Lecture 9: Context-Free Languages

Context-free grammar

- is a *language generator* that is more powerful than regular expressions.
- important for parsing programs

Example:
$\Sigma = \{a, b\}$
The language L represented by $a(a^* \cup b^*)b$ can be generated by the following set of grammar rules.

$S \to aMb$	S begins with a and ends with b and has a middle part M.
$M \to A$	The middle part can be a string A, or
$M \to B$	a string B.
$A \to e$	A string A can be the empty string, or
$A \to aA$	any number of a's.
$B \to e$	A string B can be the empty string, or
$B \to bB$	any number of b's.

A **context-free grammar** G is a 4-tuple (V, Σ, R, S) where

- V is an **alphabet** (containing nonterminals and terminals),
- $\Sigma (\subseteq V)$ is a nonempty set of **terminals** (elements of $V - \Sigma$ are the **nonterminals**),
- R is a nonempty, finite set of **rules**, with $R \subseteq (V - \Sigma) \times V^*$.
- S is the **start symbol** $(\in V - \Sigma)$.

Example:
$G = (V, \Sigma, R, S)$
where
$V = \{a, b, M, A, B\}$
$\Sigma = \{a, b\}$
$R = \{(S, aMb), (M, A), (M, B), (A, e), (A, aA), (B, e), (B, bB)\}$

- instead of writing (A, u), we will simply write $A \rightarrow u$.

Note: in general, a grammar (not necessarily a CFG) can have rules that replace any string of terminals or nonterminals by any other string of terminals or nonterminals. For example, $aB \rightarrow bA$ could be a rule. CFG restricts the rules to be $R \subseteq (V - \Sigma) \times V^*$, which means the left-hand side of each rule must be a single nonterminal.

Derivation

- If $A \to u$ and $x, y \in V^*$, we write $xAy \Rightarrow xuy$
- $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_n$ is called a **derivation** in G of w_n from w_0, where n is the length of the derivation.
- If $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_n$, we write $w_0 \Rightarrow^* w_n$
- Example: $S \Rightarrow aMb \Rightarrow aAb \Rightarrow aaAb \Rightarrow aaaAb \Rightarrow aaab$. So, $S \Rightarrow^* aaab$.
- This type of grammars is called context-free because the applications of the rules do not depend on the context of the left-hand side nonterminal.
- By definition, $u \Rightarrow^* u$ for any $u \in V^*$.
- The **language generated** by G, denoted $L(G)$, is

$$\{w \in \Sigma^* : S \Rightarrow^* w\}.$$

- A language is said to be a **context-free language** if it can be generated by a context-free grammar.

Leftmost derivation in CFGs:

The nonterminal symbol replaced at each step is the leftmost one.

Example:

$S \to e$, $S \to SS$, $S \to (S)$.

- Leftmost derivation:

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()()$$

- The following is not a non-leftmost derivation.

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ()(S) \Rightarrow ()()$$

More examples of CFLs

Example:

$G = (V, \Sigma, R, S)$
where
$V = \{a, b, S\}$
$\Sigma = \{a, b\}$
$R = \{S \to aSb, S \to e\}$

- $S \Rightarrow e$.
  $S \Rightarrow aSb \Rightarrow ab$.
  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.
  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$.
  $\cdots$

- $L(G) = \{a^n b^n : n \geq 0\}$

- Since $\{a^n b^n : n \geq 0\}$ can be generated by a context-free grammar, it is a context-free language.

- Recall $L(G)$ is not regular. Context-free grammars can describe languages that are not regular.

More examples

Example:

> L is the set of all strings of balanced left and right parentheses.
>
> E.g.
> $() \in L$.
> $()(()()) \in L$.
> $()(() \notin L$.

Is L regular? No. Prove it!

Is L context-free? Yes.

The following CFG generates it.
$G = (V, \Sigma, R, S)$ where

$$
\begin{aligned}
V &= \{S, (,)\} \\
\Sigma &= \{(,)\} \\
R &= \{S \to e, \\
&\qquad S \to SS, \\
&\qquad S \to (S)\}.
\end{aligned}
$$

More examples

Write a context-free grammar for the following language:

$$L = \{w \in \{a, b\}^* : w \text{ has the same number of } a \text{ and } b\}$$

$S \to e$
$S \to aB$
$S \to bA$
$A \to a$
$A \to aS$
$A \to bAA$
$B \to b$
$B \to bS$
$B \to aBB$

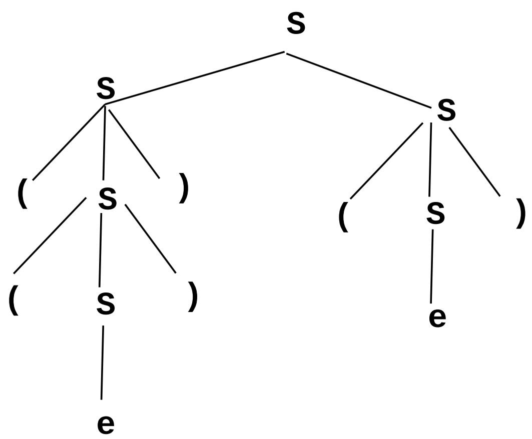
S gives a string that has the same number of a's and b's.

A gives a string that has one more a's than it has b's.

B gives a string that has one more b's than it has a's.

Parse trees – Pictorial representation of derivations.

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$



- $(())()$ is called the **yield** of the parse tree ($\in \Sigma^*$).
- The topmost node is called the **root**.
- The nodes at the bottom are called the **leaves** (labelled by terminals or e).
- internal nodes are nonterminals.

Formal inductive definition of parse trees

Given $G = (V, \Sigma, R, S)$.

1. If $a \in \Sigma$, then

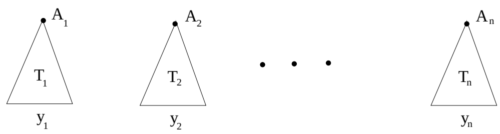
   is a parse tree.

2. If $A \to e$ is a rule in R, then

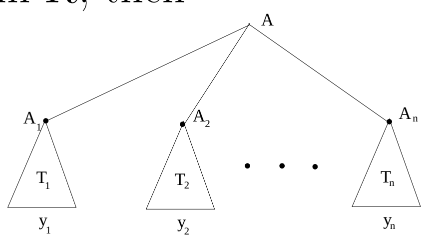
   

   is a parse tree.

3. If

   are parse trees rooted at A_i with yields y_i, and $A \to A_1 A_2 \ldots A_n$ is in R, then

   is a parse tree rooted at A with yield $y_1 y_2 \ldots y_n$.

4. Nothing else is a parse tree.

Parse Tree

A string may have more than one derivations that correspond to the same parse tree.

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$$
$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$

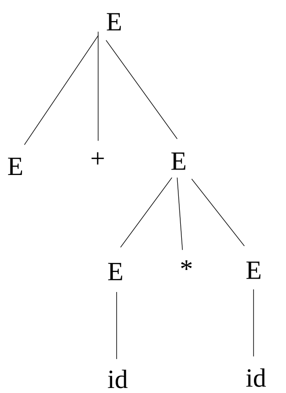
Each parse tree has exactly one leftmost derivation (it is unique because the leftmost nonterminal that is replaced and its replacement are unique).

Ambiguity

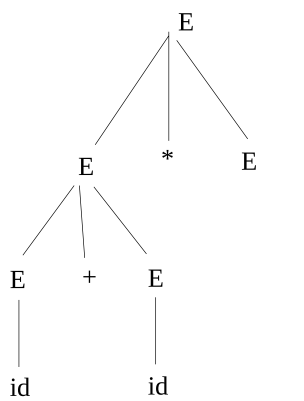
A CFG G is said to be **ambiguous** if there is a string in $L(G)$ that has two or more distinct parse trees.

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E * E \\
E &\rightarrow (E) \\
E &\rightarrow \texttt{id}
\end{aligned}
$$

Example:
The string $\texttt{id} + \texttt{id} * \texttt{id}$ has two distinct parse trees shown below.



* takes precedence over +



+ takes precedence over *

An unambiguous CFG

The following is an unambiguous CFG.

$$V = \{+, *, (,), \texttt{id}, T, F, E\}$$

$$\Sigma = \{+, *, (,), \texttt{id}\}$$

$$\begin{aligned} R = \{E &\to E + T, && \text{(R1)} \\ E &\to T, && \text{(R2)} \\ T &\to T * F, && \text{(R3)} \\ T &\to F, && \text{(R4)} \\ F &\to (E), && \text{(R5)} \\ F &\to \texttt{id}\} && \text{(R6)} \end{aligned}$$

E denotes Expression
T denotes Term
F denotes Factor.

Example: what is the unique parse tree for the expression $(\texttt{id} + \texttt{id}) * \texttt{id} + \texttt{id}$?

Parsing

- Parsing a string with respect to a grammar refers to assigning a parse tree (a.k.a. syntax tree) to the given string (so as to understand the structure of the string).

- Some grammars are ambiguous; e.g., English grammars; many English sentences have more than one parse tree and more than one meaning (e.g., "the lady hit the man with an umbrella", "he gave her cat food".)

- It is desirable to have unambiguous grammars for programming languages. In fact, for a programming language like C or C++, most of its syntax can be specified by an unambiguous grammar, but a compiler still often needs to deal with special cases. (e.g. `x * y;` in C. In fact, C is not a CFL in the first place, strictly speaking.) Parsing is a major topic in COMP3031.

- It is not always possible to "disambiguate" an ambiguous CFG. A CFL with the property that all CFGs that generate it are ambiguous is called *inherently ambiguous*.

- The most famous inherently ambiguous CFL is

$$\{a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$$

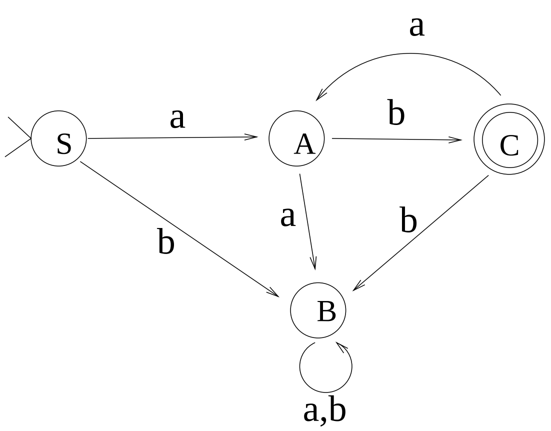
Regular languages are context-free

Theorem: All regular languages are context-free.

Proof: (by construction) Given an FA $M = (K, \Sigma, \delta, s, F)$. Construct a CFG $G = (V, \Sigma, R, S)$ such that $L(G) = L(M)$. Let

$$V = K \cup \Sigma$$
$$S = s$$
$$R = \{q \to ap : \delta(q, a) = p\} \cup \{q \to e : q \in F\}$$

Example:



$S \to aA$, $S \to bB$, $A \to aB$, $A \to bC$, $B \to aB$, $B \to bB$, $C \to aA$, $C \to bB$, $C \to e$.

We need to prove $L(G) = L(M)$.

First, we know that $e \in L(M)$ iff $s \in F$. From the construction, we have the rule $S \to e$ iff $s \in F$. Therefore, $e \in L(M)$ iff $S \Rightarrow_G e$.

Now, we consider the non-empty strings $w = \sigma_1 \cdots \sigma_n$.

$w \in L(M)$ iff there exists a computation sequence $(s, \sigma_1 \cdots \sigma_n) \vdash_M (q_1, \sigma_2 \cdots \sigma_n) \vdash_M \cdots \vdash_M (q_n, e)$ and $q_n \in F$.

Note that $(s, \sigma_1 \cdots \sigma_n) \vdash_M (q_1, \sigma_2 \cdots \sigma_n)$ iff $\delta(s, \sigma_1) = q_1$. From the construction, there is a rule $S \to \sigma_1 q_1$. Therefore, we can derive $S \Rightarrow_G \sigma_1 q_1$.

Similarly, we have the following derivation:
$S \Rightarrow_G \sigma_1 q_1 \Rightarrow_G \sigma_1 \sigma_2 q_2 \cdots \Rightarrow_G \sigma_1 \cdots \sigma_n q_n \Rightarrow_G \sigma_1 \cdots \sigma_n$ (the last step is because we have $q_n \to e$ since $q_n \in F$).

Therefore, $w \in L(M)$ iff $S \Rightarrow_G^* w$.