CS 310, Assignment 4

Answers

1. Let $\Sigma = \{a, b\}$. Give a context-free grammar that generate the following language:

$$\{a^{i+k}b^{3i+k} : i \geq 0, k \geq 1\} + \{a^{3i+k}b^{i+k} : i \geq 0, k \geq 1\}$$

ANSWER: The construction is very easy if we think of the language as $\{a^i a^k b^k b^{3i} : i \geq 0, k \geq 1\} + \{a^{3i} a^k b^k b^i : i \geq 0, k \geq 1\}$ (which is clearly equivalent to the original formulation). The start symbol S will be responsible for generating either a string from the first language (generated by the nonterminal A) or a string from the second language (with the nonterminal B being responsible for that). We therefore put:

$$S \to A \qquad S \to B$$

To specify the first language we then have the nonterminal A responsible for the outer a's and b's and another (say, X) responsible for the inner group of a's and b's, as follows:

$$\begin{array}{ll} A \to aAbbb & A \to X \\ X \to aXb & X \to ab \end{array}$$

Similarly, B will be responsible (in the second language) for the outer a's and b's.

$$B \to aaaBb \qquad B \to X$$

Note that we opportunistically re-used X for the inner a's and b's since they look the same as in the first language. It is not a mistake to use a different nonterminal for this purpose, but that nonterminal will be nonetheless wasteful since its behavious will be the same as the behaviour of X, which we have already defined.

---

2. The following grammar with start symbol S generates the language $\{a^i b^j c^k : i = j \text{ or } j = k\}$:

$$\begin{array}{llll} S \to XC & S \to AY & X \to aXb & X \to \varepsilon \\ C \to cC & C \to \varepsilon & A \to aA & A \to \varepsilon \\ Y \to bYc & Y \to \varepsilon & & \end{array}$$

   (a) Show that this grammar is ambiguous.
   Hint. Consider a string that has an equal number of a, b, and c.

ANSWER: We are looking for a string that satisfies both conditions from the disjunction defining the language. The simplest string that shows ambiguity is therefore *abc*. That string is generated by both XC and AY, as follows:

$$S \Rightarrow XC \Rightarrow aXbC \Rightarrow aXbcC \Rightarrow abcC \Rightarrow abc$$
$$S \Rightarrow AY \Rightarrow aAY \Rightarrow aAbYc \Rightarrow abYc \Rightarrow abc$$

Clearly these two derivations are not similar and so the grammar is ambiguous. The golden rule for showing that two derivations are or are not similar is to draw and then compare the respective parse trees. You are welcome (even encouraged!) to do so if you wish, but in this case the lack of similarity is obvious since with the exception of S the terminals that appear in the first derivation do not appear in the second and also the other way around.

(b) Is this language inherently ambiguous? State your thoughts one way or another (you only need to provide your thoughts, not a proof).

ANSWER: The language is inherently ambiguous. To see this we rewrite the language as $L_1 + L_2$ where $L_1 = \{a^i b^j c^k : i = j\}$ and $L_2 = \{a^i b^j c^k : j = k\}$. If we generate each of these two individual languages separately using two "sub-grammars" (like I did for this particular question) then we necessarily obtain an overall ambiguous grammar. Indeed, consider some string containing an equal number of *a*, *b*, and *c* symbols (like our old friend *abc*); this string belongs to the intersection of L_1 and L_2 and so can always be generated in two ways (one for each sub-grammar).

Now, is there a cleverer way or generating the two languages at once? That does not appear to be the case. We know how to generate an equal number of two symbols (we have seen this countless times in the course), but there is no way to implement the choice of *which* two symbols to match in an unambiguous manner.

Alternatively, we note that we can always construct a deterministic pushdown automaton for an unambiguous language (we have not talked about this in class, but even so this alternate answer could be instructive so I am including it anyway). Conversely, it is not possible to construct a deterministic pushdown automaton for an inherently ambiguous language. Consider now a pushdown automaton that accepts our language. This automaton will have to decide at the very beginning whether to compare (using the stack) the *a*'s against the *b*'s, or ignore the number of *a*'s and perform a comparison later (of *b*'s against the *c*'s). Any successful such a comparison will do, but at the beginning of the input (when we only see *a*'s) there is no way to decide which one is the winner. Furthermore, if we decide to perform one comparison and our choice proves wrong, that happens much later in the input and we have no way to come back and perform the other comparison. For all these reasons a pushdown automaton accepting our language is necessarily nondeterministic and so the language is inherently ambiguous.

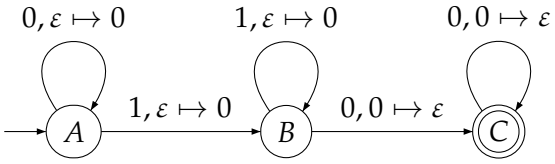
---

3. Design a *deterministic* pushdown automaton that recognizes the language

$$\{0^i 1^k 0^{i+k} : i \geq 1, k \geq 1\}$$

Draw a table that traces the behavior of your pushdown automaton on the input 0011000 and explain how this input is accepted or rejected (as the case might be).

ANSWER: We push one symbol on the stack for every 0 in the first batch, and also for all the subsequent 1's. We then pop the stack for all the 0's in the second batch and we are done. I chose to push 0 symbols on the stack but any other symbol is fine.



Note that we have at least one 1 and so we can transition on the first 1 from A to B. Having at least one 1 means that we have at least one 0 in the second batch, hence the label on the transition from B to C. It is not a mistake to mark B accepting, but we will never accept in B anyway, since there must be a 0 in the second batch (as explained above).

The following is the trace of the automaton on the string 0011000:

Input	State	Stack
0011000	A	ε
011000	A	0
11000	A	00
1000	B	000
000	B	0000
00	C	000
0	C	00
ε	C	0

At the end of the day we end up at the end of the input, in a accepting state (C), but with a non-empty stack, so the input is rejected.

---

4. Is the language $L = \{a^i b^i c^j d^i : i \geq 0, j \geq 0\}$ context-free? Prove one way or another.

ANSWER:

Assume that L is context-free and let p be the constant threshold given by the pumping lemma. We consider the string $s = a^p b^p d^p \in L$. We note the following:

(a) We can choose a string that contains c, but we can also choose a string that does not contain this symbol. Indeed, the pumping lemma applies to *any* (long enough) string in the language. We chose to go the c-less route since we can pump the c's as many times as we want and so we cannot easily (if at all) reach a contradiction when c's are present.

(b) $|s| \geq p$ and so the pumping lemma applies.

Therefore we can write $s = uvwxy$ with the substrings u, v, w, x, and y satisfy the conditions of the pumping lemma and so $uv^k wx^k y \in L$ for any $k \geq 0$.

(a) If either v or x contains occurrences of two or more distinct symbols, then uv^2wx^2y is not in $a^*b^*c^*d^*$ and so not in L, a contradiction.

(b) The only other possibility is that v and x each contains occurrences of at most one symbol. By the pumping lemma we know that v and x cannot both be ε. Thus the strings v and x contain occurrences of exactly one or of exactly two of the symbols a, b, d. This means that uv^2wx^2y cannot have equal numbers of a's, b's and d's, and so $uv^2wx^2y \notin L$, a contradiction.

In all it follows that L is not context free.
