**In21-S4-CS3513 – Programming Languages**

**HomeWork 01**

**NAME : K.S. RANASINGHE INDEX NO. : 210518H**

Question 01

* Left recursion always produces a non-LL(1) grammar;

**E -> E or T**

**-> E nor T**

**-> E xor T**

* Common prefixes always produce a non-LL(1) grammar;

**T -> F and T**

**-> F nand T**

**-> F**

**Therefore, the given grammar is non-LL(1)**

Question 02

* Fixing Left Recursion

**E -> TY { not , ( , i , true , false }**

**Y -> or TY { or }**

**-> nor TY { nor }**

**-> xor TY { xor }**

**-> { ) }**

* Fixing Common Prefixes

**T -> FX { not , ( , i , true , false }**

**X -> and T { and}**

**-> nand T { nand }**

**-> { or , nor , xor , ) }**

* Modified Grammer

|  |  |  |  |
| --- | --- | --- | --- |
|  | First Set | Follow Set | Select Set |
| **E -> TY** | { not , ( , i , true , false } | { ) } | **{ not , ( , i , true , false }** |
| **Y -> or TY** | { or , nor , xor } | { ) } | **{ or }** |
| **-> nor TY** | **{ nor }** |
| **-> xor TY** | **{ xor }** |
| **->** | **{ ) }** |
| **T -> FX** | { not , ( , i , true , false } | { or , nor , xor , ) } | **{ not , ( , i , true , false }** |
| **X -> and T** | {and, nand, ε} | { or , nor , xor , ) } | **{ and}** |
| **-> nand T** | **{ nand}** |
| **->** | **{ or , nor , xor , ) }** |
| **F -> not F** | { not , ( , i , true , false } | { and, nand, or , nor , xor , ) } | **{ not }** |
| **-> P** | **{ ( , i , true , false }** |
| **P -> ( E )** | { ( , i , true , false } | { and, nand, or , nor , xor , ) } | **{ ( }** |
| **-> i** | **{ i }** |
| **-> true** | **{ true }** |
| **-> false** | **{ false }** |

**E -> TY { not , ( , i , true , false }**

**Y -> or TY { or }**

**-> nor TY { nor }**

**-> xor TY { xor }**

**-> { ) }**

**T -> FX { not , ( , i , true , false }**

**X -> and T { and}**

**-> nand T { nand }**

**-> { or , nor , xor , ) }**

**F -> not F { not }**

**-> P { ( , i , true , false }**

**P -> ( E ) { ( }**

**-> i { i }**

**-> true { true }**

**-> false { false }**

Question 03

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | or | nor | xor | and | nand | not | true | false | i | ( | ) |
| E |  |  |  |  |  | **E -> TY** | **E -> TY** | **E -> TY** | **E -> TY** | **E -> TY** |  |
| Y | **Y -> or TY** | **Y -> nor TY** | **Y -> xor TY** |  |  |  |  |  |  |  | **Y ->** |
| T |  |  |  |  |  | **T -> FX** | **T -> FX** | **T -> FX** | **T -> FX** | **T -> FX** |  |
| X | **X ->** | **X ->** | **X ->** | **X -> and T** | **X -> nand T** |  |  |  |  |  | **X ->** |
| F |  |  |  |  |  | **F -> not F** | **F -> P** | **F -> P** | **F -> P** | **F -> P** |  |
| P |  |  |  |  |  |  | **P -> true** | **P -> false** | **P -> i** | **P -> ( E )** |  |

Question 04

|  |  |  |
| --- | --- | --- |
| Stack | Input | Table Lookup |
| E | true nand (false xor i) or not i and not false nor i ⊥ | **E -> TY** |
| TY | true nand (false xor i) or not i and not false nor i ⊥ | **T -> FX** |
| FXY | true nand (false xor i) or not i and not false nor i ⊥ | **F -> P** |
| PXY | true nand (false xor i) or not i and not false nor i ⊥ | **P -> true** |
| true XY | true nand (false xor i) or not i and not false nor i ⊥ |  |
| XY | nand (false xor i) or not i and not false nor i ⊥ | **X -> nand T** |
| Nand TY | nand (false xor i) or not i and not false nor i ⊥ |  |
| TY | (false xor i) or not i and not false nor i ⊥ | **T -> FX** |
| FXY | (false xor i) or not i and not false nor i ⊥ | **F -> P** |
| PXY | (false xor i) or not i and not false nor i ⊥ | **P -> ( E )** |
| ( E ) XY | (false xor i) or not i and not false nor i ⊥ |  |
| E ) XY | false xor i) or not i and not false nor i ⊥ | **E -> TY** |
| TY ) XY | false xor i) or not i and not false nor i ⊥ | **T -> FX** |
| FXY ) XY | false xor i) or not i and not false nor i ⊥ | **F -> P** |
| PXY ) XY | false xor i) or not i and not false nor i ⊥ | **P -> false** |
| false XY ) XY | false xor i) or not i and not false nor i ⊥ |  |
| XY ) XY | xor i) or not i and not false nor i ⊥ | **X ->** |
| Y ) XY | xor i) or not i and not false nor i ⊥ | **Y -> xor TY** |
| xor TY ) XY | xor i) or not i and not false nor i ⊥ |  |
| TY ) XY | i) or not i and not false nor i ⊥ | **T -> FX** |
| FXY ) XY | i) or not i and not false nor i ⊥ | **F -> P** |
| PXY ) XY | i) or not i and not false nor i ⊥ | **P -> i** |
| i XY ) XY | i) or not i and not false nor i ⊥ |  |
| XY ) XY | ) or not i and not false nor i ⊥ | **X ->** |
| Y ) XY | ) or not i and not false nor i ⊥ | **Y ->** |
| ) XY | ) or not i and not false nor i ⊥ |  |
| XY | or not i and not false nor i ⊥ | **X ->** |
| Y | or not i and not false nor i ⊥ | **Y -> or TY** |
| or TY | or not i and not false nor i ⊥ |  |
| TY | not i and not false nor i ⊥ | **T -> FX** |
| FXY | not i and not false nor i ⊥ | **F -> not F** |
| not FXY | not i and not false nor i ⊥ |  |
| FXY | i and not false nor i ⊥ | **F -> P** |
| PXY | i and not false nor i ⊥ | **P -> i** |
| i XY | i and not false nor i ⊥ |  |
| XY | and not false nor i ⊥ | **X -> and T** |
| and TY | and not false nor i ⊥ |  |
| TY | not false nor i ⊥ | **T -> FX** |
| FXY | not false nor i ⊥ | **F -> not F** |
| not FXY | not false nor i ⊥ |  |
| FXY | false nor i ⊥ | **F -> P** |
| PXY | false nor i ⊥ | **P -> false** |
| false XY | false nor i ⊥ |  |
| XY | nor i ⊥ | **X ->** |
| Y | nor i ⊥ | **Y -> nor TY** |
| nor TY | nor i ⊥ |  |
| TY | i ⊥ | **T -> FX** |
| FXY | i ⊥ | **F -> P** |
| PXY | i ⊥ | **P -> i** |
| i XY | i ⊥ |  |
| XY | ⊥ | **X ->** |
| Y | ⊥ | **Y ->** |
| - | ⊥ |  |

A diagram of a tree

Description automatically generatedQuestion 05

ε

ε

ε

ε

ε

ε

ε

Question 06

proc E;

        T(); Y();

Write (E → TY);

end;

proc Y;

    case Next\_Token of

        T\_or : Read(T\_or);

                T();

                Y();

Write (Y -> or TY);

        T\_nor : Read(T\_nor);

                T();

                Y();

Write (Y -> nor TY);

        T\_xor : Read(T\_xor);

                T();

                Y();

Write (Y -> xor TY);

        T\_) : Write (Y → );

        otherwise   Error;

    end;

end;

proc T;

        F(); X();

Write (T → FX);

end;

proc X;

    if Next Token = T\_and

    then

        Read(T\_and);

T();

Write (X -> and T);

    else if Next Token = T\_nand

    then

        Read(T\_nand);

T();

Write (X -> nand T);

else Write (X -> );

end;

proc F;

    case  Next Token of

        T\_true, T\_false, T\_i, T\_( : P();

Write (F -> P);

        T\_not: Read(T\_not);

F();

Write (F -> not F);

        otherwise Error;

    end;

end;

proc P;

    case Next\_Token of

        T\_true : Read(T\_true);

Write (P -> true);

        T\_false : Read(T\_false);

Write (P -> false);

        T\_i : Read(T\_i);

Write (P -> i);

        T\_( : Read(T\_( );

E();

Read(T\_) );

Write (P -> (E));

        otherwise   Error;

    end;

end;

Question 07

P -> i

F -> P

X ->

T -> FX

P -> false

F -> P

X ->

T -> FX

Y ->

Y -> nor TY

E -> TY

P -> (E)

F -> P

P -> true

F -> P

X ->

T -> FX

X -> and T

T -> FX

P -> i

F -> P

X ->

T -> FX

P -> i

F -> P

F -> not F

P -> true

F -> P

X ->

T -> FX

X -> and T

T -> FX

P -> i

F -> P

X ->

T -> FX

Y ->

Y -> xor TY

Y -> nor TY

Y -> or TY

E -> TY

A black dotted line on a white background

Description automatically generatedQuestion 08

ε

ε

ε

ε

ε

ε

ε

ε

Question 09

proc E;

        T();

        Write (E → T);

        while Next\_Token ∊ {T\_or,T\_nor,T\_xor} do

            if Next\_Token = T\_or

                then    Read (T\_or); T();

                        Write (E → E or T);

            else if Next\_Token = T\_nor

                then    Read (T\_nor); T();

                        Write (E → E nor T);

            else if Next\_Token = T\_xor

                then    Read (T\_xor); T();

                        Write (E → E xor T);

end;

proc T;

        F();

        if Next\_Token = T\_and

            then    Read (T\_and); T();

                    Write (T → F and T);

        else if Next\_Token = T\_nand

            then    Read (T\_nand); T();

                    Write (T → F nand T);

        else    Write (T → F);

end;

proc F;

    case  Next Token of

        T\_true, T\_false, T\_i, T\_( : P();

Write (F -> P);

        T\_not: Read(T\_not); F();

Write (F -> not F);

        otherwise Error;

    end;

end;

proc P;

    case Next\_Token of

        T\_true : Read(T\_true);

Write (P -> true);

        T\_false : Read(T\_false);

Write (P -> false);

        T\_i : Read(T\_i);

Write (P -> i);

        T\_( : Read(T\_( ); E(); Read(T\_) );

Write (P -> (E));

        otherwise   Error;

    end;

end;

Question 10

P -> i

F -> P

T -> F

E -> T

P -> false

F -> P

T -> F

E -> E nor T

P -> (E)

F -> P

P -> true

F -> P

T -> F

T -> F and T

E -> T

P -> i

F -> P

T -> F

E -> E or T

P -> i

F -> P

F -> not F

P -> true

F -> P

T -> F

T -> F and T

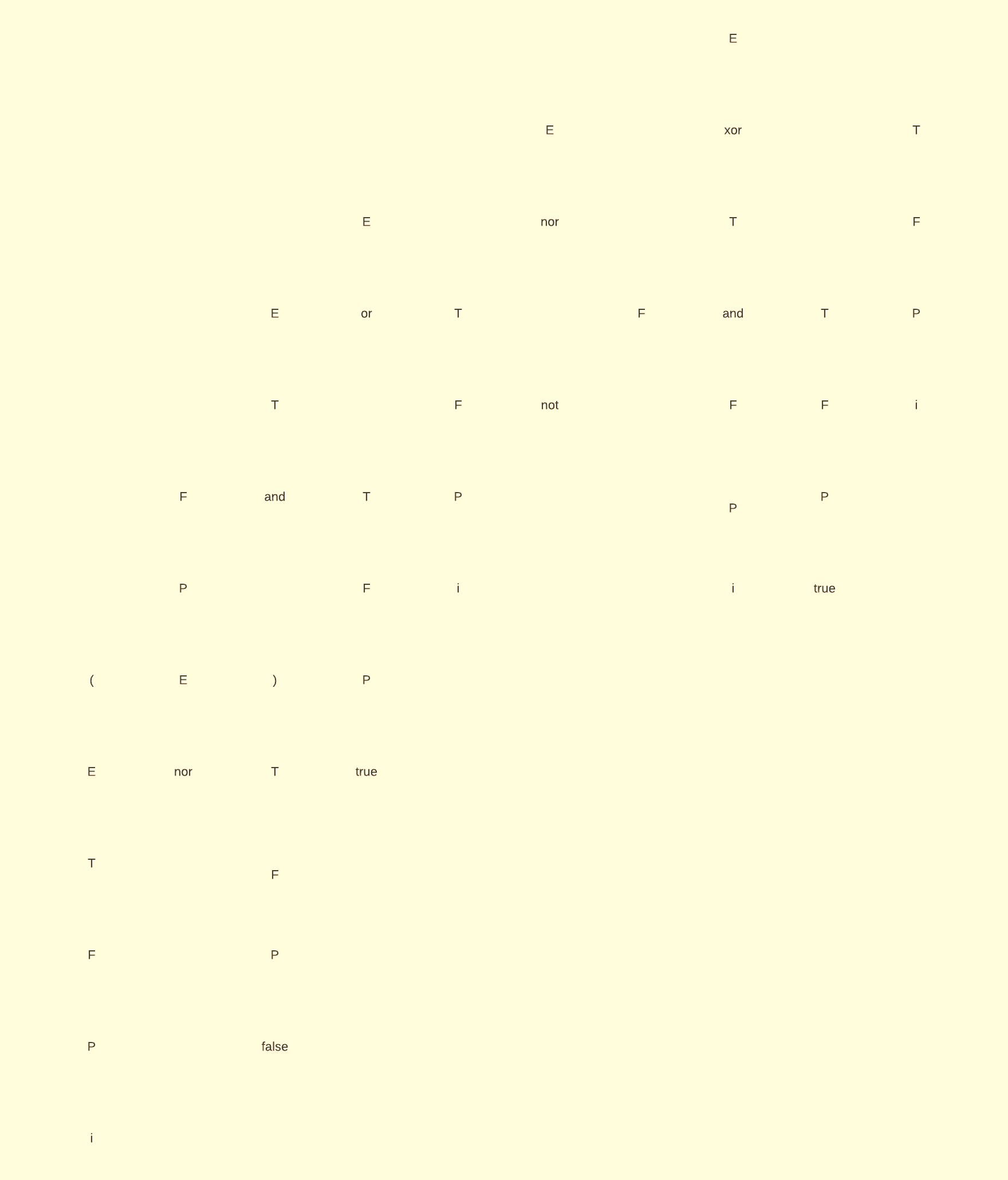
E -> E nor T

P -> i

F -> P

T -> F

E -> E xor T



Question 11

proc E;

        T();

        while Next\_Token ∊ {T\_or,T\_nor,T\_xor} do

            if Next\_Token = T\_or

                then    Read (T\_or); T();

                        Build\_tree(‘or’, 2);

            else if Next\_Token = T\_nor

                then    Read (T\_nor); T();

                        Build\_tree(‘nor’, 2);

            else if Next\_Token = T\_xor

                then    Read (T\_xor); T();

                        Build\_tree(‘xor’, 2);

end;

proc T;

        F();

        if Next\_Token = T\_and

            then    Read (T\_and); T();

                    Build\_tree(‘and’, 2);

        else if Next\_Token = T\_nand

            then    Read (T\_nand); T();

                    Build\_tree(‘nand’, 2);

end;

proc F;

    case  Next Token of

        T\_true, T\_false, T\_i, T\_( : P();

        T\_not: Read(T\_not); F();

Build\_tree(‘not’, 1);

        otherwise Error;

    end;

end;

proc P;

    case Next\_Token of

        T\_true : Read(T\_true);

Build\_tree(‘true’, 0);

        T\_false : Read(T\_false);

Build\_tree(‘false’, 0);

        T\_i : Read(T\_i);

Build\_tree(‘i’, 0);

        T\_( : Read(T\_( ); E(); Read(T\_) );

        otherwise   Error;

    end;

end;

Question 12

Build\_tree(‘i’, 0);

Build\_tree(‘false’, 0);

Build\_tree(‘nor’, 2);

Build\_tree(‘true’, 0);

Build\_tree(‘and’, 2);

Build\_tree(‘i’, 0);

Build\_tree(‘or’, 2);

Build\_tree(‘i’, 0);

Build\_tree(‘not’, 1);

Build\_tree(‘true’, 0);

Build\_tree(‘and’, 2);

Build\_tree(‘nor’, 2);

Build\_tree(‘i’, 0);

Build\_tree(‘xor’, 2);

