CMSC 141 Automata and Language Theory

Context-Free Languages

Mark Froilan B. Tandoc

September 24, 2014

Non-Regular Languages

Non-Regular Languages

- Not all languages are regular

Non-Regular Languages

- Not all languages are regular
  - e.g. $\{a^n b^n : n > 0\}$

Non-Regular Languages

- Not all languages are regular
  - e.g. $\{a^n b^n : n > 0\}$
- There are many other non-regular languages that can be very useful

Non-Regular Languages

- Not all languages are regular
  - e.g. $\{a^n b^n : n > 0\}$
- There are many other non-regular languages that can be very useful
- We need something more powerful than finite automata that can express non-regular languages

What's Wrong With FAs?

What's Wrong With FAs?

- We can only have finite number of states where we can store information, hence, finite memory.

What's Wrong With FAs?

- We can only have finite number of states where we can store information, hence, finite memory.
- A more powerful machine needs more (theoretically infinite) memory

What's Wrong With FAs?

- We can only have finite number of states where we can store information, hence, finite memory.
- A more powerful machine needs more (theoretically infinite) memory
- One simple storage we can use is a *stack*

Stack

Stack

- A stack is a data structure with some basic operations

Stack

- A stack is a data structure with some basic operations
  - **PUSH**, store data to the top of the stack,

Stack

- A stack is a data structure with some basic operations
  - **PUSH**, store data to the top of the stack,
  - **POP**, read and remove data from the top of the stack,

Stack

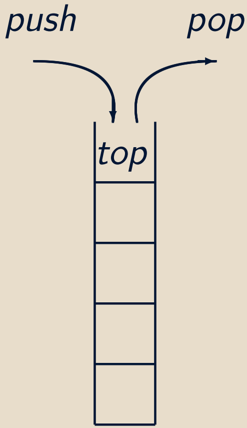
- A stack is a data structure with some basic operations
  - **PUSH**, store data to the top of the stack,
  - **POP**, read and remove data from the top of the stack,
  - PEEK/TOP, to just read data from the top of the stack, and

Stack

- A stack is a data structure with some basic operations
  - **PUSH**, store data to the top of the stack,
  - **POP**, read and remove data from the top of the stack,
  - PEEK/TOP, to just read data from the top of the stack, and
  - NOP, or no operation

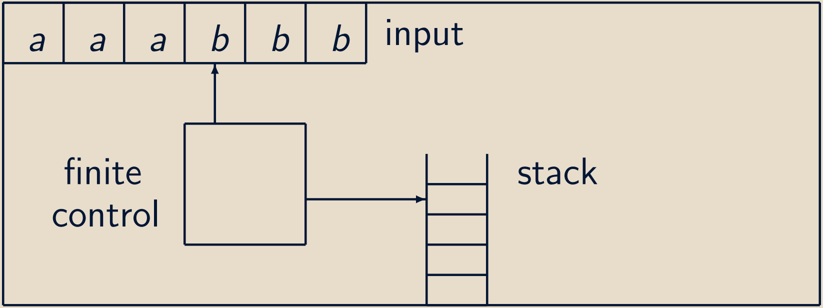
Stack

- A stack is a data structure with some basic operations
    - **PUSH**, store data to the top of the stack,
    - **POP**, read and remove data from the top of the stack,
    - PEEK/TOP, to just read data from the top of the stack, and
    - NOP, or no operation

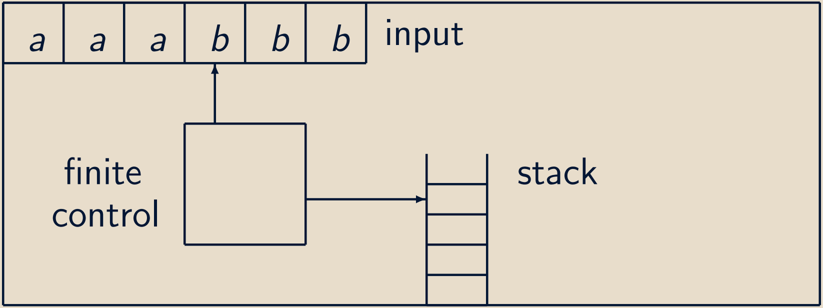


Pushdown Automata (PDA)

Pushdown Automata (PDA)

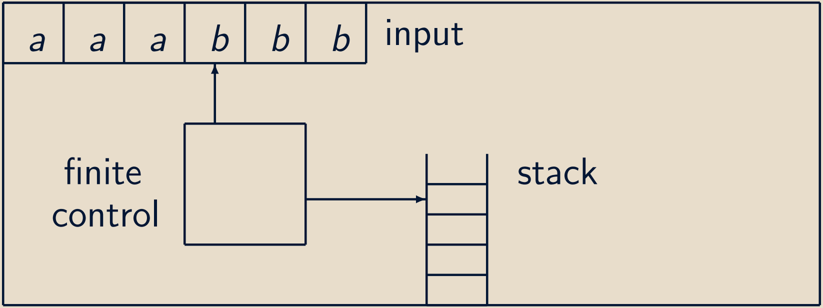


Pushdown Automata (PDA)



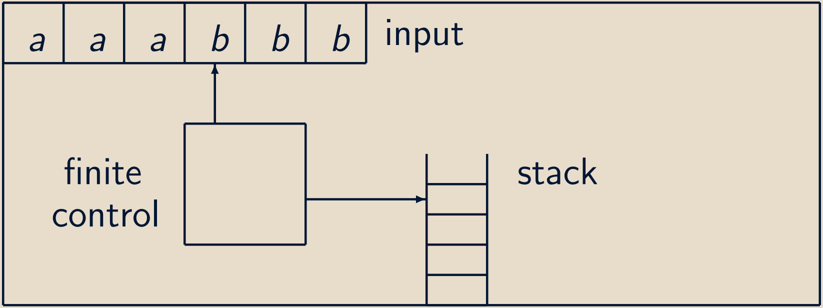
- Addition of stack for storage increases the power of the automaton

PUSHDOWN AUTOMATA (PDA)



- Addition of stack for storage increases the power of the automaton
- We can assume that the stack size is unbounded, giving us infinite memory

Pushdown Automata (PDA)



- Addition of stack for storage increases the power of the automaton
- We can assume that the stack size is unbounded, giving us infinite memory

The class of languages PDAs recognize are called Context-Free Languages (CFL)

PDA FOR $\{a^n b^n : n > 0\}$

PDA for $\{a^n b^n : n > 0\}$

- Idea is to **push** a's while we are reading them, and **pop** them one by one for every matching b.

PDA FOR $\{a^n b^n : n > 0\}$

- Idea is to **push** a's while we are reading them, and **pop** them one by one for every matching b.
- We accept the string if we ended up at the bottom of the stack after reading the whole input

PDA for $\{a^n b^n : n > 0\}$

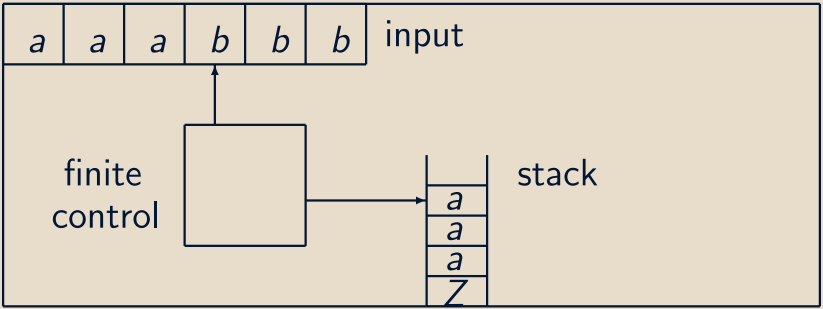
- Idea is to **push** a's while we are reading them, and **pop** them one by one for every matching b.
- We accept the string if we ended up at the bottom of the stack after reading the whole input

Bottom of the stack

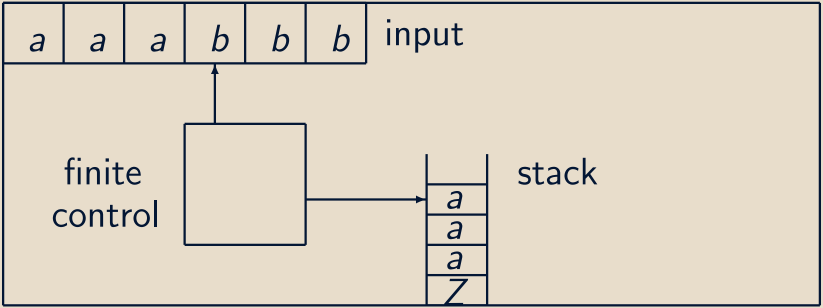
Often times, a special marker Z is placed at the bottom of the stack

PDA for $\{a^n b^n : n > 0\}$

PDA for $\{a^n b^n : n > 0\}$

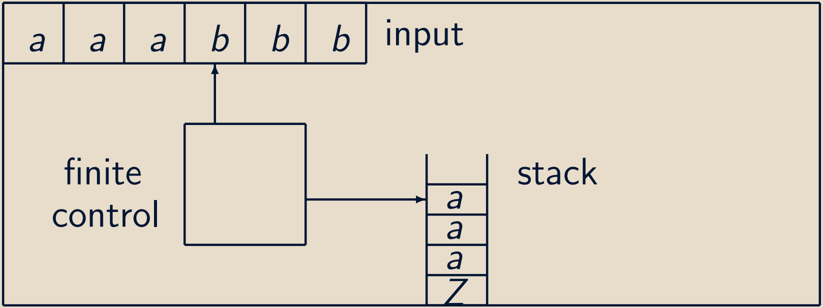


PDA FOR $\{a^n b^n : n > 0\}$



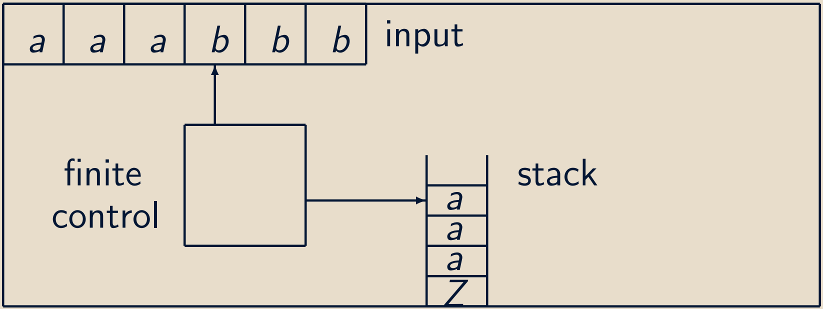
- If (a, Z) or (a, a), then push a

PDA FOR $\{a^n b^n : n > 0\}$



- If (a, Z) or (a, a), then push a
- If (b, a), then pop

PDA for $\{a^n b^n : n > 0\}$



- If (a, Z) or (a, a), then push a
- If (b, a), then pop
- If (ε, Z), then accept the string

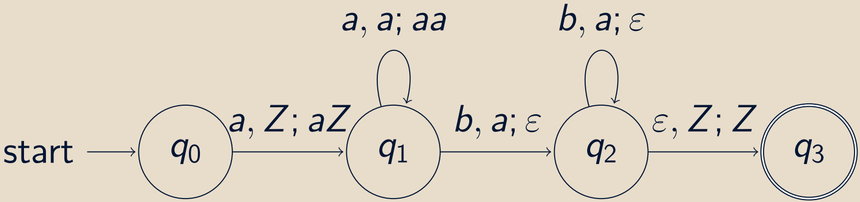
PDA for $\{a^n b^n : n > 0\}$

PDA for $\{a^n b^n : n > 0\}$

- If (a, Z) or (a, a), then push a
- If (b, a), then pop
- If (ε, Z), then accept the string

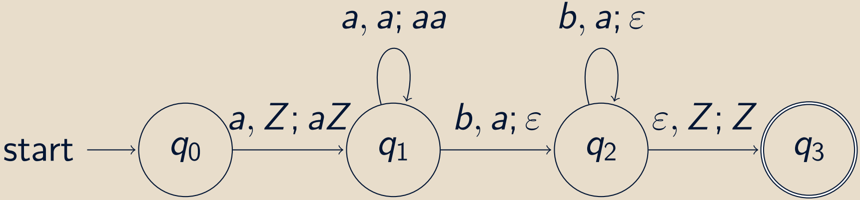
PDA for $\{a^n b^n : n > 0\}$

- If (a, Z) or (a, a), then push a
- If (b, a), then pop
- If (ε, Z), then accept the string



PDA for $\{a^n b^n : n > 0\}$

- If (a, Z) or (a, a), then push a
- If (b, a), then pop
- If (ε, Z), then accept the string

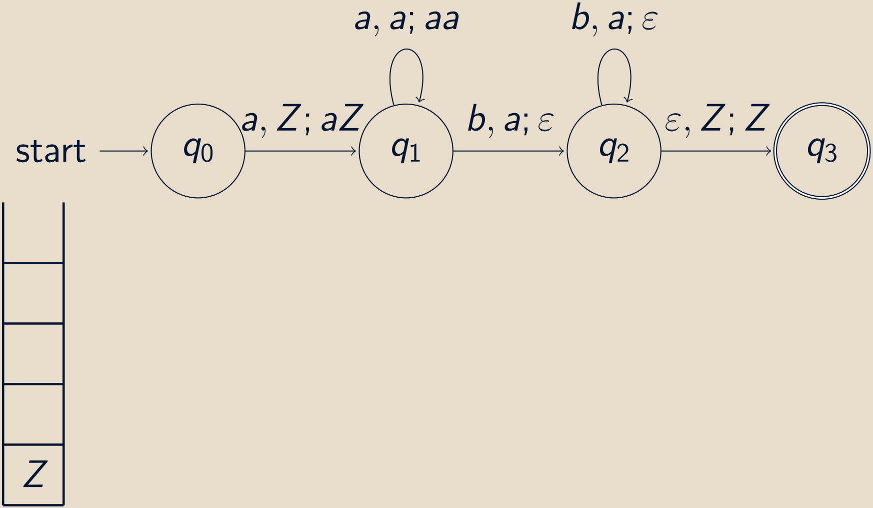


SYNTAX

(current symbol on tape,
symbol on top of the stack;
replacement symbols for the top)

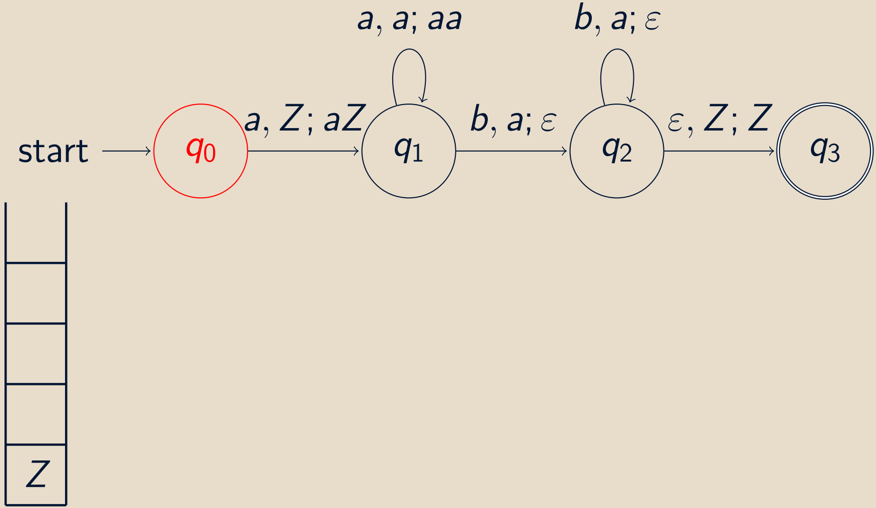
PDA for $\{a^n b^n : n > 0\}$

`aaabbb`



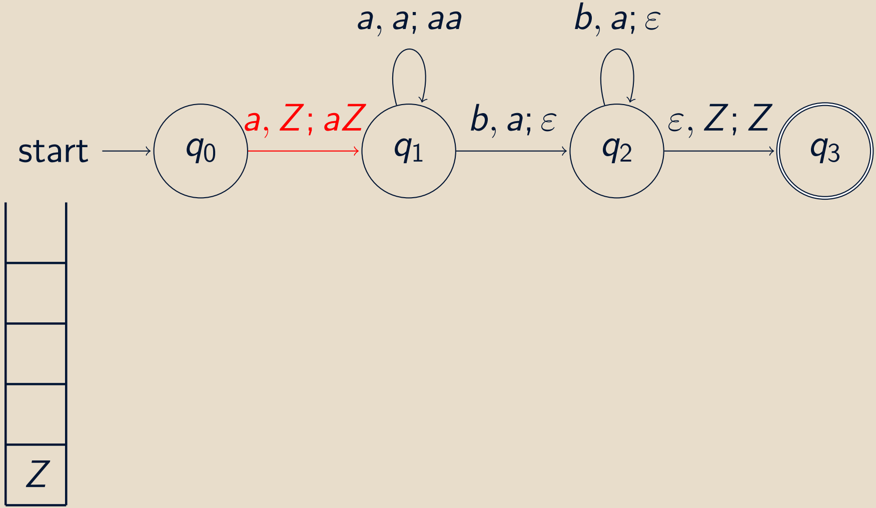
PDA for $\{a^n b^n : n > 0\}$

aaabbb



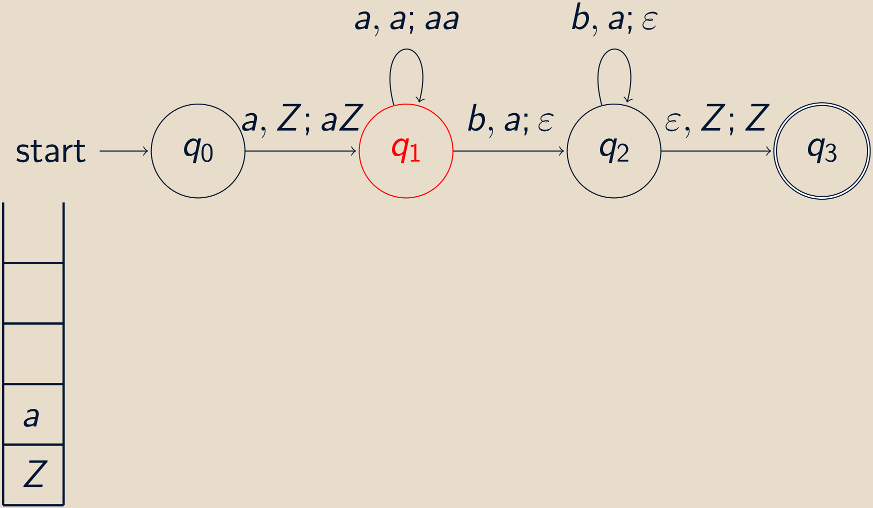
PDA for $\{a^n b^n : n > 0\}$

aaabbb



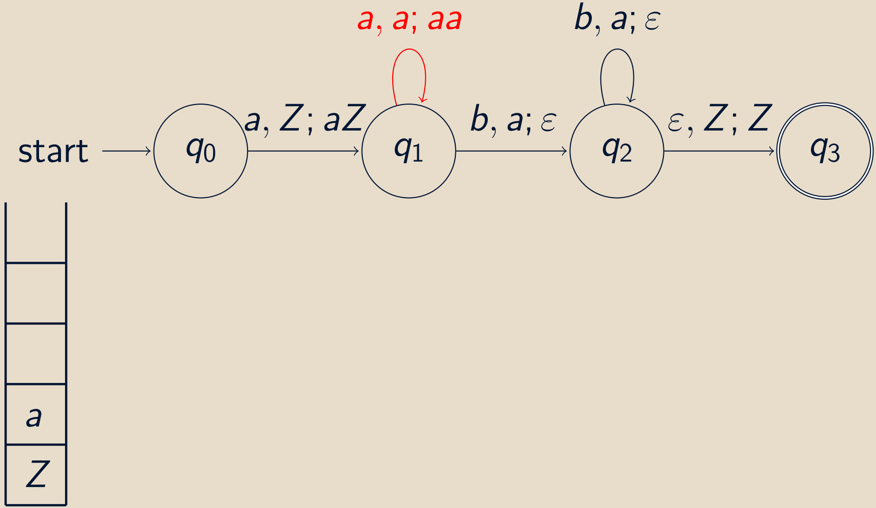
PDA for $\{a^n b^n : n > 0\}$

aaabbb



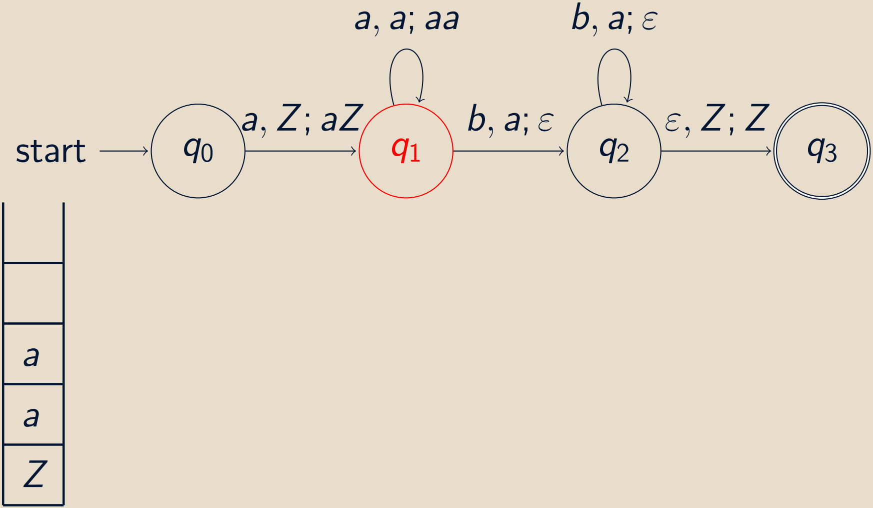
PDA for $\{a^n b^n : n > 0\}$

aaabbb



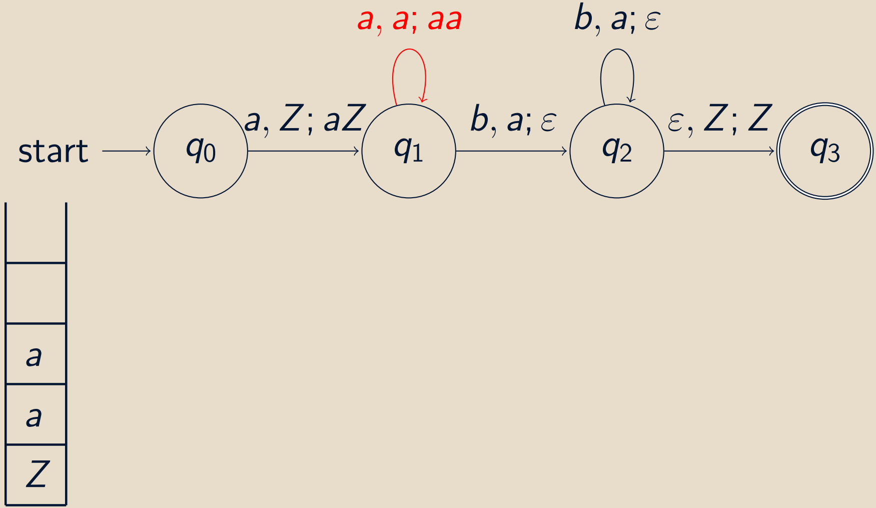
PDA for $\{a^n b^n : n > 0\}$

aaabbb



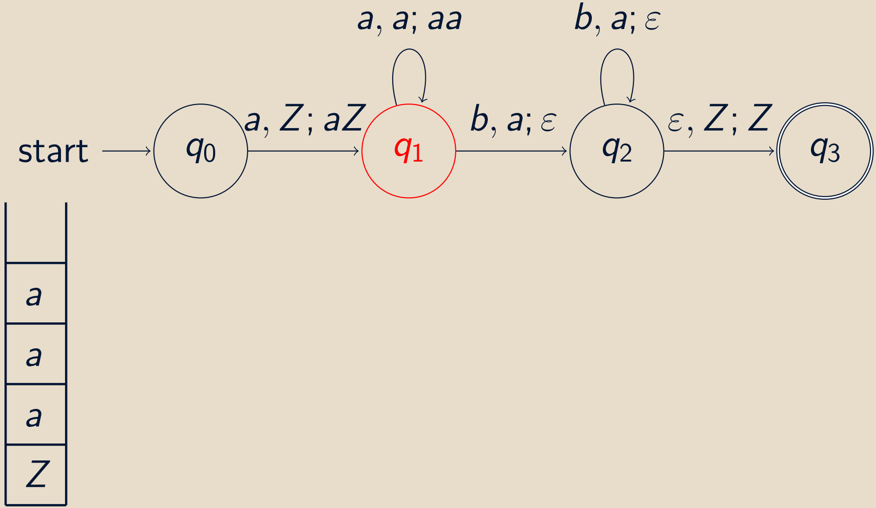
PDA for $\{a^n b^n : n > 0\}$

aaabbb



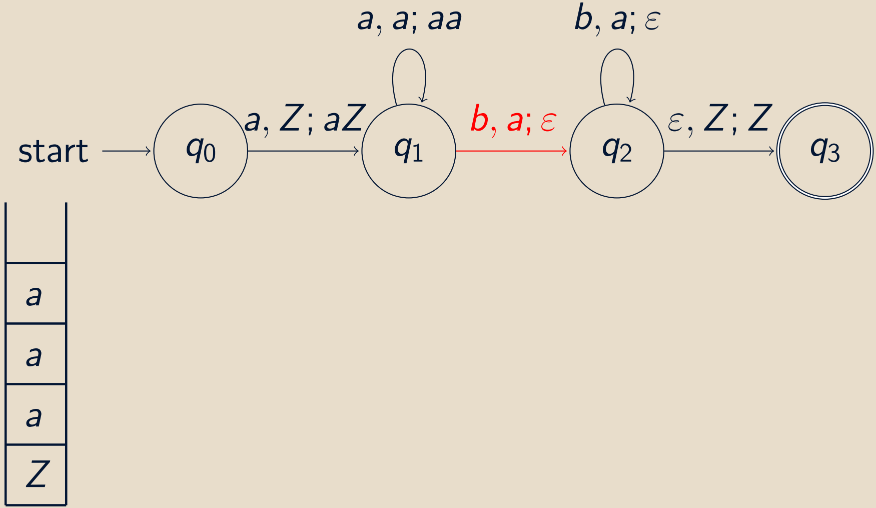
PDA for $\{a^n b^n : n > 0\}$

aaabbb



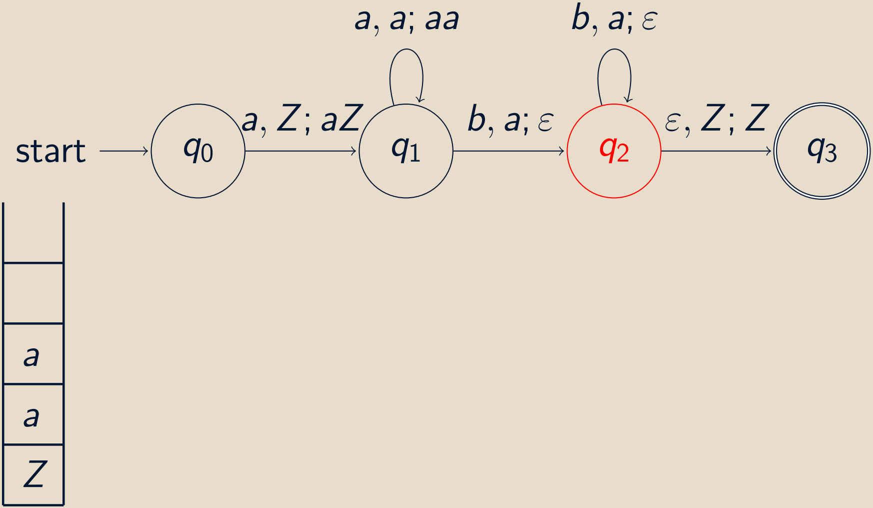
PDA for $\{a^n b^n : n > 0\}$

aaabbb



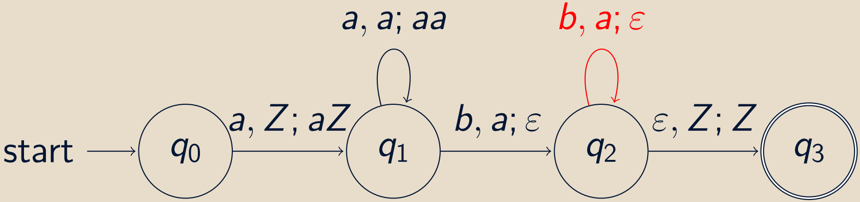
PDA for $\{a^n b^n : n > 0\}$

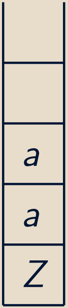
aaabbb



PDA for $\{a^n b^n : n > 0\}$

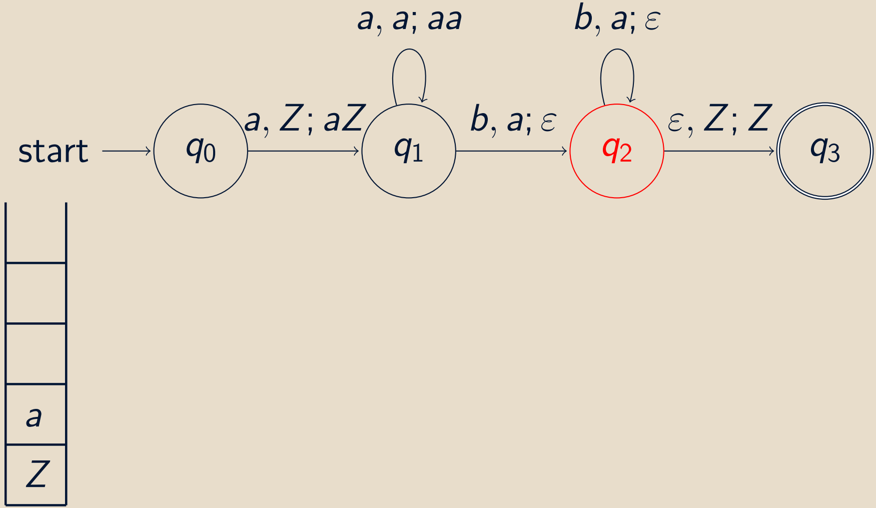
aaabbb





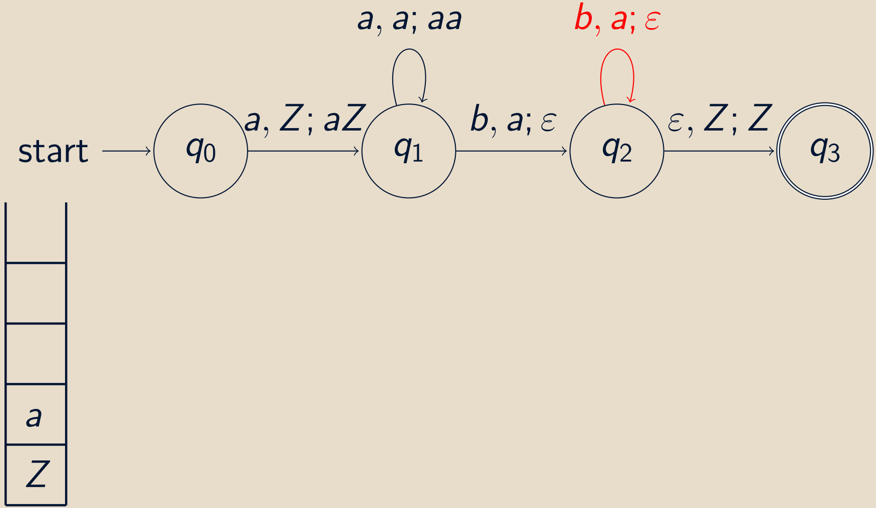
PDA FOR $\{a^n b^n : n > 0\}$

aaabb**b**



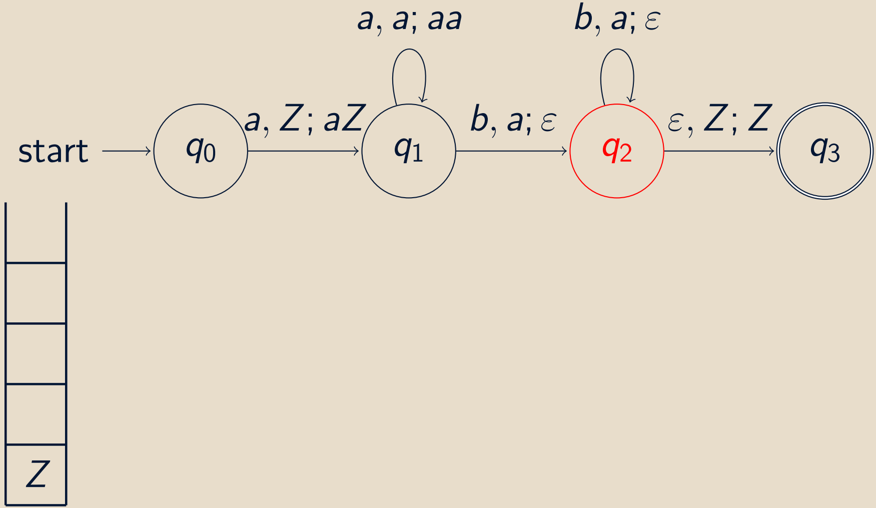
PDA for $\{a^n b^n : n > 0\}$

aaabbb



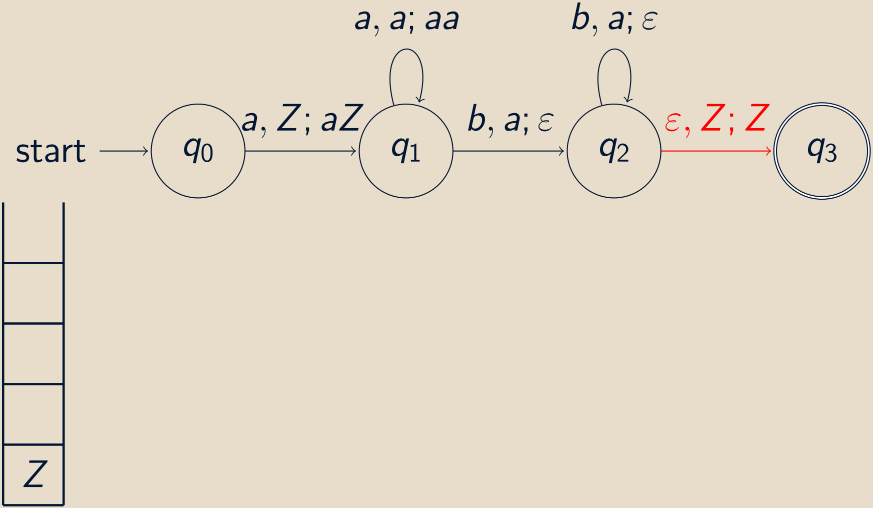
PDA for $\{a^n b^n : n > 0\}$

aaabbb



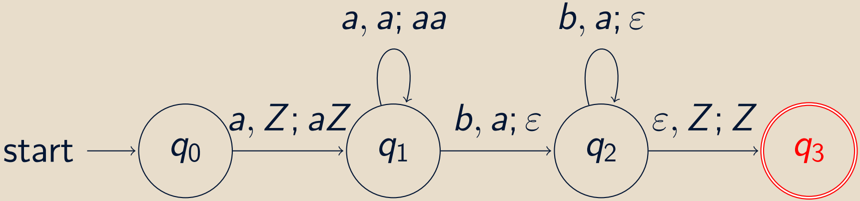
PDA for $\{a^n b^n : n > 0\}$

```
aaabbb
```



PDA for $\{a^n b^n : n > 0\}$

`aaabbb`



Formal Definition of PDA

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \rightarrow$ finite set of states

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \rightarrow$ finite set of states
- $\Sigma \rightarrow$ input alphabet

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \to$ finite set of states
- $\Sigma \to$ input alphabet
- $\Gamma \to$ stack alphabet

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \rightarrow$ finite set of states
- $\Sigma \rightarrow$ input alphabet
- $\Gamma \rightarrow$ stack alphabet
- $\delta \rightarrow$ transition function

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \rightarrow$ finite set of states
- $\Sigma \rightarrow$ input alphabet
- $\Gamma \rightarrow$ stack alphabet
- $\delta \rightarrow$ transition function
- $q_0 \rightarrow$ start/initial state

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \rightarrow$ finite set of states
- $\Sigma \rightarrow$ input alphabet
- $\Gamma \rightarrow$ stack alphabet
- $\delta \rightarrow$ transition function
- $q_0 \rightarrow$ start/initial state
- $Z \rightarrow$ initial/bottom stack symbol

Formal Definition of PDA

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$$

- $Q \rightarrow$ finite set of states
- $\Sigma \rightarrow$ input alphabet
- $\Gamma \rightarrow$ stack alphabet
- $\delta \rightarrow$ transition function
- $q_0 \rightarrow$ start/initial state
- $Z \rightarrow$ initial/bottom stack symbol
- $F \rightarrow$ set of final/accepting states

Transition Function

Transition Function

δ takes as argument a triple $\delta(q, a, X)$ where:

Transition Function

δ takes as argument a triple $\delta(q, a, X)$ where:

- q is a state in Q

Transition Function

δ takes as argument a triple $\delta(q, a, X)$ where:

- q is a state in Q
- a is either an input symbol in Σ or $a = \varepsilon$

Transition Function

δ takes as argument a triple $\delta(q, a, X)$ where:

- q is a state in Q
- a is either an input symbol in Σ or $a = \varepsilon$
- X is a stack symbol that is a member of Γ

Transition Function

δ takes as argument a triple $\delta(q, a, X)$ where:

- q is a state in Q
- a is either an input symbol in Σ or $a = \varepsilon$
- X is a stack symbol that is a member of Γ

The output is a finite set of pairs (p, γ), where p is the new state, and γ is the string of stack symbols to replace X

Instantaneous Description for PDA

Instantaneous Description for PDA

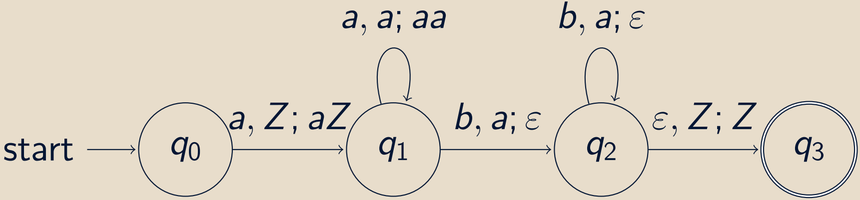
Syntax

(current state, remaining input, stack contents)

Instantaneous Description for PDA

Syntax

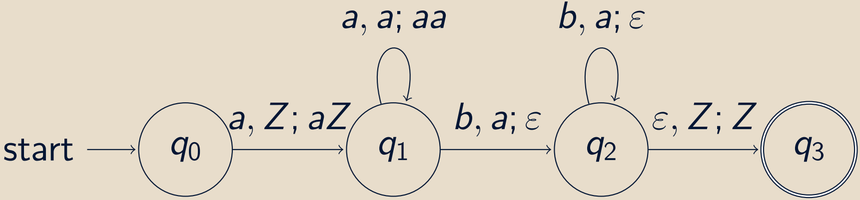
(current state, remaining input, stack contents)



Instantaneous Description for PDA

Syntax

(current state, remaining input, stack contents)



Execution of aabb

$(q_0, aabb, Z) \vdash (q_1, abb, aZ) \vdash (q_1, bb, aaZ) \vdash$
$(q_2, b, aZ) \vdash (q_2, \varepsilon, Z) \vdash (q_3, \varepsilon, Z)$

Acceptance in PDAs

Acceptance in PDAs

- A PDA accepts a string x **by final state** if (q_0, x, Z) eventually leads to $(p, \varepsilon, ?)$ for some final state p

Acceptance in PDAs

- A PDA accepts a string x **by final state** if (q_0, x, Z) eventually leads to $(p, \varepsilon, ?)$ for some final state p
- A PDA accepts a string x **by empty stack** if (q_0, x, Z) eventually leads to $(p, \varepsilon, \varepsilon)$

Acceptance in PDAs

- A PDA accepts a string x **by final state** if (q_0, x, Z) eventually leads to $(p, \varepsilon, ?)$ for some final state p
- A PDA accepts a string x **by empty stack** if (q_0, x, Z) eventually leads to $(p, \varepsilon, \varepsilon)$
- A PDA accepts a language L if every string in L is accepted and every other string is rejected

Acceptance in PDAs

- A PDA accepts a string x **by final state** if (q_0, x, Z) eventually leads to $(p, \varepsilon, ?)$ for some final state p
- A PDA accepts a string x **by empty stack** if (q_0, x, Z) eventually leads to $(p, \varepsilon, \varepsilon)$
- A PDA accepts a language L if every string in L is accepted and every other string is rejected
- The two forms of acceptance in PDAs are shown to be equivalent. That is one can be converted to the other

Examples/Exercises

Construct PDAs for the following languages:

Examples/Exercises

Construct PDAs for the following languages:

- $\{a^n b^{2n} : n > 0\} =$
  $\{\texttt{abb}, \texttt{aabbbb}, \texttt{aaabbbbbb}, \ \ldots\}$

Examples/Exercises

Construct PDAs for the following languages:

- $\{a^n b^{2n} : n > 0\} =$
  $\{\texttt{abb,aabbbb,aaabbbbbb, ...}\}$
- *palindromes* $=$
  $\{\texttt{a, b, aa, bb, aaa, bbb, aba, bab, ...}\}$

Examples/Exercises

Construct PDAs for the following languages:

- $\{a^n b^{2n} : n > 0\} =$ $\{\texttt{abb,aabbbb,aaabbbbbb, ...}\}$
- *palindromes* $=$ $\{\texttt{a, b, aa, bb, aaa, bbb, aba, bab, ...}\}$
- equal number of a's and b's (in any order) $=$ $\{\texttt{ab,ba,aabb,abab,baba,bbaa,...}\}$

Examples/Exercises

Construct PDAs for the following languages:

- $\{a^n b^{2n} : n > 0\} =$
  $\{\texttt{abb,aabbbb,aaabbbbbb, ...}\}$
- *palindromes* $=$
  $\{\texttt{a, b, aa, bb, aaa, bbb, aba, bab, ...}\}$
- equal number of a's and b's (in any order) $=$
  $\{\texttt{ab,ba,aabb,abab,baba,bbaa,...}\}$
- balance pair of parentheses $=$
  $\{\texttt{(), (()), ()(), ((())), (())(), ...}\}$

References

- Previous slides on CMSC 141
- M. Sipser. Introduction to the Theory of Computation. Thomson, 2007.
- J.E. Hopcroft, R. Motwani and J.D. Ullman. Introduction to Automata Theory, Languages and Computation. 2nd ed, Addison-Wesley, 2001.
- E.A. Albacea. Automata, Formal Languages and Computations, UPLB Foundation, Inc. 2005
- JFLAP, www.jflap.org
- Various online LaTeX and Beamer tutorials