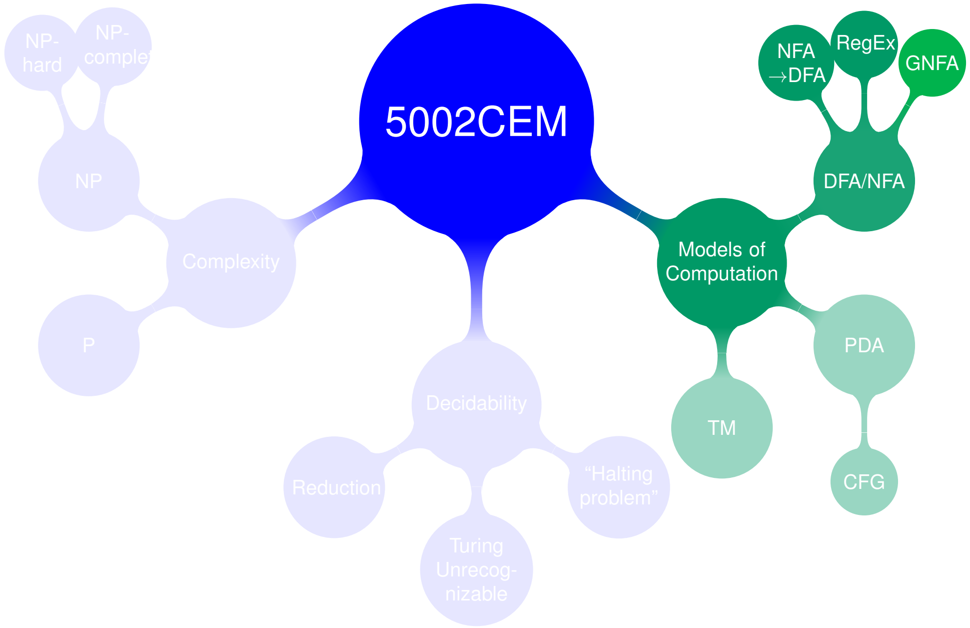
Context-Free Languages (CFLs)

Dr Kamal Bentahar

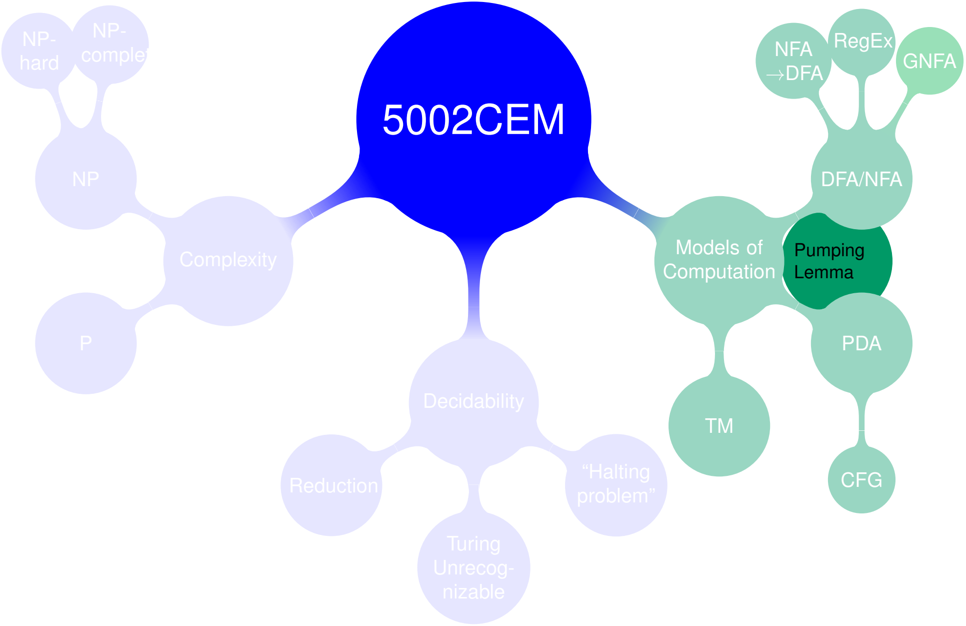
School of Computing, Electronics and Mathematics
Coventry University

Lecture 5

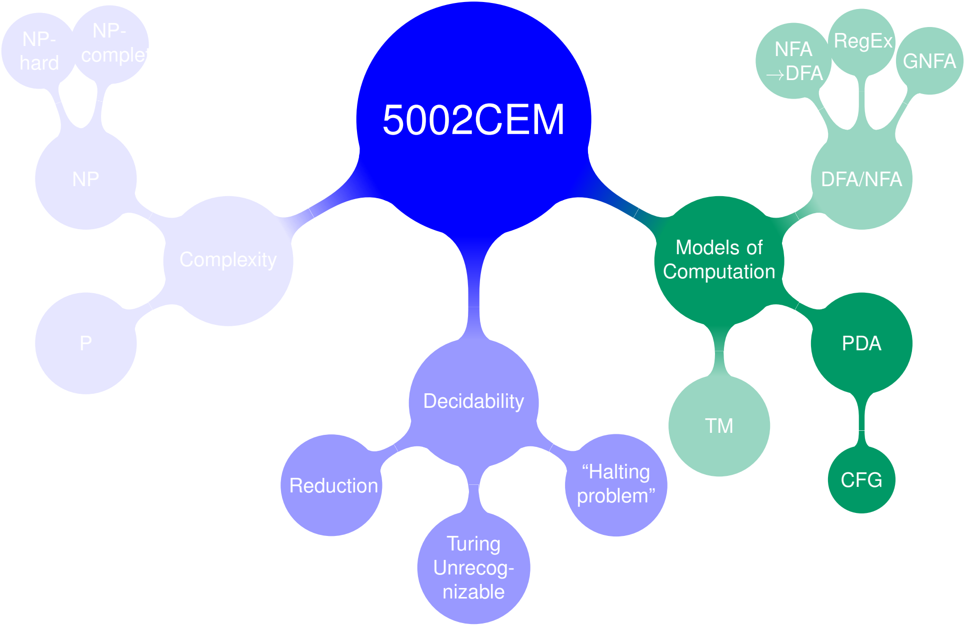
Mindmap
Language classes
NFA & Stack
PDAs
Example
Nondeterminism
Grammars
Design of CFGs



Mindmap
Language classes
NFA & Stack
PDAs
Example
Nondeterminism
Grammars
Design of CFGs

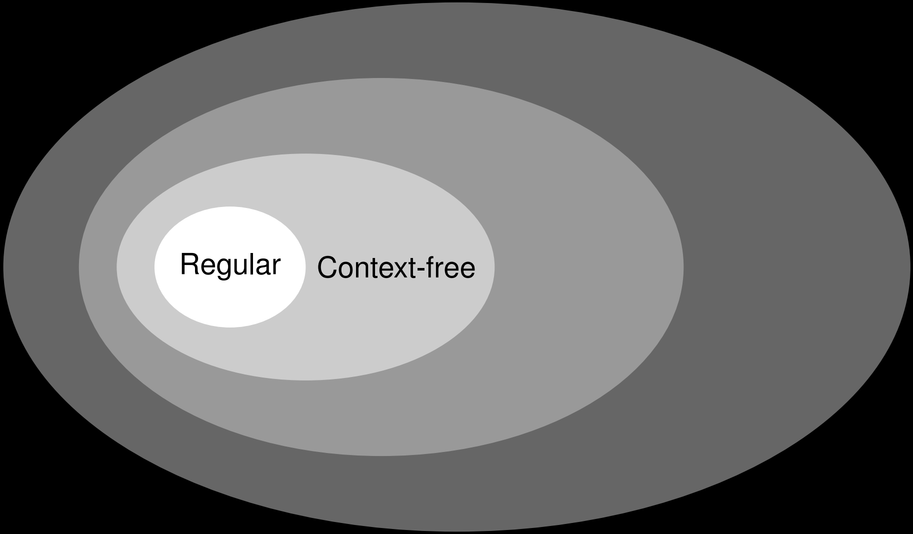


Mindmap
Language classes
NFA & Stack
PDAs
Example
Nondeterminism
Grammars
Design of CFGs



Mindmap
Language classes
NFA & Stack
PDAs
Example
Nondeterminism
Grammars
Design of CFGs

Language types



Important concepts developed so far...

- Finite State Machines (FSMs: DFA/NFA).
  states, alphabet, transitions, initial and accept states.
- Nondeterminism.
- Equivalence of models.
- Accepters vs generators.
  FSM vs Grammar/RegEx.

Making NFAs more powerful. . .

We have seen that $\{a^n b^n \mid n \geq 0\}$ is **not regular.** (Pumping Lemma)
What can we add to NFAs to enable them to recognize this language?

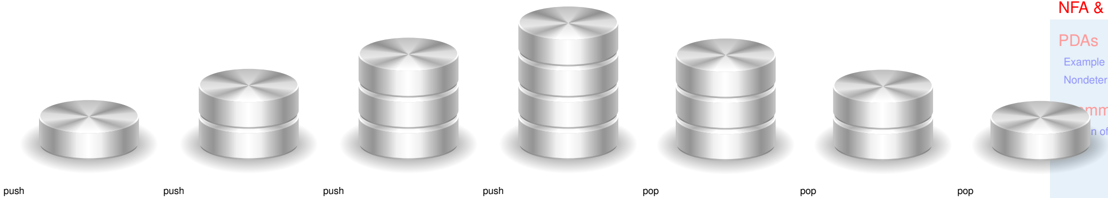
Idea!

- **Count** how many symbols have been seen.
- We can use a **stack**!

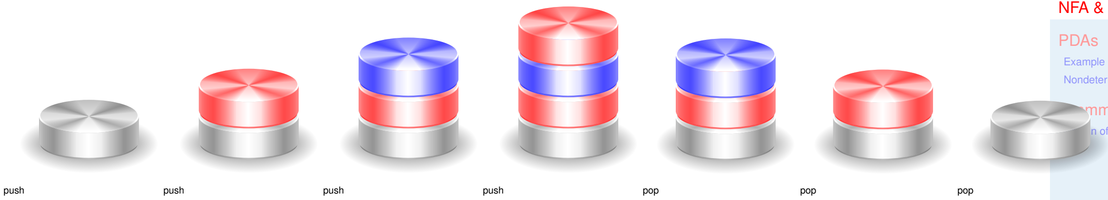
What is a "stack"?

- LIFO memory (Last-In First Out).
- Can **push** & **pop**.
- Infinite (structured) memory!

Counting using a stack



Remembering patterns using a stack

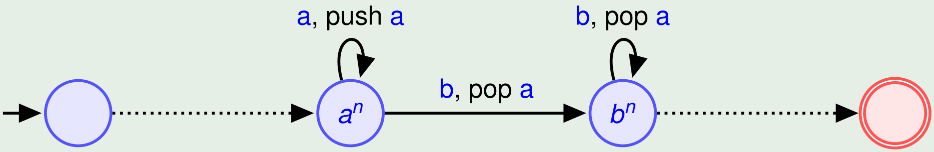


Push-Down Automata (PDAs)

- Largely the same as NFAs but with the addition of a stack memory
- More powerful than NFAs: can recognize some non-regular languages.
- On transition, the machine does not just change state: it also pushes and/or pops an item on/off the stack.

Example ($\{a^n b^n \mid n \geq 1\} = \{ab, aabb, aaabbb, \ldots\}$)

Push all the a's onto the stack, and then pop one off each time a b is read. If the stack is empty at the end then the machine accepts.



Notation and technicalities. . .

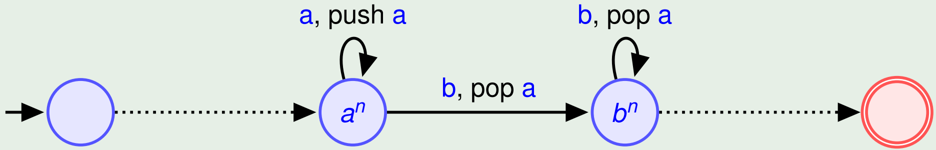
- We label transitions with: $a, b \to c$
  - a: input symbol read from the input string
  - b: symbol popped off the stack
  - c: symbol which replaces it
  - Either a, b or c may be ε

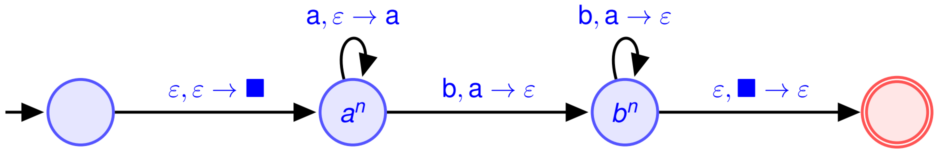
a must be read from the string **and** *b* must be present on the stack.

- **Empty stack:** No special feature for checking if the stack is empty.
  $\to$ Push a delimiting character (e.g. ■ or $) onto the stack at the beginning, then test for this character to see if it is empty.
- **End of input:** There is no specific way to test for the end of the input.
  $\to$ have no transitions out of the accept state (i.e. only the last character can reach it).

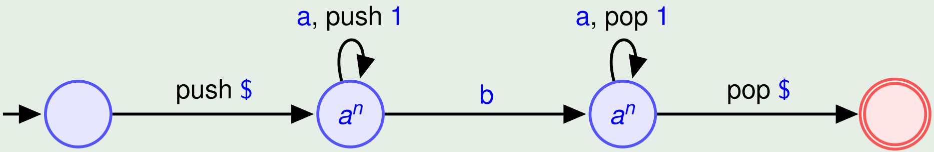
Mindmap
Language classes
NFA & Stack
PDAs
Example
Nondeterminism
Grammars
Design of CFGs

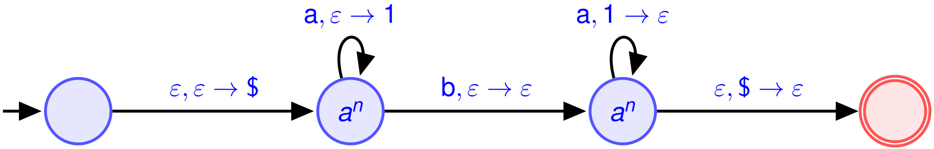
Example ($\{a^n b^n \mid n \geq 1\} = \{ab, aabb, aaabbb, \ldots\}$)





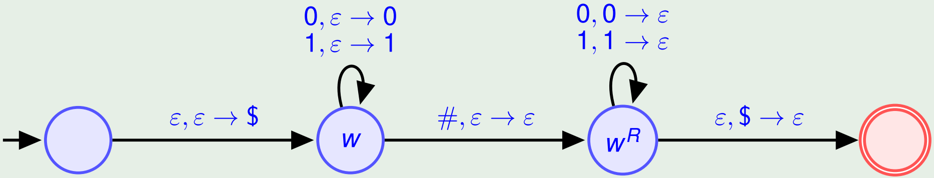
Example ($\{a^n b a^n \mid n \geq 0\}$)





Recall: w^R is w in reverse (backwards), e.g. if $w = 1011$ then $w^R = 1101$.

Example ($\{w\#w^R \mid w \in \{0,1\}^*\}$)



Mindmap
Language classes
NFA & Stack
PDAs
Example
Nondeterminism
Grammars
Design of CFGs

Closures & Nondeterminism

Like NFAs:

- Closed under union, concatenation, and star operations.
- Non-determinism behaviour, and each branch of the computation gets its **own stack**!

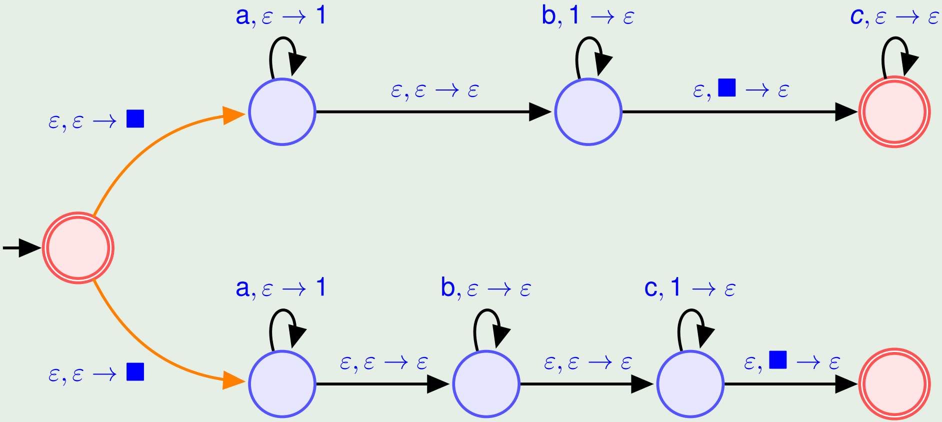
Different!

Unlike DFAs vs NFAs, deterministic PDAs are **less powerful** than non-deterministic PDAs. (i.e., they recognize less languages)
Some CFLs can only be recognized by PDAs using non-determinism.

Example ($\{a^i b^j c^k \mid i = j \text{ or } i = k\}$)

Rewrite as:

$$\{a^i b^j c^k \mid i = j\} \quad \cup \quad \{a^i b^j c^k \mid i = k\}$$



Formal definition of PDAs & CFLs

Push-Down Automata (PDAs)

A PDA is a 6-tuple $\{Q, \Sigma, \Gamma, \delta, q_{\text{start}}, F\}$ where

- Q is the set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow 2^{Q \times \Gamma_\varepsilon}$ is the transition function
- q_{start} is the start state
- F is the set of accept states

Context-Free Languages (CFLs)

A language is Context-Free **iff** it is recognized by a non-determinsitc PDA.

“Language recognition” and “Language generation”

	Regular Languages	Context-Free Languages
Recognizer:	NFA/DFA	PDA
Generator:	RegEx / **Regular** Grammar	**Context-Free** Grammar

Context-Free Grammars (CFGs):

- more powerful at describing languages than RegEx’s.
  Can be used to describe all RLs, as well as some non regular ones
- first used in the study of natural languages.

CFGs: Context-Free Grammars

Context-Free Grammars (CFGs) are defined by **production rules** such as

$$\begin{aligned} A &\rightarrow \mathrm{a}A\mathrm{b} \\ A &\rightarrow B \\ B &\rightarrow \varepsilon \end{aligned}$$

- **Terminals**: Lower case symbols.
- **Variables/Non-terminals**: Upper case symbols.
- **Start variable**.
- Only one variable to the left of the arrow.
  $\rightarrow$ "context free."

Common rule sets

- $S \to aS$ generates $\{a, aa, aaa, \ldots\}$
- $S \to aSb$ generates $\{ab, aabb, aaabbb, \ldots\}$
- $S \to AB$: concatenation.
- $S \to SS$ produces $SS, SSS, SSSS, \ldots$: star.

Grammars

Chomsky Hierarchy

Grammar	Languages	Automaton	Production rules
Type-0	Recursively Enumerable	Turing Machine (TM)	$\alpha \to \beta$ (no restrictions)
Type-1	Context Sensitive	Linear-bounded TM	$\alpha A \beta \to \alpha\gamma\beta$
Type-2	Context Free	PDA	$A \to \gamma$
Type-3	Regular	NFA	$A \to aB \mid a$

Context-Free Grammars (CFGs)

A Context Free Grammar (CFG) is a 4-tuple $\{V, \Sigma, R, S\}$ where

- V is the set of variables
- Σ is the set of terminals
- R is the set of production rules
- S is the start variable

Design of CFGs: look for **recursive structures**

Example

Design a CFG to represent the language L over $\Sigma = \{a, b\}$, given by

$$L = \{w \mid w = a^n b a^n, \quad n \geq 0\}$$

We note that

$$a^n b a^n = \begin{cases} a(a^{n-1} b a^{n-1})a & \text{for } n \geq 1 \qquad \text{(Recursive case)} \\ b & \text{for } n = 0 \qquad \text{(Base case)} \end{cases}$$

CFG:

$$\begin{aligned} A &\rightarrow aAa \\ A &\rightarrow b \end{aligned}$$

Equivalence of PDAs and CFGs → Context-Free Languages (CFLs)

- Class of languages recognized by PDAs is the same as the one generated by CFGs.
  Can be shown by providing methods to convert one to the other – refer to the textbook for a demonstration.

- We call this class: **Context-Free Languages** (CFLs).

Pumping Lemma for CFLs (Not examinable)

For the curious – not examinable!

Pumping Lemma for CFLs

If L is a CFL then there is a number p where: if w is any string in L of length at least p then w may be divided into **five** pieces $w = uxyzv$ satisfying the conditions

1. for each $k \geq 0$: $ux^kyz^kv \in L$
2. $|xz| > 0$
3. $|xyz| \leq p$