Strongly Limited Automata

Giovanni Pighizzini

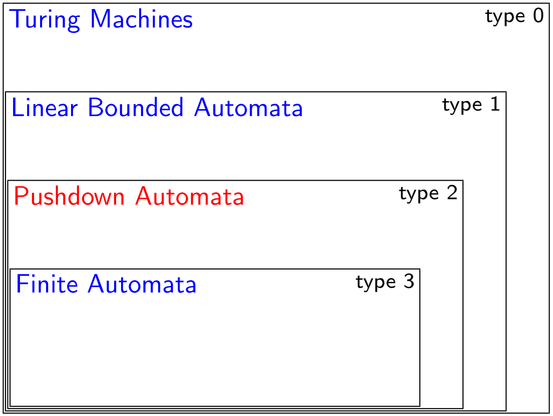
Dipartimento di Informatica
Università degli Studi di Milano, Italy

NCMA 2014

Kassel, Germany
July 28–29, 2014

UNIVERSITÀ DEGLI STUDI
DI MILANO

The Chomsky Hierarchy



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*

Limited Automata [Hibbard'67]

One-tape Turing machines with restricted rewritings

Definition

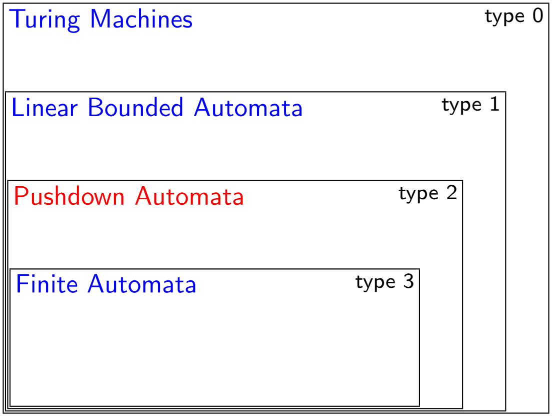
Fixed an integer $d \geq 1$, a *d-limited automaton* is

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

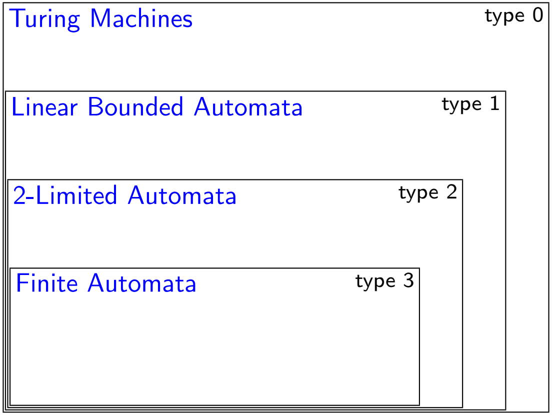
Computational power

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

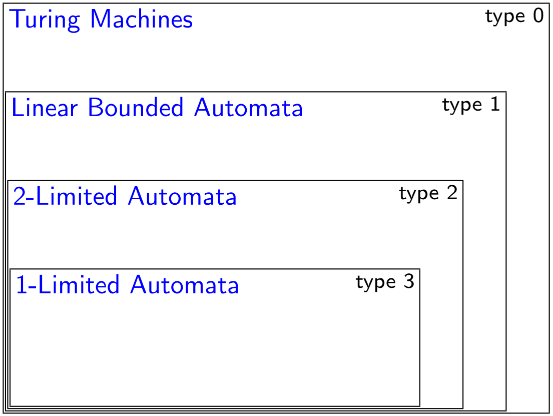
The Chomsky Hierarchy



The Chomsky Hierarchy



The Chomsky Hierarchy



Motivations

- 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

Motivations

- 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

Motivations

- 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?

Motivations

- 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

A Different Restriction: Strongly Limited Automata

- Model inspired by the algorithm used by 2-limited automata to recognize Dyck languages
- Restrictions on
  - state changes
  - head reversals
  - rewriting operations
- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related

A Different Restriction: Strongly Limited Automata

- Model inspired by the algorithm used by 2-limited automata to recognize Dyck languages
- Restrictions on
  - state changes
  - head reversals
  - rewriting operations
- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related

A Different Restriction: Strongly Limited Automata

- Model inspired by the algorithm used by 2-limited automata to recognize Dyck languages
- Restrictions on
  - state changes
  - head reversals
  - rewriting operations
- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

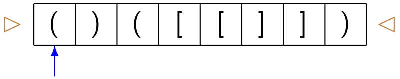
  are polynomially related

A Different Restriction: Strongly Limited Automata

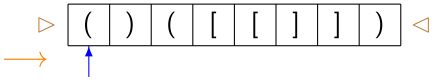
- Model inspired by the algorithm used by 2-limited automata to recognize Dyck languages
- Restrictions on
  - state changes
  - head reversals
  - rewriting operations
- Computational power: same as 2-limited automata (CFLs)
- Descriptional power: the sizes of equivalent
  - CFGs
  - PDAs
  - strongly limited automata

  are polynomially related

Dyck Language Recognition

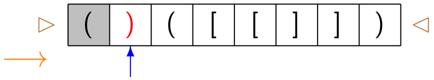


Dyck Language Recognition



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

Dyck Language Recognition



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

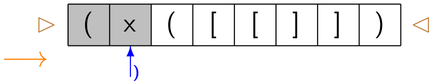
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

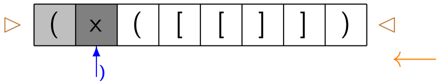
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

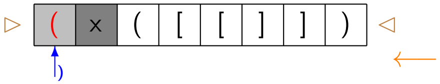
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

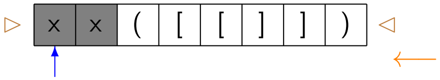
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

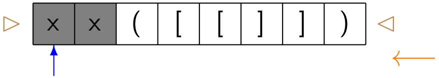
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

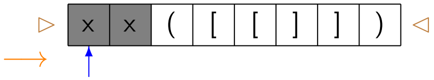
(v) Repeat from the beginning

Dyck Language Recognition



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

Dyck Language Recognition



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

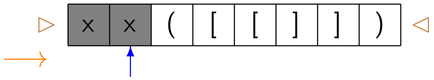
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

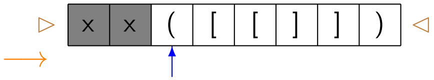
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

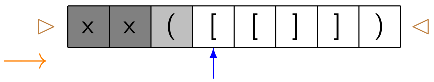
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

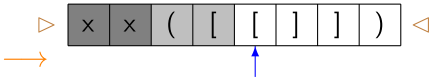
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

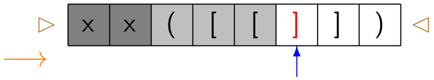
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

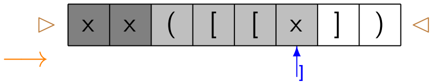
(v) Repeat from the beginning

Dyck Language Recognition



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

Dyck Language Recognition



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

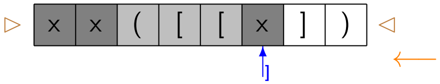
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

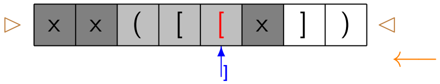
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

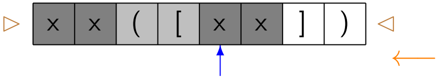
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

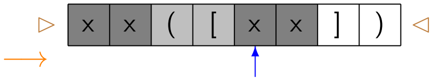
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

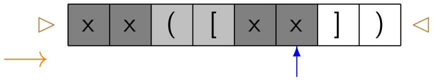
(v) Repeat from the beginning

Dyck Language Recognition



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

Dyck Language Recognition



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

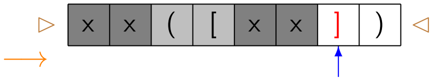
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

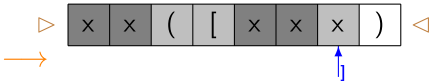
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

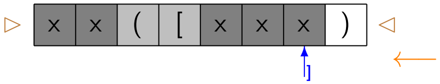
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

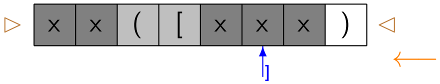
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

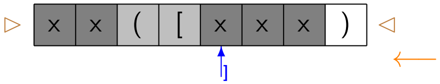
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

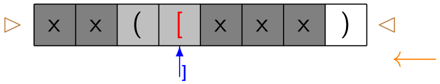
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

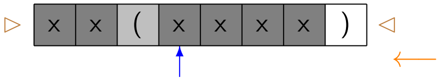
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

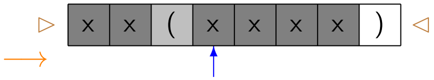
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

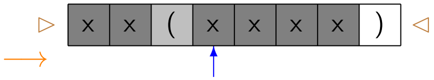
(v) Repeat from the beginning

Dyck Language Recognition



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

Dyck Language Recognition



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

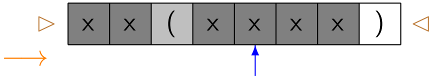
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

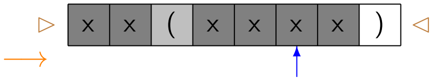
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

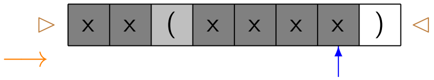
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

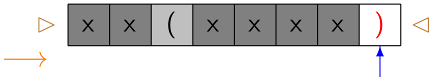
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

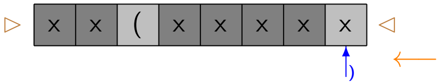
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

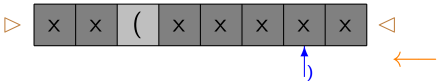
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

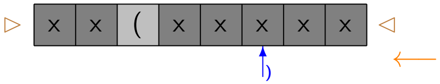
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

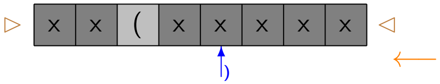
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

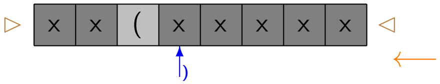
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

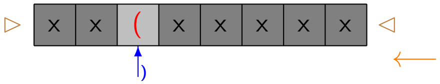
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

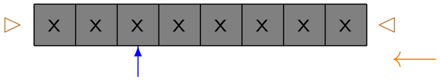
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

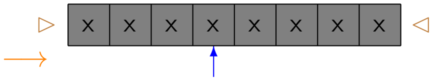
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

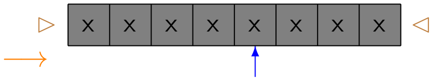
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

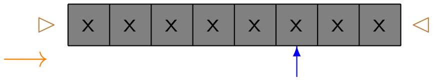
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

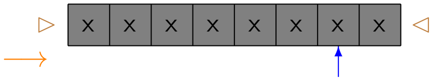
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

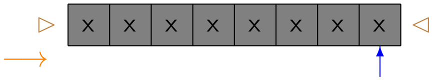
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

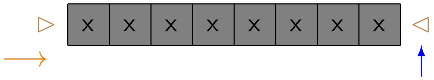
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

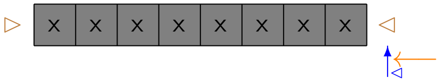
(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

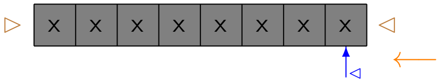
Special cases:

(i') When $\triangleleft$ is reached scan all the tape
accept iff each tape cell contains x

(iii') If in (iii) $\triangleright$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



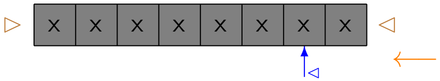
(i) Move to the right to search a closed bracket
(ii) Rewrite it by x
(iii) Move to the left to search an open bracket
(iv) If it matches then rewrite it by x
(v) Repeat from the beginning

Special cases:

(i') When $\lhd$ is reached scan all the tape
accept iff each tape cell contains x

(iii') If in (iii) $\rhd$ is reached then *reject*
(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

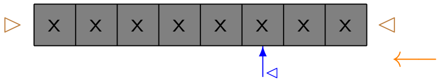
(i') When $\triangleleft$ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) $\triangleright$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

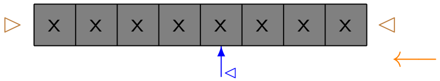
(i') When $\triangleleft$ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) $\triangleright$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

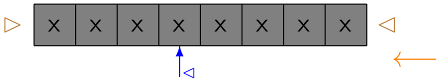
(i') When $\triangleleft$ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) $\triangleright$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

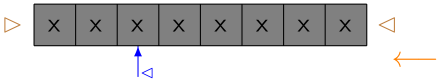
(i') When ◁ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) ▷ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

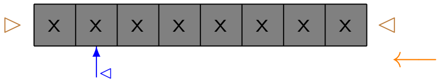
(i') When $\lhd$ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) $\rhd$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

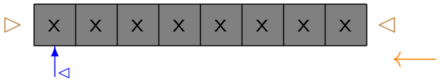
(i') When $\lhd$ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) $\rhd$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

Special cases:

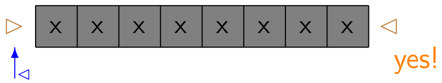
(i') When $\triangleleft$ is reached scan all the tape

accept iff each tape cell contains x

(iii') If in (iii) $\triangleright$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition

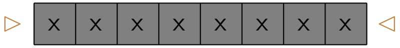


(i) Move to the right to search a closed bracket
(ii) Rewrite it by x
(iii) Move to the left to search an open bracket
(iv) If it matches then rewrite it by x
(v) Repeat from the beginning

Special cases:

(i') When ◁ is reached scan all the tape
accept iff each tape cell contains x
(iii') If in (iii) ▷ is reached then *reject*
(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

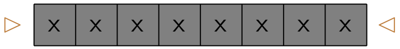
Special cases:

(i') When $\lhd$ is reached scan all the tape
accept iff each tape cell contains x

(iii') If in (iii) $\rhd$ is reached then *reject*

(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



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

(ii) Rewrite it by x

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

(iv) If it matches then rewrite it by x

(v) Repeat from the beginning

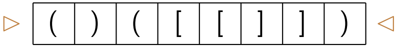
Special cases:

(i') When $\triangleleft$ is reached scan all the tape
accept iff each tape cell contains x

(iii') If in (iii) $\triangleright$ is reached then *reject*

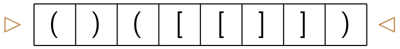
(iv') If in (iv) a no matching open bracket is found then *reject*

Dyck Language Recognition



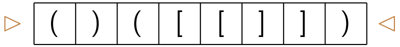
- Moves to the right:
  - to search a closed bracket
- Moves to the left:
  - to search an open bracket
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no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

Dyck Language Recognition



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

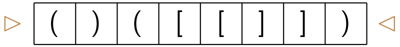
- Moves to the left:
  - to search an open bracket
  - to check the tape content in the final scan from right to left
- Rewritings:
  - each closed bracket is rewritten in the first visit
  - each open bracket is rewritten in the second visit
  - no rewritings in the final scan

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



- 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

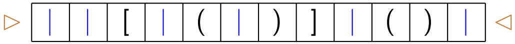
Extended Dyck Language

- Strings padded with “neutral symbols”

$$| \quad | \quad [\quad | \quad (\quad | \quad) \quad] \quad | \quad (\quad) \quad |$$

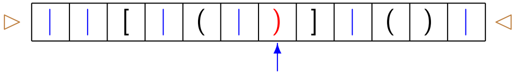
Extended Dyck Language

- Strings padded with “neutral symbols”
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x’s



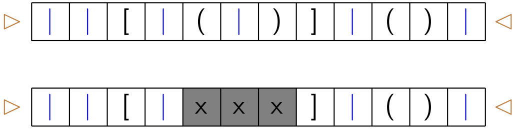
Extended Dyck Language

- Strings padded with "neutral symbols"
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x's



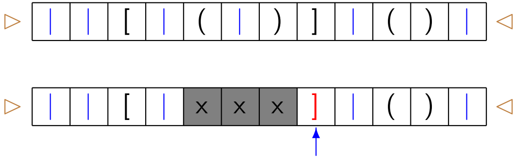
Extended Dyck Language

- Strings padded with “neutral symbols”
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x’s



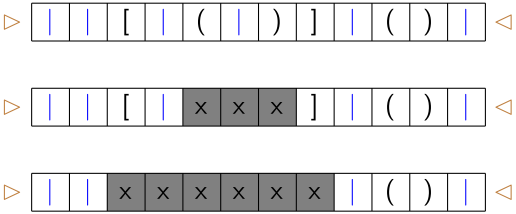
Extended Dyck Language

- Strings padded with “neutral symbols”
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x’s



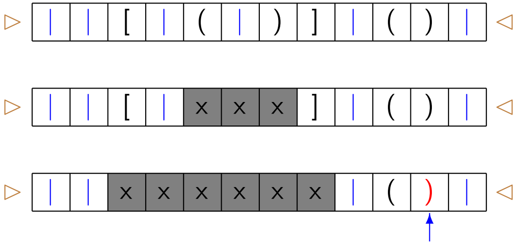
Extended Dyck Language

- Strings padded with "neutral symbols"
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x's



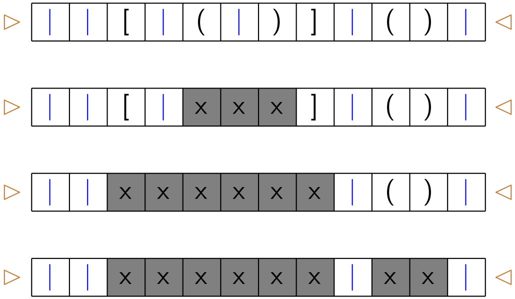
Extended Dyck Language

- Strings padded with “neutral symbols”
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x’s



Extended Dyck Language

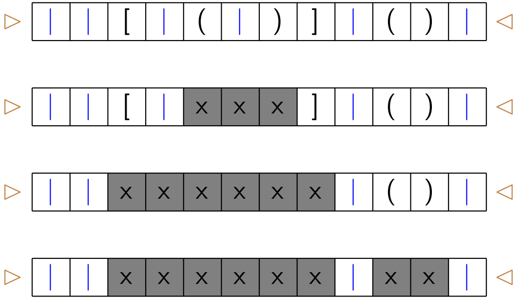
- Strings padded with “neutral symbols”
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x’s



▷ | | [| (|)] | ( ) | ◁
▷ | | [| x x x] | ( ) | ◁
▷ | | x x x x x x | ( ) | ◁
▷ | | x x x x x x | x x | ◁

Extended Dyck Language

- Strings padded with “neutral symbols”
- Similar recognition technique:
  - while moving to the left searching an open bracket, neutral symbols are rewritten
  - the tape should finally contain only neutral symbols and x’s


▷ | | [| (|)] | ( ) | ◁
▷ | | [| x x x] | ( ) | ◁
▷ | | x x x x x x | ( ) | ◁
▷ | | x x x x x x | x x | ◁

- The procedure can be adapted *to generate* strings in the language

Strongly Limited Automata

- Alphabet

  Σ input

  Γ working

  Υ $= \Sigma \cup \Gamma \cup \{\rhd, \lhd\}$ global alphabet

- States and moves

Strongly Limited Automata

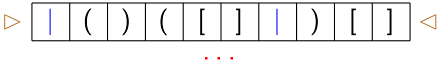
- Alphabet

Σ	input
Γ	working
Υ	$= \Sigma \cup \Gamma \cup \{\rhd, \lhd\}$ global alphabet

- 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*

Strongly Limited Automata



- Alphabet

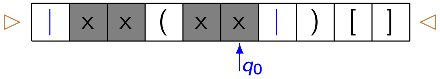
  Σ input
  Γ working
  Υ $= \Sigma \cup \Gamma \cup \{\rhd, \lhd\}$ global alphabet

- 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*

Strongly Limited Automata



- Alphabet

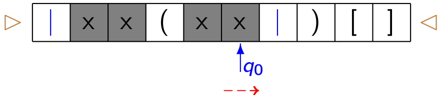
  Σ input
  Γ working
  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

  q_0 initial state, moving from left to right

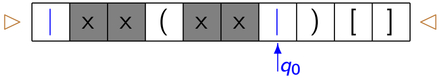
  $\dashrightarrow$ *move to the right*
  $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*

Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet
- 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*

Strongly Limited Automata



- Alphabet

  Σ input

  Γ working

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

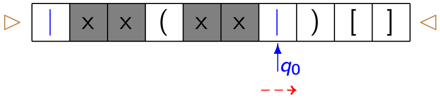
- States and moves

  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*

Strongly Limited Automata



- Alphabet

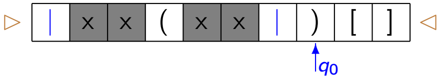
  Σ input
  Γ working
  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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*

Strongly Limited Automata



- Alphabet

  Σ input

  Γ working

  Υ $= \Sigma \cup \Gamma \cup \{\rhd, \lhd\}$ global alphabet

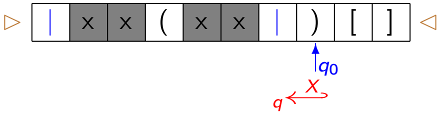
- States and moves

  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*

Strongly Limited Automata



- Alphabet

  Σ input

  Γ working

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

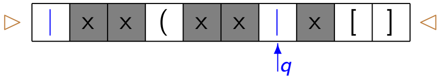
- States and moves

  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*

Strongly Limited Automata



- Alphabet

  Σ input
  Γ working
  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

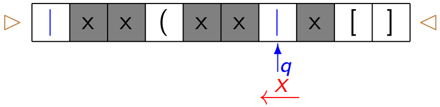
  q_0 initial state, moving from left to right

  $\dashrightarrow$ *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*

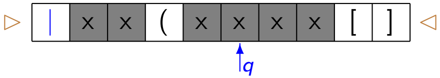
Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - $\Upsilon = \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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*

Strongly Limited Automata



- Alphabet

  Σ input

  Γ working

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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*

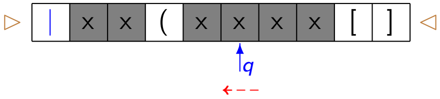
  Q_L moving from right to left

  $\dashleftarrow$ *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*

Strongly Limited Automata



- Alphabet

  Σ input
  Γ working
  $\Upsilon = \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

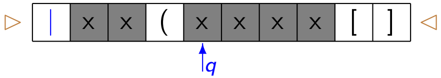
  q_0 initial state, moving from left to right

  $\dashrightarrow$ *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*

Strongly Limited Automata



- Alphabet

Σ	input
Γ	working
Υ	$= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

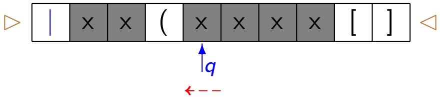
q_0 initial state, moving from left to right

$\dashrightarrow$	*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*

Strongly Limited Automata



- Alphabet

  Σ input

  Γ working

  Υ $=\Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

  q_0 initial state, moving from left to right

  $\dashrightarrow$ *move to the right*

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

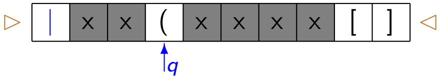
  Q_L moving from right to left

  $\dashleftarrow$ *move to the left*

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

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

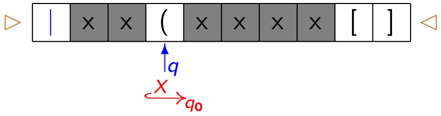
Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

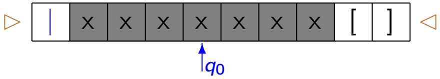
- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_{q}\xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$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

Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - $\Upsilon = \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet
- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_{q}\overset{X}{\hookleftarrow}$ 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*
    - $\overset{X}{\hookrightarrow}{}_{q_0}$ write X, enters state q_0, *turn to the right*

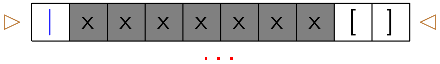
Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

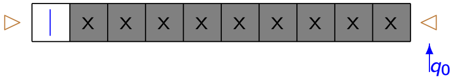
- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*
  - Q_L moving from right to left
    - $\dashleftarrow$ *move to the left*
    - $\xleftarrow{X}$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

Strongly Limited Automata



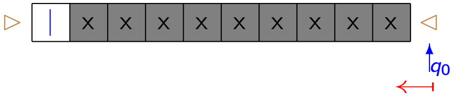
- Alphabet
  - Σ input
  - Γ working
  - $\Upsilon = \Sigma \cup \Gamma \cup \{\rhd, \lhd\}$ global alphabet
- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_{q}\!\overset{X}{\hookleftarrow}$ write $X \in \Gamma$, enter state $q \in Q_L$, *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*
    - $\overset{X}{\hookrightarrow}_{q_0}$ write X, enters state q_0, *turn to the right*

Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet
- States and moves
  - q_0 initial state, moving from left to right
    - $\dashrightarrow$ *move to the right*
    - ${}_{q}\!\hookleftarrow^{X}$ 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*
    - $\hookrightarrow^{X}{}_{q_0}$ write X, enters state q_0, *turn to the right*

Strongly Limited Automata



- Alphabet

  Σ input
  Γ working
  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

  q_0 initial state, moving from left to right

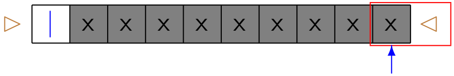
  $\dashrightarrow$ *move to the right*
  $q \xleftarrow{X}$ write $X \in \Gamma$, enter state $q \in Q_L$, *turn to the left*

  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

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

  q_0 initial state, moving from left to right

  $\dashrightarrow$ *move to the right*

  ${}_q\hookleftarrow^{X}$ 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*

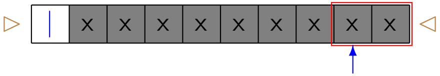
  $\hookrightarrow^{X}{}_{q_0}$ write X, enters state q_0, *turn to the right*

  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

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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*

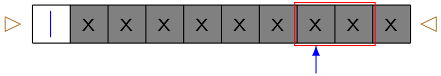
  $\xrightarrow{X}_{q_0}$ write X, enters state q_0, *turn to the right*

  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

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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*

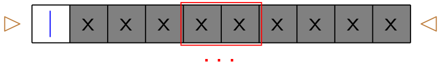
  $\xrightarrow{X}{}_{q_0}$ write X, enters state q_0, *turn to the right*

  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

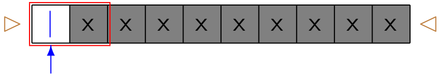
Strongly Limited Automata



- Alphabet
  - Σ input
  - Γ working
  - $\Upsilon = \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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

  Υ $= \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- States and moves

  q_0 initial state, moving from left to right

  $\dashrightarrow$ *move to the right*

  ${}_q\!\overset{X}{\hookleftarrow}$ 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*

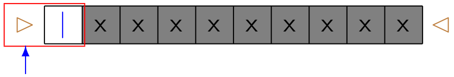
  $\overset{X}{\hookrightarrow}_{q_0}$ write X, enters state q_0, *turn to the right*

  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
  $\Upsilon = \Sigma \cup \Gamma \cup \{\triangleright, \triangleleft\}$ global alphabet

- 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

A Variant of the Chomsky-Schützenberger Theorem

$\Omega_{k,\ell}$ alphabet with k types of brackets and ℓ neutral symbols

$\widehat{D}_{k,\ell}$ *extended* Dyck language over $\Omega_{k,\ell}$

A Variant of the Chomsky-Schützenberger Theorem

$\Omega_{k,\ell}$ alphabet with k types of brackets and ℓ neutral symbols

$\widehat{D}_{k,\ell}$ *extended* Dyck language over $\Omega_{k,\ell}$

Theorem ([Okhotin'12])

$L \subseteq \Sigma^*$ is context-free iff there exist

- *integers $k, \ell \geq 1$*
- *a regular language $R \subseteq \Omega_{k,\ell}^*$*
- *a letter-to-letter homomorphism $h : \Omega_{k,\ell} \to \Sigma$*

such that $L = h(\widehat{D}_{k,\ell} \cap R)$

A Variant of the Chomsky-Schützenberger Theorem

$\Omega_{k,\ell}$ alphabet with k types of brackets and ℓ neutral symbols

$\widehat{D}_{k,\ell}$ *extended* Dyck language over $\Omega_{k,\ell}$

Theorem ([Okhotin'12])

$L \subseteq \Sigma^*$ is context-free iff there exist

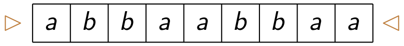
- *integers $k, \ell \geq 1$*
- *a regular language $R \subseteq \Omega_{k,\ell}^*$*
- *a letter-to-letter homomorphism $h : \Omega_{k,\ell} \to \Sigma$*

such that $L = h(\widehat{D}_{k,\ell} \cap R)$

Remarks

- k, ℓ are *polynomial* wrt the size of each CFG specifying L
- The language R is *local*

From CFLs to Strongly Limited Automata



$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

From CFLs to Strongly Limited Automata

$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$\triangleright \boxed{a}\boxed{b}\boxed{b}\boxed{a}\boxed{a}\boxed{b}\boxed{b}\boxed{a}\boxed{a} \triangleleft$

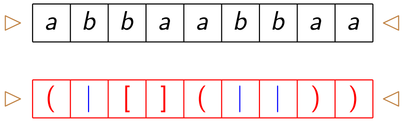
$w \in \Sigma^*$ input string

$L = h(\widehat{D}_{k,\ell} \cap R)$

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

From CFLs to Strongly Limited Automata



$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

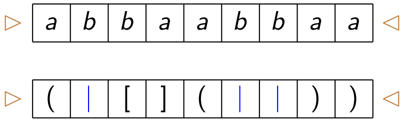
$L = h(\widehat{D}_{k,\ell} \cap R)$

$x \in \widehat{D}_{k,\ell}$

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

From CFLs to Strongly Limited Automata



$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

$L = h(\widehat{D}_{k,\ell} \cap R)$

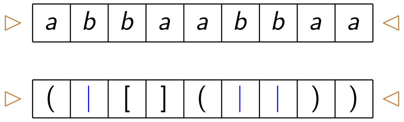
$x \in \widehat{D}_{k,\ell}$

$h(x) = w$? $x \in R$?

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

From CFLs to Strongly Limited Automata



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

$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

$L = h(\widehat{D}_{k,\ell} \cap R)$

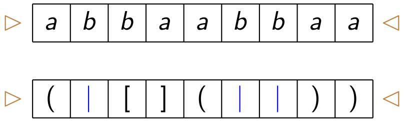
$x \in \widehat{D}_{k,\ell}$

$h(x) = w$? $x \in R$?

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

From CFLs to Strongly Limited Automata



$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

$L = h(\widehat{D}_{k,\ell} \cap R)$

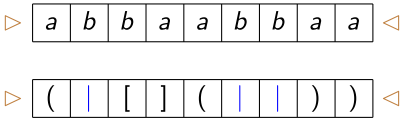
$x \in \widehat{D}_{k,\ell}$

$h(x) = w$? $x \in R$?

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

From CFLs to Strongly Limited Automata



$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

$L = h(\widehat{D}_{k,\ell} \cap R)$

$x \in \widehat{D}_{k,\ell}$

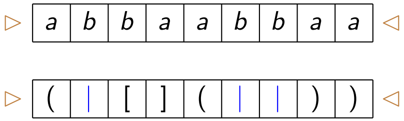
$h(x) = w$? $\quad x \in R$?

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

Given a CFG G for L, the size of M is polynomial in the size of G

From CFLs to Strongly Limited Automata



$L \subseteq \Sigma^*$ given CFL

$w \in L$?

$w \in \Sigma^*$ input string

$L = h(\widehat{D}_{k,\ell} \cap R)$

$x \in \widehat{D}_{k,\ell}$

$h(x) = w$? $\quad x \in R$?

Strongly limited automaton M for L:

- Guess and write on the tape $x \in \widehat{D}_{k,\ell}$
- While guessing each symbol x_i, check if $h(x_i) = w_i$
- In the final scan checks if $x \in R$

Given a CFG G for L, the size of M is polynomial in the size of G

CFGs $\rightarrow$ Strongly Limited Automata

Polynomial size!

Simulation of Strongly Limited Automata by PDAs

The simulation of 2-limited automata by PDAs is *exponential* in the description size [P&Pisoni'13]

Problem

How much it costs, in the description size, the simulation of strongly limited automata by PDAs*?*

This work

Polynomial cost!

Simulation of Strongly Limited Automata by PDAs

The simulation of 2-limited automata by PDAs is *exponential* in the description size [P&Pisoni'13]

Problem

How much it costs, in the description size, the simulation of strongly limited automata by PDAs*?*

This work

Polynomial cost!

Simulation of Strongly Limited Automata by PDAs

The simulation of 2-limited automata by PDAs is *exponential* in the description size [P&Pisoni'13]

Problem

How much it costs, in the description size, the simulation of strongly limited automata by PDAs*?*

This work

Polynomial cost!

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

$q \xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

${}_{q}\xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

$q \xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton $\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

${}_{q}\xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

$q \xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

$q \xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

$q \xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

Visits after 1st rewriting:
no changes of content and state

These visits do not need to be simulated

Final scan (from right to left)

Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

$\mathcal{M}$ strongly limited automaton

$\mathcal{A}$ simulating PDA

Tape cell c reached for the first time:

$\dashrightarrow$ content not modified now, but
it could be changed in the 2nd visit

guess the symbol written in the 2nd visit and
save it on the stack with the current symbol

$q \xleftarrow{X}$ content modified, head turned to the left

enter *back mode* to check previous guesses
saved on the pushdown

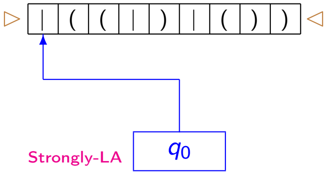
Visits after 1st rewriting:
no changes of content and state

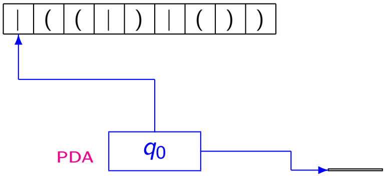
These visits do not need to be simulated

Final scan (from right to left)

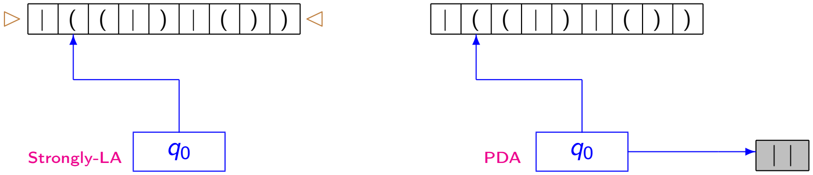
Simulated from left to right
"in parallel" with previous moves
while guessing and simulating rewritings

Simulation of Strongly Limited Automata by PDAs

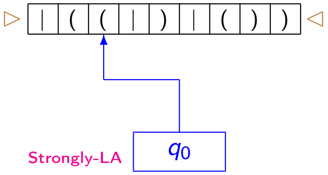


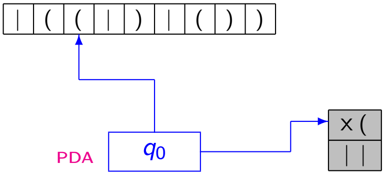


Simulation of Strongly Limited Automata by PDAs

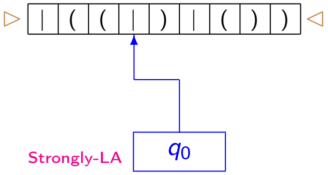


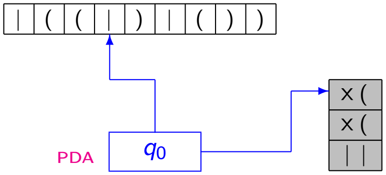
Simulation of Strongly Limited Automata by PDAs



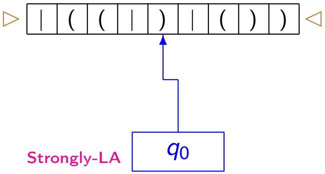


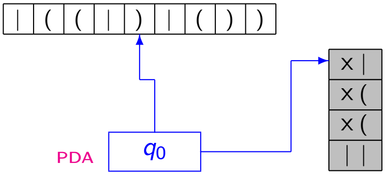
Simulation of Strongly Limited Automata by PDAs



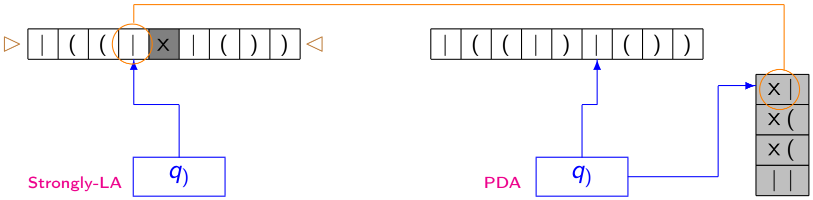


Simulation of Strongly Limited Automata by PDAs

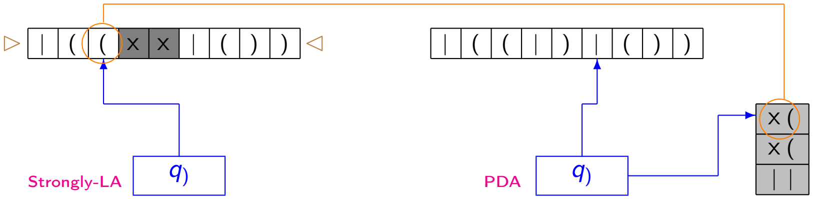




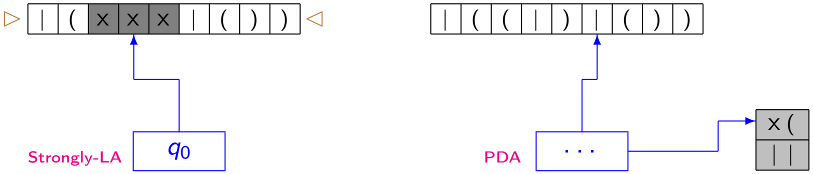
Simulation of Strongly Limited Automata by PDAs



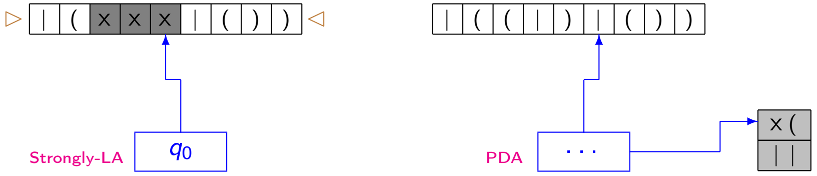
Simulation of Strongly Limited Automata by PDAs



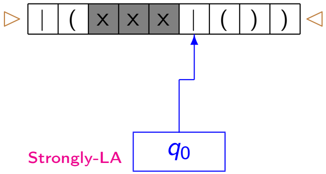
Simulation of Strongly Limited Automata by PDAs

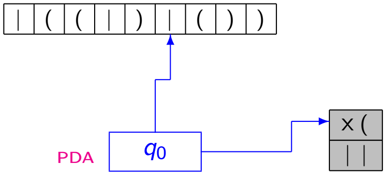


Simulation of Strongly Limited Automata by PDAs

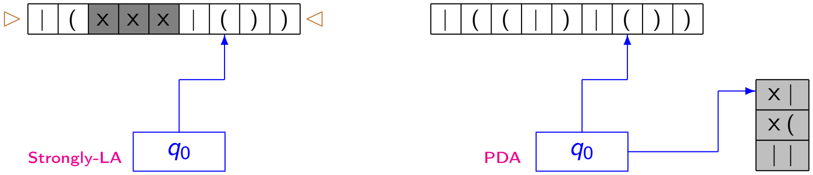


Simulation of Strongly Limited Automata by PDAs

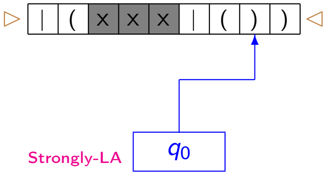


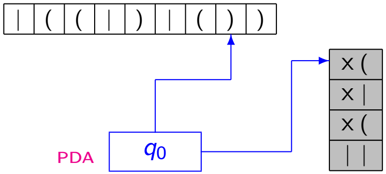


Simulation of Strongly Limited Automata by PDAs

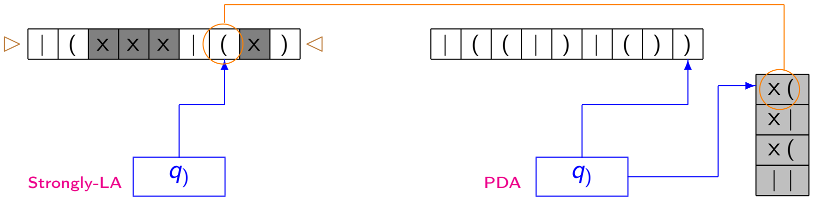


Simulation of Strongly Limited Automata by PDAs

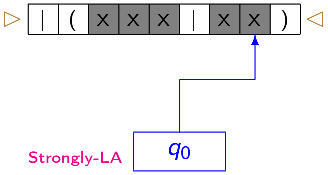


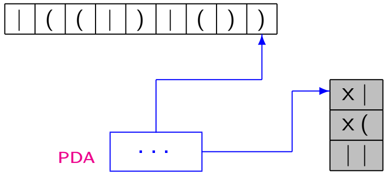


Simulation of Strongly Limited Automata by PDAs

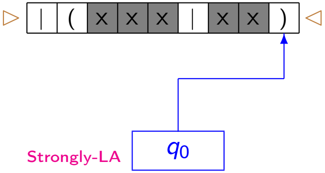


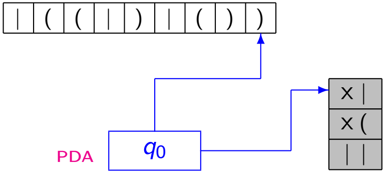
Simulation of Strongly Limited Automata by PDAs



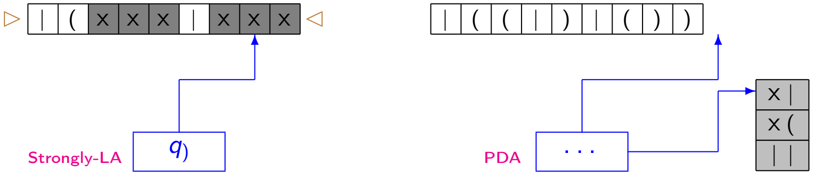


Simulation of Strongly Limited Automata by PDAs

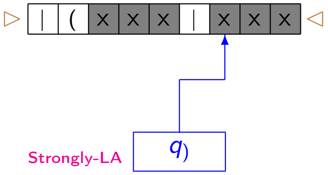


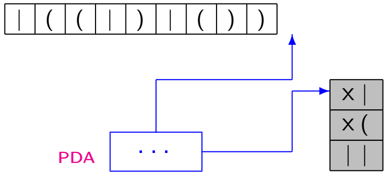


Simulation of Strongly Limited Automata by PDAs

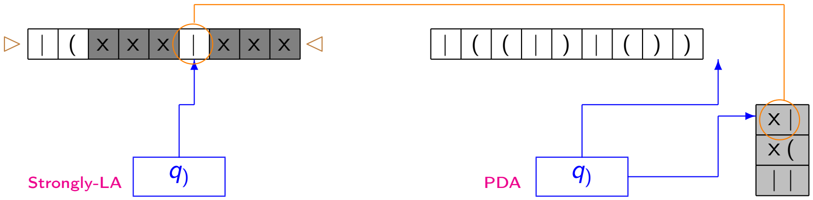


Simulation of Strongly Limited Automata by PDAs

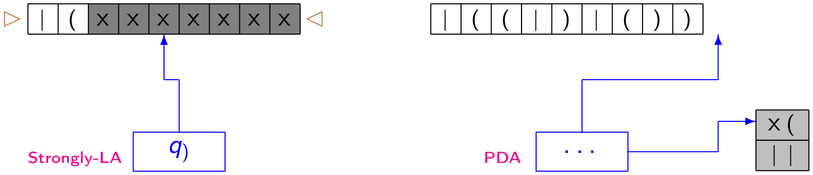




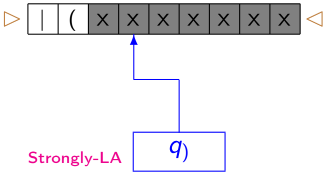
Simulation of Strongly Limited Automata by PDAs

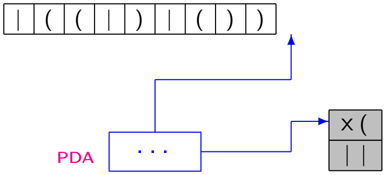


Simulation of Strongly Limited Automata by PDAs

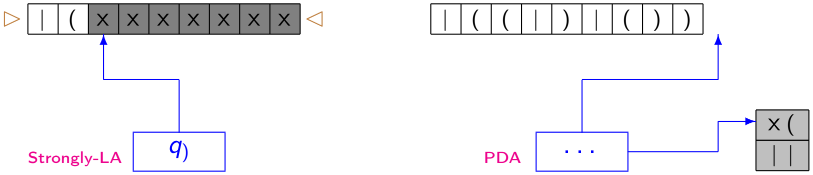


Simulation of Strongly Limited Automata by PDAs

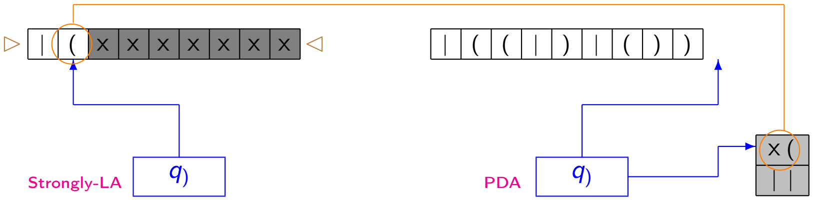




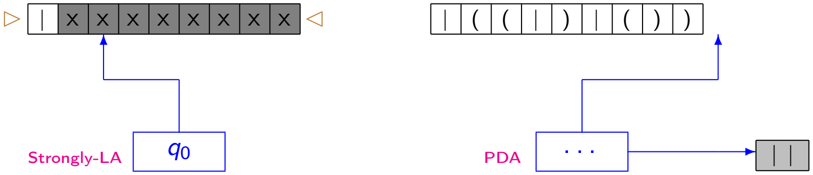
Simulation of Strongly Limited Automata by PDAs



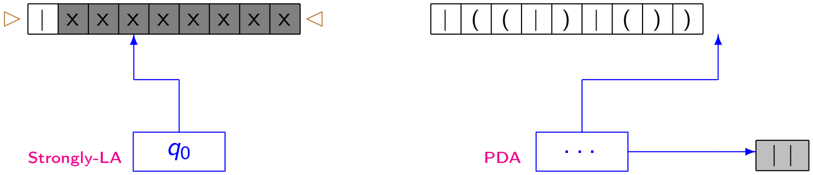
Simulation of Strongly Limited Automata by PDAs



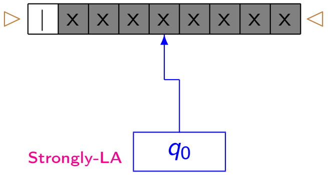
Simulation of Strongly Limited Automata by PDAs

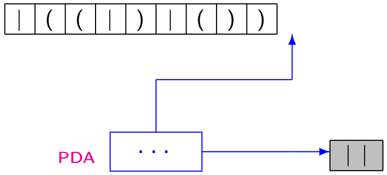


Simulation of Strongly Limited Automata by PDAs

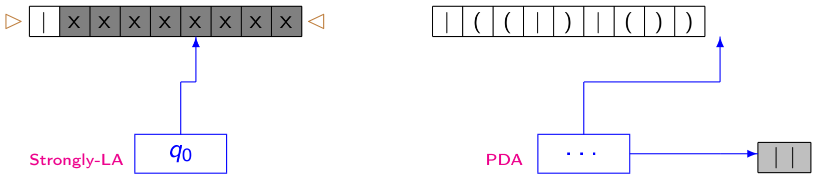


Simulation of Strongly Limited Automata by PDAs

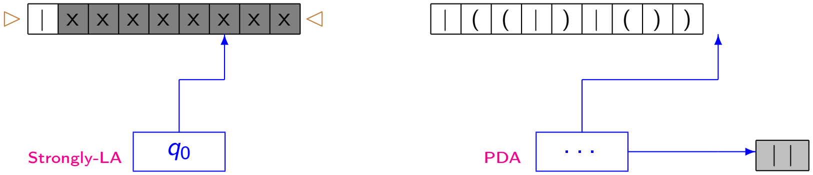




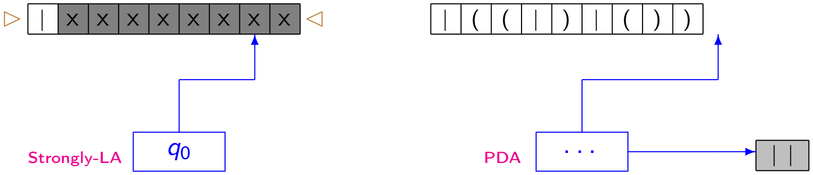
Simulation of Strongly Limited Automata by PDAs



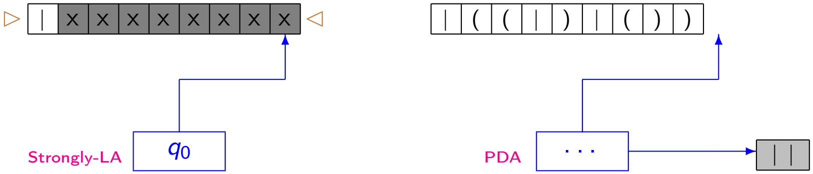
Simulation of Strongly Limited Automata by PDAs



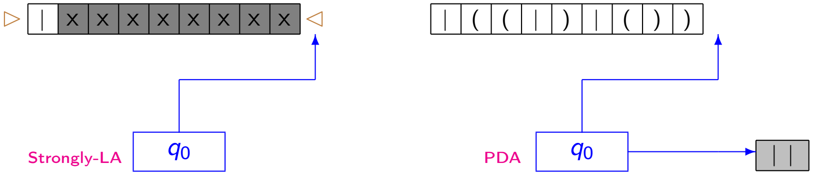
Simulation of Strongly Limited Automata by PDAs



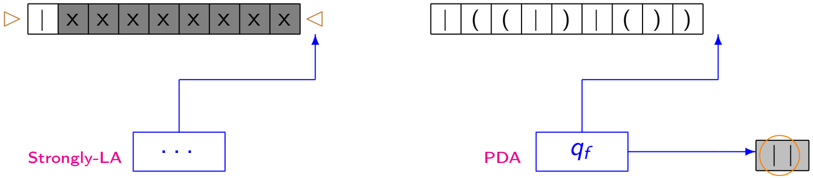
Simulation of Strongly Limited Automata by PDAs



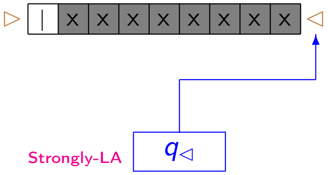
Simulation of Strongly Limited Automata by PDAs



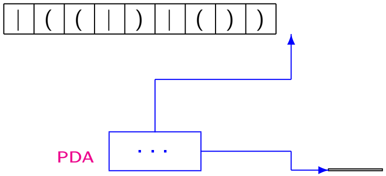
Simulation of Strongly Limited Automata by PDAs



Simulation of Strongly Limited Automata by PDAs

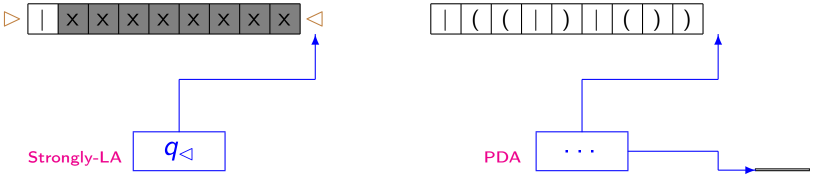


final scan from right to left



final scan already simulated

Simulation of Strongly Limited Automata by PDAs



The description of the resulting PDA has polynomial size wrt that of the given strongly limited automaton

Summing up...

- Descriptional complexity
  - Strongly limited automata
  - Context-free grammars
  - Pushdown automata

  are polynomially related in size

- 2-limited automata can be exponentially smaller [P&Pisoni'13]

Strongly Limited Automata vs Forgetting Automata

- Strongly limited automata can use different symbols to rewrite tape cells, e.g., $\{ww^R \mid w \in \{a,b\}^*$ does not contain two consecutive bs$\}$

Problem

Which class of languages is accepted by strongly limited automata that can use only one fixed symbol for rewriting?

- Forgetting Automata [Jancar&Mráz&Plátek '96]:
  - only one fixed symbol for rewriting
  - tape changes only in 1st or 2nd visit
  - no restrictions on head reversals and state changes
  - accept exactly CFLs

Problem

Study the descriptional complexity of forgetting automata

Strongly Limited Automata vs Forgetting Automata

- Strongly limited automata can use different symbols to rewrite tape cells, e.g., $\{ww^R \mid w \in \{a,b\}^*$ does not contain two consecutive bs$\}$

Problem

Which class of languages is accepted by strongly limited automata that can use only one fixed symbol for rewriting?

- Forgetting Automata [Jancar&Mráz&Plátek '96]:
  - only one fixed symbol for rewriting
  - tape changes only in 1st or 2nd visit
  - no restrictions on head reversals and state changes
  - accept exactly CFLs

Problem

Study the descriptional complexity of forgetting automata

Strongly Limited Automata vs Forgetting Automata

- Strongly limited automata can use different symbols to rewrite tape cells, e.g., $\{ww^R \mid w \in \{a,b\}^*$ does not contain two consecutive bs$\}$

Problem

Which class of languages is accepted by strongly limited automata that can use only one fixed symbol for rewriting?

- Forgetting Automata [Jancar&Mráz&Plátek '96]:
  - only one fixed symbol for rewriting
  - tape changes only in 1st or 2nd visit
  - no restrictions on head reversals and state changes
  - accept exactly CFLs

Problem

Study the descriptional complexity of forgetting automata

Strongly Limited Automata vs Forgetting Automata

- Strongly limited automata can use different symbols to rewrite tape cells, e.g., $\{ww^R \mid w \in \{a,b\}^*$ does not contain two consecutive bs$\}$

Problem

Which class of languages is accepted by strongly limited automata that can use only one fixed symbol for rewriting?

- Forgetting Automata [Jancar&Mráz&Plátek '96]:
  - only one fixed symbol for rewriting
  - tape changes only in 1st or 2nd visit
  - no restrictions on head reversals and state changes
  - accept exactly CFLs

Problem

Study the descriptional complexity of forgetting automata

Determinism vs Nondeterminism

- The conversion from CFGs to strongly limited automata uses nondeterminism
- 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

- The conversion from CFGs to strongly limited automata uses nondeterminism
- 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

- The conversion from CFGs to strongly limited automata uses nondeterminism
- 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*

Problem

Which class of languages is accepted
by deterministic strongly limited automata?

Determinism vs Nondeterminism

- The conversion from CFGs to strongly limited automata uses nondeterminism
- 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*
- Moving to the right only q_0 is used

A possible modification:
a set of states Q_R (rewritten cells still ignored)

- the simulation by PDAs remains polynomial
- languages L_1 and L_2 are accepted by *deterministic devices*

Problem

Which class of languages is accepted by the deterministic version of devices so modified?

Determinism vs Nondeterminism

- The conversion from CFGs to strongly limited automata uses nondeterminism
- 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*
- Moving to the right only q_0 is used

  A possible modification:
  a set of states Q_R (rewritten cells still ignored)
  - the simulation by PDAs remains polynomial
  - languages L_1 and L_2 are accepted by *deterministic devices*

Problem

Which class of languages is accepted
by the deterministic version of devices so modified?

Determinism vs Nondeterminism

- The conversion from CFGs to strongly limited automata uses nondeterminism
- 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*
- Moving to the right only q_0 is used

  A possible modification:
  a set of states Q_R (rewritten cells still ignored)
  - the simulation by PDAs remains polynomial
  - languages L_1 and L_2 are accepted by *deterministic devices*

Problem

Which class of languages is accepted
by the deterministic version of devices so modified?

Thank you for your attention!