CS3331 - Assignment 2 - 2018

Context-Free Languages

Due: 9 am Friday, Oct 26, 2018 (Latest to submit: 9 am Monday, Oct 29)

1. (30pt) Let $L = \{w \in \{\mathsf{A}-\mathsf{Z}, \neg, \wedge, \vee, \rightarrow, (,)\}^* : w \text{ is a syntactically legal Boolean expression}\}$.

   (a) Write an unambiguous context-free grammar that generates L. The grammar should both:

   - use the following precedence levels to the operators (from highest to lowest): $\neg, \wedge, \vee,$ and $\rightarrow$;
   - associate left given operators of equal precedence (if an operator $\sim$ is left-associative, then $\mathsf{A} \sim \mathsf{B} \sim \mathsf{C}$ is interpreted as $(\mathsf{A} \sim \mathsf{B}) \sim \mathsf{C}$).

   The operator with the lowest precedence level will be the first to be generated by the grammar (just as for the arithmetic expression grammar) and we need to separate the generation of each of the operators (in the reverse order of the precedence level) by using different non-terminal symbols. Also, to avoid abiguity we need to avoid ε-rules and symmetric right-hand sides of rules. Moreover, to ensure left-associativity, we need to ensure, when two of the same symbols appear in an unbracketed expression, that the first appearance (from the left) of the given symbol is parsed as a single expression connected to the rest. This is accomplished by having the first appearance of the symbol be generated last by the grammar. This means that the symbol on the left-hand side of a rule that generates a string containing a given operator should appear first on the right-hand side of that rule. Finally, just as for BEDMAS, brackets have the highest precedence, so again are generated last by the grammar. Taking these observations together we can produce the grammar:

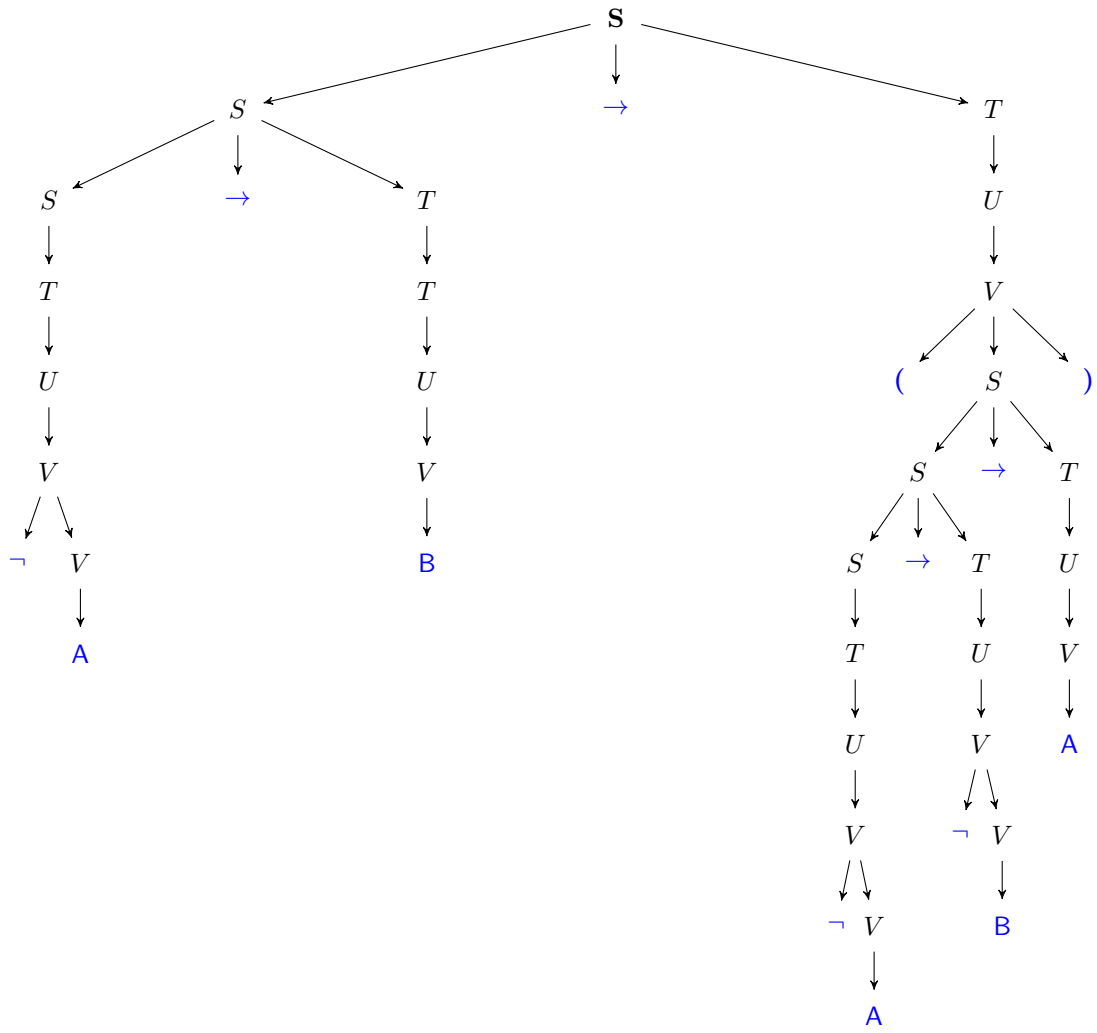
   $S \rightarrow S \rightarrow T \,|\, T$
   $T \rightarrow T \vee U \,|\, U$
   $U \rightarrow U \wedge V \,|\, V$
   $V \rightarrow \neg V \,|\, \mathsf{A}-\mathsf{Z} \,|\, (S)$

   where I have used the rule $V \rightarrow \mathsf{A}-\mathsf{Z}$ as an abbreviation for 26 rules that generate single capital Latin letters.

   (b) Show the parse tree that your grammar will produce for the string

   $$\neg\mathsf{A} \rightarrow \mathsf{B} \rightarrow (\neg\mathsf{A} \rightarrow \neg\mathsf{B} \rightarrow \mathsf{A})$$

   This sentence can be taken to be an axiom of propositional logic (defining proofs by contradiction). In our grammar above this produces the following parse tree:



2. (40pt) For each of the following languages L, prove whether L is regular, context-free but not regular, or not context-free:

   (a) $L = \{w^R x w : w \in \{\mathtt{a},\mathtt{b}\}^* \text{ and } x \in \{\mathtt{a},\mathtt{b}\}^+\}$.

   Regular.

   Because we can let $w = \varepsilon$, the language includes $\{\mathtt{a},\mathtt{b}\}^+$. The case where $w \neq \varepsilon$ is then included among the strings in $\{\mathtt{a},\mathtt{b}\}^+$. Thus, the language is precisely $(\mathtt{a} \cup \mathtt{b})^+$.

   (b) $L = \{wx : 2|w| = 3|x| \text{ and } w, x \in \mathtt{0}^+\mathtt{1}^+\}$.

   Context-free not regular.

   Notice that both the w region and the x region are of the form $\mathtt{0}^p\mathtt{1}^{p'}$, $p, p' > 0$, so the w region and x region are clearly identified for every string in the language.

   L is context-free because it can be accepted by a PDA that pushes two characters for each $\mathtt{0}$ or $\mathtt{1}$ in w and pops three characters for each $\mathtt{0}$ or $\mathtt{1}$ in x.

This can be made deterministic with four states, where the first state q_0 pushes two characters while reading 0's, moves to the second state q_1 when it reads a 1 and pushes two characters, then pushes two characters while it continues to read 1's. When it reads a 0, the w region has finished and the x region has started, so it pops three characters and moves to a third state q_2, where it pops three characters for each 0 it reads. When it starts reading 1's it moves to the final, accepting, state q_3 and pops three characters, and pops three more characters for each 1 read. If the stack ends up empty in q_3, the string read will be in L.

To show that L is not regular we use pumping. For the pumping argument let $w = \mathtt{0}^{3k}\mathtt{1}^{3k}\mathtt{0}^{2k}\mathtt{1}^{2k}$. Then $y = \mathtt{0}^p$ for some $0 < p \leq k$. Pumping out we get $w' = \mathtt{0}^{3k-p}\mathtt{1}^{3k}\mathtt{0}^{2k}\mathtt{1}^{2k}$. Since the condition for $wx \in L$ (w here not the w from the pumping theorem) forces $w = \mathtt{0}^{3k-p}\mathtt{1}^{3k} \in \mathtt{0}^+\mathtt{1}^+$ and $x = \mathtt{0}^{2k}\mathtt{1}^{2k} \in \mathtt{0}^+\mathtt{1}^+$, this string no longer satisfies the condition $2|w| = 3|x|$, so it is not in L.

(c) $L = \{wxx : |w| = 2|x| \text{ and } w \in \{\mathtt{a}, \mathtt{b}\}^* \text{ and } x \in \{\mathtt{c}\}^*\}$.

Context-free but not regular.

Because $x \in \{\mathtt{c}\}^*$, xx is also a string in $\{\mathtt{c}\}^*$ of twice the length, and the length of xx must be even, in which case $|w|$ must be even too. Thus,

$$L = \{wx : |w| = |x|, |x| \textbf{ is even}, w \in \{\mathtt{a}, \mathtt{b}\}^* \textbf{ and } x \in \{\mathtt{c}\}^*\}.$$

Notice that w contains only a's and b's, which must come before x, which has only c's.

This language can be accepted by a PDA with three states. The start state q_0 tracks whether $|w|$ is even, a second state q_1 tracks whether $|w|$ is odd. Both q_0 and q_1 push a character on the stack for each a or b read and move to the other state. If the machine reads a c from state q_0 it pops a character from the stack and moves to a third, accepting, state q_2, which pops one character for each c read. If the input string has been read and the stack is empty in q_2 the in the input string is in L. This is a deterministic PDA.

L can be shown to not be regular using a pumping argument on $w = \mathtt{a}^{2k}\mathtt{c}^{2k}$, which is in L because the a and c regions are even and equal length. Pumping in or out then breaks the requirement that the number of a's and b's is the same as the number of c's.

3. (30pt) Show that the following problem is decidable: Given a FSM M and a PDA P, is $L(M) \cap L(P)$ infinite? **Using the procedure** intersectPDAandFSM(M_1**: PDA**, M_2**: FSM**) **we can construct a PDA** $P' =$ intersectPDAandFSM(P, M) **that accepts** $L(P') = L(M) \cap L(P)$**.**

The problem now is to test whether L' is infinite we need to convert P' into an equivalent context-free grammar. This requires two steps. First we use convertPDAtorestricted(M**: PDA**) **to convert P' to P'', a PDA in restricted normal form. Then we use** buildgrammar(M**: PDA in restricted normal form**) **to convert P'' into a context-free grammar G. You could do these steps together with** PDAtoCFG(M**: PDA**) **applied to P'.**

Finally, we return decideCFLinfinite(G)**.**