Context Free Grammars and Languages

5DV037 — Fundamentals of Computer Science

Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Relevance

- Context-free grammars (CFGs) are the most important class of grammars in computer science.
- The main syntactic structure of virtually all modern programming languages is expressed using them.
- Modern parsers for programming languages are based upon them.
- Tools have been developed which generate parsers automatically from CFGs, and such tools are widely used.
- Many approaches to the modelling and understanding of natural language are also based upon context-free "backbones".
- In short, CFGs are a central notion in practical as well as theoretical computer science.

A Review of the Notion of a Grammar

Definition: A *(phrase-structure) grammar* is a four-tuple

$$G = (V, \Sigma, S, P)$$

in which

- V is a finite alphabet, called the *variables* or *nonterminal symbols*;
- Σ is a finite alphabet, called the set of *terminal symbols*;
- $S \in V$ is the *start symbol*;
- P is a finite subset of $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ called the set of *productions* or *rewrite rules*;
- $V \cap \Sigma = \emptyset$;

- The production $(w_1, w_2) \in P$ is typically written $w_1 \underset{G}{\rightarrow} w_2$, or just $w_1 \rightarrow w_2$ if the context G is clear.
- The meaning of $w_1 \rightarrow w_2$ is that w_1 may be replaced by w_2 in a string.
- Note that w_1 may be any nonempty string in this definition.

Context-Free Grammars

- In a *context-free grammar*, the left-hand side of each production must be a single nonterminal symbol.
  - Thus, the replacement is independent of the context in which the nonterminal occurs.

Definition: A *context-free grammar* or *CFG* is a four-tuple

$$G = (V, \Sigma, S, P)$$

in which

- V is a finite alphabet, called the *variables* or *nonterminal symbols*;
- Σ is a finite alphabet, called the set of *terminal symbols*;
- $S \in V$ is the *start symbol*;
- P is a finite subset of $V \times (V \cup \Sigma)^*$ called the set of *productions* or *rewrite rules*;
- $V \cap \Sigma = \emptyset$;

- Productions are thus of the form $A \to w$

for some $A \in V$ and $w \in (V \cup \Sigma)^*$.

Derivation in the Context of a CFG

Context: $G = (V, \Sigma, S, P)$ a CFG.

- Let $A \underset{G}{\rightarrow} w$, and let $\beta \in (V \cup \Sigma)^+$ be a string which contains A; *i.e.*, $\beta = \alpha_1 A \alpha_2$ for some $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$.
- A possible *single-step derivation* on w replaces A with w.
- Write $\alpha_1 A \alpha_2 \underset{G}{\Rightarrow} \alpha_1 w \alpha_2$ (or just $\alpha_1 A \alpha_2 \Rightarrow \alpha_1 w \alpha_2$).
- Note that many derivation steps may be possible on a given string.
- This process is thus inherently nondeterministic.
- Write $w \overset{*}{\underset{G}{\Rightarrow}} u$ (or just $w \overset{*}{\Rightarrow} u$) if $w = u$ or else there is a sequence

$$w = \alpha_0 \overset{*}{\underset{G}{\Rightarrow}} \alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_2 \ldots \overset{*}{\underset{G}{\Rightarrow}} \alpha_k = u$$

  called a *derivation* of u from w (for G).
- Write $w \overset{+}{\underset{G}{\Rightarrow}} u$ (or just $w \overset{+}{\Rightarrow} u$) if the derivation is at least one step long.
- The *language of G* is $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$.
- A language L is *context free* (or a *CFL*) if $L = \mathcal{L}(G)$ for some CFG G.
- The CFGs G_1 and G_2 are *equivalent* if $\mathcal{L}(G_1) = \mathcal{L}(G_2)$.

Degrees of Ambiguity for CFGs

- There are four possible levels of ambiguity with respect to derivations in a CFG $G = (V, \Sigma, S, P)$.
- First, these will be listed, and then an example of each will be presented.

Unique derivations: For each $\alpha \in \mathcal{L}(G)$, there is exactly one derivation for α.

Essentially unique derivations: The various derivations of each $\alpha \in \mathcal{L}(G)$ differ only in the order in which the variables are replaced.

- Unique *derivation tree*.

Non-unique derivations but repairable: There is some $\alpha \in \mathcal{L}(G)$ with at least two distinct derivation trees, but there is another CFG G' with $\mathcal{L}(G) = \mathcal{L}(G')$ for which each $\alpha \in \mathcal{L}(G')$ has a unique derivation tree.

Inherently non-unique derivations: For every CFG G' with $\mathcal{L}(G') = \mathcal{L}(G)$, there is some string $\alpha \in \mathcal{L}(G)$ which has at least two distinct derivation trees in G'.

An Example of Unique Derivation

Let $G = (V, \Sigma, S, P) \;\; = (\{S\}, \{a, b\}, S, \{S \to aSb \mid ab\}$

- It is easy to see that $\mathcal{L}(G) = \{a^n b^n \mid n \geq 1\}$.
- The string *aaabbb* has the unique derivation

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$$

  and hence is in $\mathcal{L}(G)$.

- In general, the string $a^k b^k$ has the unique derivation

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow \ldots \Rightarrow a^i Sb^i \ldots \Rightarrow a^{k-1} Sb^{k-1} \Rightarrow a^k b^k$$

- Thus, every string in $\mathcal{L}(G)$ has a unique derivation in G.
- This type of uniqueness is very rare in practice.

Inessential Non-Uniqueness in Derivation

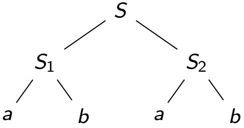
Let $G = (V, \Sigma, S, P) = (\{S, S_1, S_2\}, \{a, b\}, S,$
$\{S \to S_1 S_2,\ S_1 \to aS_1 b \mid ab,\ S_2 \to aS_2 b \mid ab\}.$

- Here $\mathcal{L}(G) = \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} \mid n_1, n_2 \geq 1\}$.
- In this case even the simple string *abab* has two distinct derivations:

  $S \Rightarrow S_1 S_2 \Rightarrow abS_2 \Rightarrow abab$

  $S \Rightarrow S_1 S_2 \Rightarrow S_1 ab \Rightarrow abab$

- However, there is only one tree-like representation of the derivation.



- Such a tree, called a *derivation tree*, provides more useful information than just a linear derivation using $\Rightarrow$.
- In this setting, it is only the *order* of replacements of the variables, and not the replacements themselves, which is not unique.
- This idea will be formalized shortly.

Inessential Non-uniqueness of derivations

- A CFG G is *ambiguous* if there is some $\alpha \in \mathcal{L}(G)$ which has two distinct derivation trees.

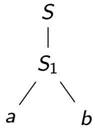
Example: Let $G = (V, \Sigma, S, P) = (\{S, S_1, S_2\}, \{a, b\}, S,$
$\{S \to S_1 S_2,\ S_1 \to aS_1 b \mid \lambda,\ S_2 \to aS_2 b \mid \lambda\})$.

- Here $\mathcal{L}(G) = \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} \mid n_1, n_2 \geq 0\}$.
- For any $k > 0$, the string $a^k b^k$ has two distinct derivations.
- Here are the two derivations for ab, represented as trees:

- This non-uniqueness issue may easily be repaired.

A Repair of the Non-Uniqueness Example

- The original grammar
  $G = (V, \Sigma, S, P) = (\{S, S_1, S_2\}, \{a, b\}, S,$
  $\{S \to S_1S_2,\ S_1 \to aS_1b \mid \lambda,\ S_2 \to aS_2b \mid \lambda\}.$
- The repaired grammar:
  $G' = (V, \Sigma, S, P') = (\{S, S_1, S_2\}, \{a, b\}, S,$
  $\{S \to \lambda \mid S_1 \mid S_1S_2,\ S_1 \to aS_1b \mid ab,\ S_2 \to aS_2b \mid ab\}.$
- The only derivation of *ab*:



- Unfortunately, it can be shown that there is no algorithm which takes as input an arbitrary CFG and decides whether or not it is ambiguous, much less construct a CFG which is equivalent.

Inherent Ambiguity

- A CFG $G = (V, \Sigma, S, P)$ is *inherently ambiguous* if for every CFG G' with $\mathcal{L}(G') = \mathcal{L}(G)$ is ambiguous.
- A CFL L is *inherently ambiguous* if every CFG G with $\mathcal{L}(G) = L$ is ambiguous.
- Thus, while ambiguity is a property of a grammar, inherent ambiguity is a property of a language and not of a specific grammar.
- Establishing that a CFL is inherently ambiguous is nontrivial.
- Here is a well-known example, presented without proof:

$$\{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

- Do important inherently ambiguous CFLs exist in practice?
- It can be proven that there is no algorithm to decide whether or not a CFG is inherently ambiguous.

A More Formal Presentation of Derivation Trees

Context: A CFG $G = (V, \Sigma, S, P)$.

- A *partial derivation tree* (or *(partial) parse tree*) for G with root $A \in V$ is a rooted tree with ordered subtrees such that
  - The root is labelled A.
  - Interior vertices are labelled with members of V.
  - Leaf vertices are labelled by members of $V \cup \Sigma \cup \{\lambda\}$.
  - If interior vertex x has label B with children labelled $c_1 \ldots c_k$ from left to right, then $B \to c_1 \ldots c_k \in P$.
    - Particularly, a leaf labelled λ can have no siblings.
- The *yield* (or *frontier*) of a partial derivation tree is the concatenation of leaf labels, read from left to right.

Observation: Let $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. Then $A \underset{G}{\to} \alpha$ iff there is a partial derivation tree for G with root A and frontier α. $\square$

- A partial derivation tree T with root S and yield $\alpha \in \Sigma^*$ is called a *derivation tree* for α.

Leftmost Derivations

- There is a natural correspondence between derivations which always replace the leftmost variable first and parse trees.
- Let $G = (V, \Sigma, S, P)$ be a CFG with $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. The derivation

$$A \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \alpha_2 \ldots \alpha_i \Rightarrow \alpha_{i+1} \ldots \alpha_n = \alpha$$

  is a *leftmost* derivation of α from A if in each step $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ the leftmost variable in the string α_i is replaced.
- A *rightmost derivation* is defined analogously.

Leftmost and Rightmost Derivation Illustrated

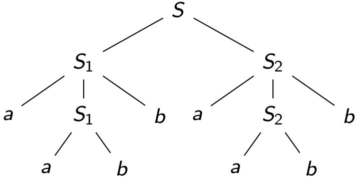
Example: $G' = (V, \Sigma, S, P') = (\{S, S_1, S_2\}, \{a, b\}, S,$
$\{S \to \lambda \mid S_1 \mid S_1S_2,\ S_1 \to aS_1b \mid ab,\ S_2 \to aS_2b \mid ab\}.$

- Here are the leftmost and rightmost derivations for *aabbaabb*:

  $S \Rightarrow S_1S_2 \Rightarrow aS_1bS_2 \Rightarrow aabbS_2 \Rightarrow aabbaS_2b \Rightarrow aabbaabb$

  $S \Rightarrow S_1S_2 \Rightarrow S_1aS_2b \Rightarrow S_1aabb \Rightarrow aaS_1bbaabb \Rightarrow aabbaabb$

- And here is the common derivation tree:



A Practical Example — If-Then-Else Ambiguity

- Consider the grammar with the following productions:

$$\langle \mathit{stmt} \rangle \rightarrow \langle \mathit{if\_stmt} \rangle \mid \langle \mathit{nif\_stmt} \rangle$$
$$\langle \mathit{if\_stmt} \rangle \rightarrow \texttt{if}\langle \mathit{cond} \rangle\texttt{then}\langle \mathit{stmt} \rangle \mid \texttt{if}\langle \mathit{cond} \rangle\texttt{then}\langle \mathit{stmt} \rangle\texttt{else}\langle \mathit{stmt} \rangle$$
$$\langle \mathit{nif\_stmt} \rangle \rightarrow s_1 \mid s_2 \mid s_3 \mid (\langle \mathit{stmt} \rangle)$$
$$\langle \mathit{cond} \rangle \rightarrow c_1 \mid c_2 \mid c_3$$

- The bracketed names are nonterminals, with $\langle \mathit{stmt} \rangle$ the start symbol.
- The terminals are $\{\texttt{if}, \texttt{then}, \texttt{else}, s_1, s_2, s_3, c_1, c_2, c_3, (,)\}$.
- The statement

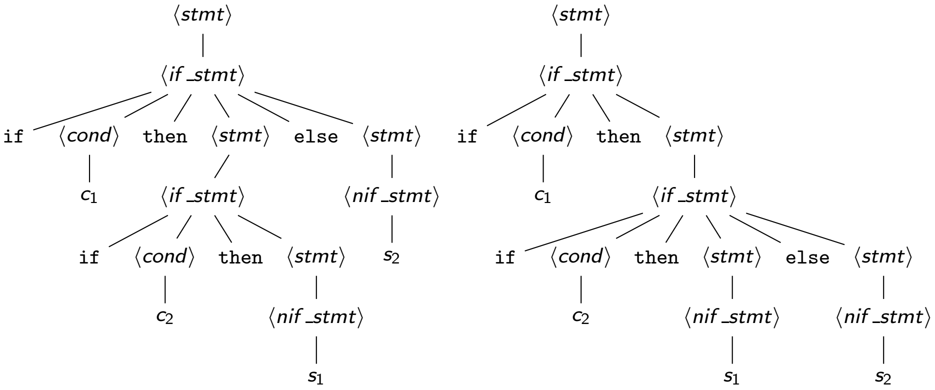
$$\texttt{if } c_1 \texttt{ then if } c_2 \texttt{ then } s_1 \texttt{ else } s_2$$

  has two parses, which corresponding to two distinct meanings, indicated by indentation:

$\texttt{if } c_1 \texttt{ then if } c_2 \texttt{ then } s_1$ $\quad\quad \texttt{else } s_2$	$\texttt{if } c_1 \texttt{ then}$ $\quad\quad \texttt{if } c_2 \texttt{ then } s_1 \texttt{ else } s_2$

The Two Derivation Trees for If-Then-Else Ambiguity

- The corresponding derivation trees:



- In the "correct" tree, the meaning of the statement is recaptured by evaluating subtrees in a bottom-up fashion.
- The tree to the right recaptures the usual convention.
  - Else-part associated with nearest then-part.

Resolution of If-Then-Else-Ambiguity

- Here is the repair of the grammar:

$$
\begin{array}{lcl}
\langle \mathit{stmt} \rangle & \rightarrow & \cancel{\langle \text{if\_stmt} \rangle} \;\; \langle \mathit{if\_then\_stmt} \rangle \mid \langle \mathit{if\_then\_else\_stmt} \rangle \\
 & & \quad \mid \langle \mathit{nif\_stmt} \rangle \\
\cancel{\langle \text{if\_stmt} \rangle} & \rightarrow & \texttt{if} \langle \mathit{cond} \rangle \texttt{then} \langle \mathit{stmt} \rangle \\
 & & \mid \texttt{if} \langle \mathit{cond} \rangle \texttt{then} \langle \mathit{stmt} \rangle \texttt{else} \langle \mathit{stmt} \rangle \\
\langle \mathit{if\_then\_stmt} \rangle & \rightarrow & \texttt{if} \langle \mathit{cond} \rangle \texttt{then} \langle \mathit{stmt} \rangle \\
\langle \mathit{if\_then\_else\_stmt} \rangle & \rightarrow & \texttt{if} \langle \mathit{cond} \rangle \texttt{then} \langle \mathit{nif\_stmt} \rangle \texttt{else} \langle \mathit{stmt} \rangle \\
\langle \mathit{nif\_stmt} \rangle & \rightarrow & s_1 \mid s_2 \mid s_3 \mid (\langle \mathit{stmt} \rangle) \\
\langle \mathit{cond} \rangle & \rightarrow & c_1 \mid c_2 \mid c_3
\end{array}
$$

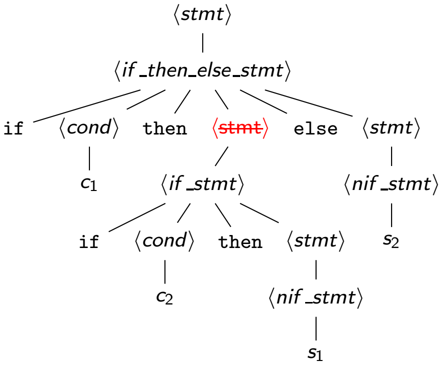
- Note in particular that $\langle \mathit{if\_stmt} \rangle$ has been replaced with
  - $\langle \mathit{if\_then\_stmt} \rangle$ and
  - $\langle \mathit{if\_then\_else\_stmt} \rangle$.
- An `if` statement in parentheses is "protected".

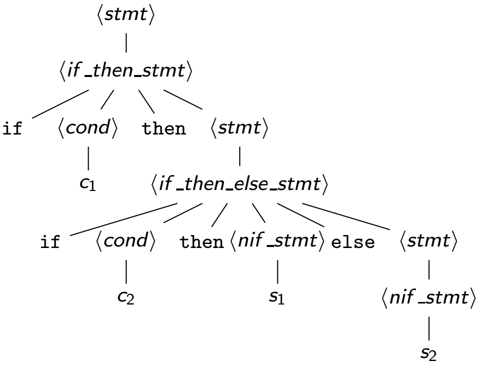
Parse Tree for the Repaired If-Then-Else Example

- The statement to be parsed is:

$$\texttt{if } c_1 \texttt{ then if } c_2 \texttt{ then } s_1 \texttt{ else } s_2$$

- To the right is the the unique pare tree in the repaired grammar.
- To the left is the old parse tree which is blocked by this new grammar.
- The one to the right is similar to the second one in the original grammar.



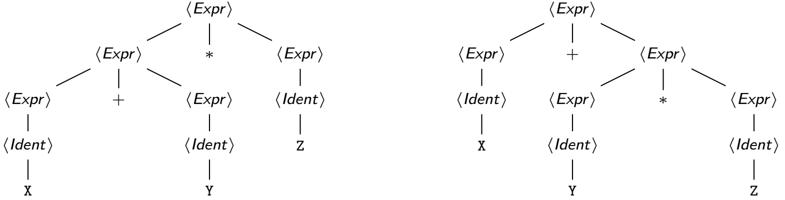


Another Practical Example – Precedence of Operations

- Here is a simple grammar for arithmetic expressions:
  Nonterminals: $\{\langle \textit{Expr} \rangle, \langle \textit{Ident} \rangle\}$.
  Terminals: $\{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \mathtt{(}, \mathtt{)}, \mathtt{+}, \mathtt{*}\}$.
  Start symbol: $\langle \text{Expr} \rangle$
  Productions:

$$\begin{array}{rcl} \langle \textit{Ident} \rangle & \rightarrow & \mathtt{A} \mid \mathtt{B} \mid \ \ldots \ \mid \mathtt{Y} \mid \mathtt{Z} \\ \langle \textit{Expr} \rangle & \rightarrow & \langle \textit{Expr} \rangle + \langle \textit{Expr} \rangle \mid \langle \textit{Expr} \rangle * \langle \textit{Expr} \rangle \mid (\langle \textit{Expr} \rangle) \mid \langle \textit{Ident} \rangle \end{array}$$

- The expression X+Y*Z has two parse trees:



Repair of the Operator-Precedence Problem

- Here is the repair using factors and terms:

Productions:

$$\begin{aligned}
\langle \mathit{Ident} \rangle &\rightarrow \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{Y} \mid \mathtt{Z} \\
\langle \mathit{Expr} \rangle &\rightarrow \langle \mathit{Expr} \rangle + \langle \mathit{Term} \rangle \mid \langle \mathit{Term} \rangle \\
\langle \mathit{Term} \rangle &\rightarrow \langle \mathit{Term} \rangle * \langle \mathit{Factor} \rangle \mid \langle \mathit{Factor} \rangle \\
\langle \mathit{Factor} \rangle &\rightarrow (\langle \mathit{Expr} \rangle) \mid \langle \mathit{Ident} \rangle
\end{aligned}$$

- The unique parse tree for `X+Y*Z`: