Todd Wenker

CSE 340

Professor Doupe

9/23/15

Homework #2

Question 1:

LSE 340 H.W 2

1. Compute FIRST and FOLLOW sets for the following gramma.

$S \rightarrow aAB \mid CD$
$A \rightarrow CD \mid SE \mid \epsilon$
$B \rightarrow aSB \mid DS$
$C \rightarrow cC \mid \epsilon$
$D \rightarrow DdC \mid \epsilon$
$E \rightarrow efg$
$F \rightarrow fg \mid \epsilon$

a. FIRST

S			✓
= { }	{ a }	{ a, c }	{ a, c, d, ε }
A			✓
= { }	{ ε }	{ a, c, ε }	{ a, c, d, e, ε }
B			✓
= { }	{ a }	{ a }	{ a, c, d, ε }
C	✓		
= { }	{ c, ε }		
D		✓	
= { }	{ ε }	{ d, ε }	
E	✓		
= { }	{ e }		
F		✓	
= { }	{ ε }	{ g, ε }	

S → aAB | CD E → efg
A → CD | SE | ε F → Fg | ε
B → aSB | DS
C → cC | ε
D → DdC | ε

b. FOLLOW sets

FIRST(S) = {a, c, d, ε} FIRST(A) = {a, c, d, e, ε}
FIRST(B) = {a, c, d, ε} First(C) = {c, ε}
FIRST(D) = {d, ε} FIRST(E) = {e}
FIRST(F) = {g, ε}

follow

S = {}	{$, a, c, d, e, }	{$, a, c, d, e}
A = {}	{a, c, d, $}	{a, c, d, $}
B = {}	{$}	{$, a, c, d, e}
C = {}	{d, $}	{$, a, c, d, e, }
D = {}	{$, a, c, d}	{$, a, c, d, e}
E = {}	{a, c, d, $}	{a, c, d, $}
F = {}	{g}	{g}

Question 2:

S → ABCD

A → CD | aA

B → b

C → cC | ε

D → dD| ε

First

FIRST(S) = {a, b, c, d}

FIRST(A) = {a, c, d, ε}

FIRST(B) = {b}

FIRST(C) = {c, ε}

FIRST(D) = {d, ε}

Follow

FOLLOW() = {$}

FOLLOW() = {b}

FOLLOW() = {c, d, $}

FOLLOW() = {b, d, $}

FOLLOW() = {b, $}

2.1

The two rules to determine if a CFG can form a predictive recursive descent parser is:

- If A → α and A→ β then FIRST(α) intersect FIRST(β) = Φ
- If ε ← FIRST(A), then FIRST(A) intersect FOLLOW(A) = Φ

The tests that this CFG needs to pass are:

FIRST(A) intersect FIRST(B) = Φ

FIRST(CD) intersect FIRST(aA) = Φ

FIRST(C) intersect FIRST(D) = Φ

FIRST(A) intersect FOLLOW(A) = Φ

FIRST(C) intersect FOLLOW(C) = Φ

FIRST(D) intersect FOLLOW(D) = Φ

Because this CFG passes these tests that could potentially introduce ambiguity, we can build a predictive recursive descent parser.

2.2

Writing a predictive recursive descent parser:

```
void parse_S(){
        t_type = getToken();
        // check FIRST(S)
        // check FIRST(A)
        if(t_type == a | | t_type == c | | t_type == d){
                ungetToken();
                parse_A();
                printf("S → A");
        }
        // check FIRST(B)
        else if (t_type == b){
                parse_B();
                printf("S→B");
        }
        else{
                syntax_error();
        }
}

void parse_A(){
        t_type =getToken();
        //check FIRST(A)
        if(t_type == c) {
                parse_C();
                printf("A→C");
        }
```

```
        else if(t_type == d){
                parse_D();
                printf("A→D");
        }
        else if(t_type == a){
                parse_A();
                printf("A→A);
        }
        else{
                syntax_error();
        }
}

void parse_B(){
        t_type = getToken();
        // check FOLLOW(B)
        if(t_type == c){
                parse_C();
                printf("B→C");
        }
        else if(t_type == d){
                parse_D();
                printf("B→D");
        }
        else if(t_type == EOF){
                printf("B→EOF");
        }
        else{
                syntax_error();
        }
}

void parse_C(){
        t_type = getToken();
        // check FIRST(C) and FOLLOW(C)
        if(t_type == c){
                parse_C();
                printf("C→ C");
        }
        else if(t_type == b){
                parse_B();
                printf("C→B");
        }
        else if (t_type == d){
                parse_D();
                printf("C→D");
```

```
        }
        else if(t_type == EOF){
                printf("C→EOF");
        }
        else{
                syntax_error();
        }

}

void parse_D(){
        t_type = getToken();
        // check FIRST(D) and FOLLOW(D)
        if(t_type == d){
                parse_D();
                printf("D→D");
        }
        else if(t_type == b){
                parse_B();
                printf("D→B");
        }
        else if(t_type == EOF){
                printf("B→EOF");
        }
        else{
                syntax_error();
        }
}
```