CSE 340 Spring 2019

Homework 2 Solution

Consider the grammar.

$S \rightarrow A\ g\ B\ C$ (1)
$A \rightarrow a\ A$ (2)
$A \rightarrow C\ B$ (3)
$B \rightarrow d\ B\ c$ (4)
$B \rightarrow \varepsilon$ (5)
$C \rightarrow f\ C$ (6)
$C \rightarrow \varepsilon$ (7)

where S, A, B, and C are the non-terminals, S is the start symbol, and a, c, d, e, f, and g are the terminals.

1.

- **Calculating FIRST sets**:

Initialization:

FIRST(a) = {a}
FIRST(c) = {c}
FIRST(d) = {d}
FIRST(f) = {f}
FIRST(g) = {g}
FIRST(ε) = {ε}
FIRST(S) = {}
FIRST(A) = {}
FIRST(B) = {}
FIRST(C) = {}

Pass 1:

FIRST(a) = {a}
FIRST(c) = {c}
FIRST(d) = {d}
FIRST(f) = {f}
FIRST(g) = {g}
FIRST(ε) = {ε}
FIRST(S) = {}
FIRST(A) = $\{a^{1, (2), \mathrm{III}}\}$
FIRST(B) = $\{d^{2, (4), \mathrm{III}}, \varepsilon^{3, (5), \mathrm{V}}\}$
FIRST(C) = $\{f^{4, (6), \mathrm{III}}, \varepsilon^{5, (7), \mathrm{V}}\}$

Note. In your solutions, you are not expected to use the superscript notation. You are only expected to give the contents of each set.

Pass 2:

FIRST(a) = {a}

FIRST(c) = {c}

FIRST(d) = {d}

FIRST(f) = {f}

FIRST(g) = {g}

FIRST(ε) = {ε}

FIRST(S) = $\{a^{6,\,(1),\,\mathrm{III}}\}$

FIRST(A) = $\{a^{1,\,(2),\,\mathrm{III}},\ f^{\,7,\,(3),\,\mathrm{III}},\ d^{8,\,(3),\,\mathrm{IV}},\ \varepsilon^{9,\,(3),\,\mathrm{V}}\}$

FIRST(B) = $\{d^{2,\,(4),\,\mathrm{III}},\ \varepsilon^{3,\,(5),\,\mathrm{V}}\}$

FIRST(C) = $\{f^{\,4,\,(6),\,\mathrm{III}},\ \varepsilon^{5,\,(7),\,\mathrm{V}}\}$

Pass 3:

FIRST(a) = {a}

FIRST(c) = {c}

FIRST(d) = {d}

FIRST(f) = {f}

FIRST(g) = {g}

FIRST(ε) = {ε}

FIRST(S) = $\{a^{6,\,(1),\,\mathrm{III}},\ f^{\,10,\,(1),\,\mathrm{III}},\ d^{11,\,(1),\,\mathrm{III}},\ g^{12,\,(1),\,\mathrm{IV}}\}$

FIRST(A) = $\{a^{1,\,(2),\,\mathrm{III}},\ f^{\,7,\,(3),\,\mathrm{III}},\ d^{8,\,(3),\,\mathrm{IV}},\ \varepsilon^{9,\,(3),\,\mathrm{V}}\}$

FIRST(B) = $\{d^{2,\,(4),\,\mathrm{III}},\ \varepsilon^{3,\,(5),\,\mathrm{V}}\}$

FIRST(C) = $\{f^{\,4,\,(6),\,\mathrm{III}},\ \varepsilon^{5,\,(7),\,\mathrm{V}}\}$

Pass 4:

FIRST(a) = {a}

FIRST(c) = {c}

FIRST(d) = {d}

FIRST(f) = {f}

FIRST(g) = {g}

FIRST(ε) = {ε}

FIRST(S) = $\{a^{6,\,(1),\,\mathrm{III}},\ f^{\,10,\,(1),\,\mathrm{III}},\ d^{11,\,(1),\,\mathrm{III}},\ g^{12,\,(1),\,\mathrm{IV}}\}$

FIRST(A) = $\{a^{1,\,(2),\,\mathrm{III}},\ f^{\,7,\,(3),\,\mathrm{III}},\ d^{8,\,(3),\,\mathrm{IV}},\ \varepsilon^{9,\,(3),\,\mathrm{V}}\}$

FIRST(B) = $\{d^{2,\,(4),\,\mathrm{III}},\ \varepsilon^{3,\,(5),\,\mathrm{V}}\}$

FIRST(C) = $\{f^{\,4,\,(6),\,\mathrm{III}},\ \varepsilon^{5,\,(7),\,\mathrm{V}}\}$

We find that there is no change in Pass 4, so we conclude that our FIRST set is as follows:

FIRST(a) = {a}
FIRST(c) = {c}
FIRST(d) = {d}
FIRST(f) = {f}
FIRST(g) = {g}
FIRST(ε) = {ε}
FIRST(S) = {a, d, f, g}
FIRST(A) = {ε, a, d, f}
FIRST(B) = {ε, d}
FIRST(C) = {ε, f}

• **Calculating FOLLOW sets**:

Initialization:

FOLLOW(S) = $\{\text{eof}^{1, (1), \text{I}}\}$
FOLLOW(A) = {}
FOLLOW(B) = {}
FOLLOW(C) = {}

Pass 1:

FOLLOW(S) = $\{\text{eof}^{1, (1), \text{I}}\}$
FOLLOW(A) = $\{\text{g}^{2, (1), \text{IV}}\}$
FOLLOW(B) = $\{\text{f}^{3, (1), \text{IV}}, \text{c}^{5, (4), \text{IV}}\}$
FOLLOW(C) = $\{\text{d}^{4, (3), \text{IV}}\}$

Pass 2:

FOLLOW(S) = $\{\text{eof}^{1, (1), \text{I}}\}$
FOLLOW(A) = $\{\text{g}^{2, (1), \text{IV}}\}$
FOLLOW(B) = $\{\text{f}^{3, (1), \text{IV}}, \text{c}^{5, (4), \text{IV}}, \text{eof}^{7, (1), \text{III}}, \text{g}^{8, (3), \text{II}}\}$
FOLLOW(C) = $\{\text{d}^{4, (3), \text{IV}}, \text{eof}^{6, (1), \text{II}}, \text{g}^{9, (3), \text{III}}\}$

Pass 3:

FOLLOW(S) = $\{\text{eof}^{1, (1), \text{I}}\}$
FOLLOW(A) = $\{\text{g}^{2, (1), \text{IV}}\}$
FOLLOW(B) = $\{\text{f}^{3, (1), \text{IV}}, \text{c}^{5, (4), \text{IV}}, \text{eof}^{7, (1), \text{III}}, \text{g}^{8, (3), \text{II}}\}$
FOLLOW(C) = $\{\text{d}^{4, (3), \text{IV}}, \text{eof}^{6, (1), \text{II}}, \text{g}^{9, (3), \text{III}}\}$

Since, there is no change during Pass 3, we find that the FOLLOW sets in the given grammar are as follows:

FOLLOW(S) = {eof}
FOLLOW(A) = {g}
FOLLOW(B) = {eof, c, f, g}
FOLLOW(C) = {eof, d, g}

2. Show that the grammar has a predictive recursive descent parser

A grammar has a recursive descent predictive parser if and only if the following two conditions hold:

I. If $A \rightarrow \alpha$ and $A \rightarrow \beta$, then $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$
II. If $\varepsilon \in FIRST(A)$, then $FIRST(A) \cap FOLLOW(A) = \emptyset$

So, we need to find the FIRST sets of the righthand sides of rules and the FOLLOW sets of non-terminals that can generate epsilon. We start by calculating FIRST and FOLLOW sets for non-terminals.

FIRST(S) = {a, d, f, g}
FIRST(A) = {ε, a, d, f}
FIRST(B) = {ε, d}
FIRST(C) = {ε, f}

FOLLOW(S) = {eof}
FOLLOW(A) = {g}
FOLLOW(B) = {eof, c, f, g}
FOLLOW(C) = {eof, d, g}

Checking for condition (1) for predictive parsing

(1) Rules for S:
There is only one rule starting with S. So, condition I holds for S.

(2) Rules starting with A: $A \rightarrow a\ A$ and $A \rightarrow C\ B$
FIRST(a A) = {a}
FIRST(C B) = {ε, d, f}
$FIRST(a\ A) \cap FIRST(C\ B) = \emptyset$
So, condition I is satisfied for the rules of A.

(3) Rules starting with B: $B \rightarrow d\ B\ c$ and $B \rightarrow$ epsilon
FIRST(d B c) = {d}
FIRST(ε) = {ε}
$FIRST(d\ B\ c) \cap FIRST(\varepsilon) = \emptyset$
So, condition I is satisfied for the rules of B.

(4) Rules starting with C: $C \rightarrow f\ C$ and $C \rightarrow \varepsilon$
FIRST(f C) = {f}

FIRST(ε) = {ε}
FIRST(f C) ∩ FIRST(ε) = ∅
So, condition I is satisfied for the rules of C.

Checking for condition (2) for predictive parsing

Non-terminal S: ε ∉ FIRST(S), **So, condition II holds for S.**

Non-terminal A: ε ∈ FIRST(A), so we should have FIRST(A) ∩ FOLLOW(A) = ∅
FIRST(A) = {ε, a, d, f}
FOLLOW(A) = {g}
FIRST(A) ∩ FOLLOW(A) = ∅
So, condition II holds for A.

Non-terminal B: ε ∈ FIRST(B), so we should have FIRST(B) ∩ FOLLOW(B) = ∅
FIRST(B) = {ε, d}
FOLLOW(B) = {eof, c, f, g}
FIRST(B) ∩ FOLLOW(B) = ∅
So, condition II holds for B.

Non-terminal C: ε ∈ FIRST(C), so we should have FIRST(C) ∩ FOLLOW(C) = ∅
FIRST(C) = {ε, f}
FOLLOW(C) = {eof, d, g}
FIRST(C) ∩ FOLLOW(C) = ∅
So, condition II holds for C.

This grammar satisfies both the conditions for predictive parsing and therefore it has a predictive parser.

3. Write the parser for the grammar.

Parser

```
Parse_Input()
{
    Parse_S();
    Token t = lexer.getToken();

    if(t.type == eof)
    {
        return;
    }
    else
    {
        syntax_error();
    }
}
```

```
Parse_S()
{
                                                    // We only have one rule of S
                                                    // so we prase right hand side A g B C directly
       Parse_A();                                   // we Parse_A() directly

       Token t1 = lexer.getToken();                 // We continue to getToken to check
       if(t1.type == g.type)                        // whether g_type
       {                                            // after successfully checking g_type
              Parse_B();                            // we parse B
              Parse_C();                            // then we parse C
          return;                                   // finish S -> A g B C
       }
       else                                         // if after parsing A, the token is not g
       {                                            // we have syntax error
              syntax_error();
       }
}

Parse_A()
{
       Token t = lexer.getToken();
       if(t.type == a.type)                         // if the next token is a, we should parse a A
       {
              Parse_A();                            // Call Parse_A() after we encounter "a"
                                                    // at this point we have seen a A of the
                                                    // righthand side a A
              return;
       }
       else if (t.type == f.type || t.type == d.type) // If next token is in FIRST(CB)-{ε} = {d, f}
       {
              lexer.ungetToken(t);                   //  we unget the token
              Parse_C();                             // and parse CB
              Parse_B();
              return;
       }
       else if (t.type == g.type)                   // if next token is in FOLLOW(A), we
       {                                            // we need to parse the rule that can generate epsilon, so
              lexer.ungetToken(t);                  // we unget the token
              Parse_C();                            // and parse CB
              Parse_B();
              return;
       }
       else
       {
              syntax_error();
       }
}
```

Note. The case of FIRST(CB)-{ε} and FOLLOW(A) can be combined in one condition to parse C B. This is what we do in the following functions.

```
Parse_B()
{
        Token t = lexer.getToken();
        if(t.type == d.type)                            // if next token is in  FIRST(d B C) = {d}
        {
               Parse_B();                               // we parse B then
               Token t1 = lexer.getToken();             // we make sure that the token after
               if(t1.type == c.type)                    // that is c
               {                                        // d B c on the righthand side
                       return;
               }
               else
               {
                       syntax_error();
               }
        }
        else if( t.type == g.type || t.type == c.type ||    // else if next token is in FOLLOW(B) =
                t.type == f.type || t.type == EOF)          // {EOF, g, c, f}
        {                                               // we parse the righthand side of the rule
                                                        // that can generate epsilon which is B -> ε
               lexer.ungetToken(t);                     // so we unget the token which is part of FOLLOW(B)
               return;                                  // and return
        }
        else
        {
               syntax_error();
        }
}

Parse_C()
{
        Token t = lexer.getToken();
        if(t.type == f.type)
        {
               Parse_C();
               return;
        }
        else if(t.type == d.type || t.type == g.type || t.type == EOF)
        {
               lexer.ungetToken(t);
               return;
        }
        else
        {
               syntax_error();
        }
}
```

Note: The above code either returns successfully or throws a syntax error. Alternatively, we could have had the parse functions return a Boolean value (true or false). In grading, we will count both as correct.

4. Give the execution trace for input cde

Execution trace of parser is as follows.

```
Parse_S()                                                     // We by calling parse_S(). At
                                                              // this point the remaining input
                                                              // is 'cde$'
      Parse_A()                                               //in parse_S(), we first
                                                              // call parse_A()
            Token t = lexer.getToken();                       // then we get a token
                                                              // t = c_type and the remaining
                                                              // input is 'de$'
            if(t.type == a_type)                              // the first condition evaluates
                                                              // to false
            else if(t.type == f_type || t.type == d_type)     // evaluates to false
            else if (t.type == g_type)                        // evaluates to false
            else                                              // so the remaining else is executed
                  syntax_error();                             // This will give us syntax error.
```