CS 3100 – Models of Computation – Fall 2011 – Notes for L19

Design a CFG for $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0,\ if\ odd(i)\ then\ j = k\}$

- Do case analysis at the top level

  ```
  Cases are: i is odd and i is even
  ```

- Write down productions for each case, naming *parsing subgoals* through non-terminals

  ```
  S -> Odd Mbc  |  Even Nbc

  Even -> epsilon | a a Even

  Odd -> a Even

  Mbc -> b Mbc c | epsilon

  Nbc -> Bs Cs

  Bs -> b Bs | epsilon

  Cs -> c Cs | epsilon
  ```

Design a CFG for $L_2 = \{a^i b^j c^k \mid i, j, k \geq 0,\ if\ (i = 1)\ then\ 0 \leq j - k \leq 1\}$

Design a PDA for L_1

Design a PDA for L_2

Direct conversion of CFG to PDA for L_1

Direct conversion of CFG to PDA for L_2

CFG for $L_3 = \{w_1 1 w_2 \ \mid \ \mid w_1 \mid = \mid w_2 \mid, \ w_1, w_2 \in \{0,1\}^*\}$

CFG for $L_{wwc} = \overline{L_{ww}}$ **where** $L_{ww} = \{ww \ \mid \ \mid w \in \{0,1\}^*\}$

Note that L_{ww} is not context free! So CFLs are not closed under complementation. They are closed under union. Thus, they can't be closed under intersection.

Show that G_1 below is consistent and complete with respect to language $L_{eqab} = \{w \mid w \in \{a,b\}^*, \text{ and } \#_a(w) = \#_b(w)\}$

```
G1: S -> a S b S | b S a S | epsilon
```

Guess the language G_2 is generating and show it is consistent and complete

```
G2: S -> ( W S | epsilon
    W -> ( W W | )
```

Simplify the grammar G_3 below, stating the steps

```
G3:
S -> A B | D
A -> 0 A | 1 B | C
B -> 2 | 3 | A
D -> A C | B D
E -> 0
```

By bottom-up marking, locate all generating symbols. Eliminate those that are not. A generating non-terminal is one which has at least one production where all the RHS non-terminals are generating. Then through graph-search (BFS or DFS or others) from S, locate those that are generating and reachable. The rest can go.

Simplify the grammar G_4 below, stating the steps

```
G4:
S -> A | B
A -> ( W A | ( X C
B -> ( W B | ( X D
W -> ( W W | ( X Y
X -> ( W X | ( X Z
W -> )
B -> epsilon
```

Purely left-linear, purely right-linear, NFAs: Convert G_5 into an NFA

```
G5:
S -> 0 A | 1 B | epsilon
A -> 1 C | 0
B -> 0 C | 1
C -> 1 | 0 C
```

Present the NFA for "second from last is a 1" as a CFG

Mixed left and right linearity *does not guarantee* that things are regular

```
Example: This is context-free.
S -> 0 T | epsilon
T -> S 1
```

Reverse the CFG G_4, presenting the result as a CFG

Note that this approach can be used to render a purely left-linear grammar as a purely right-linear one, and then one can draw the NFA.

Present G_6 as an equivalent regular expression

```
S -> T T
T -> U T | U
U -> 0 U | 1 U | epsilon
```

Can we simplify G_7 as an equivalent regular expression?

```
S -> T T | U
T -> 0 T | T 0 | #
U -> 0 U 0 0 | #
```

Argue the cases underlying this example.

A Pumping Lemma for CFLs

For example, consider the CFG

```
S -> ( S ) | T | e

T -> [ T ] | T T | e.
```

Here is an example derivation:

```
S => ( S ) => (( T )) => (( [ T ] )) => (( [  ] ))
                 ^            ^
              Occurrence-1   Occurrence-2

Occurrence-1 involves Derivation-1: T => [ T ] => [ ]
Occurrence-2 involves Derivation-2: T => e
```

Here, the second `T` arises because we took `T` and expanded it into
`[ T ]` and then to `[ ]`. Now, the basic idea is that we can use Derivation-1 used in the first occurrence in place of Derivation-2, to obtain a longer string:

```
S => (S) => ((T))  => (( [ T ] )) => (( [[ T ]] )) => (( [[ ]] ))
               ^            ^
              Occurrence-1 Use Derivation-1 here
```

In the same fashion, we can use Derivation-2 in place of Derivation-1 to obtain a shorter string, as well:

```
S => ( S ) => ( ( T ) ) => ( ( ) )
                  ^
                  Use Derivation-2 here
```

When all this happens, we can find a repeating non-terminal that can be pumped up or down. In our present example, it is clear that we can manifest $(([^i\ ]^i))$ for $i \geq 0$ by either applying `Derivation-2` directly, or by applying some number of `Derivation-1`s followed by `Derivation-2`. In order to conveniently capture the conditions mentioned so far, it is good to resort to parse trees. Consider a CFG with $|V|$ non-terminals, and with the right-hand side of each rule containing at most b syntactic elements (terminals or non-terminals). Consider a b-ary tree built up to height $|V| + 1$, as shown in Figure 1. The string yielded on the frontier of the tree $w = uvxyz$. If there are two such parse trees for w, pick the one that has the fewest number of nodes. Now, if we avoid having the same non-terminal used in any path from the root to a leaf, basically each path will "enjoy" a growth up to height at most $|V|$ (recall that the leaves are terminals). The string $w = uvxyz$ is, in this case, of length at most $b^{|V|}$. This implies that *if we force the string to be of length* $b^{|V|+1}$ (called p hereafter), a parse tree for this string will have some path that repeats a non-terminal. Call the higher occurrence V_1 and the lower occurrence (contained within V_1) V_2. Pick the lowest two such repeating pair of non-terminals. Now, we have these facts:

- $|vxy| \leq p$; if not, we would find two other non-terminals that exist lower in the parse tree than V_1 and V_2, thus violating the fact that V_1 and V_2 are the lowest two such.
- $|vx| \geq 1$; if not, we will in fact have $w = uxz$, for which a shorter parse tree exists (namely, the one where we directly employ V_2).
- Now, by pumping, we can obtain the desired repetitions of v and y, as described in Theorem 0.1.

Theorem 0.1 *Given any CFG $G = (N, \Sigma, P, S)$, there exists a number p such that given a string w in $L(G)$ such that $|w| \geq p$, we can split w into $w = uvxyz$ such that $|vy| > 0$, $|vxy| \leq p$, and for every $i \geq 0$, $uv^ixy^iz \in L(G)$.*

We can apply this Pumping Lemma for CFGs in the same manner as we did for regular sets. For example, let us sketch that L_{ww} of page **??** is not context-free.

Illustration 0.1 Suppose L_{ww} were a CFL. Then the CFL Pumping Lemma would apply. Let p be the pumping length associated with a CFG of this language. Consider the string $0^p1^p0^p1^p$ which is in L_{ww}. The segments v and y of the Pumping Lemma are contained within the first 0^p1^p block, in the middle 1^p0^p block or in the last 0^p1^p block, and in each of these cases, it could also have fallen entirely within a 0^p block or a 1^p block. By pumping up or down, we will then obtain a string that is not within L_{ww}.

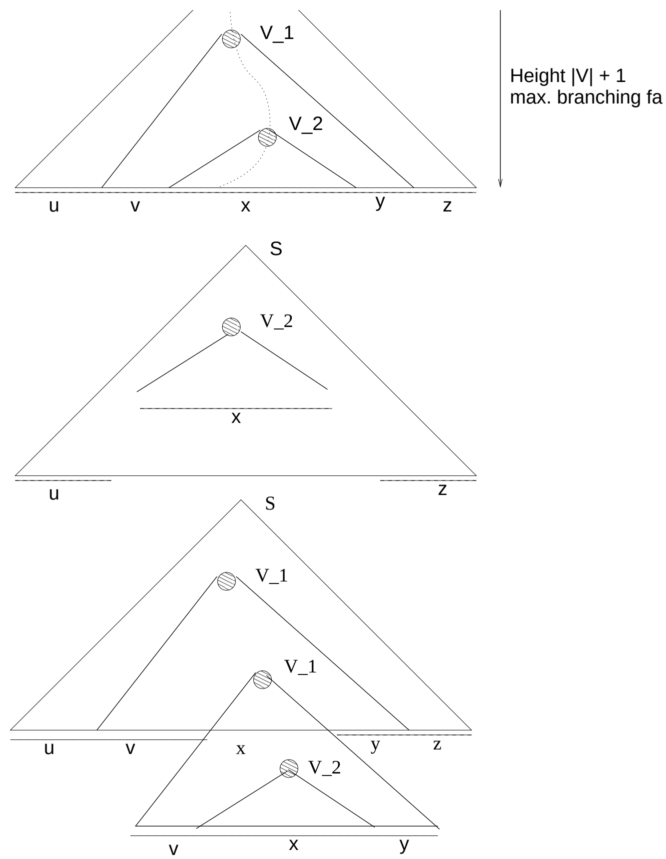

Figure 1: Depiction of a parse tree for the CFL Pumping Lemma. The upper drawing shows a very long path that repeats a non-terminal, with the lowest two repetitions occurring at V_2 and V_1 (similar to `Occurrence-1` and `Occurrence-2` as in the text). With respect to this drawing: (i) the middle drawing indicates what happens if the derivation for V_2 is applied in lieu of that of V_1, and (ii) the bottom drawing depicts what happens if the derivation for V_2 is replaced by that for V_1, which, in turn, contains a derivation for V_2

If-then-else Ambiguity

An important practical example of ambiguity arises in the context of grammars pertaining to `if` statements, as illustrated below:

```
STMT ->   if EXPR then STMT
        | if EXPR then STMT else STMT
        | OTHER

OTHER -> p

EXPR  -> q
```

The reason for ambiguity is that the `else` clause can match either of the *then* clauses. Compiler writers avoid the above if-then-else ambiguity by modifying the above grammar in such a way that the `else` matches with the closest unmatched `then`. One example of such a rewritten grammar is the following:

```
STMT ->  MATCHED | UNMATCHED

MATCHED -> if EXPR then MATCHED else MATCHED | OTHER

UNMATCHED ->  if EXPR then STMT
            | if EXPR then MATCHED else UNMATCHED

OTHER -> p

EXPR  -> q
```

This forces the `else` to go with the closest previous unmatched `then`.

An example of an inherently ambiguous language is

$$\{0^i 1^j 2^k \mid i, j, k \geq 0 \wedge i = j \vee j = k\}.$$

Machines	Languages	Nature of grammar
DFA/NFA	Regular	Left-linear or Right-linear productions
DPDA	Deterministic CFL	Each LHS has one non-terminal The productions are deterministic
NPDA (or "PDA")	CFL	Each LHS has only one non-terminal
LBA	Context Sensitive Languages	LHS may have length > 1, but $\mid$ LHS$\mid \leq \mid$RHS$\mid$, ignoring ε productions
DTM/NDTM	Recursively Enumerable	General grammars ($\mid$LHS$\mid \geq \mid$RHS$\mid$ allowed)

Figure 2: The Chomsky hierarchy and allied notions