# Languages and Machines (Module 7 TCS+IAM)

## L&M 6: Pushdown Automata (PDA) and Context-Free Languages

Alexander Skopalik

Discrete Mathematics and Mathematical Programming, Applied Mathematics, University of Twente

Lecture 6

# Contents

# Contents

# Stack

Pushdown Automata extend Finite Automata with a Stack.

A Stack:

- holds an unbounded amount of elements
- can be accessed only in a restricted way
  - We can push an element on top of the stack
  - We can pop an element from the stack

LIFO = Last-in, First-out

# Pushdown automata (PDAs)

## Definition

A *pushdown automaton* is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

- $Q$ is a set of states
- $\Sigma$ is *input* alphabet
- $\Gamma$ is *stack* alphabet
- $\delta \subseteq Q \times \Sigma^\lambda \times \Gamma^\lambda \times \Gamma^\lambda \times Q$ is transition relation ($X^\lambda = X \cup \{\lambda\}$)
- $q_0 \in Q$ is start state
- $F \subseteq Q$ is set of final states

# Pushdown automata (PDAs)

## Definition

A *pushdown automaton* is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

- $Q$ is a set of states
- $\Sigma$ is *input* alphabet
- $\Gamma$ is *stack* alphabet
- $\delta \subseteq Q \times \Sigma^\lambda \times \Gamma^\lambda \times \Gamma^\lambda \times Q$ is transition relation ($X^\lambda = X \cup \{\lambda\}$)
- $q_0 \in Q$ is start state
- $F \subseteq Q$ is set of final states

Differences with finite automata:

- $\Gamma$ is the *stack alphabet* (cf. variables in CFGs)
- Transitions: 5-tuples $(q, a, A, A', q')$ instead of 3-tuples $(q, a, q')$
- We write $q \xrightarrow{a \;\; A/A'} q'$ instead of $(q, a, A, A', q') \in \delta$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \quad A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)

# Computations of a pushdown automaton

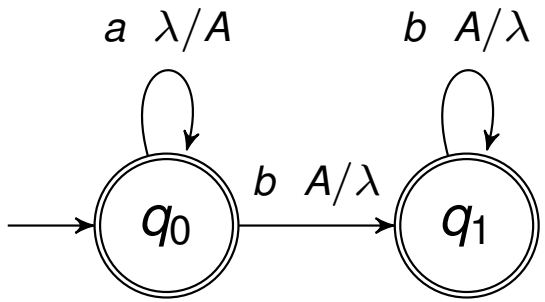
- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \quad A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\qquad (q_0, b, A, \lambda, q_1), \\
&\qquad (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

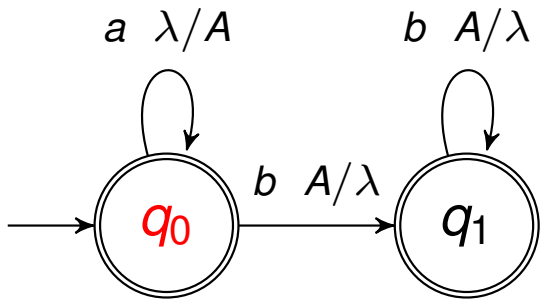
$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\qquad (q_0, b, A, \lambda, q_1), \\
&\qquad (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$



# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\quad\; (q_0, b, A, \lambda, q_1), \\
&\quad\; (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$



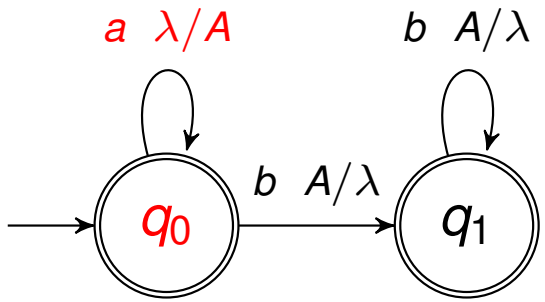

*Computation:*

$[q_0, aabb, \lambda]$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\qquad (q_0, b, A, \lambda, q_1), \\
&\qquad (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$



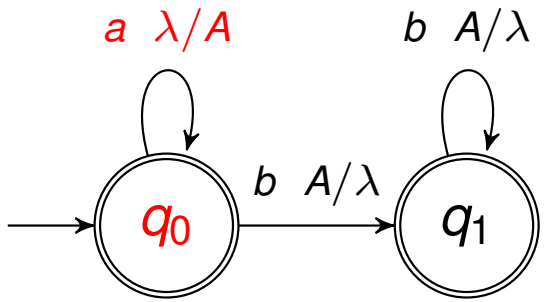

*Computation:*

$[q_0, aabb, \lambda]$
$\vdash [q_0, abb, A]$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\quad\; (q_0, b, A, \lambda, q_1), \\
&\quad\; (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$



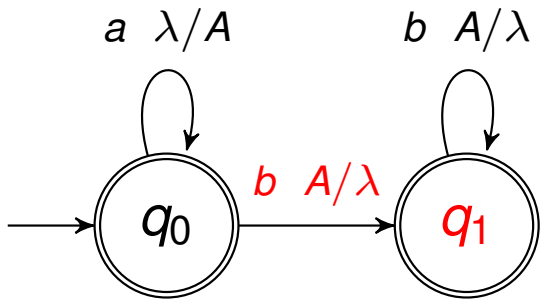

*Computation:*

$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
&\vdash [q_0, abb, A] \\
&\vdash [q_0, bb, AA]
\end{aligned}
$$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\quad\; (q_0, b, A, \lambda, q_1), \\
&\quad\; (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$



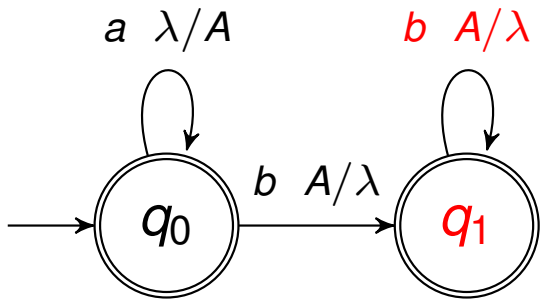

*Computation:*

$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
&\vdash [q_0, abb, A] \\
&\vdash [q_0, bb, AA] \\
&\vdash [q_1, b, A]
\end{aligned}
$$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\qquad (q_0, b, A, \lambda, q_1), \\
&\qquad (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$



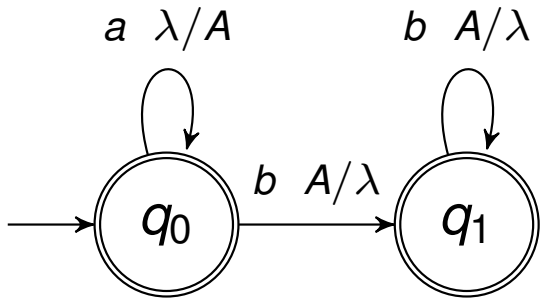

*Computation:*

$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
&\vdash [q_0, abb, A] \\
&\vdash [q_0, bb, AA] \\
&\vdash [q_1, b, A] \\
&\vdash [q_1, \lambda, \lambda]
\end{aligned}
$$

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\quad\; (q_0, b, A, \lambda, q_1), \\
&\quad\; (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$




*Computation:*

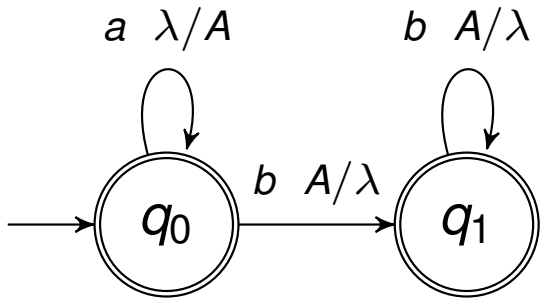
$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
&\vdash [q_0, abb, A] \\
&\vdash [q_0, bb, AA] \\
&\vdash [q_1, b, A] \\
&\vdash [q_1, \lambda, \lambda]
\end{aligned}
$$

Done: $q_1 \in F$, stack is empty

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\qquad (q_0, b, A, \lambda, q_1), \\
&\qquad (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$




*Computation:*

$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
&\vdash [q_0, abb, A] \\
&\vdash [q_0, bb, AA] \\
&\vdash [q_1, b, A] \\
&\vdash [q_1, \lambda, \lambda]
\end{aligned}
$$

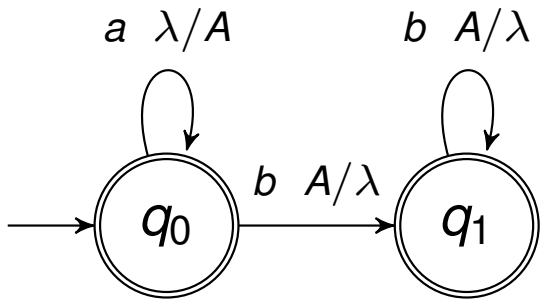
Done: $q_1 \in F$, stack is empty

- Language of this automaton?

# Computations of a pushdown automaton

- Configurations are combination $[q, w, \alpha]$, where $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$ ($\alpha$ is the *stack*).
- Effect of a transition $q \xrightarrow{a \;\; A/B} q'$:
  - Input symbol $a$ is *recognized* (can be $\lambda$)
  - Stack symbol $A$ is *popped* (can be $\lambda$)
  - Stack symbol $B$ is *pushed* (can be $\lambda$)
- Example:

$$
\begin{aligned}
Q &= \{q_0, q_1\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{A\} \\
F &= \{q_0, q_1\} \\
\delta &= \{(q_0, a, \lambda, A, q_0), \\
&\qquad (q_0, b, A, \lambda, q_1), \\
&\qquad (q_1, b, A, \lambda, q_1)\}
\end{aligned}
$$




*Computation:*

$$
\begin{aligned}
&[q_0, aabb, \lambda] \\
&\vdash [q_0, abb, A] \\
&\vdash [q_0, bb, AA] \\
&\vdash [q_1, b, A] \\
&\vdash [q_1, \lambda, \lambda]
\end{aligned}
$$

Done: $q_1 \in F$, stack is empty

- Language of this automaton? $\{a^i b^i \mid i \geq 0\}$

# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)

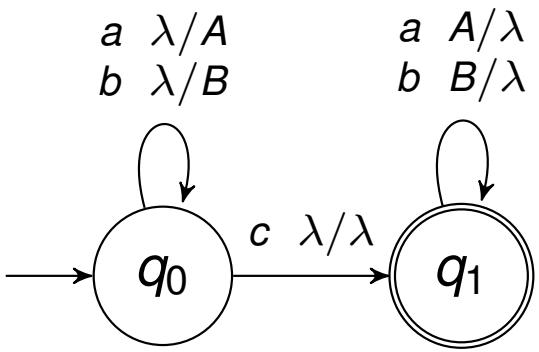
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)



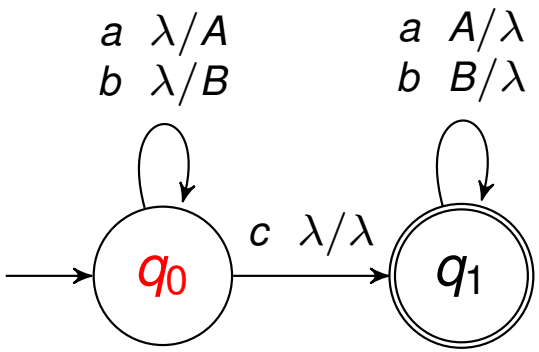
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)




$[q_0, abcba, \lambda]$
$\vdash$

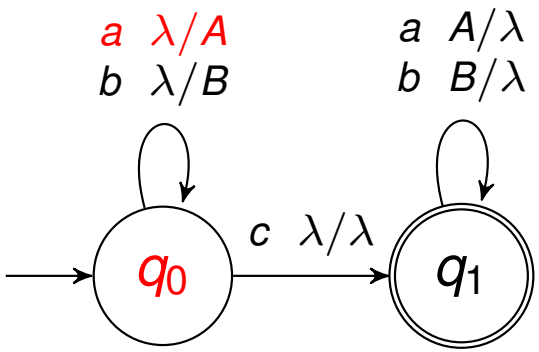
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)




$[q_0, abcba, \lambda]$
$\vdash [q_0, bcba, A]$
$\vdash$

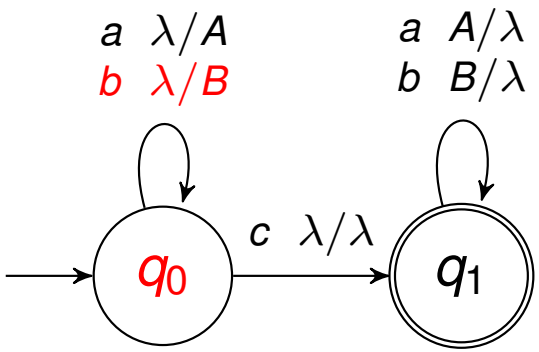
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)




$[q_0, abcba, \lambda]$
$\vdash [q_0, bcba, A]$
$\vdash [q_0, cba, BA]$
$\vdash$

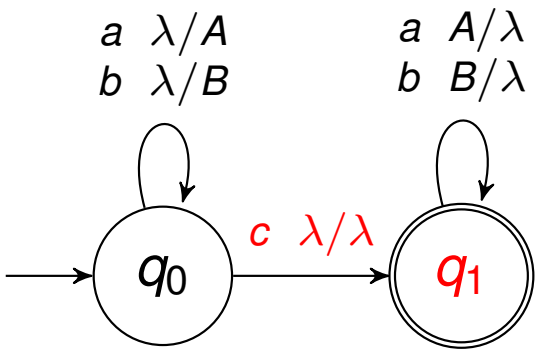
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)




$[q_0, abcba, \lambda]$
$\vdash [q_0, bcba, A]$
$\vdash [q_0, cba, BA]$
$\vdash [q_1, ba, BA]$
$\vdash$

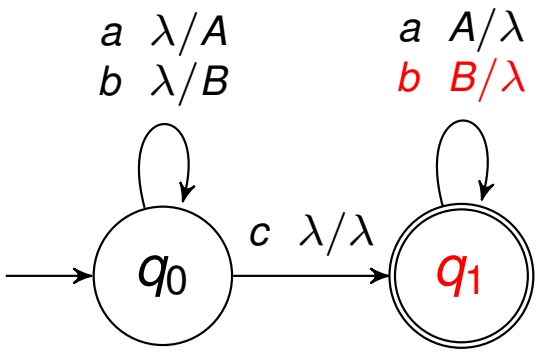
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)




$[q_0, abcba, \lambda]$
$\vdash [q_0, bcba, A]$
$\vdash [q_0, cba, BA]$
$\vdash [q_1, ba, BA]$
$\vdash [q_1, a, A]$
$\vdash$

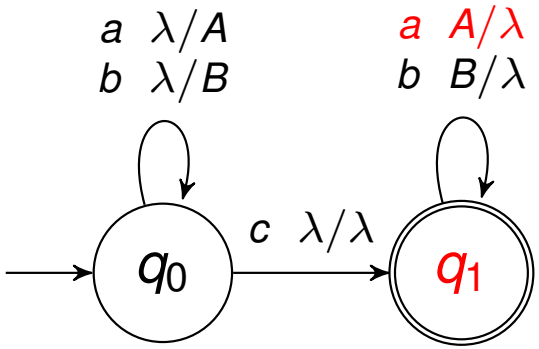
# Language of a pushdown automaton

## Definition

The language $L(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$, for which a computation exists such that:

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for certain $q \in F$.

Example: Language $\{wcw^R \mid w \in \{a, b\}^*\}$ ($R$ the reverse operator)




$[q_0, abcba, \lambda]$
$\vdash [q_0, bcba, A]$
$\vdash [q_0, cba, BA]$
$\vdash [q_1, ba, BA]$
$\vdash [q_1, a, A]$
$\vdash [q_1, \lambda, \lambda]$

# Connection with finite automata

- Both are *machine models*
- Pushdown automaton has unbounded (infinite) *memory*
- Memory is only partially accessible (only top of the stack)

# Connection with finite automata

- Both are *machine models*
- Pushdown automaton has unbounded (infinite) *memory*
- Memory is only partially accessible (only top of the stack)

**Every finite automaton can be converted into a pushdown automaton**

# Connection with finite automata

- Both are *machine models*
- Pushdown automaton has unbounded (infinite) *memory*
- Memory is only partially accessible (only top of the stack)

**Every finite automaton can be converted into a pushdown automaton**

- Set of stack variables empty
- Stack always remains empty
- Regular transition $q \xrightarrow{a} q'$ converted to "stack transition" $q \xrightarrow{a\ \ \lambda/\lambda} q'$

# Contents

# Deterministic pushdown automata

## Definition: Overlapping transitions

Transitions $q \xrightarrow{x_1 \quad A_1/B_1} q_1$ and $q \xrightarrow{x_2 \quad A_2/B_2} q_2$ *overlap* if

- $x_1 = x_2$ or $x_1 = \lambda$ or $x_2 = \lambda$, ánd
- $A_1 = A_2$ or $A_1 = \lambda$ or $A_2 = \lambda$

# Deterministic pushdown automata

## Definition: Overlapping transitions

Transitions $q \xrightarrow{x_1 \quad A_1/B_1} q_1$ and $q \xrightarrow{x_2 \quad A_2/B_2} q_2$ *overlap* if

- $x_1 = x_2$ or $x_1 = \lambda$ or $x_2 = \lambda$, ánd
- $A_1 = A_2$ or $A_1 = \lambda$ or $A_2 = \lambda$

Intuition: overlapping transitions leave options open

- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{a \quad A/B_2} q_2$
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{a \quad \lambda/B_2} q_2$
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{\lambda \quad A/B_2} q_2$
- $q \xrightarrow{a \quad \lambda/B_1} q_1$ versus $q \xrightarrow{\lambda \quad A/B_2} q_2$
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{\lambda \quad \lambda/B_2} q_2$

# Deterministic pushdown automata

## Definition: Overlapping transitions

Transitions $q \xrightarrow{x_1 \quad A_1/B_1} q_1$ and $q \xrightarrow{x_2 \quad A_2/B_2} q_2$ *overlap* if

- $x_1 = x_2$ or $x_1 = \lambda$ or $x_2 = \lambda$, ánd
- $A_1 = A_2$ or $A_1 = \lambda$ or $A_2 = \lambda$

Intuition: overlapping transitions leave options open

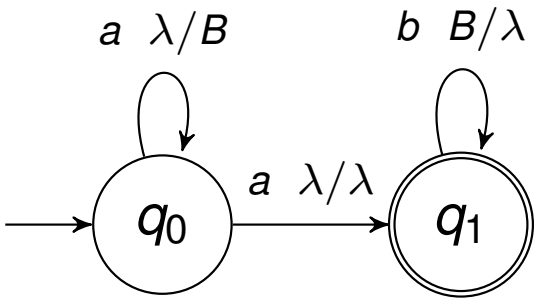
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{a \quad A/B_2} q_2$
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{a \quad \lambda/B_2} q_2$
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{\lambda \quad A/B_2} q_2$
- $q \xrightarrow{a \quad \lambda/B_1} q_1$ versus $q \xrightarrow{\lambda \quad A/B_2} q_2$
- $q \xrightarrow{a \quad A/B_1} q_1$ versus $q \xrightarrow{\lambda \quad \lambda/B_2} q_2$

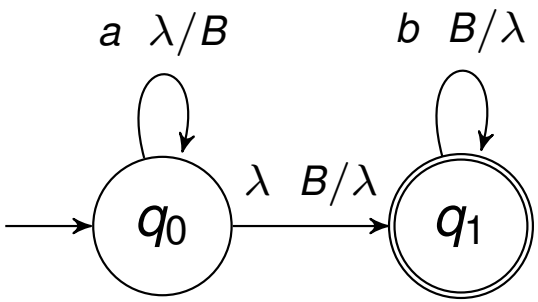
## Definition: Determinism

Pushdown automata without overlapping transitions are *deterministic*

# Examples (non-)deterministic pushdown automata

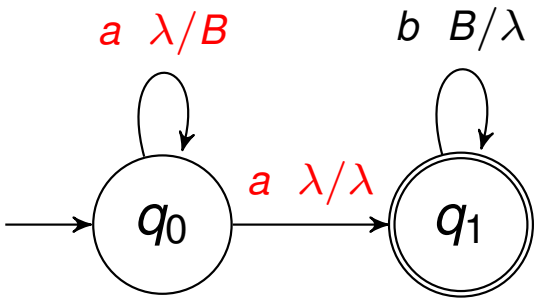
The following automata are equivalent (language: $\{a^m b^n \mid m = n + 1\}$)



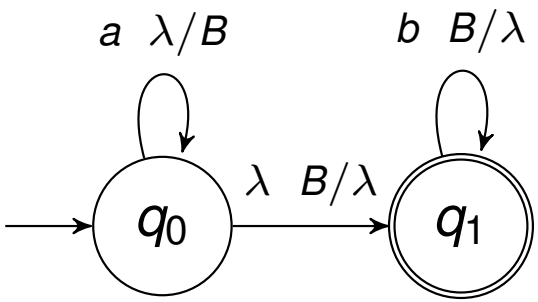



# Examples (non-)deterministic pushdown automata

The following automata are equivalent (language: $\{a^m b^n \mid m = n + 1\}$)

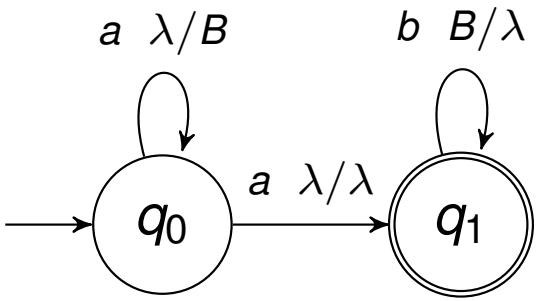



overlap

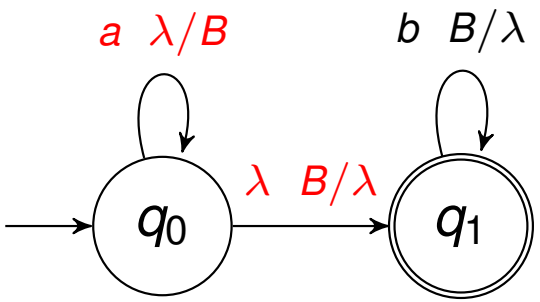


# Examples (non-)deterministic pushdown automata

The following automata are equivalent (language: $\{a^m b^n \mid m = n + 1\}$)



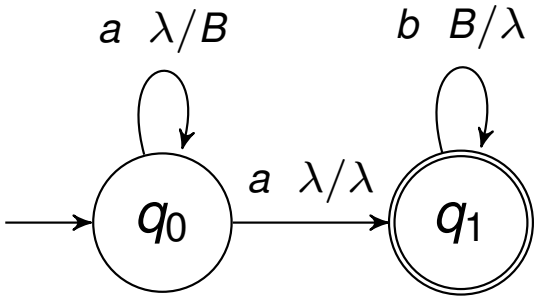

overlap



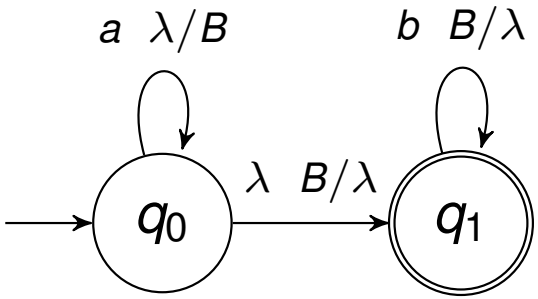

overlap

# Examples (non-)deterministic pushdown automata

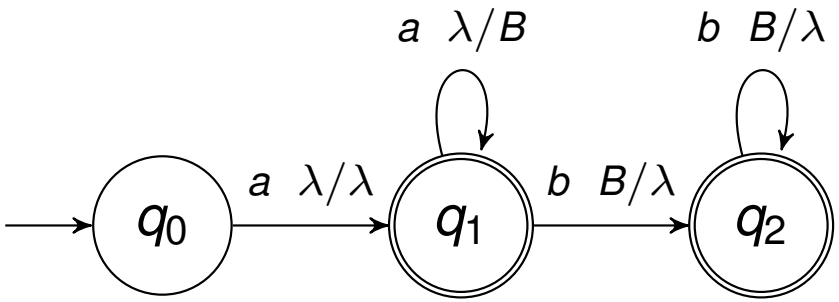
The following automata are equivalent (language: $\{a^m b^n \mid m = n + 1\}$)




overlap

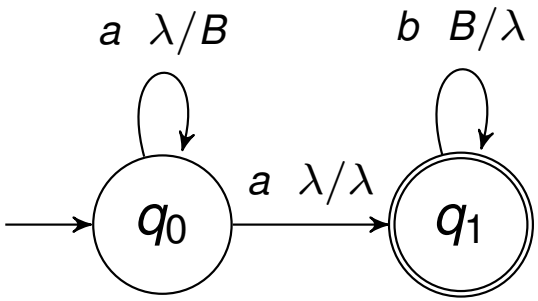



overlap

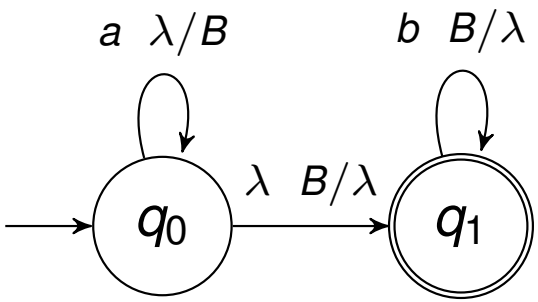


# Examples (non-)deterministic pushdown automata

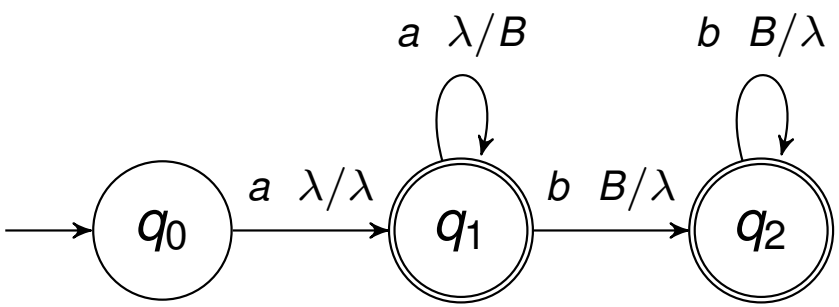
The following automata are equivalent (language: $\{a^m b^n \mid m = n + 1\}$)




overlap



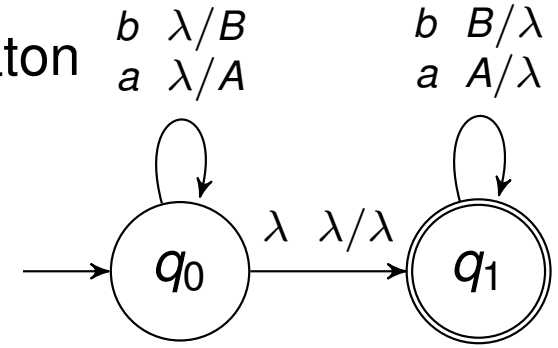

overlap




deterministic

# Strength of deterministic pushdown automata

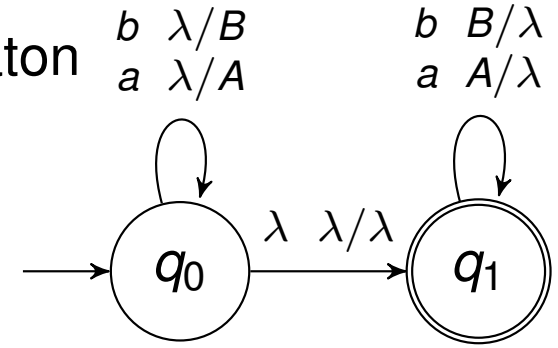
- Consider the automaton




- Language:

# Strength of deterministic pushdown automata

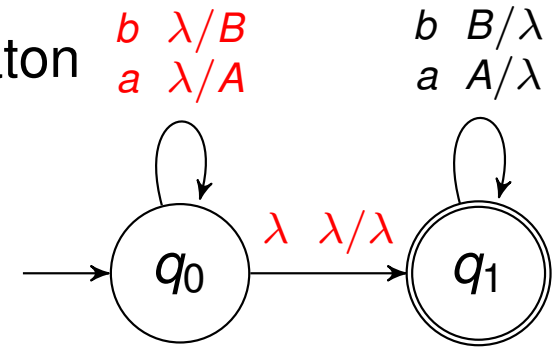
- Consider the automaton




- Language: $L = \{ww^R \mid w \in \{a, b\}^*\}$

# Strength of deterministic pushdown automata

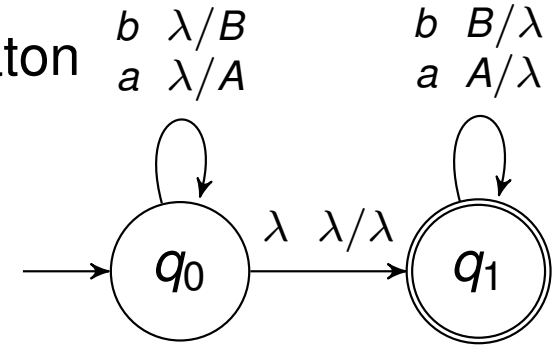
- Consider the automaton




- Language: $L = \{ww^R \mid w \in \{a, b\}^*\}$
- The outgoing transitions of $q_0$ overlap
  - When recognizing *abba* we should go to $q_1$ after *ab*
  - When recognizing *abbbba* we should remain in $q_0$ after *ab*
  - In general: the automaton must *guess* the middle of the word

# Strength of deterministic pushdown automata

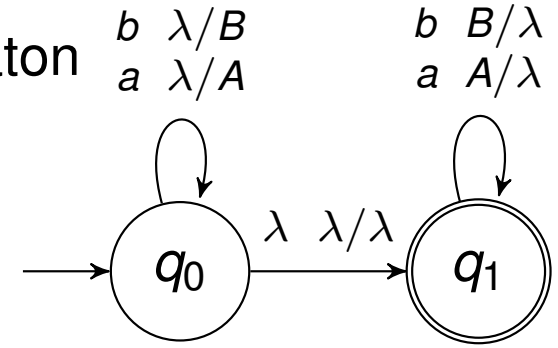
- Consider the automaton




- Language: $L = \{ww^R \mid w \in \{a, b\}^*\}$
- The outgoing transitions of $q_0$ overlap
  - When recognizing *abba* we should go to $q_1$ after *ab*
  - When recognizing *abbbba* we should remain in $q_0$ after *ab*
  - In general: the automaton must *guess* the middle of the word
- $L$ has *no* deterministic pushdown automaton!

# Strength of deterministic pushdown automata

- Consider the automaton




- Language: $L = \{ww^R \mid w \in \{a, b\}^*\}$
- The outgoing transitions of $q_0$ overlap
  - When recognizing *abba* we should go to $q_1$ after *ab*
  - When recognizing *abbbba* we should remain in $q_0$ after *ab*
  - In general: the automaton must *guess* the middle of the word
- $L$ has *no* deterministic pushdown automaton!
- Thus deterministic pushdown automata are less powerful
  - Different from finite automata

# What's next?

## Intriguing Matter

- We will find out if the following holds:
  $(L = \mathcal{L}(G)$ for a CFG $G) \Leftrightarrow (L = \mathcal{L}(M)$ for a PDA $M)$

# What's next?

## Intriguing Matter

- We will find out if the following holds:
  $(L = \mathcal{L}(G)$ for a CFG $G) \Leftrightarrow (L = \mathcal{L}(M)$ for a PDA $M)$
- Does the following also hold?
  $L = \mathcal{L}(G)$ for an unambiguous CFG $G \Leftrightarrow$
  $L = \mathcal{L}(M)$ for a deterministic PDA $M$?

# What's next?

## Intriguing Matter

- We will find out if the following holds:
  $(L = \mathcal{L}(G)$ for a CFG $G) \Leftrightarrow (L = \mathcal{L}(M)$ for a PDA $M)$
- Does the following also hold?
  $L = \mathcal{L}(G)$ for an unambiguous CFG $G \Leftrightarrow$
  $L = \mathcal{L}(M)$ for a deterministic PDA $M$?
- No! Unambiguous grammar for $ww^R$: $S \to aSa \mid bSb \mid \lambda$
  Each PDA for palindromes must "guess" the middle.

# What's next?

## Intriguing Matter

- We will find out if the following holds:
  $(L = \mathcal{L}(G)$ for a CFG $G) \Leftrightarrow (L = \mathcal{L}(M)$ for a PDA $M)$
- Does the following also hold?
  $L = \mathcal{L}(G)$ for an unambiguous CFG $G \Leftrightarrow$
  $L = \mathcal{L}(M)$ for a deterministic PDA $M$?
- No! Unambiguous grammar for $ww^R$: $S \to aSa \mid bSb \mid \lambda$
  Each PDA for palindromes must "guess" the middle.

The class of DPDAs is quite different from the class of PDAs

http://en.wikipedia.org/wiki/Deterministic_pushdown_automaton

# Contents

# Playing with acceptance conditions

## Definition: Acceptance by final state

The language $L_F(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$ with a computation,

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \alpha]$ with $q \in F$ and $\alpha \in \Gamma^*$.

*Remark:* The $F$ in $L_F$ stands for “final”.

# Playing with acceptance conditions

## Definition: Acceptance by final state

The language $L_F(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$ with a computation,

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \alpha]$ with $q \in F$ and $\alpha \in \Gamma^*$.

*Remark:* The $F$ in $L_F$ stands for "final".

## Definition: Acceptance by empty stack

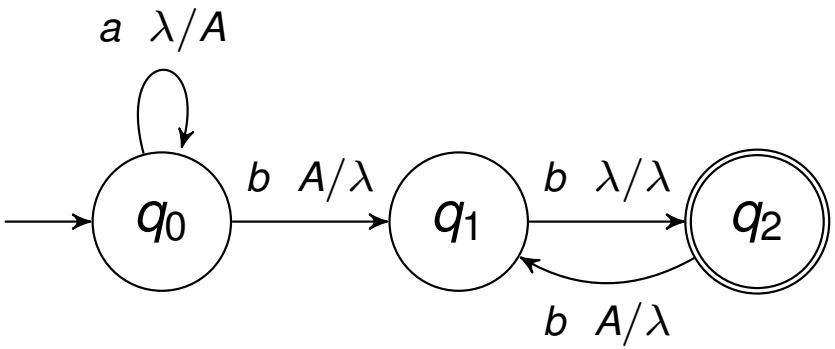
The language $L_E(M)$ of a pushdown automaton $M = \langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$ is the set of words $w$ for which a *non-empty* computation exists, such that

- beginning in $[q_0, w, \lambda]$
- ending in $[q, \lambda, \lambda]$ for some $q \in Q$.

*Remark:* $E$ in $L_E$ stands for "empty". The $F$ is now superfluous.

# Alternative acceptance: Example

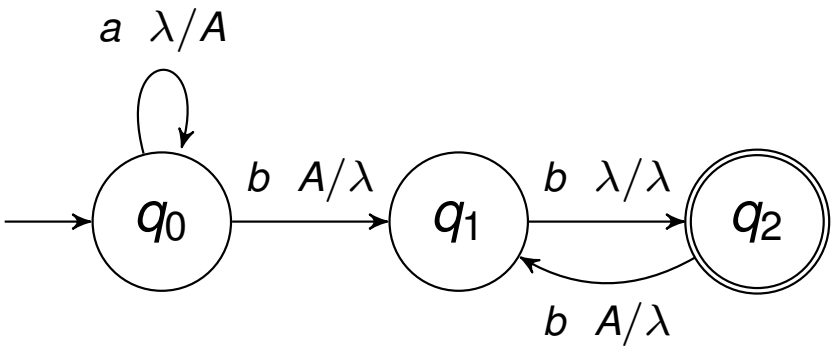
Example pushdown automaton:




- $L(M) =$
- $L_F(M) =$
- $L_E(M) =$

# Alternative acceptance: Example

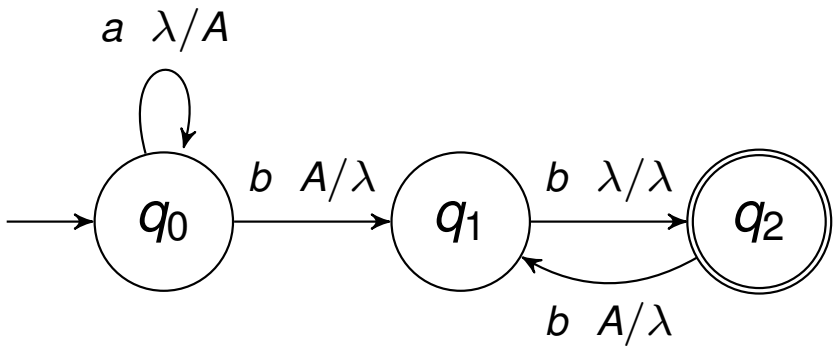
Example pushdown automaton:




- $L(M) = \{a^n b^{2n} \mid n > 0\}$
- $L_F(M) =$
- $L_E(M) =$

# Alternative acceptance: Example

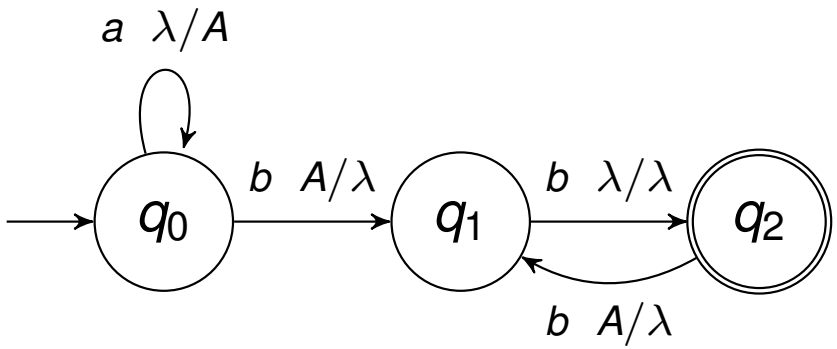
Example pushdown automaton:




- $L(M) = \{a^n b^{2n} \mid n > 0\}$
- $L_F(M) = \{a^n b^{2k} \mid n > 0, 0 < k \leq n\}$
  ... (every computation that ends in $q_2$ is done)
- $L_E(M) =$

# Alternative acceptance: Example

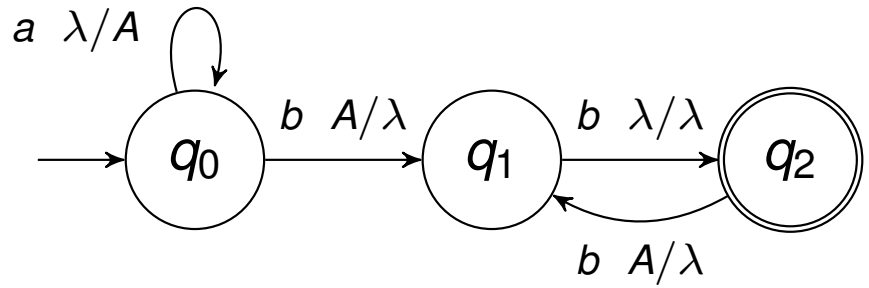
Example pushdown automaton:




- $L(M) = \{a^n b^{2n} \mid n > 0\}$
- $L_F(M) = \{a^n b^{2k} \mid n > 0, 0 < k \leq n\}$
  ... (every computation that ends in $q_2$ is done)
- $L_E(M) = \{a^n(b^{2n-1} \cup b^{2n}) \mid n > 0\}$
  ... (the stack is already empty in $q_1$, so the computation is done)

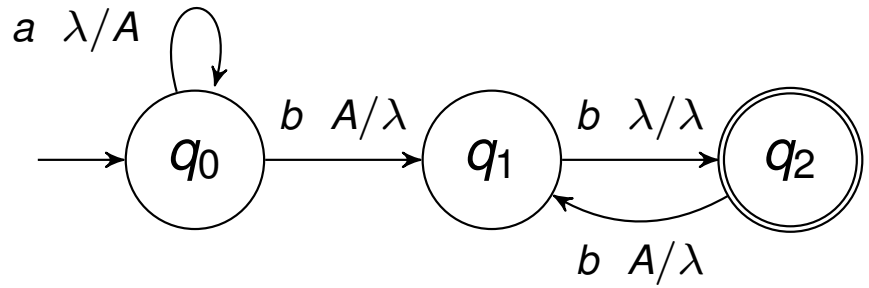
# From and to acceptance by final state: Example

- $L(M) = \{a^n b^{2n} \mid n>0\}$ and $L_F(M) = \{a^n b^{2k} \mid n>0, 0<k\leq n\}$:



# From and to acceptance by final state: Example

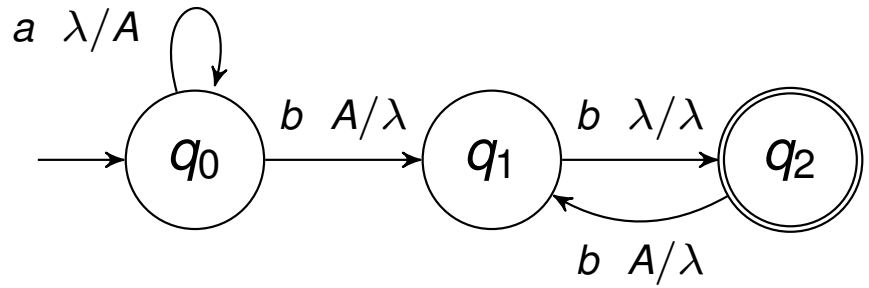
- $L(M) = \{a^n b^{2n} \mid n>0\}$ and $L_F(M) = \{a^n b^{2k} \mid n>0, 0<k\leq n\}$:



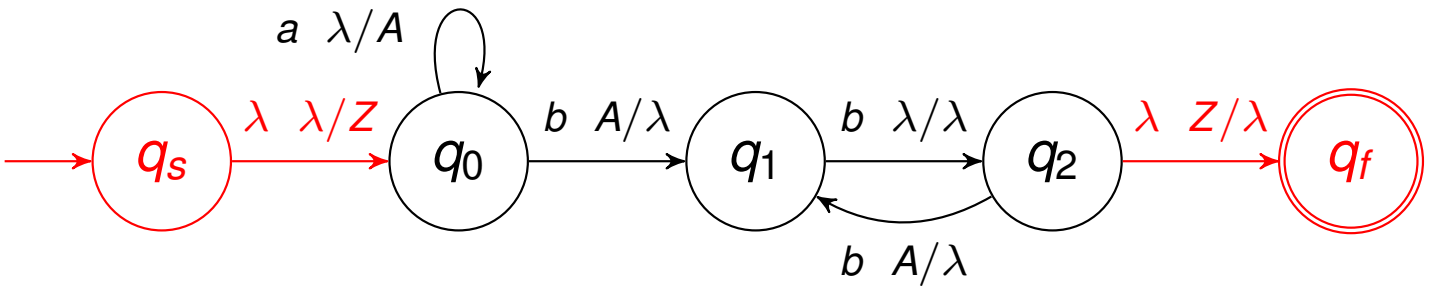

- $M_1$ such that $L_F(M_1) = L(M)$:

# From and to acceptance by final state: Example

- $L(M) = \{a^n b^{2n} \mid n>0\}$ and $L_F(M) = \{a^n b^{2k} \mid n>0, 0<k\leq n\}$:

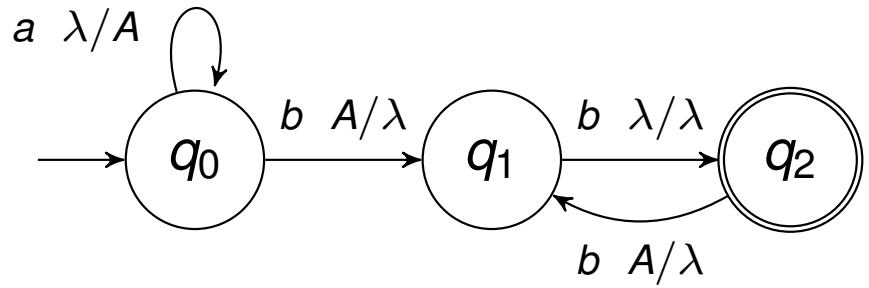



- $M_1$ such that $L_F(M_1) = L(M)$:

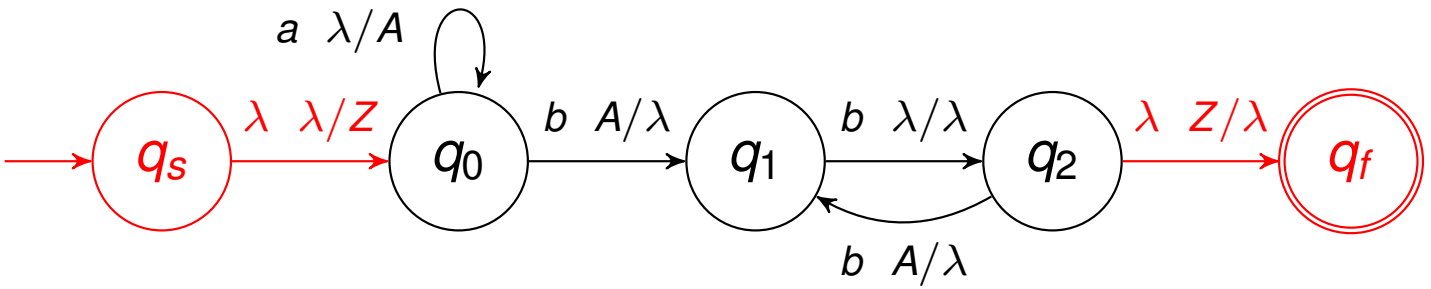


# From and to acceptance by final state: Example

- $L(M) = \{a^n b^{2n} \mid n>0\}$ and $L_F(M) = \{a^n b^{2k} \mid n>0, 0<k\leq n\}$:



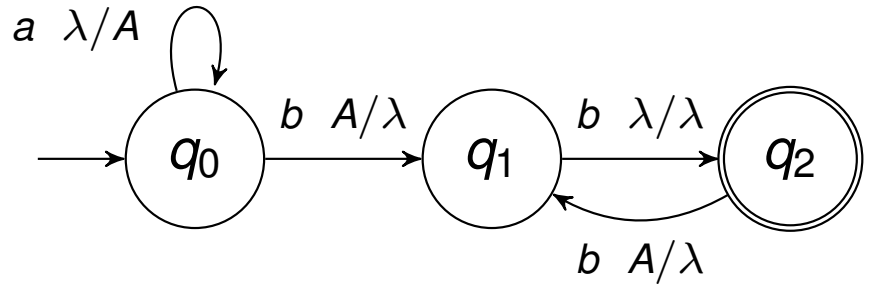

- $M_1$ such that $L_F(M_1) = L(M)$:



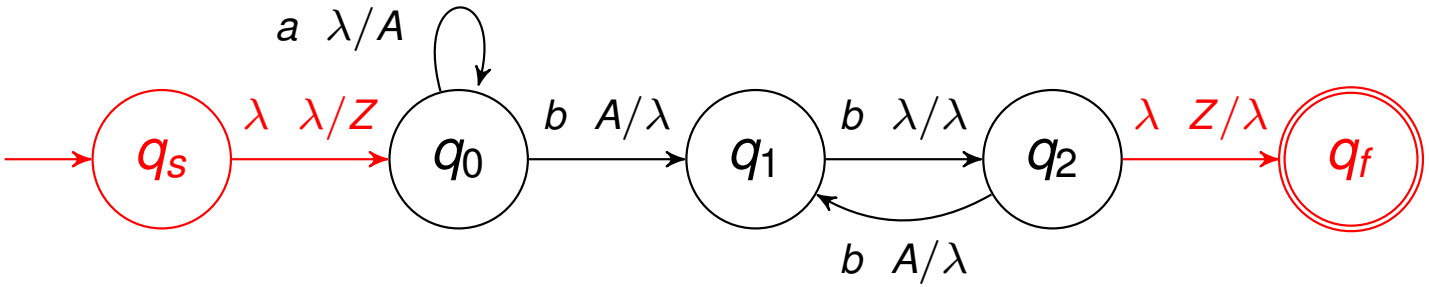

- $M_2$ such that $L(M_2) = L_F(M)$:

# From and to acceptance by final state: Example

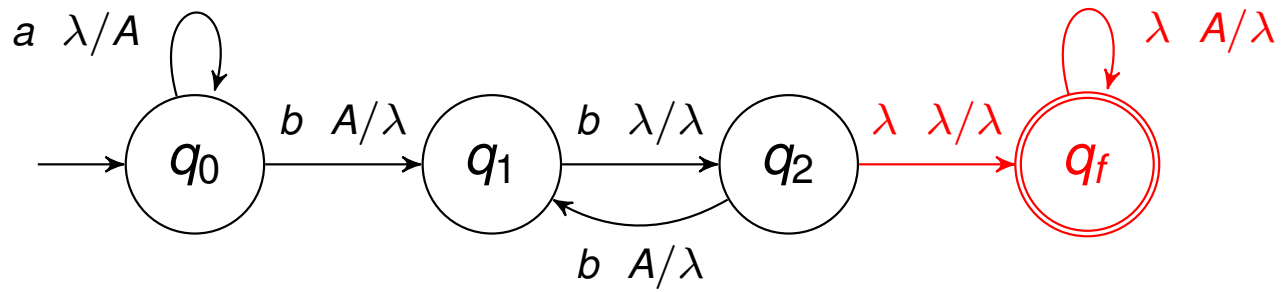
- $L(M) = \{a^n b^{2n} \mid n>0\}$ and $L_F(M) = \{a^n b^{2k} \mid n>0, 0<k\leq n\}$:




- $M_1$ such that $L_F(M_1) = L(M)$:




- $M_2$ such that $L(M_2) = L_F(M)$:



# Acceptance by final state

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_F$ exists such that $L_F(M_F) = X$

# Acceptance by final state

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_F$ exists such that $L_F(M_F) = X$

Differently formulated:

**Lemma: Transformation from and to acceptance by final state**

1. For every PDA $M$ a PDA $M'$ exists such that $L_F(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_F(M)$

# Acceptance by final state

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_F$ exists such that $L_F(M_F) = X$

Differently formulated:

**Lemma: Transformation from and to acceptance by final state**

1. For every PDA $M$ a PDA $M'$ exists such that $L_F(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_F(M)$

**Construction**

# Acceptance by final state

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_F$ exists such that $L_F(M_F) = X$

Differently formulated:

**Lemma: Transformation from and to acceptance by final state**

1. For every PDA $M$ a PDA $M'$ exists such that $L_F(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_F(M)$

**Construction**

1. $Q' = Q \uplus \{q_s, q_f\}$, $\Gamma' = \Gamma \uplus \{Z\}$, $F' = \{q_f\}$;
   new transformations $q_s \xrightarrow{\lambda \ \lambda/Z} q_0$, and $q \xrightarrow{\lambda \ Z/\lambda} q_f$ (all $q \in F$)

# Acceptance by final state

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_F$ exists such that $L_F(M_F) = X$

Differently formulated:

**Lemma: Transformation from and to acceptance by final state**

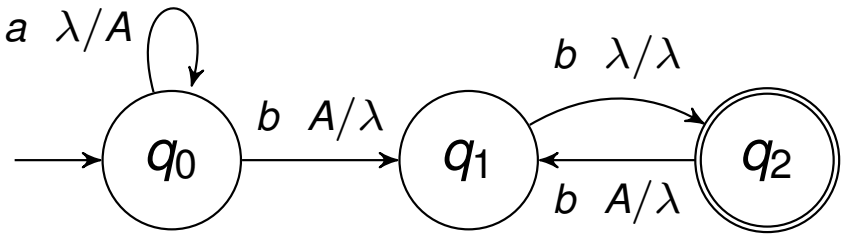
1. For every PDA $M$ a PDA $M'$ exists such that $L_F(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_F(M)$

**Construction**

1. $Q' = Q \uplus \{q_s, q_f\}$, $\Gamma' = \Gamma \uplus \{Z\}$, $F' = \{q_f\}$;
   new transformations $q_s \xrightarrow{\lambda \ \lambda/Z} q_0$, and $q \xrightarrow{\lambda \ Z/\lambda} q_f$ (all $q \in F$)
2. $Q' = Q \uplus \{q_f\}$, $F' = \{q_f\}$; new transformations
   $q \xrightarrow{\lambda \ \lambda/\lambda} q_f$ (all $q \in F$), and $q_f \xrightarrow{\lambda \ A/\lambda} q_f$ (all $A \in \Gamma$)

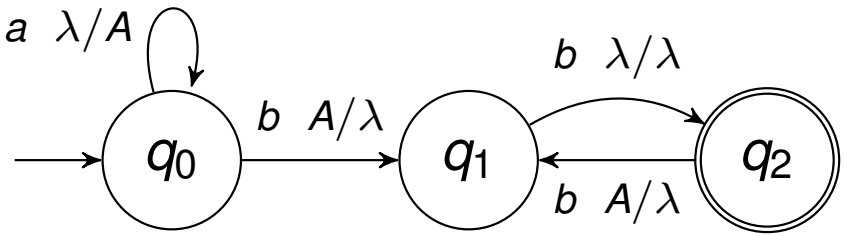
# From and to acceptance by empty stack: Example

- $M$, with $L_E(M) = \{a^n(b^{2n-1} \cup b^{2n}) \mid n > 0\}$:



# From and to acceptance by empty stack: Example

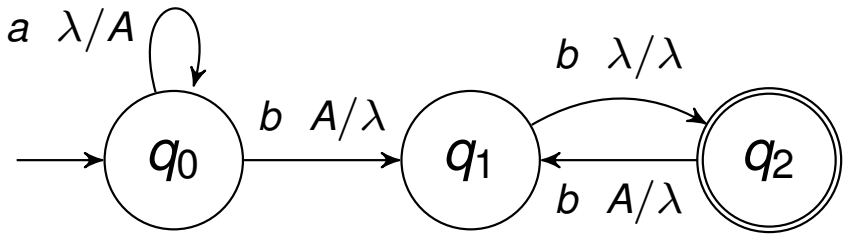
- $M$, with $L_E(M) = \{a^n(b^{2n-1} \cup b^{2n}) \mid n > 0\}$:



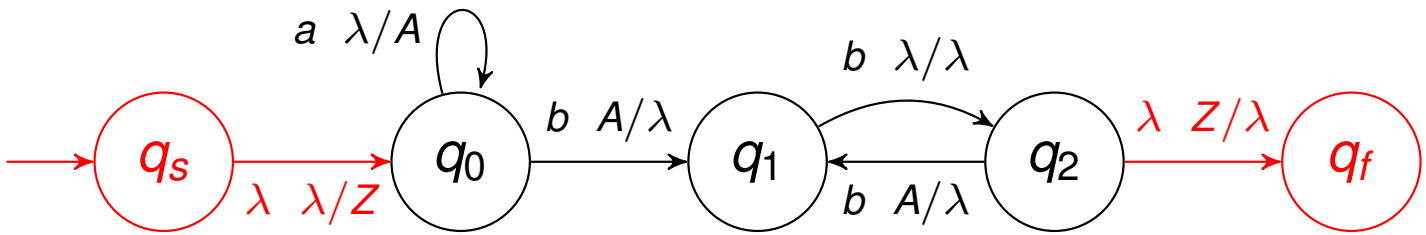

- $M_3$ such that $L_E(M_3) = L(M)$:

# From and to acceptance by empty stack: Example

- $M$, with $L_E(M) = \{a^n(b^{2n-1} \cup b^{2n}) \mid n > 0\}$:

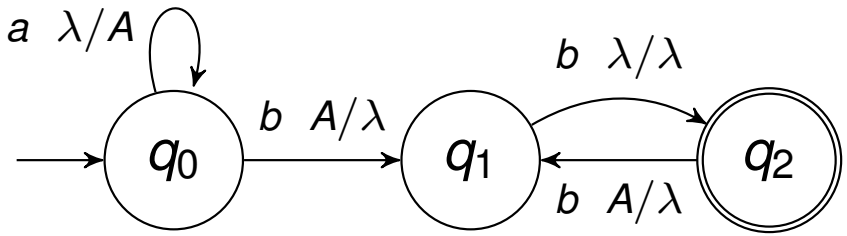



- $M_3$ such that $L_E(M_3) = L(M)$: Identical to $M_1$, except $F_3 = \emptyset$:

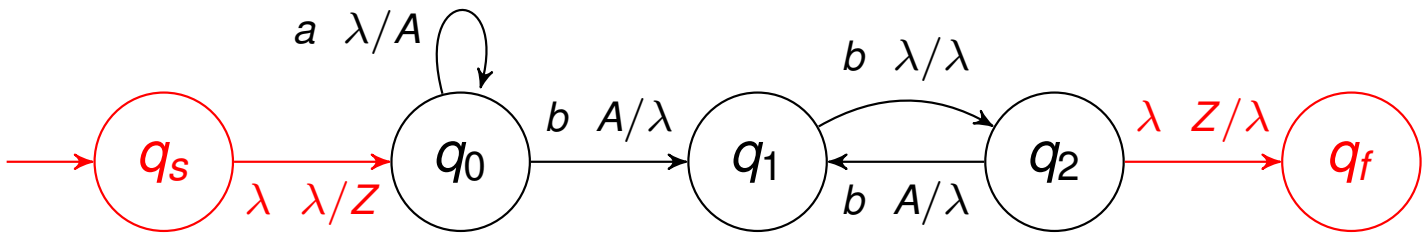


# From and to acceptance by empty stack: Example

- $M$, with $L_E(M) = \{a^n(b^{2n-1} \cup b^{2n}) \mid n > 0\}$:



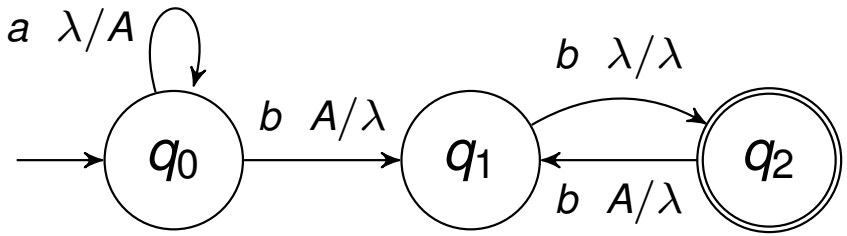

- $M_3$ such that $L_E(M_3) = L(M)$: Identical to $M_1$, except $F_3 = \emptyset$:



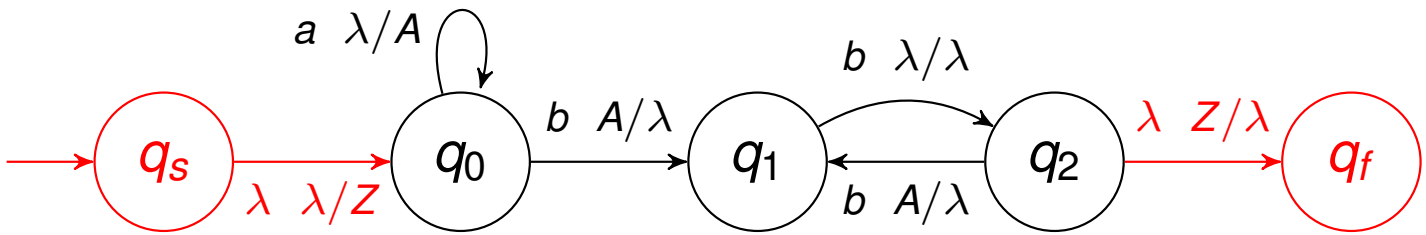

- $M_4$ such that $L(M_4) = L_E(M)$:

# From and to acceptance by empty stack: Example

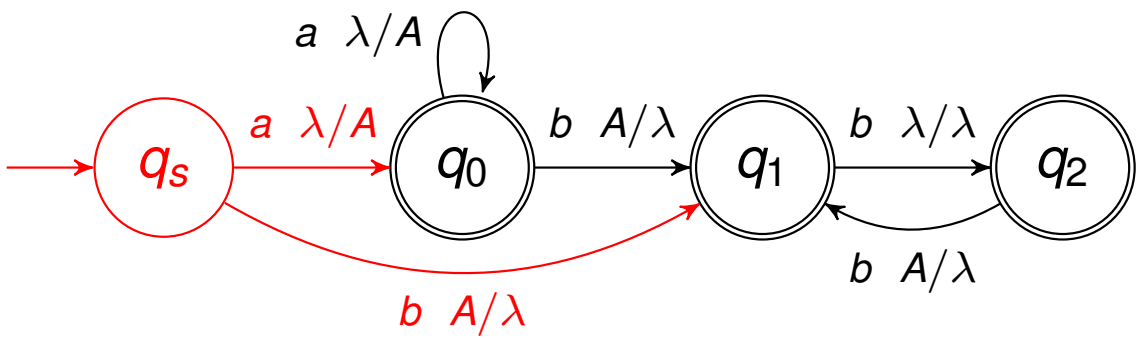
- $M$, with $L_E(M) = \{a^n(b^{2n-1} \cup b^{2n}) \mid n > 0\}$:




- $M_3$ such that $L_E(M_3) = L(M)$: Identical to $M_1$, except $F_3 = \emptyset$:




- $M_4$ such that $L(M_4) = L_E(M)$:



# Acceptance by empty stack

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_E$ exists such that $L_E(M_E) = X$

# Acceptance by empty stack

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_E$ exists such that $L_E(M_E) = X$

This is equivalent to the following statements:

**Lemma: Transformation from and to acceptance by empty stack**

1. For every PDA $M$ a PDA $M'$ exists such that $L_E(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_E(M)$

# Acceptance by empty stack

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_E$ exists such that $L_E(M_E) = X$

This is equivalent to the following statements:

**Lemma: Transformation from and to acceptance by empty stack**

1. For every PDA $M$ a PDA $M'$ exists such that $L_E(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_E(M)$

**Construction**

# Acceptance by empty stack

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_E$ exists such that $L_E(M_E) = X$

This is equivalent to the following statements:

**Lemma: Transformation from and to acceptance by empty stack**

1. For every PDA $M$ a PDA $M'$ exists such that $L_E(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_E(M)$

**Construction**

1. The same as for acceptance by final state!

# Acceptance by empty stack

**Theorem: For every language $X \subseteq \Sigma^*$ the following is equivalent:**

- A PDA $M$ exists such that $L(M) = X$;
- A PDA $M_E$ exists such that $L_E(M_E) = X$

This is equivalent to the following statements:

**Lemma: Transformation from and to acceptance by empty stack**

1. For every PDA $M$ a PDA $M'$ exists such that $L_E(M') = L(M)$
2. For every PDA $M$ a PDA $M'$ exists such that $L(M') = L_E(M)$

**Construction**

1. The same as for acceptance by final state!
2. $Q' = Q \uplus \{q_s\}$, $\Gamma' = \Gamma$, $F' = Q$; new transitions $q_s \xrightarrow{x \quad A/B} q'$ for all $q_0 \xrightarrow{x \quad A/B} q'$

# Extended pushdown automata

## Definition

An *extended* pushdown automata has a transition function

$$\delta \subseteq Q \times \Sigma^{\lambda} \times \Gamma^{\lambda} \times \Gamma^{*} \times Q$$

instead of

$$\delta \subseteq Q \times \Sigma^{\lambda} \times \Gamma^{\lambda} \times \Gamma^{\lambda} \times Q$$

and thus transitions of the form $q \xrightarrow{a \;\; A/W} q'$ with $W \in \Gamma^{*}$.

# Extended pushdown automata

## Definition

An *extended* pushdown automata has a transition function

$$\delta \subseteq Q \times \Sigma^{\lambda} \times \Gamma^{\lambda} \times \Gamma^{*} \times Q$$

instead of

$$\delta \subseteq Q \times \Sigma^{\lambda} \times \Gamma^{\lambda} \times \Gamma^{\lambda} \times Q$$

and thus transitions of the form $q \xrightarrow{a \;\; A/W} q'$ with $W \in \Gamma^{*}$.

- Extended pushdown automata can push multiple symbols simultaneously
- Results in more compact automata

# Extended pushdown automata

## Definition

An *extended* pushdown automata has a transition function

$$\delta \subseteq Q \times \Sigma^{\lambda} \times \Gamma^{\lambda} \times \Gamma^{*} \times Q$$

instead of

$$\delta \subseteq Q \times \Sigma^{\lambda} \times \Gamma^{\lambda} \times \Gamma^{\lambda} \times Q$$

and thus transitions of the form $q \xrightarrow{a \;\; A/W} q'$ with $W \in \Gamma^{*}$.

- Extended pushdown automata can push multiple symbols simultaneously
- Results in more compact automata

## Transformation from extended to normal automata

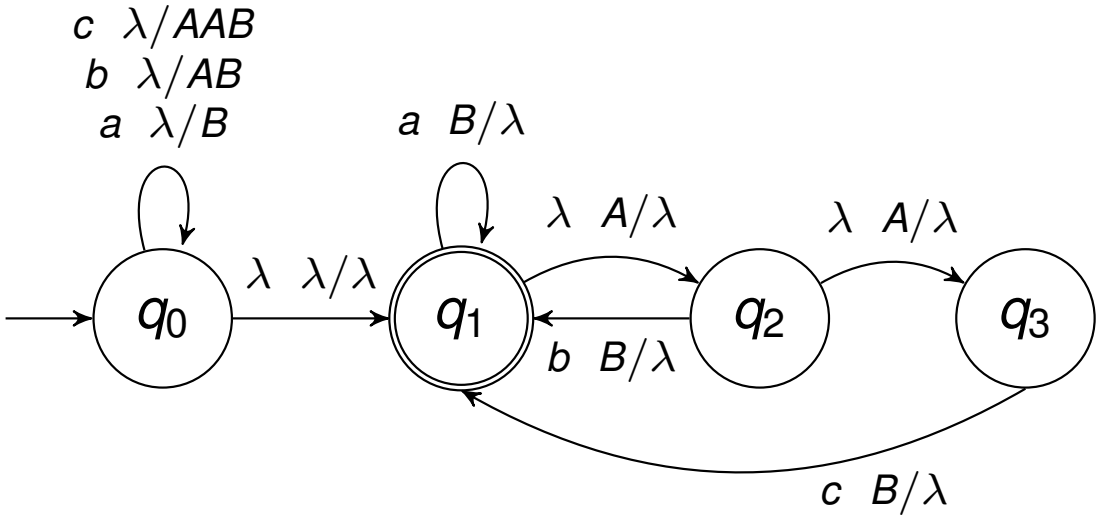
An equivalent normal PDA exists for every extended PDA.

# Extended pushdown automata: example

- Language: “even” palindromes over $\{a, b, c\}$

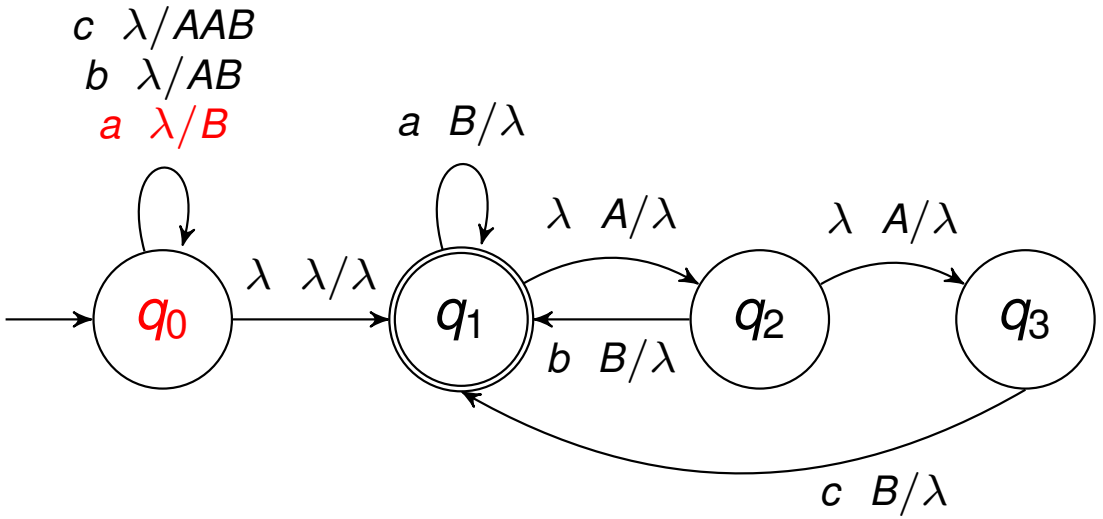
# Extended pushdown automata: example

- Language: "even" palindromes over $\{a, b, c\}$



# Extended pushdown automata: example

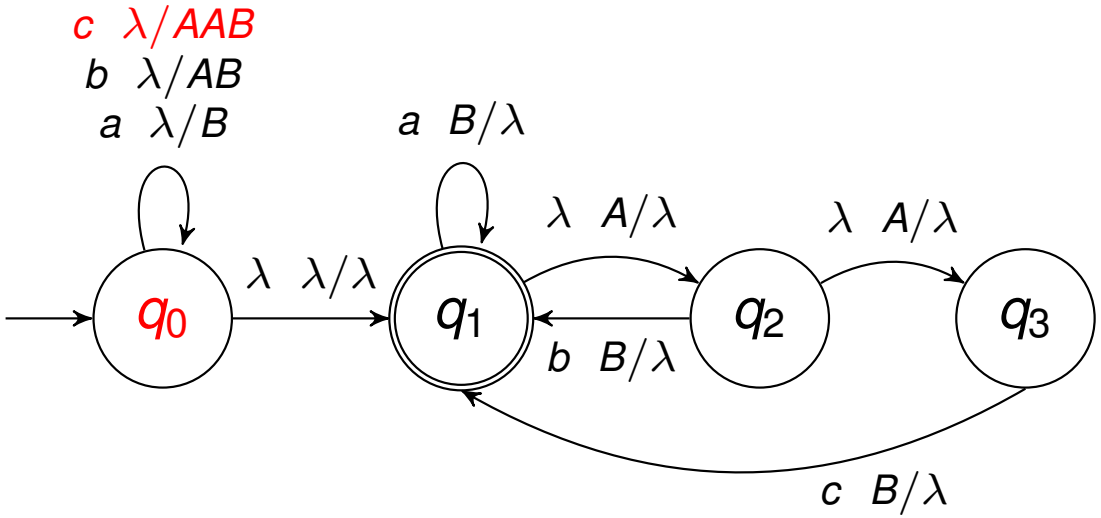
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash$

# Extended pushdown automata: example

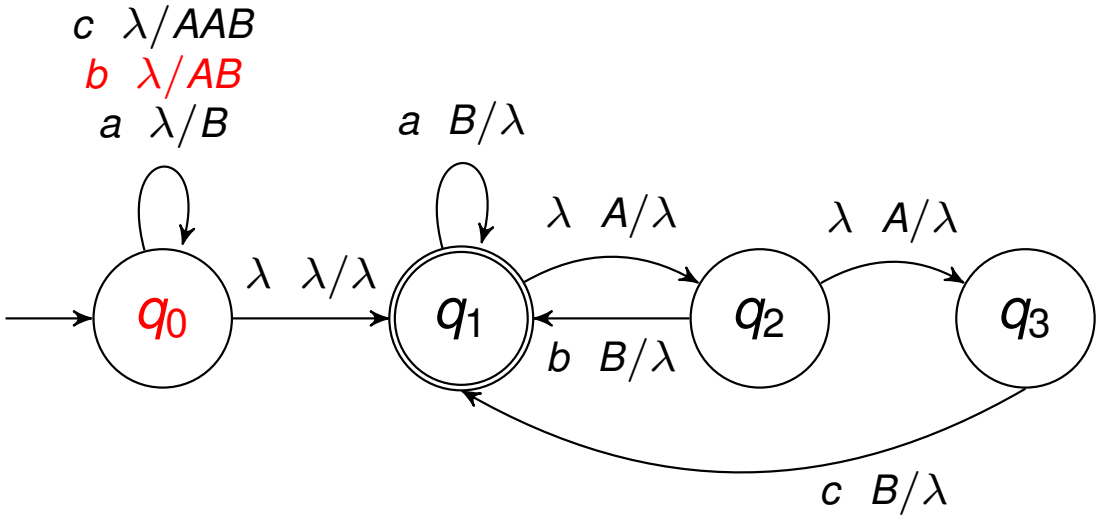
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash$

# Extended pushdown automata: example

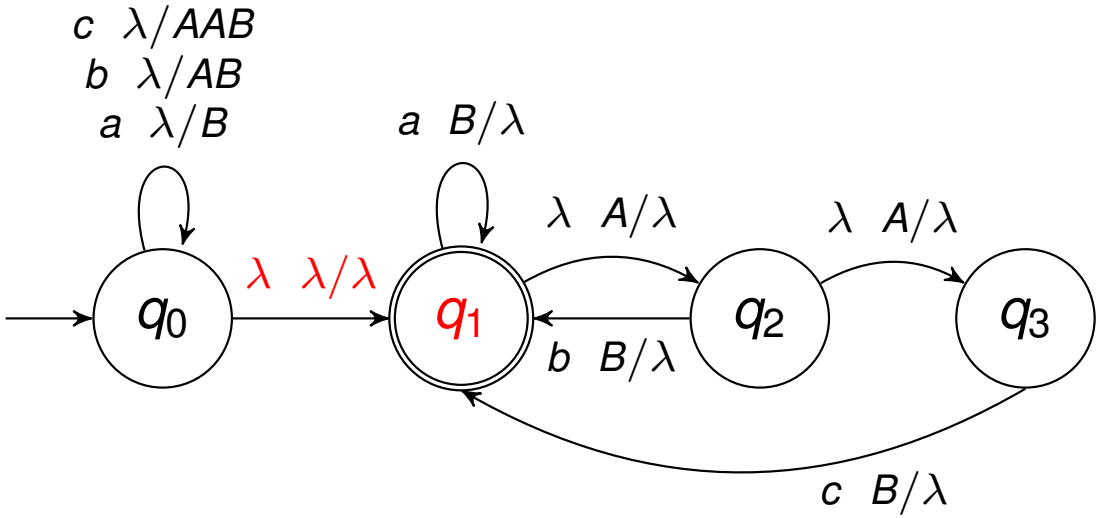
- Language: "even" palindromes over $\{a, b, c\}$




$$
\begin{aligned}
&[q_0, acbbca, \lambda] \\
&\vdash [q_0, cbbca, B] \\
&\vdash [q_0, bbca, AABB] \\
&\vdash [q_0, bca, ABAABB] \\
&\vdash
\end{aligned}
$$

# Extended pushdown automata: example

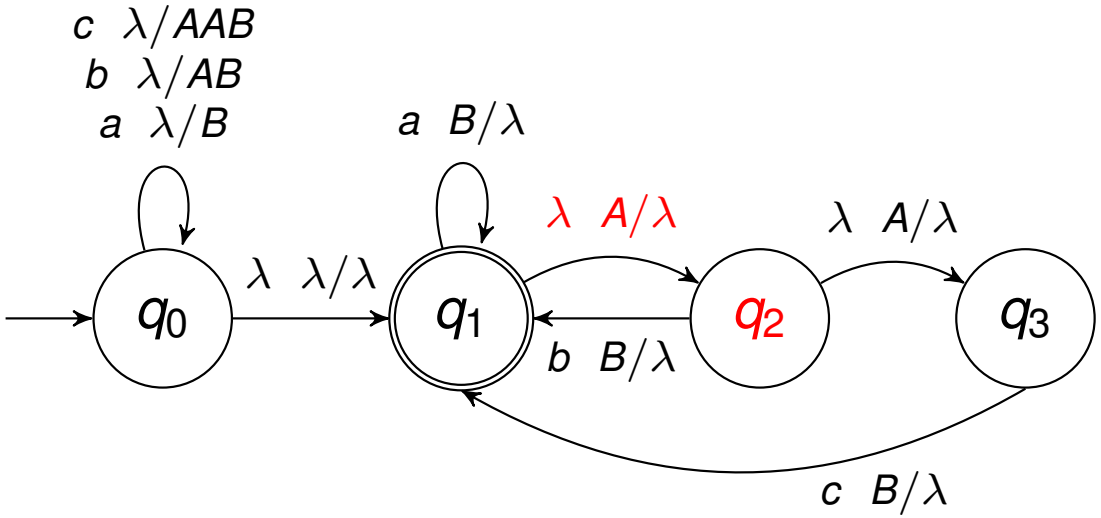
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash$

# Extended pushdown automata: example

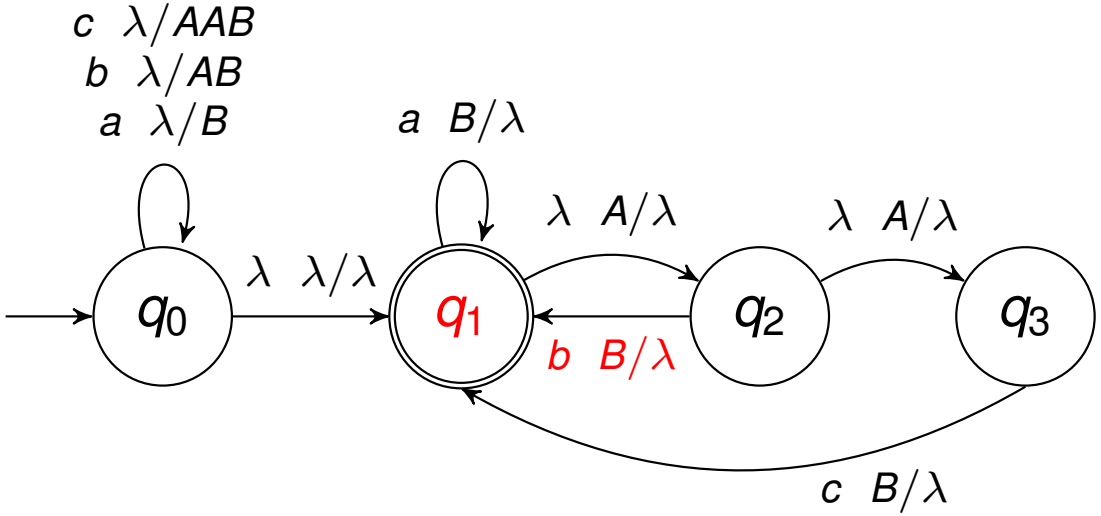
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash [q_2, bca, BAABB]$
$\vdash$

# Extended pushdown automata: example

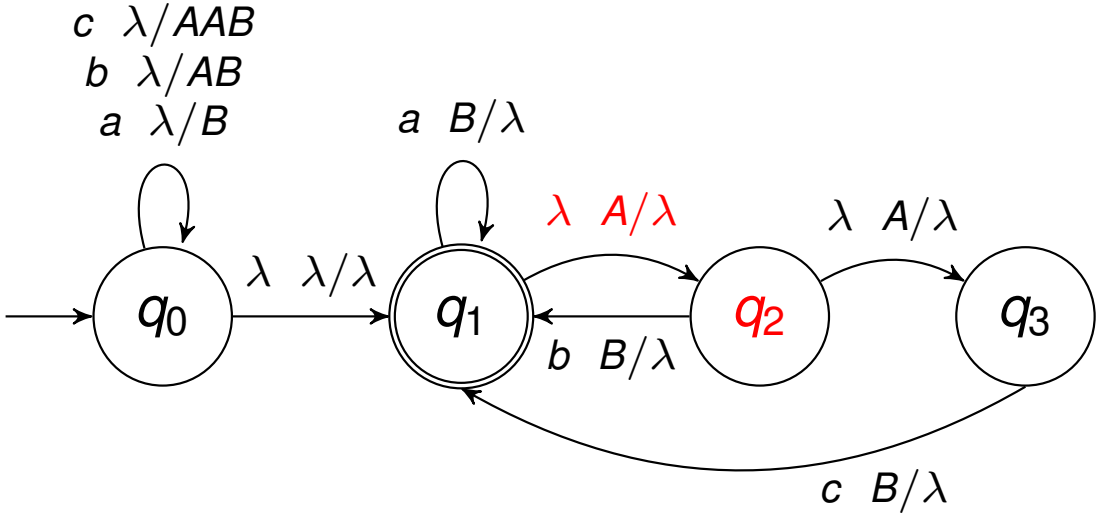
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash [q_2, bca, BAABB]$
$\vdash [q_1, ca, AABB]$
$\vdash$

# Extended pushdown automata: example

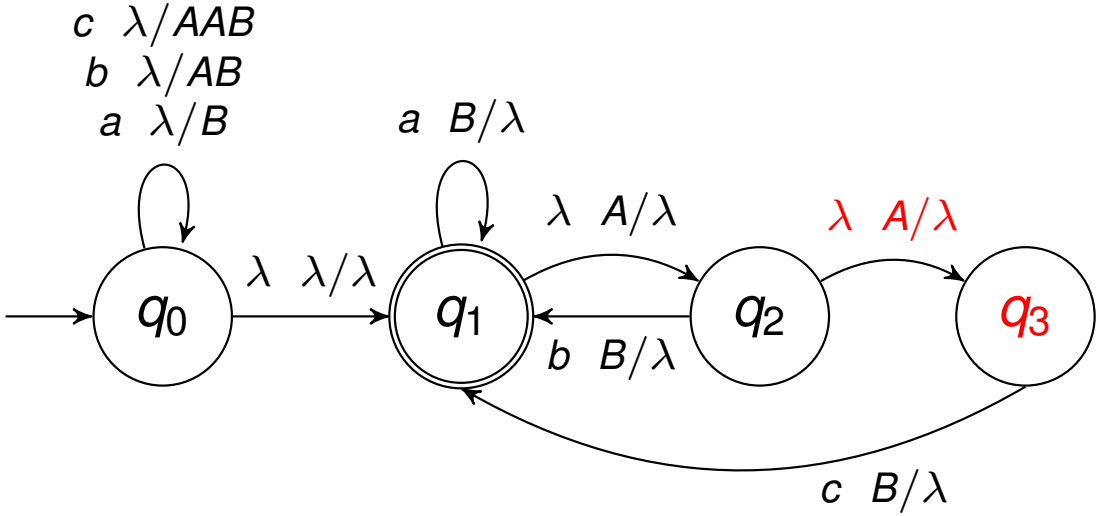
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash [q_2, bca, BAABB]$
$\vdash [q_1, ca, AABB]$
$\vdash [q_2, ca, ABB]$
$\vdash$

# Extended pushdown automata: example

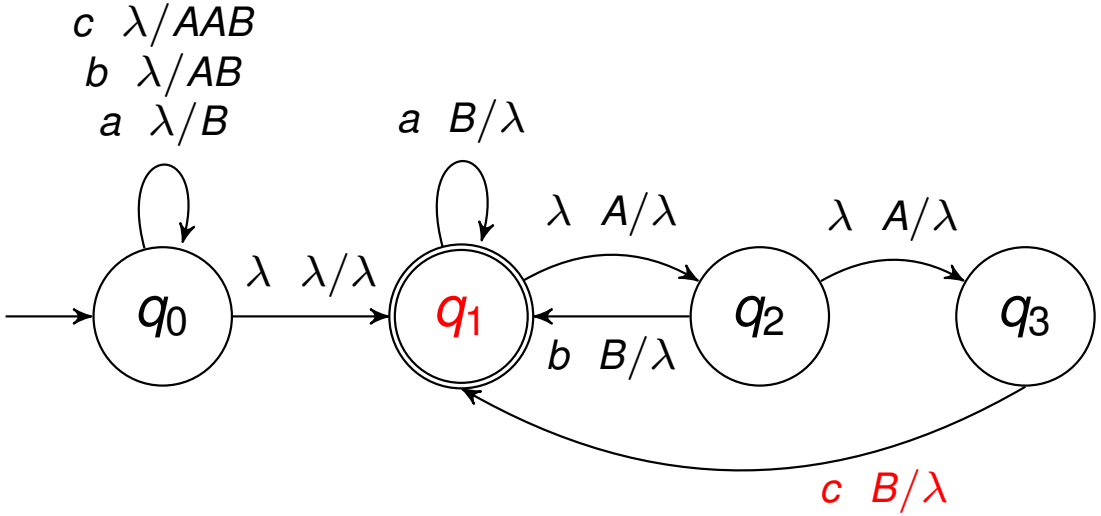
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash [q_2, bca, BAABB]$
$\vdash [q_1, ca, AABB]$
$\vdash [q_2, ca, ABB]$
$\vdash [q_3, ca, BB]$
$\vdash$

# Extended pushdown automata: example

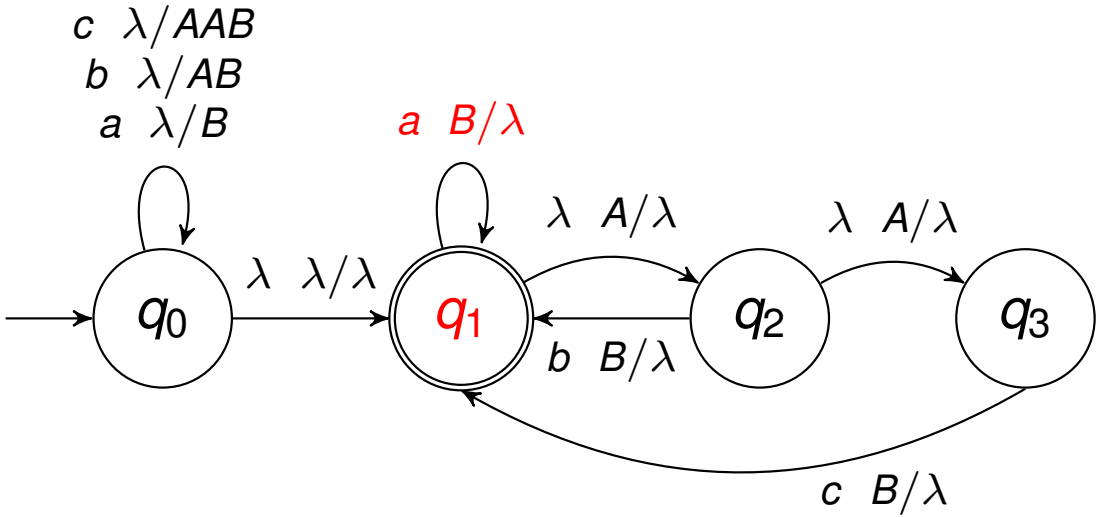
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash [q_2, bca, BAABB]$
$\vdash [q_1, ca, AABB]$
$\vdash [q_2, ca, ABB]$
$\vdash [q_3, ca, BB]$
$\vdash [q_1, a, B]$
$\vdash$

# Extended pushdown automata: example

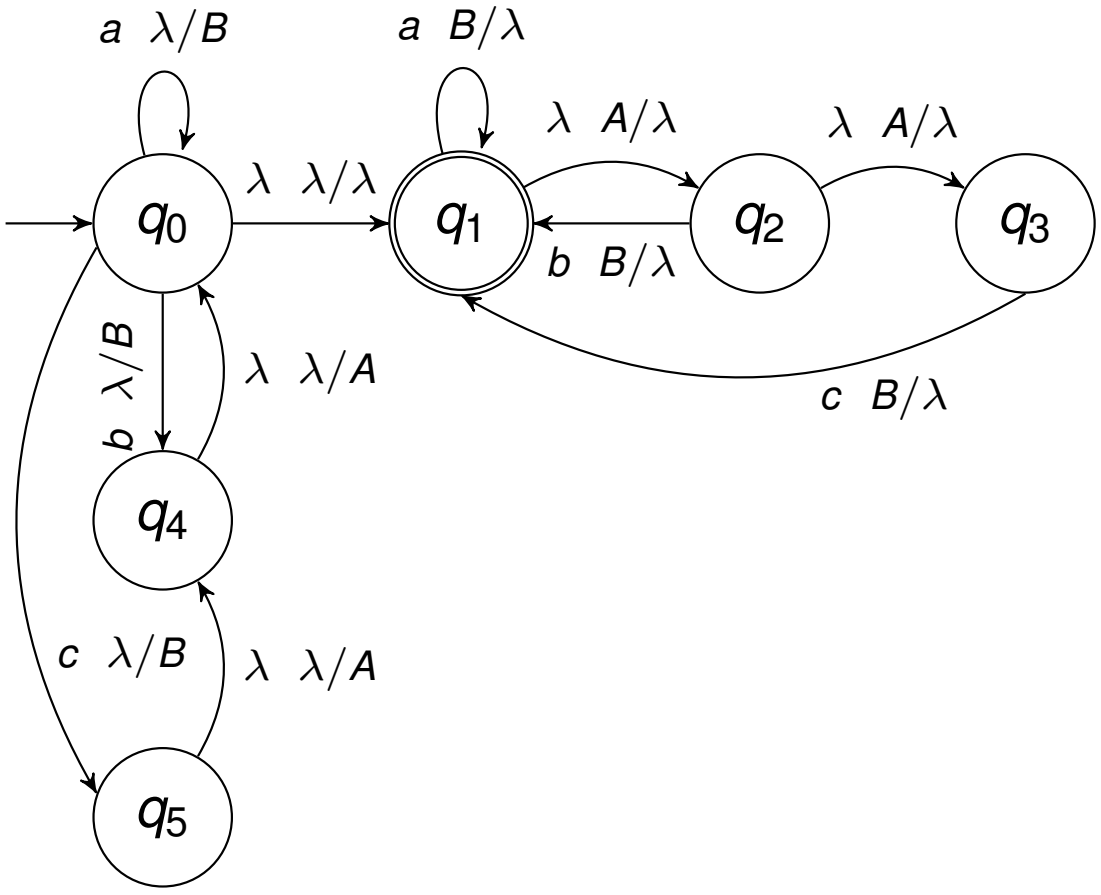
- Language: "even" palindromes over $\{a, b, c\}$




$[q_0, acbbca, \lambda]$
$\vdash [q_0, cbbca, B]$
$\vdash [q_0, bbca, AABB]$
$\vdash [q_0, bca, ABAABB]$
$\vdash [q_1, bca, ABAABB]$
$\vdash [q_2, bca, BAABB]$
$\vdash [q_1, ca, AABB]$
$\vdash [q_2, ca, ABB]$
$\vdash [q_3, ca, BB]$
$\vdash [q_1, a, B]$
$\vdash [q_1, \lambda, \lambda]$

# Extended pushdown automata: example

- Language: "even" palindromes over $\{a, b, c\}$




- Equivalent normal pushdown automaton

# Contents

# From CFGs to PDAs

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

# From CFGs to PDAs

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

## Theorem 1

An *extended* PDA $M$ exists for every GNF $G$ such that $L(M) = L(G)$ (even one with 2 states!)

# From CFGs to PDAs

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

## Theorem 1

An *extended* PDA $M$ exists for every GNF $G$ such that $L(M) = L(G)$ (even one with 2 states!)

## Theorem 2

A GNF $G'$ exists for every CFG $G$ such that $L(G) = L(G')$ (continue transforming)

# From CFGs to PDAs

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

## Theorem 1

An *extended* PDA $M$ exists for every GNF $G$ such that $L(M) = L(G)$ (even one with 2 states!)

## Theorem 2

A GNF $G'$ exists for every CFG $G$ such that $L(G) = L(G')$ (continue transforming)

## Theorem 3

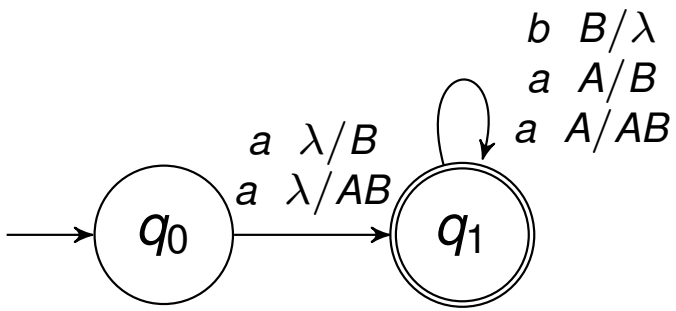
A CFG $G$ exists for every PDA $M$ such that $L(M) = L(G)$ (it's possible, but a very tedious construction)

# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

- Constructed (extended) automaton:



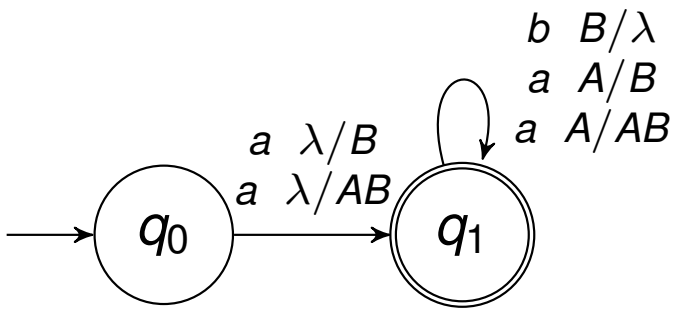
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

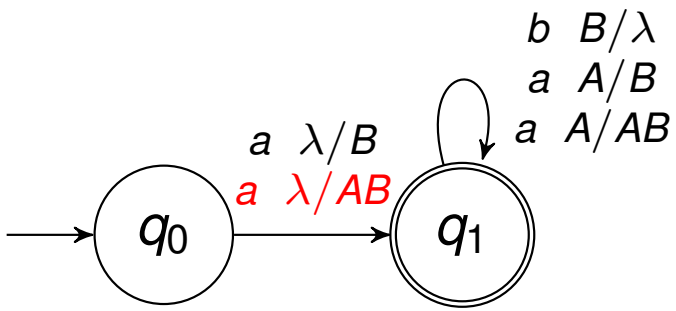
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

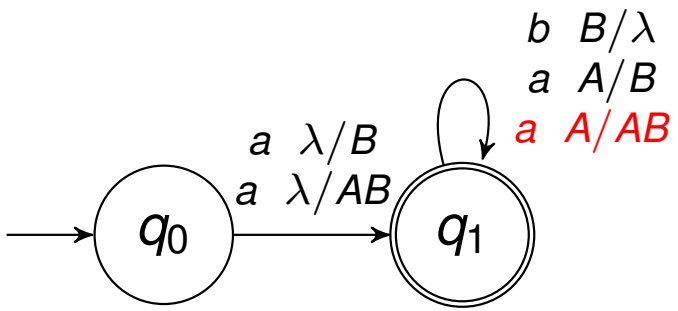
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

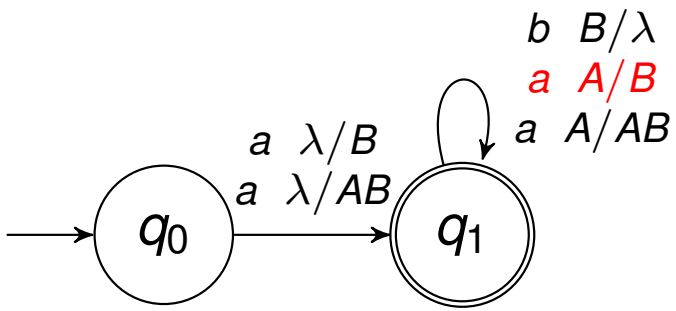
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

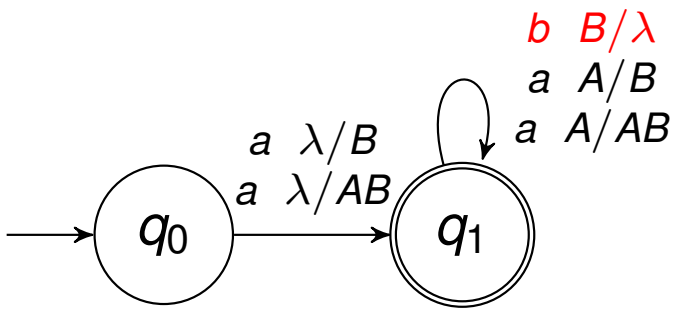
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

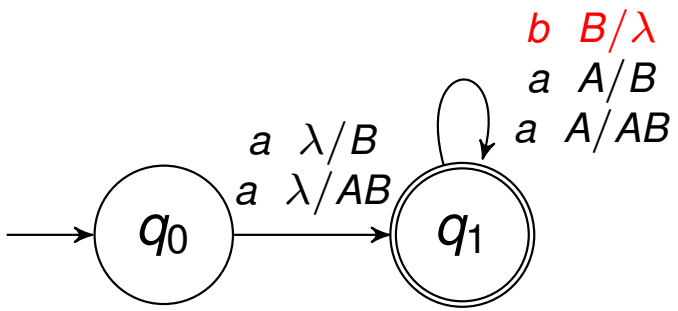
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

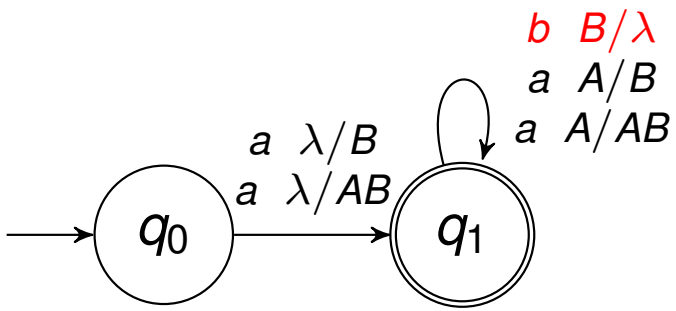
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

  $S \to aAB \mid aB$
  $A \to aAB \mid aB$
  $B \to b$

  $S \Rightarrow aAB \Rightarrow aaABB$
  $\Rightarrow aaaBBB \Rightarrow aaabBB$
  $\Rightarrow aaabbB \Rightarrow aaabbb$

- Constructed (extended) automaton:




$[q_0, aaabbb, \lambda]$
$\vdash [q_1, aabbb, AB]$
$\vdash [q_1, abbb, ABB]$
$\vdash [q_1, bbb, BBB]$
$\vdash [q_1, bb, BB]$
$\vdash [q_1, b, B]$
$\vdash [q_1, \lambda, \lambda]$

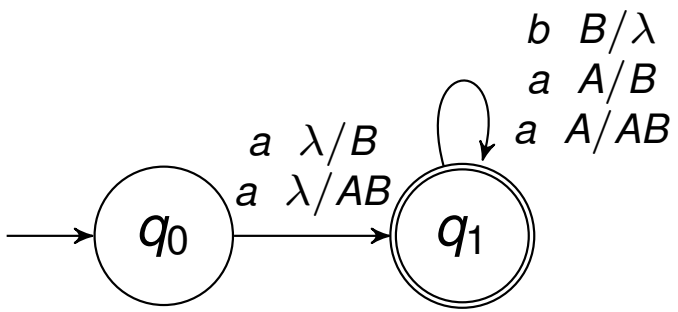
# From CFGs to PDAs: Example

- GNF for $\{a^i b^i \mid i > 0\}$:

$$S \to aAB \mid aB$$
$$A \to aAB \mid aB$$
$$B \to b$$

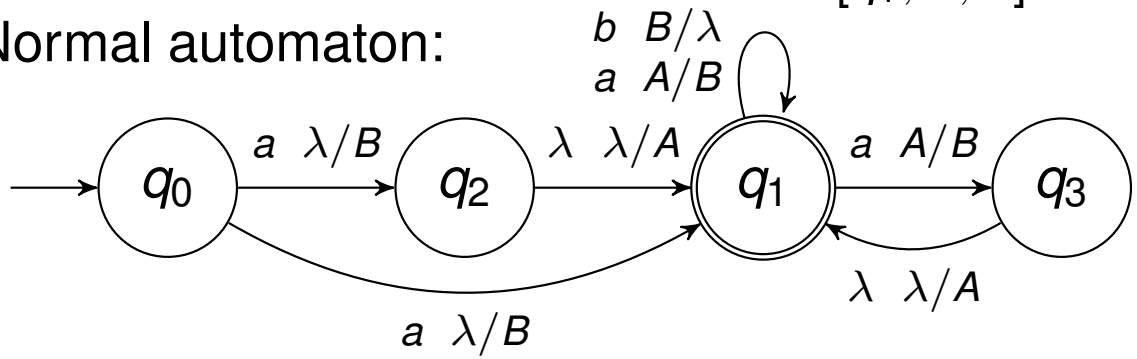
$$S \Rightarrow aAB \Rightarrow aaABB \Rightarrow aaaBBB \Rightarrow aaabBB \Rightarrow aaaabbB \Rightarrow aaabbb$$

- Constructed (extended) automaton:




$$[q_0, aaabbb, \lambda] \vdash [q_1, aabbb, AB] \vdash [q_1, abbb, ABB] \vdash [q_1, bbb, BBB] \vdash [q_1, bb, BB] \vdash [q_1, b, B] \vdash [q_1, \lambda, \lambda]$$

- Normal automaton:



# From CFGs to PDAs (3)

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

# From CFGs to PDAs (3)

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

## Theorem

An *extended* PDA $M$ exists for every GNF $G$ such that $L(M) = L(G)$

# From CFGs to PDAs (3)

## Definition: Greibach Normal Form (GNF)

$G = \langle V, \Sigma, P, S \rangle$ is in *Greibach normal form* if for all rules:

- $A \to a\, B_1\, B_2 \cdots B_n$ ($n \geq 0$, all $B_i \neq S$), ór
- $S \to \lambda$

## Theorem

An *extended* PDA $M$ exists for every GNF $G$ such that $L(M) = L(G)$

## Construction

$M = \langle \{q_0, q_1\}, \Sigma, V \setminus \{S\}, \delta, \{q_1\} \rangle$ with transitions

1. $q_0 \xrightarrow{\lambda \;\; \lambda/\lambda} q_1$ as $(S \to \lambda) \in P$;
2. $q_0 \xrightarrow{a \;\; \lambda/W} q_1$ for all $(S \to aW) \in P$, where $W \in V^*$;
3. $q_1 \xrightarrow{a \;\; A/W} q_1$ for all $(A \to aW) \in P$ and $A \in V \setminus \{S\}$

# Construction GNF

**Theorem**

A GNF $G'$ exists for every grammar $G$ such that $L(G) = L(G')$

**Transformation steps:**

1. Stratify: order variables, and from low to high
   - Inlining rules that start with "lower" variable
   - Removing direct left recursion
2. From high to low: inlining remaining start variables
3. Introducing special non-terminals $R_a \rightarrow a$

# Construction GNF

## Theorem

A GNF $G'$ exists for every grammar $G$ such that $L(G) = L(G')$

**Transformation steps:**

1. Stratify: order variables, and from low to high
   - Inlining rules that start with "lower" variable
   - Removing direct left recursion
2. From high to low: inlining remaining start variables
3. Introducing special non-terminals $R_a \to a$

## Intermediate steps: Stratified grammar

$G$ is *stratified* if the non-terminals are ordered, and

- $S \to \lambda$, or
- $A \to aw$ with $a \in \Sigma$ and $w \in (V \setminus \{S\} \cup \Sigma)^*$
- $A \to Bw$ with $B \in V$ and $w \in (V \setminus \{S\} \cup \Sigma)^*$, and $A < B$

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables:
   - Inline lower variables in $S$-rule:
   - Inline lower variables in $A$-rule:
   - Remove left recursion in $A$:
   - Inline lower variable $A$ in $B$-rule:
   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule:
   - Inline lower variables in $A$-rule:
   - Remove left recursion in $A$:
   - Inline lower variable $A$ in $B$-rule:
   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule:
   - Remove left recursion in $A$:
   - Inline lower variable $A$ in $B$-rule:
   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule: unnecessary
   - Remove left recursion in $A$:
   - Inline lower variable $A$ in $B$-rule:
   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \qquad A \to Ac \mid Bc \mid b \qquad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule: unnecessary
   - Remove left recursion in $A$: Helper variable $A' < A$

$$A \to BcA' \mid bA' \mid Bc \mid b$$
$$A' \to cA' \mid c$$

   - Inline lower variable $A$ in $B$-rule:
   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \qquad A \to Ac \mid Bc \mid b \qquad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule: unnecessary
   - Remove left recursion in $A$: Helper variable $A' < A$

     $$A \to BcA' \mid bA' \mid Bc \mid b$$
     $$A' \to cA' \mid c$$

   - Inline lower variable $A$ in $B$-rule:
   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule: unnecessary
   - Remove left recursion in $A$: Helper variable $A' < A$

$$A \to BcA' \mid bA' \mid Bc \mid b$$
$$A' \to cA' \mid c$$

   - Inline lower variable $A$ in $B$-rule:

$$B \to BcA'a \mid bA'a \mid Bca \mid ba$$

   - Remove left recursion in $B$:

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule: unnecessary
   - Remove left recursion in $A$: Helper variable $A' < A$

$$A \to BcA' \mid bA' \mid Bc \mid b$$
$$A' \to cA' \mid c$$

   - Inline lower variable $A$ in $B$-rule:

$$B \to BcA'a \mid bA'a \mid Bca \mid ba$$

   - Remove left recursion in $B$: Helper variable $B' < B$

$$B \to bA'aB' \mid baB' \mid bA'a \mid ba$$
$$B' \to cA'aB' \mid caB' \mid cA'a \mid ca$$

# Construction GNF: Example

Initial grammar: $S \to Aa \quad A \to Ac \mid Bc \mid b \quad B \to Aa$

1. Stratify
   - Order variables: $S < A < B$
   - Inline lower variables in $S$-rule: unnecessary
   - Inline lower variables in $A$-rule: unnecessary
   - Remove left recursion in $A$: Helper variable $A' < A$

$$A \to BcA' \mid bA' \mid Bc \mid b$$
$$A' \to cA' \mid c$$

   - Inline lower variable $A$ in $B$-rule:

$$B \to BcA'a \mid bA'a \mid Bca \mid ba$$

   - Remove left recursion in $B$: Helper variable $B' < B$

$$B \to bA'aB' \mid baB' \mid bA'a \mid ba$$
$$B' \to cA'aB' \mid caB' \mid cA'a \mid ca$$

# Construction GNF: Example, steps 2 and 3

State of affairs: Stratified grammar

$$
\begin{aligned}
S &\to Aa \\
A' &\to cA' \mid c \\
A &\to BcA' \mid bA' \mid Bc \mid b \\
B' &\to cA'aB' \mid caB' \mid cA'a \mid ca \\
B &\to bA'aB' \mid baB' \mid bA'a \mid ba
\end{aligned}
$$

2. Inline higher variables (first $B > A$, then $A > S$)

3. Use special variables

# Construction GNF: Example, steps 2 and 3

State of affairs: Stratified grammar

$$
\begin{aligned}
S &\to Aa \\
A' &\to cA' \mid c \\
A &\to BcA' \mid bA' \mid Bc \mid b \\
B' &\to cA'aB' \mid caB' \mid cA'a \mid ca \\
B &\to bA'aB' \mid baB' \mid bA'a \mid ba
\end{aligned}
$$

2. Inline higher variables (first $B > A$, then $A > S$)

$$
\begin{aligned}
A \to\ & bA'aB'cA' \mid baB'cA' \mid bA'acA' \mid bacA' \mid bA' \mid \\
& bA'aB'c \mid baB'c \mid bA'ac \mid bac \mid b \\
S \to\ & bA'aB'cA'a \mid baB'cA'a \mid bA'acA'a \mid bacA'a \mid bA'a \mid \\
& bA'aB'ca \mid baB'ca \mid bA'aca \mid baca \mid ba
\end{aligned}
$$

3. Use special variables

# Construction GNF: Example, steps 2 and 3

State of affairs: Stratified grammar

$$
\begin{aligned}
S &\to Aa \\
A' &\to cA' \mid c \\
A &\to BcA' \mid bA' \mid Bc \mid b \\
B' &\to cA'aB' \mid caB' \mid cA'a \mid ca \\
B &\to bA'aB' \mid baB' \mid bA'a \mid ba
\end{aligned}
$$

2. Inline higher variables (first $B > A$, then $A > S$)

$$
\begin{aligned}
A \to\ & bA'aB'cA' \mid baB'cA' \mid bA'acA' \mid bacA' \mid bA' \mid \\
& bA'aB'c \mid baB'c \mid bA'ac \mid bac \mid b \\
S \to\ & bA'aB'cA'a \mid baB'cA'a \mid bA'acA'a \mid bacA'a \mid bA'a \mid \\
& bA'aB'ca \mid baB'ca \mid bA'aca \mid baca \mid ba
\end{aligned}
$$

3. Use special variables

# Construction GNF: Example, steps 2 and 3

State of affairs: Stratified grammar

$$
\begin{aligned}
S &\to Aa \\
A' &\to cA' \mid c \\
A &\to BcA' \mid bA' \mid Bc \mid b \\
B' &\to cA'aB' \mid caB' \mid cA'a \mid ca \\
B &\to bA'aB' \mid baB' \mid bA'a \mid ba
\end{aligned}
$$

2. Inline higher variables (first $B > A$, then $A > S$)

$$
\begin{aligned}
A \to\ & bA'aB'cA' \mid baB'cA' \mid bA'acA' \mid bacA' \mid bA' \mid \\
& bA'aB'c \mid baB'c \mid bA'ac \mid bac \mid b \\
S \to\ & bA'aB'cA'a \mid baB'cA'a \mid bA'acA'a \mid bacA'a \mid bA'a \mid \\
& bA'aB'ca \mid baB'ca \mid bA'aca \mid baca \mid ba
\end{aligned}
$$

3. Use special variables

# Construction GNF: Example, steps 2 and 3

State of affairs: Stratified grammar

$$
\begin{aligned}
S &\to Aa \\
A' &\to cA' \mid c \\
A &\to BcA' \mid bA' \mid Bc \mid b \\
B' &\to cA'aB' \mid caB' \mid cA'a \mid ca \\
B &\to bA'aB' \mid baB' \mid bA'a \mid ba
\end{aligned}
$$

2. Inline higher variables (first $B > A$, then $A > S$)

$$
\begin{aligned}
A \to\ & bA'aB'cA' \mid baB'cA' \mid bA'acA' \mid bacA' \mid bA' \mid \\
& bA'aB'c \mid baB'c \mid bA'ac \mid bac \mid b \\
S \to\ & bA'aB'cA'a \mid baB'cA'a \mid bA'acA'a \mid bacA'a \mid bA'a \mid \\
& bA'aB'ca \mid baB'ca \mid bA'aca \mid baca \mid ba
\end{aligned}
$$

3. Use special variables $R_a \to a$, $R_c \to c$

# Construction GNF: Example, steps 2 and 3

State of affairs: Stratified grammar

$$
\begin{aligned}
S &\to Aa \\
A' &\to cA' \mid c \\
A &\to BcA' \mid bA' \mid Bc \mid b \\
B' &\to cA'aB' \mid caB' \mid cA'a \mid ca \\
B &\to bA'aB' \mid baB' \mid bA'a \mid ba
\end{aligned}
$$

2. Inline higher variables (first $B > A$, then $A > S$)

$$
\begin{aligned}
A \to\ & bA'aB'cA' \mid baB'cA' \mid bA'acA' \mid bacA' \mid bA' \mid \\
& bA'aB'c \mid baB'c \mid bA'ac \mid bac \mid b \\
S \to\ & bA'aB'cA'a \mid baB'cA'a \mid bA'acA'a \mid bacA'a \mid bA'a \mid \\
& bA'aB'ca \mid baB'ca \mid bA'aca \mid baca \mid ba
\end{aligned}
$$

3. Use special variables $R_a \to a$, $R_c \to c$

# Construction GNF: End result

We started with the following Context-free Grammer:

$$S \to Aa$$

$$A \to Ac \mid Bc \mid b$$

$$B \to Aa$$

# Construction GNF: End result

We started with the following Context-free Grammer:

$$S \to Aa$$
$$A \to Ac \mid Bc \mid b$$
$$B \to Aa$$

We finally ended up with this equivalent Greibach Normal Form:

$$\begin{aligned}
S &\to bA'R_aB'R_cA'R_a \mid bR_aB'R_cA'R_a \mid bA'R_aR_cA'R_a \mid bR_aR_cA'R_a \mid bA'R_a \mid \\
&\quad bA'R_aB'R_cR_a \mid bR_aB'R_cR_a \mid bA'R_aR_cR_a \mid bR_aR_cR_a \mid bR_a \\
A &\to bA'R_aB'R_cA' \mid bR_aB'R_cA' \mid bA'R_aR_cA' \mid bR_aR_cA' \mid bA' \mid \\
&\quad bA'R_aB'R_c \mid bR_aB'R_c \mid bA'R_aR_c \mid bR_aR_c \mid b \\
B &\to bA'R_aB' \mid bR_aB' \mid bA'R_a \mid bR_a \\
A' &\to cA' \mid c \\
B' &\to cA'R_aB' \mid cR_aB' \mid cA'R_a \mid cR_a \\
R_a &\to a \\
R_b &\to b
\end{aligned}$$

# Overview: Seen so far

Pushdown automata (PDA's): $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

- Stack alphabet $\Gamma$
- Transitions $q \xrightarrow{a \quad A/B} q'$

Variations on pushdown automata:

- Deterministic pushdown automata: *less* powerful
- Extended transitions $q \xrightarrow{a \quad A/W} q'$ with $W \in \Gamma^*$
- Acceptance by final state and/or empty stack

Connection between CFGs and PDAs

- Greibach normal form for CFGs
- Transformation CFGs $\rightarrow$ Greibach normal form
- Transformation Greibach normal form $\rightarrow$ extended PDAs
- Also possible: Transformation PDAs $\rightarrow$ CFGs

Next: Beyond Context-free languages

- Pumping lemma for CFGs
- Example: $\{a^i b^i c^i \mid i \geq 0\}$ not context-free

# Overview: Seen so far

Pushdown automata (PDA's): $\langle Q, \Sigma, \Gamma, \delta, q_0, F \rangle$

- Stack alphabet $\Gamma$
- Transitions $q \xrightarrow{a \quad A/B} q'$

Variations on pushdown automata:

- Deterministic pushdown automata: *less* powerful
- Extended transitions $q \xrightarrow{a \quad A/W} q'$ with $W \in \Gamma^*$
- Acceptance by final state and/or empty stack

Connection between CFGs and PDAs

- Greibach normal form for CFGs
- Transformation CFGs $\rightarrow$ Greibach normal form
- Transformation Greibach normal form $\rightarrow$ extended PDAs
- Also possible: Transformation PDAs $\rightarrow$ CFGs

Next: Beyond Context-free languages

- Pumping lemma for CFGs
- Example: $\{a^i b^i c^i \mid i \geq 0\}$ not context-free