

## CSE 401 - LL Semantics - Week 5

### Canonical LL(1) Problems and their Solutions:

#### FIRST Conflict:

Both productions of  $A$  have  $\alpha$  in their FIRST sets

0.  $A ::= \alpha\beta \mid \alpha\gamma$

#### Solution:

Factor out the prefix ( $\alpha$ )

0.  $A ::= \alpha \text{ Tail}$

1.  $\text{Tail} ::= \beta \mid \gamma$

#### Left Recursion:

Special FIRST conflict:  $\beta$  in FIRST for both productions

0.  $A ::= A\alpha \mid \beta$

#### Solution:

Create recursive tail from suffix of recursive production

1.  $\text{Tail} ::= \alpha \text{ Tail}$

Append Tail to non recursive productions

0.  $A ::= \beta \text{ Tail}$

1.  $\text{Tail} ::= \alpha \text{ Tail}$

Add empty string ( $\epsilon$ ) as a rhs for the tail production

0.  $A ::= \beta \text{ Tail}$

1.  $\text{Tail} ::= \alpha \text{ Tail} \mid \epsilon$

#### FIRST FOLLOW Conflict:

$B$  is nullable,  $\alpha$  in FIRST & FOLLOW

0.  $A ::= B\alpha$

1.  $B ::= \alpha \mid \epsilon$

#### Solution:

Substitute  $B$  into  $A$

0.  $A ::= \alpha\alpha \mid \alpha$

Factor out the prefix ( $\alpha$ )

0.  $A ::= \alpha \text{ Tail}$

1.  $\text{Tail} ::= \alpha \mid \epsilon$

#### Indirect Left Recursion:

Recursively alternates between  $A$  &  $B$

0.  $A ::= B\beta$

1.  $B ::= A \mid \alpha$

#### Solution:

Substitute  $B$  into  $A$

0.  $A ::= A\beta \mid \alpha\beta$

Solve like normal Left Recursion

0.  $A ::= \alpha\beta \text{ Tail}$

1.  $\text{Tail} ::= \beta \text{ Tail} \mid \epsilon$

Edit the following Grammars to make them LL(1). Then walk through the top down parse for the string given in the parenthesis (Grammars 1 and 2 are review from last week)

**Grammar 1 (“azx”)**

0.  $S ::= a B \mid a w$

1.  $B ::= C x \mid y$

2.  $C ::= \epsilon \mid z$

**Grammar 2 (“ax”)**

0.  $S ::= a B$

1.  $B ::= C x \mid y$

2.  $C ::= \epsilon \mid x$

**Grammar 3 (“azx”)**

0.  $S ::= S B \mid a \mid w$

1.  $B ::= C x \mid y$

2.  $C ::= \epsilon \mid z$

**Grammar 4 (“azx”)**

0.  $S ::= B w \mid a B$

1.  $B ::= S \mid z x$

**Example pseudocode of a recursive-descent parser for grammar 2:  
(Helpful example for Homework 3: Programming!!)**

```
// S ::= aB
void S() {
    match(a)
    B()
}

// B ::= Cx | y
void B() {
    if next is y {
        match(y)
    } else {
        C()
        match(x)
    }
}

// C ::=  $\epsilon$  | x
// *One* implementation of
// the C() method
void C() {
    if next is x {
        match(x)
    } else if (lookahead is
        not in follow set for C){
        error
    }
    // else, epsilon so do
    // nothing
}
```