COMS W4115 Fall 2020: Homework Assignment 2

Programming Languages and Translators

Deadline: Monday, November 2, 2020 by 11:59 PM

Overview

This homework assignment will test your knowledge of context-free grammars (CFGs) and parsing, which you have learned in class. Please submit your assignment via Gradescope by the deadline. This is an *individual* assignment, and all answers must be *typed*. However, you are allowed to hand-draw any parse trees and scan and attach them in your final submission. Finally, you must adhere to the academic integrity policies of the course. In all context-free grammars, assume that S is the start state unless otherwise specified.

Total Points: 85
Number of Problems: 5

Problems

1. **(8 Points)** For each of the following context-free grammars, state the number of valid inputs supported by that grammar. As an example, the grammar $G : E \to a \mid b \mid c$ results in 3 valid inputs—a, b, and c. Note that the empty string may be a valid input wherever applicable. You also do not need to list the different possible inputs.

   (a) **(4 Points)**

$$\begin{gathered} S \to A \\ A \to BBBwx \\ B \to CCCyz \\ C \to a \mid b \end{gathered}$$

   (b) **(4 Points)**

$$\begin{gathered} S \to D \\ D \to EEE \\ E \to FFF \\ F \to c \mid d \mid \epsilon \end{gathered}$$

2. **(8 Points)** Suppose there is a context-free grammar G such that the language of G, $L(G)$, supports palindrome inputs, where inputs consist of characters from the alphabet $\Sigma = \{a, c, e, r\}$.

   Examples of valid inputs for L:

   - $racecar$
   - a
   - The empty string

   Examples of invalid inputs for L:

   - $race$ (not a palindrome)
   - $abcba$ ($b \notin \Sigma$)
   - ack (composite of the first two cases)

   (a) **(6 Points)** Construct G.

   (b) **(2 Points)** Is L a regular language? Explain why or why not.

3. **(18 Points)** *Recursive descent* is a parsing technique that generates a parse tree using a top-down approach. The technique takes advantage of backtracking when there is a mismatch between a token and a production rule. However, recursive descent is not free of its limitations.

   For each of the following context-free grammars, identify whether it is possible to apply recursive descent parsing on all possible inputs.

   - If it is possible, clearly explain why it is possible (you do not need to provide a formal proof). Further, provide two non-trivial examples (*e.g.*, not the empty string) of valid inputs for the given CFG.
   - If it is not possible, clearly explain why it is not possible and provide an example of an input that would not be properly parsed by the given CFG using recursive descent. Finally, rewrite the production rules of the CFG (except for $S \to E$; leave it as is), using as few production rules as possible, such that recursive descent can be applied.

   (a) **(6 Points)**

$$\begin{gathered} S \to E \\ E \to XR \mid \epsilon \\ X \to \epsilon \mid X \\ R \to \text{int} * E \mid \epsilon \end{gathered}$$

   (b) **(6 Points)**

$$\begin{gathered} S \to E \\ E \to AB \mid \epsilon \\ A \to (A) \mid \text{int} \mid +E \\ B \to \text{int}\ B \mid * B \mid * \mid \text{int} \end{gathered}$$

   (c) **(6 Points)**

$$\begin{gathered} S \to E \\ E \to WXY \\ W \to a \mid Yb \\ X \to (W) \mid Z * Z \\ Y \to \epsilon \mid E + X \\ Z \to \text{int}\ Z \mid \text{int} \mid + \mid \epsilon \end{gathered}$$

4. **(26 Points)** Consider a context-free grammar G given by the following set of production rules:

$$S \to X$$
$$X \to X * X \mid X + X \mid (X) \mid \text{int}$$

Further, suppose we have an input stream of tokens given by $S = 8 * 2 + (10 * 4) + 9$.

(a) **(10 Points)** Using the CFG G, construct two parse trees for S.

(b) **(4 Points)** Let C be the number of unique parse trees that can be generated for S, given G. Find C.

(c) **(12 Points)** A CFG is considered *ambiguous* if there is more than one left-most derivation tree or more than one right-most derivation tree.

Is G an ambiguous grammar? If it is not ambiguous, please explain why not. If it is ambiguous, explain how you would resolve this ambiguity, and rewrite the production rules of G (except for $S \to X$; leave it as is), using as few production rules as possible, such that G becomes unambiguous.

5. **(25 Points)** Consider a simple markup language L that uses tags (similar to those in HTML and XML). To create tags, L supports the following tokens as terminal states: $\{<, >, /, =, word\}$.

   The following list provides the criteria for this language:

   - Every tag begins with $<$ and ends with $>$.
   - A tag may be an opening tag or a closing tag.
   - Inside an opening tag, the first token that follows $<$ is a word ($word$) representing the name of the tag. The tag also contains an optional list of *attributes*, which are pairs of words connected by $=$ (*i.e.*, $word = word$).
   - In a closing tag, the first token that follows $<$ is $/$, followed by the name of the tag (again, $word$). There are no attributes in the closing tag.
   - Every opening tag must be paired with a matching closing tag.
   - Any number of words or tags may appear between an opening tag and a closing tag.

   Examples of valid inputs for L:

   - $< word >< /word >$
   - $< word > word < /word >$
   - $< word >< word\ word = word > word\ word\ word < /word >< /word >$
   - $< word\ word = word\ word = word >< /word >$
   - $< word > word\ word < word\ word = word >< /word >< /word >$
   - $< word > word < word\ word = word > word < /word > word < /word >$

   Examples of invalid inputs for L:

   - $<>< / >$ (no presence of $word$ following $<$ in the opening tag and following $/$ in the closing tag)
   - $< word >$ (no matching closing tag)
   - $< word\ word =>< /word >$ (incomplete attribute)
   - $< word >< /word\ word = word >$ (closing tag contains attributes)

   This problem has several edge cases, so to simplify the problem, we have these additional assumptions:

   - Every tag must either be an opening tag or a closing tag; there are no other possibilities.
   - Assume that there always exists a space between two words; therefore, *wordword* is equivalent to *word word*. You do not have to worry about spaces.
   - Include \$, the character to indicate the end of the token stream, in your answers wherever applicable.
   - *word* is a single terminal, just like *int*; treat it as one token, not four individual character tokens.
   - There can be words outside of a set of opening and closing tags, as long as this set of tags is nested within a set of outer tags encapsulating the words (see valid examples 5 and 6 above).
   - It is valid to have an ambiguous grammar, as long as it is correct.
   - You do not have to worry about whether the opening and closing tags are the same.
   - Only consider the criteria explicitly defined in this problem; do not make assumptions related to real markup languages.

   (a) **(5 Points)** Construct a context-free grammar G for the language L.
   (b) **(8 Points)** Write the $FIRST$ and $FOLLOW$ sets for G.
   (c) **(10 Points)** Show the $LL(1)$ parsing table for G.
   (d) **(2 Points)** Is G an $LL(1)$ grammar? Explain why or why not.