Date: Name: Abdul Aziz

Compiler Construction

Assignment No 03

BCSF19A026

1) Removing Null prodution:

ifStmt → if(BoolExpr)Stmt ElsePart | if(BoolExpr)Stmt

ElsePart → else Stmt

CompoundStmt → {Stmt list} | {}

Stmtlist → stmtlist stmt | Stmt

2) Removing Cycles:

since there is no cycle so, we skip this part.

3) Removing unit prodution:

Arglist → type Identifier | Arglist , Arg

Stmt → while(BoolExpr) stmt | Expr ; | {Stmtlist} | {} | Type Identifier |
if (BoolExpr) stmt Else part | if (BoolExpr)stmt | ;

Mag → Mag+term | Mag-term | term + Factor | Term/factor |
(Expr) | Identifier | Number

Date:

Term → Term * Factor | Term/Factor | (Expr) | Identifier | number

Stmtlist → stmtlist stmt | while(BoolExpr)stmt | Expr ; | {stmtlist} | {} | type Id ; | if (BoolExpr) stmt Else part | if (BoolExpr) stmt

4) Removing Direct leftRecursion:

Arglist → type Identifier Arglist'

Arglist' → , Arg Arglist' | ε

stmt list → stmt stmtlist'

stmtlist' → stmt stmtlist' | ε

Mag → Term + Factor Mag' | Term/Factor Mag' | (Expr) Mag' | Identifier Mag' | number Mag'

Mag' → +Term Mag' | -Term Mag' | ε

Term → (Expr) Term' | Identifier Term' | number Term'

Term' → +Factor Term' | /Factor Term' | ε

5) Removing Indirect left Recursion.

since There is no Indirect left Recursion so. we skip this part.

Date:

6) Removing Left Fectoring :

stmt → while(BoolExpr) stmt | Expr ; | { stmt 1' |
type Identifier | if stmt2' | ;

Stmt 1' → stmt list } | }

Stmt2' → (BoolExpr) Stmt Stmt3'

Stmt 3' → Elsepart | ε

if Stmt → if (Bool Expr) stmt if Stmt'

if stmt' → Else part | ε

Compare → == | < Compare1 | > Compare2

Compare1 → = | > | ε

Compare2 → = | ε

Mag → Term Mag2 | (Expr) Mag' | identier |
number mag'

mag' → + Term Mag' | - Term Mag' | ε

Mag2 → * Factor Mag' | / Factor Mag'

Date:

CFG: Now the final CFG is ready

Function → Type Identifier (Arglist) compound Stmt
Arg list → Type Identifier Arglist'
Arglist' → , Arg ArgList' | ε
Arg → Type Identifier
stmt → While (Bool Expr) Stmt | Expr ; | { stmt1' } |
Type Identifier; | ; | if (Bool Expr) Stmt Stmt3'
stmt1' → stmt list list } | }
Stmt 3'' → Elsepart | ε
Declaration → Type Identifier;
Type → int | float
While Stmt → While (Bool Expr) Stmt
if Stmt → if (Bool Expr) Stmt ifStmt'
ifstmt' → Elsepart | ε
Else part → else Stmt
Compound Stmt → { Stmt list } | { }
Stmt list → Stmt Stmtlist'
stmt list' → Stmt stmtlist' | ε
Bool Expr → Identifier Compare Mag
Compare → == | < compare1 | > compare2

Date:

compar1 → =|>|ε

compar2 → =|ε

Expr → identifier = Mag

Mag → Termp Mag2 |(expr) Mag'| identifier|number Mag'

Mag' → +Term Mag'| -Term Mag' |ε

Mag2 → *Factor Mag'| /Factor Mag'

Term → (Expr) Term'| identifier Term'|number Term'

Term' → *Factor Term'| /Factor Term'|ε

Factor → (Expr)| Identifier| number

Date:

Calculation First set & Follow set

Non Terminal	First set	Follow set
Function	{int, float}	{ $ }
Arglist	{int, float}	{) }
Arg list'	{ , , ε }	{) }
Arg	{ int, float }	{ , ,) }
st mt	{while, id, {, int, float, if, ; }	{else, while, id, if, {, int, float, ; }
stmt1'	{while, id, {, int, float, if, ;, ε}	{while, id, if, {, int, float, ;, else}
stmt3'	{ else, ε }	{while, id, if, {, int, float, else}
Declaration	{int, float}	{ }
Type	{ int, float }	{ id }
while Stmt	{while}	{ }
if StmT	{if}	{ }
if stmt'	{else, ε}	{ }
Elsepart	{else}	{else, while, id, if, int, float, ;, ;, {}
CompoundStmt	{ { }	{ $ }
stmt list	{while, id, if, {, int, float, ;}	{ } }
stmt list'	{while, id, if, {, int, float, ;, ε}	{ } }
Bool Expr	{id}	{) }
compare	{=, <, >}	{ (, id, number }
compare1	{ =, >, ε }	{ (, id, number }
compare2	{ >, ε }	{ (, id, number }
Expr	{id}	{ ;,) }
Mag	{ (, id, number }	{), ; }
Mag'	{+, −, ε}	{), ; }

Date:

Non Terminal	Find Set	follow set
mag2	{ *, / }	{ ;.) }
Term	{ (, id, number}	{*, /, +, -,), ; }
Term'	{ ε, *, / }	{*, /, +, -, ;,)}
Factor	{id, number, (}	{ *, /, +, -, ;,) }