CSE340 FALL 2020 - HOMEWORK 2

Due: Monday September 28, 2020 by 11:59 PM on Canvas

All submissions should be typed, no exceptions.

1. Consider the grammar.

S → C B b	(1)
A → a A	(2)
A → ε	(3)
B → d B e	(4)
B → & A B C !	(5)
B→ C	(6)
C → c C a	(7)
C → ε	(8)

where S, A, B, C and D are the non-terminals, S is the start symbol, and a, b , c , f , g , '&' and '!' are the terminals.

Solution:

1.1. **FIRST sets**. Do the following

1.1.1. Do an initialization pass by applying FIRST sets rules I and II. Show the resulting FIRST sets

FIRST(a) = {a}
FIRST(b) = {b}
FIRST(c) = {c}
FIRST(d) = {d}
FIRST(e) = {e}
FIRST(&) = {&}
FIRST(!) = {!}
FIRST(ε) = {ε}
FIRST(S) = {}
FIRST(A) = {}
FIRST(B) = {}
FIRST(C) = {}

1.1.2. Do one pass on the grammar rules in the order in which they are listed as follows. For each grammar rules, apply FIRST set rule III, then apply FIRST set rule IV then apply FIRST set rule V. Show the resulting FIRST sets after this one pass.

FIRST(b) = {b}
FIRST(c) = {c}
FIRST(d) = {d}
FIRST(e) = {e}
FIRST(&) = {&}
FIRST(!) = {!}
FIRST(ε) = {ε}
FIRST(S) = {}

FIRST(A) = {a, ε}
FIRST(B) = {d, &}
FIRST(C) = {c, ε}

1.1.3. Show the final result of the calculation of the FIRST sets

FIRST(b) = {b}
FIRST(c) = {c}
FIRST(d) = {d}
FIRST(e) = {e}
FIRST(&) = {&}
FIRST(!) = {!}
FIRST(ε) = {ε}
FIRST(S) = {c, d, &, b}
FIRST(A) = {a, ε}
FIRST(B) = {d, &, c, ε}
FIRST(C) = {c, ε}

1.2. **FOLLOW sets**. Do the following

1.2.1. Do an initialization pass by applying FOLLOW set rules I. Then do one pass by applying FOLLOW sets rules IV, and V. Show the resulting FOLLOW sets

FOLLOW(S) = {$}
FOLLOW(A) = {d, &, c, !}
FOLLOW(B) = {b, e, c, !}
FOLLOW(C) = {d, &, c, b, !, a}

1.2.2. Do one pass on the grammar rules in the order in which they are listed as follows. For each grammar rules, apply FOLLOW set rule II, then apply FOLLOW set rule III. Show the resulting FOLLOW sets after this one pass.

FOLLOW(S) = {$}
FOLLOW(A) = {d, &, c, !}
FOLLOW(B) = {b, e, c, !}
FOLLOW(C) = {d, &, c, b, !, a, e}

1.2.3. Show the final result of the calculation of the FOLLOW sets

FOLLOW(S) = {$}
FOLLOW(A) = {d, &, c, !}
FOLLOW(B) = {b, e, c, !}
FOLLOW(C) = {d, &, c, b, !, a, e}

2. Consider the grammar

$S \rightarrow A\ B \mid C\ D$

$A \rightarrow a\ A\ b \mid \varepsilon$

$B \rightarrow D\ C \mid A$

$C \rightarrow b\ B\ c \mid \varepsilon$

$D \rightarrow b \mid \varepsilon$

Show that this grammar does not have a recursive descent predictive parser. To get full credit you should show all the conditions of predictive parsing that are not satisfied by this grammar. You do not need to show the conditions of predictive parsing that are satisfied. You will need to calculate FIRST and FOLLOW sets but you do not need to show how you calculated them. You will need to explicitly show which conditions of predictive parsing are not satisfied.

Solution:

The first and follow sets are as follows:

FIRST SETS:

FIRST(a)	{a}
FIRST(b)	{b}
FIRST(c)	{c}
FIRST(ε)	{ε}
FIRST(S)	{a, b, ε}
FIRST(A)	{a, ε}
FIRST(B)	{b, ε, a}
FIRST(C)	{b, ε}
FIRST(D)	{b, ε}

FOLLOW SETS:

FOLLOW(S)	{EOF}
FOLLOW(A)	{b, a, EOF, c}
FOLLOW(B)	{c, EOF}
FOLLOW(C)	{b, EOF, c}

FOLLOW(D)	{b, EOF, c}

To prove that the given grammar has a predictive recursive descent parser, it should satisfy the following conditions

1. If A → α and A → β are two grammar rules, then FIRST(α) ∩ FIRST(β) = ∅
2. If ε ∈ FIRST(A), then FIRST(A) ∩ FOLLOW(A) = ∅

Condition 1:

If the non-terminal symbol appears on the left-hand side *of only one rule*, condition 1 is immediately satisfied. We consider non-terminals that have more than one rule. These non-terminals are S, A, B, C and D.

a) Non-terminal S:

$S \rightarrow A\ B$
$S \rightarrow C\ D$

FIRST(AB) = {a, b, ε}
FIRST(CD) = {b, ε}

FIRST(AB) ∩ FIRST(CD) ≠ ∅

So, condition 1 is not satisfied for S.

b) Non-terminal A:

$A \rightarrow a\ A\ b$
$A \rightarrow \varepsilon$

FIRST(aAb) = {a}
FIRST(ε) = {ε}

FIRST(aAb) ∩ FIRST(ε) = ∅

So, condition 1 is satisfied for A.

c) Non-terminal B:

$B \rightarrow D\ C$
$B \rightarrow A$

FIRST(DC) = {b, ε}
FIRST(A) = {a, ε}

FIRST(DC) ∩ FIRST(A) ≠ ∅

So, condition 1 is not satisfied for B.

d) Non-terminal C:

$C \rightarrow b\ B\ c$
$C \rightarrow \varepsilon$

FIRST(bBc) = {b}
FIRST(ε) = {ε}

FIRST(bBc) ∩ FIRST(ε) = ∅

So, condition 1 is satisfied for C.

e) Non-terminal D:

$D \rightarrow b$
$D \rightarrow \varepsilon$

$FIRST(b) = \{b\}$
$FIRST(\varepsilon) = \{\varepsilon\}$

$FIRST(b) \cap FIRST(\varepsilon) = \varnothing$

Hence, condition 1 is satisfied for D.

We have shown that the condition 1 is satisfied only for non-terminals A, C and D but not satisfied for non-terminals S and B. We now consider the second condition.

Condition 2:

For every non-terminal X such that $\varepsilon \in FIRST(X)$, we need to show that $FIRST(X) \cap FOLLOW(X) = \varnothing$. The non-terminals that have ε in their first sets are S, A, B, C and D.

$FIRST(S) \cap FOLLOW(S) = \{a, b, \varepsilon\} \cap \{EOF\} = \varnothing$
$FIRST(A) \cap FOLLOW(A) = \{a, \varepsilon\} \cap \{b, a, EOF, c\} = \{a\}$, which is not empty
$FIRST(B) \cap FOLLOW(B) = \{b, \varepsilon, a\} \cap \{c, EOF\} = \varnothing$
$FIRST(C) \cap FOLLOW(C) = \{b, \varepsilon\} \cap \{b, EOF, c\} = \{b\}$, which is not empty
$FIRST(D) \cap FOLLOW(D) = \{b, \varepsilon\} \cap \{b, EOF, c\} = \{b\}$, which is not empty

As highlighted above, the condition 2 is not satisfied for non-terminals A, C and D.

Therefore, this grammar doesn't satisfy both conditions 1 and 2. Given that the grammar doesn't satisfy both conditions for the existence of predictive recursive descent parser, the grammar doesn't have a predictive recursive descent parser.

3. Consider the grammar

$S \rightarrow A\ c\ B \mid e\ C\ A$

$A \rightarrow a \mid B\ D$

$B \rightarrow b\ B \mid \varepsilon$

$C \rightarrow c\ C \mid \varepsilon$

$D \rightarrow d\ D \mid \varepsilon$

3.1. Show that the grammar has a predictive recursive descent parser. You should show that the conditions of predictive parsing apply for every non-terminal.

Answer:

The first and follow sets are as follows:

FIRST SETS:

FIRST(a)	{a}
FIRST(b)	{b}

FIRST(c)	{c}
FIRST(d)	{d}
FIRST(e)	{e}
FIRST(ε)	{ε}
FIRST(S)	{e, a, b, d, c}
FIRST(A)	{a, b, d, ε}
FIRST(B)	{b, ε}
FIRST(C)	{c, ε}
FIRST(D)	{d, ε}

FOLLOW SETS:

FOLLOW(S)	{EOF}
FOLLOW(A)	{c, EOF}
FOLLOW(B)	{d, EOF, c}
FOLLOW(C)	{a, b, d, EOF}
FOLLOW(D)	{c, EOF}

To prove that the given grammar has a predictive recursive descent parser, it should satisfy the following conditions

1. If A → α and A → β are two grammar rules, then FIRST(α) ∩ FIRST(β) = ∅
2. If ε ∈ FIRST(A), then FIRST(A) ∩ FOLLOW(A) = ∅

Condition 1:

If the non-terminal symbol appears on the left-hand side of only one rule, condition 1 is immediately satisfied. We consider non-terminals that have more than one rule. These non-terminals are S, A, B, C and D.

a) Non-terminal **S**

S → A c B
S → e C A

FIRST(AcB) = {a, b, d, c}
FIRST(eCA) = {e}

FIRST(AcB) ∩ FIRST(eCA) = ∅
Hence, condition 1 is satisfied for S

b) Non-terminal **A**

A → a

$A \rightarrow B\ D$

$FIRST(a) = \{a\}$
$FIRST(BD) = \{b, d, \varepsilon\}$

$FIRST(a) \cap FIRST(BD) = \varnothing$
Hence, condition 1 is satisfied for A.

c) Non-terminal **B**

$B \rightarrow b\ B$
$B \rightarrow \varepsilon$

$FIRST(bB) = \{b\}$
$FIRST(\varepsilon) = \{\varepsilon\}$

$FIRST(bB) \cap FIRST(\varepsilon) = \varnothing$
Hence, condition 1 is satisfied for B

d) Non-terminal **C**

$C \rightarrow c\ C$
$C \rightarrow \varepsilon$

$FIRST(cC) = \{c\}$
$FIRST(\varepsilon) = \{\varepsilon\}$

$FIRST(cC) \cap FIRST(\varepsilon) = \varnothing$
Hence, condition 1 is satisfied for C

e) Non-terminal **D**

$D \rightarrow d\ D$
$D \rightarrow \varepsilon$

$FIRST(dD) = \{d\}$
$FIRST(\varepsilon) = \{\varepsilon\}$

$FIRST(dD) \cap FIRST(\varepsilon) = \varnothing$
Hence, condition 1 is satisfied for D

We have shown that the grammar satisfies the first condition for predictive parsing. Now, we consider the second condition.

Condition 2:

If $\varepsilon \in FIRST(A)$, then $FIRST(A) \cap FOLLOW(A) = \varnothing$

For every non-terminal such that $\varepsilon \in FIRST(A)$, we need to show that $FIRST(A) \cap FOLLOW(A) = \varnothing$. The non-terminals that have ε in their first sets are A, B, C and D.

$FIRST(A) \cap FOLLOW(A) = \{a, b, d, \varepsilon\} \cap \{c, EOF\} = \varnothing$

$FIRST(B) \cap FOLLOW(B) = \{b, \varepsilon\} \cap \{d, EOF, c\} = \varnothing$

$FIRST(C) \cap FOLLOW(C) = \{c, \varepsilon\} \cap \{a, b, d, EOF\} = \varnothing$

$FIRST(D) \cap FOLLOW(D) = \{d, \varepsilon\} \cap \{c, EOF\} = \varnothing$

As shown above, condition 2 is satisfied for non-terminals A, B, C and D. Therefore, this grammar satisfies condition 2.

Given that the grammar satisfies both conditions for existence of a predictive recursive descent parser, the grammar has a predictive recursive descent parser.

3.2. Write parse_input()

```
void parse_input()
{
      parse_S();
      lexer.expect(EOF);
}
```

3.3. Write parse_S()

```
void parse_S()
{  // S -> A c B
   // S -> e C A
   // FIRST(A c B) = { a, b, d, c}
   // FIRST(e C A) = { e }

   Token t = lexer.peek(1);
   if ((t.token_type == a_type)||(t.token_type == b_type) ||
       (t.token_type == d_type)||(t.token_type == c_type))    // S -> A c B
   {
        parse_A();
        lexer.expect(c_type);
        parse_B();
   }
   else if (t.token_type == e_type)  // S -> e C A
   {
        lexer.expect(e-type);
        parse_C();
        parse_A();
   }
   else
        syntax_error();
}
```

3.4. Write parse_A().

```
void parse_A()
{  // A -> a
   // A -> B D
   // FIRST(a) = { a }
   // FIRST(B D) = { b, d, ε }
   // FOLLOW(A) = { c, EOF}

   Token t = lexer.peek();
   if (t.token_type == a_type)     // A -> a
   {
        lexer.expect(a_type);
   }
   else if ((t.token_type == b_type) || (t.token_type == d_type))  // A -> B D
   {
        parse_B();
        parse_D();
   }
   else if ((t.token_type == c_type) || // Since, A generates epsilon, if
            (t.token_type == EOF))      // token in FOLLOW(A) parse rule
                                        // that generates epsilon: A -> B D
   {
        parse_B();
        parse_D();
   }
   else
      syntax_error();
}
```

3.5. Give a **full execution** trace for your parser from part 3.2 above on input c b d c

Your parser should follow the general model of predictive parser that we saw in class. In particular, for non-terminals that can generate ε, the parser should check the FOLLOW set before choosing to parse the righthand side that generates ε.

Solution 3.3

```
parse_input()
      parse_S()
            peek()   // next token is c
            parse_A()
                  peek()                    // next token is c
                  parse_B()
                       peek()               // next token is c
                       return;              // B -> epsilon
                  parse_D()
                        peek()              // next token is c
```

```
                    return;                 // D -> epsilon
            expect(c-type)                  // c consumed
            parse_B()
                  peek()                    // next token is b
                  expect(b-type)            // b consumed
                  parse_B()
                        peek()              // next token is d
                        return;             // B -> epsilon
      expect(EOF)                           // next token is d but EOF expected
            syntax_error()                  // Hence, syntax_error
```

4. We say that a symbol of a grammar is useless if it does not appear in the derivation of a string (possibly empty) of terminals. Formally, *A is useful* if there exists a derivation S ⇒* x A y ⇒* w where w is a string of terminals. *A is useless* if it is not useful. Note that a useful symbol can be either a terminal or a non-terminal. Also, a useless symbol can be a terminal or a non-terminal.

   Consider the grammar

   S → A b B | D c A

   A → a B

   B → b A | A E

   E → e D | ε

   C → c A | E

   D → d D | A C D

   Which are the useful symbols of this grammar? For every useful symbol, give a derivation that shows that the symbol is useful.

   Solution:

   Useful symbols are those symbols that appear in derivations of strings of terminals (including the empty string). In the given grammar, there is no derivation starting from S that results in a string of terminals. It follows that S is useless and all other symbols are useless.

   To see why S cannot generate any string of terminals, we show that A cannot generate a string of terminals. Since both righthand sides of the rules for S have A, it follows that S itself cannot generate a string of terminals and no string of terminals is generated by this grammar. So, the whole grammar is useless.

   We show that A does not derive a string of terminals. A derivation starting from A will have to start in one of the following two ways (these are all the possibilities):

   - A => a B => a b **A**
   - A => a B => **A** E

   If we continue with the derivations, the **A** in red, will result in another A in two steps and there is no way to "get rid" of A in the derivation

   In summary, there is no derivation starting from S that generates a string of terminals, so all symbols are useless.