Parsing Expression Grammars

Overview & an Equivalent Computational Model

Alexander Rubtsov

HSE University, Moscow, Russia

Based on MFCS 2024 paper
(joint work with Nikita Chudinov)
September 26, 2024

Parsing Expression Grammars

Parsing Expression Grammars

Motivation, Definition, Examples

Practical Motivation

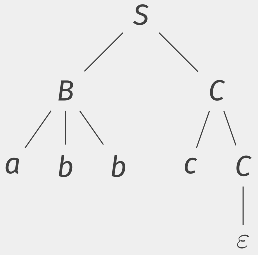
Classical Parsing Approaches

- Top-Down: LL(k)
  easy to design, but the power is limited
- Bottom-Up: LR(k)
  powerful (generate all DCFLs), but hard to design

Parsing Expression Grammars

- PEGs can be considered as a generalization of LL-grammars
- PEGs are more powerful than LR-grammars, but there is no (currently?) direct translation LR-grammars to PEGs
- PEGs now being used in compilers
  Python replaced LL(1)-parser by PEG
- PEGs are popular for solving parsing problems

PEGs as LL(k)-generalization



a	b	b	c	$\triangleleft$

LL(1)-Grammar:

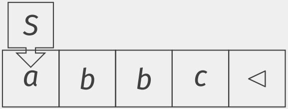
$$S \rightarrow BC$$
$$B \rightarrow abb \mid b$$
$$C \rightarrow cC \mid \varepsilon$$

PEGs as LL(k)-generalization

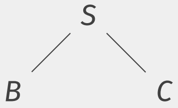
S

LL(1)-Grammar:

$$S \to BC$$
$$B \to abb \mid b$$
$$C \to cC \mid \varepsilon$$

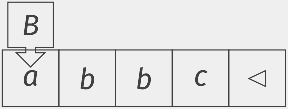


PEGs as LL(k)-generalization

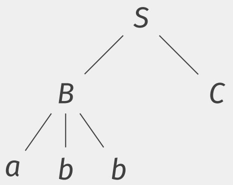


LL(1)-Grammar:

$$S \rightarrow BC$$
$$B \rightarrow abb \mid b$$
$$C \rightarrow cC \mid \varepsilon$$

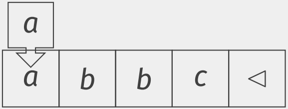


PEGs as LL(k)-generalization

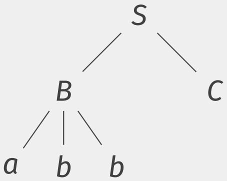


LL(1)-Grammar:

$$S \rightarrow BC$$
$$B \rightarrow abb \mid b$$
$$C \rightarrow cC \mid \varepsilon$$

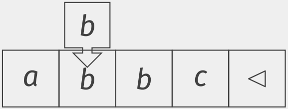


PEGs as LL(k)-generalization

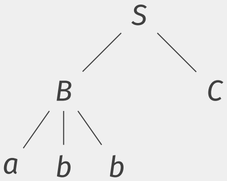


LL(1)-Grammar:

$$S \rightarrow BC$$
$$B \rightarrow abb \mid b$$
$$C \rightarrow cC \mid \varepsilon$$

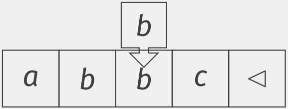


PEGs as LL(k)-generalization

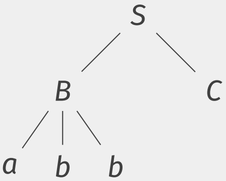


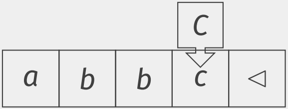
LL(1)-Grammar:

$$S \to BC$$
$$B \to abb \mid b$$
$$C \to cC \mid \varepsilon$$



PEGs as LL(k)-generalization

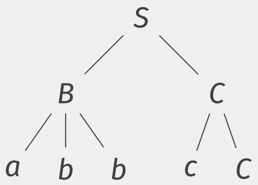




LL(1)-Grammar:

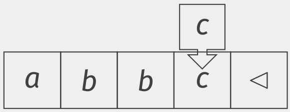
$$S \rightarrow BC$$
$$B \rightarrow abb \mid b$$
$$C \rightarrow cC \mid \varepsilon$$

PEGs as LL(k)-generalization

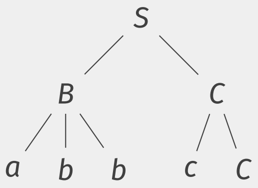


LL(1)-Grammar:

$$S \to BC$$
$$B \to abb \mid b$$
$$C \to cC \mid \varepsilon$$

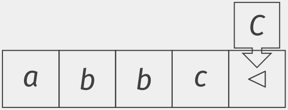


PEGs as LL(k)-generalization

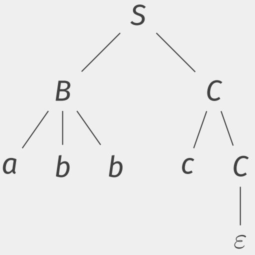


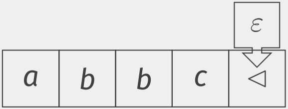
LL(1)-Grammar:

$$S \to BC$$
$$B \to abb \mid b$$
$$C \to cC \mid \varepsilon$$



PEGs as LL(k)-generalization





LL(1)-Grammar:

$$S \to BC$$
$$B \to abb \mid b$$
$$C \to cC \mid \varepsilon$$

PEGs as LL(k)-generalization

CFG

$$S \to AB \mid BC$$
$$A \to aA \mid a$$
$$B \to abb \mid b$$
$$C \to cC \mid \varepsilon$$

PEG

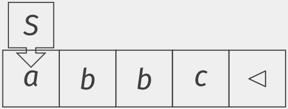
$$S \leftarrow AB \mathbin{/} BC$$
$$A \leftarrow aA \mathbin{/} a$$
$$B \leftarrow abb \mathbin{/} b$$
$$C \leftarrow cC \mathbin{/} \varepsilon$$

PEGs as LL(k)-generalization

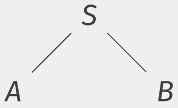
S

PEG:

$$S \leftarrow AB \;/\; BC$$
$$A \leftarrow aA \;/\; a$$
$$B \leftarrow abb \;/\; b$$
$$C \leftarrow cC \;/\; \varepsilon$$

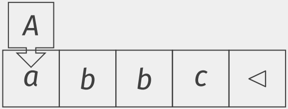


PEGs as LL(k)-generalization

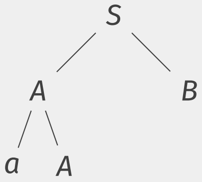


PEG:

$$S \leftarrow AB \,/\, BC$$
$$A \leftarrow aA \,/\, a$$
$$B \leftarrow abb \,/\, b$$
$$C \leftarrow cC \,/\, \varepsilon$$

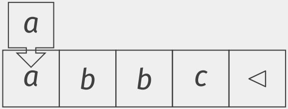


PEGs as LL(k)-generalization

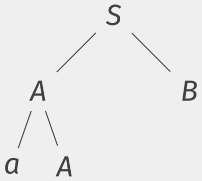


PEG:

$$S \leftarrow AB \ / \ BC$$
$$A \leftarrow aA \ / \ a$$
$$B \leftarrow abb \ / \ b$$
$$C \leftarrow cC \ / \ \varepsilon$$

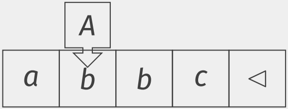


PEGs as LL(k)-generalization

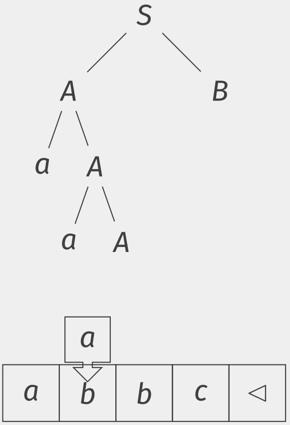


PEG:

$$S \leftarrow AB \mathbin{/} BC$$
$$A \leftarrow aA \mathbin{/} a$$
$$B \leftarrow abb \mathbin{/} b$$
$$C \leftarrow cC \mathbin{/} \varepsilon$$



PEGs as LL(k)-generalization



PEG:

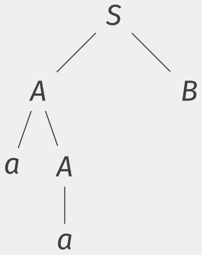
$S \leftarrow AB \;/\; BC$

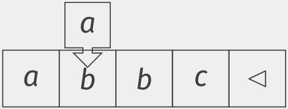
$A \leftarrow aA \;/\; a$

$B \leftarrow abb \;/\; b$

$C \leftarrow cC \;/\; \varepsilon$

PEGs as LL(k)-generalization

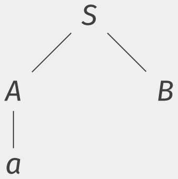


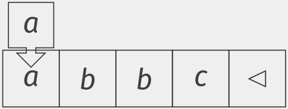


PEG:

$$S \leftarrow AB \;/\; BC$$
$$A \leftarrow aA \;/\; a$$
$$B \leftarrow abb \;/\; b$$
$$C \leftarrow cC \;/\; \varepsilon$$

PEGs as LL(k)-generalization

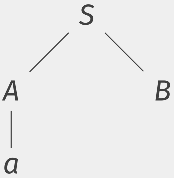




PEG:

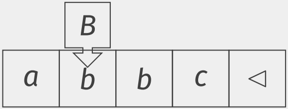
$$S \leftarrow AB \,/\, BC$$
$$A \leftarrow aA \,/\, a$$
$$B \leftarrow abb \,/\, b$$
$$C \leftarrow cC \,/\, \varepsilon$$

PEGs as LL(k)-generalization

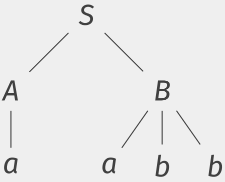


PEG:

$$S \leftarrow AB \;/\; BC$$
$$A \leftarrow aA \;/\; a$$
$$B \leftarrow abb \;/\; b$$
$$C \leftarrow cC \;/\; \varepsilon$$

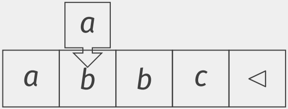


PEGs as LL(k)-generalization

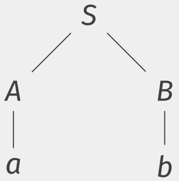


PEG:

$$S \leftarrow AB \,/\, BC$$
$$A \leftarrow aA \,/\, a$$
$$B \leftarrow abb \,/\, b$$
$$C \leftarrow cC \,/\, \varepsilon$$

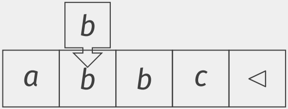


PEGs as LL(k)-generalization

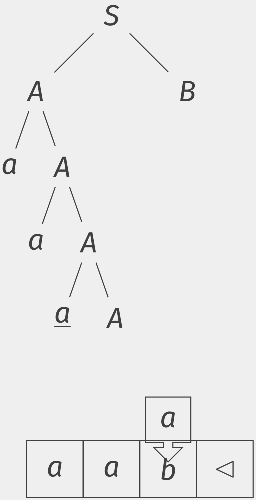


PEG:

$$S \leftarrow AB \,/\, BC$$
$$A \leftarrow aA \,/\, a$$
$$B \leftarrow abb \,/\, b$$
$$C \leftarrow cC \,/\, \varepsilon$$

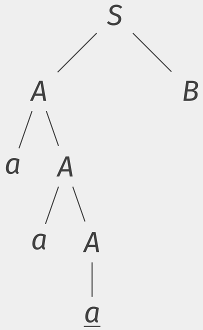


PEGs as LL(k)-generalization



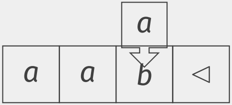
PEG:

$$S \leftarrow AB \,/\, BC$$
$$A \leftarrow aA \,/\, a$$
$$B \leftarrow abb \,/\, b$$
$$C \leftarrow cC \,/\, \varepsilon$$

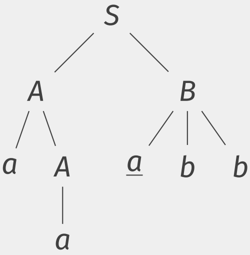


PEG:

$$S \leftarrow AB \,/\, BC$$
$$A \leftarrow aA \,/\, a$$
$$B \leftarrow abb \,/\, b$$
$$C \leftarrow cC \,/\, \varepsilon$$

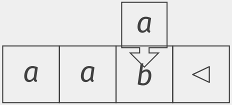


PEGs as LL(k)-generalization

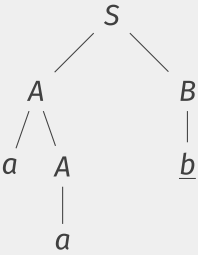


PEG:

$$S \leftarrow AB \;/\; BC$$
$$A \leftarrow aA \;/\; a$$
$$B \leftarrow abb \;/\; b$$
$$C \leftarrow cC \;/\; \varepsilon$$

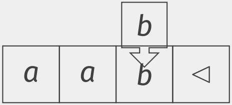


PEGs as LL(k)-generalization

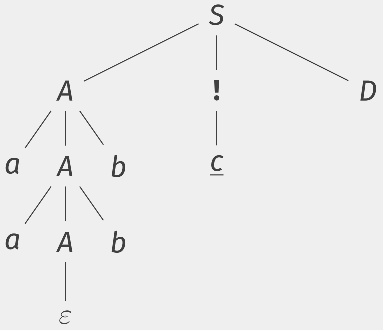


PEG:

$$S \leftarrow AB \;/\; BC$$
$$A \leftarrow aA \;/\; a$$
$$B \leftarrow abb \;/\; b$$
$$C \leftarrow cC \;/\; \varepsilon$$

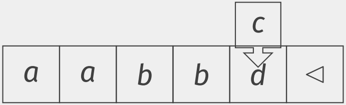


PEGS EXAMPLES: OPERATOR !



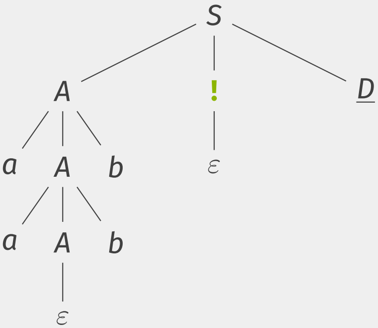
PEG:

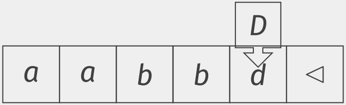
$$S \leftarrow A(!c)D \;/\; A'B$$
$$A \leftarrow aAb \;/\; \varepsilon$$
$$A' \leftarrow aA' \;/\; \varepsilon$$
$$B \leftarrow bBc \;/\; \varepsilon$$
$$D \leftarrow dD \;/\; \varepsilon$$



$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator !



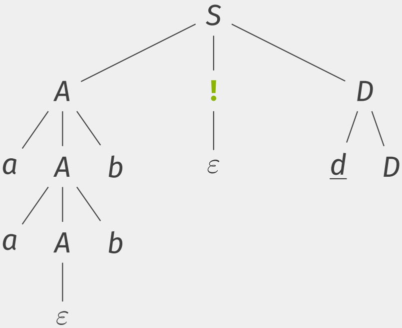


PEG:

$$S \leftarrow A(!c)D \,/\, A'B$$
$$A \leftarrow aAb \,/\, \varepsilon$$
$$A' \leftarrow aA' \,/\, \varepsilon$$
$$B \leftarrow bBc \,/\, \varepsilon$$
$$D \leftarrow dD \,/\, \varepsilon$$

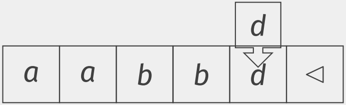
$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator !



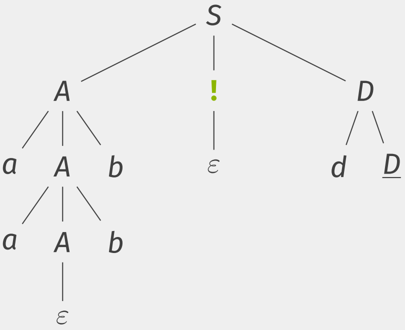
PEG:

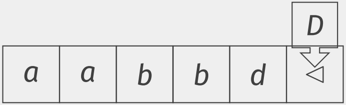
$$
\begin{aligned}
S &\leftarrow A(!c)D \,/\, A'B \\
A &\leftarrow aAb \,/\, \varepsilon \\
A' &\leftarrow aA' \,/\, \varepsilon \\
B &\leftarrow bBc \,/\, \varepsilon \\
D &\leftarrow dD \,/\, \varepsilon
\end{aligned}
$$



$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator !



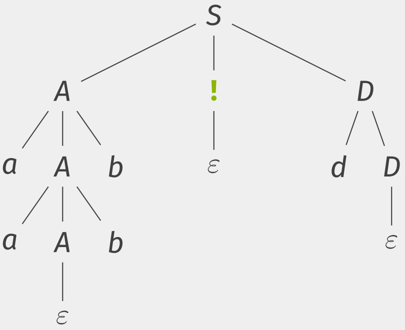


PEG:

$$S \leftarrow A(!c)D \;/\; A'B$$
$$A \leftarrow aAb \;/\; \varepsilon$$
$$A' \leftarrow aA' \;/\; \varepsilon$$
$$B \leftarrow bBc \;/\; \varepsilon$$
$$D \leftarrow dD \;/\; \varepsilon$$

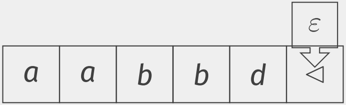
$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator !



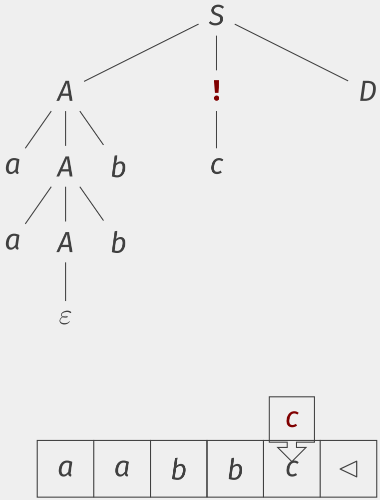
PEG:

$$S \leftarrow A(!c)D \,/\, A'B$$
$$A \leftarrow aAb \,/\, \varepsilon$$
$$A' \leftarrow aA' \,/\, \varepsilon$$
$$B \leftarrow bBc \,/\, \varepsilon$$
$$D \leftarrow dD \,/\, \varepsilon$$



$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator !

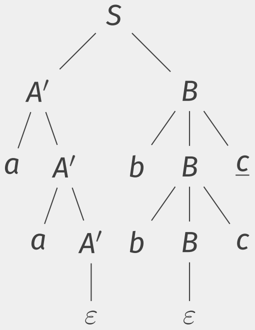


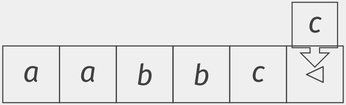
PEG:

$$S \leftarrow A(!c)D \;/\; A'B$$
$$A \leftarrow aAb \;/\; \varepsilon$$
$$A' \leftarrow aA' \;/\; \varepsilon$$
$$B \leftarrow bBc \;/\; \varepsilon$$
$$D \leftarrow dD \;/\; \varepsilon$$

$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator !





PEG:

$$S \leftarrow A(!c)D \;/\; A'B$$
$$A \leftarrow aAb \;/\; \varepsilon$$
$$A' \leftarrow aA' \;/\; \varepsilon$$
$$B \leftarrow bBc \;/\; \varepsilon$$
$$D \leftarrow dD \;/\; \varepsilon$$

$$a^n b^n d^* \cup a^* b^n c^n$$

PEGs Examples: operator &

& is a syntactic sugar

$$\&X = \mathbf{!}(\mathbf{!}X)$$

- if X yields to fail then $\mathbf{!}X$ yields to ε
- if X does not yield to fail then $\&X$ yields to ε

$$G : S \leftarrow (\&(Ac))BC \quad A \leftarrow aAb \,/\, \varepsilon \quad B \leftarrow aB \,/\, a \quad C \leftarrow bCc \,/\, \varepsilon$$

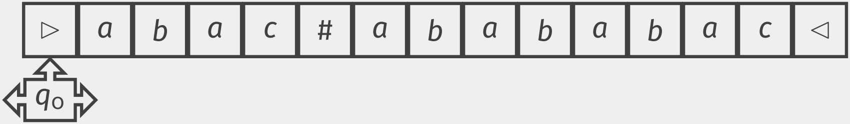
$$L(G) = \{a^n b^n c^n \mid n \geq 1\}$$

Discussion

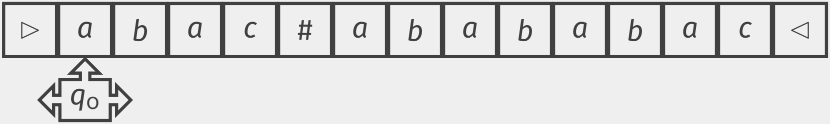
- "Concatenation" in PEGs is not a real concatenation!
  - Conjecture: PELs are not closed over concatenation
- PELs are closed over Boolean operations: $\Gamma_{\mathrm{Bool}}(\mathrm{PEL}) = \mathrm{PEL}$
- $\mathrm{PEL} \setminus \mathrm{CFL} \neq \varnothing$
- *Open question:* $\mathrm{CFL} \setminus \mathrm{PEL} \stackrel{?}{=} \varnothing$
  - $\Omega(n^{1+\varepsilon})$ bound on CFLs parsing implies $\mathrm{CFL} \setminus \mathrm{PEL} \neq \varnothing$

Pointer Pushdown Automata

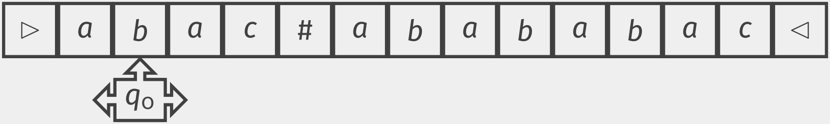
Pointer Pushdown Automata



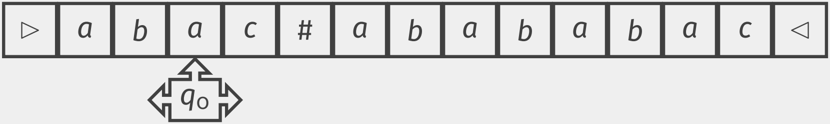
Pointer Pushdown Automata



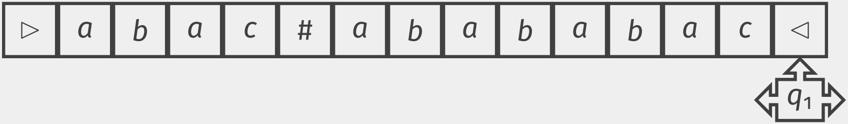
Pointer Pushdown Automata



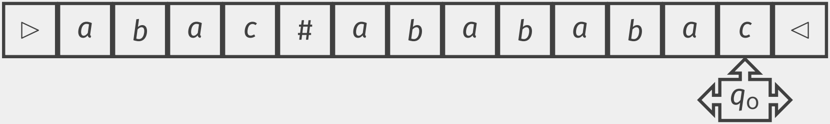
Pointer Pushdown Automata



Pointer Pushdown Automata



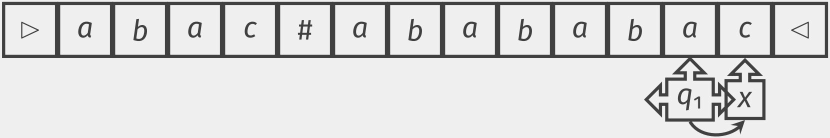
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$

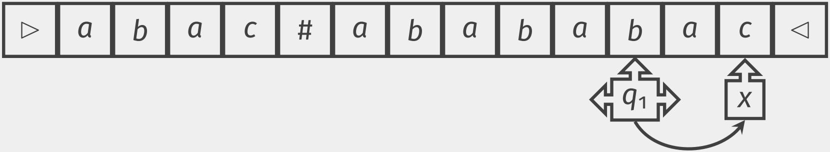
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$

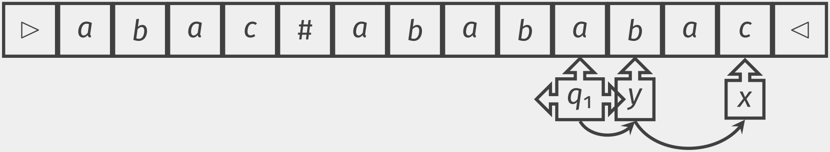
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$

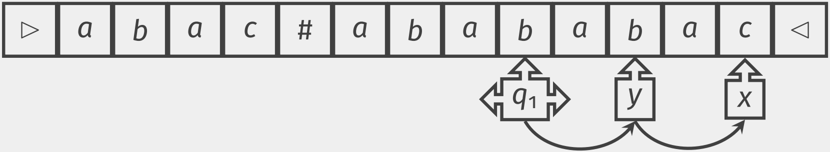
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$

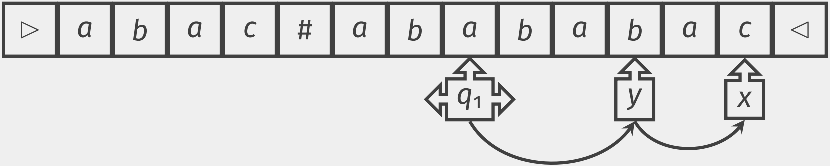
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$

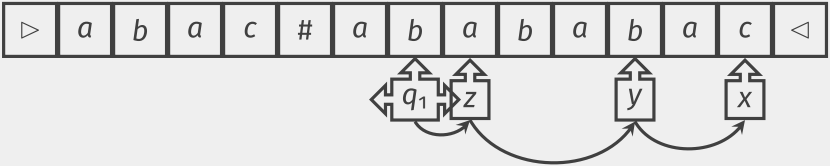
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$
- push(z) $\leftarrow$

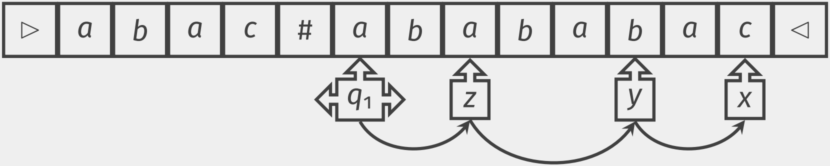
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$
- push(z) $\leftarrow$

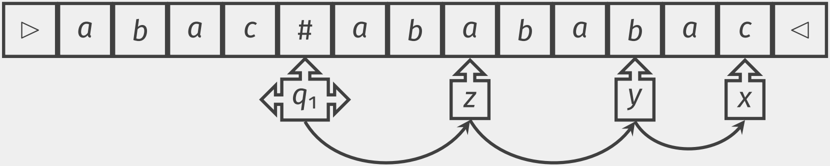
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$
- push(z) $\leftarrow$

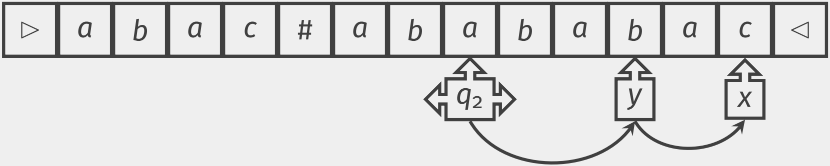
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$
- push(z) $\leftarrow$
- pop(z) $\uparrow$

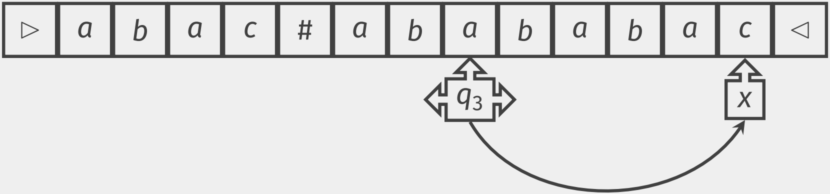
Pointer Pushdown Automata



Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$
- push(z) $\leftarrow$
- pop(z) $\uparrow$
- pop(y) $\downarrow$

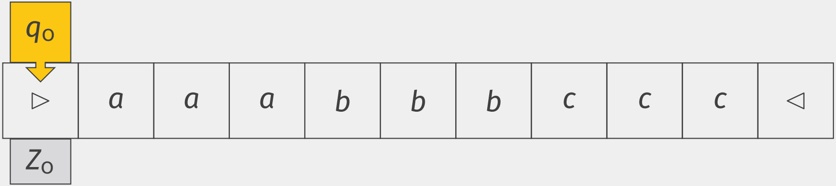
Pointer Pushdown Automata



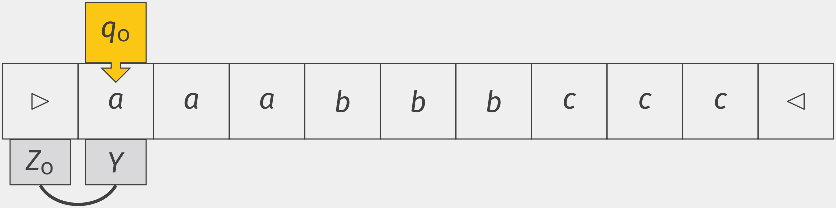
Stack Operations:

- push(x) $\leftarrow$
- push(y) $\leftarrow$
- push(z) $\leftarrow$
- pop(z) $\uparrow$
- pop(y) $\downarrow$

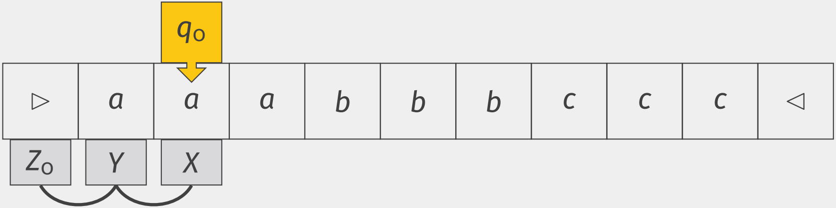
$a^n b^n c^n$



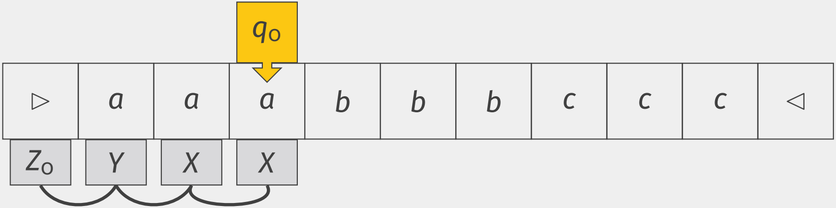
$a^n b^n c^n$



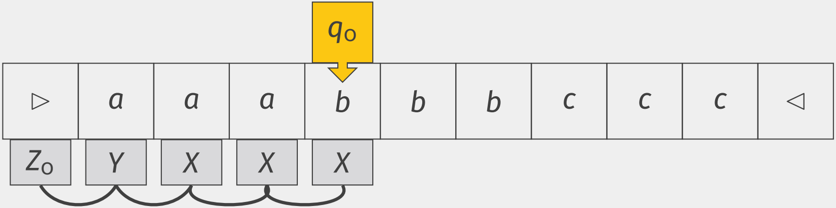
$a^n b^n c^n$



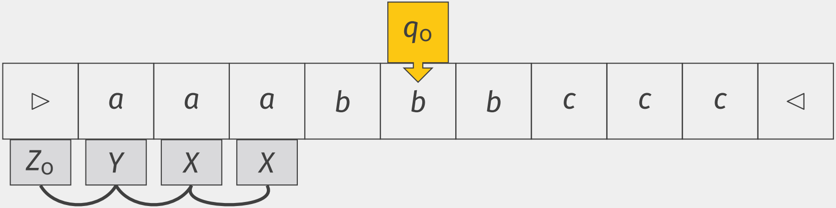
$a^n b^n c^n$



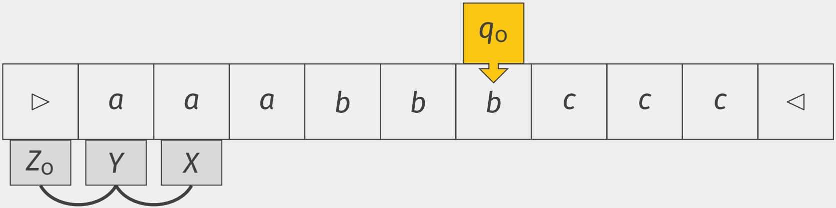
$a^n b^n c^n$



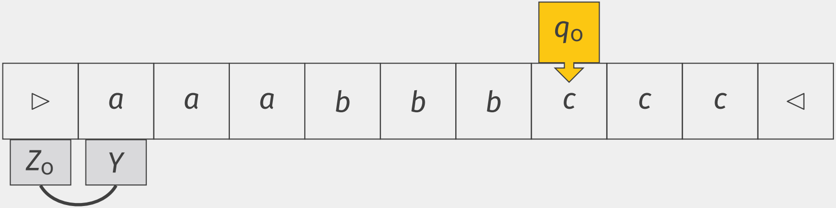
$a^n b^n c^n$



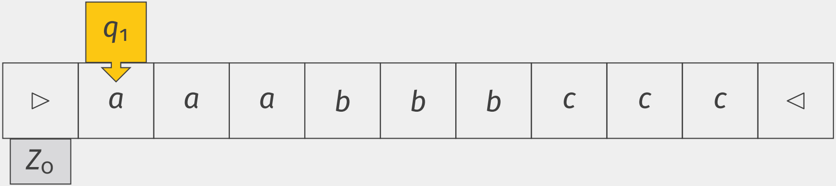
$a^n b^n c^n$



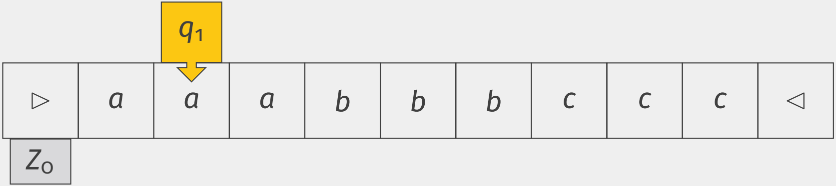
$a^n b^n c^n$



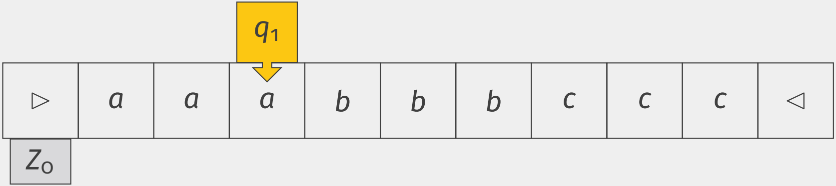
$a^n b^n c^n$



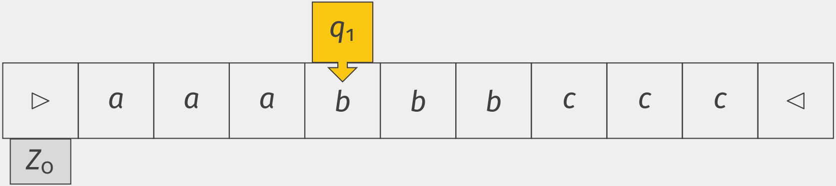
$a^n b^n c^n$



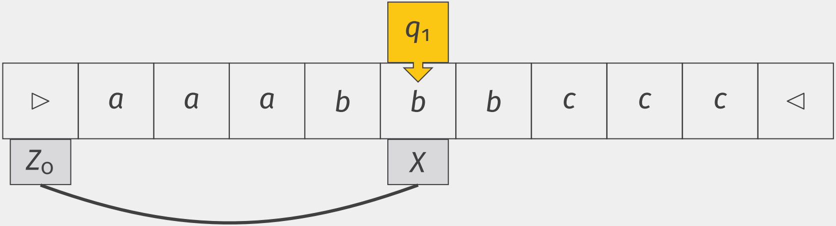
$a^n b^n c^n$



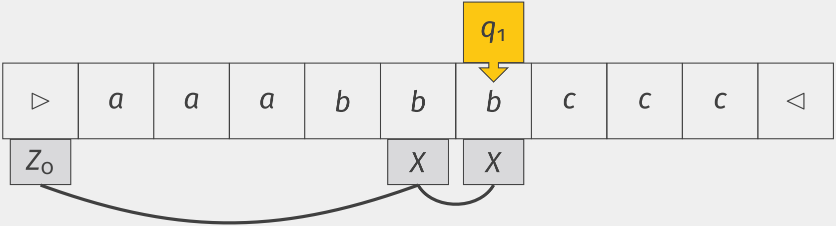
$a^n b^n c^n$



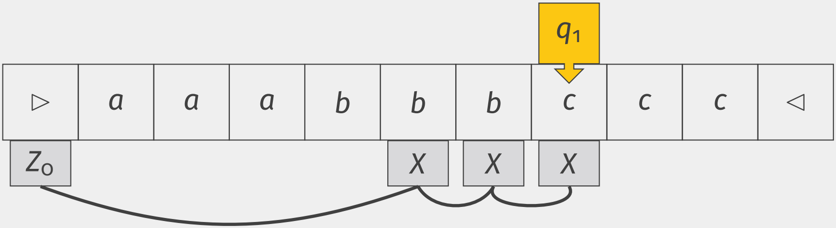
$a^n b^n c^n$



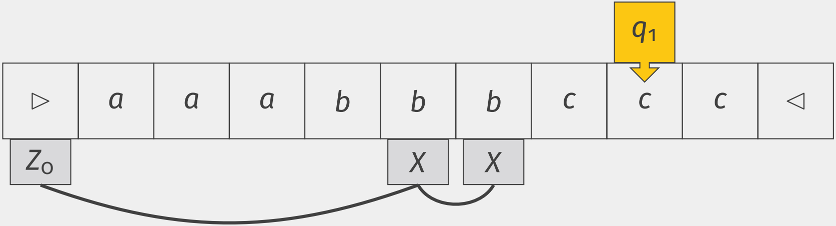
$a^n b^n c^n$



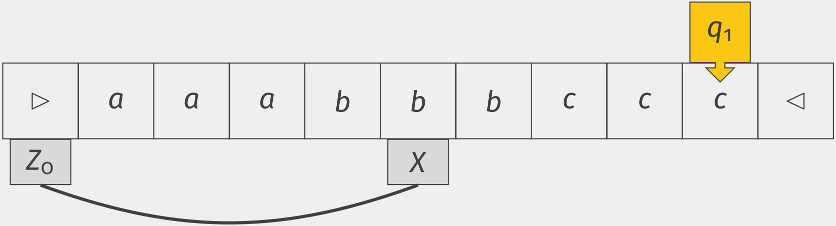
$a^n b^n c^n$



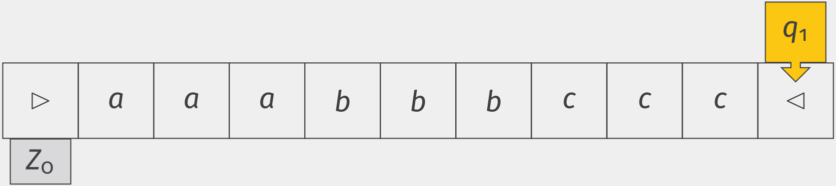
$a^n b^n c^n$



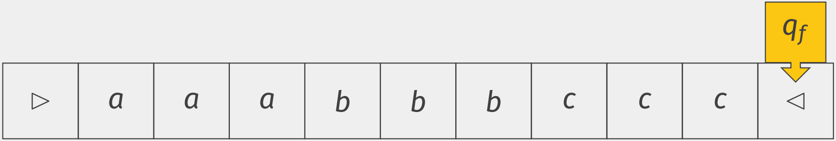
$a^n b^n c^n$



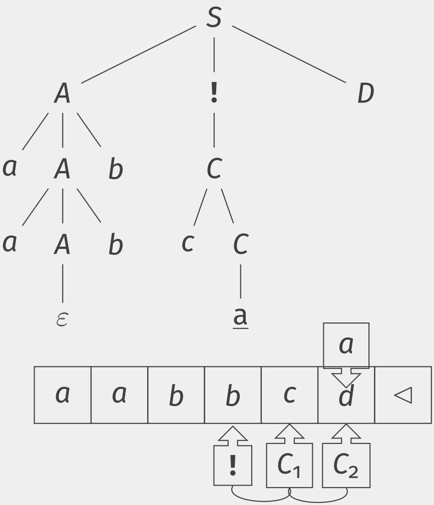
$a^n b^n c^n$



$a^n b^n c^n$



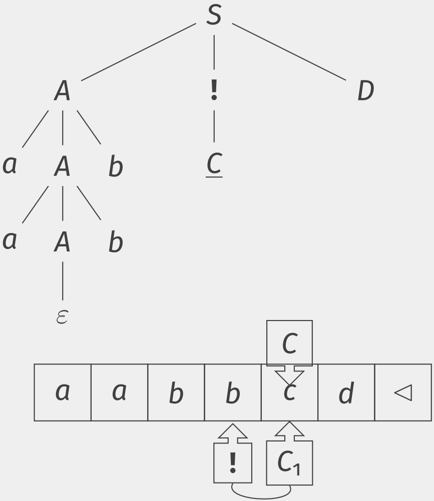
PEG → DPPDA (IDEA)



PEG:

$$S \leftarrow A(!C)D \,/\, A'B$$
$$A \leftarrow aAb \,/\, \varepsilon$$
$$A' \leftarrow aA' \,/\, \varepsilon$$
$$B \leftarrow bBc \,/\, \varepsilon$$
$$C \leftarrow cC \,/\, a$$
$$D \leftarrow dD \,/\, \varepsilon$$

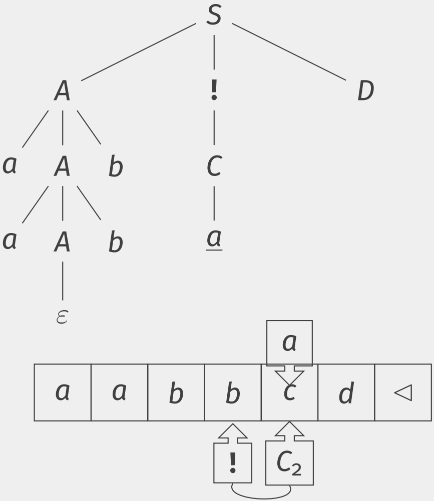
PEG $\to$ DPPDA (IDEA)



PEG:

$$S \leftarrow A(!C)D \,/\, A'B$$
$$A \leftarrow aAb \,/\, \varepsilon$$
$$A' \leftarrow aA' \,/\, \varepsilon$$
$$B \leftarrow bBc \,/\, \varepsilon$$
$$C \leftarrow cC \,/\, a$$
$$D \leftarrow dD \,/\, \varepsilon$$

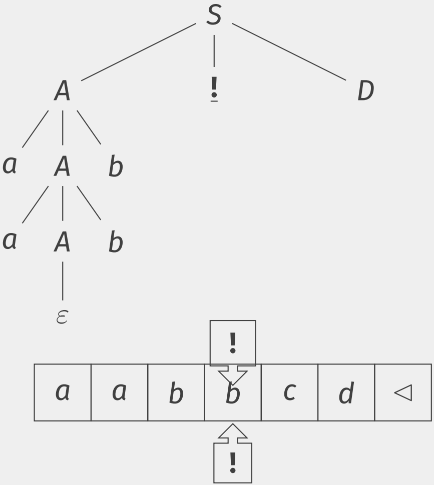
PEG $\to$ DPPDA (IDEA)



PEG:

$$S \leftarrow A(!C)D \,/\, A'B$$
$$A \leftarrow aAb \,/\, \varepsilon$$
$$A' \leftarrow aA' \,/\, \varepsilon$$
$$B \leftarrow bBc \,/\, \varepsilon$$
$$C \leftarrow cC \,/\, a$$
$$D \leftarrow dD \,/\, \varepsilon$$

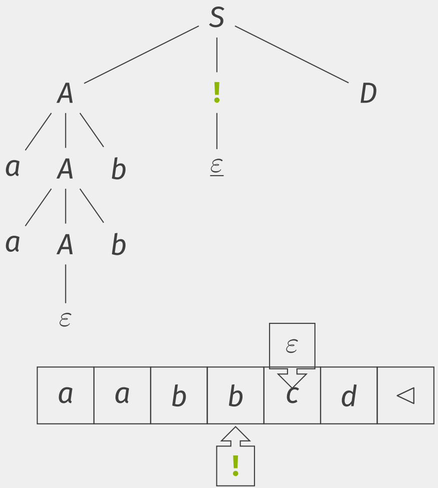
PEG $\rightarrow$ DPPDA (idea)



PEG:

$$S \leftarrow A(\mathbf{!}C)D \;/\; A'B$$
$$A \leftarrow aAb \;/\; \varepsilon$$
$$A' \leftarrow aA' \;/\; \varepsilon$$
$$B \leftarrow bBc \;/\; \varepsilon$$
$$C \leftarrow cC \;/\; a$$
$$D \leftarrow dD \;/\; \varepsilon$$

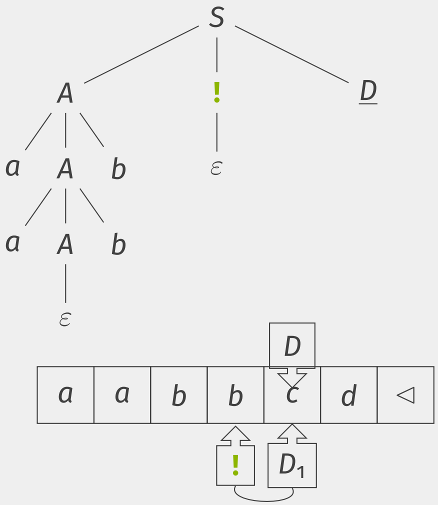
PEG → DPPDA (IDEA)



PEG:

$$S \leftarrow A(!C)D \mathbin{/} A'B$$
$$A \leftarrow aAb \mathbin{/} \varepsilon$$
$$A' \leftarrow aA' \mathbin{/} \varepsilon$$
$$B \leftarrow bBc \mathbin{/} \varepsilon$$
$$C \leftarrow cC \mathbin{/} a$$
$$D \leftarrow dD \mathbin{/} \varepsilon$$

PEG → DPPDA (idea)

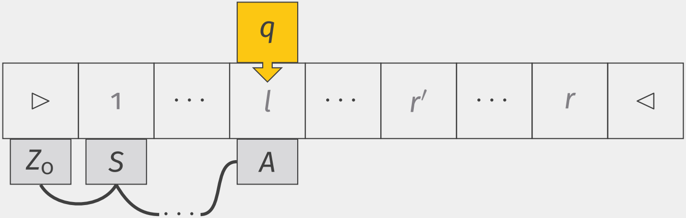


PEG:

$$S \leftarrow A(!C)D \,/\, A'B$$
$$A \leftarrow aAb \,/\, \varepsilon$$
$$A' \leftarrow aA' \,/\, \varepsilon$$
$$B \leftarrow bBc \,/\, \varepsilon$$
$$C \leftarrow cC \,/\, a$$
$$D \leftarrow dD \,/\, \varepsilon$$

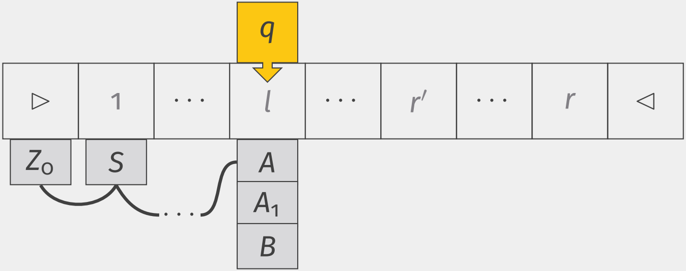
PEG $\to$ DPPDA (PART OF CONSTRUCTION)

$$A \leftarrow BC$$



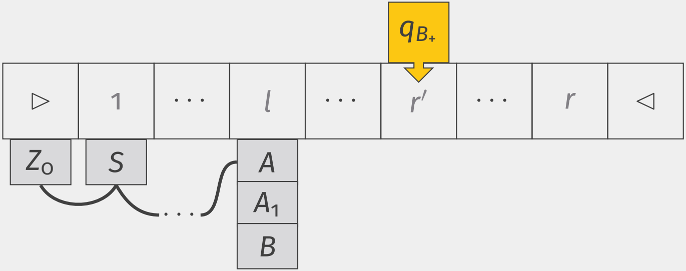
PEG $\to$ DPPDA (PART OF CONSTRUCTION)

$$A \leftarrow BC$$



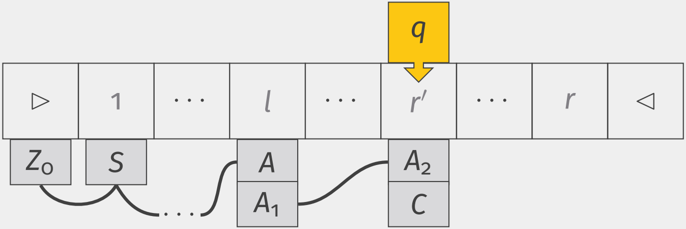
PEG $\to$ DPPDA (part of construction)

$$A \leftarrow BC$$



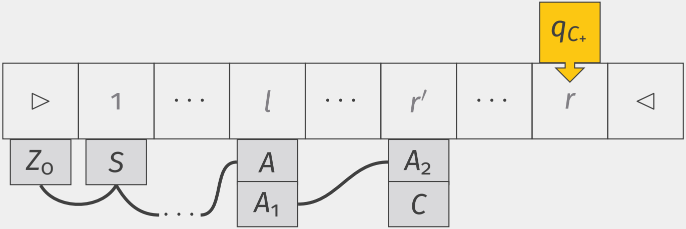
PEG $\to$ DPPDA (part of construction)

$$A \leftarrow BC$$



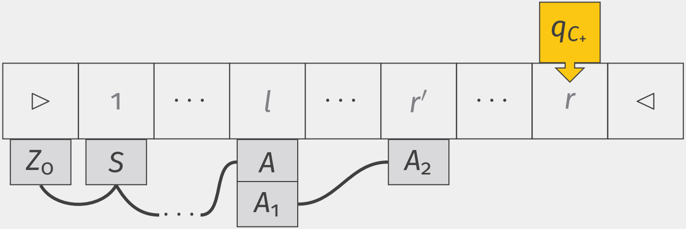
PEG $\rightarrow$ DPPDA (PART OF CONSTRUCTION)

$A \leftarrow BC$



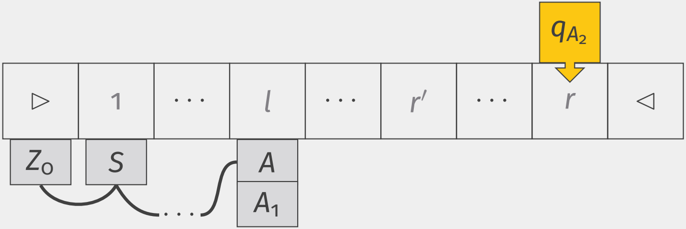
PEG $\rightarrow$ DPPDA (PART OF CONSTRUCTION)

$$A \leftarrow BC$$



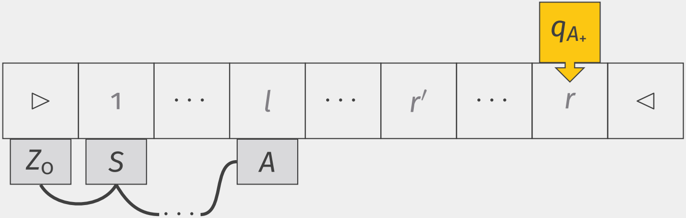
PEG $\to$ DPPDA (PART OF CONSTRUCTION)

$$A \leftarrow BC$$

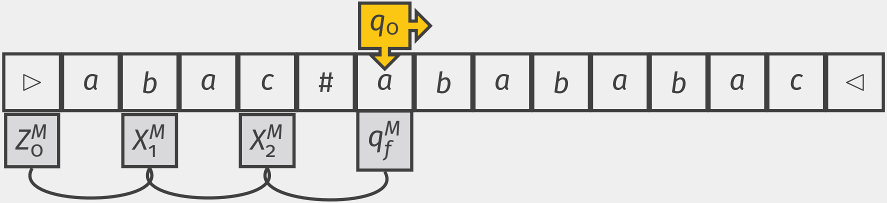


PEG $\to$ DPPDA (PART OF CONSTRUCTION)

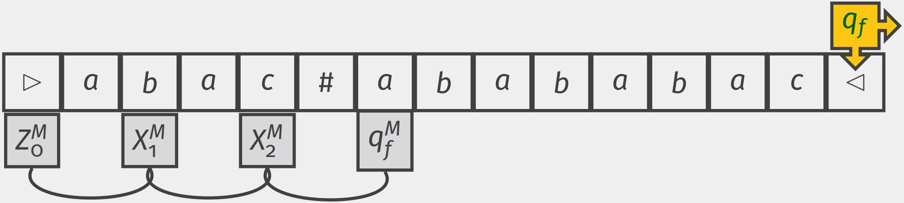
$$A \leftarrow BC$$



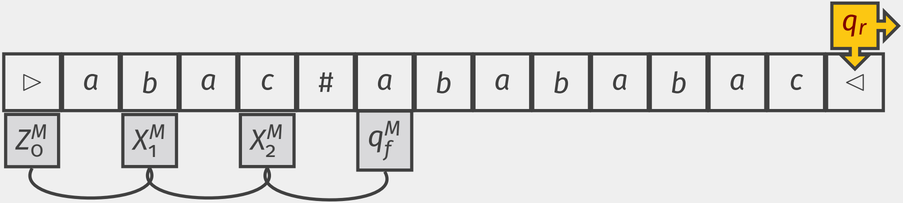
DCFL · PEL = PEL



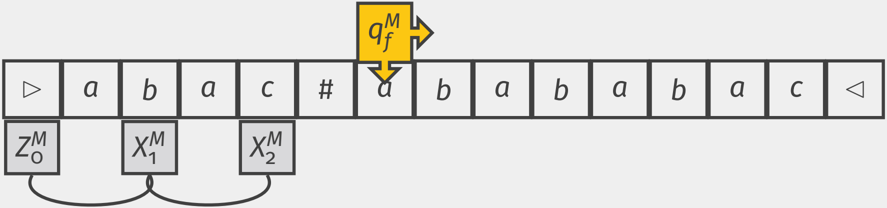
DCFL · PEL = PEL



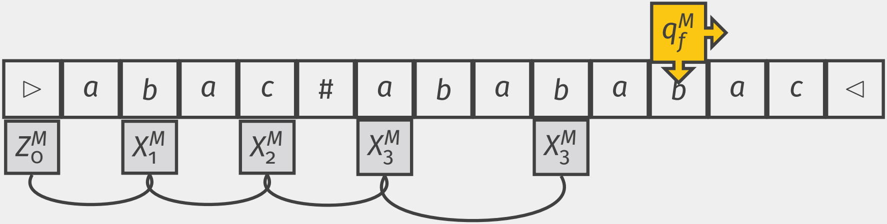
DCFL · PEL = PEL



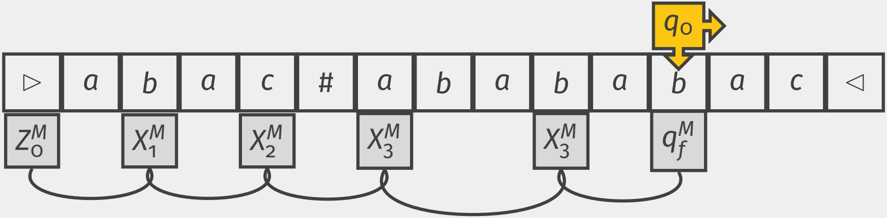
DCFL · PEL = PEL



DCFL · PEL = PEL



DCFL · PEL = PEL



Theorem

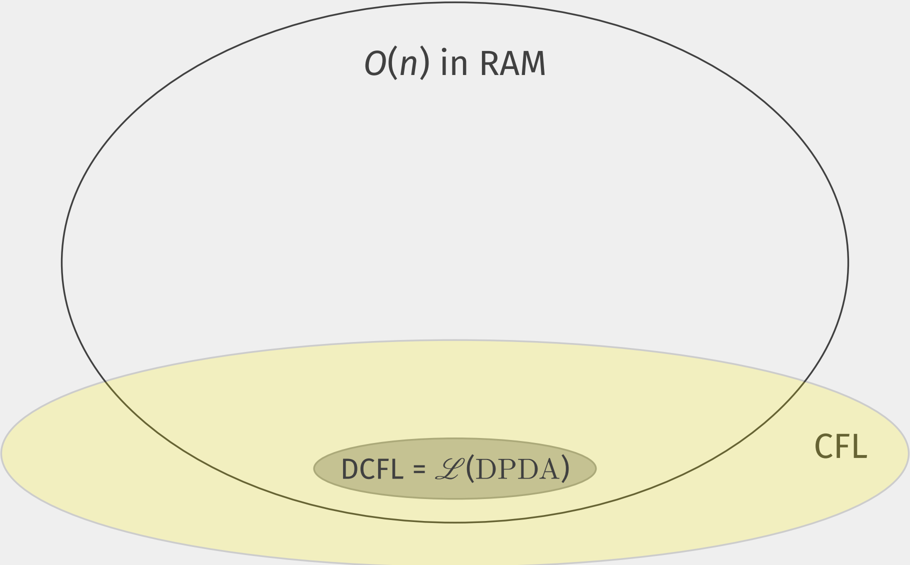
$\Gamma_{\mathrm{REG}}(\mathrm{DCFL}) \cdot \mathrm{PEL} = \mathrm{PEL}$

$$\psi : a(a|bc^*)^*b \mapsto L_a(L_a|L_bL_c^*)^*L_b, \text{ where } L_a, L_b, L_c \in \mathrm{DCFL}$$

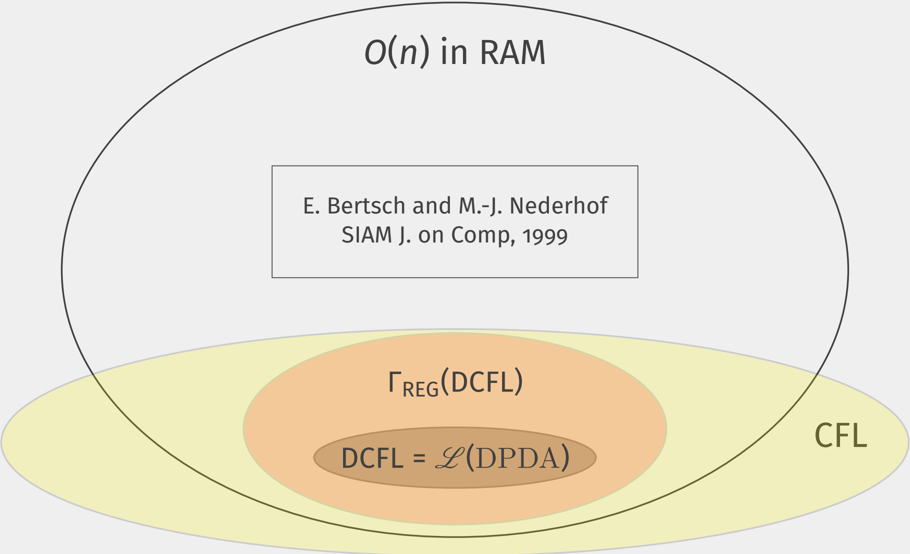
$$\Gamma_{\mathrm{REG}}(\mathrm{DCFL}) = \bigcup_{\psi,R} \psi(R)$$

Overview of Results on Computational Complexity

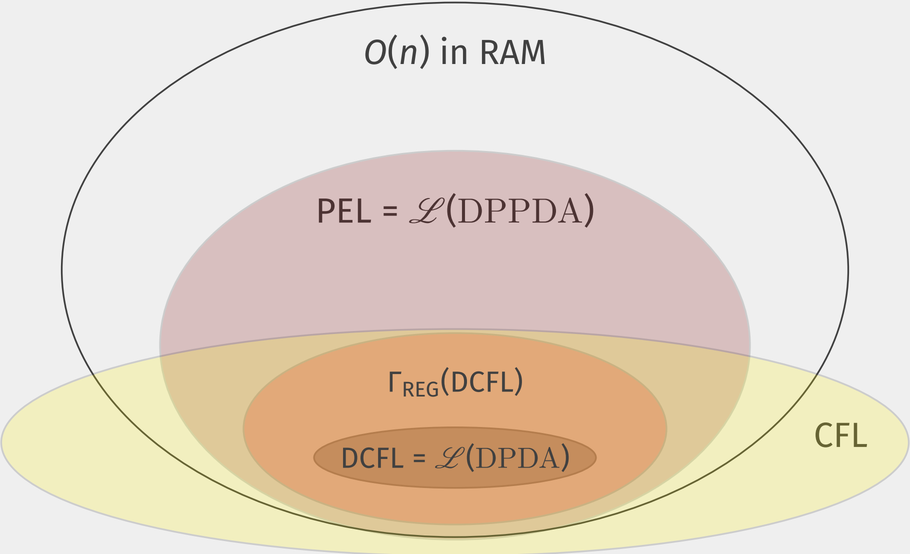
Results on Complexity [1/2]



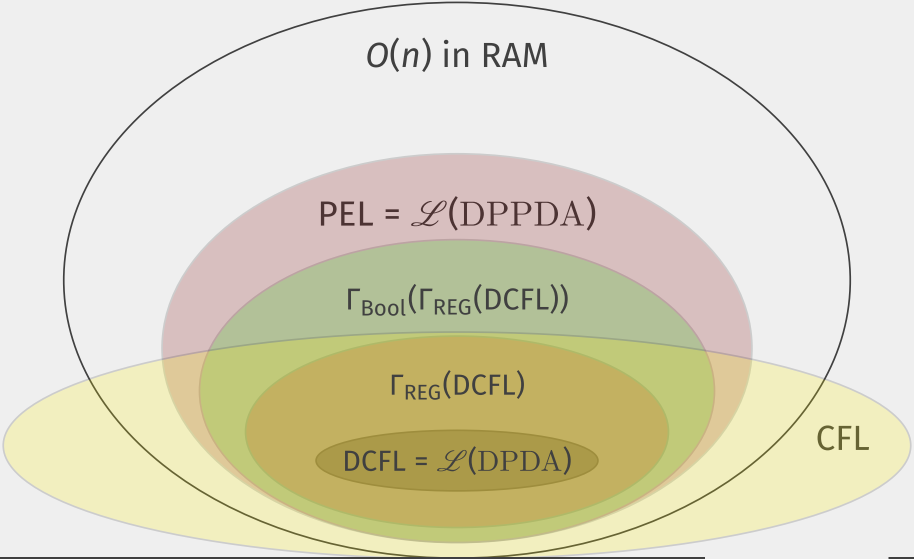
Results on Complexity [1/2]



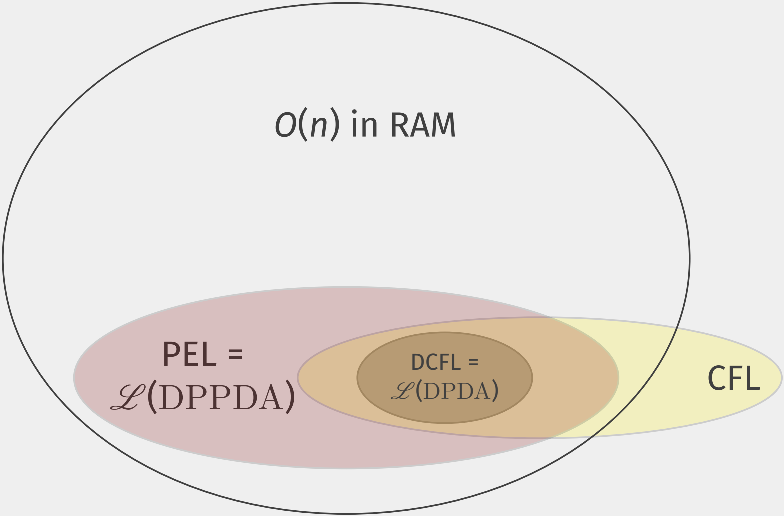
Results on Complexity [1/2]



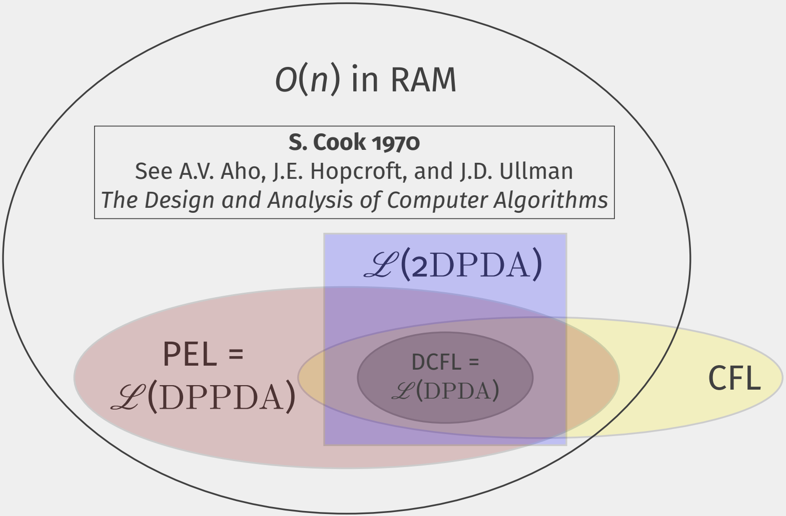
Results on Complexity [1/2]



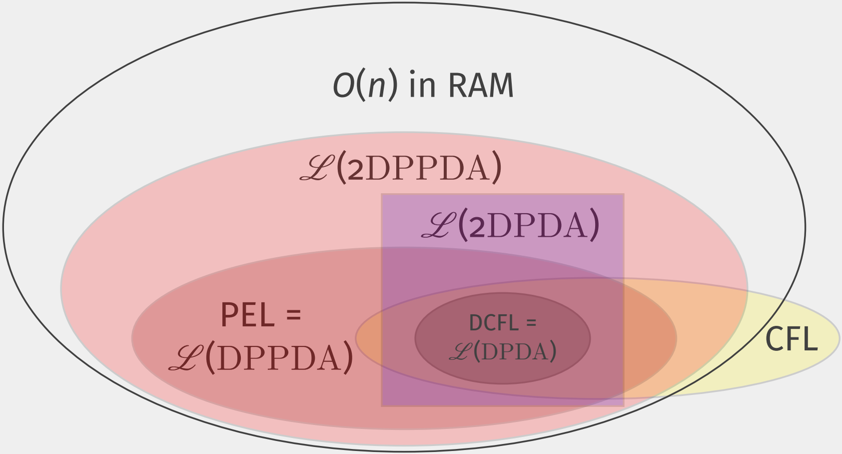
Results on Complexity [2/2]



RESULTS ON COMPLEXITY [2/2]



Results on Complexity [2/2]



Results and Conclusion

Overview of Previous Results

- PEGs is an upgrade of Top-Down Parsing Languages (TDPLs) introduced by A. Birman and J. Ullman in the 1960-s
  - DCFL $\subseteq$ TDPL, $a^n b^n c^n \in$ TDPL
  - Linear-time parsing algorithm, which was impractical in 60-s, because of memory overhead

Overview of Previous Results

- PEGs is an upgrade of Top-Down Parsing Languages (TDPLs) introduced by A. Birman and J. Ullman in the 1960-s
  - DCFL $\subseteq$ TDPL, $a^n b^n c^n \in$ TDPL
  - Linear-time parsing algorithm, which was impractical in 60-s, because of memory overhead
- B. Ford invented PEGs in 2004
  - PEGs are equivalent to TDPLs and generalized TDPLs
  - Moreover PEGs without **!** operation generate PEL
  - Practical linear-time parsing algorithm for PEGs (Packrat Parsing)

Overview of Previous Results

- PEGs is an upgrade of Top-Down Parsing Languages (TDPLs) introduced by A. Birman and J. Ullman in the 1960-s
  - DCFL $\subseteq$ TDPL, $a^n b^n c^n \in$ TDPL
  - Linear-time parsing algorithm, which was impractical in 60-s, because of memory overhead
- B. Ford invented PEGs in 2004
  - PEGs are equivalent to TDPLs and generalized TDPLs
  - Moreover PEGs without **!** operation generate PEL
  - Practical linear-time parsing algorithm for PEGs (Packrat Parsing)
- Python replaced LL-parser by PEG (PEP 617, 2020)

Overview of Previous Results

- PEGs is an upgrade of Top-Down Parsing Languages (TDPLs) introduced by A. Birman and J. Ullman in the 1960-s
  - DCFL $\subseteq$ TDPL, $a^n b^n c^n \in$ TDPL
  - Linear-time parsing algorithm, which was impractical in 60-s, because of memory overhead
- B. Ford invented PEGs in 2004
  - PEGs are equivalent to TDPLs and generalized TDPLs
  - Moreover PEGs without **!** operation generate PEL
  - Practical linear-time parsing algorithm for PEGs (Packrat Parsing)
- Python replaced LL-parser by PEG (PEP 617, 2020)
- B. Loff, N. Moreira, and R. Reis presented the first computational model for PEGs (DLT, 2018)
  - $a^{2^n} \in$ PEL and palindromes of length 2^n in PEL
  - Structural Results: there is no pumping lemma for PEL

Our Results

- New computational model: Pushdown Pointer Automata
  - $\mathscr{L}(\mathrm{DPPDA})$ = PEL
  - Linear-time simulation algorithm for 2-DPPDA (in RAM), modification of S. Cook algorithm for 2-DPDA
  - Clarification of PEL place among the formal languages
  - Simplicity: now the inclusion DCFL $\subseteq$ PEL is trivial
- **Thm.** $\Gamma_{\mathrm{REG}}(\mathrm{DCFL}) \cdot \mathrm{PEL} = \mathrm{PEL}$
  - **Corollary:** $\Gamma_{\mathrm{REG}}(\mathrm{DCFL}) \in \mathrm{PEL} \Rightarrow \Gamma_{\mathrm{Bool}}(\Gamma_{\mathrm{REG}}(\mathrm{DCFL})) \in \mathrm{PEL} \Rightarrow$ $\Rightarrow \Gamma_{\mathrm{Bool}}(\Gamma_{\mathrm{REG}}(\mathrm{DCFL}))$ is $O(n)$-recognizable in RAM.
    - It is a simplification and upgrade of the previously known result: $\Gamma_{\mathrm{REG}}(\mathrm{DCFL})$ is $O(n)$-recognizable [E. Bertsch and M.-J. Nederhof, SIAM J. on Comp, 1999]
- We hope that our model will rase interest to PELs in TCS community

Discussion

- The relations between following models are unknown:
  - DPPDA vs 2-DPDA
  - 2-DPDA vs 2-DPPDA
  - 2-DPDA vs 2-NPDA
  - 2-NPDA vs 2-NPPDA
- It is unknown whether CFL $\overset{?}{\subseteq} \mathscr{L}(2\text{-}\mathrm{DPPDA})$

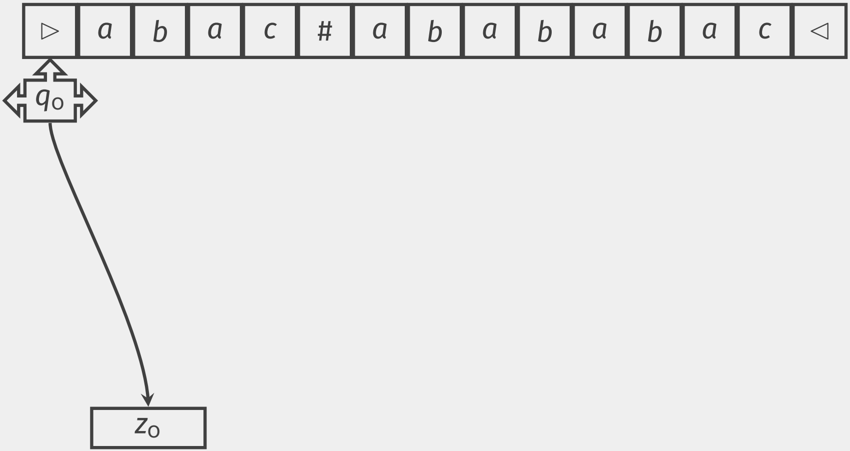
News

Now practically-motivated class PEL is among these classes

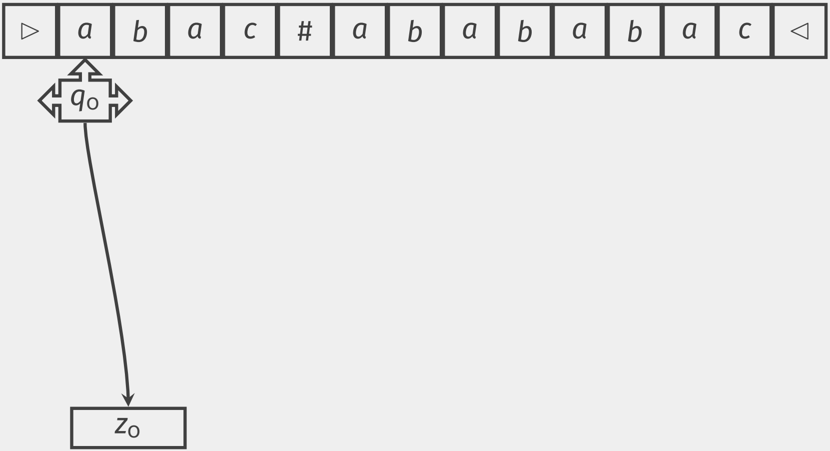
Remark

Many questions for 2-NDPA from Rupak Majumdar's talk are relevant for 2-NPPDA

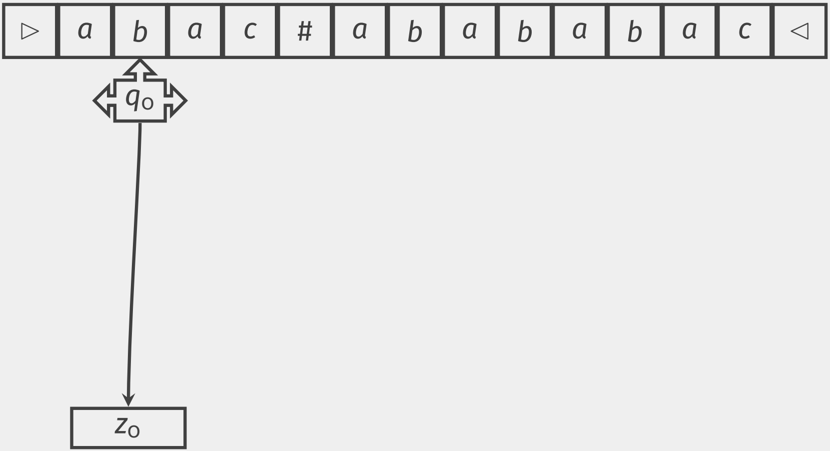
KMP-algorithm + 2DPDA



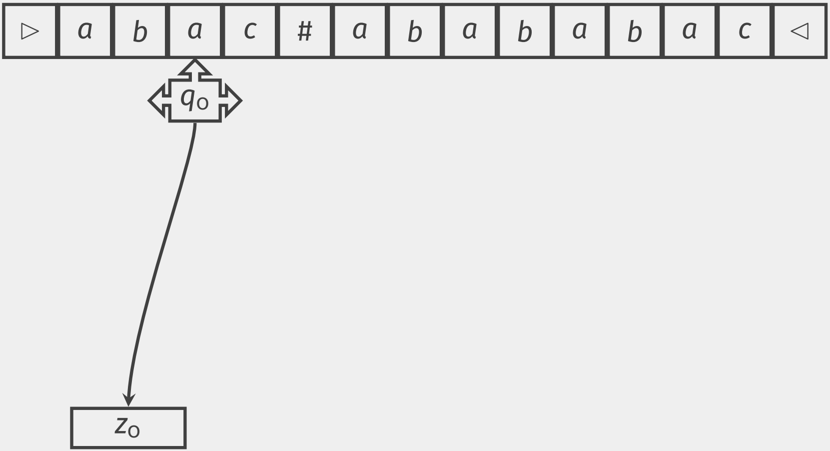
KMP-algorithm + 2DPDA



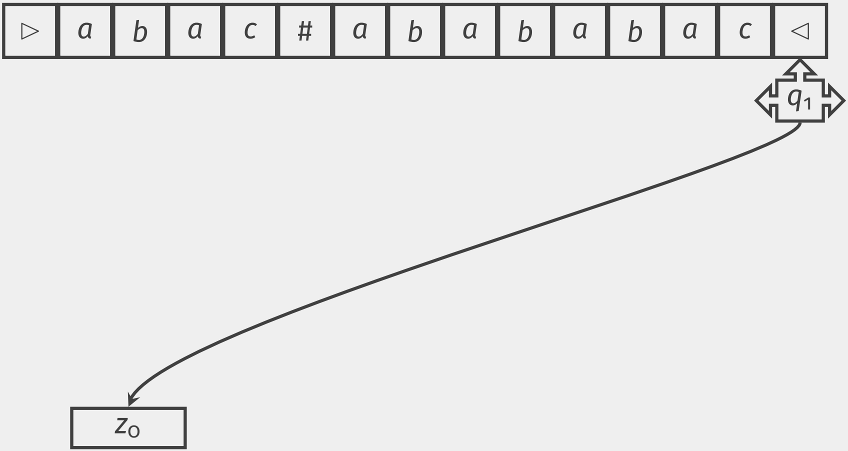
KMP-algorithm + 2DPDA



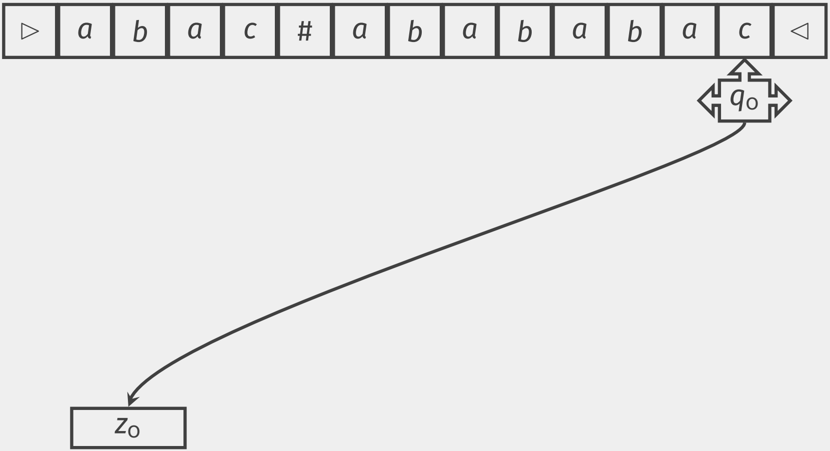
KMP-algorithm + 2DPDA



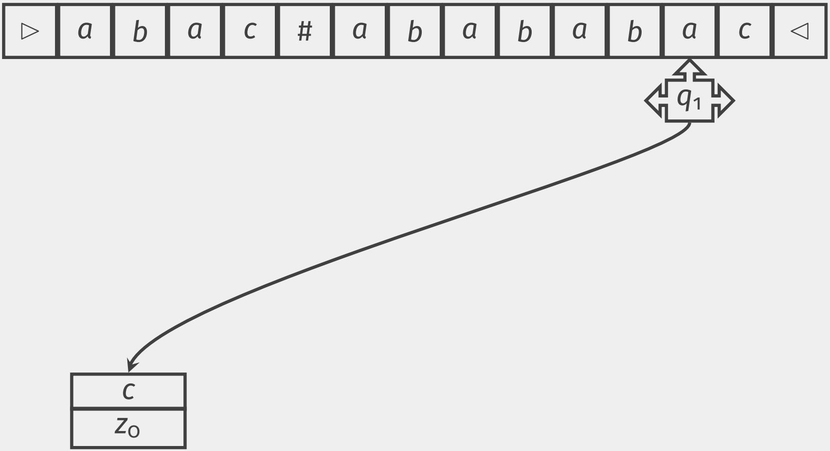
KMP-algorithm + 2DPDA



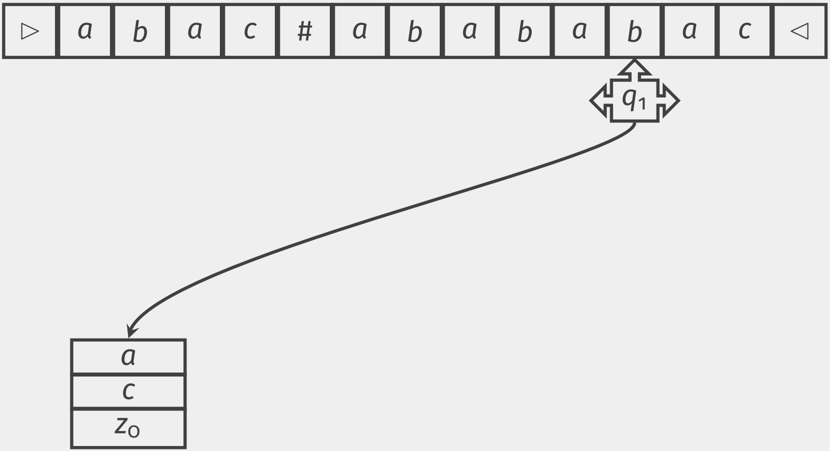
KMP-algorithm + 2DPDA



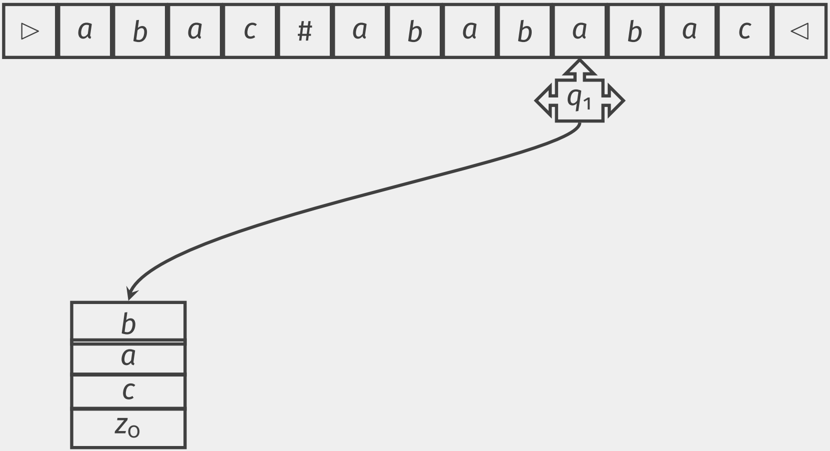
KMP-algorithm + 2DPDA



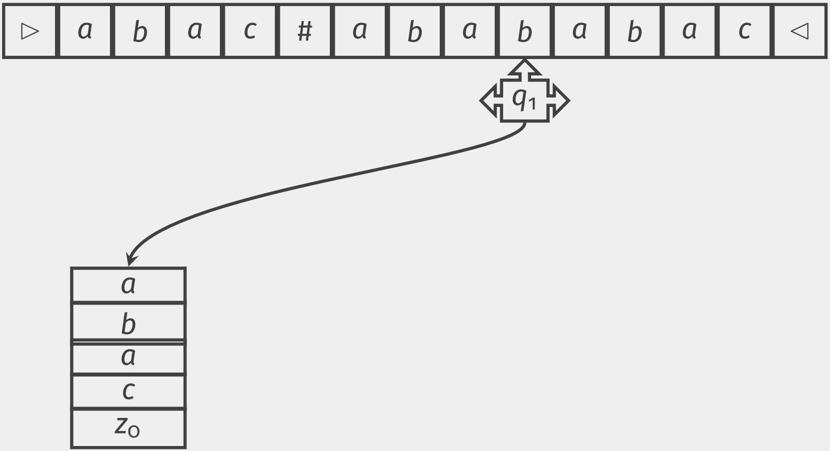
KMP-algorithm + 2DPDA



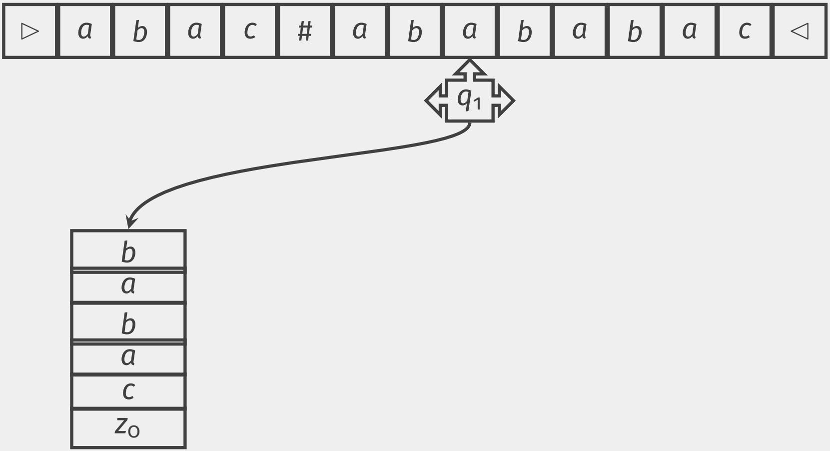
KMP-algorithm + 2DPDA



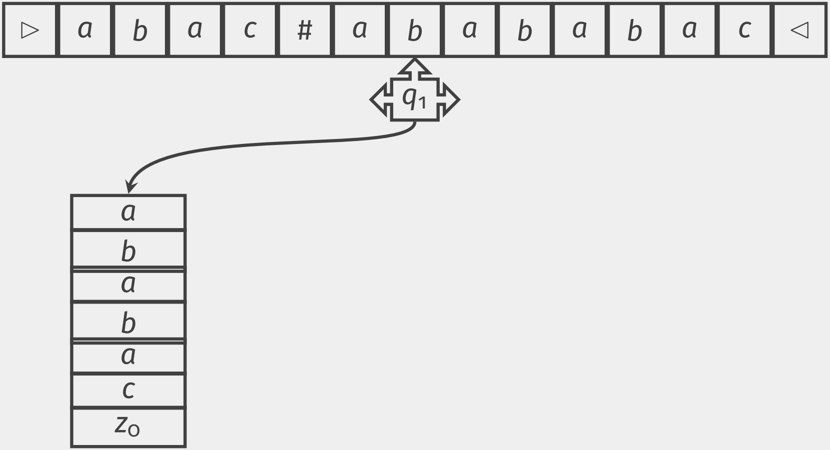
KMP-algorithm + 2DPDA



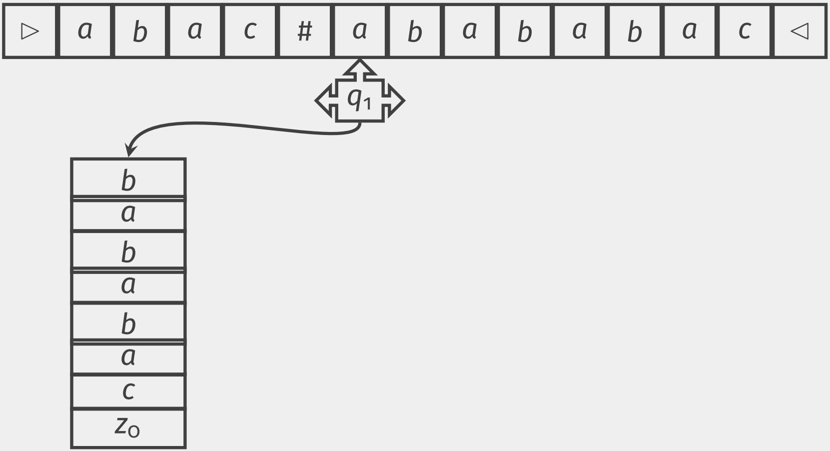
KMP-algorithm + 2DPDA



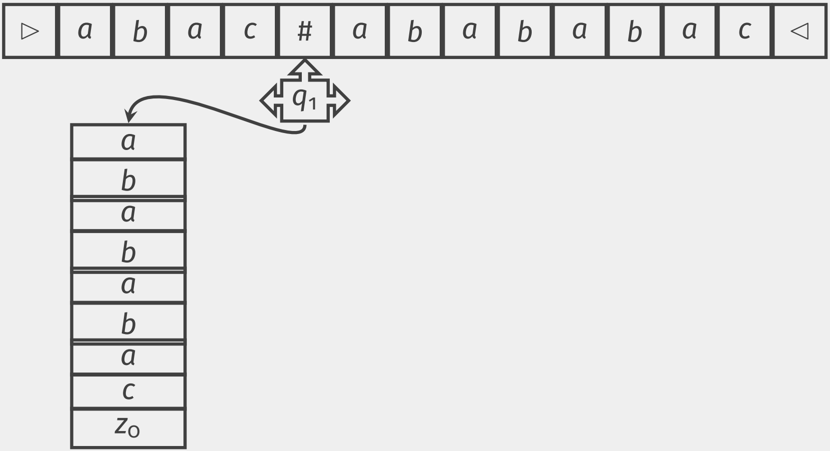
KMP-algorithm + 2DPDA



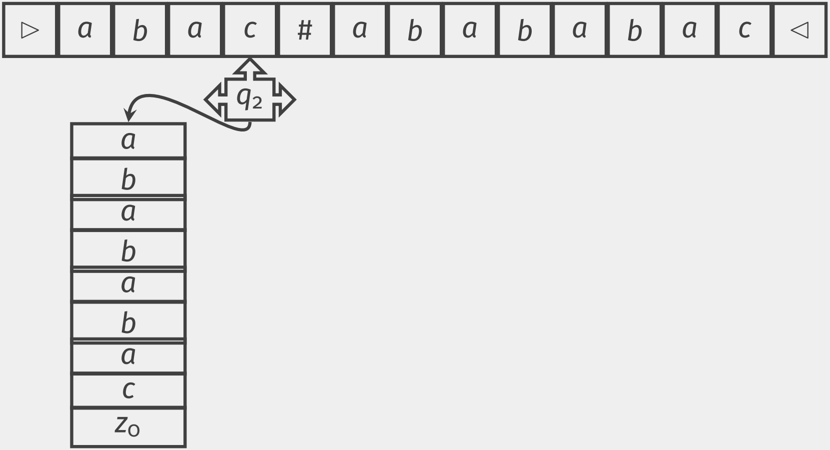
KMP-ALGORITHM + 2DPDA



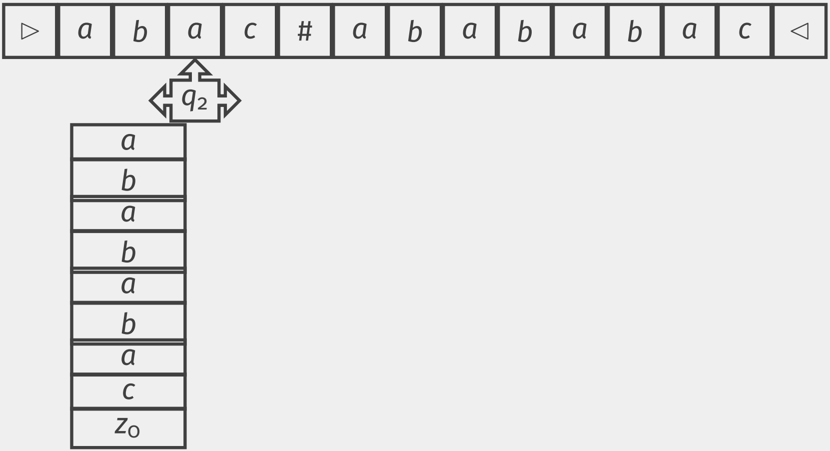
KMP-algorithm + 2DPDA



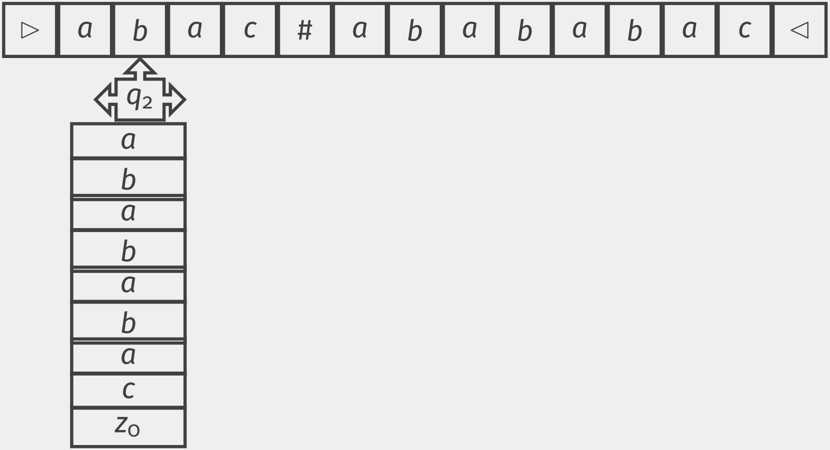
KMP-ALGORITHM + 2DPDA



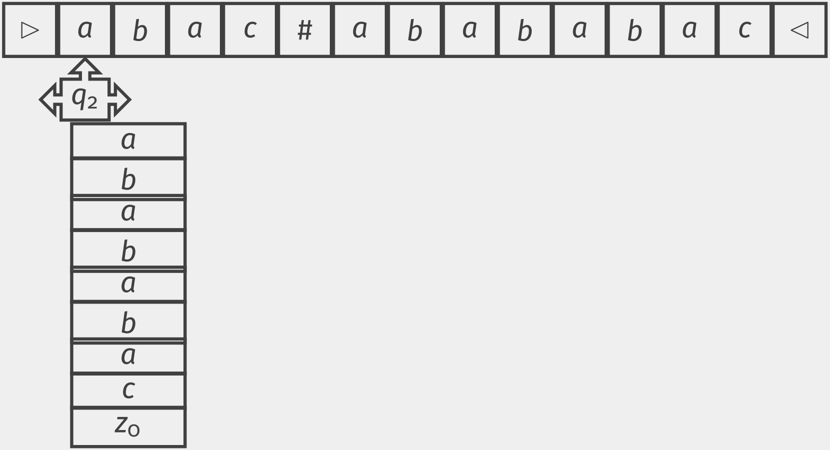
KMP-ALGORITHM + 2DPDA



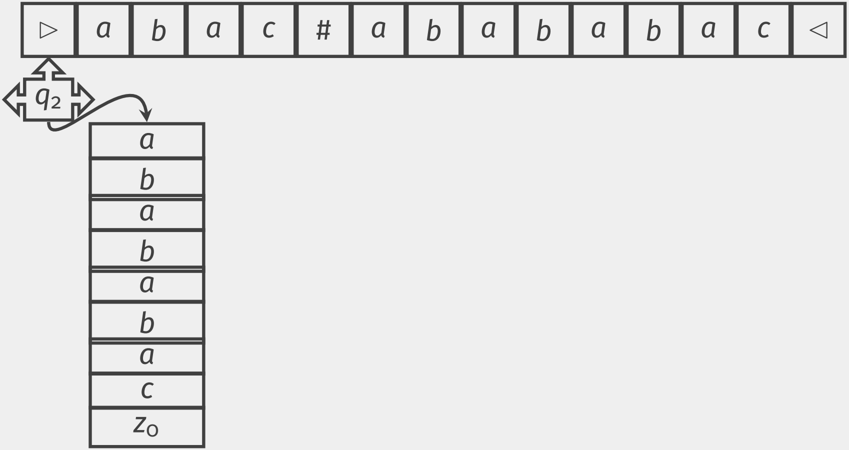
KMP-algorithm + 2DPDA



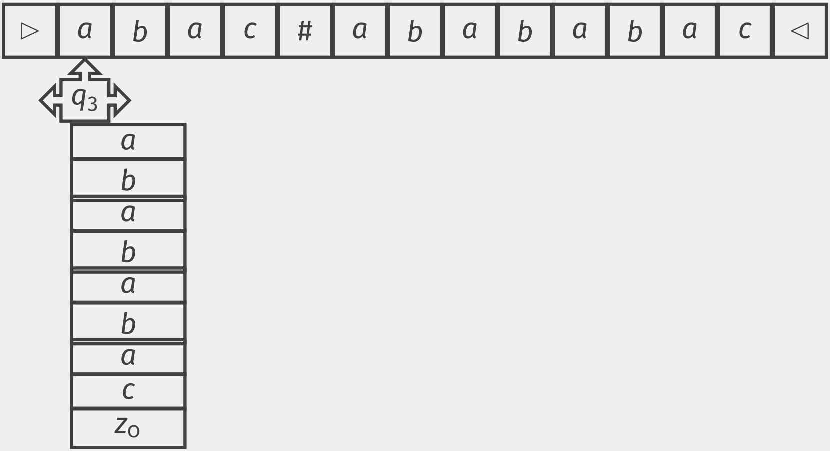
KMP-ALGORITHM + 2DPDA



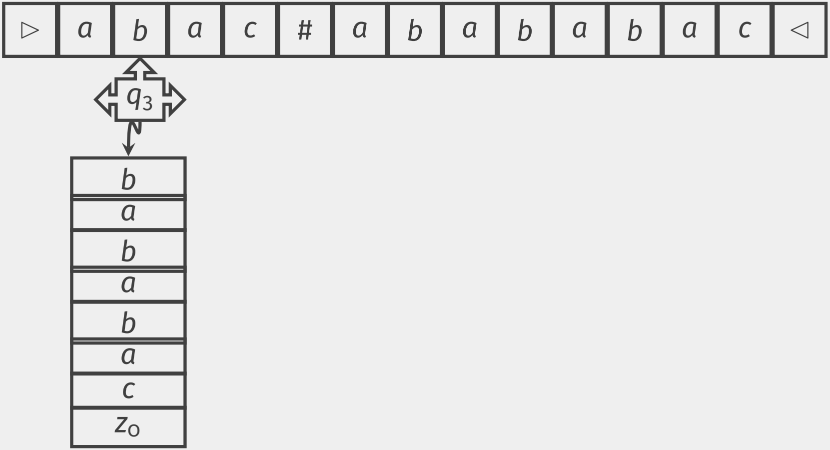
KMP-algorithm + 2DPDA



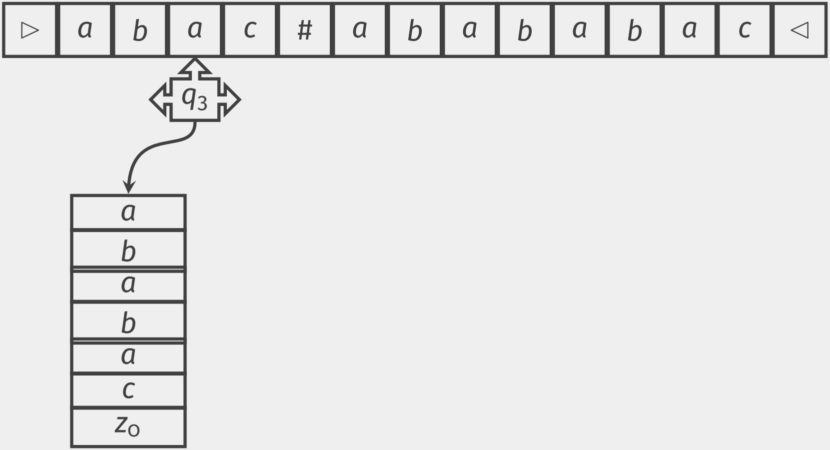
KMP-ALGORITHM + 2DPDA



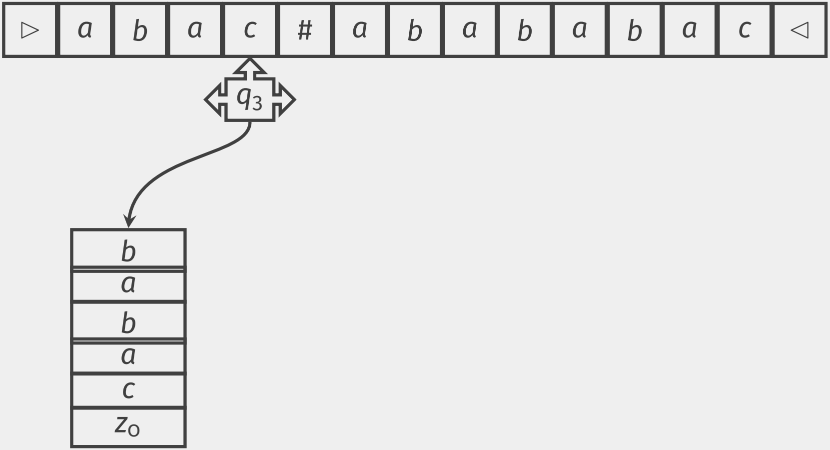
KMP-algorithm + 2DPDA



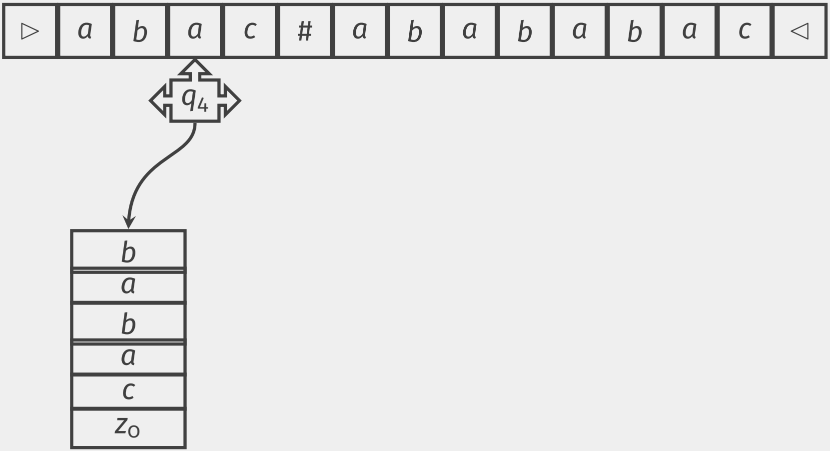
KMP-algorithm + 2DPDA



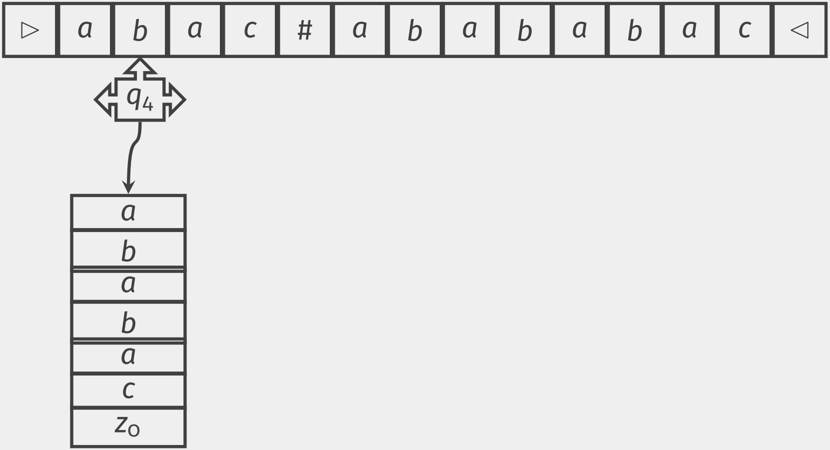
KMP-algorithm + 2DPDA



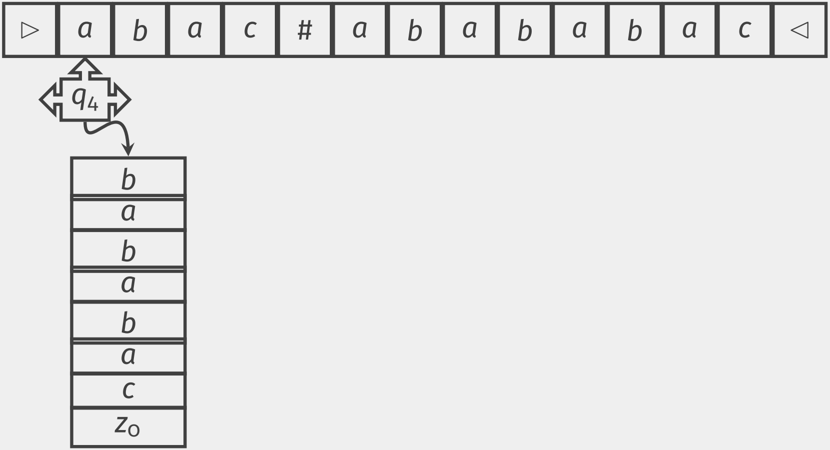
KMP-algorithm + 2DPDA



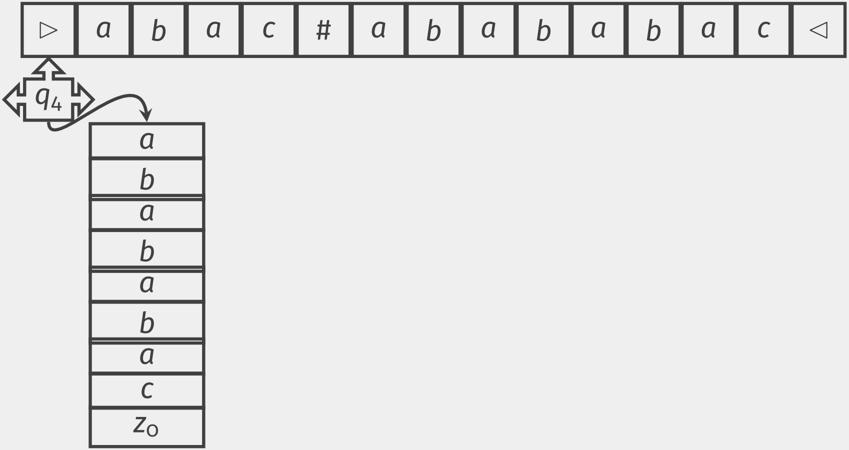
KMP-algorithm + 2DPDA



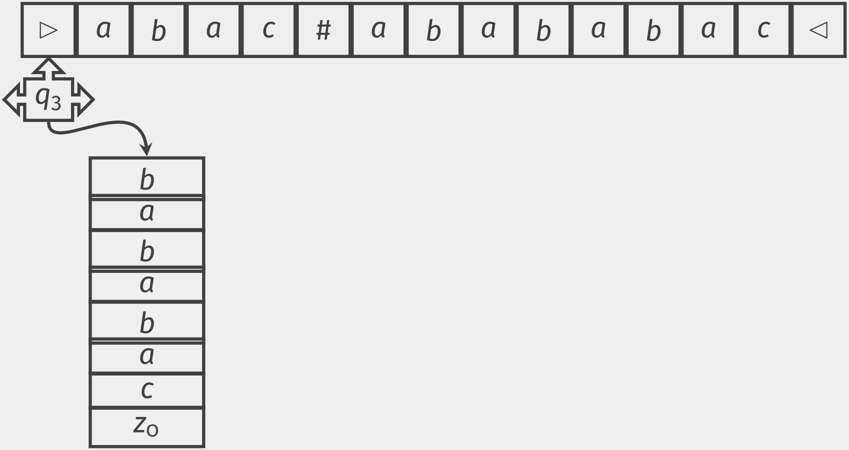
KMP-algorithm + 2DPDA



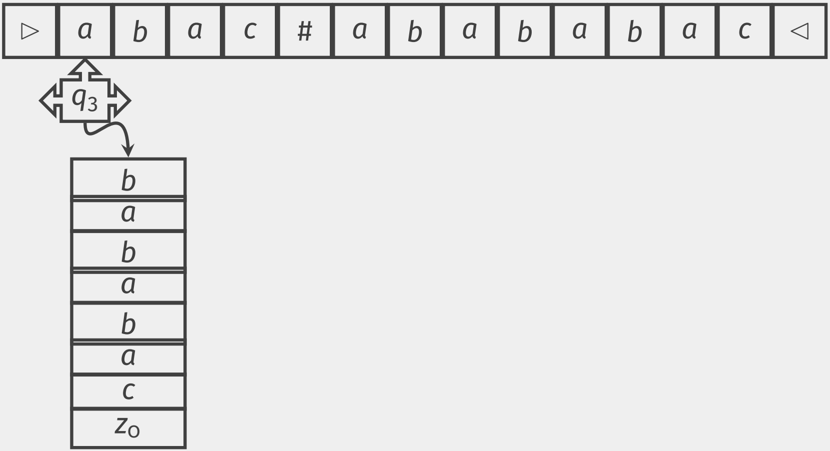
KMP-algorithm + 2DPDA



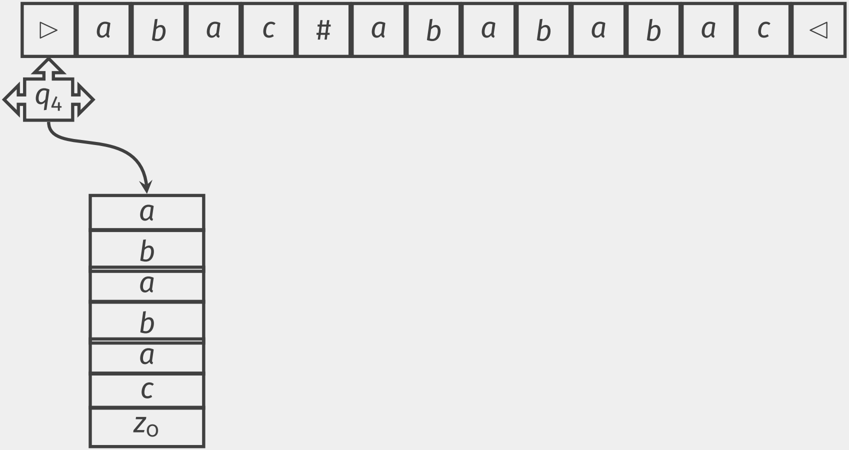
KMP-algorithm + 2DPDA



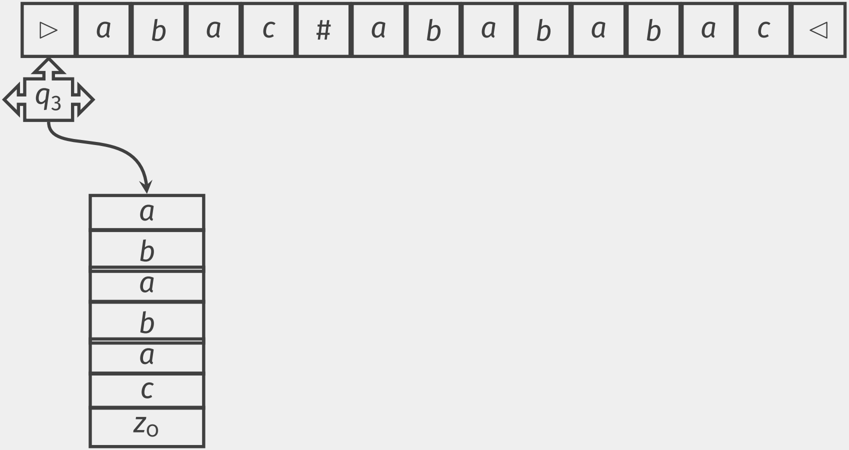
KMP-ALGORITHM + 2DPDA



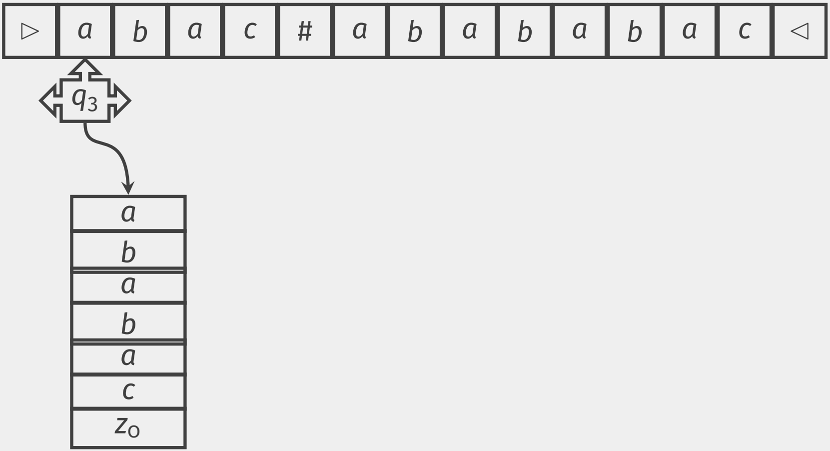
KMP-algorithm + 2DPDA



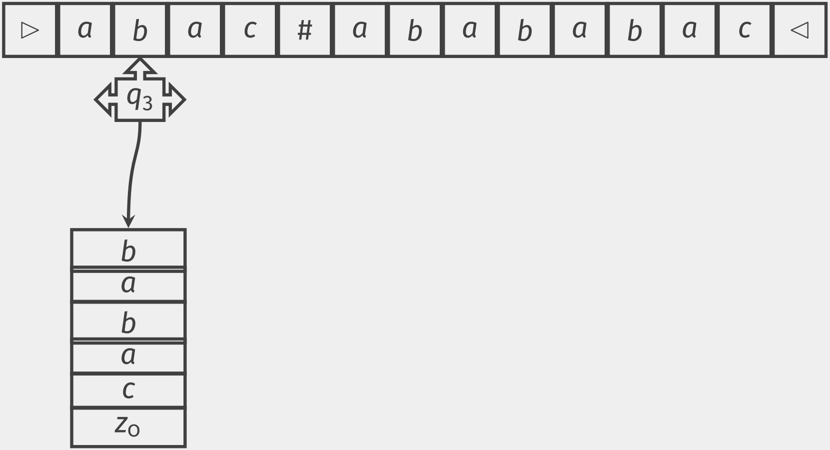
KMP-algorithm + 2DPDA



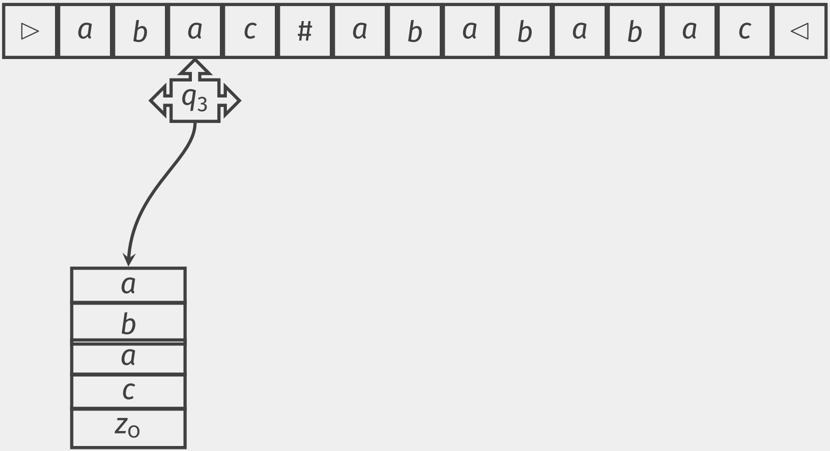
KMP-algorithm + 2DPDA



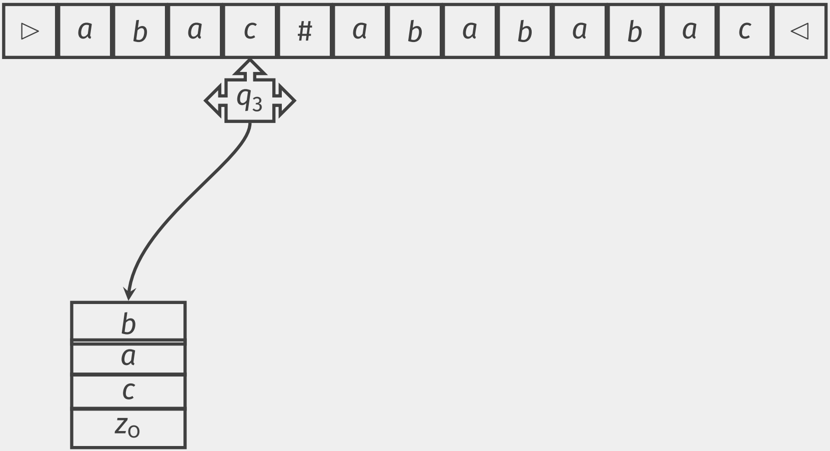
KMP-algorithm + 2DPDA



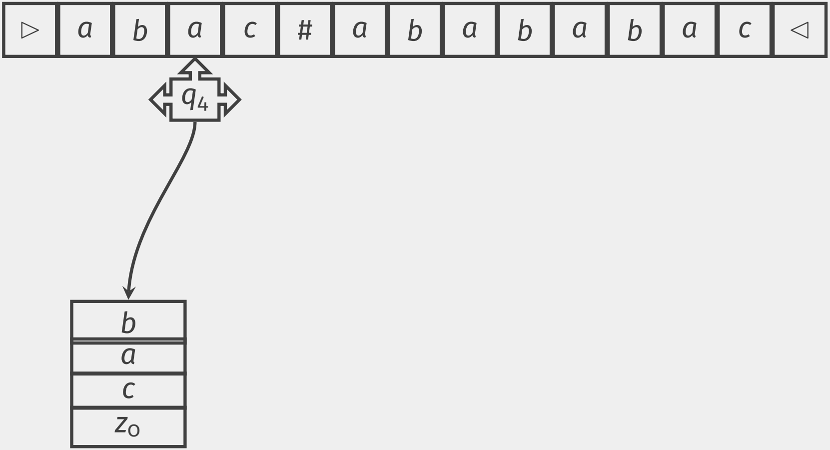
KMP-algorithm + 2DPDA



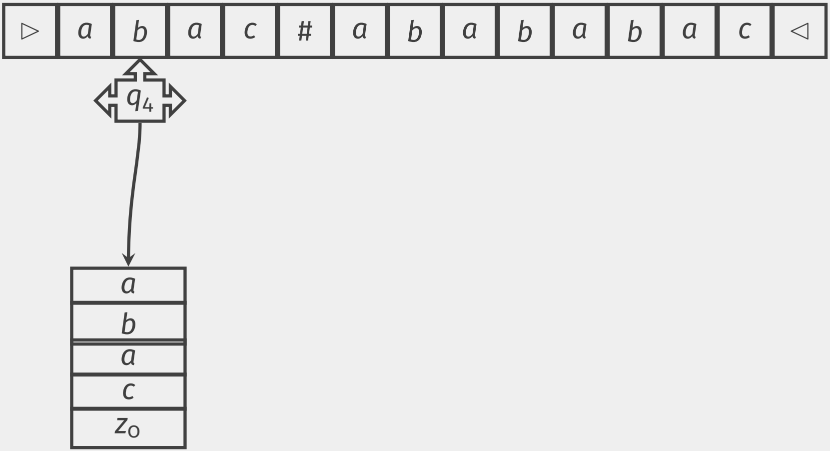
KMP-algorithm + 2DPDA



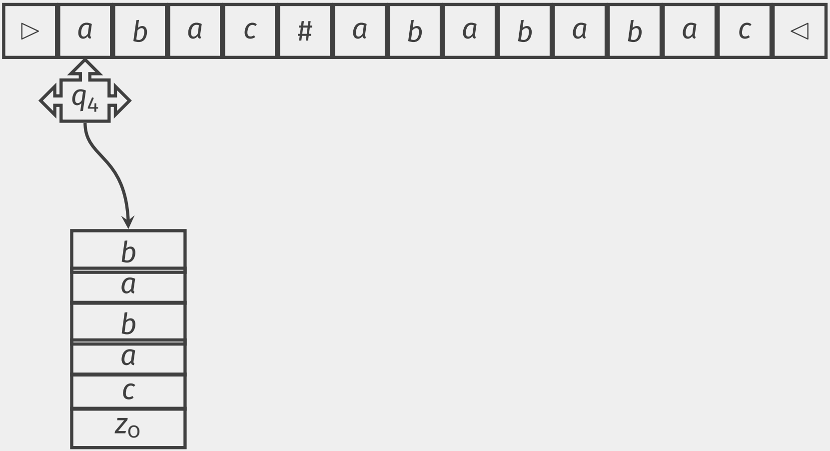
KMP-ALGORITHM + 2DPDA



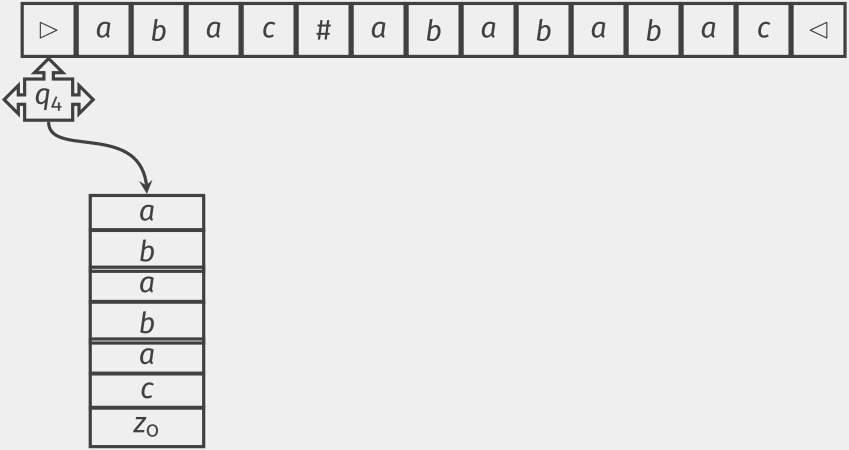
KMP-algorithm + 2DPDA



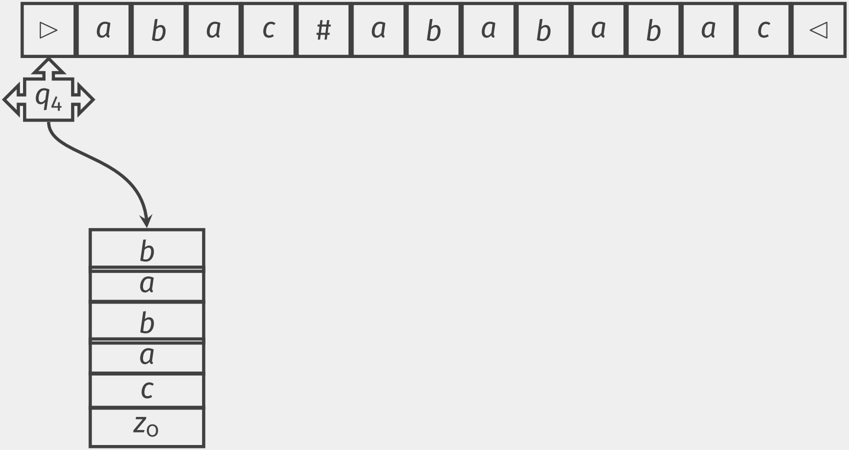
KMP-algorithm + 2DPDA



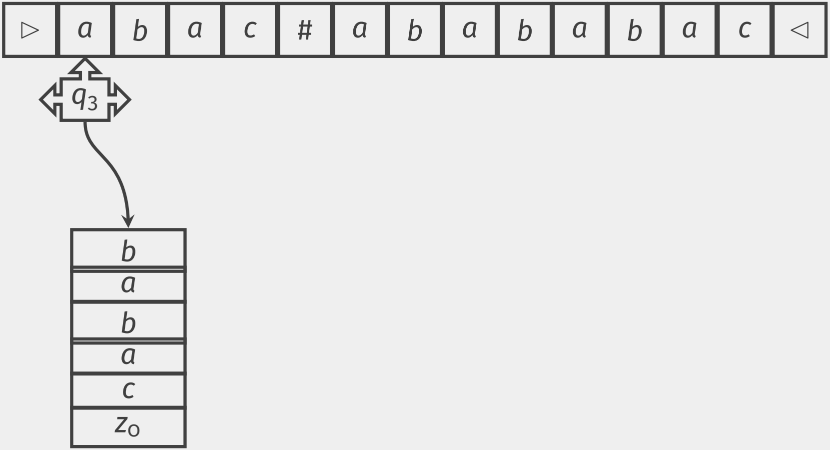
KMP-algorithm + 2DPDA



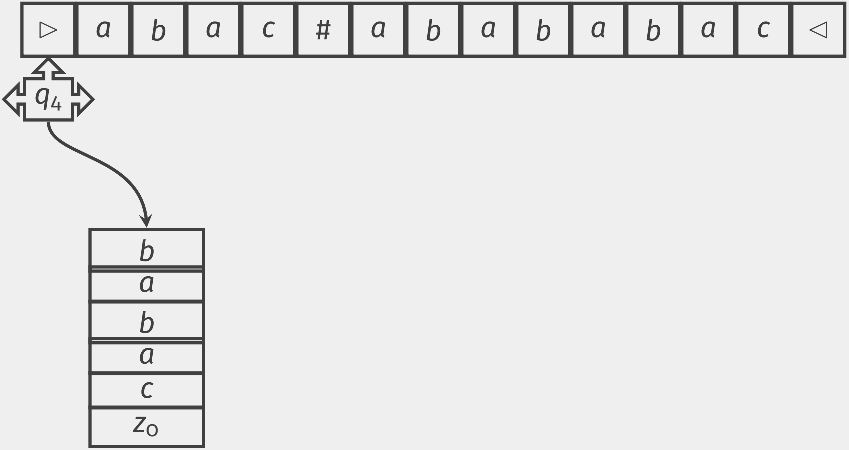
KMP-algorithm + 2DPDA



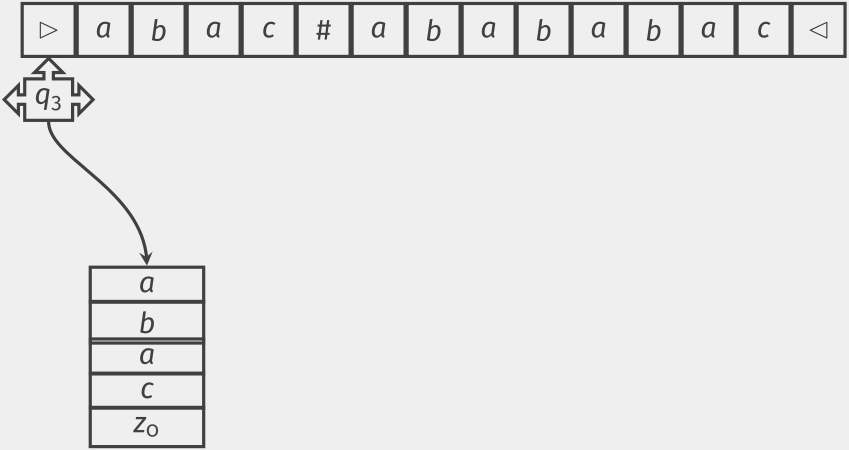
KMP-algorithm + 2DPDA



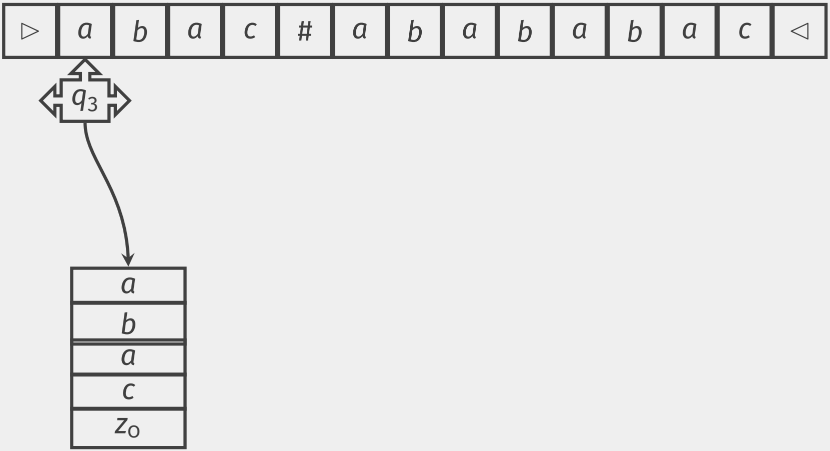
KMP-algorithm + 2DPDA



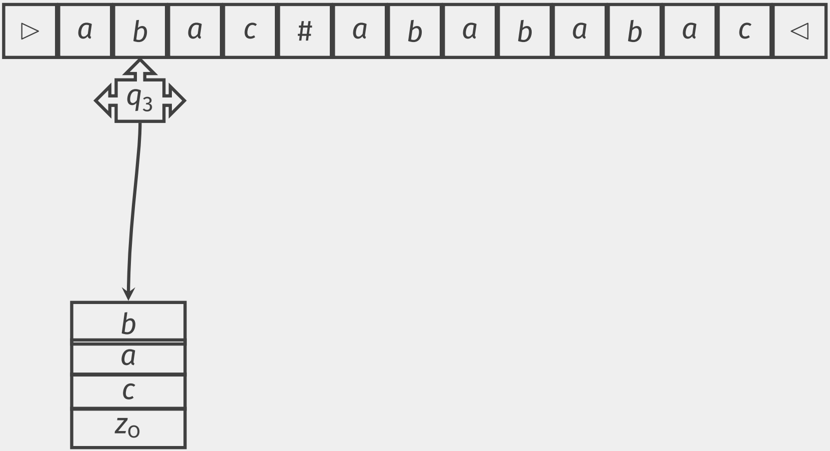
KMP-algorithm + 2DPDA



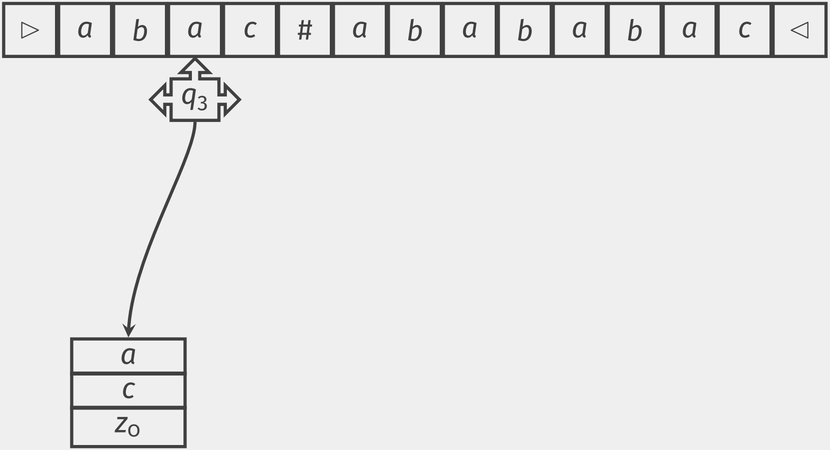
KMP-algorithm + 2DPDA



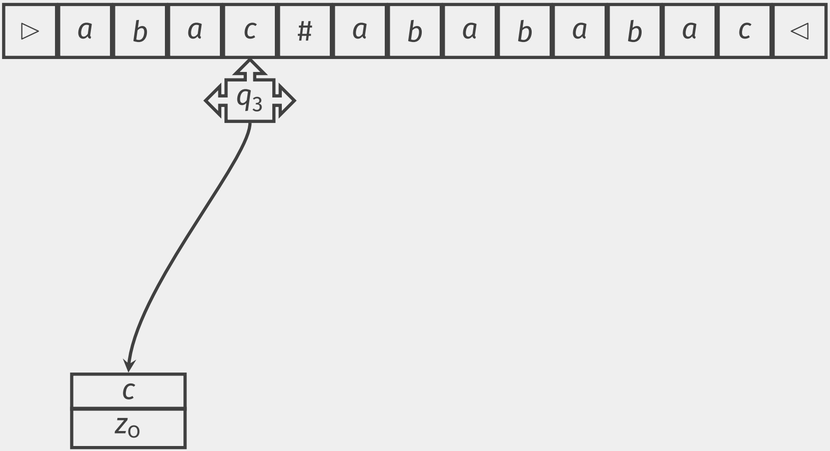
KMP-algorithm + 2DPDA



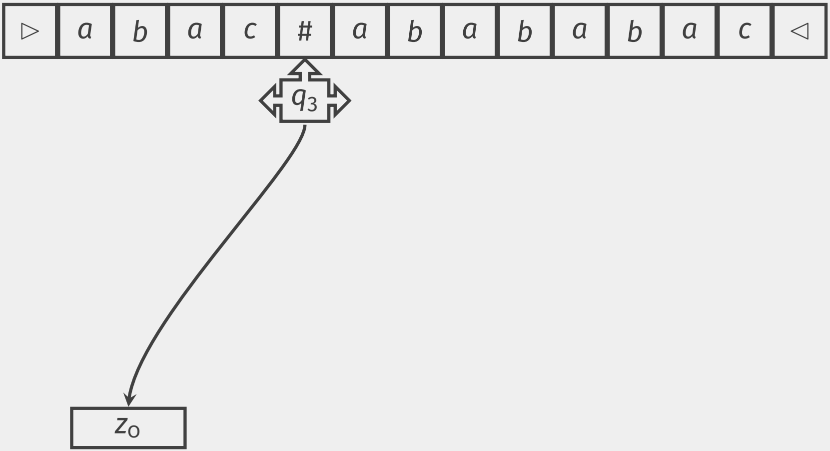
KMP-algorithm + 2DPDA



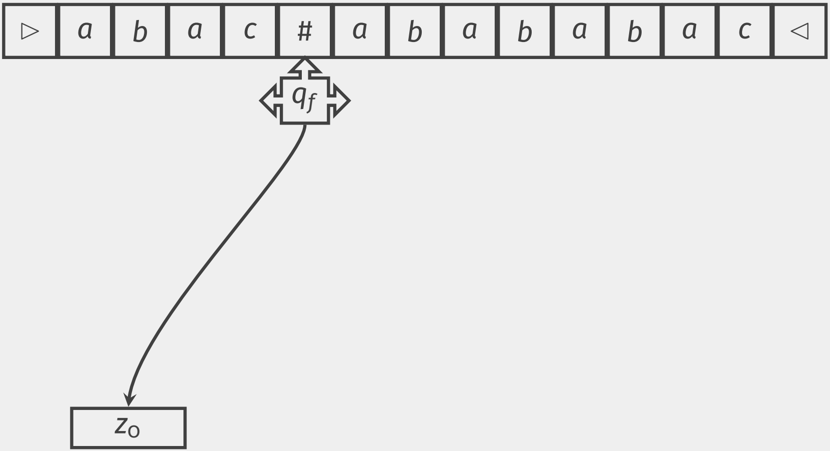
KMP-algorithm + 2DPDA



KMP-algorithm + 2DPDA



KMP-ALGORITHM + 2DPDA



Cook's Theorem

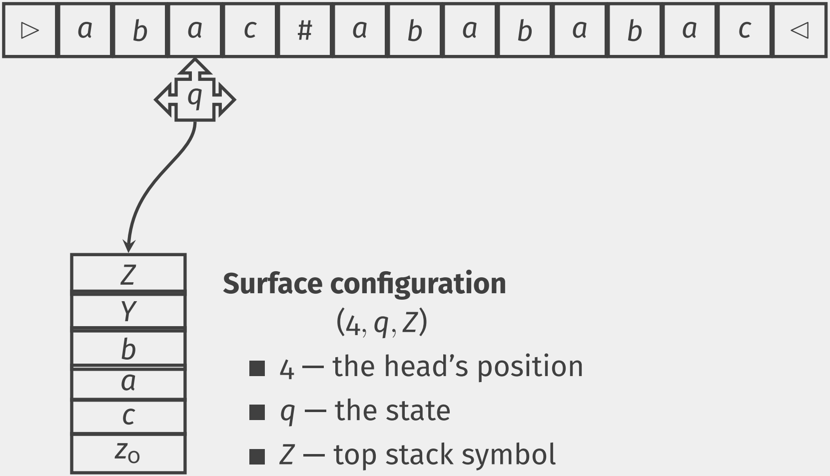
Theorem (Cook 1972)

A 2-DPDA-recognizable language is recognizable in linear time (in the RAM model).

Cook also provided a linear-time simulation algorithm.

- KMP algorithm has been investigated by Knuth by this simulation (and independently discovered by Morris without it).
- LR-parsers are 1-DPDA, so Cook's results show that there is an option of linear time parsing for wider class than DCFL.

Proof of Cook's Theorem



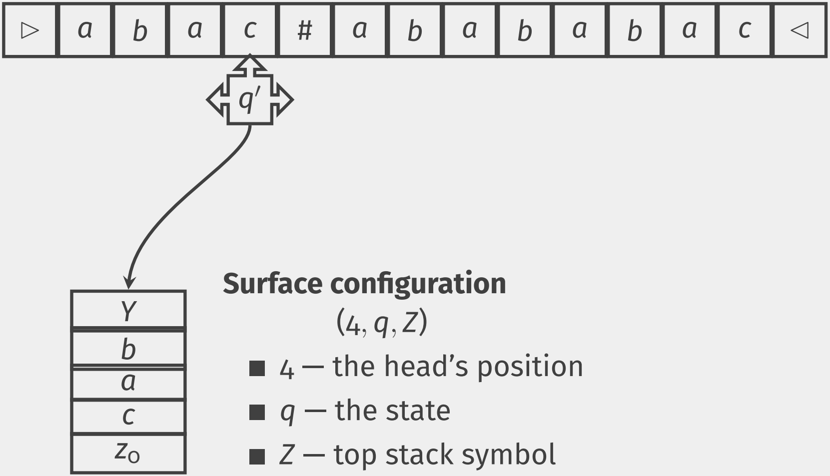
Surface configuration

$$(4, q, Z)$$

- 4 — the head's position
- q — the state
- Z — top stack symbol

Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

Proof of Cook's Theorem



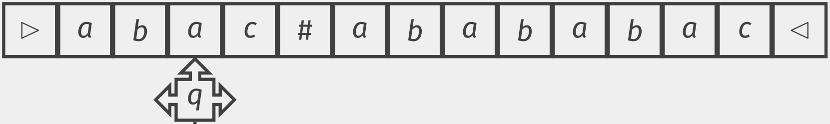
Surface configuration

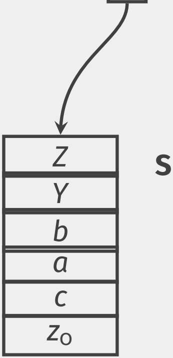
$$(4, q, Z)$$

- 4 — the head's position
- q — the state
- Z — top stack symbol

Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

Proof of Cook's Theorem





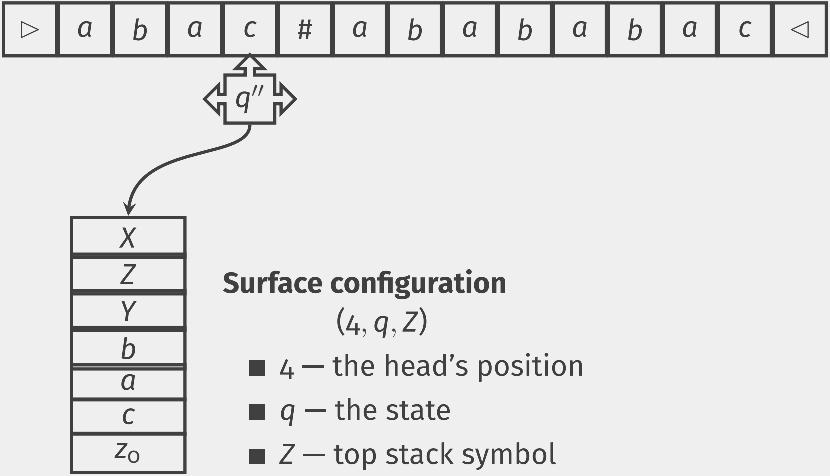
Surface configuration

$$(4, q, Z)$$

- 4 — the head's position
- q — the state
- Z — top stack symbol

Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

Proof of Cook's Theorem



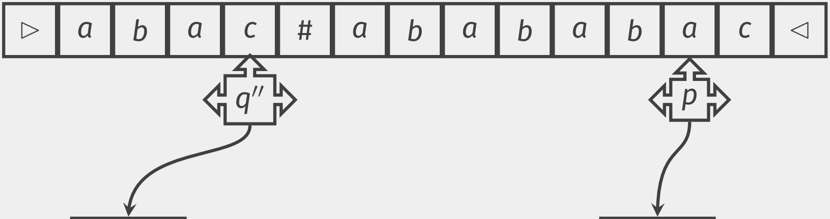
Surface configuration

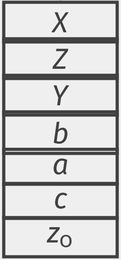
$$(4, q, Z)$$

- 4 — the head's position
- q — the state
- Z — top stack symbol

Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

Proof of Cook's Theorem

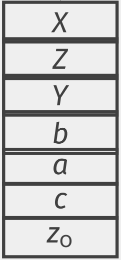




Surface configuration

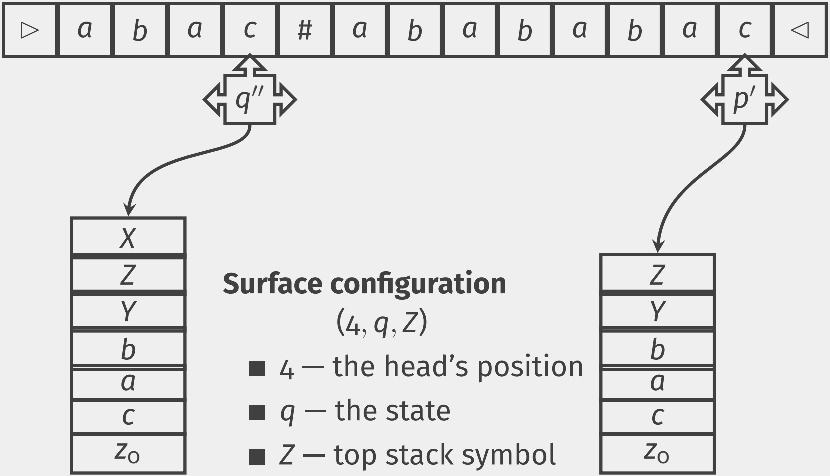
$(4, q, Z)$

- 4 — the head's position
- q — the state
- Z — top stack symbol



Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

Proof of Cook's Theorem



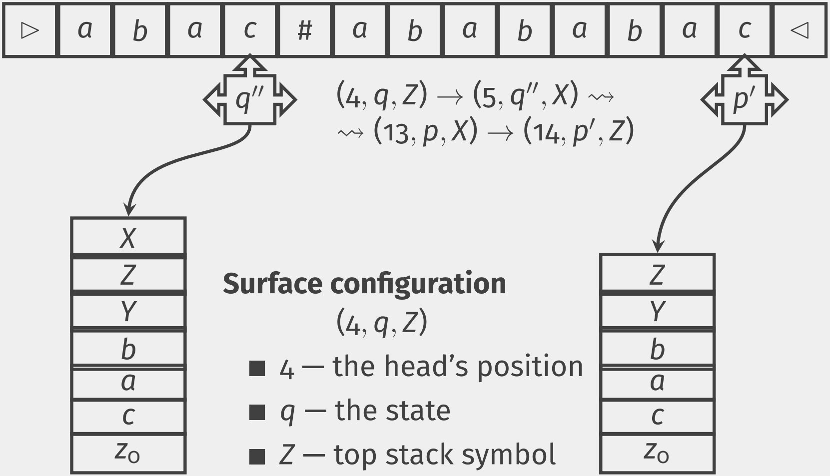
Surface configuration

$(4, q, Z)$

- 4 — the head's position
- q — the state
- Z — top stack symbol

Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

Proof of Cook's Theorem



$(4, q, Z) \to (5, q'', X) \leadsto$
$\leadsto (13, p, X) \to (14, p', Z)$

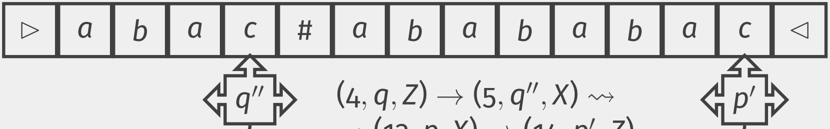
Surface configuration

$(4, q, Z)$

- 4 — the head's position
- q — the state
- Z — top stack symbol

Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.

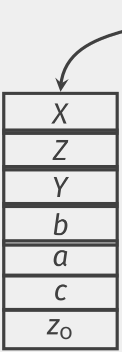
Proof of Cook's Theorem



$(4, q, Z) \to (5, q'', X) \rightsquigarrow$
$\rightsquigarrow (13, p, X) \to (14, p', Z)$

$T(5, q'', X) = (13, p, X)$

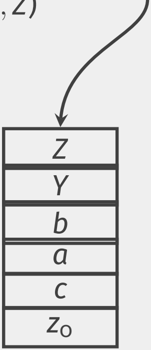
$T(4, q, Z) = T(14, p', Z)$



Surface configuration

$(4, q, Z)$

- 4 — the head's position
- q — the state
- Z — top stack symbol



Terminator for (i, q, Z) is (j, p, Z) s.t. Z is poped.