COSC341 TUTORIALS 9 AND 10

Exercises about building pushdown automata, context-free grammars, and checking whether or not languages are context-free. As usual, $\Sigma = \{a, b\}$ unless otherwise specified.

1. *Build a PDA and describe a context-free grammar for each of the following languages:*

   (a) Equal*, the set of strings having the same number of a's as b's in any order.*

   PDA: Use a stack alphabet comprising symbols A and B and a bottom-of-stack marker $\#$. When reading an input if the stack is empty, or the letter on the stack top matches the letter read, push another letter of that type onto the stack (i.e., if top is A and we read a, we push A). If the stack is not empty and the top is of the opposite type, then pop it instead (i.e., if top is B and we read a, we pop).

   CFG:

   $$S \to \epsilon \mid aSbS \mid bSaS$$

   The first S in each case above could be restricted to a word which has equal a's and b's in which each prefix has at least as many a's as b's (i.e., the specified b in the production rule is the first one that equalises the number of a's and b's), but this is not necessary.

   (b) $\{a^n b^n c^m \mid n, m \geq 0\}$.

   PDA: Just push A's when reading a's and pop them when reading b's. Allow a non-deterministic transition to another state (the only accepting one) in which only c's can be read but they don't affect the stack. The only way to get empty stack and accepting state is to be in the language above. If you just want 'empty stack only' as the acceptance criterion, then start by pushing an X onto the stack without consuming any input, then process a's and b's as above, and allow a transition that pops X after which only c's can be read but they don't affect the stack.

   CFG:

   $$\begin{aligned} S &\to TC \\ T &\to \epsilon \mid aTb \\ C &\to \epsilon \mid cC. \end{aligned}$$

(c) BalancedParentheses, *the set of strings over* $\{(,),a,b\}$ *in which the parentheses are properly balanced (and the other symbols can occur arbitrarily).*

PDA: Ignore anything that's not a parenthesis. Push a token for each (and pop one for each).

CFG:

$$S \to \epsilon \mid aS \mid bS \mid (S)S$$

(d) PostFix, *the set of strings over* $\{a,+,-\}$ *that represent legitimate expressions written in postfix notation, where* $+$ *is a binary operator, and* $-$ *a unary operator. That is,* a *should be accepted and pushed at any time,* $+$ *can be applied only if there are at least two elements in the stack and reduces the stack size by 1, and* $-$ *can be applied only if the stack is non-empty and does not change the size of the stack.*

PDA: Is basically described in the question. The only slight subtlety is that we should use a different symbol for the first item in the stack as for subsequent items so that we can tell whether $+$ is allowed.

CFG:

$$S \to \epsilon \mid T$$
$$T \to a \mid TT+ \mid T-$$

2. *Verify that if* L *and* K *are context-free languages, and* R *is a regular language then all of* LK*,* $L \cup K$*,* L^* *and* $L \cap R$ *are context-free.*

Suppose we have PDAs for L and for K that accept by empty stack.

To get one for LK modify the one for L to start by adding a new symbol X to the initially empty stack. Add a transition that consumes no input, pops the X and transfers control to the PDA for K.

To get one for $L \cup K$ just start with a non-deterministic transition to one or the other.

To get one for L^* use the same idea as for LK - just that if you see X at the top of the stack you can either pop it (and accept), or restart (the fact that it's there means you have seen something in L so you're either accepting that, or concatenating it to . . .)

For $L \cap R$ just change the control of the underlying finite state machine to pairs of states, one from L and one from a DFA for R. To accept, you probably want to do the X trick as above, and allow the X to be popped and execution terminated only if the current state of the DFA is accepting.

3. *Apply the pumping lemma and write out detailed arguments showing that the following languages are not context-free:*

   Note that I've been deliberately repetitive in the structure of the arguments – this is intended to emphasise that the application of the pumping lemma is a fairly standardised process.

   (a) $\{a^n b^m a^n b^m \mid n, m \geq 0\}$.

   If this language were context-free then the pumping lemma would apply to it. That is, for any sufficiently long word in the language we could find a short internal segment that could be pumped. But consider the word $a^N b^N a^N b^N$ where N is larger than our criterion for "sufficiently long". A short internal segment can't include both parts of the first a^N and parts of the last b^N so, whatever we pump it will no longer match the length of the part that's missed. Therefore, we get a word not in the language and hence a contradiction, i.e., the language is not context-free.

   (b) $\{a^p \mid p \text{ is prime}\}$.

   If this language were context-free then the pumping lemma would apply to it. That is, for any sufficiently long word in the language we could find a short internal segment that could be pumped. But consider a word a^N in the language (so N is prime) where N is larger than our criterion for "sufficiently long". Pumping means that for some $0 < t < N$ all the numbers $N + kt$ for $k \geqslant 0$ would be in the language, i.e., prime. But, choosing $k = N$ we get $N + Nt = N(t+1)$ which is visibly not prime as both N and $t + 1$ are greater than 1. So we get a word not in the language and hence a contradiction, i.e., the language is not context-free.

   $\{a^n b^n a^n \mid n \geq 0\}$.

If this language were context-free then the pumping lemma would apply to it. That is, for any sufficiently long word in the language we could find a short internal segment that could be pumped. But consider a word $a^N b^N a^N$ in the language where N is larger than our criterion for "sufficiently long". But now the short internal segment can't contain parts of both the initial block of a's and the final block of a's. Therefore, pumping it either changes the balance between those two blocks or, if neither is involved changes the number of b's without changing the number of a's. In either case we get a word not in the language and hence a contradiction, i.e., the language is not context-free.

4. *Find two context-free languages whose intersection is not context-free.*

   By a modification of the argument in 1(b) above, both the language $\{a^n b^n a^k : k, n \geqslant 0\}$ and the language $\{a^k b^n a^k : k, n \geqslant 0\}$ are context-free. But their intersection is the language $\{a^n b^n a^n \mid n \geq 0\}$ which we saw in Question 3(c) was not context-free.

5. *Show, by intersection with a suitable regular language and deriving a contradiction, that* Square*, the set of all words of the form ww where $w \in \{a, b\}^*$, is not context-free.*

   The language $\{a^n b^m a^n b^m \mid n, m \geq 0\}$ is the intersection of Square and the regular language $a^* b^* a^* b^*$. So, if Square were context-free, then so would $\{a^n b^m a^n b^m \mid n, m \geq 0\}$ be, according to the last part of Question 2. But we know this is not the case, so Square can't be context-free.

6. *We saw in Question 3(c) that $\{a^n b^n a^n \mid n \geq 0\}$ is not context-free. Imagine a machine like a PDA that uses a queue rather than a stack. How could you recognise this language?*

   Start by reading a's and pushing A's. Now switch to reading b's, removing an A and pushing (in queue fashion) a B. Finally, read a's and remove B's. If the queue winds up empty you have exactly a word of the form you wanted to recognise.