Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 13

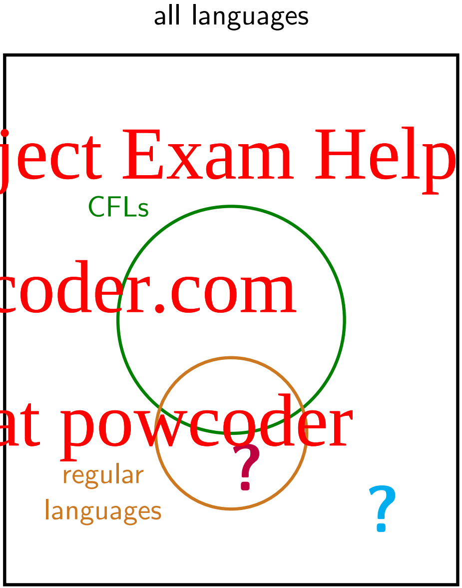
(A) Regular Grammars,
(B) Pushdown Automata

slides by Graham Farr
based in part on previous slides by David Albrecht

Overview

- { Regular Languages } $\subseteq$ { CFLs }
- Pushdown Automaton (PDA)
- Constructing PDA to accept a Regular Grammar

Is every Regular Language
a Context-Free Language?



NFA to CFG

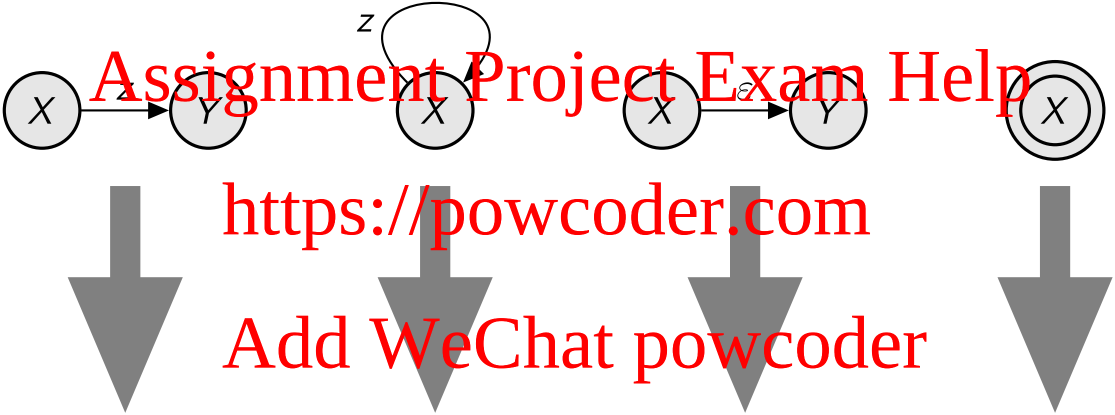
Input: an NFA

1. Name all the states in the NFA by a symbol.
   - Call the Start State S.
2. For each ordered pair of states X, Y linked by an arc labelled z, create production rule $X \longrightarrow zY$.
3. For each Final State X, create production rule $X \longrightarrow \varepsilon$.

Output: the CFG consisting of

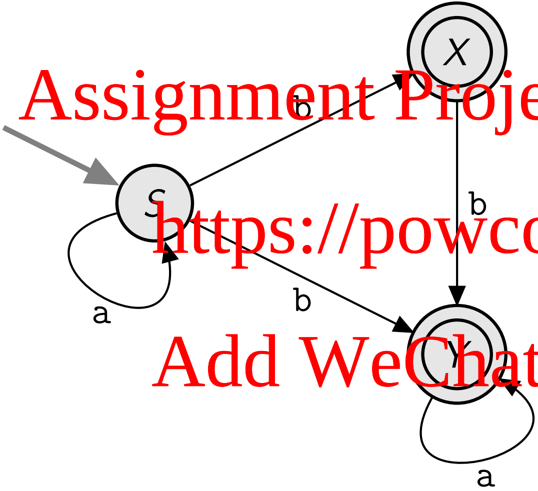
- terminals: the alphabet of the NFA;
- non-terminals: the symbols representing the states of the NFA;
- all the production rules we have created.

NFA to CFG



$X \longrightarrow zY$ $\quad$ $X \longrightarrow z\,X$ $\quad$ $X \longrightarrow Y$ $\quad$ $X \longrightarrow \varepsilon$

NFA to CFG: Example



$$
\begin{aligned}
S &\longrightarrow \mathtt{a}S \\
S &\longrightarrow \mathtt{b}X \\
S &\longrightarrow \mathtt{b}Y \\
X &\longrightarrow \mathtt{b}Y \\
Y &\longrightarrow \mathtt{a}Y \\
X &\longrightarrow \varepsilon \\
Y &\longrightarrow \varepsilon
\end{aligned}
$$

Regular Grammar

Definitions

Semiwords are of the form:

terminal terminal . . . terminal Nonterminal

A CFG is called a **Regular Grammar** if all its production rules are in one of the following forms:

Nonterminal $\longrightarrow$ semiword

or

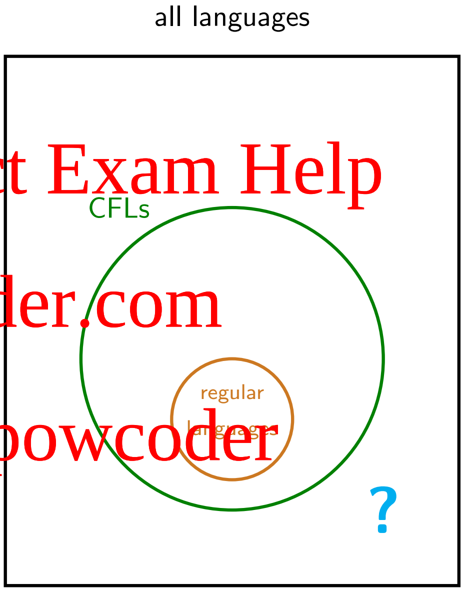
Nonterminal $\longrightarrow$ string of terminals

Theorem.
Every Regular Language can be generated by a Regular Grammar.

Proof Idea:
A regular language is recognised by some NFA
Observe that our construction NFA $\longrightarrow$ CFG produces a regular grammar. $\square$

Theorem.
Every Regular Grammar generates a Regular Language.

Proof: Exercise.



Overview

Is there a state machine for Context Free Languages?

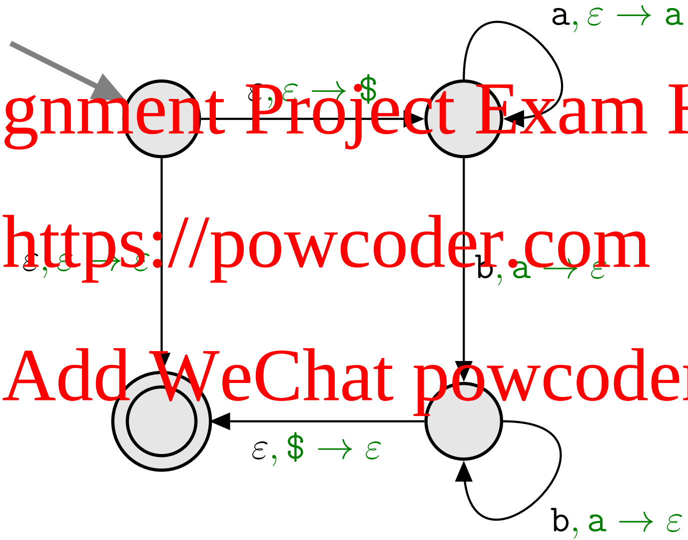
Pushdown Automaton (PDA)

- A Nondeterministic Finite Automaton (NFA) with a Stack.

- Can be used to represent Context Free Languages.

- Many parsers use a Pushdown Automaton
  - ...including the parsers generated by some compiler-compilers.

Pushdown Automaton (PDA)



Pushdown Automaton (PDA)

The Stack

- storage for letters
  - serves as a memory

- two operations:
  - **Push**: puts a letter on the top of the stack
  - **Pop**: takes a letter off the top of the stack.

Transitions

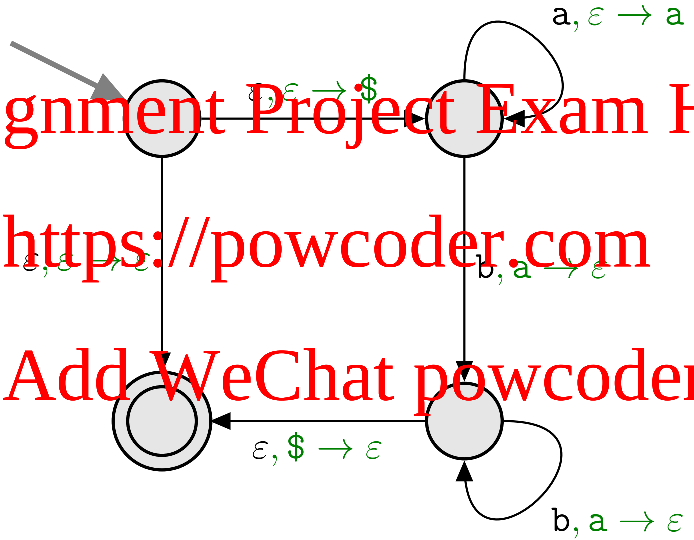
$$x, y \longrightarrow z$$

which means when the machine is reading x, if there is a y on top of the stack, it is replaced by z.

- If x is ε then no letter is read from the tape.
- If y is ε then no letter is popped from the stack.

- If z is ε then no letter is pushed onto the stack.

Pushdown Automaton (PDA)



Pushdown Automaton (PDA)

A **Pushdown Automaton** consists of:

- an **input alphabet**: the set of possible input letters.
- a **stack alphabet**: the set of possible stack letters.
- a stack
- a finite set of states
  - One called the **Start State**
  - Some (maybe none) called **Final States**
- A set of transitions between states

$$x, y \longrightarrow z$$

which means when the machine is reading x, if there is a y on top of the stack, it is replaced by z.

Pushdown Automaton (PDA)

Definitions

A string is **accepted** by a PDA if there exists at least one path through the PDA for this string that ends in a Final State.

A string is **rejected** by a PDA if for all paths through the PDA for this string, the PDA either crashes or ends in a non-Final State.

The set of strings accepted by the PDA is called the **language accepted** by the PDA.

HALF-AND-HALF

$$\text{HALF-AND-HALF} := \{a^n b^n : n \geq 0\}$$
$$= \{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

Using the Pumping Lemma we showed that this language was non-regular.

Consider:

$$S \longrightarrow aSb \mid \varepsilon$$

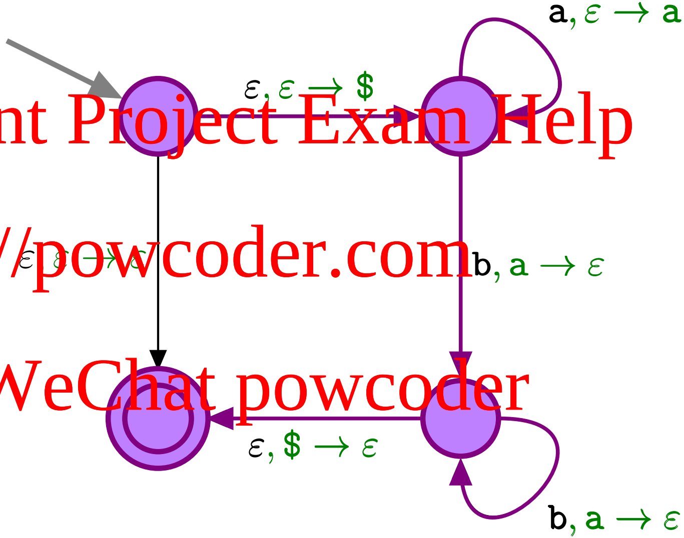
So it is a Context-Free Language.

PDA for HALF-AND-HALF

Input: aaabbb

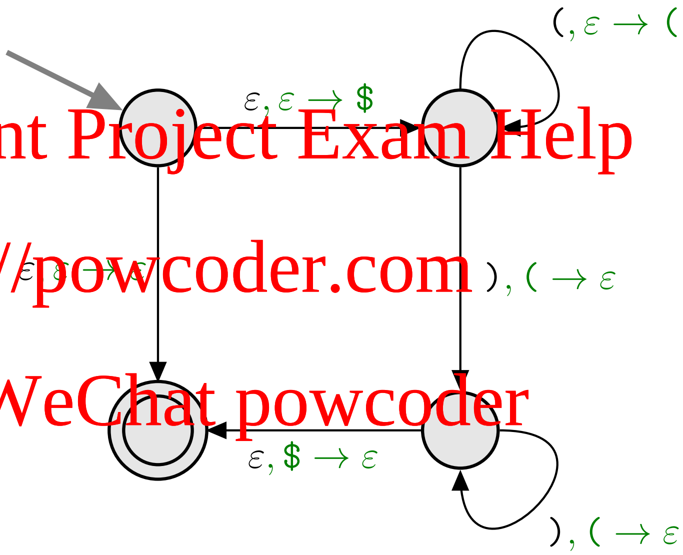
Stack:

a
a
a
$

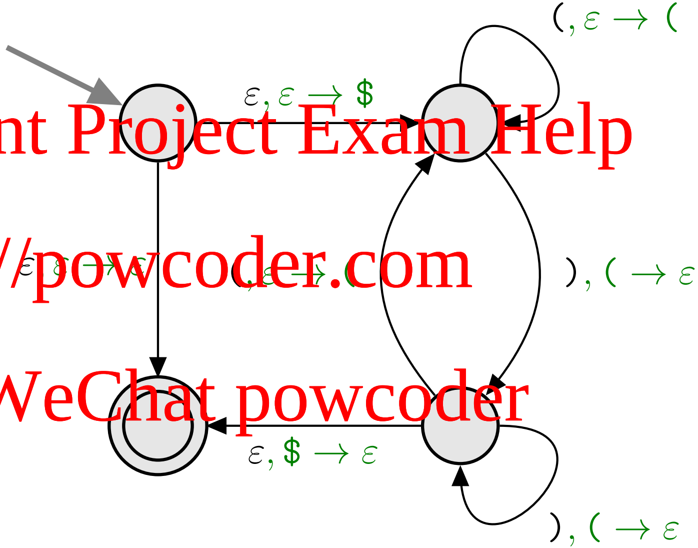


PDA for PARENTHESES

1.	S	$\rightarrow$	ε
2.	S	$\rightarrow$	(S)
3.			



PDA for PARENTHESES



1.	S	$\to$	ε
2.	S	$\to$	(S)
3.	S	$\to$	SS

Regular Languages $\subseteq$ languages accepted by PDAs

Exercise:

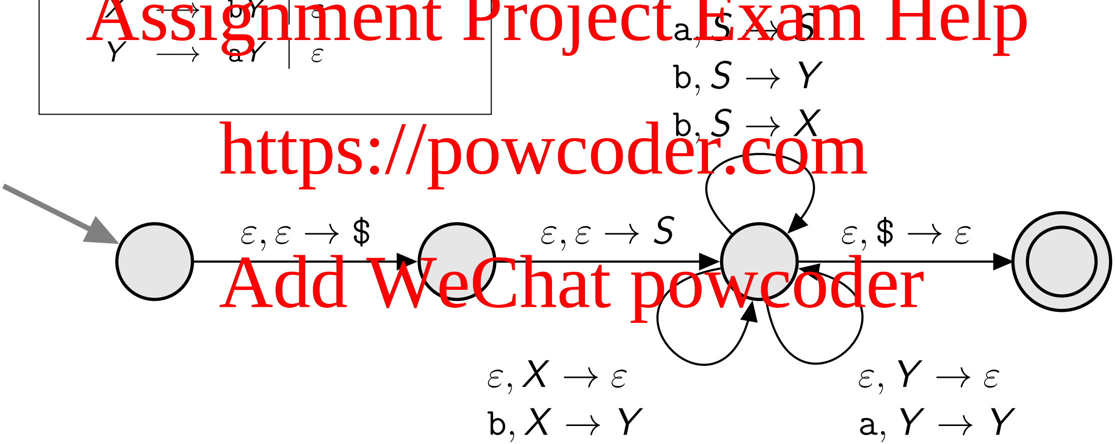
- What do you have to do to restrict a PDA so that it is just an NFA?

An NFA is a special case of a PDA.
So

$$\{\text{regular languages}\} \subseteq \{\text{languages recognised by a PDA}\}$$

Regular Languages $\subseteq$ languages accepted by PDAs

$$
\begin{aligned}
S &\longrightarrow \mathtt{a}S \mid \mathtt{b}Y \mid \mathtt{b}X \\
X &\longrightarrow \mathtt{b}Y \mid \varepsilon \\
Y &\longrightarrow \mathtt{a}Y \mid \varepsilon
\end{aligned}
$$



Revision

Things to think about:

- Which of the PDAs we've met are *deterministic*?
- Suppose we plot stack height against time for our PDA for the Dyck language. What would it look like?
- How can we construct, from a given CFG, a PDA for the same language?
- How can we construct, from a given PDA, a CFG for the same language?

Reading: Sipser, Section 2.2, pp. 111–116.