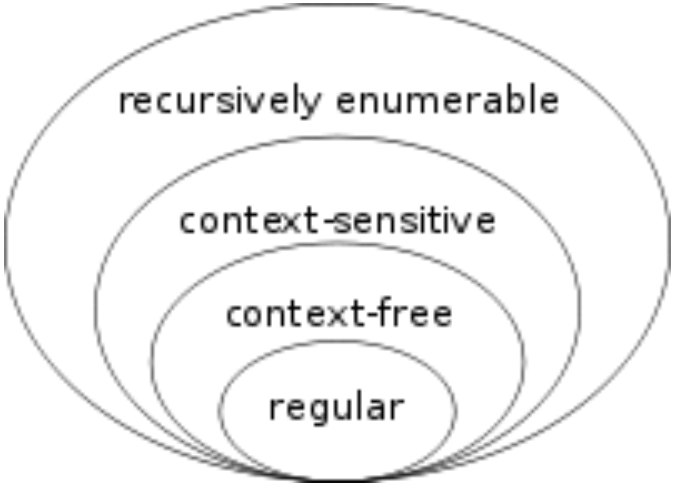
Lab 4

COP3402 - Jan 31

Chomsky Hierarchy (HW2 Q1 & Q2)

1. All context-free languages are also regular languages
   a. True
   b. False ←

2. All regular languages are also context-free languages
   a. True ←
   b. False



HW2 Q3

Is this context-free grammar regular, where A is a nonterminal, a is a terminal, and ε is the empty string?

A -> A
A -> a
A -> ε

HW2 Q4

Is this context-free grammar regular?

A -> a A c
A -> ε

HW2 Q5

Is this regular language also a context-free language?

R= (0|1)*11*0

HW2 Q6

6. Consider S -> aSa | bSb | a | b;
   The language generated by the above grammar over the alphabet {a,b} is the set of:
   a. All palindromes
   b. All even length palindromes
   c. All odd length palindromes
   d. Strings that begin and end with the same symbol

HW2 Q7

7. Consider the CFG with {S,A,B) as the non-terminal alphabet, {a,b) as the terminal alphabet, S as the start symbol and the following set of production rules

```
S --> aB        S --> bA

B --> b         A --> a

B --> bS        A --> aS

B --> aBB       A --> bAA
```

Which of the following strings is generated by the grammar?

a. abbbba
b. aaaabb
c. aabbab
d. aabbbb

Recursive Descent Parser

A **recursive descent parser** is a kind of **top-down parser** built from a set of mutually **recursive procedures** (or a non-recursive equivalent) where **each such procedure implements one of the nonterminals of the grammar**.

Notice: Non-terminals are procedures (functions)

Demo: Write recursive descent parser

Grammar (in CFG format):

S → aB

B → bB | epsilon

Non-terminals: S, B

Terminals: a, b, epsilon

Functions/variables that can be used:

`current_token` : The current token

`next_token()` : Advance to the next token

`error()` : Error function to be called if the string does not match with the grammar

Demo: Write recursive descent parser

Part of the grammar for PL/0 (in EBNF format):

```
funcdecls ::= { FUNC IDENT LPAREN [ formals ] RPAREN [ COLON type ] block }

formals    ::= IDENT COLON type { COMMA IDENT COLON type }
```

Non-terminals: funcdecls, formals, block, type
Terminals: FUNC, IDENT, LPAREN, RPAREN, COLON, COMMA (remember the lexer assignment!)

Functions/variables that can be used:

- **`current_token`** : The current token
- **`next_token()`** : Advance to the next token
- **`error()`** : Error function to be called if the string does not match with the grammar

Assume that the functions **`formals(), type(), block()`** are already implemented (we can call them)