

Context-Free Languages (Read Ch. 5 in Linz Book)

Regular languages:

- keywords in a programming language
- names of identifiers
- integers
- all misc symbols: = ;

Not Regular languages:

- $\{a^n cb^n | n > 0\}$
- expressions - $((a + b) - c)$
- block structures ($\{\}$ in Java/C++ and begin ... end in pascal)

Definition: A grammar $G=(V,T,S,P)$ is context-free if all productions are of the form

$$A \rightarrow x$$

Where $A \in V$ and $x \in (V \cup T)^*$.

Definition: L is a context-free language (CFL) iff \exists context-free grammar (CFG) G s.t. $L=L(G)$.

Example: $G=(\{S\},\{a,b\},S,P)$

$$S \rightarrow aSb \mid ab$$

Derivation of aaabbb:

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

$L(G) =$

Example: $G = (\{S\}, \{a, b\}, S, P)$

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$$

Derivation of ababa:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow ababa$$

$$\Sigma = \{a, b\}, L(G) =$$

Example: $G = (\{S, A, B\}, \{a, b, c\}, S, P)$

$$\begin{aligned} S &\rightarrow AcB \\ A &\rightarrow aAa \mid \lambda \\ B &\rightarrow Bbb \mid \lambda \end{aligned}$$

$$L(G) =$$

Derivations of aacbb:

1. $S \Rightarrow \underline{A}cB \Rightarrow a\underline{A}cB \Rightarrow aac\underline{B} \Rightarrow aac\underline{B}bb \Rightarrow aacbb$
2. $S \Rightarrow Ac\underline{B} \Rightarrow Ac\underline{B}bb \Rightarrow \underline{A}cbb \Rightarrow a\underline{A}cbb \Rightarrow aacbb$

Note: Next variable to be replaced is underlined.

Definition: Leftmost derivation - in each step of a derivation, replace the leftmost variable. (see derivation 1 above).

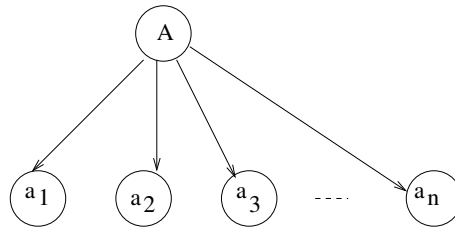
Definition: Rightmost derivation - in each step of a derivation, replace the rightmost variable. (see derivation 2 above).

Derivation Trees (also known as “parse trees”)

A derivation tree represents a derivation but does not show the order productions were applied.

A derivation tree for $G = (V, T, S, P)$:

- root is labeled S
- leaves labeled x, where $x \in T \cup \{\lambda\}$
- nonleaf vertices labeled A, $A \in V$
- For rule $A \rightarrow a_1 a_2 a_3 \dots a_n$, where $A \in V$, $a_i \in (T \cup V \cup \{\lambda\})$,



Example: $G = (\{S, A, B\}, \{a, b, c\}, S, P)$

$$\begin{aligned}
 S &\rightarrow AcB \\
 A &\rightarrow aAa \mid \lambda \\
 B &\rightarrow Bbb \mid \lambda
 \end{aligned}$$

Definitions Partial derivation tree - subtree of derivation tree.

If partial derivation tree has root S then it represents a sentential form.

Leaves from left to right in a derivation tree form the *yield* of the tree.

Yield (w) of derivation tree is such that $w \in L(G)$.

The yield for the example above is

Example of partial derivation tree that has root S:

The yield of this example is _____ which is a sentential form.

Example of partial derivation tree that does not have root S:

Membership Given CFG G and string $w \in \Sigma^*$, is $w \in L(G)$?

If we can find a derivation of w , then we would know that w is in $L(G)$.

Motivation

G is grammar for Java
 w is Java program.
 Is w syntactically correct?

Example

$G = (\{S\}, \{a, b\}, S, P)$, $P =$

$$S \rightarrow SS \mid aSa \mid b \mid \lambda$$

$L_1 = L(G) =$

Is $abbab \in L(G)$?

Exhaustive Search Algorithm

For all $i=1,2,3,\dots$

Examine all sentential forms yielded by i substitutions

Example: Is $abbab \in L(G)$?

Theorem If CFG G does not contain rules of the form

$$\begin{aligned} A &\rightarrow \lambda \\ A &\rightarrow B \end{aligned}$$

where $A, B \in V$, then we can determine if $w \in L(G)$ or if $w \notin L(G)$.

• **Proof:** Consider

1. length of sentential forms
2. number of terminal symbols in a sentential form

Example: Let $L_2 = L_1 - \{\lambda\}$. $L_2 = L(G)$ where G is:

$$S \rightarrow SS \mid aa \mid aSa \mid b$$

Show $baaba \notin L(G)$.

- $i=1$
1. $S \Rightarrow SS$
 2. $S \Rightarrow aSa$
 3. $S \Rightarrow aa$
 4. $S \Rightarrow b$

- $i=2$
1. $S \Rightarrow SS \Rightarrow SSS$
 2. $S \Rightarrow SS \Rightarrow aSaS$
 3. $S \Rightarrow SS \Rightarrow aaS$
 4. $S \Rightarrow SS \Rightarrow bS$
 5. $S \Rightarrow aSa \Rightarrow aSSa$
 6. $S \Rightarrow aSa \Rightarrow aaSaa$
 7. $S \Rightarrow aSa \Rightarrow aaaa$
 8. $S \Rightarrow aSa \Rightarrow aba$

Definition Simple grammar (or s-grammar) has all productions of the form:

$$A \rightarrow ax$$

where $A \in V$, $a \in T$, and $x \in V^*$ AND any pair (A, a) can occur in at most one rule.

Ambiguity

Definition: A CFG G is ambiguous if \exists some $w \in L(G)$ which has two distinct derivation trees.

Example Expression grammar

$G = (\{E, I\}, \{a, b, +, *, (,)\}, E, P), P =$

$$\begin{aligned} E &\rightarrow E+E \mid E * E \mid (E) \mid I \\ I &\rightarrow a \mid b \end{aligned}$$

Derivation of $a+b*a$ is:

$$E \Rightarrow \underline{E}+E \Rightarrow \underline{I}+E \Rightarrow a+\underline{E} \Rightarrow a+\underline{E}*E \Rightarrow a+\underline{I}*E \Rightarrow a+b*\underline{E} \Rightarrow a+b*\underline{I} \Rightarrow a+b*a$$

Corresponding derivation tree is:

Another derivation of $a+b*a$ is:

$$E \Rightarrow \underline{E}*E \Rightarrow \underline{E}+E * E \Rightarrow \underline{I}+E * E \Rightarrow a+\underline{E}*E \Rightarrow a+\underline{I}*E \Rightarrow a+b*\underline{E} \Rightarrow a+b*\underline{I} \Rightarrow a+b*a$$

Corresponding derivation tree is:

Rewrite the grammar as an unambiguous grammar. (with meaning that multiplication has higher precedence than addition)

$$\begin{aligned} E &\rightarrow E+T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow I \mid (E) \\ I &\rightarrow a \mid b \end{aligned}$$

There is only one derivation tree for $a+b*a$:

Definition If L is CFL and G is an unambiguous CFG s.t. $L=L(G)$, then L is unambiguous.

Backus-Naur Form of a grammar:

- Nonterminals are enclosed in brackets $\langle \rangle$
- For “ \rightarrow ” use instead “ $::=$ ”

Sample C++ Program:

```
main ()
{
    int a;    int b;    int sum;
    a = 40;   b = 6;    sum = a + b;
    cout << "sum is " << sum << endl;
}
```

“Attempt” to write a CFG for C++ in BNF (Note: $\langle \text{program} \rangle$ is start symbol of grammar.)

```
 $\langle \text{program} \rangle ::= \text{main } () \langle \text{block} \rangle$ 
 $\langle \text{block} \rangle ::= \{ \langle \text{stmt-list} \rangle \}$ 
 $\langle \text{stmt-list} \rangle ::= \langle \text{stmt} \rangle \mid \langle \text{stmt} \rangle \langle \text{stmt-list} \rangle \mid \langle \text{decl} \rangle \mid \langle \text{decl} \rangle \langle \text{stmt-list} \rangle$ 
 $\langle \text{decl} \rangle ::= \text{int } \langle \text{id} \rangle ; \mid \text{double } \langle \text{id} \rangle ;$ 
 $\langle \text{stmt} \rangle ::= \langle \text{asgn-stmt} \rangle \mid \langle \text{cout-stmt} \rangle$ 
 $\langle \text{asgn-stmt} \rangle ::= \langle \text{id} \rangle = \langle \text{expr} \rangle ;$ 
 $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \langle \text{expr} \rangle * \langle \text{expr} \rangle \mid ( \langle \text{expr} \rangle ) \mid \langle \text{id} \rangle$ 
 $\langle \text{cout-stmt} \rangle ::= \text{cout } \langle \text{out-list} \rangle ;$ 
```

etc., Must expand all nonterminals!

So a derivation of the program test would look like:

```
 $\langle \text{program} \rangle \Rightarrow \text{main } () \langle \text{block} \rangle$ 
 $\Rightarrow \text{main } () \{ \langle \text{stmt-list} \rangle \}$ 
 $\Rightarrow \text{main } () \{ \langle \text{decl} \rangle \langle \text{stmt-list} \rangle \}$ 
 $\Rightarrow \text{main } () \{ \text{int } \langle \text{id} \rangle ; \langle \text{stmt-list} \rangle \}$ 
 $\Rightarrow \text{main } () \{ \text{int } a ; \langle \text{stmt-list} \rangle \}$ 
 $\stackrel{*}{\Rightarrow} \text{complete C++ program}$ 
```

More on CFG for C++

We can write a CFG G s.t. $L(G) = \{\text{syntactically correct C++ programs}\}$.

But note that $\{\text{semantically correct C++ programs}\} \subset L(G)$.

Can't recognize redeclared variables:

```
int x;
double x;
```

Can't recognize if formal parameters match actual parameters in number and types:

```
declar: int Sum(int a, int b, int c) ...
call:   newsum = Sum(x,y);
```