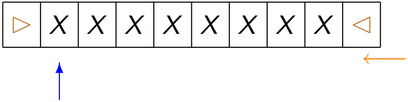
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

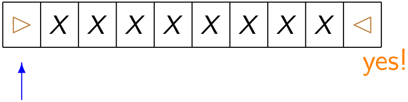
(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses



(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses

▷	X	X	X	X	X	X	X	X	◁

(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Example: Balanced Parentheses

▷	X	X	X	X	X	X	X	X	◁

(i) Move to the right to search a closed parenthesis

(ii) Rewrite it by X

(iii) Move to the left to search an open parenthesis

(iv) Rewrite it by X

(v) Repeat from the beginning

Special cases:

(i') If in (i) the right end of the tape is reached then scan all the tape and *accept* iff all tape cells contain X

(iii') If in (iii) the left end of the tape is reached then *reject*

Cells can be rewritten only in the first 2 visits!

d-Limited Automata: Computational Power

$d = 1$: regular languages [Wagner&Wechsung'86]

$d \geq 2$: context-free languages [Hibbard'67]

d-Limited Automata: Computational Power

$d = 1$: regular languages [Wagner&Wechsung'86]

$d \geq 2$: context-free languages [Hibbard'67]

d-Limited Automata: Computational Power

$d = 1$: regular languages [Wagner&Wechsung'86]

$d \geq 2$: context-free languages [Hibbard'67]

Our Contributions

$d = 1$: regular languages [Wagner&Wechsung'86]
Descriptional complexity aspects

$d \geq 2$: context-free languages [Hibbard'67]

Our Contributions

$d = 1$: regular languages [Wagner&Wechsung'86]
Descriptional complexity aspects

$d \geq 2$: context-free languages [Hibbard'67]
New transformation

context-free languages $\rightarrow$ 2-limited automata

based on the Chomsky-Schützenberger Theorem

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs:
  [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*
  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states
- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

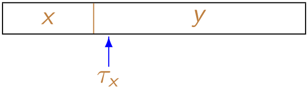
- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs: [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*
  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states
- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs: [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*
  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states
- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

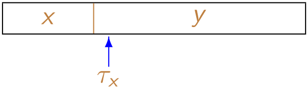
- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs: [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*



  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states
- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs: [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*



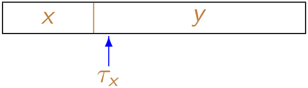
$\tau_x \subseteq Q \times Q$

$(p, q) \in \tau_x$ iff 

  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states
- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs:
  [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*



$\tau_x \subseteq Q \times Q$

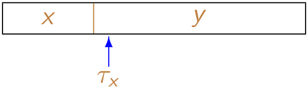
$(p, q) \in \tau_x$ iff 

  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states

- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs:
  [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*



$\tau_x \subseteq Q \times Q$

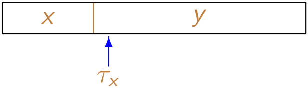
$(p, q) \in \tau_x$ iff 

  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states

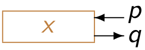
- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs:
  [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*



$\tau_x \subseteq Q \times Q$

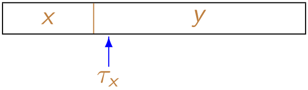
$(p, q) \in \tau_x$ iff 

  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states

- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

Simulation of 1-Limited Automata by Finite Automata

- Main idea:
  transformation of *two-way* NFAs into *one-way* DFAs: [Shepherdson'59]
  - First visit to a cell: direct simulation
  - Further visits: *transition tables*



$\tau_x \subseteq Q \times Q$

$(p, q) \in \tau_x$ iff 

  - Finite control of the simulating DFA:
    - transition table of the already scanned input prefix
    - set of possible current states

- Simulation of 1-LAs:
  - The scanned input prefix is rewritten by a *nondeterministically chosen string*
  - The simulating DFA keeps in its finite control a *sets of transition tables*

1-Limited Automata → Finite Automata: Upper Bounds

Theorem

Let M be a 1-LA *with n states.*

- *There exists an equivalent* DFA *with* $2^{n \cdot 2^{n^2}}$ *states.*
- *There exists an equivalent* NFA *with* $n \cdot 2^{n^2}$ *states.*

If M is deterministic then there exists an equivalent DFA *with no more than* $n \cdot (n+1)^n$ *states.*

	DFA	NFA
nondet. 1-LA		
det. 1-LA		

These upper bounds do not depend on the alphabet size of M!

The gaps are optimal!

1-Limited Automata $\longrightarrow$ Finite Automata: Upper Bounds

Theorem

Let M be a 1-LA *with n states.*

- *There exists an equivalent* DFA *with $2^{n \cdot 2^{n^2}}$ states.*
- *There exists an equivalent* NFA *with $n \cdot 2^{n^2}$ states.*

If M is deterministic then there exists an equivalent DFA *with no more than $n \cdot (n+1)^n$ states.*

	DFA	NFA
nondet. 1-LA	$2^{n \cdot 2^{n^2}}$	
det. 1-LA		

These upper bounds do not depend on the alphabet size of M!

The gaps are optimal!

1-Limited Automata → Finite Automata: Upper Bounds

Theorem

Let M be a 1-LA *with n states.*

- *There exists an equivalent* DFA *with $2^{n \cdot 2^{n^2}}$ states.*
- *There exists an equivalent* NFA *with $n \cdot 2^{n^2}$ states.*

If M is deterministic then there exists an equivalent DFA *with no more than $n \cdot (n+1)^n$ states.*

	DFA	NFA
nondet. 1-LA	$2^{n \cdot 2^{n^2}}$	$n \cdot 2^{n^2}$
det. 1-LA		

These upper bounds do not depend on the alphabet size of M!

The gaps are optimal!

1-Limited Automata $\longrightarrow$ Finite Automata: Upper Bounds

Theorem

Let M be a 1-LA *with n states.*

- *There exists an equivalent* DFA *with $2^{n \cdot 2^{n^2}}$ states.*
- *There exists an equivalent* NFA *with $n \cdot 2^{n^2}$ states.*

If M is deterministic then there exists an equivalent DFA *with no more than $n \cdot (n+1)^n$ states.*

	DFA	NFA
nondet. 1-LA	$2^{n \cdot 2^{n^2}}$	$n \cdot 2^{n^2}$
det. 1-LA	$n \cdot (n+1)^n$	$n \cdot (n+1)^n$

These upper bounds do not depend on the alphabet size of M!

The gaps are optimal!

1-Limited Automata $\longrightarrow$ Finite Automata: Upper Bounds

Theorem

Let M be a 1-LA with n states.

- *There exists an equivalent DFA with $2^{n \cdot 2^{n^2}}$ states.*
- *There exists an equivalent NFA with $n \cdot 2^{n^2}$ states.*

If M is deterministic then there exists an equivalent DFA with no more than $n \cdot (n+1)^n$ states.

	DFA	NFA
nondet. 1-LA	$2^{n \cdot 2^{n^2}}$	$n \cdot 2^{n^2}$
det. 1-LA	$n \cdot (n+1)^n$	$n \cdot (n+1)^n$

These upper bounds do not depend on the alphabet size of M!

The gaps are optimal!

1-Limited Automata $\longrightarrow$ Finite Automata: Upper Bounds

Theorem

Let M be a 1-LA *with n states.*

- *There exists an equivalent* DFA *with $2^{n \cdot 2^{n^2}}$ states.*
- *There exists an equivalent* NFA *with $n \cdot 2^{n^2}$ states.*

If M is deterministic then there exists an equivalent DFA *with no more than $n \cdot (n+1)^n$ states.*

	DFA	NFA
nondet. 1-LA	$2^{n \cdot 2^{n^2}}$	$n \cdot 2^{n^2}$
det. 1-LA	$n \cdot (n+1)^n$	$n \cdot (n+1)^n$

These upper bounds do not depend on the alphabet size of M!

The gaps are optimal!

Optimality: the Witness Languages

Given $n \geq 1$:

$$a_1 \quad a_2 \quad \cdots \quad a_n \quad a_{n+1}\, a_{n+2} \cdots \quad a_{2n} \quad \cdots \quad a_{\dots} \quad a_{\dots} \quad \cdots \quad a_{kn}$$

$$L_n \quad =$$

Optimality: the Witness Languages

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \; \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_2} \quad \cdots \quad \underbrace{a_{\dots} \quad a_{\dots} \quad \cdots \quad a_{kn}}_{x_k}$$

$$L_n \quad = \{x_1 x_2 \cdots x_k \mid \quad k \geq 0,\ x_1, x_2, \dots, x_k \in \{0,1\}^n,$$

Optimality: the Witness Languages

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \; \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_2} \quad \cdots \quad \underbrace{a_{\ldots} \quad a_{\ldots} \quad \cdots \quad a_{kn}}_{x_k}$$

At least n of these blocks contain the same factor

$$\begin{aligned} L_n \quad = \{x_1 x_2 \cdots x_k \mid \quad & k \geq 0,\ \ x_1, x_2, \ldots, x_k \in \{0,1\}^n, \\ & \exists i_1 < i_2 < \cdots < i_n \in \{1, \ldots, k\}, \\ & x_{i_1} = x_{i_2} = \cdots = x_{i_n} \} \end{aligned}$$

Optimality: the Witness Languages

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \ \underbrace{a_{n+1}\, a_{n+2} \ \cdots \ a_{2n}}_{x_2} \ \cdots \ \underbrace{a_{\ldots} \quad a_{\ldots} \quad \cdots \quad a_{kn}}_{x_k}$$

At least n of these blocks contain the same factor

$$\begin{aligned} L_n \quad = \{x_1 x_2 \cdots x_k \mid \quad & k \geq 0,\ \ x_1, x_2, \ldots, x_k \in \{0,1\}^n, \\ & \exists i_1 < i_2 < \cdots < i_n \in \{1, \ldots, k\}, \\ & x_{i_1} = x_{i_2} = \cdots = x_{i_n} \} \end{aligned}$$

Example ($n = 3$): 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1

Optimality: the Witness Languages

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \; \underbrace{a_{n+1} \, a_{n+2} \; \cdots \; a_{2n}}_{x_2} \; \cdots \; \underbrace{a_{\dots} \quad a_{\dots} \quad \cdots \quad a_{kn}}_{x_k}$$

At least n of these blocks contain the same factor

$$L_n = \{x_1 x_2 \cdots x_k \mid \; k \geq 0, \; x_1, x_2, \ldots, x_k \in \{0,1\}^n, \; \exists i_1 < i_2 < \cdots < i_n \in \{1, \ldots, k\}, \; x_{i_1} = x_{i_2} = \cdots = x_{i_n} \}$$

Example ($n = 3$): 0 0 1|1 1 0|0 1 1|1 1 0|1 1 0|1 1 1|0 1 1

Optimality: the Witness Languages

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \; \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_2} \quad \cdots \quad \underbrace{a_{\dots} \quad a_{\dots} \quad \cdots \quad a_{kn}}_{x_k}$$

At least n of these blocks contain the same factor

$$\begin{aligned} L_n \quad = \{x_1 x_2 \cdots x_k \mid \quad & k \geq 0,\ \ x_1, x_2, \dots, x_k \in \{0,1\}^n, \\ & \exists i_1 < i_2 < \cdots < i_n \in \{1, \dots, k\}, \\ & x_{i_1} = x_{i_2} = \cdots = x_{i_n} \} \end{aligned}$$

Example ($n = 3$): 0 0 1 | 1 1 0 | 0 1 1 | 1 1 0 | 1 1 0 | 1 1 1 | 0 1 1

How to Recognize L_n: 1-Limited Automata

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*
- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions
- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 $(n = 3)$

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*

- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions

- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

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

$\longrightarrow$

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*

- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions

- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

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

$\longleftarrow$

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*

- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions

- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

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

$\longrightarrow$

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*

- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions

- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

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

$\longrightarrow$

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*

- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions

- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

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

$\longrightarrow$

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*

- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions

- $O(n)$ states

How to Recognize L_n: 1-Limited Automata

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

- Nondeterministic strategy:
  Guess the leftmost positions of n input blocks containing the same factor and *Verify*
- Implementation:
  1. Mark n tape cells
  2. Count the tape modulo n to check whether or not:
     - the input length is a multiple of n, and
     - the marked cells correspond to the leftmost symbols of some blocks of length n
  3. Compare, symbol by symbol, each two consecutive blocks of length n that start from the marked positions
- $O(n)$ states

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

How to Recognize L_n: Deterministic Finite Automata

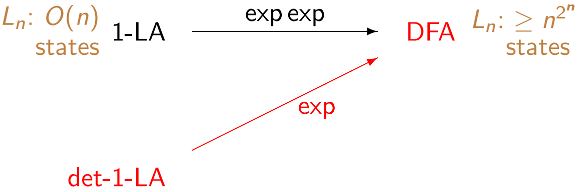
- Idea:
  - For each $x \in \{0,1\}^n$ count how many blocks coincide with x
  - Accept if and only if one of the counters reaches the value n
- State upper bound:
  - Finite control:
    a counter (up to n) for each possible block of length n
  - There are 2^n possible different blocks of length n
  - Number of states double exponential in n
    more precisely $(2^n - 1) \cdot n^{2^n} + n$
- State lower bound:
  - n^{2^n} (standard distinguishability arguments)

The state gap between 1-LAs and DFAs is double exponential!

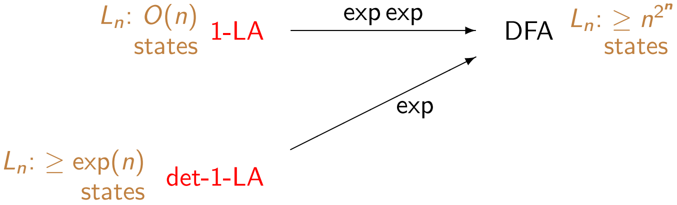
Nondetermism vs. Determinism in 1-LAs

L_n: $O(n)$ states **1-LA** $\xrightarrow{\text{exp exp}}$ **DFA** L_n: $\geq n^{2^n}$ states

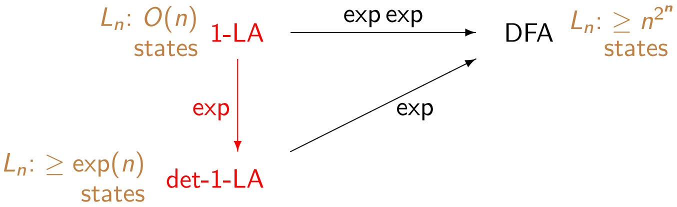
Nondetermism vs. Determinism in 1-LAs



Nondetermism vs. Determinism in 1-LAs



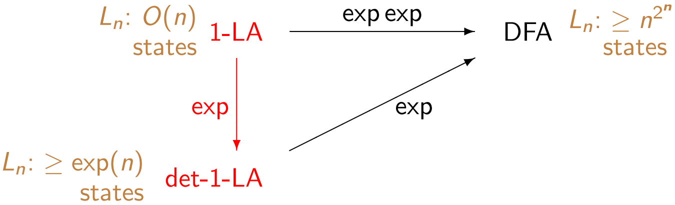
Nondetermism vs. Determinism in 1-LAs



Corollary

Removing nondeterminism from 1-LAs requires exponentially many states.

Nondetermism vs. Determinism in 1-LAs



Corollary

Removing nondeterminism from 1-LAs *requires exponentially many states.*

Cfr. Sakoda and Sipser question [Sakoda&Sipser'78]:

How much it costs in states to remove nondeterminism from two-way finite automata?

More Than One Rewriting

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

We present a construction of 2-LAs from CFLs based on:

Theorem ([Chomsky&Schützenberger'63])

Every context-free language $L \subseteq \Sigma^*$ can be expressed as

$$L = h(D_k \cap R)$$

where, for $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$:

- $D_k \subseteq \Omega_k^*$ *is a Dyck language*
- $R \subseteq \Omega_k^*$ *is a regular language*
- $h : \Omega_k \to \Sigma^*$ *is an homomorphism*

Furthermore, it is possible to restrict to *non-erasing* homomorphisms [Okhotin'12]

More Than One Rewriting

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

We present a construction of 2-LAs from CFLs based on:

Theorem ([Chomsky&Schützenberger'63])

Every context-free language $L \subseteq \Sigma^*$ can be expressed as

$$L = h(D_k \cap R)$$

where, for $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$:

- $D_k \subseteq \Omega_k^*$ *is a Dyck language*
- $R \subseteq \Omega_k^*$ *is a regular language*
- $h : \Omega_k \to \Sigma^*$ *is an homomorphism*

Furthermore, it is possible to restrict to *non-erasing* homomorphisms [Okhotin'12]

More Than One Rewriting

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

We present a construction of 2-LAs from CFLs based on:

Theorem ([Chomsky&Schützenberger'63])

Every context-free language $L \subseteq \Sigma^*$ can be expressed as

$$L = h(D_k \cap R)$$

where, for $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$*:*

- $D_k \subseteq \Omega_k^*$ *is a Dyck language*
- $R \subseteq \Omega_k^*$ *is a regular language*
- $h : \Omega_k \to \Sigma^*$ *is an homomorphism*

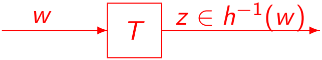
Furthermore, it is possible to restrict to *non-erasing* homomorphisms [Okhotin'12]

From CFLs to 2-LAs

L context-free language, with $L = h(D_k \cap R)$

- T nondeterministic transducer computing h^{-1}
- A_D 2-LA accepting the Dyck language D_k
- A_R finite automaton accepting R

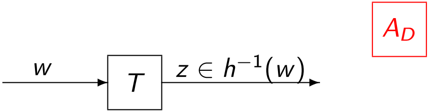
From CFLs to 2-LAs



L context-free language, with $L = h(D_k \cap R)$

- T nondeterministic transducer computing h^{-1}
- A_D 2-LA accepting the Dyck language D_k
- A_R finite automaton accepting R

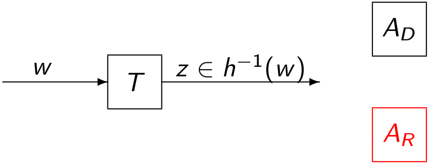
From CFLs to 2-LAs



L context-free language, with $L = h(D_k \cap R)$

- T nondeterministic transducer computing h^{-1}
- A_D 2-LA accepting the Dyck language D_k
- A_R finite automaton accepting R

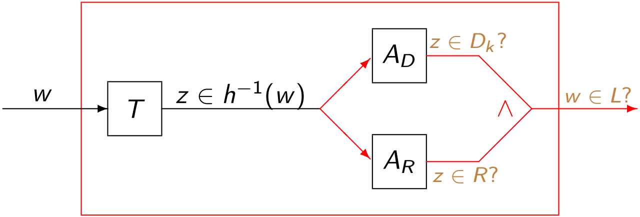
From CFLs to 2-LAs



L context-free language, with $L = h(D_k \cap R)$

- T nondeterministic transducer computing h^{-1}
- A_D 2-LA accepting the Dyck language D_k
- A_R finite automaton accepting R

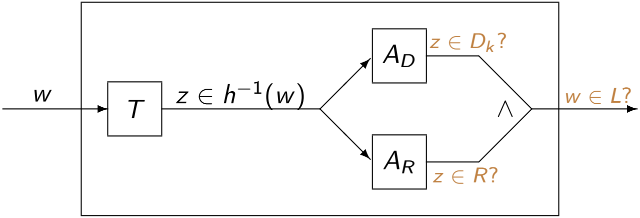
From CFLs to 2-LAs

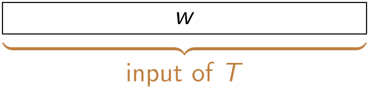


L context-free language, with $L = h(D_k \cap R)$

- T nondeterministic transducer computing h^{-1}
- A_D 2-LA accepting the Dyck language D_k
- A_R finite automaton accepting R

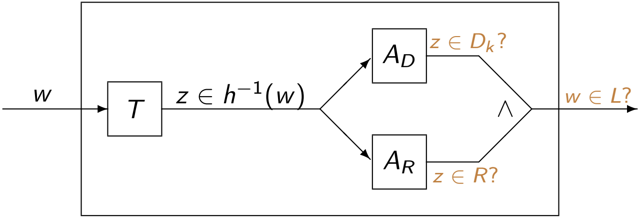
From CFLs to 2-LAs





$$z = \sigma_1 \sigma_2 \cdots \sigma_k \in h^{-1}(w)$$

From CFLs to 2-LAs



u_1	u_2	$\cdots$	u_k

input of T

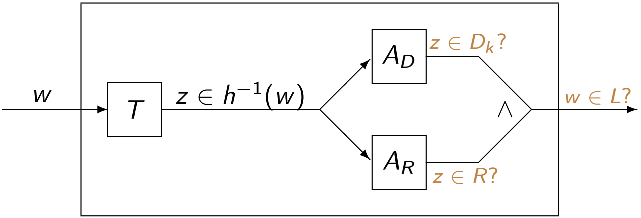
$$z = \sigma_1\sigma_2 \cdots \sigma_k \in h^{-1}(w)$$

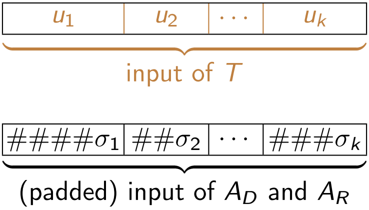
$$h(\sigma_i) = u_i$$

$\#\#\#\#\sigma_1$	$\#\#\sigma_2$	$\cdots$	$\#\#\#\sigma_k$

Non erasing homomorphism!

From CFLs to 2-LAs



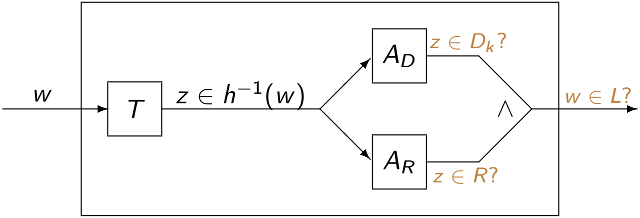


$z = \sigma_1\sigma_2\cdots\sigma_k \in h^{-1}(w)$

$h(\sigma_i) = u_i$

Non erasing homomorphism!

From CFLs to 2-LAs



u_1	u_2	$\cdots$	u_k

input of T

$z = \sigma_1\sigma_2\cdots\sigma_k \in h^{-1}(w)$

$h(\sigma_i) = u_i$

$\#\#\#\#\sigma_1$	$\#\#\sigma_2$	$\cdots$	$\#\#\#\sigma_k$

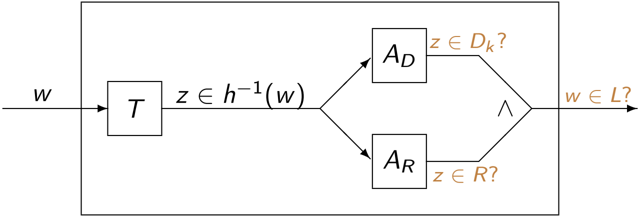
(padded) input of A_D and A_R

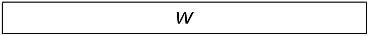
Non erasing homomorphism!

Not stored into the tape!

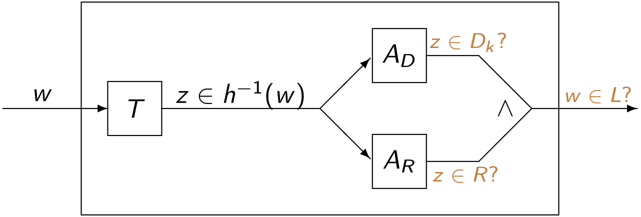
Each σ_i is produced "on the fly"

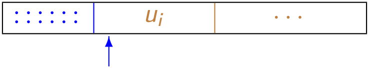
From CFLs to 2-LAs





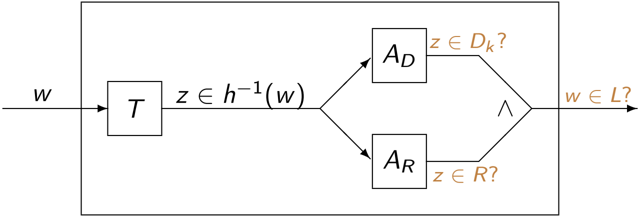
From CFLs to 2-LAs

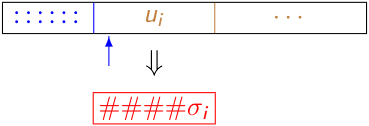




$w = \cdots u_i \cdots$

From CFLs to 2-LAs



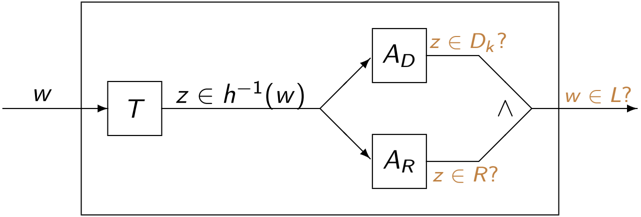


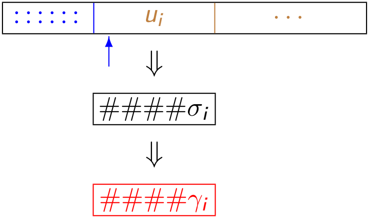
$$w = \cdots u_i \cdots$$

$$\Downarrow$$

$$h(\sigma_i) = u_i$$

From CFLs to 2-LAs





$w = \cdots u_i \cdots$

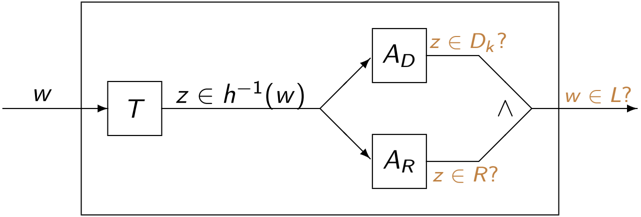
$\Downarrow$

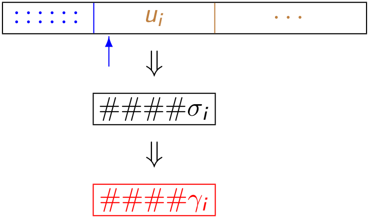
$h(\sigma_i) = u_i$

$\Downarrow$

γ_i: first rewriting by A_D

From CFLs to 2-LAs





$w = \cdots u_i \cdots$

$\Downarrow$

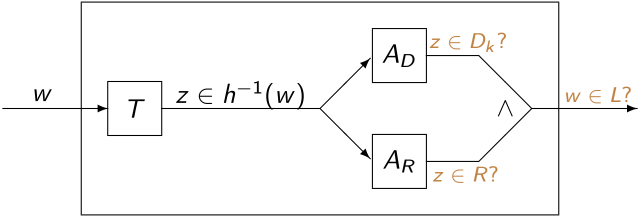
$h(\sigma_i) = u_i$

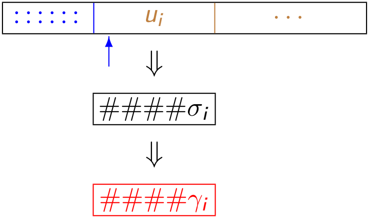
$\Downarrow$

γ_i: first rewriting by A_D

- On the tape, u_i is replaced directly by $\#\#\#\#\gamma_i$
- One move of A_R on input σ_i is also simulated

From CFLs to 2-LAs





$w = \cdots u_i \cdots$

$\Downarrow$

$h(\sigma_i) = u_i$

$\Downarrow$

γ_i: first rewriting by A_D

- On the tape, u_i is replaced directly by $\#\#\#\#\gamma_i$
- One move of A_R on input σ_i is also simulated

Final Remarks: 1-Limited Automata

- Nondeterministic 1-LAs can be
  - double exponentially smaller than one-way deterministic automata
  - exponentially smaller than one-way nondeterministic and two-way deterministic/nondeterminstic automata
- Witness languages over a two letter alphabet

What about the unary case?

Theorem

For each prime p, the language $(a^{p^2})^*$ is accepted by a deterministic 1-LAs with $p+1$ states, while it needs p^2 states to be accepted by any 2NFA.

We expect state gaps smaller than in the general case

Final Remarks: 1-Limited Automata

- Nondeterministic 1-LAs can be
  - double exponentially smaller than one-way deterministic automata
  - exponentially smaller than one-way nondeterministic and two-way deterministic/nondeterminstic automata
- Witness languages over a two letter alphabet

What about the unary case?

Theorem

For each prime p, the language $(a^{p^2})^*$ is accepted by a deterministic 1-LAs with $p+1$ states, while it needs p^2 states to be accepted by any 2NFA.

We expect state gaps smaller than in the general case

Final Remarks: 1-Limited Automata

- Nondeterministic 1-LAs can be
  - double exponentially smaller than one-way deterministic automata
  - exponentially smaller than one-way nondeterministic and two-way deterministic/nondeterminstic automata
- Witness languages over a two letter alphabet

What about the unary case?

Theorem

For each prime p, the language $(a^{p^2})^*$ is accepted by a deterministic 1-LAs with $p+1$ states, while it needs p^2 states to be accepted by any 2NFA.

We expect state gaps smaller than in the general case

Final Remarks: 1-Limited Automata

- Nondeterministic 1-LAs can be
  - double exponentially smaller than one-way deterministic automata
  - exponentially smaller than one-way nondeterministic and two-way deterministic/nondeterminstic automata
- Witness languages over a two letter alphabet

What about the unary case?

Theorem

For each prime p, the language $(a^{p^2})^*$ is accepted by a deterministic 1-LAs with $p+1$ states, while it needs p^2 states to be accepted by any 2NFA.

We expect state gaps smaller than in the general case

Final Remarks: d-Limited Automata, $d \geq 2$

- Descriptional complexity aspects
  - Case $d = 2$ [P&Pisoni NCMA2013]
  - Case $d > 2$ under investigation
- Determinism vs. nondeterminism
  - Deterministic 2-LAs characterize deterministic CFLs [P&Pisoni NCMA2013]
  - Infinite hierarchy
    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 [Hibbard'67]

Final Remarks: d-Limited Automata, $d \geq 2$

- Descriptional complexity aspects
  - Case $d = 2$ [P&Pisoni NCMA2013]
  - Case $d > 2$ under investigation
- Determinism vs. nondeterminism
  - Deterministic 2-LAs characterize deterministic CFLs
    [P&Pisoni NCMA2013]
  - Infinite hierarchy
    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
    [Hibbard'67]

Thank you for your attention!