Context-free languages and Grammars

KR Chowdhary
Professor & Head
Email: kr.chowdhary@acm.org
webpage:www.krchowdhary.com

Department of Computer Science and Engineering
MBM Engineering College, Jodhpur

September 26, 2011

Introduction

- Context-free languages(CFL) are more powerful than regular languages. Like, regular expressions are recognizers of regular languages, the context-free grammars (CFG) are generators of CFL.
- CFG is finite specification of rules to generate infinite context-free language.
- Regular languages are subset of CFL.
- CFLs and CFGs are fundamentals to computer science, because they help in describing the structure of programming languages.
- All the HLL are in the category of CFL. Though natural languages (NLs) are not CFL, but their analysis is possible only when they are treated as CFLs.
- Consider generating all the strings of regex $a^*(b^* + c^*)$. The approach can be:
  a. write character a zero or more times
  b. arbitrarily choose b or c and write it arbitrary times
  c. write c.
- if this string already exists in the list, ignore it, else keep the string. Running this method (algorithm) indefinitely, one can list all the strings (sentences) of languages specified by regex above.

Generating Language strings

- Let $L = L(a^*(b^* + c^*)$ is language corresponding to the regex. We can generate all the strings by an alternate method, using production/substitution rules:

  a. $S \rightarrow AMb$

  b. $A \rightarrow \varepsilon$

  c. $A \rightarrow aA$

  d. $M \rightarrow B$

  e. $M \rightarrow C$

  f. $B \rightarrow \varepsilon$

  g. $B \rightarrow bB$

  h. $C \rightarrow \varepsilon$

  i. $C \rightarrow \varepsilon$

  j. $C \rightarrow cC$

  The symbol $\rightarrow$ stand for "can be substituted by", and $\Rightarrow$ stand for "derives".

Consider generating the string $w = aaccb$ using these production rules. (The expression string, like $aacCb$ or AMb, during the derivation is *called sential form*).

$$
\begin{aligned}
S &\Rightarrow AMb \text{ ;by rule a}\\
&\Rightarrow aAMb \text{ ;by rule c}\\
&\Rightarrow aaAMb \text{ ;by rule c}\\
&\Rightarrow aaMb \text{ ;by rule b}\\
&\Rightarrow aaCb \text{ ;by rule e}\\
&\Rightarrow aacCb \text{ ;by rule j}\\
&\Rightarrow aaccCb \text{ ;by rule j}\\
&\Rightarrow aaccb \text{ ;by rule i}\\
&\therefore, aaccb \in L.
\end{aligned}
$$

Generating language strings

Let us try to generate the strings of language $L = \{a^n b^n | n \geq 0\}$ using similar rules. The rules this time are: $S \rightarrow aSb$, $S \rightarrow \varepsilon$. Consider deriving $w = aaabbb$.

$$
\begin{aligned}
w &\Rightarrow aSb;\ \text{apply first rule}\\
&\Rightarrow aaSbb;\ \text{apply first rule}\\
&\Rightarrow aaaSbbb;\ \text{apply first rule}\\
&\Rightarrow aaabbb;\ \text{apply second rule}
\end{aligned}
$$

Therefore, $w \Rightarrow^* aaabbb$, and the language is $L = \{a^n b^n | n \geq 0\}$. Now, there is time to define a the generator of these languages, the grammar. The context-free grammar G is defined as $G = \{V, \Sigma, S, P\}$, where

- V is finite set of variables symbols, appearing in the process of derivation
- Σ is set of terminal symbols (appearing in the final generated sentence), $V \cap \Sigma = \phi$
- S is start symbol
- P is set of production/ substitution rules of the form $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.
- Symbols in upper case in the begin of English alphabets are used as variable symbols (non-terminal symbols), i.e. A, B, C, D

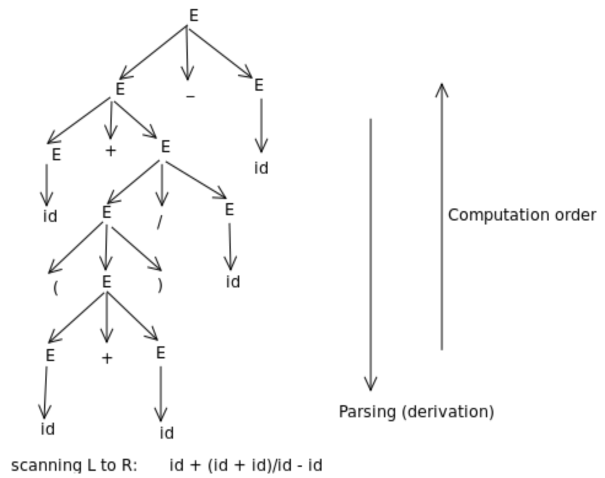
Derivations

- **Definition:**Context-free grammar: A context-free grammar is regular if productions are like:
  $A \rightarrow a,\ A \rightarrow aB,\ A \rightarrow \varepsilon\ \square$
  where, $a \in \Sigma$, and $A, B \in V$ (are non-terminals). For the regular expression $a^*(b^* + c^*)b$, a CFG was used in the previous slides to generate the regular language, which is generted by the regex also. This confirms the definition.
  Derivation: Let a derivation be: $\alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \cdots \Rightarrow_G \alpha_n$,
  then it can be written as $\alpha_1 \Rightarrow_G^* \alpha_n$
- In a derivation $\beta A \gamma \Rightarrow_G \beta \alpha \gamma$, the symbol A can be always substitued by α, if there is production like $A \rightarrow \alpha$, irrespective of presence of substrings β and α around the non-terminal sysmbol A. Language having this property is called context-free. The substrings β and α are clled context of variable A.
- The relation $\Rightarrow$ is *reflexice, anti-symmetric, and transitive*, hence it is an partial ordering relation.
- Language acceptability using CFG: $L = L(G) = \{w \in \Sigma | S \Rightarrow_G^* w\}$
- Two grammars G_1, G_2 are equal if the languages generated by them are same. $G_1 \equiv G_1 \Rightarrow L(G_1) \equiv L(G_2)$

Derivations

Example: Given grammar $G = \{V, \Sigma, S, P\}$, $\Sigma = \{+, -, *, /, (,), id\}$, $V = \{E\}$, $S = E$, and $P = \{E \rightarrow E + E | E - E | E/E | E * E | (E) | id\}$, find out the derivation and derivation tree for $id * (id + id) - id$.
The generating process is shown below using derivation as well as through **syntax** or derivation tree. The computation follows after the derivation is complete.

$$
\begin{aligned}
E &\Rightarrow E - E \Rightarrow E * E - E \\
&\Rightarrow id * E/E - E \Rightarrow id * (E)/E - E \\
&\Rightarrow id * (E + E)/E - E \Rightarrow id * (id + E)/E - E \\
&\Rightarrow id * (id + id)/E - E \\
&\Rightarrow id * (id + id)/id - E \\
&\Rightarrow id * (id + id)/id - id
\end{aligned}
$$

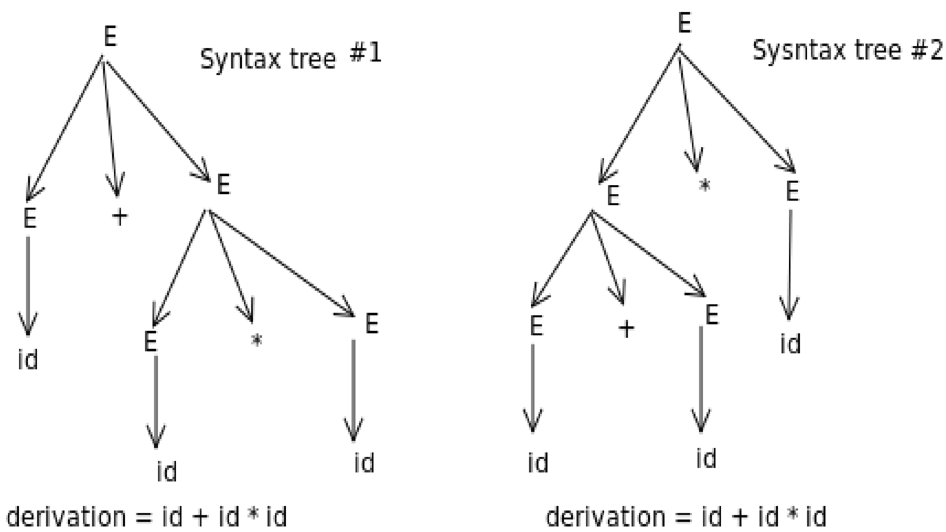


Derivations and ambiguity

- Compilers use derivation (syntax) trees to derive a given expression. If it succeed to derive give expression uisng the syntx rules, then the expression is syntatically correct, else wrong.
- During the derivation, e.g., $S \Rightarrow ABC$, we may start by first replacing left hand variables first (left hand dervation) or the right hand variable first (right hand derivation). In both the cases the end result is going to be the same. Only, the order of application of rules differ.
- If a langauge $L = (G)'s$ expression can be derived using two or more different derivation trees, then the correspondig grammar G is called ambiguous grammar.
- If a grammar has maximum n number of derivation trees, then the degree of ambiguity for this language as well as grammar is n.
- It is recursively unsolvable, whether an arbitrary grammar is ambiguous. Hence, there does not exist an algorithm to find out whether a given grammar is ambiguous.
- A grammar is unambiguos if every $w \in L(G)$ has a uique parse-tree,
- A grammar is called reduced, if, every **nonterminal** appears in some derivation.

Ambiguity

- Show that grammar for $id + id * id$ is ambiguous.



- The two derivation trees have different semantics: first calculates id+(id*id) while the second does it (id+id)*id, hence the grammar is ambiguous.

Removing ambiguity

- The general case of detection of ambiguity in a grammar is unsolvable. However, if it is found that the grammar is ambiguous, it can be made unambiguous by adding few more non-terminals in the grammar.

  Example: Given $\Sigma = \{(,),+,*,id\}$, $P = \{E \rightarrow E+E|E*E|(E)|id\}$, which is an ambiguous grammar, find out its equivalent unambiguous grammar.

  Solution: Let $V = \{E, T, F\}$, and $P = \{E \rightarrow T, T \rightarrow F, F \rightarrow id, E \rightarrow E+T, T \rightarrow T*F, F \rightarrow (E)\}$.

  Note that, you can derive a string is one way only.

$$\begin{aligned}
E &\Rightarrow E+T \\
&\Rightarrow T+T \\
&\Rightarrow F+T \Rightarrow id+T \\
&\Rightarrow id+T*F \Rightarrow id+F*F \\
&\Rightarrow id+id*F \Rightarrow id+id*id
\end{aligned}$$