Cpts 317
Fri Feb 14
Topic
Context-Free Language
Context-Free Languages

Regular Language

Reg. Expressions

NFA

CF Grammar

PDA

non-regular

E.g.: \( B = \{ 0^n 1^n | n \geq 0 \} \)

CFG: Major Applications

- Natural Language Processing
- Compilers
Context-free Grammars

Example of a CFG, we will call it $G_1$:

$$A \rightarrow OA1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

A grammar consists of:

- Substitution rules (productions)
- Variable
- Terminal (analogous to input alphabet)
- One variable designated as start variable
Ex. for $G_1$

$A \Rightarrow OA1 \Rightarrow OOA11 \Rightarrow O00A111$

$\Rightarrow O000A1111 \Rightarrow O000B1111$

$\Rightarrow O000\#1111$
Generation of strings from a grammar (to define a language)

1. Write down the start variable (the variable on the left-hand side of the top rule)

2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.

3. Repeat step 2 until no variable remains.
All strings generated in this way constitute the language of the grammar.

We write $L(G_i)$ for the language of Grammar $G_i$.

$L(G_i) = \{ 0^n \# 1^n \mid n \geq 0 \}$.

Any language that can be generated by some context-free grammar is called context-free language (CFL).

CFL vs. Context-Sensitive Language
Notation

If $u, v$ and $w$ are strings of variables and terminals, and $A \rightarrow w$ is a rule,
- we say that
  \[ uAv \text{ yields } uAw \]
  and write it as $uAv \Rightarrow uAw$

- we say that
  \[ u \text{ drives } v, \text{ and write as } u \Rightarrow v \]
  if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$
  exists for $k \geq 0$ and
  \[ u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v \]

The language of the grammar is $\{ w \in \Sigma^* | S \Rightarrow w \}$
Formal Definition of a CFG

A CFG is a 4-tuple \((V, \Sigma, R, s)\),
where

1. \(V\) is a finite set called the variable,
2. \(\Sigma\) is a finite set (disjoint from \(V\)) called the terminals,
3. \(R\) is a finite set of rules, with each rule being a variable and a string of variables and terminals, and
4. \(s \in V\) is the start variable.
Example

\[ G_3 = ( \{ s \}, \{ a, b \}, R, s ) \]

The set of rules, \( R \), is:

\[ s \rightarrow a \, s \, b \]
\[ s \rightarrow s \, s \]
\[ s \rightarrow \epsilon \]

A shorthand for something like this is

\[ s \rightarrow a \, s \, b \mid s \, s \mid \epsilon \]

\( abab \quad aaabbb \quad aababb \)
Example

$$G_4 = (V, \Sigma, R, <EXPR>)$$

V is \( \{ <EXPR>, <TERM>, <FACTOR> \} \)

\( \Sigma \) is \( \{ 0, +, \times, c, ) \} \)

The rules are:

\[
<EXPR> \rightarrow <EXPR> + <TERM> \mid <TERM>
\]

\[
<TERM> \rightarrow <TERM> \times <FACTOR> \mid <FACTOR>
\]

\[
<FACTOR> \rightarrow ( <EXPR> ) \mid a
\]
Consider the strings
\[ a + a \times a \]
\[ (a + a) \times a \]

These strings can be generated with grammar 64.

The parse trees are:

```
<EXPR>   <TERM>  <_FACTOR>
    /     /       ?
<TERM>   /       <TERM>  <FACTORS>
    /     /         ?
<FACTORS> /           <FACTORS>
      /     /      ?
  a      +     a        a
```
\((a + a) \times a\)
Designing CFGs

As with design of FA, design of CFG requires creativity.

Useful techniques

1. Break down to simpler piece
   (Many CFLs are union of simpler CFLs)

Individual grammars can be merged into a grammar for the original language by combining their rules, and then adding the new rule

\[ S \rightarrow S_1 \mid S_2 \mid \cdots \mid S_k \],

where the variable \( S_i \) are start variables for individual grammars.
Example

To get a grammar for the language
\[ \{ \text{0}^n \text{1}^m \mid n \geq 0 \} \cup \{ \text{1}^n \text{0}^m \mid n \geq 0 \}, \]
first construct the grammar
\[
S_1 \rightarrow \text{0}S_1 \text{1} \mid \varepsilon
\]
for the language \( \{ \text{0}^n \text{1}^m \mid n \geq 0 \} \),
and the grammar
\[
S_2 \rightarrow \text{1}S_2 \text{0} \mid \varepsilon
\]
for the language \( \{ \text{1}^n \text{0}^m \mid n \geq 0 \} \), and then
add the rule \( S \rightarrow S_1 \mid S_2 \) to give
the grammar
\[
S \rightarrow S_1 \mid S_2 \\
S_1 \rightarrow \text{0}S_1 \text{1} \mid \varepsilon \\
S_2 \rightarrow \text{1}S_2 \text{0} \mid \varepsilon
\]
2. Constructing a CFG for a language that happens to be regular is easy if you can first construct a DFA for that language.

You can convert any DFA into an equivalent CFG as follows:

- Mark a variable $R_i$ for each state $q_i$ of the DFA.
- Add the rule $R_i \rightarrow a R_j$ to the CFG if $S(q_i, a) = q_j$ is a transition in the DFA.
- Add the rule $R_i \rightarrow \epsilon$ if $q_i$ is an accept state of the DFA.
• Make $R_0$ the start variable of the grammar, where $R_0$ is the start state of the machine.

• Verify on your own that the resulting CFG generates the same language that the DFA recognizes.
Certain CFLs contain strings with two substrings that are "linked" in the sense that a machine for such a language would need to remember an unbounded amount of information about one of the substrings to verify that it corresponds properly to the other substring.

E.g. This situation occurs in the language \( \{0^n1^n \mid n \geq 0\} \) because a machine would need to remember the number of 0s in order to verify that it equals the number of 1s.
You can construct a CFG to handle this situation by using a rule of the form

$$R \rightarrow uRv$$

which generates strings where in the portion containing the $u$'s corresponds to the portion containing the $v$'s.
4 In more complex languages, the strings may contain certain structure that appear recursively as part of other (or the same) structure.

E.g. This situation occurs in the G4 we saw earlier.

Any time the symbol $a$ appears, an entire parenthesized expression might appear recursively instead.

To achieve this effect, place the variable symbol generating the structure in an location of the rule, corresponding to where the structure may recursively appear.