CMPSC 461: Programming Language Concepts, Fall 2025
Assignment 2

Prof. Suman Saha

Due: 11:59 PM, September 19, 2025

General Instructions:
You need to submit your homework to **Gradescope**. Do **every problem on a separate page and mark** them before submitting. **If the questions are not marked** or are submitted with incorrect page-to-question mapping, the question will be **deducted partial points**. Make sure your name and PSU ID are legible on the first page of your assignment.

You are required to submit your assignments in typed format, please follow the latex/doc template (which can be found on Canvas) for the homework submission. Furthermore, please note that no handwritten submissions (of any form on paper or digital) will be accepted.

**(Kindly refer to the syllabus for late submission and academic integration policies.)

Assignment Specific Instructions:

1. The sample examples provided in the questions below are just for your reference and do not cover every possible scenario your solution should cover. Therefore, it is advised that you think through all the corner cases before finalizing your answer.
2. Students are expected to answer the questions in a way that shows their understanding of the concepts rather than just mentioning the answers. The rubric does contain partial points to encourage brief conceptual explanations.
3. Until specifically asked, you are not expected to generate an unambiguous Context-free Grammar.

Problem 1: Context Free Grammar I [$5 + 5 = 10$ pts]

1. Give a context-free grammar for the language below:

$$L : \{wcw^R \mid w \in \{a, b\}^* \text{ and } |w| \text{ is even}\}$$

where w^R is the reverse of w.

2. Give a context-free grammar for the language below:

$$L : \{a^i b^j c^k \mid i, j, k \geq 1 \text{ and } i + k = j\}$$

Solution

1.

$$S \to aaSaa \mid abSba \mid bbSbb \mid baSab \mid c$$

2. Note that $i + k = j$. This gives us $a^i b^{i+k} c^k$, which simplifies to $(ab)^i (bc)^k$ after rearranging the terms. We observe that any string in our language L is the concatenation of strings in two languages $L_1 : \{(ab)^i \mid i \geq 1\}$ and $L_2 : \{(bc)^k \mid k \geq 1\}$. This gives us

$$\begin{array}{lcl} S & \to & AC \\ A & \to & aAb \mid ab \\ C & \to & bCc \mid bc \end{array}$$

Problem 2: Context Free Grammar II and BNF,EBNF $[4 + 3 + 3 = 10$ pts$]$

1. Consider the language

$$L = \{w^p x^{2n} y^{3n} z^m \mid p > m, \text{ and } m, n \geq 1\}.$$

m,n,p are integers.

   (a) What is the shortest string in L if m is a multiple of 3?

   (b) Write a context-free grammar to generate L.

2. Convert the following BNF of a (simplified) C function call to an EBNF

```
    <func_call>  ::= id ( ) | id ( <param_list> )
    <param_list> ::= <expr> | <param_list> , <expr>
```

3. Convert the following EBNF of a (simplified) C variable declaration to a BNF

```
    <declaration> ::= [static] <type> <id> { , <id> }
```

Solution

1. (a) Generate with m=3, p=4, n = 1: wwwwxxyyyzzz

   (b)

$$\begin{aligned} S &\rightarrow wS \mid wB \\ B &\rightarrow wBz \mid wAz \\ A &\rightarrow xxAyyy \mid xxyyy \end{aligned}$$

2. $< func\_call >::= id([< param\_list >])$
   $< param\_list >::=< expr > \{, < expr >\}$

3. Here, we need to create another non-terminal to deal with zero or more $< id >$
   $< declaration >::= static < type >< id\_list > \mid < type >< id\_list >$
   $< id\_list >::=< id > \mid < id\_list >, < id >$

Problem 3: Ambiguity [6 + 4 = 10 pts]

For each of the following grammars, determine whether it is ambiguous or unambiguous. If the grammar is ambiguous, explain with an example string demonstrating the ambiguity. Additionally, provide an equivalent unambiguous grammar. If the grammar is unambiguous, specify any precedence and associativity rules it enforces, where applicable.

1. `<stmt> ::= if <expr> then <stmt> | if <expr> then <stmt> else <stmt> | <other_stmt>`

2. `E → E + E | E * E | E ^ E | id | ( E )`

Solution

1. (6 pts) This is the classic "dangling else" problem. An example of an ambiguous string is: `if E1 then if E2 then S1 else S2` i.e `if E1 then (if E2 then S1) else S2` or `if E1 then (if E2 then S1 else S2)`. The grammar is ambiguous, and there are multiple solutions to resolve the ambiguity. One solution is to match every else with the closest unmatched if.

```
<stmt> ::= <matched_stmt> | <unmatched_stmt>
<matched_stmt> ::= if <expr> then <matched_stmt> else <matched_stmt>
| <other_stmt>
<unmatched_stmt> ::= if <expr> then <stmt>
| if <expr> then <matched_stmt> else <unmatched_stmt>
```

2. (4 pts) The grammar is ambiguous since it ignores operator precendence. For example, the string `id + id * id` could be matched as `(id + id) * id` or `id + (id * id)`. To resolve ambiguity, we enforce an order of precedence in the grammar.

```
E → E + T | T
T → T * F | F
F → P ^ F | P
P → id | ( E )
```

Problem 4: Ambiguity and Derivation [$4 + 3 + 3 = 10$ pts]

1. Consider the following grammar for a markup language:

   ```
   S -> <tag>S</tag> | text | SS
   ```

   Derive the string $< b > hello < /b >< i > world < /i >$ using this grammar. Show the parse tree along with the corresponding leftmost and rightmost derivations.

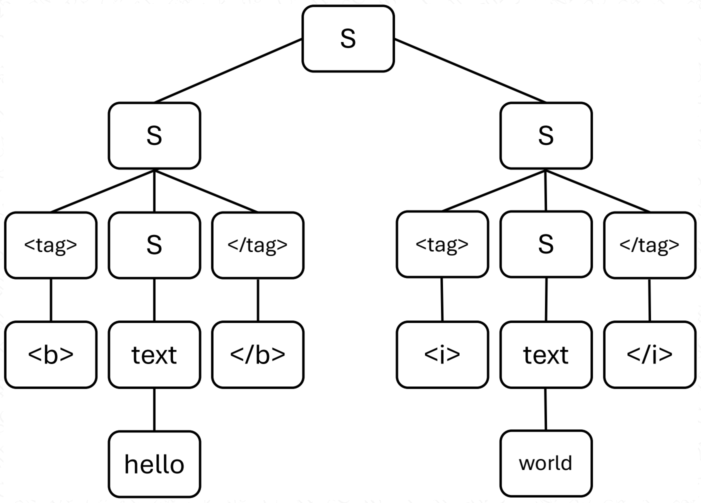
2. **Claim**: Given a grammar G and two different derivations for a string s, it is not sufficient to conclude that the grammar is ambiguous. It requires two different left-most or right-most derivations.

   (a) Substantiate the claim above and explain why it is important to have two different left-most or right-most derivations and not simply two different derivations. Use the grammar $G = \{S \to A + A; A \to a\}$ and the string $s = a + a$ as an example.

   (b) Consider the language $L = \{a^i b^j c^k | i = j \text{ or } j = k, \text{ and } i, j, k \geq 0\}$. The grammar $G' = \{S \to AC|DB; A \to aAb|\epsilon; C \to cC|\epsilon; D \to aD|\epsilon; B \to bBc|\epsilon\}$ generates this language. Determine if this grammar ambiguous. If it is, justify your answer with a specific string.

Solution

1. (4 pts) Parse tree for $< b > hello < /b >< i > world < /i >$:



   Left-most derivation: $S \Rightarrow SS \Rightarrow < tag > S < /tag > \ S \Rightarrow < b > S < /b > \ S \Rightarrow < b > text < /b > \ S \Rightarrow < b > hello < /b > \ S \Rightarrow < b > hello < /b > < tag > S < /tag > \Rightarrow < b > hello < /b > < i > S < /i > \Rightarrow < b > hello < /b > < i > text < /i > \Rightarrow < b > hello < /b > < i > world < /i >$

   Right-most derivation: $S \Rightarrow SS \Rightarrow S \ < tag > S < /tag > \Rightarrow S \ < i > S < /i > \Rightarrow S \ < i > text < /i > \Rightarrow S \ < i > world < /i > \Rightarrow < tag > S < /tag > < i > world < /i > \Rightarrow < b > S < /b > < i > world < /i) \Rightarrow < b > text < /b > < i > world < /i > \Rightarrow < b > hello < /b > < i > world < /i >$

2. (a) (3 pts) The distinction between two different left-most/right-most derivations and two different derivations is important as the former implies that a grammar assigns different parse trees for the same string. For any given parse tree, there should be exactly one left-most or right-most derivation for it to be unambiguous. For the provided grammar $G = \{A \to A + A; A \to a\}$ and string $s = a + a$, we have the following derivations:

$$D_{left} : S \Rightarrow A + A \Rightarrow a + A \Rightarrow a + a$$
$$D_{right} : S \Rightarrow A + A \Rightarrow A + a \Rightarrow a + a$$

For the two different left-most and right-most derivations above, we observe that there is only one choice at each step for the left-most and right-most non-terminals. Therefore, since we only have one left-most/right-most derivation, we have one parse tree. The grammar G has multiple derivations but is unambiguous.

(b) (3 pts) The grammar G' is ambiguous. Strings that are generated when $i = j = k$ can have multiple parse trees. For example, consider the string abc. Two leftmost derivations are

$$D_1 : S \Rightarrow AC \Rightarrow aAbC \Rightarrow abC \Rightarrow abcC \Rightarrow abc$$
$$D_2 : S \Rightarrow DB \Rightarrow aDB \Rightarrow aB \Rightarrow abBc \Rightarrow abc$$

Problem 5: Regular Expression to CFG [5 + 5 = 10 pts]

For each of the following regular expressions, design a context-free grammar (CFG) that recognizes the corresponding regular language. Please note that your derived grammar should not accept any strings not recognized by the corresponding regular expression.

1. `[a-zA-Z_][a-zA-Z0-9_]*`
   This regular expression describes an identifier in a programming language. For example abc, $\_a9$, $\_\_a\_b7$, etc.. It rejects identifier names starting with numbers i.e 82_a, $931b$, etc.

2. `(http|https)://(www\.)?[a-zA-Z0-9]+\.(com|org|edu)`
   This regular expression describes a simplified Uniform Resource Identifier (URI). For example, `https://www.google.com`

Solution

1. (5 pts) We observe that the regex is a concatenation of two parts. The first part is the starting character. The rest are the other characters in the variable name. The following CFG recognizes the regex

```
<identifier> -> <start_char> <other_chars>
<other_chars> -> <char> <other_chars> | ϵ
<start_char> -> <letter> | _
<char> -> <letter> | <digit> | _
<letter> -> a | b | c | ... | z | A | B | C | ... | Z
<digit> -> 0 | 1 | ... | 9
```

2. (5 pts) The URI consists of four parts: the scheme (http/https), an optional "www.", the domain (alphanumeric), and a top level domain (".com", ".org", or ".edu"). The following CFG recognizes the regex

```
<uri> -> <scheme> "://" <www> <domain> "." <tld>
<scheme> -> http | https
<www> -> www. | ϵ
<domain> -> <domain_start> <domain_other>
<domain_other> -> <domain_start> <domain_other> | ϵ
<domain_start> -> letter | digit
<tld> -> .com | .org | .edu
<letter> -> a | b | c | ... | z | A | B | C | ... | Z
<digit> -> 0 | 1 | ... | 9
```