**CPSC 323, Handout No.5, Removing left recursive CFG, FIRST and FOLLOW**

This handout is to provide you enough information to write a simple compiler for a given grammar. **a. Left recursion removal**

Consider CFG: S🡪 Sa | λ. Let’s trace this grammar few times

S

/ \

S a

/ \

S a

/ \

S a

/ λ

if S is a function to test something in your program, then according to the tracing process the function calls itself without testing any condition and the process loops forever. This problem is called left-recursion that must be resolved. In general, a CFG is in left-recursion format if the right-hand-side begins with same non-terminal as the left side.

**Removing left recursive grammars**: Any left recursive CFG can be eliminated by rewriting the CFG using the following rules

|  |  |
| --- | --- |
| CFG in left-recursion form | Removing the left-recursion production |
| A🡪 A α| β β is betha  α is alpha  Trace w=βααα  A  / \  A α  / \  A α  / \  A α  β ααα= βααα | Introduce a new non-terminal A’  A →β A’  A’ → αA’  A’→ λ    Trace w=βααα  A / \ β A’ / A’ α / \ α A’ / \A’  α λ=βααα |

**Example.** Remove left-recursions from the following CFG

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CFG in left-recursion form | Same CFG after removing left-recursion productions | | | | | | |
| E→E + T ----------- α = +T  E→E - T ----------- α =-T  E→ T ----------- β =T | Introduce a new non-terminal E’ | | | | | | |
| E→β E’, ------------  E’→α E’-------------  E’→αE’ -------------  E’→ λ | | | E→ T E’ | |  | |
| E’→+TE’ | |
| E’→-TE’ | |
| E’→λ |  |
| T→T \* F ………. α =\*F  T→T / F ………. α = /F  T→F ………. β =F | Introduce a new non-terminal T’ | | | | | | |
|  | T → FT’  T’→\*FT’  T’→/FT’ | | | | |  |
| T’→ λ |  | | | |
| E→E + T E→TE’,  E→E – T E’→+TE’ | -TE’  E→T E’ →λ  T→T \*F T→FT’  T→T /F T’→\*FT’ | /FT’  T→F T’→λ  F→(E )  F→a  This is the general CFG for expressions in C++. E is Expression, T is Term. And F is Factor | Let Q=E’ and R=T’ | | | | | | |
|  | E→TQ  Q→+TQ  Q→-TQ  Q→λ  T→FR  R→\*FR  R→/FR  R→λ  F→( E )  F→a | | | | |  |

Trace a\*(a + a) using the last two grammars to see the application of removing left recursions.

|  |  |
| --- | --- |
| 1. E→E + T 2. E→E – T 3. E→T 4. T→T \*F 5. T→T /F 6. T→F 7. F→(E ) 8. F→a | Start at E, we have 3 options 1,2, or 3. We use 3  E  |  T  For T, we have 3 options 4,5, and 6. We use 4  E |  T  |  T \* F  | / | \  F ( E )  / | \  E + T  T F  F  a \* ( a + a )  Problem: too many choices at some steps |

Remove left recursions and trace Start at E, there is only one option for E, E→TQ

1. E→TQ E
2. Q→+TQ T Q
3. Q→-TQ F R
4. Q→λ \* F R
5. T→FR ( E )
6. R→\*FR T Q
7. R→/FR F R + T Q
8. R→λ F R
9. F→( E ) a \* ( a λ + a λ λ ) λ λ=a\*(a+a)
10. F→a

For T, there is only one option: T→FR.

For Q, there are three options, to trace + use Q→+TQ , for – use Q→-TQ, and if there is no need use Q→λ .

For R, there are three options, to trace \* use

R→\*FR, for / use R→/FR, and if there is no need use

R→ λ

1. **Removing λ from FA (we have done it before) Example.**

Suppose in our design we came up with the following FA in which there is(are) some λ ‘s .

b λ a

b a

-

1

2

3

+4

Construct a table and use the table to design a new FA without λ input

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **T** | **able** |  | **FA without λ** |  |
|  |  | a | b |  | a b  -    +      b | + |
| {1} | { } | {2,4} |
| {2} | {1,4} | { } |
| {3} | {1,4} | { } |
| {4} | { } | { } |
| {1,4} | { } | {2,4} |
| {2,4} | {1,4} | { } |

1. **Removing λ from CFG (we have done it before)**

**Examples**. Remove λ from the following CFG

S→aS | bS | λ

S→ aS | aλ |bS | bλ or S→aS | bS | a | b

1. **Left factoring**

Given A→aB | aD, both right-hand-sides start with a. If we factor by a we get aB | aD = aB + aD = a( B+D) = aT, where T→B | D. Hence, A→aA | aB is the same as A→aT, T→B | D

1. **Converting EBNF to BNF**

Recall: [a] means either a of λ. {a} means a0, a, a2, …..

Convert: <Expr>→<Term> { ( +|- )<Term> }

<Term> →<Factor>{ ( \* | / ) <Factor>}

<Factor> → id | constant | ( <Expr> )

Let <Expr> be E, <Term> be T, and <Factor> be F. Then the grammar becomes

E→T{ +T | -T } or E→TQ, Q→+TQ | -TQ | λ

T→F{\*F | /F } or T→FR, R→\*FR |/FR | λ ; F→id | constant | ( E )

1. **The FIRST set members**

Lets go back to CFG which is not left-recursive.

E→TQ, Q→ +TQ, Q→-TQ, Q→ λ

For Q we have three options, +TQ, -TQ , or just λ. By knowing the first terminal on the righthand-side, we can make a decision which grammar to use. For example, if we needed +, then we use Q→+TQ, in case of – we use Q→-TQ, and if we don’t need to continue, choose Q→λ.

Therefore for : Q→+TQ, Q→-TQ, Q→ λ FIRST(Q) = {+ - λ } and

For : R→\*FR, R→/FR, R→ λ FIRST( R) ={ \* / λ }

For: E→TQ, members of FIRST(T) are in FIRST( E )

Now we look at some examples to understand this concept better.

**Example.** Find the members of **FIRST** set for the following CFG

|  |
| --- |
| i. S→Ab |Bc non-terminals={ S, A, B, C, D}  A→Df | CA terminals = { b, c, f, g, e, c, h, i}  B→gA | e  C→dC | c  D→h | i  To trace w=hfb, we must start at S. There are two choices, either use S→Ab or S→Bc. If we know the members of FIRST(A) or FIRST(B) we will make a good decision to either use S→Ab or S→Bc.  The long method to find the FIRST members of all non- terminals is to expand the grammar hfb  Dfb FIRST(D)={h , i} Ab ifb dcAb  S CAb FIRST(C )={d , c} cAb gAc  Bc FIRST(B) ={g , e} ec  FIRST(A) = FIRST(D) U FIRST(C )= {h,i ,d, c}, FIRST(S)=FIRST(A) U FIRST(B)={h,i,d,c,g,e} Now, to see the power of knowing the members of FIRST of each non-terminal, we trace w=hfb again:  To decide whether to use S→Ab or S→Bc, since h is a member of FIRST(A) we use S→Ab  S  / \  A b  Now for expanding A, we have two choices: A→Df or A→ CA. To make that decision we check to find whether f is a member of FIRST(D) or a member of FIRST(B). Since f is a member of  FIRST(D), therefore we use A→Df  S  / \  A b  / \  D f  /  h f b= w |
| ii. S→ABCd non-terminals={S,A,B,C}  A→e|f|λ terminals ={e,f,g,h,p,q,λ}  B→g|h|λ  C→p|q Expand this grammar  eBCd  S  ABCd fBCd gCd λBCd =BCd  hCd  λCd=Cd pd qd |
| eBCd  S  ABCd fBCd gCd λBCd =BCd  hCd  λCd=Cd pd qd    FIRST(C )={p,q}  FIRST( B)= {g,h, λ}  FIRST(A) = {e,f , λ}  FIRST( S) = {FIRST(A)- λ} U {FIRST(B )- λ } U FIRST( C )  ={ e, f, g, h, p, q}    The key to this problem is in S→ABCd, FIRST(S) = FIRST (A)= {e,f, λ}. If we use λ, the grammar becomes S→ λBCd or S→BCd. Now FIRST(S) = FIRST(B )={g,h, λ}, if we use λ, the grammar becomes S→BCd= λCd or S→Cd. In this new grammar for S, FIRST(S)=FIRST(C)={p,q}. Hence, altogether we obtain  FIRST( S) = {FIRST(A)- λ} U {FIRST(B )- λ } U FIRST( C )= { e,f, g,h, p,q} |
| **Rules to find members of FIRST**   1. X→ x | λ x and λ are terminals, hence both are in FIRST ( X )={ x, λ } 2. X→ yA y is terminal, therefore y is in FIRST( X ) 3. X→AB, A→ λ FIRST( A ) is subset of FIRST( X). Since A→ λ, then X→AB= λB=B or FIRST( B) is also a subset of FIRST (X). Hence,   FIRST( X) = {FIRST(A) – λ} U FIRST (B) |

Consider these rules, lets find the members of FIRST of both previous examples faster

|  |  |
| --- | --- |
| **CFG** | **Members of FIRST of each non-terminal** |
| S→Ab |Bc | 5.FIRST(S )=FIRST(A) U FIRST(B) ={h,i,d,c,g,e} rule (c) |
| A→Df | CA | 4.FIRST(A)=FIRST(D) U FIRST(C)= {h,i,d,c} rule (a,c) |
| B→gA | e | 3. FIRST(B)= {g,e} rule(a,b) |
| C→dC | c | 2. FIRST(C)={d,c} rule(a,b) |
| D→h | i | 1.Start from here: FIRST(D) = { h,i}, rule(a) |

|  |  |
| --- | --- |
| **CFG** | **Members of FIRST of non-terminals** |
| S→ABCd | 4.FIRST(S)={FIRST(A)- λ}U{FIRST(B)- λ}U{FIRST(C)={e,f,g, h,p,q} |
| A→e|f|λ | 3. FIRST( A) = {e,f, λ} |
| B→g|h|λ | 2. FIRST(B)={g,h, λ} |
| C→p|q | 1.Start here: FIRST (C )= {p,q} |

|  |  |
| --- | --- |
| **CFG** | **Members of FIRST set** |
| E→TQ | FIRST(E) = FIRST (T)= { (, a} |
| Q→+TQ  Q→-TQ  Q→λ | FIRST(Q )={+, -, λ } |
| T→FR | FIRST(T)= FIRST(F)= { (, a} |
| R→\*FR  R→/FR  R→λ | FIRST(R )={\*, /, λ} |
| F→( E )  F→a | FIRST(F) = { ( , a} |

|  |  |
| --- | --- |
| **CFG** | **Members of FIRST set** |
| E→E + T  E→E – T  E→T | The first two productions are not providing any help to find the FIRST (T). The last rule implies: FIRST( E) = FIRST(T) = { (, a } |
| T→T \*F  T→T /F  T→F | The first two rules are not providing any info for FIRST(T). The last rule implies:  FIRST(T)= FIRST(F) = { (, a} |
| F→(E )  F→a | FIRST (F ) = { (, a } |

**g. FOLLOW set members**

Given a non-terminal A, the FOLLOW(A) is the set of terminals comes after A in a given CFG. For example if X→AB, then FOLLOW(A) are the FIRST (B)

**Rules to find the members of FOLLOW set**

1. If S is the starting symbol, then $ is in FOLLOW(S)
2. If A→Xα, then α in FOLLOW(X)
3. If A→XY, then (i) members of FIRST(Y) are in FOLLOW(X) (ii) members of FOLLOW(A) are in FOLLOW (Y ). (iii) If Y→λ, then A→Xλ=X which implies members of FOLLOW(A) are in FOLLOW(X)

**Example.** It is required to find the members of FIRST of all non-terminals before finding the members of FOLLOW. This is a detail example of how to find the FOLLOW elements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CFG** | **Members of FIRST** |  | |  | |
| 1. E→TQ 2. Q→+TQ 3. Q→-TQ 4. Q→λ 5. T→FR 6. R→\*FR 7. R→/FR 8. R→λ 9. F→( E ) 10. F→a | FIRST(E) = FIRST(T)= { ( , i}    FIRST(Q) = { +, - λ }    FIRST(T)=FIRST(F) ={ (, a}      FIRST( R)= {\*, /, λ}    FIRST(F) = { (, a } |  |  | FIRST |  |
| E | ( i |
| Q | + - λ |
| T | ( a |
| R | \* / λ |
| F | ( a |
|  |  |

Now, let’s find the members of FOLLOW of each non-terminals E, Q, T, R, and F

# FOLLOW (E )

-E is starting symbol, $ is in FOLLOW( E )

- list all productions have E on the right-hand-side

F→( E ) implies ) is in follow (E )

Therefore : FOLLOW( E )= { $, ) }

# FOLLOW(Q)

List all production with Q on their right-hand-side

-E→TQ, members of FOLLOW( E) are in FOLLOW(Q), FOLLOW(Q)={ $, ) }

-Q→+TQ is not adding anything to FOLLOW(Q)

-Q→-TQ is not adding anything to FOLLOW(Q)

Therefore FOLLOW( Q )= { $, ) }

# FOLLOW(T)

List productions with T on their right-hand-side

-E→TQ, {FIRST(Q) – λ} are in FOLLOW(T), FOLLOW(T) = { +, - }

-Q→ λ , implies E→Tλ =T which conclude FOLOOW(E ) are in FOLLOW( T)

-Q→+TQ and Q→-TQ are not adding any element to FOLLOW( T), because we already added the {FIRST(Q) – λ} to the FOLLOW of T

Therefore FOLLOW(T) ={FIRST(Q)-λ} U FOLLOW(E )= {+, - , $, ) }

# FOLLOW (R )

List all production with R on their right side

-T→FR, FOLLOW(T) are in FOLLOW(R)

-R→\*FR and R→/FR are not adding any element to FOLLOW(R )

Therefore: FOLLOW( R ) = FOLLOW(T)= {+, -, S, ) }

## FOLLOW( F )

T→FR, { FIRST( R)- λ} are in FOLLOW ( F ), FOLLOW(F) = { \*, / }

R→λ, implies T→Fλ=F which indicates FOLLOW(T) are in FOLLOW(F)

### Therefore: FOLLOW(F) = {\*, /} U Follow(T)= {\*, /, +, -, S, ) }

Summarize all findings

|  |  |  |  |
| --- | --- | --- | --- |
| **Non-terminals** | **FOLLOW members** |  | **FIRST members** |
| E | ) $ | ( i |  |
| Q | ) $ | + - λ |  |
| T | + - ) $ | ( a |  |
| R | + - ) $ | \* / λ |  |
| F | \* / + - ) $ | ( a |  |

**Assignment No.5** ( FIRST and FOLLOW sets ) Names: Richard Gresham

**Names:** Sean McCarthy

**Use the provided tables to answer the question**.

Given the following CFG

(i) **A🡪 A + B (ii) A 🡪 A-B (iii)A🡪 A \* B (iv) A🡪 B (v) B 🡪 ( A) | a;**

(5 points) Remove left-recursions. Find members of **FIRST** and **FOLLOW**

|  |  |  |  |
| --- | --- | --- | --- |
| **Remove left recursions** | **States** | **FIRST** | **FOLLOW** |
| A🡪 BA’  A’🡪+BA’  A’🡪-BA’  A’🡪\*BA’  A’🡪λ  B🡪(A) | a | A | ( a | $ ) |
| A’ | + - \* λ | $ ) |
| B | ( a | $ ) + - \* |

2. (5 points) Given CFG below. Complete the table where Terminals={i d ( , ) if else $ }, Nonterminals={S L C E }

|  |  |  |
| --- | --- | --- |
| **States** | **FIRST** | **FOLLOW** |
| S | if i | $ else ; |
| L | i d | ) |
| C | λ , | ) |
| E | i d | , ) if i else ; |

**}S 🡪 if ( E ) S else S;**

S 🡪 **i ( L )**

**L 🡪 E C**

**C 🡪** **λ**

**C 🡪** **, E C**

**E 🡪 i**

**E 🡪 d**

**3. (5 points)** Complete the table for the given CFG where Terminals={e f h g f i }, Non-terminals={S A B C

D }

**S 🡪AB**

**S 🡪 C f**

|  |  |  |
| --- | --- | --- |
| **States** | **FIRST** | **FOLLOW** |
| S | e λ h g f | $ |
| A | e λ | h |
| B | h | $ |
| C | g f | f |
| D | g | g f |

**A 🡪 e f**

**A 🡪** λ

**B 🡪 hg**

**C 🡪 DD**

**C 🡪 fi**

D 🡪 g

4. (5 points) Complete the table for this CFG. Terminals={ b d a} , Non-terminals={ S T U V }

|  |  |  |
| --- | --- | --- |
| **States** | **FIRST** | **FOLLOW** |
| S | a λ b | $ |
| T | a λ | b $ |
| U | b | $ b |
| V | b d | a $ b |

1. 🡪 **TU**
2. 🡪 **aVa**
3. 🡪 **λ**
4. 🡪 **bVT**
5. → **Ub**

**V** → **d**

CPSC 323

Quiz No.5 (20 points) name: Richard Gresham

Consider the following CFG. (i) remove all left-recursions (ii) find the members of FIRST and FOLLOW of the grammar after removing left recursions

|  |  |
| --- | --- |
| **The given CFG** | **Remove all left recursions** |
| **A**→ **A + B**   1. → **A \* B**   **A**→ **B**   1. → **B\*D**   **B**→ **D**  **D**→ **( A)**  **D**→ **a | d** | A 🡪 BA’  A’ 🡪 +BA’  A’ 🡪 \*BA’  A’ 🡪 λ  B 🡪 DB’  B’ 🡪 \*DB’  B’ 🡪 λ  D 🡪 (A)  D 🡪 a | d |
| **Members of FIRST after removing left rec.** | **Members of FOLLOW after removing left rec.** |
| |  |  | | --- | --- | | Non-terminal | FIRST members | | A | a d ( | | A’ | + \* λ | | B | a d ( | | B’ | \* λ | | D | a d ( | | |  |  | | --- | --- | | Non-terminals | FOLLOW members | | A | $ ) | | A’ | $ ) | | B | + \* $ ) | | B’ | + \* $ ) | | D | + \* $ ) | |