Restricted Turing Machines and Language Recognition

Giovanni Pighizzini

Dipartimento di Informatica
Università degli Studi di Milano, Italy

LATA 2016 – Prague
March 14-18, 2016

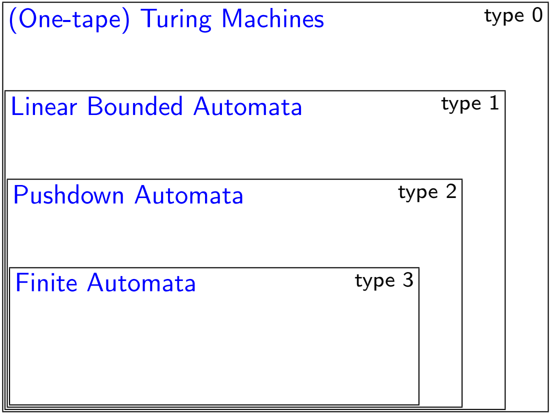
UNIVERSITÀ DEGLI STUDI
DI MILANO

General Contents

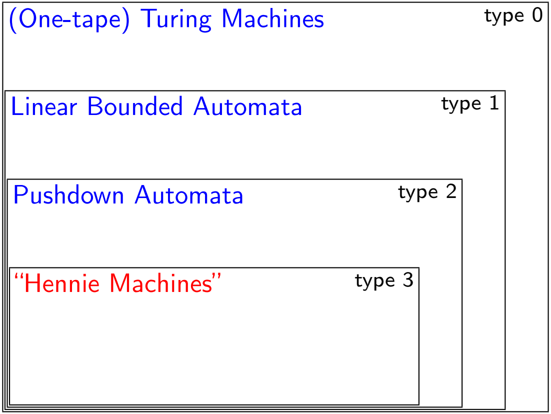
Part I: Fast One-Tape Turing Machines
Hennie Machines & C

Part II: One-Tape Turing Machines
with Rewriting Restrictions
Limited Automata & C

The Chomsky Hierarchy



The Chomsky Hierarchy



Outline

- Limited automata
- Equivalence with CFLs
- Determinism vs nondeterminism
- Descriptional complexity aspects
- 1-limited automata and regular languages
- Related models

Part II: One-Tape TMs with Rewriting Restrictions

Outline

- Limited automata
- Equivalence with CFLs
- Determinism vs nondeterminism
- Descriptional complexity aspects
- 1-limited automata and regular languages
- Related models

Part II: One-Tape TMs with Rewriting Restrictions

Outline

- Limited automata
- Equivalence with CFLs
- Determinism vs nondeterminism
- Descriptional complexity aspects
- 1-limited automata and regular languages
- Related models

Outline

- Limited automata
- Equivalence with CFLs
- Determinism vs nondeterminism
- Descriptional complexity aspects
- 1-limited automata and regular languages
- Related models

Outline

- Limited automata
- Equivalence with CFLs
- Determinism vs nondeterminism
- Descriptional complexity aspects
- 1-limited automata and regular languages
- Related models

Outline

- Limited automata
- Equivalence with CFLs
- Determinism vs nondeterminism
- Descriptional complexity aspects
- 1-limited automata and regular languages
- Related models

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*

Limited Automata [Hibbard '67]

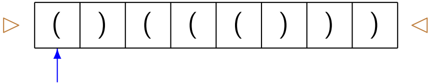
One-tape Turing machines with restricted rewritings

Definition

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

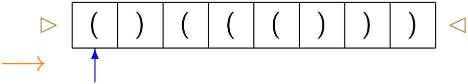
- a one-tape Turing machine
- which is allowed to rewrite the content of each tape cell *only in the first d visits*

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

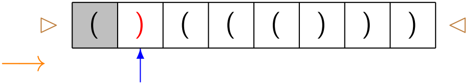
(ii) Rewrite it by x

(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

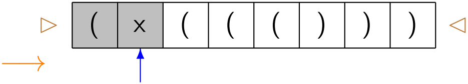
(ii) Rewrite it by x

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

(iv) Rewrite it by x

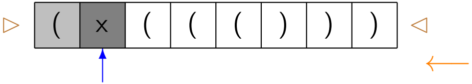
(v) Repeat from the beginning

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

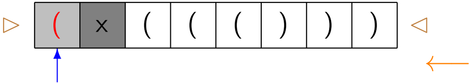
(ii) Rewrite it by x

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

(iv) Rewrite it by x

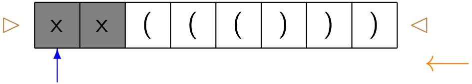
(v) Repeat from the beginning

Example: Balanced Parentheses



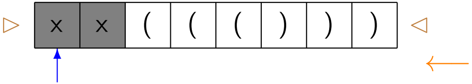
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

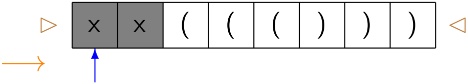
(ii) Rewrite it by x

(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

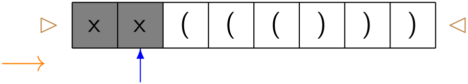
(ii) Rewrite it by x

(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

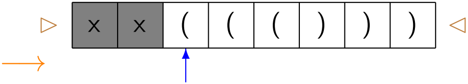
(ii) Rewrite it by x

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

(iv) Rewrite it by x

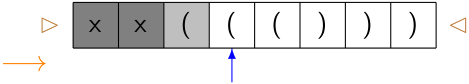
(v) Repeat from the beginning

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

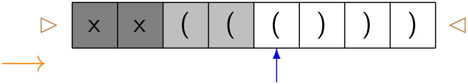
(ii) Rewrite it by x

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

(iv) Rewrite it by x

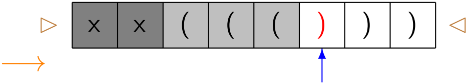
(v) Repeat from the beginning

Example: Balanced Parentheses



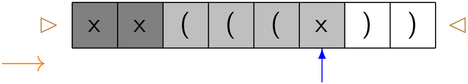
<ol type="i">
Move to the right to search a closed parenthesis
Rewrite it by x
Move to the left to search an open parenthesis
Rewrite it by x
Repeat from the beginning

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

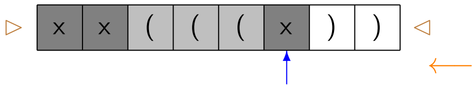
(ii) Rewrite it by x

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

(iv) Rewrite it by x

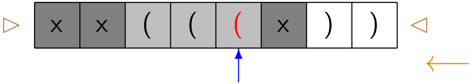
(v) Repeat from the beginning

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

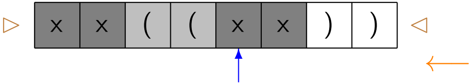
(ii) Rewrite it by x

(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

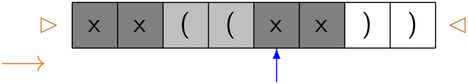
(ii) Rewrite it by x

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

(iv) Rewrite it by x

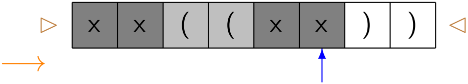
(v) Repeat from the beginning

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

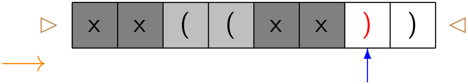
(ii) Rewrite it by x

(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

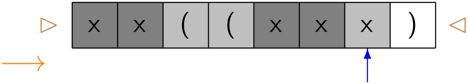
(ii) Rewrite it by x

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

(iv) Rewrite it by x

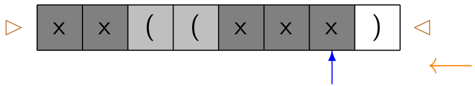
(v) Repeat from the beginning

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

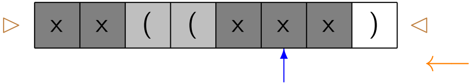
(ii) Rewrite it by x

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

(iv) Rewrite it by x

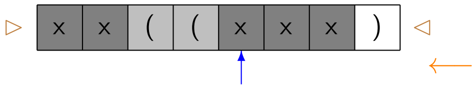
(v) Repeat from the beginning

Example: Balanced Parentheses



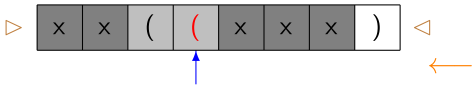
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

Example: Balanced Parentheses



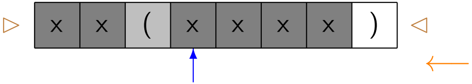
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

Example: Balanced Parentheses



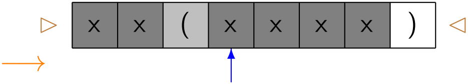
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

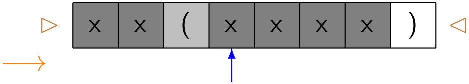
(ii) Rewrite it by x

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

(iv) Rewrite it by x

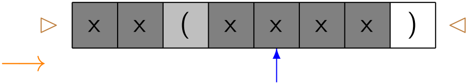
(v) Repeat from the beginning

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

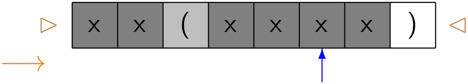
(ii) Rewrite it by x

(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

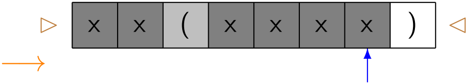
(ii) Rewrite it by x

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

(iv) Rewrite it by x

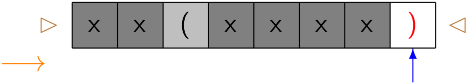
(v) Repeat from the beginning

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

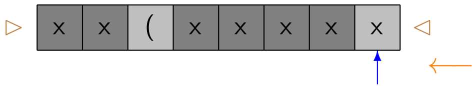
(ii) Rewrite it by x

(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

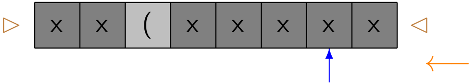
(ii) Rewrite it by x

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

(iv) Rewrite it by x

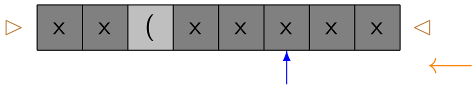
(v) Repeat from the beginning

Example: Balanced Parentheses



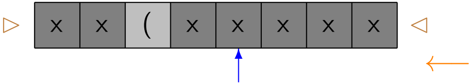
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

Example: Balanced Parentheses



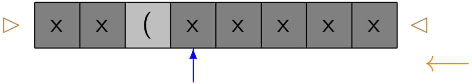
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

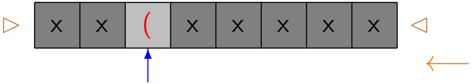
(ii) Rewrite it by x

(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

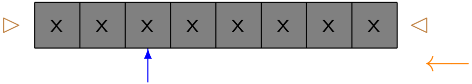
(ii) Rewrite it by x

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

(iv) Rewrite it by x

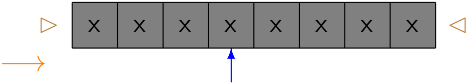
(v) Repeat from the beginning

Example: Balanced Parentheses



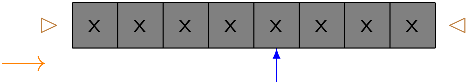
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

Example: Balanced Parentheses



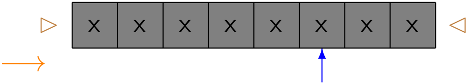
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

Example: Balanced Parentheses



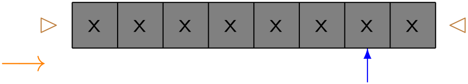
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

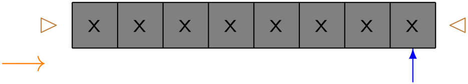
(ii) Rewrite it by x

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

(iv) Rewrite it by x

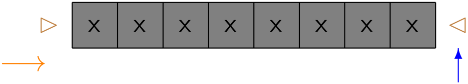
(v) Repeat from the beginning

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

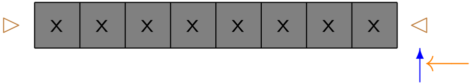
(ii) Rewrite it by x

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

(iv) Rewrite it by x

(v) Repeat from the beginning

Example: Balanced Parentheses

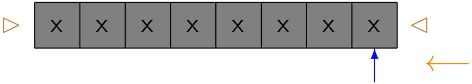


(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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



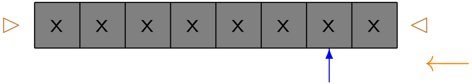
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

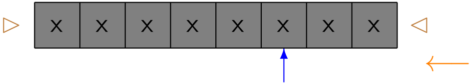


(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

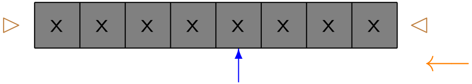
(v) Repeat from the beginning

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

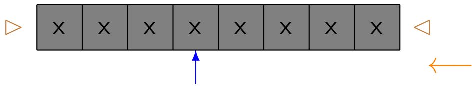
(v) Repeat from the beginning

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

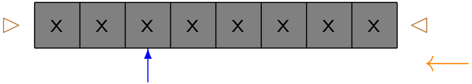
(v) Repeat from the beginning

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

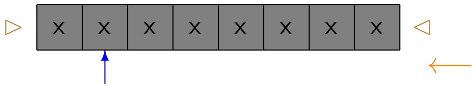
(v) Repeat from the beginning

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

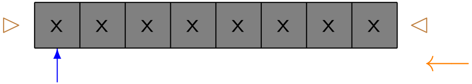


(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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



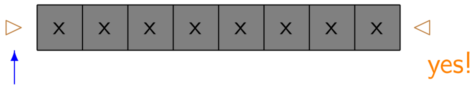
(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

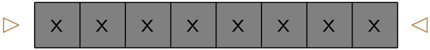


(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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

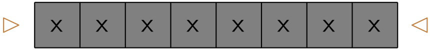


(i) Move to the right to search a closed parenthesis
(ii) Rewrite it by x
(iii) Move to the left to search an open parenthesis
(iv) Rewrite it by x
(v) Repeat from the beginning

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*

Each cell is rewritten only in the first 2 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 rewrite the content of each tape cell *only in the first d visits*

Limited Automata [Hibbard '67]

One-tape Turing machines with restricted rewritings

Definition

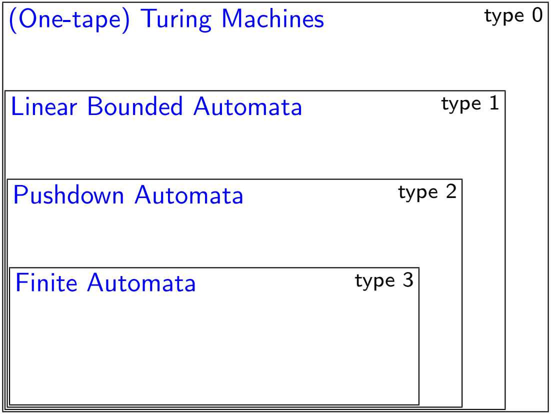
Fixed an integer $d \geq 1$, a *d-limited automaton* is

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

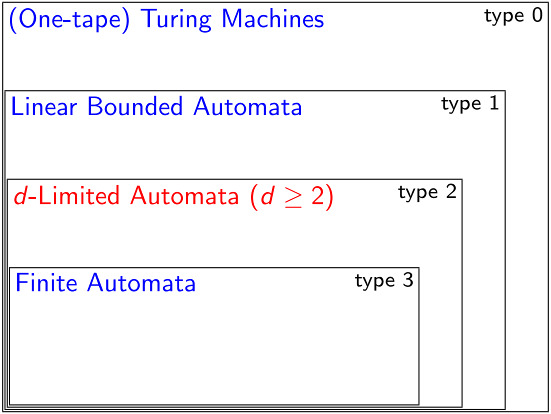
Computational power

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

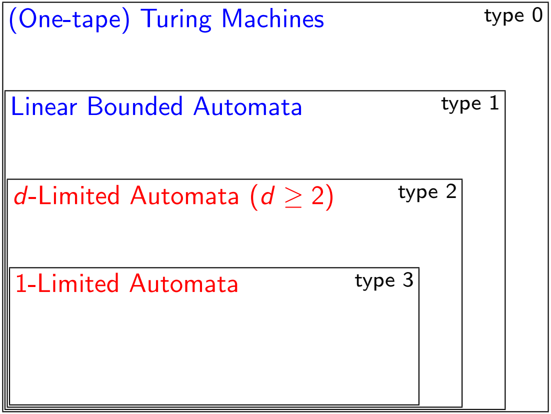
The Chomsky Hierarchy



The Chomsky Hierarchy



The Chomsky Hierarchy



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

Main tool:

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]

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

Main tool:

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*

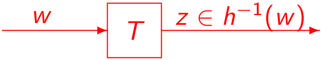
Furthermore, it is possible to restrict to *non-erasing* homomorphisms [Okhotin '12]

Why Each CFL is Accepted by a 2-LA

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

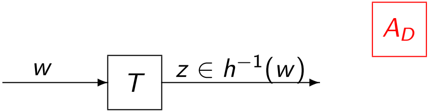
Why Each CFL is Accepted by a 2-LA



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

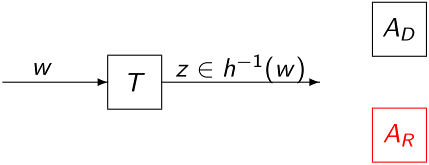
Why Each CFL is Accepted by a 2-LA



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

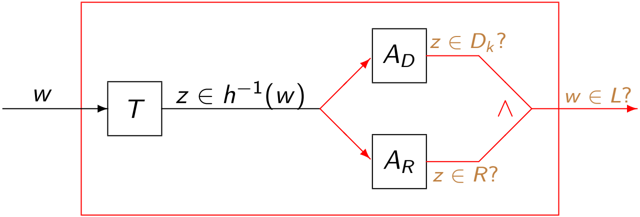
Why Each CFL is Accepted by a 2-LA



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

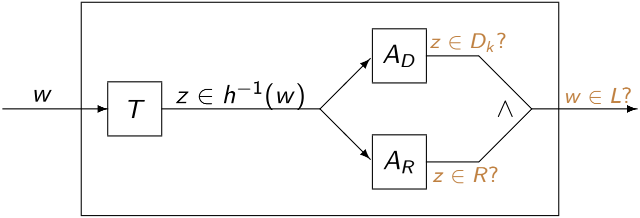
Why Each CFL is Accepted by a 2-LA

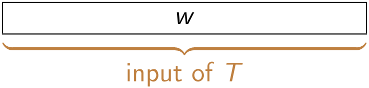


L context-free language, with $L = h(D_k \cap R)$

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

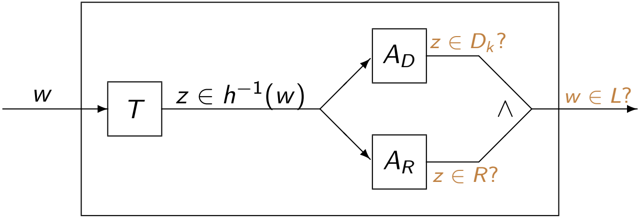
Why Each CFL is Accepted by a 2-LA

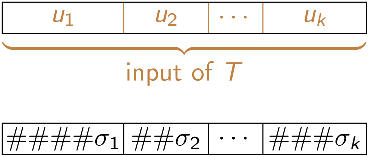




$$z = \sigma_1 \sigma_2 \cdots \sigma_k \in h^{-1}(w)$$

Why Each CFL is Accepted by a 2-LA



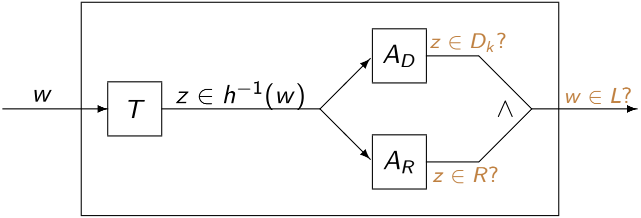


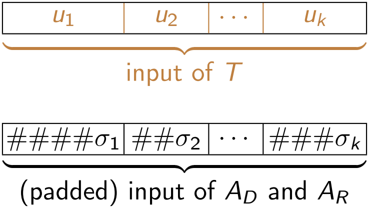
$z = \sigma_1\sigma_2\cdots\sigma_k \in h^{-1}(w)$

$h(\sigma_i) = u_i$

Non erasing homomorphism!

Why Each CFL is Accepted by a 2-LA



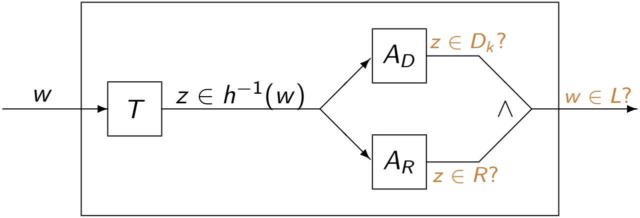


$z = \sigma_1 \sigma_2 \cdots \sigma_k \in h^{-1}(w)$

$h(\sigma_i) = u_i$

Non erasing homomorphism!

Why Each CFL is Accepted by a 2-LA



u_1	u_2	$\cdots$	u_k

input of T

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

$$h(\sigma_i) = u_i$$

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

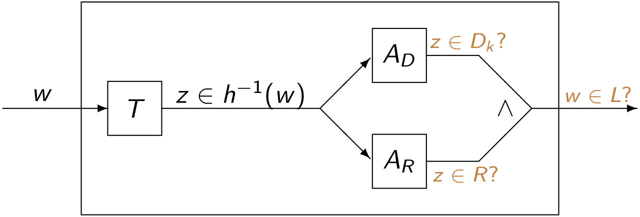
(padded) input of A_D and A_R

Non erasing homomorphism!

Not stored into the tape!

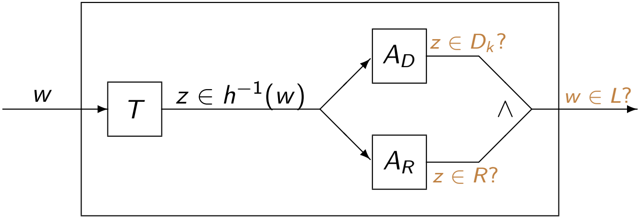
Each σ_i is produced "on the fly"

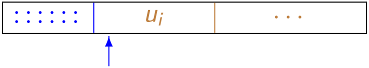
Why Each CFL is Accepted by a 2-LA



w

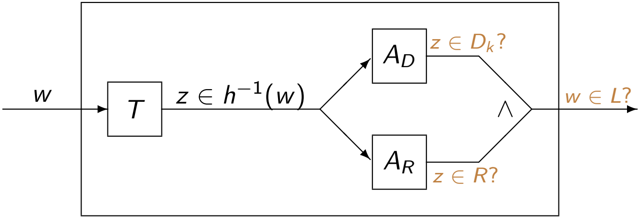
Why Each CFL is Accepted by a 2-LA

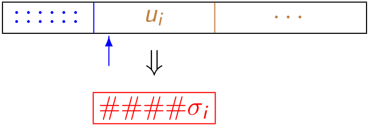




$w = \cdots u_i \cdots$

Why Each CFL is Accepted by a 2-LA



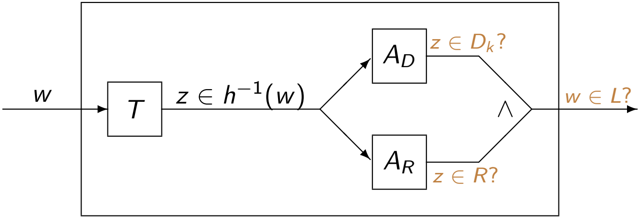


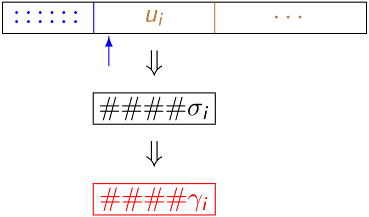
$$w = \cdots u_i \cdots$$

$$\Downarrow$$

$$h(\sigma_i) = u_i$$

Why Each CFL is Accepted by a 2-LA





$w = \cdots u_i \cdots$

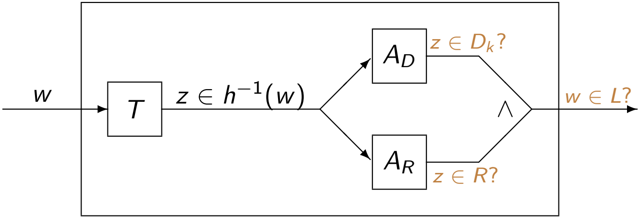
$\Downarrow$

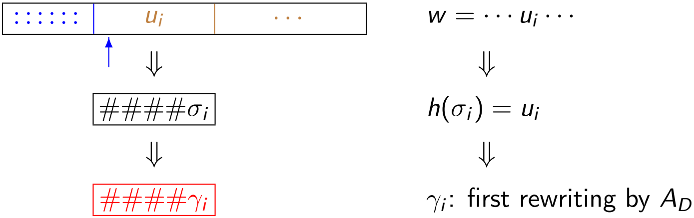
$h(\sigma_i) = u_i$

$\Downarrow$

γ_i: first rewriting by A_D

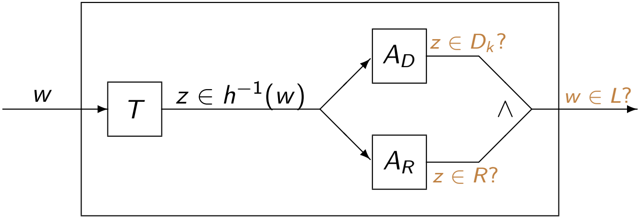
Why Each CFL is Accepted by a 2-LA

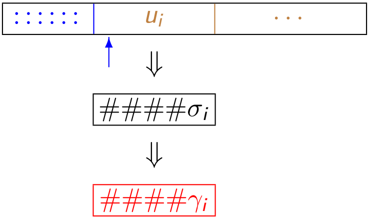




- On the tape, u_i is replaced directly by $\#\#\#\#\gamma_i$
- One move of A_R on input σ_i is also simulated

Why Each CFL is Accepted by a 2-LA





$w = \cdots u_i \cdots$

$\Downarrow$

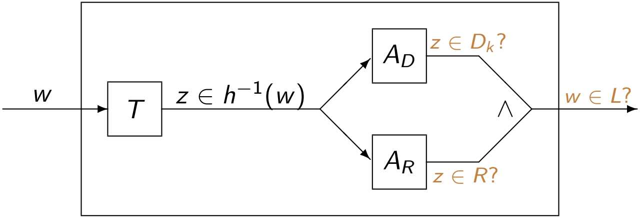
$h(\sigma_i) = u_i$

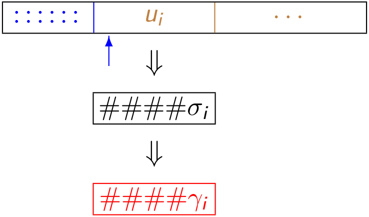
$\Downarrow$

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

Why Each CFL is Accepted by a 2-LA





$w = \cdots u_i \cdots$

$\Downarrow$

$h(\sigma_i) = u_i$

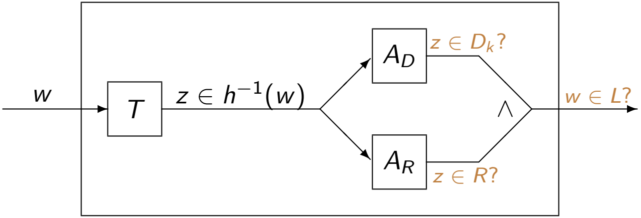
$\Downarrow$

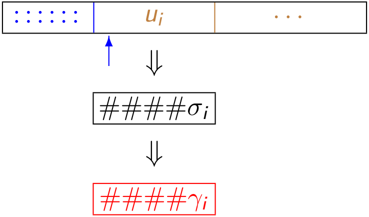
γ_i: first rewriting by A_D

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

The resulting machine is a 2-LA recognizing the given CFL

Why Each CFL is Accepted by a 2-LA





$w = \cdots u_i \cdots$

$\Downarrow$

$h(\sigma_i) = u_i$

$\Downarrow$

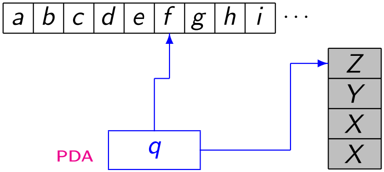
γ_i: first rewriting by A_D

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

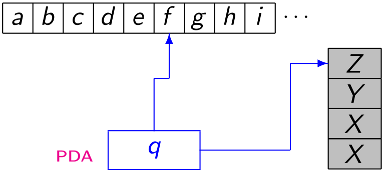
The resulting machine is a 2-LA recognizing the given CFL

PDAs vs Limited Automata

Simulation of Pushdown Automata by 2-Limited Automata



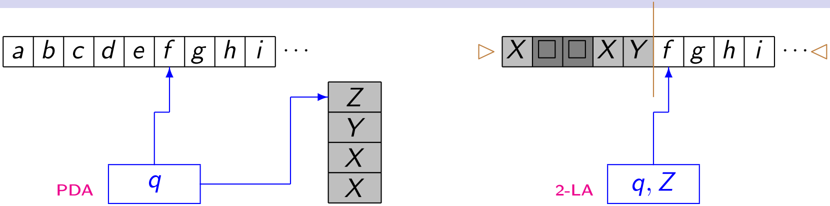
Simulation of Pushdown Automata by 2-Limited Automata



Normal form for (D)PDAs:

- at each step, the stack height increases at most by 1
- ϵ-moves cannot push on the stack

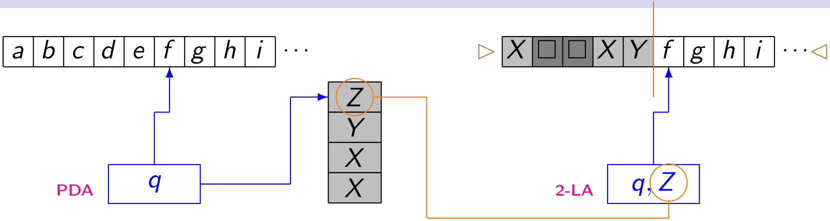
Simulation of Pushdown Automata by 2-Limited Automata



Normal form for (D)PDAs:

- at each step, the stack height increases at most by 1
- ϵ-moves cannot push on the stack

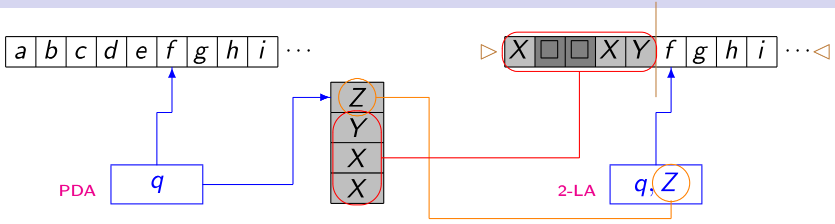
Simulation of Pushdown Automata by 2-Limited Automata



Normal form for (D)PDAs:

- at each step, the stack height increases at most by 1
- ϵ-moves cannot push on the stack

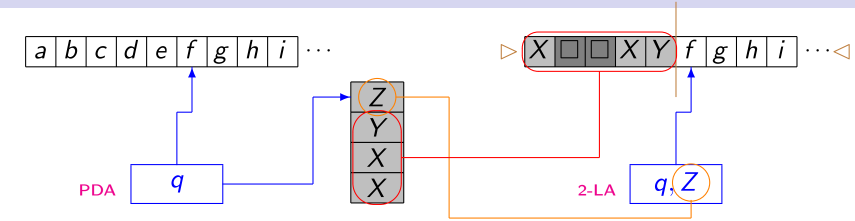
Simulation of Pushdown Automata by 2-Limited Automata



Normal form for (D)PDAs:

- at each step, the stack height increases at most by 1
- ϵ-moves cannot push on the stack

Simulation of Pushdown Automata by 2-Limited Automata



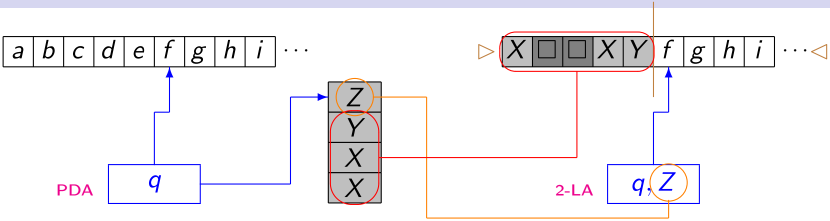
Normal form for (D)PDAs:

- at each step, the stack height increases at most by 1
- ϵ-moves cannot push on the stack

Each PDA can be simulated by an equivalent 2-LA

- Polynomial size
- Determinism is preserved

Simulation of Pushdown Automata by 2-Limited Automata



Normal form for (D)PDAs:

- at each step, the stack height increases at most by 1
- ϵ-moves cannot push on the stack

Each PDA can be simulated by an equivalent 2-LA

- Polynomial size
- Determinism is preserved

Simulation of 2-Limited Automata by Pushdown Automata

Problem

What about the converse simulation, namely that of 2-LAs *by PDAs?*

[Hibbard '67]

Original simulation

[P.&Pisoni '15]

Reformulation

- Exponential cost
- Determinism is preserved (extra costs)

Simulation of 2-Limited Automata by Pushdown Automata

Problem

What about the converse simulation, namely that of 2-LAs by PDAs?

[Hibbard '67]

Original simulation

[P.&Pisoni '15]

Reformulation

- Exponential cost
- Determinism is preserved (extra costs)

Simulation of 2-Limited Automata by Pushdown Automata

Problem

What about the converse simulation, namely that of 2-LAs by PDAs?

[Hibbard '67]

Original simulation

[P.&Pisoni '15]

Reformulation

- Exponential cost
- Determinism is preserved (extra costs)

Transition Tables of 2-LAs

- Fixed a 2-limited automaton
- *Transition table* τ_w $\quad$ w is a "frozen" string

$$\tau_w \subseteq Q \times \{-1, +1\} \times Q \times \{-1, +1\}$$

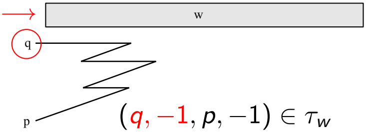
$(q, d', p, d'') \in \tau_w$ iff M on a tape segment containing w has a computation path:

- entering the segment in q from d'
- exiting the segment in p to d''
- left $= -1$, right $= +1$

Transition Tables of 2-LAs

- Fixed a 2-limited automaton
- *Transition table* τ_w $\qquad$ w is a "frozen" string

$$\tau_w \subseteq Q \times \{-1, +1\} \times Q \times \{-1, +1\}$$



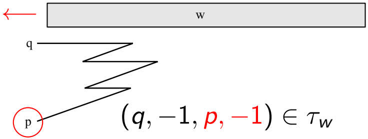
$(q, d', p, d'') \in \tau_w$ iff M on a tape segment containing w has a computation path:

- entering the segment in q from d'
- exiting the segment in p to d''
- left $= -1$, right $= +1$

Transition Tables of 2-LAs

- Fixed a 2-limited automaton
- *Transition table* τ_w $\qquad$ w is a "frozen" string

$$\tau_w \subseteq Q \times \{-1,+1\} \times Q \times \{-1,+1\}$$



$(q, -1, p, -1) \in \tau_w$

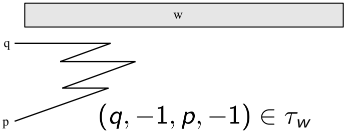
$(q, d', p, d'') \in \tau_w$ iff M on a tape segment containing w has a computation path:

- entering the segment in q from d'
- exiting the segment in p to d''
- left $= -1$, right $= +1$

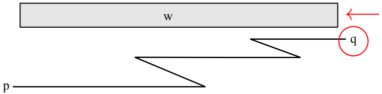
Transition Tables of 2-LAs

- Fixed a 2-limited automaton
- *Transition table* τ_w — w is a “frozen” string

$$\tau_w \subseteq Q \times \{-1, +1\} \times Q \times \{-1, +1\}$$



$(q, -1, p, -1) \in \tau_w$



$(q, +1, p, -1) \in \tau_w$

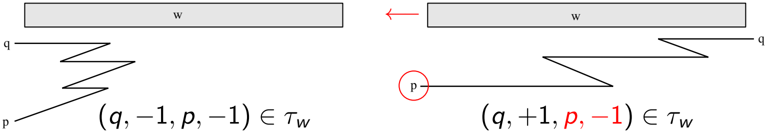
$(q, d', p, d'') \in \tau_w$ iff M on a tape segment containing w has a computation path:

- entering the segment in q from d'
- exiting the segment in p to d''
- left $= -1$, right $= +1$

Transition Tables of 2-LAs

- Fixed a 2-limited automaton
- *Transition table* τ_w $\qquad$ w is a "frozen" string

$$\tau_w \subseteq Q \times \{-1,+1\} \times Q \times \{-1,+1\}$$



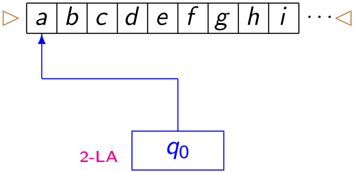
$(q,-1,p,-1) \in \tau_w$ $\qquad$ $(q,+1,p,-1) \in \tau_w$

$(q,d',p,d'') \in \tau_w$ iff M on a tape segment containing w has a computation path:

- entering the segment in q from d'
- exiting the segment in p to d''
- left $=-1$, right $=+1$

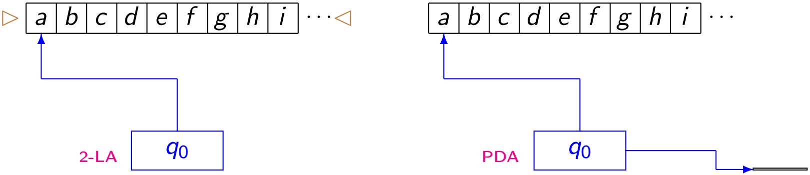
Simulation of 2-LAs by PDAs

Initial configuration



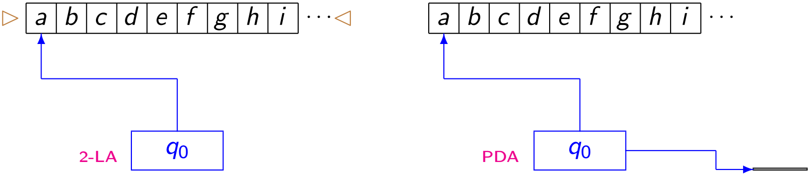
Simulation of 2-LAs by PDAs

Initial configuration

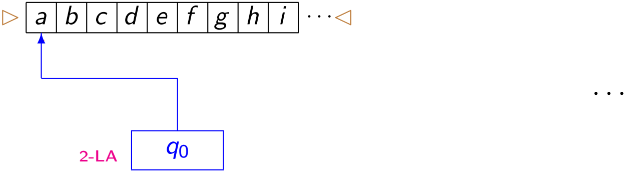


Simulation of 2-LAs by PDAs

Initial configuration

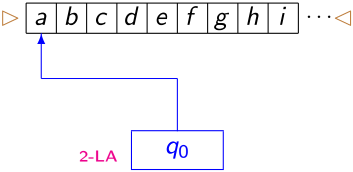


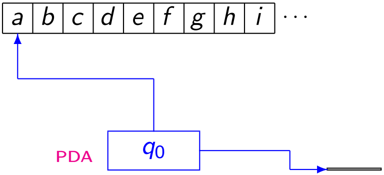
Some computation steps...



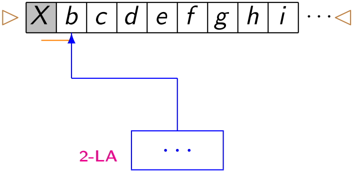
Simulation of 2-LAs by PDAs

Initial configuration





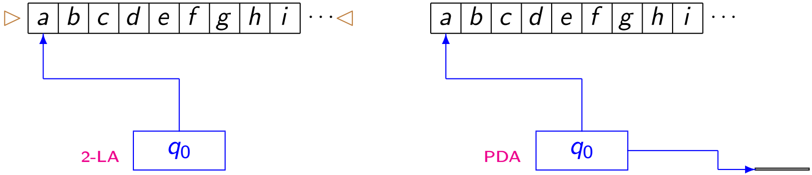
Some computation steps...



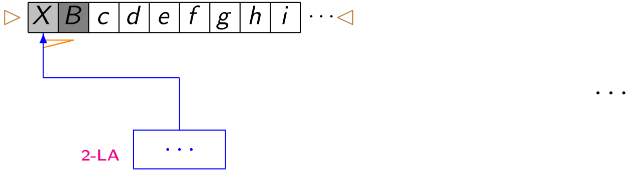
...

Simulation of 2-LAs by PDAs

Initial configuration

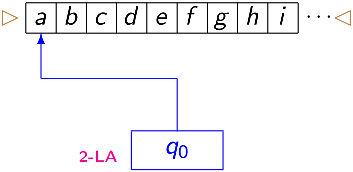


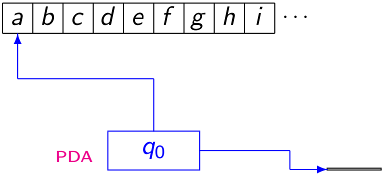
Some computation steps...



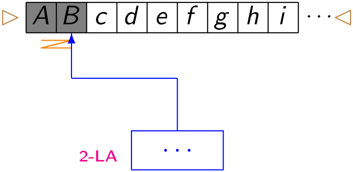
Simulation of 2-LAs by PDAs

Initial configuration





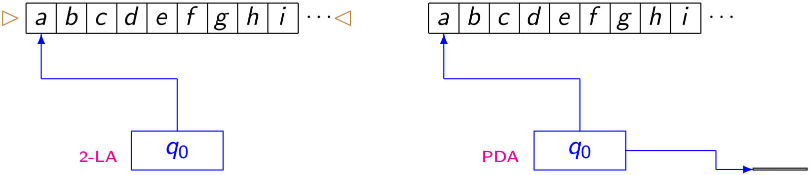
Some computation steps...



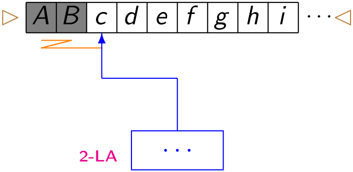
...

Simulation of 2-LAs by PDAs

Initial configuration



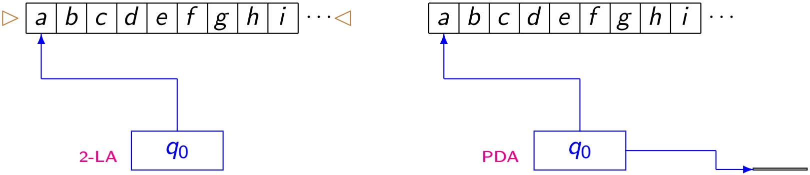
Some computation steps...



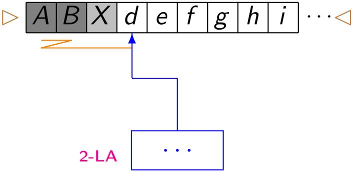
...

Simulation of 2-LAs by PDAs

Initial configuration



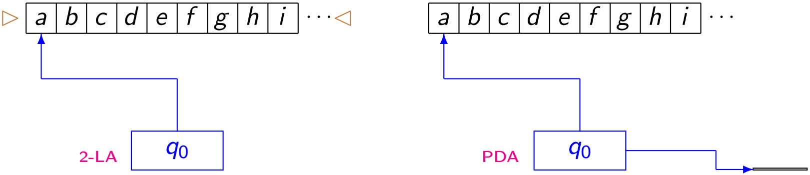
Some computation steps...



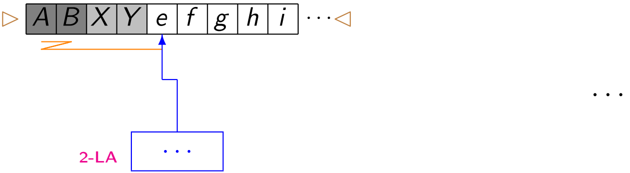
...

Simulation of 2-LAs by PDAs

Initial configuration

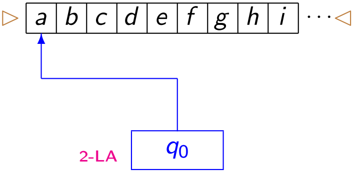


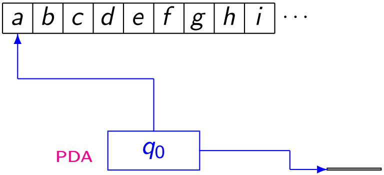
Some computation steps...



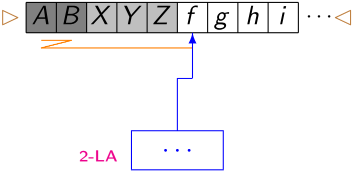
Simulation of 2-LAs by PDAs

Initial configuration





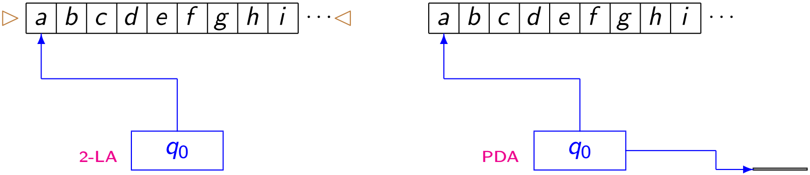
Some computation steps...



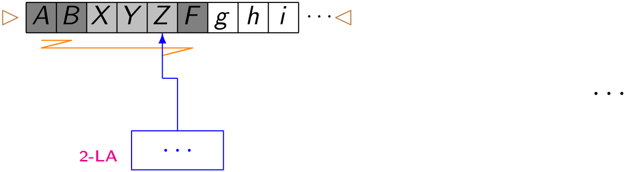
...

Simulation of 2-LAs by PDAs

Initial configuration

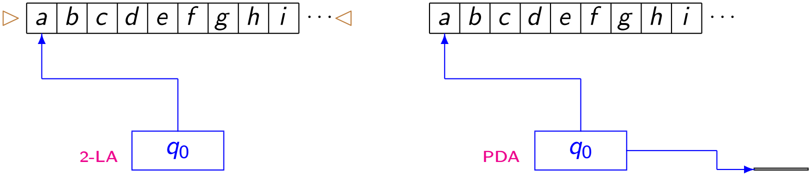


Some computation steps...

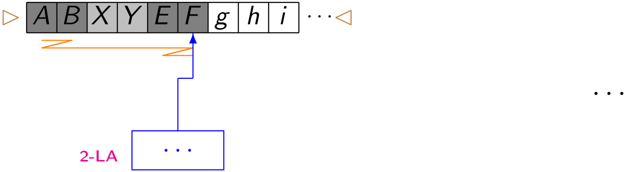


Simulation of 2-LAs by PDAs

Initial configuration

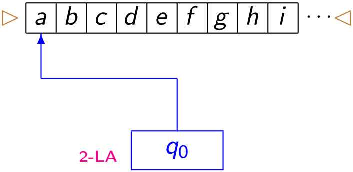


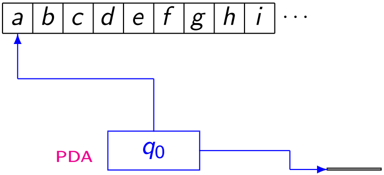
Some computation steps...



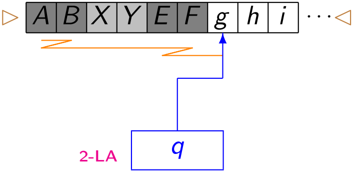
Simulation of 2-LAs by PDAs

Initial configuration





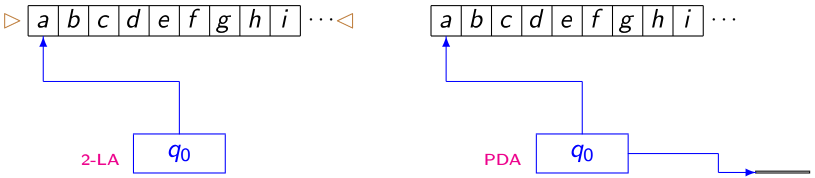
Some computation steps...



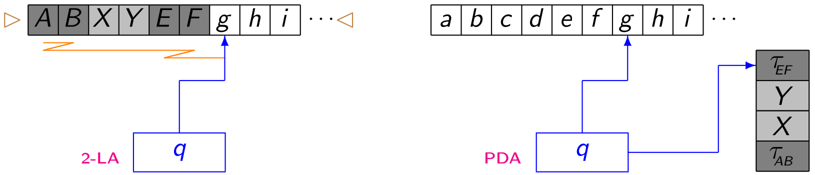
...

Simulation of 2-LAs by PDAs

Initial configuration

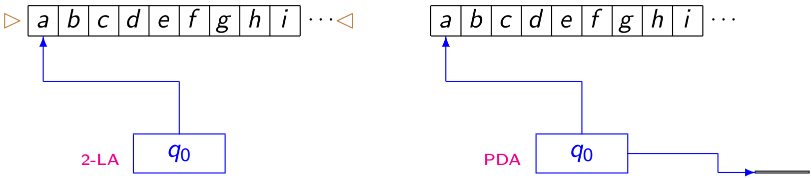


After some steps...

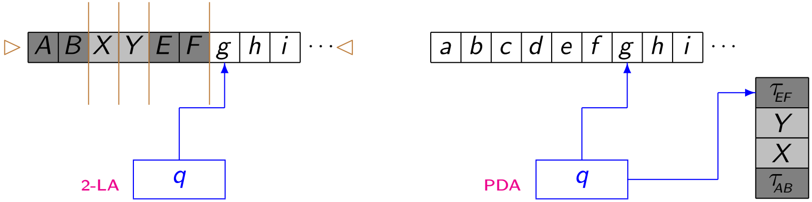


Simulation of 2-LAs by PDAs

Initial configuration

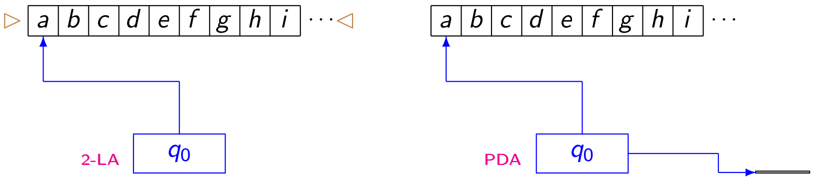


After some steps...

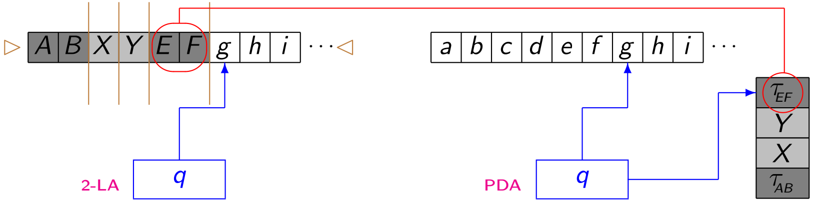


Simulation of 2-LAs by PDAs

Initial configuration

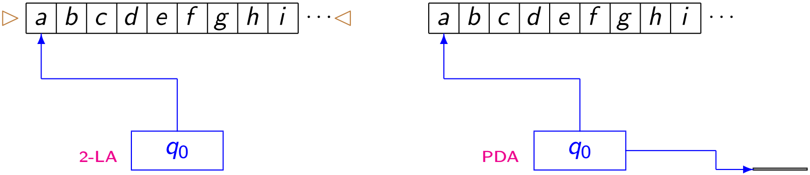


After some steps...

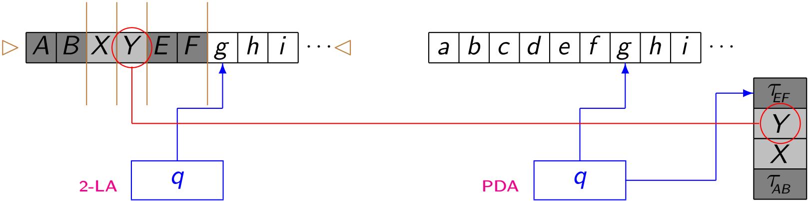


Simulation of 2-LAs by PDAs

Initial configuration

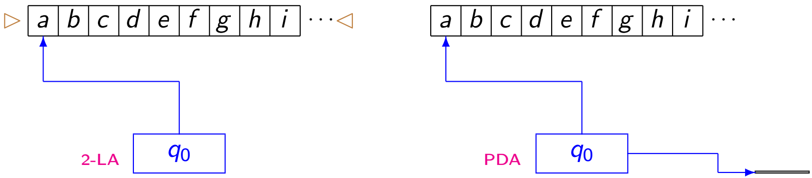


After some steps...

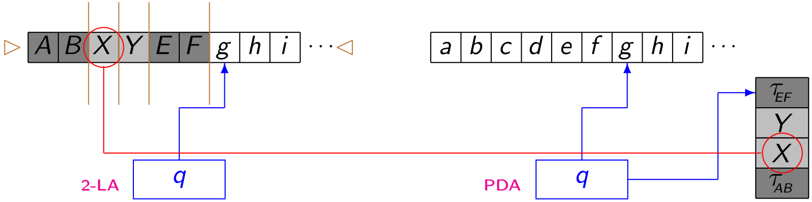


Simulation of 2-LAs by PDAs

Initial configuration

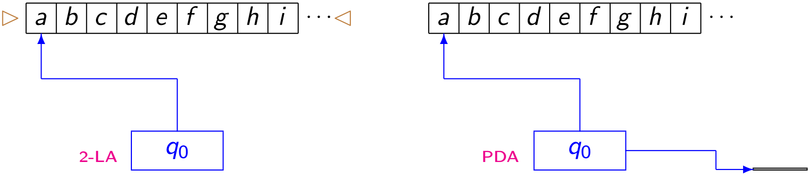


After some steps...

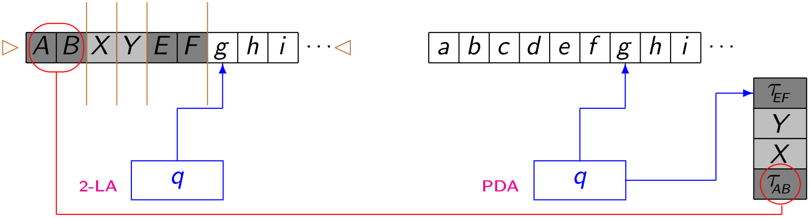


Simulation of 2-LAs by PDAs

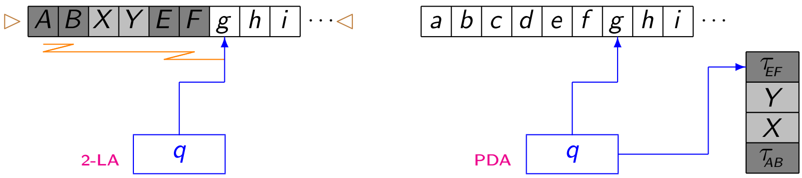
Initial configuration



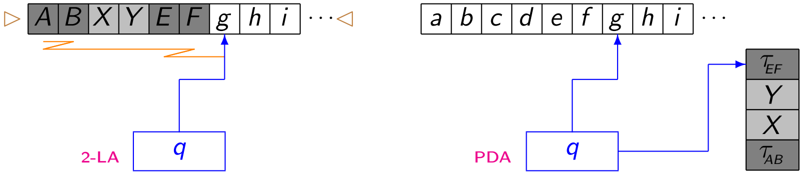
After some steps...



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs

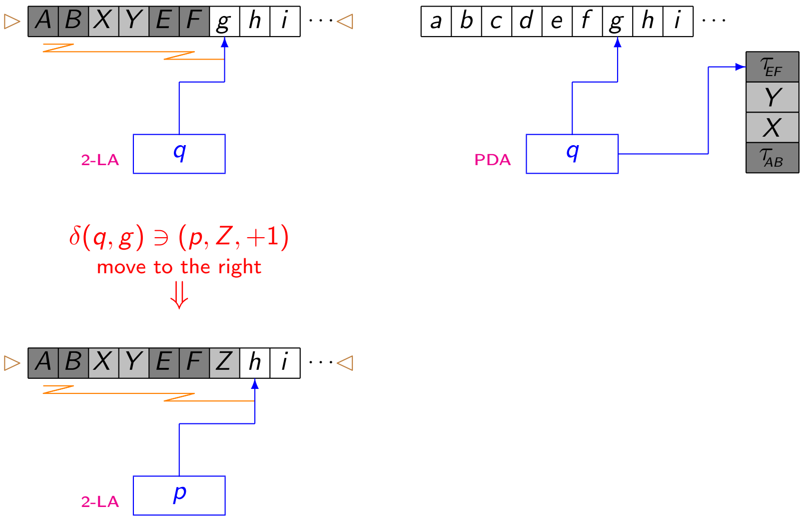


$\delta(q, g) \ni (p, Z, +1)$

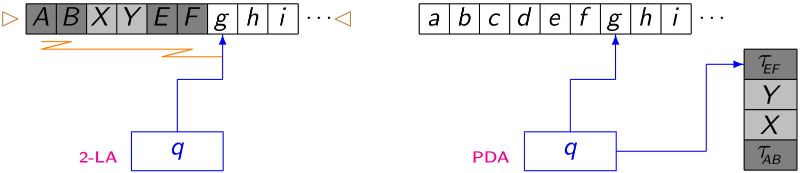
move to the right

$\Downarrow$

Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



$\delta(q, g) \ni (p, Z, +1)$

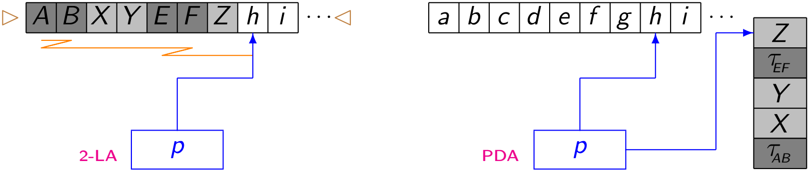
move to the right

$\Downarrow$

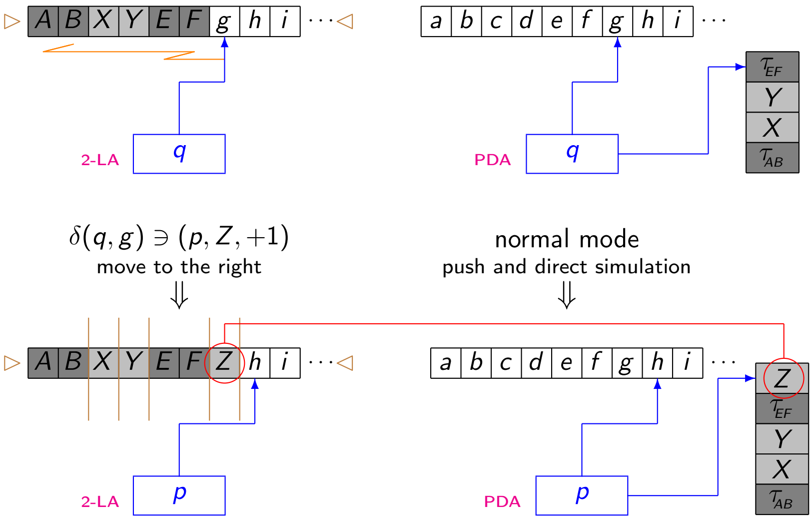
normal mode

push and direct simulation

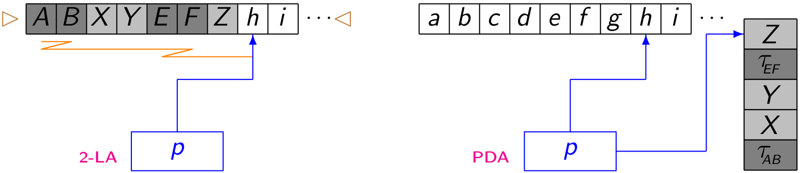
$\Downarrow$



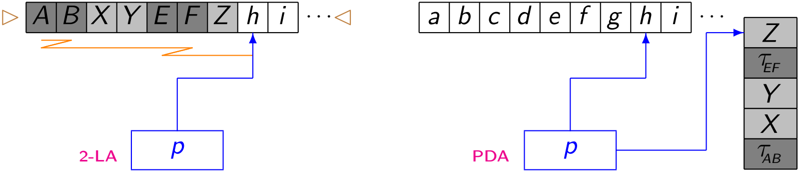
Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs

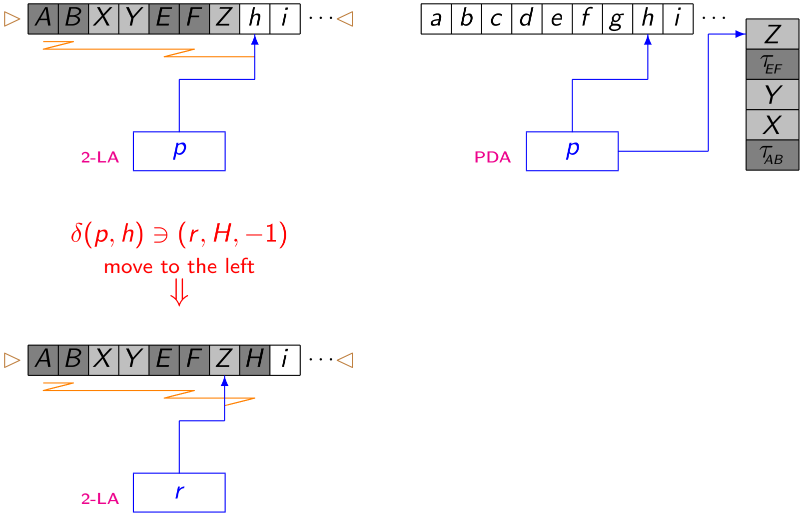


$\delta(p, h) \ni (r, H, -1)$

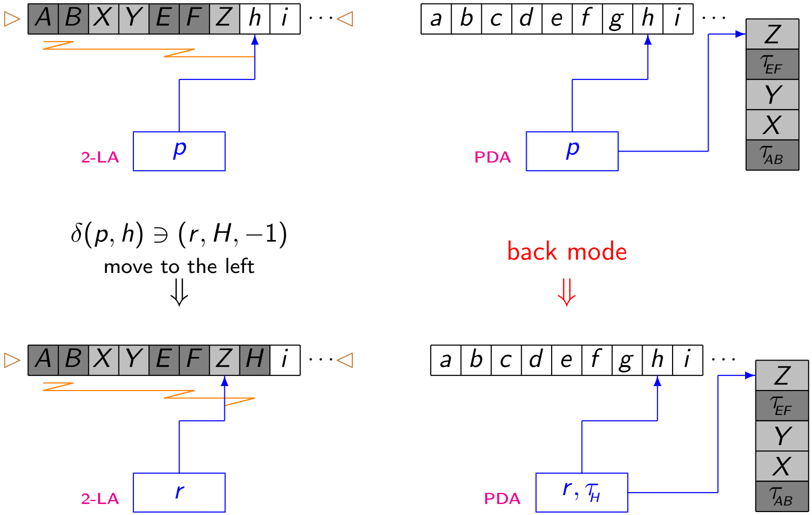
move to the left

$\Downarrow$

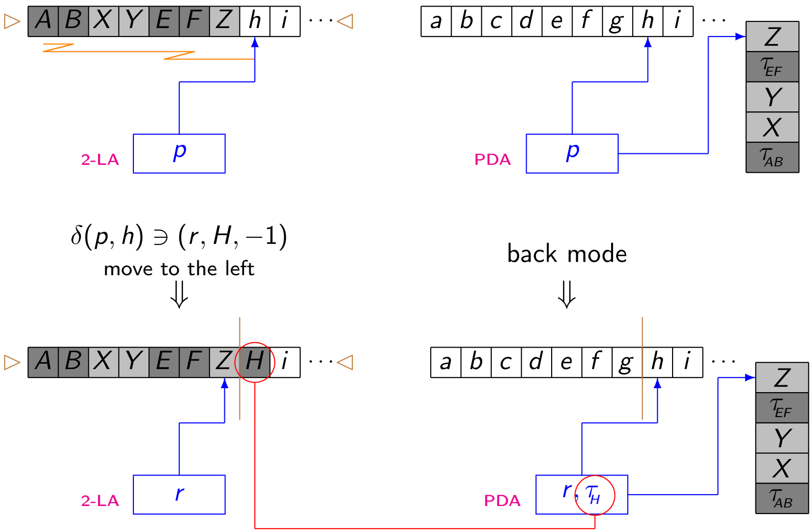
Simulation of 2-LAs by PDAs



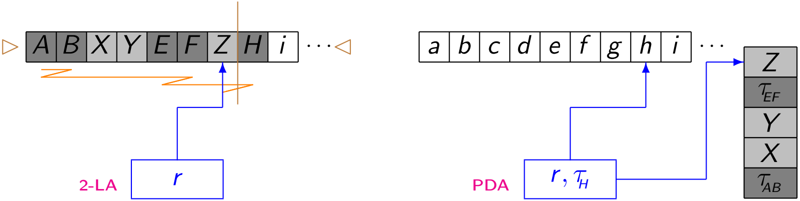
Simulation of 2-LAs by PDAs



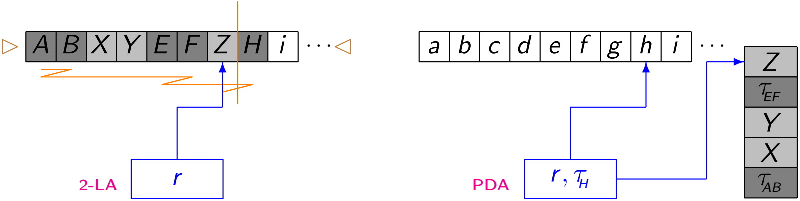
Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs

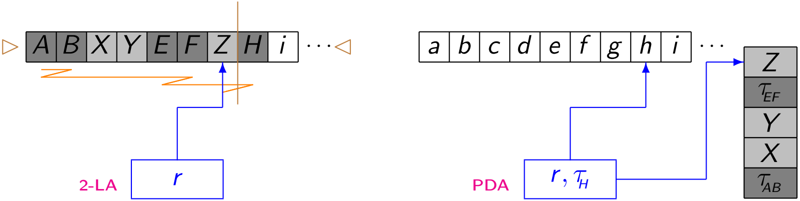


$\delta(r, Z) \ni (q, G, -1)$

move to the left

$\Downarrow$

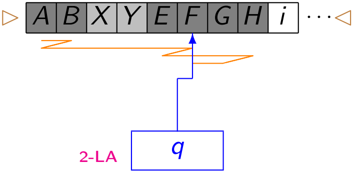
Simulation of 2-LAs by PDAs



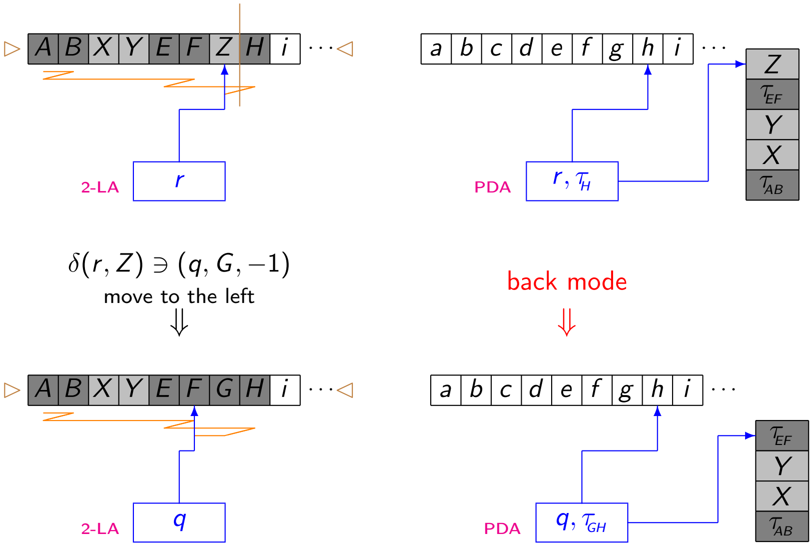
$\delta(r, Z) \ni (q, G, -1)$

move to the left

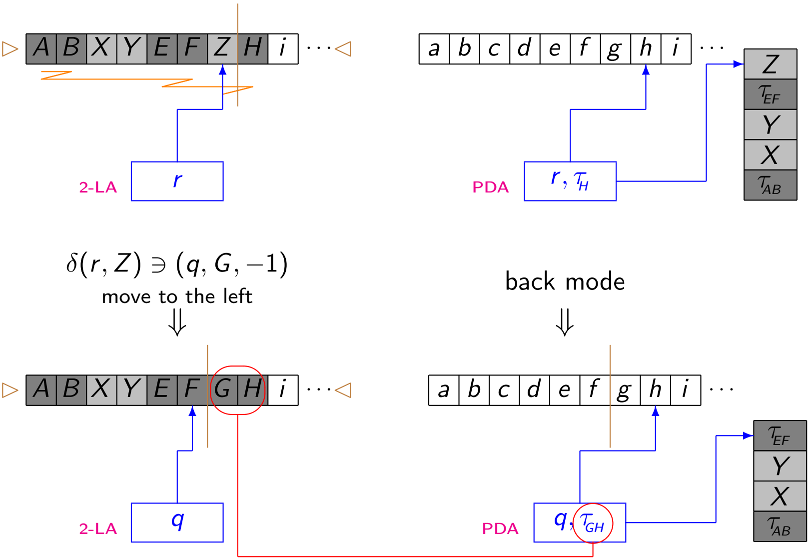
$\Downarrow$



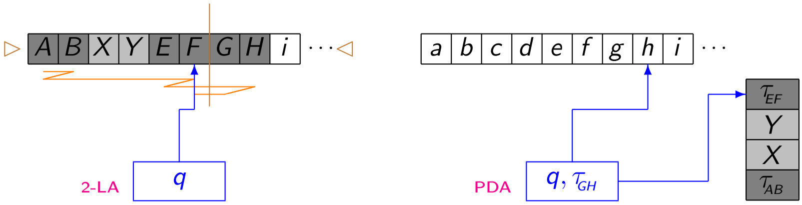
Simulation of 2-LAs by PDAs



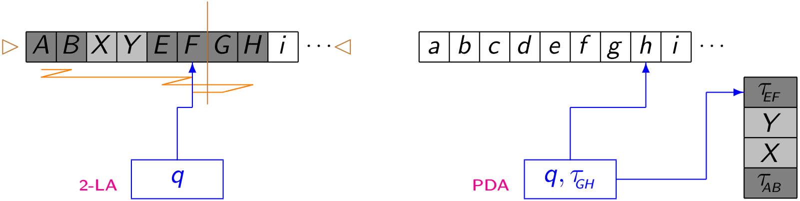
Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs

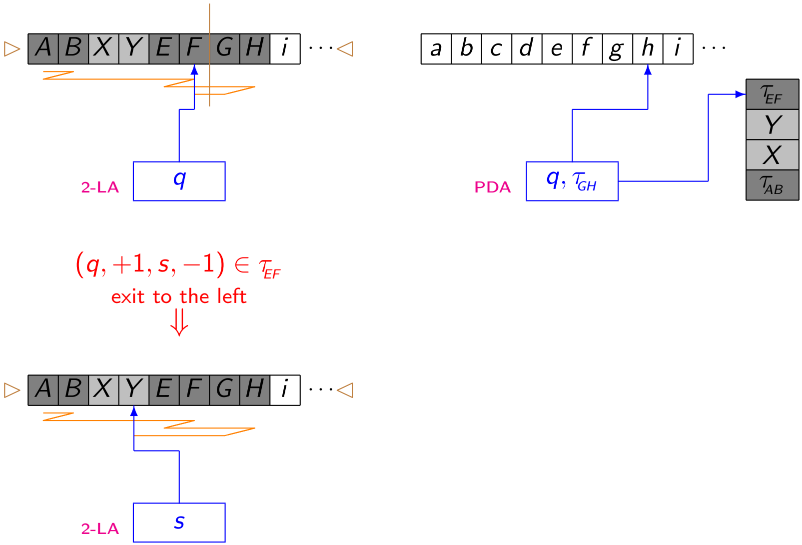


$(q, +1, s, -1) \in \mathcal{T}_{EF}$

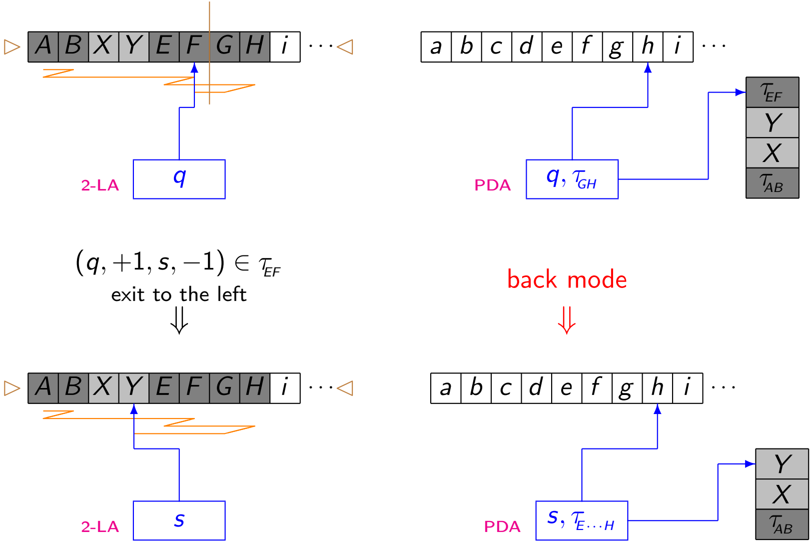
exit to the left

$\Downarrow$

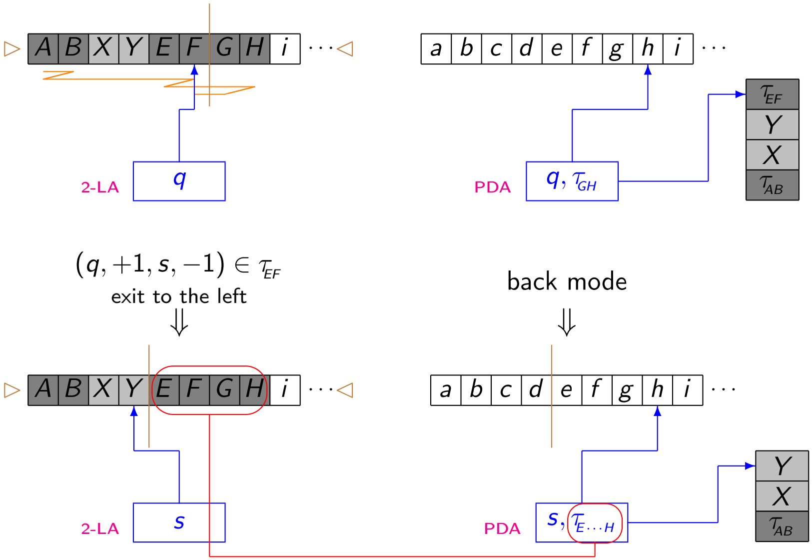
Simulation of 2-LAs by PDAs



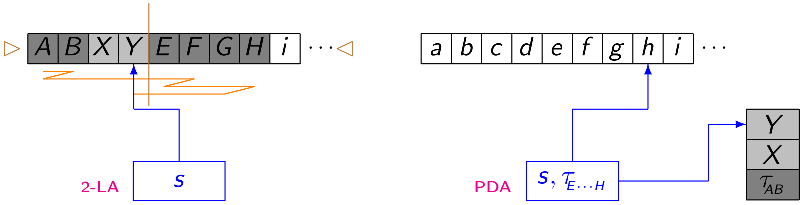
Simulation of 2-LAs by PDAs



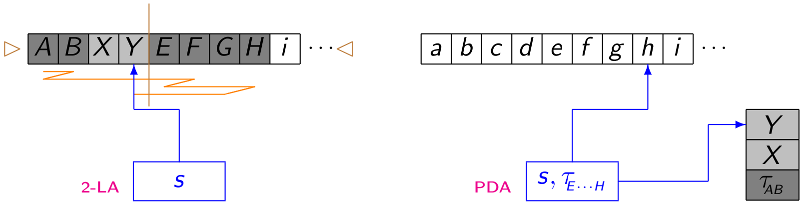
Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs

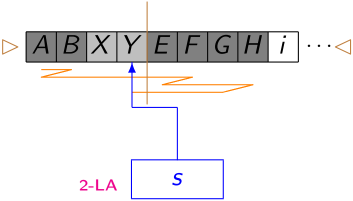


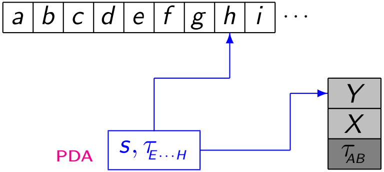
$\delta(s, Y) \ni (p, D, +1)$

move to the right

$\Downarrow$

Simulation of 2-LAs by PDAs

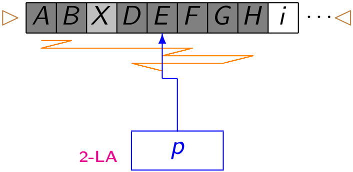




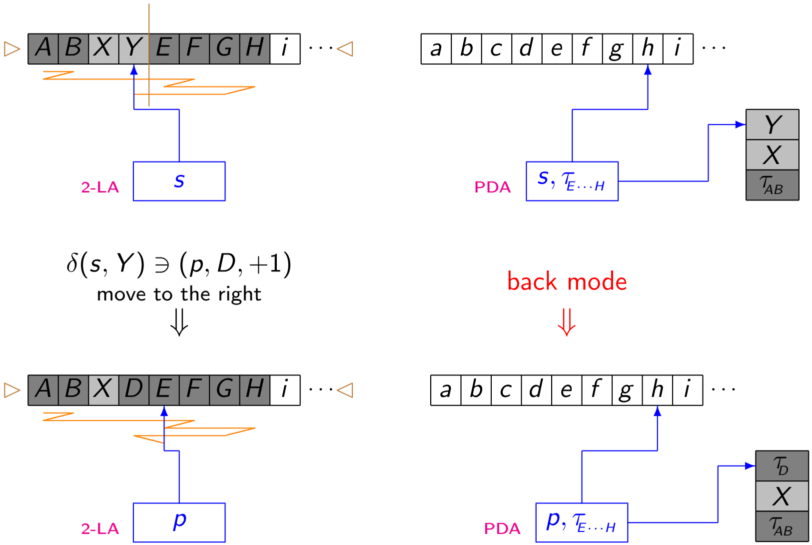
$$\delta(s, Y) \ni (p, D, +1)$$

move to the right

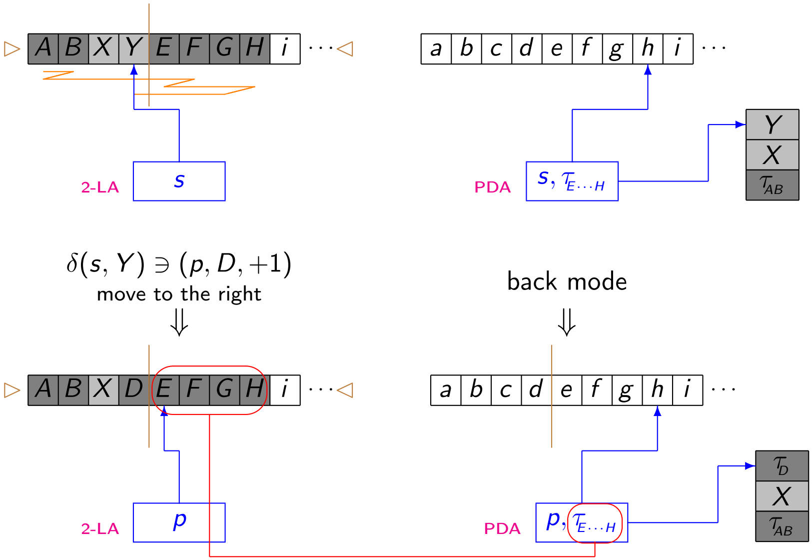
$\Downarrow$



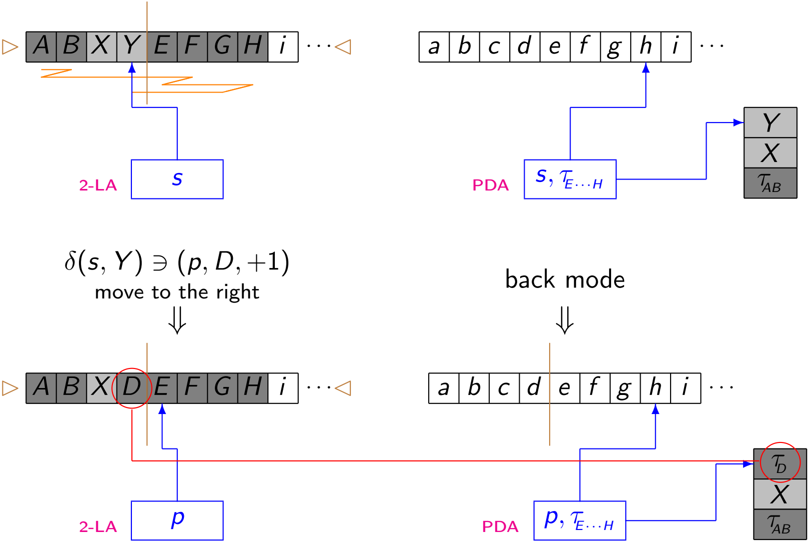
Simulation of 2-LAs by PDAs



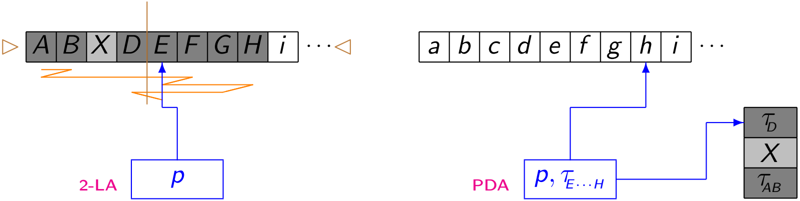
Simulation of 2-LAs by PDAs



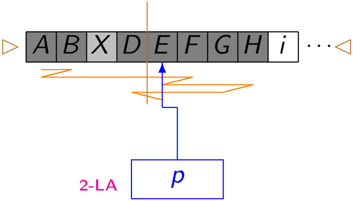
Simulation of 2-LAs by PDAs

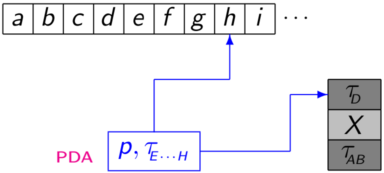


Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



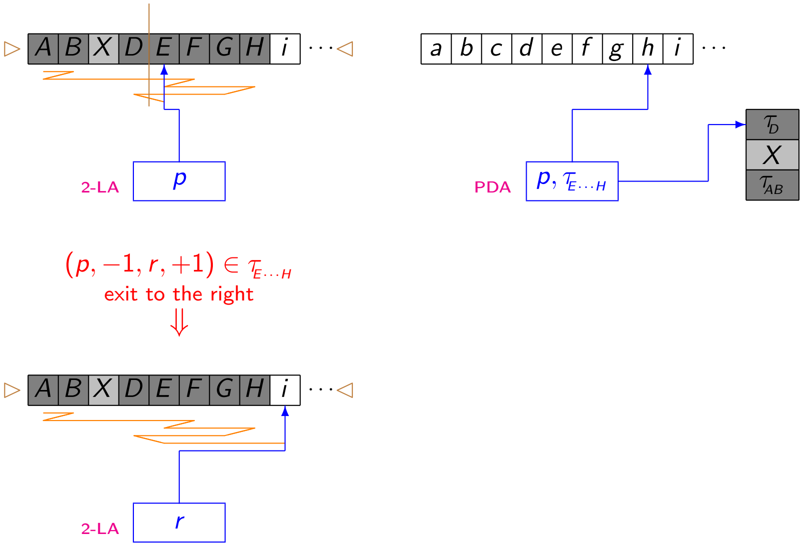


$(p, -1, r, +1) \in \mathcal{T}_{E\cdots H}$

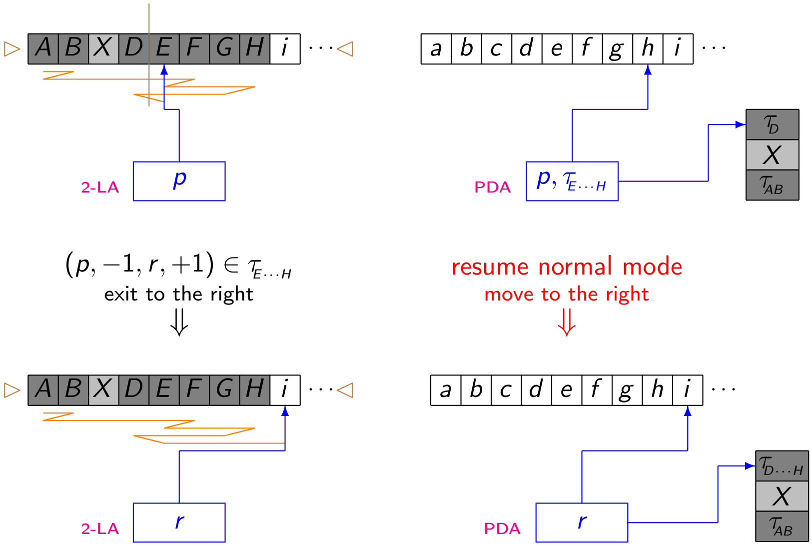
exit to the right

$\Downarrow$

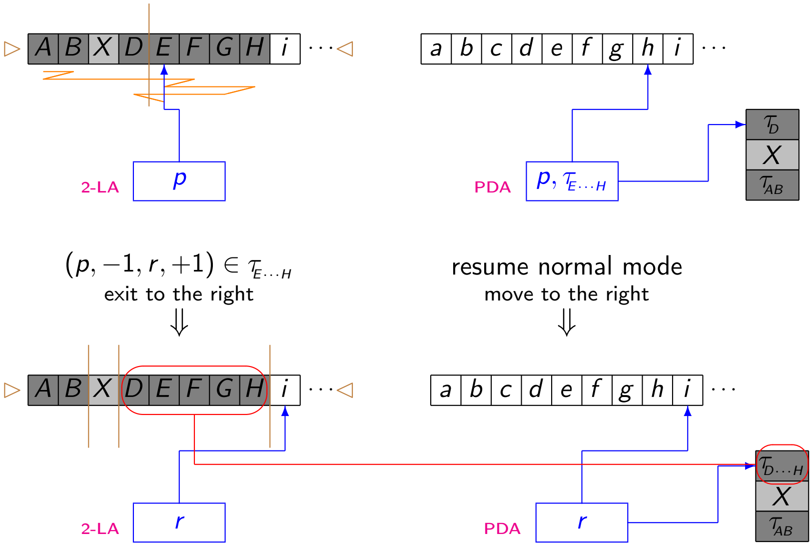
Simulation of 2-LAs by PDAs



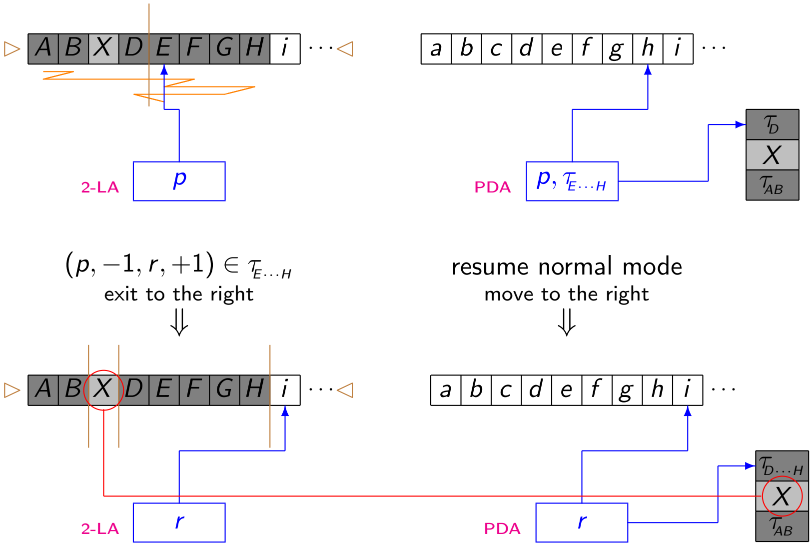
Simulation of 2-LAs by PDAs



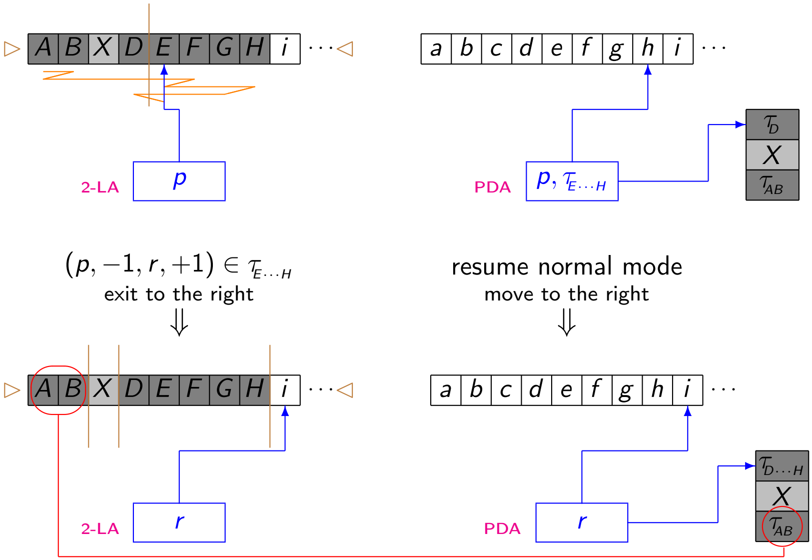
Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs



Simulation of 2-LAs by PDAs

Summing up...

Given a 2-LA M with:

- n states
- m symbol working alphabet

Given a 2-LA M with:

- n states
- m symbol working alphabet

At most 2^{4n^2} many different tables!

Simulation of 2-LAs by PDAs

Summing up...

Given a 2-LA M with:

- n states
- m symbol working alphabet

At most 2^{4n^2} many different tables!

Resulting PDA:

- States
  Normal mode: states of M
  Back mode: (q, τ)
  q state of M, τ transition table

States
$2n(2^{4n^2}+1)+1$

Simulation of 2-LAs by PDAs

Summing up...

Given a 2-LA M with:

- n states
- m symbol working alphabet

At most 2^{4n^2} many different tables!

Resulting PDA:

- States
  Normal mode: states of M
  Back mode: (q, τ)
  q state of M, τ transition table
- Pushdown symbols
  - Tape symbols of M
  - Transition tables

States
$2n(2^{4n^2} + 1) + 1$

Pushdown symbols
$m + 2^{4n^2}$

Given a 2-LA M with:

- n states
- m symbol working alphabet

At most 2^{4n^2} many different tables!

Resulting PDA:

- States
  Normal mode: states of M
  Back mode: (q, τ)
  q state of M, τ transition table
- Pushdown symbols
  - Tape symbols of M
  - Transition tables
- Each move can increase the stack height at most by 1

States
$2n(2^{4n^2} + 1) + 1$

Pushdown symbols
$m + 2^{4n^2}$

Simulation of 2-LAs by PDAs

Summing up...

Given a 2-LA M with:

- n states
- m symbol working alphabet

At most 2^{4n^2} many different tables!

Resulting PDA:

- States
  Normal mode: states of M
  Back mode: (q, τ)
  q state of M, τ transition table
- Pushdown symbols
  - Tape symbols of M
  - Transition tables
- Each move can increase the stack height at most by 1

States

$2n(2^{4n^2} + 1) + 1$

Pushdown symbols

$m + 2^{4n^2}$

2-LAs → PDAs

Exponential cost

Optimality: the Witness Languages K_n

Given $n \geq 1$:

$$K_n \quad =$$

Optimality: the Witness Languages K_n

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_k} \quad \underbrace{b_1 \quad b_2 \quad \cdots \quad b_n}_{x}$$

$$K_n \quad = \{x_1 x_2 \cdots x_k x \mid \quad k \geq 0,\ x_1, x_2, \ldots, x_k, x \in \{0,1\}^n,$$

Optimality: the Witness Languages K_n

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_k} \quad \underbrace{b_1 \quad b_2 \quad \cdots \quad b_n}_{x}$$

$\cdots$

At least n of these blocks are equal
to the last block x

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

Optimality: the Witness Languages K_n

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_k} \quad \underbrace{b_1 \quad b_2 \quad \cdots \quad b_n}_{x}$$

$\ldots$

At least n of these blocks are equal
to the last block x

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

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

Optimality: the Witness Languages K_n

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_k} \quad \underbrace{b_1 \quad b_2 \quad \cdots \quad b_n}_{x}$$

At least n of these blocks are equal
to the last block x

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

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

Optimality: the Witness Languages K_n

Given $n \geq 1$:

$$\underbrace{a_1 \quad a_2 \quad \cdots \quad a_n}_{x_1} \quad \cdots \quad \underbrace{a_{n+1}\, a_{n+2} \quad \cdots \quad a_{2n}}_{x_k} \quad \underbrace{b_1 \quad b_2 \quad \cdots \quad b_n}_{x}$$

...

At least n of these blocks are equal
to the last block x

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

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

How to Recognize K_n

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

1. Scan all the tape from left to right
2. Start to move to the left and mark the rightmost n symbols
3. Compare each block of length n (from the right), symbol by symbol, with the last block
4. When the left end of the tape is reached accept if and only if the number of block equal to the last one is $\geq n$

How to Recognize K_n

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

1. Scan all the tape from left to right
2. Start to move to the left and mark the rightmost n symbols
3. Compare each block of length n (from the right), symbol by symbol, with the last block
4. When the left end of the tape is reached accept if and only if the number of block equal to the last one is $\geq n$

How to Recognize K_n

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

1. Scan all the tape from left to right
2. Start to move to the left and mark the rightmost n symbols
3. Compare each block of length n (from the right), symbol by symbol, with the last block
4. When the left end of the tape is reached accept if and only if the number of block equal to the last one is $\geq n$

How to Recognize K_n

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

1. Scan all the tape from left to right
2. Start to move to the left and mark the rightmost n symbols
3. Compare each block of length n (from the right), symbol by symbol, with the last block
4. When the left end of the tape is reached accept if and only if the number of block equal to the last one is $\geq n$

How to Recognize K_n

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

1. Scan all the tape from left to right
2. Start to move to the left and mark the rightmost n symbols
3. Compare each block of length n (from the right), symbol by symbol, with the last block
4. When the left end of the tape is reached accept if and only if the number of block equal to the last one is $\geq n$

Complexity:

- K_n is accepted by a deterministic 2-LA with $O(n^2)$ states and a fixed working alphabet
- Each PDA accepting K_n has size at least exponential in n (Proof based on the *interchange lemma* for CFLs)

How to Recognize K_n

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

1. Scan all the tape from left to right
2. Start to move to the left and mark the rightmost n symbols
3. Compare each block of length n (from the right), symbol by symbol, with the last block
4. When the left end of the tape is reached accept if and only if the number of block equal to the last one is $\geq n$

Complexity:

- K_n is accepted by a deterministic 2-LA with $O(n^2)$ states and a fixed working alphabet
- Each PDA accepting K_n has size at least exponential in n (Proof based on the *interchange lemma* for CFLs)

Simulation of 2-LAs by PDAs

Cost of the simulation

- Exponential size for the simulation of 2-LAs by PDAs
- Optimal

Computational Power of Limited Automata

From the simulations:

- 2-Limited Automata $\equiv$ CFLs

What about d-Limited Automata, with $d > 2$?

- They are still characterize CFLs [Hibbard '67]
- They can be simulated by exponentially larger PDAs [Kutrib&P.&Wendlandt subm.]

What about 1-Limited Automata?

- Regular languages [Wagner&Wechsung '86]

Computational Power of Limited Automata

From the simulations:

- 2-Limited Automata $\equiv$ CFLs

What about d-Limited Automata, with $d > 2$?

- They are still characterize CFLs [Hibbard '67]
- They can be simulated by exponentially larger PDAs [Kutrib&P.&Wendlandt subm.]

What about 1-Limited Automata?

- Regular languages [Wagner&Wechsung '86]

Computational Power of Limited Automata

From the simulations:

- 2-Limited Automata $\equiv$ CFLs

What about d-Limited Automata, with $d > 2$?

- They are still characterize CFLs [Hibbard '67]
- They can be simulated by exponentially larger PDAs [Kutrib&P.&Wendlandt subm.]

What about 1-Limited Automata?

- Regular languages [Wagner&Wechsung '86]

Computational Power of Limited Automata

From the simulations:

- 2-Limited Automata $\equiv$ CFLs

What about d-Limited Automata, with $d > 2$?

- They are still characterize CFLs [Hibbard '67]
- They can be simulated by exponentially larger PDAs [Kutrib&P.&Wendlandt subm.]

What about 1-Limited Automata?

- Regular languages [Wagner&Wechsung '86]

Computational Power of Limited Automata

From the simulations:

- 2-Limited Automata $\equiv$ CFLs

What about d-Limited Automata, with $d > 2$?

- They are still characterize CFLs [Hibbard '67]
- They can be simulated by exponentially larger PDAs [Kutrib&P.&Wendlandt subm.]

What about 1-Limited Automata?

- Regular languages [Wagner&Wechsung '86]

Computational Power of Limited Automata

From the simulations:

- 2-Limited Automata $\equiv$ CFLs

What about d-Limited Automata, with $d > 2$?

- They are still characterize CFLs [Hibbard '67]
- They can be simulated by exponentially larger PDAs [Kutrib&P.&Wendlandt subm.]

What about 1-Limited Automata?

- Regular languages [Wagner&Wechsung '86]

Determinism vs Nondeterminism

- Determinism is preserved by the exponential simulation of 2-limited automata by PDAs
provided that the input of the PDA is right end-marked
- *Without end-marker:* double exponential simulation
- *Conjecture:* this cost cannot be reduced
- The converse simulation also preserve determinsm

Deterministic 2-Limited Automata ≡ DCFLs

[P.&Pisoni '15]

Determinism vs Nondeterminism

- Determinism is preserved by the exponential simulation of 2-limited automata by PDAs
  provided that the input of the PDA is right end-marked
- *Without end-marker:* double exponential simulation
- *Conjecture:* this cost cannot be reduced
- The converse simulation also preserve determinsm

Deterministic 2-Limited Automata ≡ DCFLs

[P.&Pisoni '15]

Determinism vs Nondeterminism

- Determinism is preserved by the exponential simulation of 2-limited automata by PDAs
  provided that the input of the PDA is right end-marked
- *Without end-marker:* double exponential simulation
- *Conjecture:* this cost cannot be reduced
- The converse simulation also preserve determinsm

Deterministic 2-Limited Automata ≡ DCFLs

[P.&Pisoni '15]

Determinism vs Nondeterminism

- Determinism is preserved by the exponential simulation of 2-limited automata by PDAs
  provided that the input of the PDA is right end-marked
- *Without end-marker:* double exponential simulation
- *Conjecture:* this cost cannot be reduced
- The converse simulation also preserve determinsm

Deterministic 2-Limited Automata ≡ DCFLs

[P.&Pisoni '15]

Determinism vs Nondeterminism

- Determinism is preserved by the exponential simulation of 2-limited automata by PDAs
  provided that the input of the PDA is right end-marked
- *Without end-marker:* double exponential simulation
- *Conjecture:* this cost cannot be reduced
- The converse simulation also preserve determinsm

Deterministic 2-Limited Automata $\equiv$ DCFLs

[P.&Pisoni '15]

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

1-Limited Automata

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 DFA which simulates the two-way NFA:

  - transition table of the already scanned input prefix
  - set of possible current states

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 DFA which simulates the two-way NFA:

  - transition table of the already scanned input prefix
  - set of possible current states

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*

  for $x \in \Sigma^*$, $\tau_x \subseteq Q \times Q$: $(p, q) \in \tau_x$ iff 

- Finite control of the DFA which simulates the two-way NFA:

  - transition table of the already scanned input prefix
  - set of possible current states

Simulation of 1-Limited Automata by Finite Automata

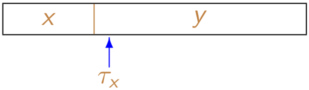
Main idea: transformation of *two-way* NFAs into *one-way* DFAs
[Shepherdson '59]

- First visit to a cell: direct simulation

- Further visits: *transition tables*

  for $x \in \Sigma^*$, $\tau_x \subseteq Q \times Q$: $(p, q) \in \tau_x$ iff 

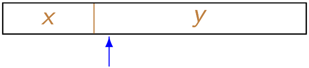
- Finite control of the DFA which simulates the two-way NFA:



  - transition table of the already scanned input prefix
  - set of possible current states

Simulation of 1-Limited Automata by Finite Automata

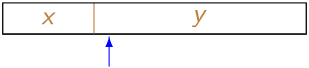
Simulation of 1-LAs: [Wagner&Wechsung '86]



- The transition table depends on the string used to rewrite the input prefix x
- This string was nondeterministically chosen by the 1-LA

Simulation of 1-Limited Automata by Finite Automata

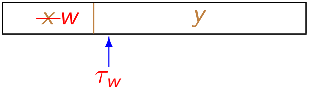
Simulation of 1-LAs: [Wagner&Wechsung '86]



- The transition table depends on the string used to rewrite the input prefix x
- This string was nondeterministically chosen by the 1-LA

Simulation of 1-Limited Automata by Finite Automata

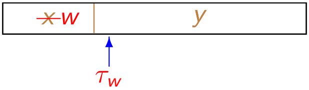
Simulation of 1-LAs: [Wagner&Wechsung '86]



- The transition table depends on the string used to rewrite the input prefix x
- This string was nondeterministically chosen by the 1-LA

Simulation of 1-Limited Automata by Finite Automata

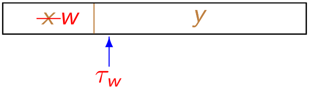
Simulation of 1-LAs: [Wagner&Wechsung '86]



- The transition table depends on the string used to rewrite the input prefix x
- This string was nondeterministically chosen by the 1-LA

Simulation of 1-Limited Automata by Finite Automata

Simulation of 1-LAs: [Wagner&Wechsung '86]



- The transition table depends on the string used to rewrite the input prefix x
- This string was nondeterministically chosen by the 1-LA

The simulating DFA keeps in its finite control a *sets of transition tables*

1-Limited Automata $\rightarrow$ 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 $\rightarrow$ 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 $\to$ 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 $\to$ 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 → 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 $\rightarrow$ 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 [P.&Pisoni '14]

Fixed $n \geq 1$:

$$L_n =$$

Optimality: the Witness Languages [P.&Pisoni '14]

Fixed $n \geq 1$:

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

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

Optimality: the Witness Languages [P.&Pisoni '14]

Fixed $n \geq 1$:

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

At least n of these blocks are equal

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

Optimality: the Witness Languages [P.&Pisoni '14]

Fixed $n \geq 1$:

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

At least n of these blocks are equal

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

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 [P.&Pisoni '14]

Fixed $n \geq 1$:

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

At least n of these blocks are equal

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

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 [P.&Pisoni '14]

Fixed $n \geq 1$:

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

At least n of these blocks are equal

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

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

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 (3 tape scans):
  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 (3 tape scans):
  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 (3 tape scans):
  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 (3 tape scans):
  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 (3 tape scans):
  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 (3 tape scans):
  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 (3 tape scans):
  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 (3 tape scans):
  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!

How to Recognize L_n: Nondeterministic Finite Automata

- Idea:
  - *Guess* $x \in \{0,1\}^n$
  - *Verify* whether or not n blocks in the input contains x
- State upper bound:
  - Finite control: a counter $\leq n$ for the occurrences of x, and a counter modulo n for input positions
  - Number of states: $O(n^2 \cdot 2^n)$
- State lower bound:
  - $n^2 \cdot 2^n$ (fooling set technique)

How to Recognize L_n: Nondeterministic Finite Automata

- Idea:
  - *Guess* $x \in \{0,1\}^n$
  - *Verify* whether or not n blocks in the input contains x
- State upper bound:
  - Finite control: a counter $\leq n$ for the occurrences of x, and a counter modulo n for input positions
  - Number of states: $O(n^2 \cdot 2^n)$
- State lower bound:
  - $n^2 \cdot 2^n$ (fooling set technique)

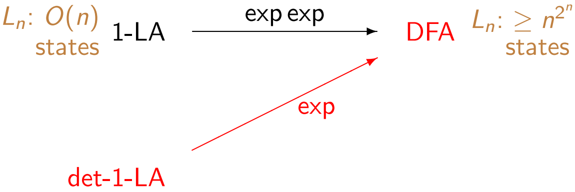
How to Recognize L_n: Nondeterministic Finite Automata

- Idea:
  - *Guess* $x \in \{0, 1\}^n$
  - *Verify* whether or not n blocks in the input contains x
- State upper bound:
  - Finite control: a counter $\leq n$ for the occurrences of x, and a counter modulo n for input positions
  - Number of states: $O(n^2 \cdot 2^n)$
- State lower bound:
  - $n^2 \cdot 2^n$ (fooling set technique)

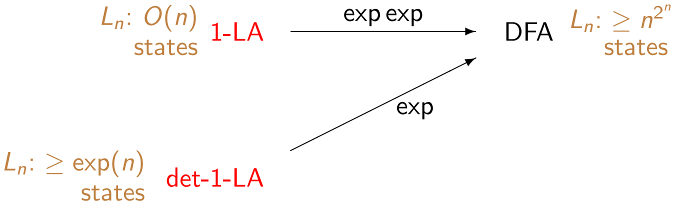
Nondetermism vs. Determinism in 1-LAs

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

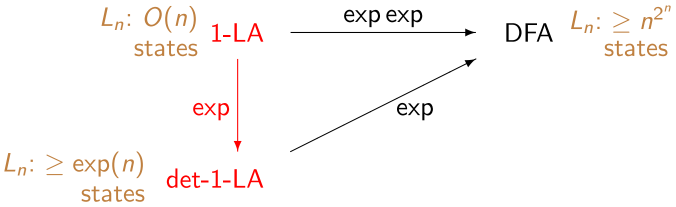
Nondetermism vs. Determinism in 1-LAs



Nondetermism vs. Determinism in 1-LAs



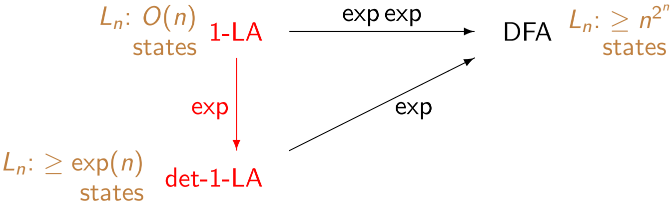
Nondetermism vs. Determinism in 1-LAs



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?

Strongly Limited Automata

Different Restrictions

- Dyck languages are accepted without fully using capabilities of 2-limited automata
- Chomsky-Schützenberger Theorem: Recognition of CFLs can be reduced to recognition of Dyck languages

Different Restrictions

- Dyck languages are accepted without fully using capabilities of 2-limited automata
- Chomsky-Schützenberger Theorem: Recognition of CFLs can be reduced to recognition of Dyck languages

Different Restrictions

- Dyck languages are accepted without fully using capabilities of 2-limited automata
- Chomsky-Schützenberger Theorem: Recognition of CFLs can be reduced to recognition of Dyck languages

Question

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

Different Restrictions

- Dyck languages are accepted without fully using capabilities of 2-limited automata
- Chomsky-Schützenberger Theorem: Recognition of CFLs can be reduced to recognition of Dyck languages

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 a fixed symbol)
- No other changes of the tape are allowed

Strongly Limited Automata [P.'15]

- Model inspired by the algorithm used by 2-limited automata to recognize Dyck languages
- Restrictions on
  - state changes
  - head reversals
  - rewriting operations

Strongly Limited Automata [P.'15]

- Model inspired by the algorithm used by 2-limited automata to recognize Dyck languages
- Restrictions on
  - state changes
  - head reversals
  - rewriting operations

Dyck Language Recognition

▷ | (|) | (| [| [|] |] |) | ◁

- Moves to the right:
  - to search a closed bracket
- Moves to the left:
  - to search an open bracket
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no rewritings in the final scan

Dyck Language Recognition

▷	(	)	(	[	[	]	]	)	◁

- Moves to the right:
  - to search a closed bracket

Only one state q_0!

- Moves to the left:
  - to search an open bracket
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no rewritings in the final scan

Dyck Language Recognition

▷	(	)	(	[	[	]	]	)	◁

- Moves to the right:
  - to search a closed bracket

Only one state q_0!

- Moves to the left:
  - to search an open bracket
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no rewritings in the final scan

Dyck Language Recognition

▷	(	)	(	[	[	]	]	)	◁

- Moves to the right:
  - to search a closed bracket — Only one state q_0!
- Moves to the left:
  - to search an open bracket — One state for each type of bracket!
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no rewritings in the final scan

Dyck Language Recognition

$\triangleright$ | (|) | (| [| [|] |] |) | $\triangleleft$

- Moves to the right:
  - to search a closed bracket — Only one state q_0!
- Moves to the left:
  - to search an open bracket — One state for each type of bracket!
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no rewritings in the final scan

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
  - Q_L moving from right to left
  - Q_F final scan
    when $\lhd$ is reached move from right to left and
    test the membership of the tape content to a "local" language

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_q\xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X}_{q_0}$ write X, enters state q_0, *turn to the right*
  - Q_Υ final scan
    when $\lhd$ is reached move from right to left and
    test the membership of the tape content to a “local” language

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working
- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_q\overset{X}{\longleftarrow\!\!\supset}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\overset{X}{\longleftarrow}$ write X, do not change state, *move to the left*
    - $\overset{X}{\longrightarrow}_{q_0}$ write X, enters state q_0, *turn to the right*
  - $Q_\top$ final scan

    when $\lhd$ is reached move from right to left and test the membership of the tape content to a "local" language

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_{q}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X}{}_{q_0}$ write X, enters state q_0, *turn to the right*
  - Q_{Υ} final scan
    when $\lhd$ is reached move from right to left and
    test the membership of the tape content to a "local" language

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_{q}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X}_{q_0}$ write X, enters state q_0, *turn to the right*
  - $Q_\curlyvee$ final scan

    when $\lhd$ is reached move from right to left and

    test the membership of the tape content to a "local" language

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X} q_0$ write X, enters state q_0, *turn to the right*
  - Q_Υ final scan
    - *when $\lhd$ is reached move from right to left and*
    - *test the membership of the tape content to a "local" language*

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X} q_0$ write X, enters state q_0, *turn to the right*
  - Q_Υ final scan
    when $\triangleleft$ is reached move from right to left and
    test the membership of the tape content to a "local" language

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X} q_0$ write X, enters state q_0, *turn to the right*
  - Q_Υ final scan
    - *when $\lhd$ is reached move from right to left and*
    - *test the membership of the tape content to a "local" language*

Strongly Limited Automata

- Alphabet
  - Σ input
  - Γ working

- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$ write X, do not change state, *move to the left*
    - $\xrightarrow{X} q_0$ write X, enters state q_0, *turn to the right*
  - Q_Υ final scan
    - *when $\lhd$ is reached move from right to left and*
    - *test the membership of the tape content to a "local" language*

Strongly Limited Automata

- Alphabet

  Σ input

  Γ working

- States and moves

  q_0 initial state, moving from left to right

  $\dashrightarrow$ *move to the right*

  $q \overset{X}{\longleftarrow}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*

  Q_L moving from right to left

  $\dashleftarrow$ *move to the left*

  $\overset{X}{\longleftarrow}$ write X, do not change state, *move to the left*

  $\overset{X}{\longrightarrow} q_0$ write X, enters state q_0, *turn to the right*

  Q_Υ final scan

  when $\lhd$ is reached move from right to left and

  test the membership of the tape content to a “local” language

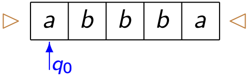
Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{\mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$
q_0
$Q_L = \{q_a, q_b\}$

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

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

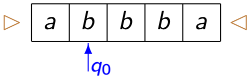
q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

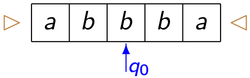
q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

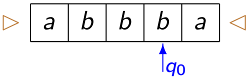
q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

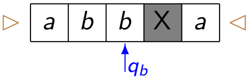
q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\overset{X}{\longleftrightarrow}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

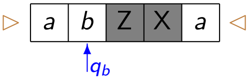
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\overset{Z}{\longleftarrow}$ write Z, do not change state, *move to the left*

$\overset{Y}{\hookrightarrow}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

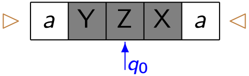
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

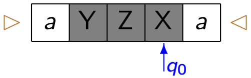
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

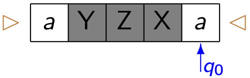
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\xleftarrow{X}$ write X $\in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

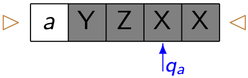
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

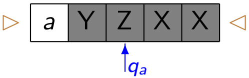
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

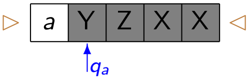
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

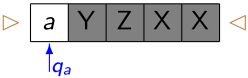
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

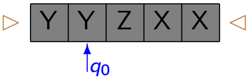
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

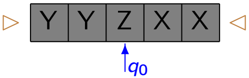
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

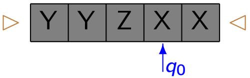
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

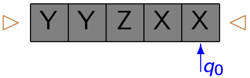
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$q_\sigma \xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

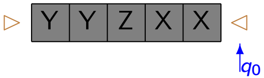
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Transitions:

q_0 $\dashrightarrow$ *move to the right*

other possibility in cell not yet rewritten:

$_{q_\sigma}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q_\sigma \in Q_L$, *turn to the left*

q_σ moving from right to left

cells already rewritten: $\dashleftarrow$ *move to the left*

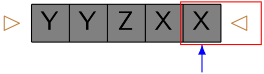
cells containing $\gamma \in \{a, b\}$, nondeterministically select between:

$\xleftarrow{Z}$ write Z, do not change state, *move to the left*

$\xrightarrow{Y}_{q_0}$ write Y, enters state q_0, *turn to the right* (only if $\gamma = \sigma$)

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Final phase:

- The string between the end-markers should belong to

$$Y^*ZX^* + Y^*X^*$$

  with the exceptions of inputs of length ≤ 1

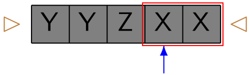
- The following two-letter factors are allowed:

  $\triangleright Y \quad YY \quad YZ \quad ZX \quad YX \quad XX \quad X\triangleleft$

  $\triangleright a \quad \triangleright b \quad a\triangleleft \quad b\triangleleft \quad \triangleright\triangleleft$

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Final phase:

- The string between the end-markers should belong to

$$Y^*ZX^* + Y^*X^*$$

  with the exceptions of inputs of length ≤ 1

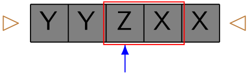
- The following two-letter factors are allowed:

  $\triangleright Y$ $\quad YY$ $\quad YZ$ $\quad ZX$ $\quad YX$ $\quad XX$ $\quad X\triangleleft$

  $\triangleright a$ $\quad \triangleright b$ $\quad a\triangleleft$ $\quad b\triangleleft$ $\quad \triangleright\triangleleft$

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Final phase:

- The string between the end-markers should belong to

$$Y^*ZX^* + Y^*X^*$$

  with the exceptions of inputs of length ≤ 1

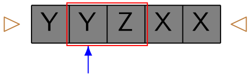
- The following two-letter factors are allowed:

  $\triangleright$Y YY YZ ZX YX XX X$\triangleleft$

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

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Final phase:

- The string between the end-markers should belong to

$$Y^*ZX^* + Y^*X^*$$

  with the exceptions of inputs of length ≤ 1

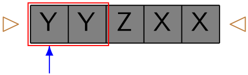
- The following two-letter factors are allowed:

  $\triangleright Y \quad YY \quad YZ \quad ZX \quad YX \quad XX \quad X\triangleleft$

  $\triangleright a \quad \triangleright b \quad a\triangleleft \quad b\triangleleft \quad \triangleright\triangleleft$

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Final phase:

- The string between the end-markers should belong to

$$Y^*ZX^* + Y^*X^*$$

  with the exceptions of inputs of length ≤ 1

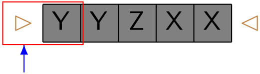
- The following two-letter factors are allowed:

  $\triangleright Y \quad YY \quad YZ \quad ZX \quad YX \quad XX \quad X\triangleleft$

  $\triangleright a \quad \triangleright b \quad a\triangleleft \quad b\triangleleft \quad \triangleright\triangleleft$

Strongly Limited Automata: Palindromes

$\Sigma = \{a, b\}$, $\Gamma = \{X, Y, Z\}$
q_0
$Q_L = \{q_a, q_b\}$



Final phase:

- The string between the end-markers should belong to

$$Y^*ZX^* + Y^*X^*$$

  with the exceptions of inputs of length ≤ 1

- The following two-letter factors are allowed:

  $\triangleright Y \quad YY \quad YZ \quad ZX \quad YX \quad XX \quad X\triangleleft$

  $\triangleright a \quad \triangleright b \quad a\triangleleft \quad b\triangleleft \quad \triangleright\triangleleft$

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

  - 2-limited automata can be exponentially smaller
- CFLs $\rightarrow$ strongly limited automata:

  conversion from CFGs which heavily uses nondeterminism

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

  - 2-limited automata can be exponentially smaller
- CFLs $\rightarrow$ strongly limited automata:

  conversion from CFGs which heavily uses nondeterminism

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

  - 2-limited automata can be exponentially smaller
- CFLs $\rightarrow$ strongly limited automata:

  conversion from CFGs which heavily uses nondeterminism

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

  - 2-limited automata can be exponentially smaller
- CFLs $\rightarrow$ strongly limited automata:

  conversion from CFGs which heavily uses nondeterminism

Determinism vs Nondeterminism

What is the power of *deterministic* strongly limited automata?

- Each deterministic strongly limited automaton can be simulated by a deterministic 2-LA
- Deterministic languages as
  $L_1 = \{ca^n b^n \mid n \geq 0\} \cup \{da^{2n} b^n \mid n \geq 0\}$
  $L_2 = \{a^n b^{2n} \mid n \geq 0\}$
  are not accepted by *deterministic strongly limited automata*

Determinism vs Nondeterminism

What is the power of *deterministic* strongly limited automata?

- Each deterministic strongly limited automaton can be simulated by a deterministic 2-LA
- Deterministic languages as

$$L_1 = \{ca^n b^n \mid n \geq 0\} \cup \{da^{2n} b^n \mid n \geq 0\}$$
$$L_2 = \{a^n b^{2n} \mid n \geq 0\}$$

are not accepted by *deterministic strongly limited automata*

Determinism vs Nondeterminism

What is the power of *deterministic* strongly limited automata?

- Each deterministic strongly limited automaton can be simulated by a deterministic 2-LA
- Deterministic languages as

$$L_1 = \{ca^nb^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$$
$$L_2 = \{a^nb^{2n} \mid n \geq 0\}$$

  are not accepted by *deterministic strongly limited automata*

Determinism vs Nondeterminism

What is the power of *deterministic* strongly limited automata?

- Each deterministic strongly limited automaton can be simulated by a deterministic 2-LA
- Deterministic languages as

$$L_1 = \{ca^n b^n \mid n \geq 0\} \cup \{da^{2n} b^n \mid n \geq 0\}$$
$$L_2 = \{a^n b^{2n} \mid n \geq 0\}$$

  are not accepted by *deterministic strongly limited automata*

Proper subclass of deterministic context-free languages

Determinism vs Nondeterminism: a Small Change

- Moving to the right, a strongly limited automaton can use only q_0
- A possible modification:

  a set of states Q_R used while moving to the right

  - the simulation by PDAs remains polynomial
  - $L_1 = \{ca^nb^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$

    $L_2 = \{a^nb^{2n} \mid n \geq 0\}$

    are accepted by *deterministic devices*

Problem

What is the class of languages accepted
by the deterministic version of devices so obtained?

Determinism vs Nondeterminism: a Small Change

- Moving to the right, a strongly limited automaton can use only q_0
- A possible modification:

  a set of states Q_R used while moving to the right

  - the simulation by PDAs remains polynomial
  - $L_1 = \{ca^n b^n \mid n \geq 0\} \cup \{da^{2n} b^n \mid n \geq 0\}$

    $L_2 = \{a^n b^{2n} \mid n \geq 0\}$

    are accepted by *deterministic devices*

Problem

What is the class of languages accepted by the deterministic version of devices so obtained?

Determinism vs Nondeterminism: a Small Change

- Moving to the right, a strongly limited automaton can use only q_0
- A possible modification:

  a set of states Q_R used while moving to the right

  - the simulation by PDAs remains polynomial
  - $L_1 = \{ca^nb^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$

    $L_2 = \{a^nb^{2n} \mid n \geq 0\}$

    are accepted by *deterministic devices*

Problem

What is the class of languages accepted by the deterministic version of devices so obtained?

Determinism vs Nondeterminism: a Small Change

- Moving to the right, a strongly limited automaton can use only q_0
- A possible modification:

  a set of states Q_R used while moving to the right

  - the simulation by PDAs remains polynomial
  - $L_1 = \{ca^nb^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$
    $L_2 = \{a^nb^{2n} \mid n \geq 0\}$
    are accepted by *deterministic devices*

Problem

What is the class of languages accepted
by the deterministic version of devices so obtained?

Determinism vs Nondeterminism: a Small Change

- Moving to the right, a strongly limited automaton can use only q_0
- A possible modification:

  a set of states Q_R used while moving to the right

  - the simulation by PDAs remains polynomial
  - $L_1 = \{ca^nb^n \mid n \geq 0\} \cup \{da^{2n}b^n \mid n \geq 0\}$
    $L_2 = \{a^nb^{2n} \mid n \geq 0\}$
    are accepted by *deterministic devices*

Problem

What is the class of languages accepted
by the deterministic version of devices so obtained?

Final Remarks

Active Visits ad Return Complexity

Active visit of a tape cell: any visit changing the content

Active Visits ad Return Complexity

Active visit of a tape cell: any visit changing the content

Return Complexity

Maximum number of visits to a tape cell counted starting from the *first* active visit [Wechsung '75]

Active Visits ad Return Complexity

Active visit of a tape cell: any visit changing the content

Return Complexity
Maximum number of visits to a tape cell counted starting from the *first* active visit [Wechsung '75]

Dual Return Complexity
Maximum number of visits to a tape cell counted up to the *last* active visit

dret-c(d) $\equiv$ d-limited automata

Active Visits ad Return Complexity

Active visit of a tape cell: any visit changing the content

Return Complexity

Maximum number of visits to a tape cell counted
starting from the *first* active visit [Wechsung '75]

- ret-c(1): regular languages
- ret-c(d), $d \geq 2$: context-free languages
- ret-c(2) *deterministic:* not comparable with DCFLs

Dual Return Complexity

Maximum number of visits to a tape cell
counted up to the *last* active visit

dret-c(d) $\equiv$ d-limited automata

Active Visits ad Return Complexity

Active visit of a tape cell: any visit changing the content

Return Complexity

Maximum number of visits to a tape cell counted starting from the *first* active visit [Wechsung '75]

- ret-c(1): regular languages
- ret-c(d), $d \geq 2$: context-free languages
- ret-c(2) *deterministic:* not comparable with DCFLs

Dual Return Complexity

Maximum number of visits to a tape cell counted up to the *last* active visit

dret-c(d) $\equiv$ d-limited automata

ret-c($f(n)$)=dret-c($f(n)$) =1AuxPDA($f(n)$)

[Wechsung&Brandstädt '79]

Thank you for your attention!