Formal Grammars

Motivation: Building a compiler

- A compiler is a large, complex software system.
- There are two appoaches to building them.
- The Old Way:
  - For each programming language (C, Pascal, Turing, Java, Lisp, ...) a compiler is carefully built by a team a skilled experts.
  - This process is slow, error-prone & expensive. Can take years.

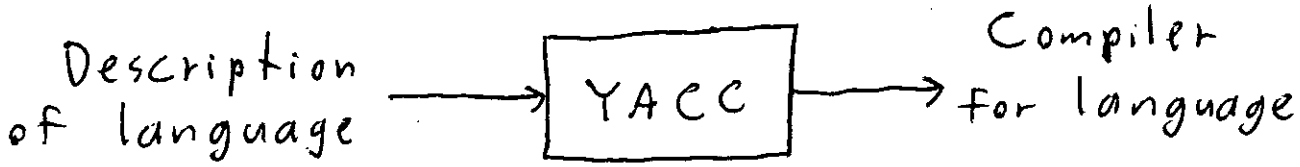
- The Modern Way:

  - 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, ie, a set of rules describing the syntactically correct programs.

- Grammars can describe not only programming languages, but also Human languages (eg, French, English).

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

Example

Grammar Rules:

<S> → <NP> <VP} ①

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

<NP> → <Noun> | <adj> <Noun> ②

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

<VP> → <Verb> | <Verb> <NP> ③

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

Vocabulary Rules:

<Noun> → Tarazon | Jane

<Verb> → go | like | hit

<adj> → strong | pretty

non-terminal symbols

terminal symbols

Derivations

eg. 1. <S> → <NP> <VP> by ①

→ <Noun> <VP> by ②

→ <Noun> <Verb> by ③

→ Tarazan <Verb>

→ Tarazan Go

eg. 2. <S> → <NP> <VP> by ①

→ <NP> <Verb> <NP> by ③

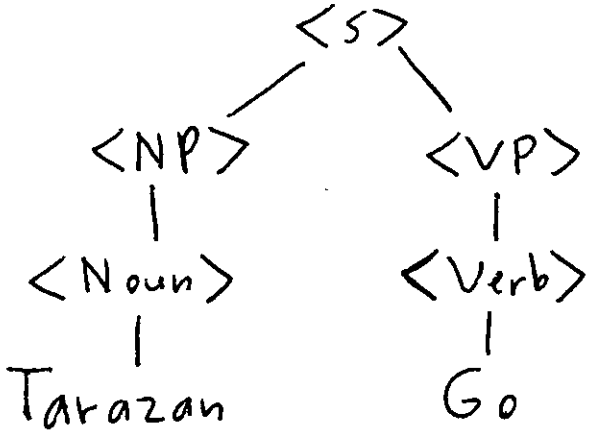
→ <adj> <Noun> <Verb> <NP> by ②

→ <adj> <Noun> <Verb> <adj> <Noun> ②

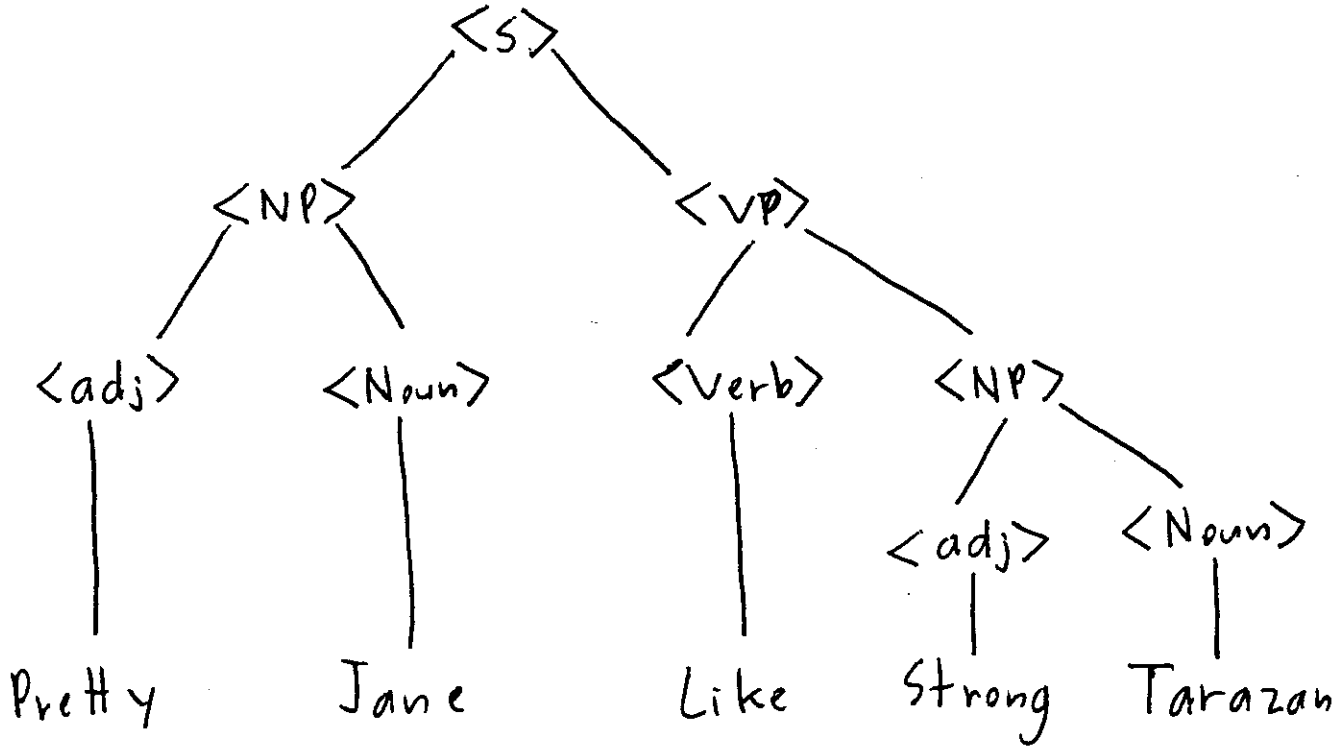
→ Pretty Jane Like Strong Tarazan

Parse Trees

eg. 1.



eg. 2.



Recursive Grammars

eg. <NP> → <Noun> | <adj> <NP>

Sample Derivation:

<NP> → <adj> <NP>

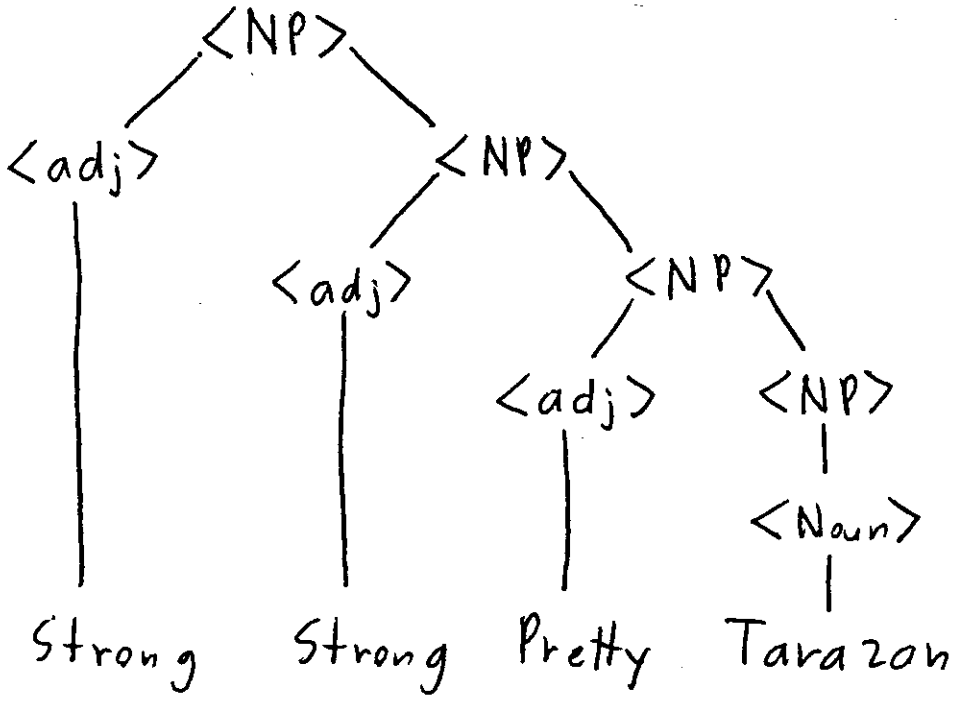
→ <adj> <adj> <NP>

→ <adj> <adj> <adj> <NP>

→ <adj> <adj> <adj> <Noun>

→ Strong Strong Pretty Tarazon

The Parse Tree:



Grammars and Languages

- 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. <S>, <NP>, <VP>, <adj>

These represent grammatical phrases.

③ A special non-terminal called the starting symbol. eg. <S>

④ A set of production rules.

eg. $\langle NP \rangle \longrightarrow \langle Noun \rangle \mid \langle adj \rangle \langle NP \rangle$

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

Def: A language is 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

$$\alpha A \beta \longrightarrow \alpha B_1 B_2 \cdots B_n \beta$$

- This means that the sequence $\alpha A \beta$ can be rewritten as $\alpha B_1 B_2 \cdots B_n \beta$.

- Equivalently, it means that A can be rewritten as $B_1 B_2 \cdots B_n$, but only in the context of α and β.

- Context sensitive grammars rarely arise in CS, & we shall not consider them further