Context Free Languages and Grammars

Lecture 8

Feb 13, 2020

Programming Language Design

Question: What is a valid C program? Or a Python program?

Question: Given a string w what is an algorithm to check whether w is a valid C program? The parsing problem.

Context Free Languages and Grammars

- Programming Language Specification
- Parsing
- Natural language understanding
- Generative model giving structure
- ...

CFLs provide a good balance between expressivity and tractability. Limited form of recursion.

Programming Languages

```
<relational-expression> ::= <shift-expression>
                          | <relational-expression> < <shift-expression>
                          | <relational-expression> > <shift-expression>
                          | <relational-expression> <= <shift-expression>
                          | <relational-expression> >= <shift-expression>

<shift-expression> ::= <additive-expression>
                     | <shift-expression> << <additive-expression>
                     | <shift-expression> >> <additive-expression>

<additive-expression> ::= <multiplicative-expression>
                        | <additive-expression> + <multiplicative-expression>
                        | <additive-expression> - <multiplicative-expression>

<multiplicative-expression> ::= <cast-expression>
                              | <multiplicative-expression> * <cast-expression>
                              | <multiplicative-expression> / <cast-expression>
                              | <multiplicative-expression> % <cast-expression>

<cast-expression> ::= <unary-expression>
                    | ( <type-name> ) <cast-expression>

<unary-expression> ::= <postfix-expression>
                     | ++ <unary-expression>
                     | -- <unary-expression>
                     | <unary-operator> <cast-expression>
                     | sizeof <unary-expression>
                     | sizeof <type-name>

<postfix-expression> ::= <primary-expression>
                       | <postfix-expression> [ <expression> ]
                       | <postfix-expression> ( {<assignment-expression>}* )
                       | <postfix-expression> . <identifier>
                       | <postfix-expression> -> <identifier>
                       | <postfix-expression> ++
                       | <postfix-expression> --
```

Natural Language Processing

English sentences can be described as

$$
\begin{aligned}
&\langle S\rangle \to \langle NP\rangle\langle VP\rangle \\
&\langle NP\rangle \to \langle CN\rangle \mid \langle CN\rangle\langle PP\rangle \\
&\langle VP\rangle \to \langle CV\rangle \mid \langle CV\rangle\langle PP\rangle \\
&\langle PP\rangle \to \langle P\rangle\langle CN\rangle \\
&\langle CN\rangle \to \langle A\rangle\langle N\rangle \\
&\langle CV\rangle \to \langle V\rangle \mid \langle V\rangle\langle NP\rangle \\
&\langle A\rangle \to \text{a} \mid \text{the} \\
&\langle N\rangle \to \text{boy} \mid \text{girl} \mid \text{flower} \\
&\langle V\rangle \to \text{touches} \mid \text{likes} \mid \text{sees} \\
&\langle P\rangle \to \text{with}
\end{aligned}
$$

English Sentences

Examples

$$\overbrace{\underbrace{\text{a}}_{\text{article}}\ \underbrace{\text{boy}}_{\text{noun}}}^{\text{noun-phrs}}\ \overbrace{\underbrace{\text{sees}}_{\text{verb}}}^{\text{verb-phrs}}$$

$$\overbrace{\underbrace{\text{the}}_{\text{article}}\ \underbrace{\text{boy}}_{\text{noun}}}^{\text{noun-phrs}}\ \overbrace{\underbrace{\text{sees}}_{\text{verb}}\ \underbrace{\text{a flower}}_{\text{noun-phrs}}}^{\text{verb-phrs}}$$

Example

- Symbols $= \{0, 1\}$
- Varibale S.
- Apply following rules recursively starting with S:
  $S \to \epsilon, S \to 0, S \to 1,$
  $S \to 0S0, S \to 1S1$

Example

- Symbols $= \{0, 1\}$
- Varibale S.
- Apply following rules recursively starting with S:
  $S \to \epsilon, S \to 0, S \to 1,$
  $S \to 0S0, S \to 1S1$

$$S \rightsquigarrow 0S0 \rightsquigarrow 01S10 \rightsquigarrow 011S110 \rightsquigarrow 01110$$

- Symbols $= \{0, 1\}$
- Varibale S.
- Apply following rules recursively starting with S:
  $S \to \epsilon, S \to 0, S \to 1,$
  $S \to 0S0, S \to 1S1$

0111110

$$S \leadsto 0S0 \leadsto 01S10 \leadsto 011S110 \leadsto \cancel{01110}$$

Starting from S, what all strings can we generate like this?

Palindromes

- Madam in Eden I'm Adam
- Dog doo? Good God!
- Dogma: I am God.
- A man, a plan, a canal, Panama
- Are we not drawn onward, we few, drawn onward to new era?
- Doc, note: I dissent. A fast never prevents a fatness. I diet on cod.
- `http://www.palindromelist.net`

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols
- T is a finite set of terminal symbols (alphabet)

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form
  $A \rightarrow \alpha$
  where $A \in V$ and α is a string in $(V \cup T)^*$.
  Formally, $P \subset V \times (V \cup T)^*$.

Context Free Grammar (CFG) Definition

Definition

A CFG is is a quadruple $G = (V, T, P, S)$

- V is a finite set of non-terminal symbols
- T is a finite set of terminal symbols (alphabet)
- P is a finite set of productions, each of the form
  $A \to \alpha$
  where $A \in V$ and α is a string in $(V \cup T)^*$.
  Formally, $P \subset V \times (V \cup T)^*$.
- $S \in V$ is a start symbol

Palindrome Example

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}$
(also written as $\{S \to \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1\}$)

$$S \rightsquigarrow 0S0 \rightsquigarrow 01S10 \rightsquigarrow 011S110 \rightsquigarrow 01110$$

Palindrome Example

- $V = \{S\}$
- $T = \{0, 1\}$
- $P = \{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}$
(also written as $\{S \to \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1\}$)

$$S \rightsquigarrow 0S0 \rightsquigarrow 01S10 \rightsquigarrow 011S110 \rightsquigarrow 01110$$

Example

$L = \{0^n 1^n \mid n \geq 0\}$

$V = \{S\}$

$T = \{0, 1\}$

$P = \{ S \rightarrow \epsilon, \underline{S \rightarrow 0S1} \}$

$S \rightarrow 0S1 \rightarrow 0\underline{0S1}1 \Rightarrow 00\underline{0S1}11$

$\rightarrow 000\underline{\epsilon}111$

Example

$L = \{0^n 1^n \mid n \geq 0\}$

$S \to \epsilon \mid 0S1$

Notation and Convention

Let $G = (V, T, P, S)$ then

- $a, b, c, d, \ldots,$ in T (terminals)
- $A, B, C, D, \ldots,$ in V (non-terminals)
- $u, v, w, x, y, \ldots$ in T^* for strings of terminals
- $\alpha, \beta, \gamma, \ldots$ in $(V \cup T)^*$
- X, Y, Z in $V \cup T$

“Derives” relation

Formalism for how strings are derived/generated

Definition

Let $G = (V, T, P, S)$ be a CFG. For strings $\alpha_1, \alpha_2 \in (V \cup T)^*$ we say α_1 derives α_2 denoted by $\alpha_1 \rightsquigarrow_G \alpha_2$ if there exist strings β, γ, δ in $(V \cup T)^*$ such that

- $A \rightarrow \gamma$ is in P.
- $\alpha_1 = \beta A \delta$
- $\alpha_2 = \beta \gamma \delta$

Examples: For $S \rightarrow \epsilon \mid 0S1$,

$S \rightsquigarrow \epsilon$, $S \rightsquigarrow 0S1$, $0S1 \rightsquigarrow 00S11$, $0S1 \rightsquigarrow 01$.

"Derives" relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.

"Derives" relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \leadsto^k \alpha_2$ inductive defined:

- $\alpha_1 \leadsto^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \leadsto^k \alpha_2$ if $\alpha_1 \leadsto \beta_1$ and $\beta_1 \leadsto^{k-1} \alpha_2$.
- Alternative defn: $\alpha_1 \leadsto^k \alpha_2$ if $\alpha_1 \leadsto^{k-1} \beta_1$ and $\beta_1 \leadsto \alpha_2$

"Derives" relation continued

Definition

For integer $k \geq 0$, $\alpha_1 \rightsquigarrow^k \alpha_2$ inductive defined:

- $\alpha_1 \rightsquigarrow^0 \alpha_2$ if $\alpha_1 = \alpha_2$
- $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow \beta_1$ and $\beta_1 \rightsquigarrow^{k-1} \alpha_2$.
- Alternative defn: $\alpha_1 \rightsquigarrow^k \alpha_2$ if $\alpha_1 \rightsquigarrow^{k-1} \beta_1$ and $\beta_1 \rightsquigarrow \alpha_2$

$\rightsquigarrow^*$ is the reflexive and transitive closure of $\rightsquigarrow$.

$\alpha_1 \rightsquigarrow^* \alpha_2$ if $\alpha_1 \rightsquigarrow^k \alpha_2$ for some $k \geq 0$.

$$\alpha_1 \rightsquigarrow^{k_1} \alpha_2 \rightsquigarrow^{k_2} \alpha_3$$
$$\Downarrow$$
$$\alpha_1 \rightsquigarrow^{k_1+k_2} \alpha_3$$

Examples: For $S \rightarrow \epsilon \mid 0S1$,
$S \rightsquigarrow^* \epsilon$, $0S1 \rightsquigarrow^* 0000011111$.

Context Free Languages

Definition

The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{w \in T^* \mid S \rightsquigarrow^* w\}$.

Context Free Languages

Definition

The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{w \in T^* \mid S \rightsquigarrow^* w\}$.

Definition

A language L is context free (CFL) if it is generated by a context free grammar. That is, there is a CFG G such that $L = L(G)$.

Examples

$L_1 = \{0^n 1^n \mid n \geq 0\}$

$G_1 = \{S \to \epsilon \mid 0S1\}$

$L_2 = \{0^n 1^m \mid m > n \geq 0\}$

$G_2 = \{ \underbrace{S \to 1,\ S \to 0S1}_{m = n+1},\ S \to S1 \}$

$m \neq n$!

$L_3 = \{0^n 1^m \mid m < n\}$

$0^{n-m}(0^m 1^m)$

$G_3 = \{ S \to 0,\ S \to 0S,\ S \to 0S1 \}$

Examples

$L_4 = \{w \in \{(,)\}^* \mid w$ is properly nested string of parenthesis$\}$

$\boxed{((())) \; ()(())}$ ✓ $)($ ✗

$\{ S \to \epsilon, \boxed{S \to (S)}, \; S \to S(S)S, \; S \to SS \}$

$L_5 = \{w \in \{\mathbf{0}, \mathbf{1}\}^* \mid w$ has twice as many $\mathbf{1}$s as $\mathbf{0}$'s$\}$

Inductive proofs for CFGs

Question: Given a CFG G, How do we formally prove that $L(G) = L$?

Example: $G : S \to \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

Theorem

For $L = \{palindromes\} = \{w \mid w = w^R\}$, $L(G) = L$.

Inductive proofs for CFGs

Question: Given a CFG G, How do we formally prove that $L(G) = L$?

Example: $G : S \rightarrow \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

Theorem

For $L = \{palindromes\} = \{w \mid w = w^R\}$, $L(G) = L$.

Two directions:

- $L(G) \subseteq L$, that is, $S \rightsquigarrow^* w$ then $w = w^R$
- $L \subseteq L(G)$, that is, $w = w^R$ then $S \rightsquigarrow^* w$

$\mathbf{L(G) \subseteq L}$

Show that if $S \rightsquigarrow^* w$ then $w = w^R$

By induction on length of derivation, meaning
For all $k \geq 1$, $S \rightsquigarrow^{*k} w$ implies $w = w^R$.

$\mathbf{L(G) \subseteq L}$

Show that if $S \rightsquigarrow^* w$ then $w = w^R$

$\boxed{S \rightarrow \epsilon / 0 / 1 / 0S0 / 1S1}$

By induction on length of derivation, meaning
For all $k \geq 1$, $S \rightsquigarrow^{*k} w$ implies $w = w^R$.

- If $S \rightsquigarrow^1 w$ then $w = \epsilon$ or $w = 0$ or $w = 1$. Each case $w = w^R$.
- Assume that for all $k < n$, that if $S \rightsquigarrow^k w$ then $w = w^R$
- Let $S \rightsquigarrow^n w$ (with $n > 1$). Wlog w begin with $\mathbf{0}$.
  - Then $S \rightarrow 0S0 \rightsquigarrow^{n-1} 0u0$ where $w = 0u0$.
  - And $S \rightsquigarrow^{n-1} u$ and hence IH, $u = u^R$.
  - Therefore $w^R = (0u0)^R = (u0)^R 0 = 0u^R 0 = 0u0 = w$.

$L \subseteq L(G)$

Show that if $w = w^R$ then $S \rightsquigarrow^* w$.

By induction on $|w|$
That is, for all $k \geq 0$, $|w| = k$ and $w = w^R$ implies $S \rightsquigarrow^* w$.

Exercise: Fill in proof.

$S \rightsquigarrow 0110$

$S \rightarrow 0S0 \rightarrow 01S10 \rightarrow 0110$

Mutual Induction

Situation is more complicated with grammars that have multiple non-terminals.

See Section 5.3.2 of the notes for an example proof.

Closure Properties: Union

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Closure Properties: Union

$G_1 = (V_1, T, P_1, \underline{S_1})$ and $G_2 = (V_2, T, P_2, \underline{S_2})$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under union. $\mathbf{L_1, L_2}$ *CFLs implies* $\mathbf{L_1 \cup L_2}$ *is a CFL.*

$G = (V, T, P, S)$ s.t. $L(G) = L_1 \cup L_2$

$V = V_1 \cup V_2$

$P = \{S \to S_1 \mid S_2\} \cup P_1 \cup P_2$

$S \to \underline{S_1} \mid \underline{S_2}$

Closure Properties: Concatenation

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under concatenation. L_1, L_2 *CFLs implies* $L_1 \bullet L_2$ *is a CFL.*

$xy \in L_1 \cdot L_2$ iff
$x \in L_1$, $y \in L_2$

$\{S \to S_1 \cdot S_2\} \cup P_1 \cup P_2$

Closure Properties: Kleene Star

$G_1 = (V_1, T, P_1, S_1)$ and $G_2 = (V_2, T, P_2, S_2)$
Assumption: $V_1 \cap V_2 = \emptyset$, that is, non-terminals are not shared

Theorem

CFLs are closed under Kleene star. L_1 CFL implies L_1^* is a CFL.

$L = 0^*$

L_1^*

$S \to \epsilon \mid S0$

$S S_1$

$S \to (V \cup T)^*$

L^*

$L_1 \subseteq T^*$

Exercise

- Prove that every regular language is context-free using previous closure properties.
- Prove that language $\boldsymbol{L} = \{$Regular expressions over an alphabet $\boldsymbol{\Sigma}\}$ is context-free, but not regular.

$L \subset \{0, 1, +, *, (,)\}^*$

Closure Properties of CFLs continued

Theorem

CFLs are not closed under complement or intersection.

Theorem

If L_1 is a CFL and L_2 is regular then $L_1 \cap L_2$ is a CFL.

Canonical non-CFL

Theorem

$L = \{a^n b^n c^n \mid n \geq 0\}$ *is not context-free.*

Proof based on pumping lemma for CFLs. Technical and outside the scope of this class.

Language recognition for CFLs

Algorithmic question: Given CFG G and string $w \in \Sigma^*$ is $w \in L(G)$?

Language recognition for CFLs

Algorithmic question: Given CFG G and string $w \in \Sigma^*$ is $w \in L(G)$?

Later in course: algorithm for above problem that runs in $O(|w|^3)$ time for any fixed grammar G. Via dynamic programming.

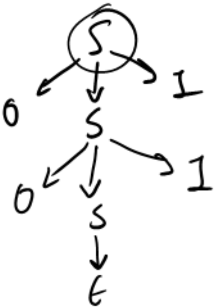
Hence parsing problem for programming languages is solvable. However cubic time algorithm is too slow! For this reason grammars for PLs are restricted even further to make parsing algorithm faster (essentially linear time) — see CS 421 and compiler courses.

Parse Trees or Derivation Trees

A tree to represent the derivation $S \rightsquigarrow^* w$.

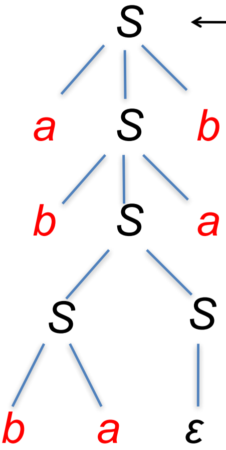
- Rooted tree with root labeled S
- Non-terminals at each internal node of tree
- Terminals at leaves
- Children of internal node indicate how non-terminal was expanded using a production rule

$S \to \epsilon \mid 0S1$



$S \rightsquigarrow 0011$

Example



A derivation tree for *abbaab*

(also called "parse tree")

$S \rightarrow aSb \mid bSa \mid SS \mid ab \mid ba \mid \varepsilon$

A corresponding derivation of *abbaab*

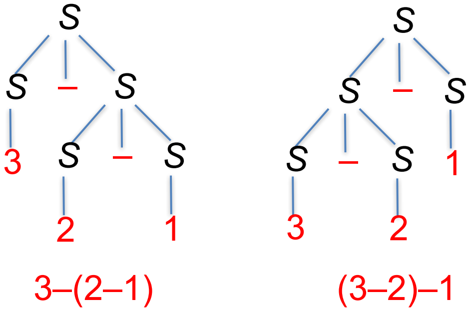
$S \rightarrow aSb \rightarrow abSab \rightarrow abSSab \rightarrow abbaSab \rightarrow abbaab$

Ambiguity in CFLs

Definition

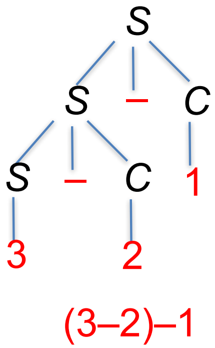
A CFG G is ambiguous if there is a string $w \in L(G)$ with two different parse trees. If there is no such string then G is unambiguous.

Example: $S \rightarrow S - S \mid 1 \mid 2 \mid 3$



Ambiguity in CFLs

- Original grammar: $S \to S - S \mid 1 \mid 2 \mid 3$
- Unambiguous grammar:
  $S \to S - C \mid 1 \mid 2 \mid 3$
  $C \to 1 \mid 2 \mid 3$



The grammar forces a parse corresponding to left-to-right evaluation.

Inherently ambiguous languages

Definition

A CFL L is inherently ambiguous if there is no unambiguous CFG G such that $L = L(G)$.

Inherently ambiguous languages

Definition

A CFL L is inherently ambiguous if there is no unambiguous CFG G such that $L = L(G)$.

- There exist inherently ambiguous CFLs.
  Example: $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$

Inherently ambiguous languages

Definition

A CFL L is inherently ambiguous if there is no unambiguous CFG G such that $L = L(G)$.

- There exist inherently ambiguous CFLs.
  Example: $L = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$
- Given a grammar G it is undecidable to check whether $L(G)$ is inherently ambiguous. No algorithm!

Normal Forms

Normal forms are a way to restrict form of production rules

Advantage: Simpler/more convenient algorithms and proofs

Normal Forms

Normal forms are a way to restrict form of production rules

Advantage: Simpler/more convenient algorithms and proofs

Two standard normal forms for CFGs

- Chomsky normal form
- Greibach normal form

Normal Forms

Chomsky Normal Form:

- Productions are all of the form $A \to BC$ or $A \to a$.
  If $\epsilon \in L$ then $S \to \epsilon$ is also allowed.
- Every CFG G can be converted into CNF form via an efficient algorithm
- Advantage: parse tree of constant degree.

$S \to \epsilon \mid 0S1$

Normal Forms

Chomsky Normal Form:

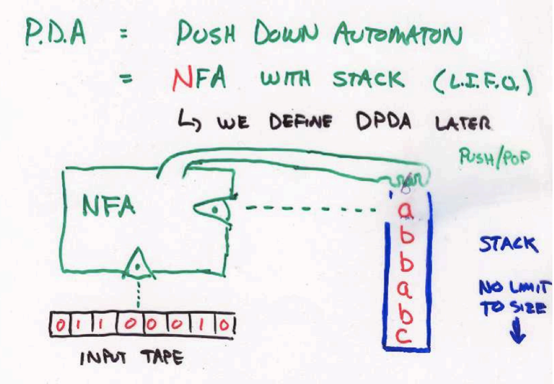
- Productions are all of the form $A \to BC$ or $A \to a$. If $\epsilon \in L$ then $S \to \epsilon$ is also allowed.
- Every CFG G can be converted into CNF form via an efficient algorithm
- Advantage: parse tree of constant degree.

Greiback Normal Form:

- Only productions of the form $A \to a\beta$ are allowed.
- All CFLs without ϵ have a grammar in GNF. Efficient algorithm.
- Advantage: Every derivation adds exactly one terminal.

Things to know: Pushdown Automata

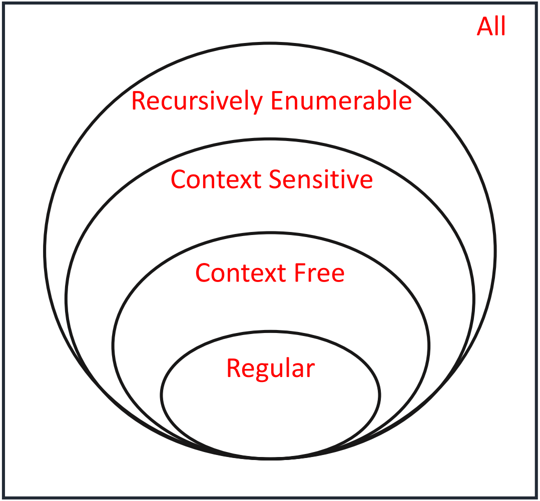
PDA: a NFA coupled with a stack



PDAs and CFGs are equivalent: both generate exactly CFLs. PDA is a machine-centric view of CFLs. Helps prove that the intersection of a CFL and a regular language is a CFL.

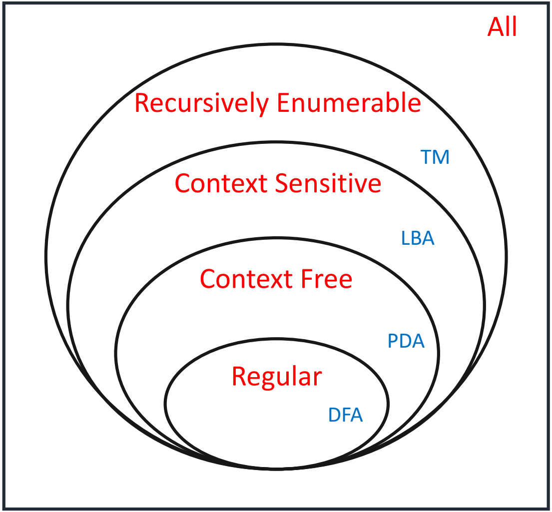
Language classes: Chomsky Hierarchy

Generative models for languages based on grammars.



Chomsky Hierarchy and Machines

For each class one can define a corresponding class of machines.



Chomsky Hierarchy

See Wikipedia article for more on Chomsky Hierarchy including the grammar rules for Context Sensitive Languages etc.
https://en.wikipedia.org/wiki/Chomsky_hierarchy