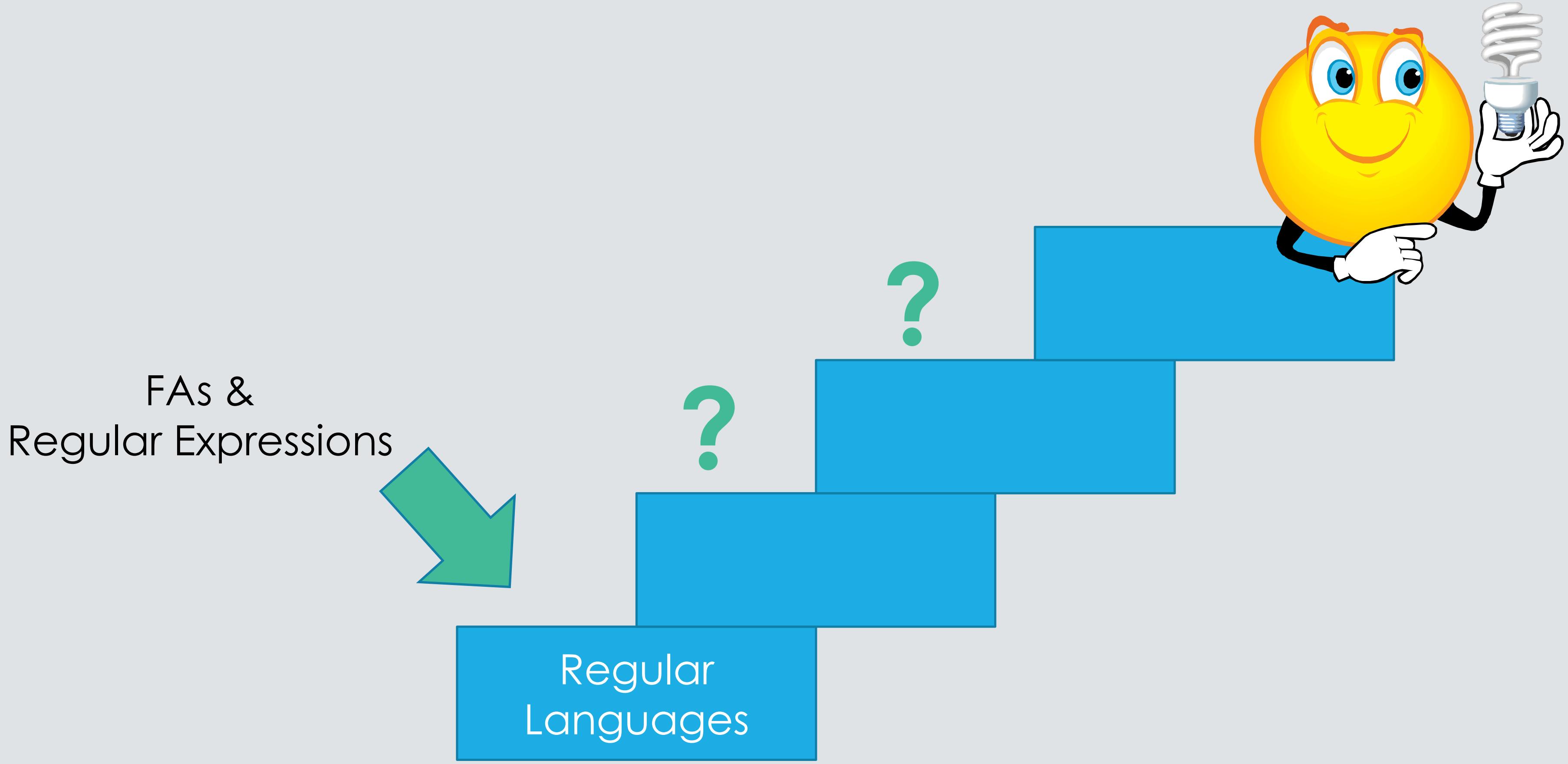
CHAPTER 2

Context-Free Languages

Languages

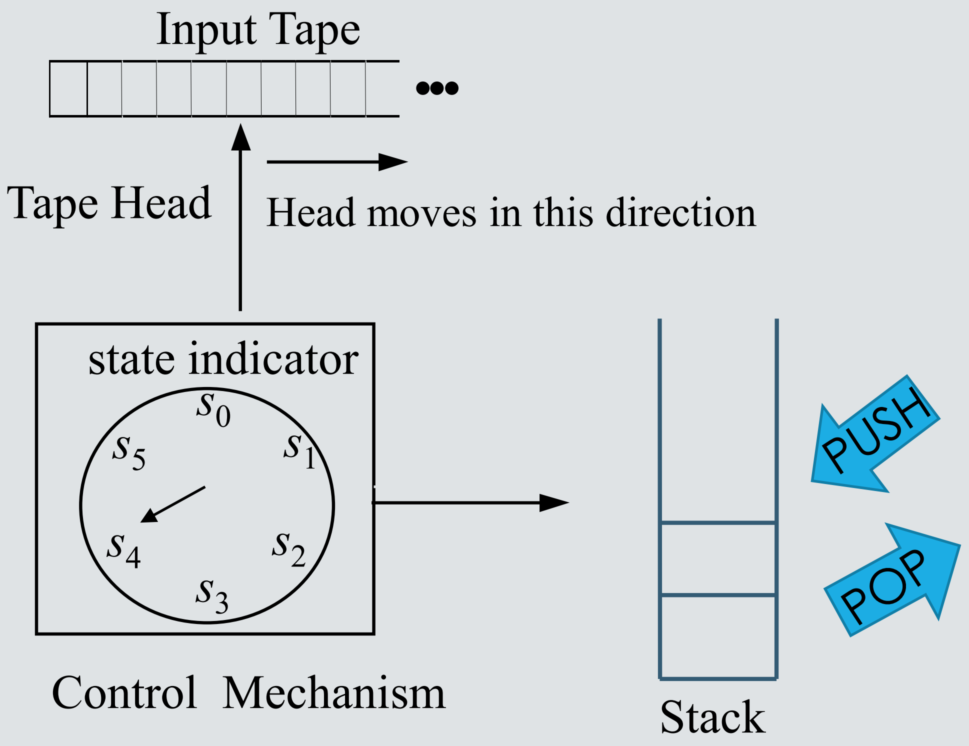


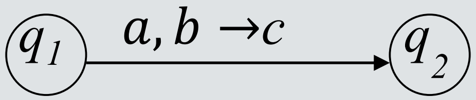
Pushdown Automata

- PDA= FA + stack
  - Is an enhanced FAs with an internal memory system, i.e., a (pushdown) stack
  - Overcomes the memory limitations and increases the processing power of FAs
  - PDA is equivalent in power to CFG
    - Either can be used to recognize or generate a languages (whichever more easily describe the language)
- Formal definition
  - A pushdown automaton (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where
    - Q is a finite set of states
    - Σ is a finite set of input symbols, called input alphabet
    - Γ is a finite set of stack symbols, called stack alphabet
    - $q_0 \in Q$, is the start state
    - $F \subseteq Q$, is the set of final states
    - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow Q \times (\Gamma \cup \{\epsilon\})$, a (partial) transition function

PDA

- Diagram

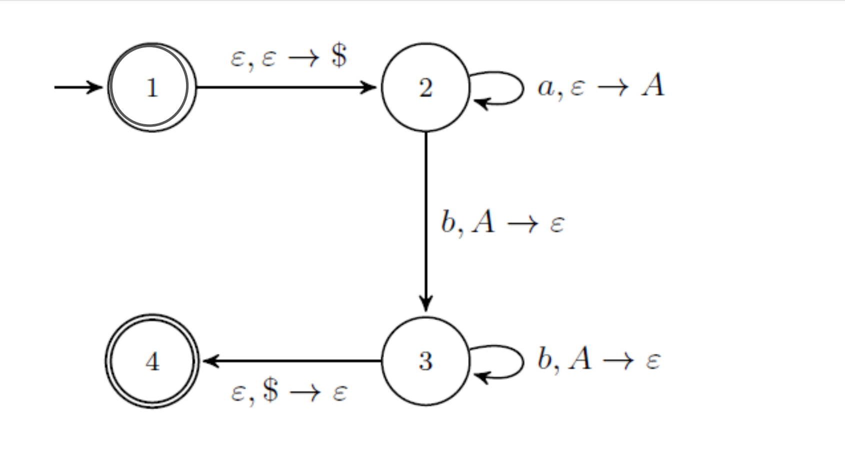




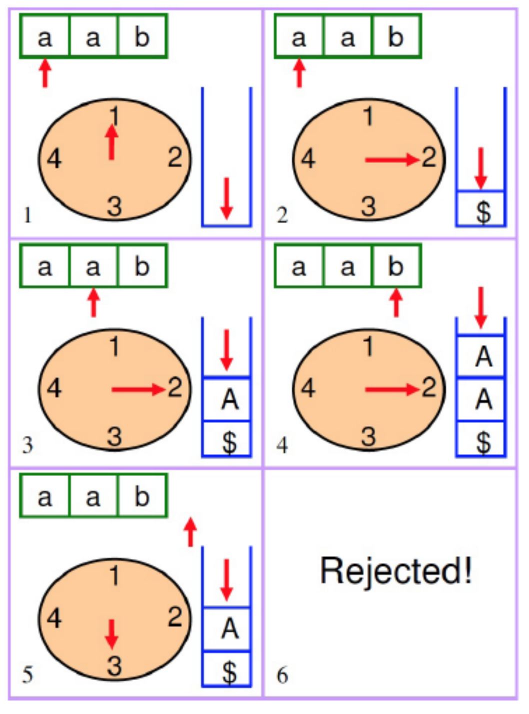
$a, \epsilon \to c$: read a, push c
$a, b \to \epsilon$: read a, pop b
$a, \epsilon \to \epsilon$: read a, do not change the stack
$\epsilon, \epsilon \to \epsilon$: do not read the input symbol and do not change the stack
$a, b \to c$: read a, pop b, push c

PDA - Example

- Consider the following PDA, which recognizes language $\{a^n b^n \mid n \geq 0\}$



Is "aab" accepted by the PDA?



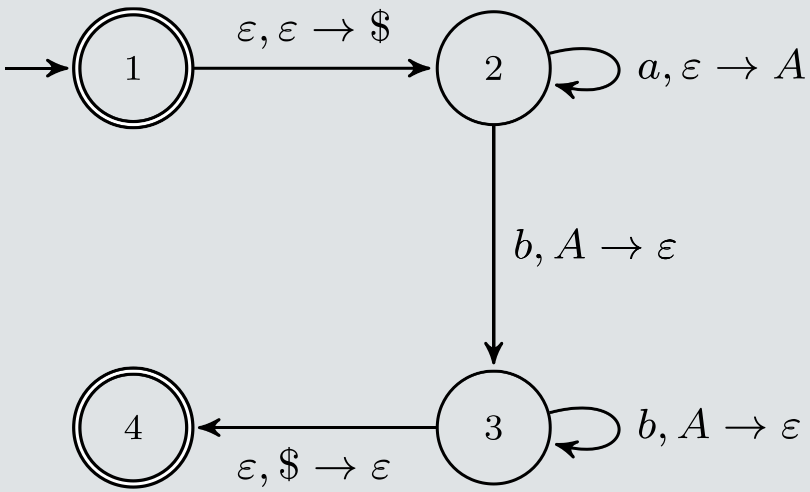
$$
\begin{aligned}
& (1, aab, \epsilon) \\
\Longrightarrow\ & (2, aab, \$) \\
\Longrightarrow\ & (2, ab, A\$) \\
\Longrightarrow\ & (2, b, AA\$) \\
\Longrightarrow\ & (3, \epsilon, A\$)
\end{aligned}
$$

Use 3-tuples that describe the configuration of the PDA (*current state, remaining input string, content of stack*)

Traverse the graph and keep track of the stack

PDA - Example

- Consider the following PDA, which recognizes language $\{a^n b^n \mid n \geq 0\}$



Is "aabb" accepted by the PDA?
(on whiteboard)

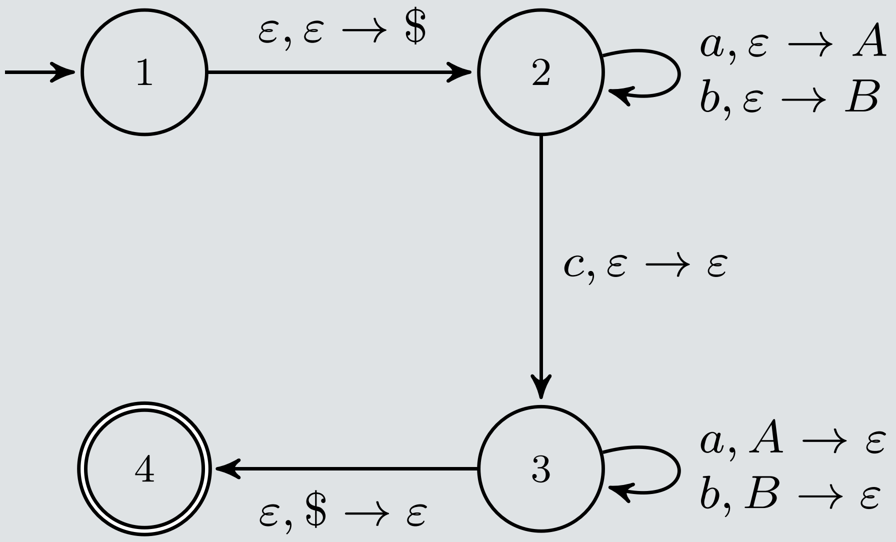
PDA - Design

- Since FA and PDA only differ on a stack, lets focus on how to use the stack
  - For counting,
    - Same number of a's and b's
    - More a's than b's
  - For making copy of a string
    - Palindromes
    - Reversing strings

PDA - Example

- Building a PDA that recognizes palindromes with a “c” in the middle
  - We use the stack to remember the first half of a palindrome
  - Once we see symbol c, we know that it is the middle of the palindrome, and then we can compare the second half with the contact of the stack
  - Starting with state 1, push $ onto the stack to mark the bottom of the stack
  - State 2 makes a copy of all the symbols before c
  - State 3 compares the symbols after c with the stack symbols

$$L_1 = \{wcw^R \mid w \text{ is a string over } \{a, b\}\}$$

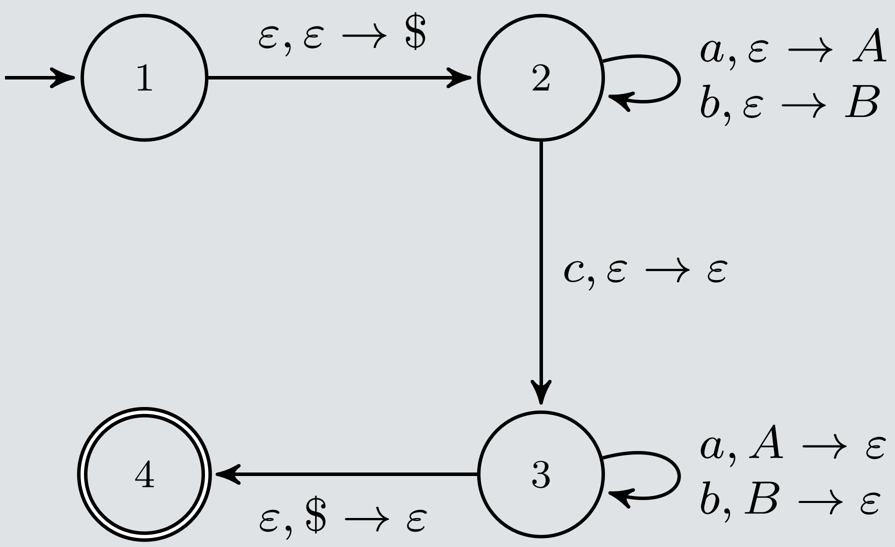


PDA - In-Class Exercise

Trace the computation of PDA1 for the following strings:

$$L_1 = \{wcw^R \mid w \text{ is a string over } \{a, b\}\}$$

1. **abacabb**
2. **abacab**
3. **abcbaa**



PDA1

PDA - In-class Exercise

- Design PDAs for the following languages

$$L_2 = \{a^m b^n \mid m, n \geq 0 \text{ and } m \geq n\}$$

$$L_3 = \{a^m b^n \mid m, n \geq 0 \text{ and } m < n\}$$

$$L_4 = \{a^m b^{2m} \mid m \geq 0\}$$

$$L_5 = \{a^m b^{m+2} \mid m \geq 0\}$$

PDA

- Clarifications
  - An input string is accepted by a PDA, if at least one copy of the PDA stops at a final state after reading the input string. Note that it is not necessary to have an empty stack at a final state, even though usually the stack is empty when a PDA stops.
  - A stack is a “last in, first out" storage device. Two operations with a stack: push and pop.
    - After the following operations: “push a”, “push b”, “push a”, “pop a”, “push b”, “pop b”, “pop b”, what are the elements in the stack?
  - There are two types of PDAs: deterministic and nondeterministic
    - They have different power (unlike FAs), and given that nondeterministic PDAs have the same power as context-free grammars, we will study only nondeterministic PDAs

Non-deterministic PDAs

- All pervious examples have deterministic PDAs
- For regular languages DFA and NFA have the same power, i.e., both of them recognize regular languages.
  - A DFA can simulate multiple copies on an NFA by having states that are power sets of an NFA.
- Not the case for PDAs
  - Non-deterministic PDA is more powerful than deterministic PDA.
  - In order of a deterministic PDA “simulate” a non-deterministic PDA it must have multiple stacks, one for each copy of a non-deterministic PDA.
  - But a deterministic PDA only has one stack – only can simulate a single copy of a non-deterministic PDA.
- We will study non-deterministic PDAs since they recognize **context-free languages**

Non-deterministic PDA

- Non-deterministic PDA that recognizes even length palindromes

$$L_2 = \{ww^R \mid w \text{ is a string over } \{a,b\}\}$$

- This time, we do not know where the middle point of the palindrome is.
  - For example, consider an input string starting with *abb*. When we read the third symbol, i.e., the last *b*, we do not know whether

  1. It is still in the first half of the input string like "*abbaabba*".
  2. It is the first symbol of the second half of the input sting like "abba".
- PDA can "guess" by try both possibilities by using non-determinism.

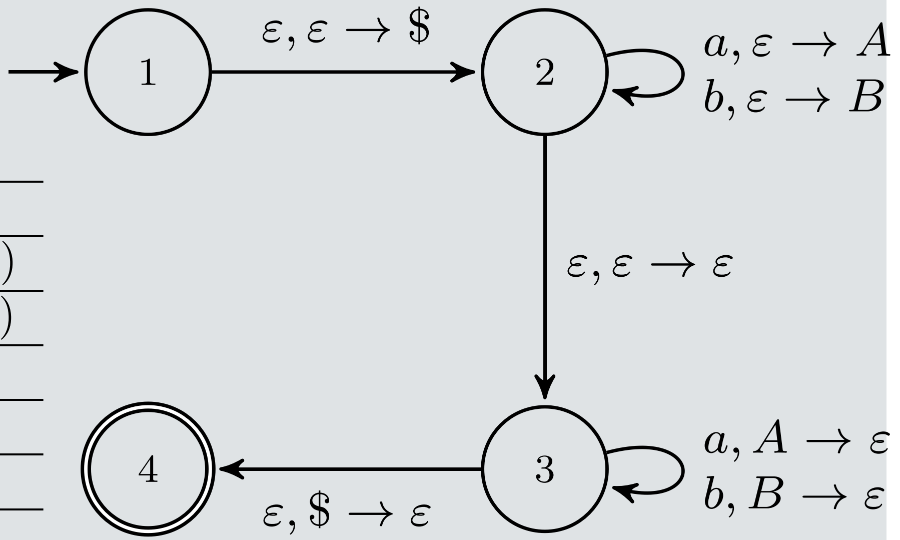
Non-deterministic PDA

Let's try an input string!

$$L_2 = \{ww^R \mid w \text{ is a string over } \{a, b\}\}$$

Try input string "*abba*"

copy 1	copy 5	copy 4	copy 3	copy 2
$(1, abba, \varepsilon)$				
$\vdash (2, abba, \$)$				
$\vdash (2, bba, A\$)$				$\vdash (3, abba, \$)$
$\vdash (2, ba, BA\$)$			$\vdash (3, bba, A\$)$	$\vdash (4, abba, \varepsilon)$
$\vdash (2, a, BBA\$)$		$\vdash (3, ba, BA\$)$	die	die
$\vdash (2, \varepsilon, ABBA\$)$	$\vdash (3, a, BBA\$)$	$\vdash (3, a, A\$)$		
$\vdash (3, \varepsilon, ABBA\$)$	die	$\vdash (3, \varepsilon, \$)$		
die		$\vdash (4, \varepsilon, \varepsilon)$		
		accepts		



In-class Exercise

- Build a (non-deterministic) PDA that recognizes the following language

$$L_3 = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$$

$$L_4 = \{w \mid w \text{ is a palindrome over } \{a, b\}\}$$

A PDA recognizes both even and odd length palindromes

Describing Languages

- Finite automata & regular expressions describe regular languages
- Can’t help is describing simple languages, e.g., $\{0^n 1^n \mid n \geq 0\}$

Context-Free Grammars

- More powerful method of describing languages (handle descriptions of features that have a recursive structure)

Context-Free Grammar

- Formal definition. A context-free grammar is a quadruple (V, Σ, R, S), where
  - V is a finite set of variables (non-terminals)
  - Σ, the alphabet, is a finite set of terminal symbols
  - R is a finite set of rules from
  - S ($\in V$) is the start variable
- Observations
  - A production rule has the form A → w, where A $\in V$ and w ∈ (V ∪ Σ)*
  - A rule of the form A → w applied to the string uAv yields uwv, and u and v define the *context* in which A occurs.
  - Because the context places no limitations on the applicability of a rule, such a grammar is called context-free grammar (CFG)

Context-Free Grammar

- Example of a grammar:

$$A \to 0A1$$
$$A \to \varepsilon$$

  - Has two **rules** (or substitution rules or production rules)
  - ***A*** on the left hand side is called a **variable**
  - The variable of the first rule is the start variable
  - All other symbols (except ε) are **terminals**, which are **0** and **1**.
- Can use one line to describe two or more rules

$$A \to 0A1 \mid \varepsilon$$

- The operator | is the union operator for production rules (OR)

Context-Free Grammar

- Grammar
  - Collection of substitution (or production) rules consisting of: variables, terminals, and start variable
  - Example: $\{0^n 1^n \mid n > 0\}$

Start variable

Variables

$$S \rightarrow A$$
$$A \rightarrow 0B1$$
$$B \rightarrow \epsilon \mid 0B1$$

Terminals

Can be written as
$B \rightarrow \epsilon$
$B \rightarrow 0B1$

CFG - Derivation

- General idea
  - u, v, and w are strings of variables and terminals
  - A → w is a rule of the grammar

uAv **yields/ derives** uwv

uAv ⟹ uwv

- Multiple substitutions result in a derivation
  - We say we can derive a string **v** from a string **u** (**u** ⟹* **v**) if for k ≥ 0 exists $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$

- Example
  - 0B1 → 00B11 → 000B111 → 000ε111 → 000111
  - 0B1 derives 000111, 0B1 ⟹* 000111

S→A
A→0B1
B→ϵ | 0B1

CFG - Language

- The ***language*** of a grammar G is the set of *terminal strings derivable* from the start symbol of G
  - $L(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \}$

- Generating string of a language based on grammar descriptions
  - Consider the language $\{0^n 1^n \mid n > 0\}$ and the grammar

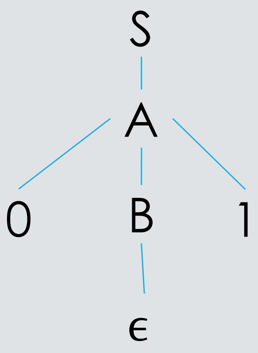
$S \rightarrow A$
$A \rightarrow 0B1$
$B \rightarrow \epsilon \mid 0B1$

$$S \Rightarrow A \Rightarrow 0B1 \Rightarrow 00B11 \Rightarrow 00\epsilon11 \Rightarrow 0011$$

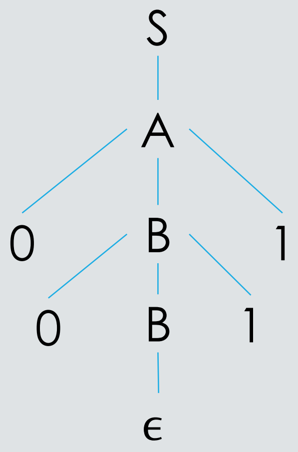
CFG - Parse Tree

- Another way to describe the sequence of substitutions
- The top node is the start variable
- Leaves (external nodes) are either terminals or ε
- Branches (internal nodes) are variables

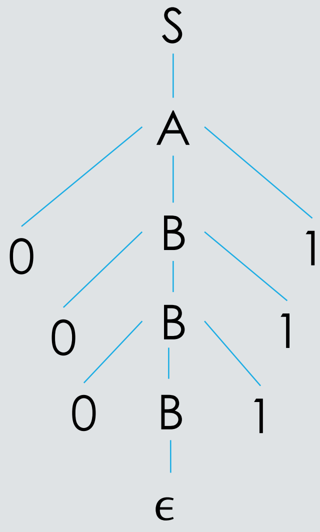
$S \rightarrow A$
$A \rightarrow 0B1$
$B \rightarrow \epsilon \mid 0B1$



Parse tree for 01



Parse tree for 0011



Parse tree for 000111

CFG – Left-Most Derivations

- Left- (Right-)most derivations:
  - Derivation that transforms the 1st variable occurring in a string from left-to-right (right-to-left)

$$
\begin{aligned}
&G = (V, \Sigma, P, S) \\
&V = \{S, A\} \\
&\Sigma = \{a, b\} \\
&P:\quad S \to AA \\
&\qquad A \to AAA \mid bA \mid Ab \mid a
\end{aligned}
$$

(a)	(b)	(c)	(d)
$S \Rightarrow AA$	$S \Rightarrow AA$	$S \Rightarrow AA$	$S \Rightarrow AA$
$\Rightarrow aA$	$\Rightarrow AAAA$	$\Rightarrow Aa$	$\Rightarrow aA$
$\Rightarrow aAAA$	$\Rightarrow aAAA$	$\Rightarrow AAAa$	$\Rightarrow aAAA$
$\Rightarrow abAAA$	$\Rightarrow abAAA$	$\Rightarrow AAbAa$	$\Rightarrow aAAa$
$\Rightarrow abaAA$	$\Rightarrow abaAA$	$\Rightarrow AAbaa$	$\Rightarrow abAAa$
$\Rightarrow ababAA$	$\Rightarrow ababAA$	$\Rightarrow AbAbaa$	$\Rightarrow abAbAa$
$\Rightarrow ababaA$	$\Rightarrow ababaA$	$\Rightarrow Ababaa$	$\Rightarrow ababAa$
$\Rightarrow ababaa$	$\Rightarrow ababaa$	$\Rightarrow ababaa$	$\Rightarrow ababaa$

Left **Right**

CFG - Ambiguity

- The possibility of a string having several derivations introduces the notion of **ambiguity**
- If a grammar generates the same string from several derivations, the string is derived **ambiguously**
- If a grammar generates some string ambiguously, then the grammar is **ambiguous**
- A grammar is **unambiguous** if, at each derivation step, there is only one rule that can lead to a derivation of the desired string.
- There are some context-free languages that cannot be generated by any *unambiguous* grammars. Such languages are called **inherently ambiguous**

Designing Context-Free Grammars

- From a definition of a language
  - Consider if it is a union of simple CFGs and merge simpler CFGs
  - Try to decompose the language into the concatenation of several short ones
  - Consider how valid strings in the language are linked. E.g. for $\{0^n1^n \mid n \geq 1\}$, you'll need a rule that accounts for the fact that every time a "0" is seen, a "1" is required: $S_1 \rightarrow 0S_21$
  - Decompose the language into the concatenation of several short ones, so that we can find a recursive structure.
  - Try to look for patterns in the language, to create rules

Example 1

$L_1 = \{a^n b^n \mid n \geq 0\}$

1. $S \to aSb$
2. $S \to \varepsilon$

In-class Exercises

- Design CFG for the following languages:

$$L_{e1} = \{b^{2m}a^m \mid m \geq 0\}$$

Example 2

$$L_2 = \{a^m b^n \mid m > n \geq 0\}$$

Concatenation of smaller languages: $L_2 = L_{21} L_{22}$ where

- $L_{21} = \{a^i \mid i > 0\}$
- $L_{22} = L_1$ -- the same number of a's and b's

1. $S \to S_{21}S_{22}$
2. $S_{21} \to aS_{21} \mid a$ (the grammar for L_{21})
3. $S_{22} \to aS_{22}b \mid \varepsilon$ (the grammar for L_{22} as in **Example 1**)

In-class Exercises

- Design CFG for the following language as a concatenation of two languages:

$$L_{e2} = \{b^{2m}a^{m+k} \mid m, k \geq 0, k < 3\}$$

Example 3

$$L_3 = \{a^m b^n \mid m \neq n\}$$

$L_3 = L_{31} \mid L_{32}$

- $L_{31} = L_2$ – more a's than b's
- L_{32} = { $a^m b^n$ | n > m >= 0} -- more b's than a's
- $L_{32} = L_{321} L_{322}$ -- similar to L2 decomposition
  - $L_{321} = L_1$
  - $L_{322} = \{b^i \mid i > 0\}$

1. $S \to S_{31} \mid S_{32}$
2. $S_{31} \to S_{311} S_{312}$ (the 1st rule in L_2 grammar as in **Example 2**)
3. $S_{311} \to aS_{311} \mid a$ (the 2nd rule in L_2 grammar as in **Example 2**)
4. $S_{312} \to aS_{312}b \mid \varepsilon$ (the grammar for L_1 as in **Example 1**)
5. $S_{32} \to S_{312} S_{322}$
6. $S_{322} \to bS_5 \mid b$

In-class Exercises

- Design CFG for the following language as a union of two languages:

$$L_{e3} = \{b^{2m}a^{m+k} \mid m, k \geq 0, k \neq 3\}$$

Example 4

$$L_4 = \{x \text{ } over \text{ } \{a, b\} | \text{ } x \text{ } is \text{ } a \text{ } palindrome\}$$

- A palindrome has a recursive structure
- If we remove the fist symbol and the last symbol then the remaining string is till a palindrome
- Two cases for which it holds
  1. The first and the last symbols are a’s
  2. The first and the last symbols are b’s
- Special cases for all palindromes with length less than 2

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$$

In-class Exercise

$L_{e4} = \{x \text{ over } \{a, b\} \mid x \text{ is } \textbf{not} \text{ a palindrome}\}$

CFG – Example Mini-Compiler

- This is example 2.4 from the book (Chapter 2.1, Page 105)

Consider grammar $G_4 = (V, \Sigma, R, \langle\text{EXPR}\rangle)$.

V is $\{\langle\text{EXPR}\rangle, \langle\text{TERM}\rangle, \langle\text{FACTOR}\rangle\}$ and Σ is $\{\texttt{a}, \texttt{+}, \texttt{x}, \texttt{(}, \texttt{)}\}$. The rules are

$$
\begin{aligned}
\langle\text{EXPR}\rangle &\rightarrow \langle\text{EXPR}\rangle\texttt{+}\langle\text{TERM}\rangle \mid \langle\text{TERM}\rangle \\
\langle\text{TERM}\rangle &\rightarrow \langle\text{TERM}\rangle\texttt{x}\langle\text{FACTOR}\rangle \mid \langle\text{FACTOR}\rangle \\
\langle\text{FACTOR}\rangle &\rightarrow \texttt{(}\,\langle\text{EXPR}\rangle\,\texttt{)} \mid \texttt{a}
\end{aligned}
$$

The two strings a+axa and (a+a)xa can be generated with grammar G_4.

CFG – Example Mini-Compiler

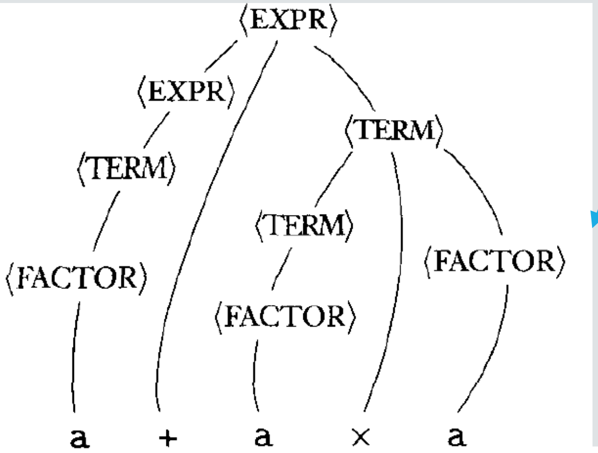
- This is example 2.4 from the book (Chapter 2.1, Page 105)

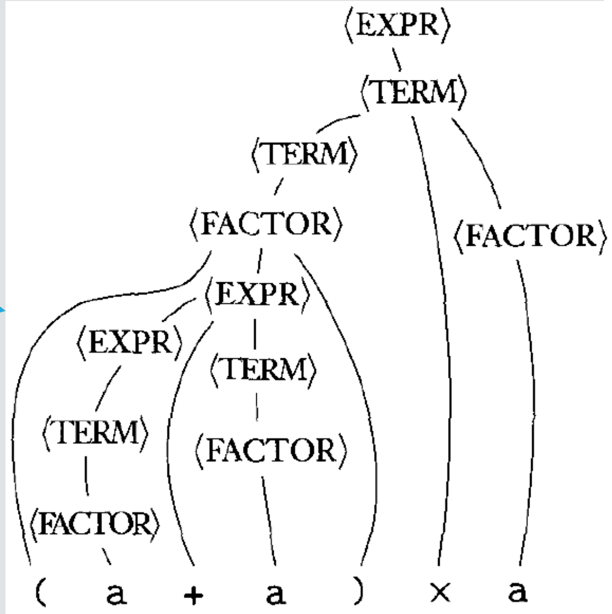
Consider grammar $G_4 = (V, \Sigma, R, \langle \text{EXPR} \rangle)$.

V is $\{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$ and Σ is $\{\mathtt{a}, +, \times, (,)\}$. The rules are

$$
\begin{aligned}
\langle \text{EXPR} \rangle &\to \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \\
\langle \text{TERM} \rangle &\to \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle &\to (\langle \text{EXPR} \rangle) \mid \mathtt{a}
\end{aligned}
$$

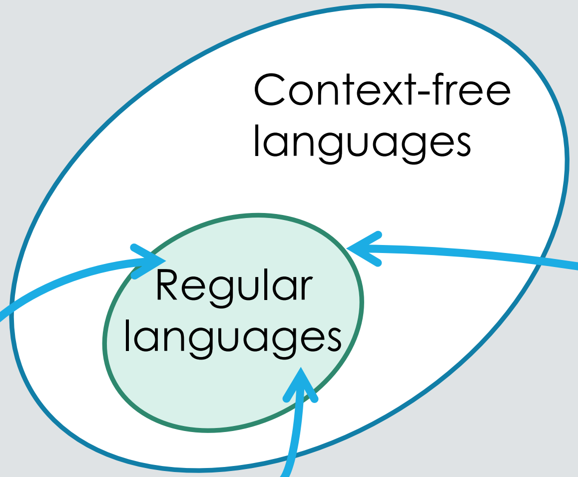
The two strings $\mathtt{a+a\times a}$ and $\mathtt{(a+a)\times a}$ can be generated with grammar G_4.





Regular and Context-free Languages

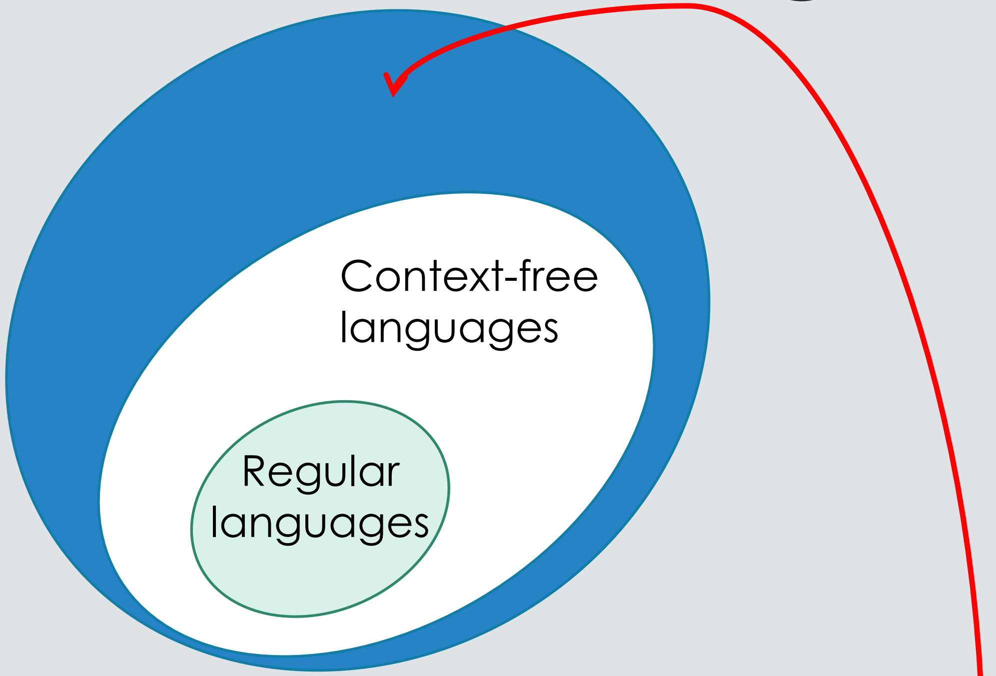
- Relationship between regular and context-free languages



- How to create a grammar for a DFA?
- How to create a grammar for a RegEx?

Pumping Lemma for Regular Languages

Non-Context Free Languages



Certain languages are not context free

- To prove it, we rely on the "pumping lemma for context-free languages"

Pumping lemma for context-free languages

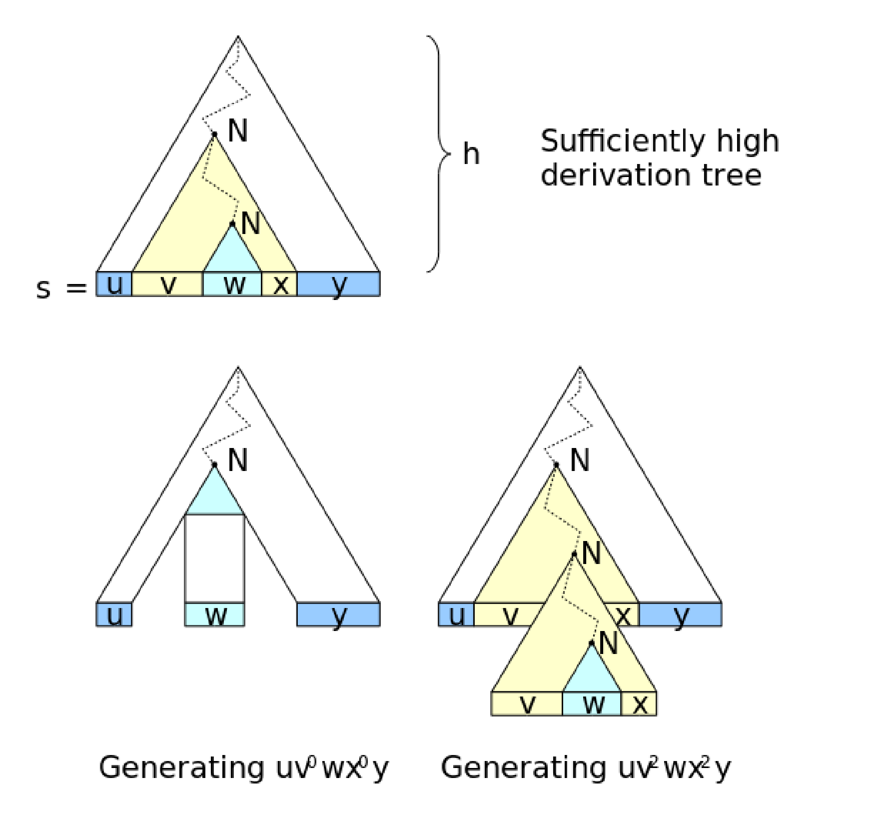
- Theorem 2.34: If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into five pieces $s=uvxyz$ satisfying the conditions
  1. For each $i \geq 0$, $uv^ixy^iz \in A$
  2. $|v| + |y| > 0$
  3. $|vxy| \leq p$

    - Condition 2 guarantees that either v or y is not the empty string
    - Condition 3 states that v, x, and y have together a length of at most p

Informally

- The pumping lemma for CFL's states that for sufficiently long strings in a CFL, we can find two, short, nearby substrings that we can "pump" in tandem and the resulting string must also be in the language

Pumping lemma for context-free languages



Explore the possibilities to find a contradiction:

- v and y contain only one type of alphabet symbol
- Either v or y contains more than one type of symbol

Theorem: The language

$$L = \{a^n b^n c^n : n \geq 0\}$$

is **not** context free

Proof: Use the Pumping Lemma for context-free languages

$$L = \{a^n b^n c^n : n \geq 0\}$$

Assume for contradiction that L is context-free

Let m be the critical length of the pumping lemma

Pick any string $w \in L$ with length $|w| \geq m$

We pick: $w = a^m b^m c^m$

We can write: $w = uvxyz$

With lengths $|vxy| \leq m$ and $|vy| \geq 1$

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Pumping Lemma says:

$uv^i xy^i z \in L$ for all $i \geq 0$

We examine **<u>all</u>** the possible locations of string vxy in w

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 1: vxy is in a^m

$$v = a^{k_1} \qquad y = a^{k_2} \qquad k_1 + k_2 \geq 1$$

$$\underbrace{a...a}_{u}\underbrace{a...a}_{v}\underbrace{a...a}_{x}\underbrace{a...aa...a}_{y}\underbrace{bbb...bbb\,ccc...ccc}_{z}$$

$$\overbrace{a...aa...aa...aa...aa...a}^{m}\;\overbrace{bbb...bbb}^{m}\;\overbrace{ccc...ccc}^{m}$$

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

$$v = a^{k_1} \qquad y = a^{k_2} \qquad k_1 + k_2 \geq 1$$

$$\underbrace{\overbrace{\underbrace{a...a}_{u}\underbrace{a...a}_{v^2}\underbrace{a...a}_{x}\underbrace{a...a...a}_{y^2}}^{m+k_1+k_2}\ \underbrace{\overbrace{bbb...bbb}^{m}\ \overbrace{ccc...ccc}^{m}}_{z}}$$

However: $uv^2xy^2z = a^{m+k_1+k_2} b^m c^m \notin L$

Contradiction!!!

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 2: vxy is in b^m Similar to case 1

$$\underbrace{\overbrace{aaa\ldots aaa}^{m}}_{u}\ \underbrace{\overbrace{b\ldots bb\ldots bb\ldots b}^{m}}_{vxy}\ \underbrace{\overbrace{ccc\ldots ccc}^{m}}_{z}$$

However: $uv^2xy^2z = a^m b^{m+k_1+k_2} c^m \notin L$

Contradiction!!!

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 3: vxy is in c^m

Similar to case 1

$$\underbrace{\overbrace{aaa...aaa}^{m}\ \overbrace{b...bb...bb...b}^{m}}_{u}\ \overbrace{\underbrace{ccc...c}_{vxy}\underbrace{cc}_{z}}^{m}$$

However: $uv^2xy^2z = a^m b^m c^{m+k_1+k_2} \notin L$

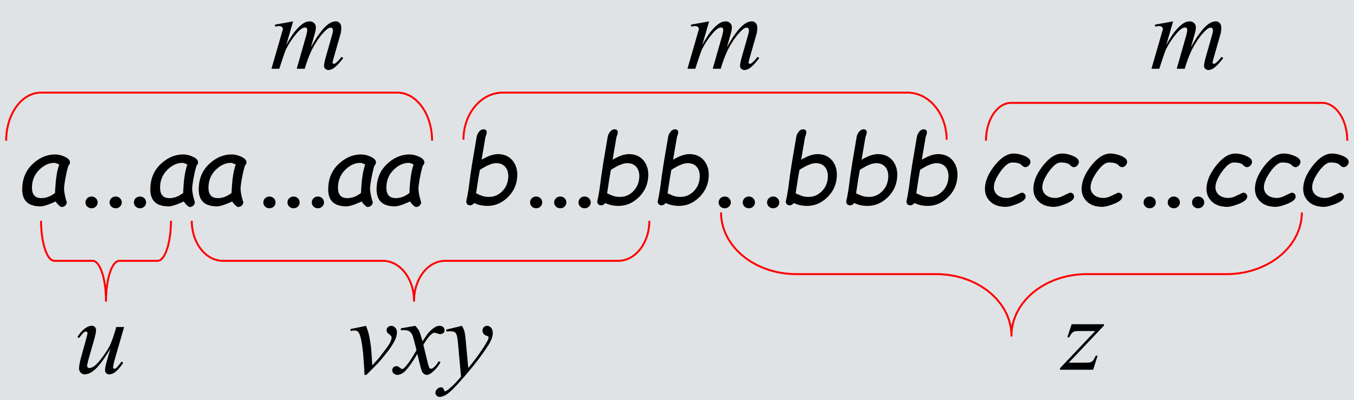
Contradiction!!!

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 4: vxy overlaps a^m and b^m



$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Sub-case 1: v contains only a $\quad y$ contains only b

$$\overbrace{a\ldots aa\ldots aa\ldots a}^{m}\ \overbrace{b\ldots bb\ldots bb\ldots b}^{m}\ \overbrace{ccc\ldots ccc}^{m}$$

$$\underbrace{a\ldots a}_{u}\,\underbrace{a\ldots a}_{v}\,\underbrace{a\ b\ldots}_{x}\,\underbrace{b\ldots b}_{y}\,\underbrace{\ldots b\ ccc\ldots ccc}_{z}$$

However: $uv^2xy^2z = a^{m+k_1} b^{m+k_2} c^m \notin L$

Contradiction!!!

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Sub-case 2: v contains a and b $\quad$ y contains only b

$$\underbrace{\overbrace{a \ldots a a \ldots a}^{m}}_{u} \; \overbrace{b \ldots b b \ldots b b \ldots b b \ldots b}^{m} \; \overbrace{ccc \ldots ccc}^{m}$$

$u \quad v \quad x \quad y \quad z$

However: $uv^2xy^2z = a^{m-k_1}(ab)^{2k_1}b^{m-k_1+k_2}c^m \notin L$

Contradiction!!!

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Sub-case 3: v contains only a y contains a and b

Similar to sub-case 2

$$\overbrace{\underbrace{a\ldots a}_{u}\underbrace{a\ldots a}_{v}\underbrace{a}_{x}}^{m}\overbrace{\underbrace{b}_{y}\ldots bb\ldots bb\ldots bb\ldots b}^{m}\overbrace{ccc\ldots ccc}^{m}$$

z = $b\ldots bb\ldots bb\ldots bb\ldots b\,ccc\ldots ccc$

However: $uv^2xy^2z = a^{m+k_1-k_2}(ab)^{2k_2}b^{m-k_2}c^m \notin L$

Contradiction!!!

$$L = \{a^n b^n c^n : n \geq 0\}$$

$$w = a^m b^m c^m$$

$$w = uvxyz \qquad |vxy| \leq m \qquad |vy| \geq 1$$

Case 5: vxy overlaps b^m and c^m

Similar to case 4

Contradiction!!!

In all cases we obtained a contradiction

Conclusion: L is not context-free

Pumping Lemma: a Conversation

I think I can show $0^n 1^n 2^n$ isn't context free

Hmm.. Not sure I believe.. Tell me about it

Pumping Lemma: a Conversation

*If that thing is context free, I get this magic constant **p***

Oh, I remember! You can divide any word in that thing into three parts…

Pumping Lemma: a Conversation

No! Pay attention! It's FIVE parts, ***w=uvxyz****!*

Ok.
Let's say $\mathbf{w=0^p1^p2^p}$.
Then what?

Pumping Lemma: a Conversation

w can be broken down into uvxyz, where ***vy is nonempty*** *and* ***vxy has length at most p***

I see. Then v and y can touch ***at most two sections***

Pumping Lemma: a Conversation

Right!
*So by **pumping** up you can create a string that **can't be** in that thing*

I still don't get it…

Pumping lemma for CFL

Example: Show that $A=\{a^nb^nc^n \mid n \geq 0\}$ is not context free

- Assume A is a CFL
- Let p be the pumping length for A that is guarantee to exist by the pumping lemma
- Select $s = a^pb^pc^p$, such that $s \in A$ and $|s| \geq p$. Re-write $s = uvxyz$
- Based on the pumping lemma, for each $i \geq 0$, $uv^ixy^iz \in A$. However, we show that s cannot be pumped, without violating one of the 3 constraints defined in Theorem 2.34
  - When both v and y contain only one type of symbol, v does not contain both a's and b's or b's and c's, and the same hold for y. Thus, for i=2, i.e., uv^2xy^2z, s can't contain equal number of a's b's and c's, and thus $s \notin A$, which violates condition 1 and is a contradiction.
  - When either v or y contain more than one type of symbol, then for for i=2, i.e., uv^2xy^2z, may contain equal number of alphabet symbols, but not in the correct order. Hence $s \notin A$, which is a contradiction
- Given the contradiction, the original assumption must be false, proving that A is not a CFL

Pumping Lemma – More Examples

- A=$\{a^p b^q c^r \mid p < q \ and \ q < r\}$
- B=$\{0^p 1^q 0^p 1^q \mid p, q > 0\}$
- C={ww| w is a string over {s, t} and |w|>0}

CFL or Not?

- $L = \{a^p b^p c^p \mid p > 0\}$
- $A = \{a^p b^p c^q \mid p, q > 0\}$
- $B = \{a^p b^q c^q \mid p, q > 0\}$
- $A \cap B =$?