Context-Free Languages

Regular languages:

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

Not Regular languages:

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

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

$$\mathbf{A \rightarrow x}$$

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)

$$\mathbf{S \rightarrow aSb \mid ab}$$

Derivation of aaabbb:

$$\mathbf{S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb}$$

L(G) =

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

$$\mathbf{S} \rightarrow \mathbf{aSa} \mid \mathbf{bSb} \mid \mathbf{a} \mid \mathbf{b} \mid \lambda$$

Derivation of ababa:

$$\mathbf{S} \Rightarrow \mathbf{aSa} \Rightarrow \mathbf{abSba} \Rightarrow \mathbf{ababa}$$

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

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

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

L(G) =

Derivations of aacbb:

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

   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).

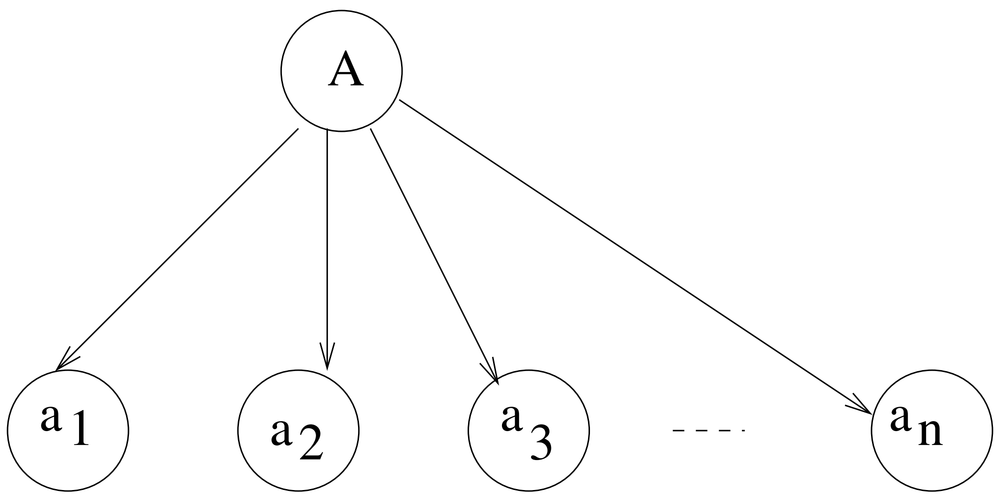
Definition: Rightmost derivation - in each step of a derivation, replace the rightmost variable. (see derivation 2 above).

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$\rightarrow a_1 a_2 a_3 \ldots a_n$, where A$\in$V, $a_i \in$(T$\cup$V$\cup\{\lambda\}$),**



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

$$S \rightarrow AcB$$
$$A \rightarrow aAa \mid \lambda$$
$$B \rightarrow Bbb \mid \lambda$$

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∈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 C++.
w is C++ program.
Is w syntactically correct?

Example

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

$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathbf{aSa} \mid \mathbf{b} \mid \lambda$$

L_1=**L(G)** =

Is abbab $\in$ **L(G)?**

Exhaustive Search Algorithm

For all i=1,2,3,...
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

$$\mathbf{A} \rightarrow \lambda$$
$$\mathbf{A} \rightarrow \mathbf{B}$$

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:

$$\mathbf{S \rightarrow SS \mid aa \mid aSa \mid b}$$

Show baaba $\notin$ L(G).

i=1

1. $\mathbf{S \Rightarrow SS}$
2. $\mathbf{S \Rightarrow aSa}$
3. $\mathbf{S \Rightarrow aa}$
4. $\mathbf{S \Rightarrow b}$

i=2

1. $\mathbf{S \Rightarrow SS \Rightarrow SSS}$
2. $\mathbf{S \Rightarrow SS \Rightarrow aSaS}$
3. $\mathbf{S \Rightarrow SS \Rightarrow aaS}$
4. $\mathbf{S \Rightarrow SS \Rightarrow bS}$
5. $\mathbf{S \Rightarrow aSa \Rightarrow aSSa}$
6. $\mathbf{S \Rightarrow aSa \Rightarrow aaSaa}$
7. $\mathbf{S \Rightarrow aSa \Rightarrow aaaa}$
8. $\mathbf{S \Rightarrow aSa \Rightarrow aba}$

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

$$\mathbf{A \rightarrow ax}$$

where A∈V, a∈T, and x∈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=

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

Derivation of a+b∗a is:

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

Corresponding derivation tree is:

Another derivation of a+b∗a is:

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

Corresponding derivation tree is:

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

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

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 $<>$
- 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: <program> is start symbol of grammar.)

```
<program>    ::= main () <block>
<block>      ::= { <stmt-list> }
<stmt-list>  ::= <stmt> | <stmt><stmt-list>
                 <decl> | <decl><stmt-list>
<decl>       ::= int <id> ; | double <id> ;
<stmt>       ::= <asgn-stmt> | <cout-stmt>
<asgn-stmt>  ::= <id> = <expr> ;
<expr>       ::= <expr> + <expr>
                 | <expr> * <expr>
                 | ( <expr> ) | <id>
<cout-stmt>  ::= cout <out-list> ;
```

etc., Must expand all nonterminals!

So a derivation of the program test would look like:

<program> ⇒ **main () <block>**
⇒ **main () { <stmt-list> }**
⇒ **main () { <decl> <stmt-list> }**
⇒ **main () { int <id>; <stmt-list>**
⇒ **main () { int a ; <stmt-list> }**
$\stackrel{*}{\Rightarrow}$ *complete C++ program*

More on CFG for C++

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

But note that {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);
```