Cpts 317

Wed Feb 19

Topic

Context-free Grammars
Agenda

* Finish discussion of designing CFGs

* Ambiguity

* Chomsky Normal Form (CNF)
Designing CFGs

General Techniques

1. Break into simpler parts
2. Go via DFA, first when (if) the language is regular
3. Deal with "linked" substrings
4. Deal with recursive structure

Saw in last lecture

In note of last lecture
already seen, one more
(repeated in next few pages
for convenience)
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 \( \{0^i 1^n \mid n \geq 0\} \) 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.
In more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures.

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 the location of the rule, corresponding to where the structure may recursively appear.
Ambiguity

* Sometimes a grammar can generate the same string in several different ways.

* Such a string will have several different parse trees and thus different meanings.

* If a grammar generates the same string in several different ways, we say the string is derived ambiguously in that grammar.

* If a grammar generates some string ambiguously, we say that the grammar is ambiguous.
Example
Consider this grammar $G_5$:

\[
<EXPR> \rightarrow <EXPR> + <EXPR> \\
<EXPR> \rightarrow <EXPR> \times <EXPR> \\
<EXPR> \rightarrow (<EXPR>) \\
<EXPR> \rightarrow a
\]

$G_5$ generates the string $a + a \times a$ ambiguously. Here are its two different parse trees.
* In contrast, the grammar $G_4$ we saw in last lecture generates the same language, but is unambiguous, since it generates strings that have unique parse trees.

* As a reminder, $G_4$ was:

\[
\begin{align*}
  \langle \text{EXPR} \rangle & \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \\
  \langle \text{TERM} \rangle & \rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle \\
  \langle \text{FACTOR} \rangle & \rightarrow (\langle \text{EXPR} \rangle) \mid a
\end{align*}
\]

* The "conversion" of $G_5$ into $G_4$ in fact illustrates some of the techniques employed in converting an ambiguous grammar into an unambiguous one.
* However, such conversions are not always possible.

* There are some context-free languages that can be generated only by ambiguous grammars.

* This is in fact close to one of the fundamental "negative" results we have in the theory of computation.

* We will learn towards the end of the course that there is no algorithm whatsoever that can even tell us whether a CFG is ambiguous in the first place.
* Let us return to ambiguity, and formalize the notion.

* When we say a grammar generates a string ambiguously, we mean that the string has two different parse trees, not two different derivations.

* Two derivations may differ merely in the order in which they replace variables, yet not in their overall structure.

* For example, the expression 2 + 4 may be derived from 65 in two ways:

  \[
  E \Rightarrow E + E \Rightarrow E + 4 \Rightarrow 2 + 4 \\
  E \Rightarrow E + E \Rightarrow 2 + E \Rightarrow 2 + 4
  \]
* To concentrate on structure, we define a type of derivation that replaces variables in a fixed order.

* A derivation of a string $w$ in a grammar $G$ is a leftmost derivation if at every step the leftmost remaining variable is the one replaced.

* Now we are ready for a formal definition of ambiguous grammars.

* Definition

A string $w$ is derived ambiguously in CFG $G$ if it has two or more different leftmost derivations. Grammar $G$ is ambiguous if it generates some string ambiguously.
Chomsky Normal Form

One of the simplest and most useful forms for parsing CFGs in CNF.

* Definition

A CFG is in CNF if every rule is of the form

\[ A \rightarrow BC \]
\[ A \rightarrow a \]

where \( a \) is any terminal and \( A, B, \) and \( C \) are any variables – except that \( B \) & \( C \) may not be the start variable. In addition, we permit the rule \( S \rightarrow E \), where \( S \) is the start variable.
**Theorem**

Any context-free language is generated by a context-free grammar in CNF.

**Proof Idea**

- We convert any grammar $G$ into CNF.
- The conversion has several stages wherein rules that violate the conditions are replaced with equivalent ones that are satisfying.
- First, we add a new start variable.
- Then, we eliminate all $e$-rules of the form $A \rightarrow e$.
- We also eliminate all unit rules of the form $A \rightarrow B$. 
- In both of the above cases, we patch up the grammar to be sure that it still generates the same language.

- Finally, we convert the remaining rules into the proper form.

Proof: See the book (page 109) for a formal proof and procedure for conversion.

Example in the next page illustrates the procedure.
Example (Example 2.10 in the book)

$G_6:

S \rightarrow ASA | aB
A \rightarrow B | s
B \rightarrow b | e

1.

$S_0 \rightarrow S$
$S \rightarrow ASA | aB
A \rightarrow B | s
B \rightarrow b | e$

2. Remove epsilon rule $B \rightarrow e$

$S_0 \rightarrow S
S \rightarrow ASA | aB | a
A \rightarrow B | s | e
B \rightarrow b$

R \rightarrow uAv
Add a rule
R \rightarrow uv
Remove \( A \rightarrow E \)

\[
S_0 \rightarrow S \\
S \rightarrow ASA \mid aB \mid a \mid CA \mid AS \mid S \\
A \rightarrow B/S \\
B \rightarrow b
\]

3.a.i) Remove unit rule \( S \rightarrow S \)

\[
S_0 \rightarrow S \\
S \rightarrow ASA \mid aB/a \mid SA \mid AS \\
A \rightarrow B/S \\
B \rightarrow b
\]

ii) Remove unit rule \( S_0 \rightarrow S \)

\[
S_0 \rightarrow ASA \mid aB/a \mid SA \mid AS \\
S \rightarrow ASA \mid aB/a \mid SA \mid AS \\
A \rightarrow B/S \\
B \rightarrow b
\]
3b. i) Remove unit rule $A \rightarrow B$

$S_0 \rightarrow \text{ASA} | aB | a | SA | AS$

$S \rightarrow \text{ASA} | aB | a | SA | AS$

$A \rightarrow s | b$

$B \rightarrow b$

ii) Remove unit rule $A \rightarrow S$

$S_0 \rightarrow \text{ASA} | aB | a | SA | AS$

$S \rightarrow \text{ASA} | aB | a | SA | AS$

$A \rightarrow b | \text{ASA} | aB | a | SA | AS$

$B \rightarrow b
4. Convert the remaining rules into the proper form by adding additional variable \( \alpha \) rule.

\[
S_0 \rightarrow AA_1 \mid UV \mid a \mid SA \mid AS
\]

\[
S \rightarrow AA_1 \mid UV \mid a \mid SA \mid AS
\]

\[
A \rightarrow b \mid AA_1 \mid UV \mid a \mid SA \mid AS
\]

\[
A_0 \rightarrow SA
\]

\[
\alpha \rightarrow A
\]

\[
B \rightarrow b
\]