Formal Grammars

Motivation: Building a compiler

- A compiler is a large, complex software system.

- There are two approaches to building them.

- The Old Way:
  
  • For each programming language (C, Pascal, Turing, Java, Lisp, ...) a compiler is carefully built by a team of skilled experts.
  
  • This process is slow, error-prone and expensive. Can take years.
- The Modern Way:

  - Use a **compiler compiler**

  - This is a program that takes a description of a language as input, and produces a compiler for the language as output.

  - This is much faster and reliable.

  - A popular compiler compiler is YACC (**Yet Another Compiler Compiler**).
  (For details, type "man yacc" to Unix.)
Language Description

Question: How do we describe a programming language?

Answer: With a formal grammar, i.e., a set of rules describing the syntactically correct programs.

- Grammars can describe not only programming languages but also human languages (e.g., French, English).

- The theory of formal grammars was developed in linguistics (originally by Noam Chomsky), and adopted by computer science later.
Example

Grammar Rules:

\[ \langle s \rangle \rightarrow \langle NP \rangle \langle VP \rangle \]  \hspace{1cm} (1)

"A Sentence is a Noun Phrase followed by a Verb Phrase."

\[ \langle NP \rangle \rightarrow \langle \text{Noun} \rangle \mid \langle \text{adj} \rangle \langle \text{Noun} \rangle \]  \hspace{1cm} (2)

"A Noun Phrase is either a noun or an adjective followed by a noun."

\[ \langle VP \rangle \rightarrow \langle \text{Verb} \rangle \mid \langle \text{Verb} \rangle \langle \text{NP} \rangle \]  \hspace{1cm} (3)

"A Verb Phrase is either a verb or a verb followed by a noun phrase."

Vocabulary Rules:

\[ \langle \text{Noun} \rangle \rightarrow \text{Tarazon} \mid \text{Jane} \]

\[ \langle \text{Verb} \rangle \rightarrow \text{go} \mid \text{like} \mid \text{hit} \]

\[ \langle \text{adj} \rangle \rightarrow \text{strong} \mid \text{pretty} \]

\underline{non-terminal symbols} \hspace{5cm} \underline{terminal symbols}
Derivations

eg. 1. \[ s \rightarrow NP \ V \]  
\rightarrow Noun \ V  
\rightarrow Noun \ Verb  
\rightarrow Tarazan \ Verb  
\rightarrow Tarazan Go

eg. 2. \[ s \rightarrow NP \ V \]  
\rightarrow NP \ Verb \ NP  
\rightarrow adj \ Noun \ Verb \ NP  
\rightarrow adj \ Noun \ Verb \ adj \ Noun  
\rightarrow Pretty Jane Like Strong Tarazan
Parse Trees

eg. 1.

<s>
  <NP>
    <Noun>
    Tarazan
  <VP>
    <Verb>
    Go

eg. 2.

<s>
  <NP>
    <adj>
    Pretty
    <Noun>
    Jane
  <VP>
    <Verb>
    Like
  <NP>
    <adj>
    Strong
    <Noun>
    Tarazan
Recursive Grammars

eg. \( <NP> \rightarrow <Noun> \mid <adj> <NP> \)

Sample Derivation:

\[ <NP> \rightarrow <adj> <NP> \]
\[ \rightarrow <adj> <adj> <NP> \]
\[ \rightarrow <adj> <adj> <adj> <NP> \]
\[ \rightarrow <adj> <adj> <adj> <Noun> \]
\[ \rightarrow \text{Strong Strong Pretty Tarazon} \]

The Parse Tree:

```
  (NP)
 /   \
|     |
( NP)   ( NP)
 \
  |     |
(adj) ( adj) (Noun)
  |     |     |
Strong Strong Pretty Tarazon
```
- A **grammar** is a set of rules for describing a language.

- A **language** is a set of sentences.

- A **sentence** is a finite sequence of words.

- Grammars are classified according to the kinds of rules they use.

- All the grammars we have seen so far are **context free**.
Context-Free Grammars

A context-free grammar has four parts:

1. A set of **terminal** symbols.
   
   eg. Tarzan, Jane, pretty.
   
   These are the words of the language.

2. A set of **non-terminal** symbols.
   
   eg. \texttt{<s>, <NP>, <VP>, <adj>}
   
   These represent grammatical phrases.

3. A special **non-terminal** called the **starting symbol**, eg, \texttt{<s>}
   
   e.g. \( \langle NP \rangle \rightarrow \langle \text{Noun} \rangle \mid \langle \text{adj} \rangle \langle NP \rangle \)

   The left-hand side of a rule must be a single non-terminal symbol.

**Def:** A language is \underline{context-free} if every sentence can be derived from the starting symbol using the rules of a context-free grammar.
Context-Sensitive Grammars

- The production rules may have more than one symbol on the left-hand side.
- In particular, rules have the form
  \[ \alpha A \beta \rightarrow \alpha B_1 B_2 \ldots B_n \beta \]
- This means that the sequence \( \alpha A \beta \)
  can be rewritten as \( \alpha B_1 B_2 \ldots B_n \beta \).
- Equivalently, it means that \( A \) can be rewritten as \( B_1 B_2 \ldots B_n \), but only in the context of \( \alpha \) and \( \beta \).
- Context-sensitive grammars rarely arise in CS, & we shall not consider them further.