IISPL Parser Generator and Applications

How to use IISPLParserGenerator System
A simple desk-top calculator
A simple desk-top calculator using a class object
A desktop calculator with ambiguous grammars
A infix-to-postfix conversion
A English-to-Korean translator

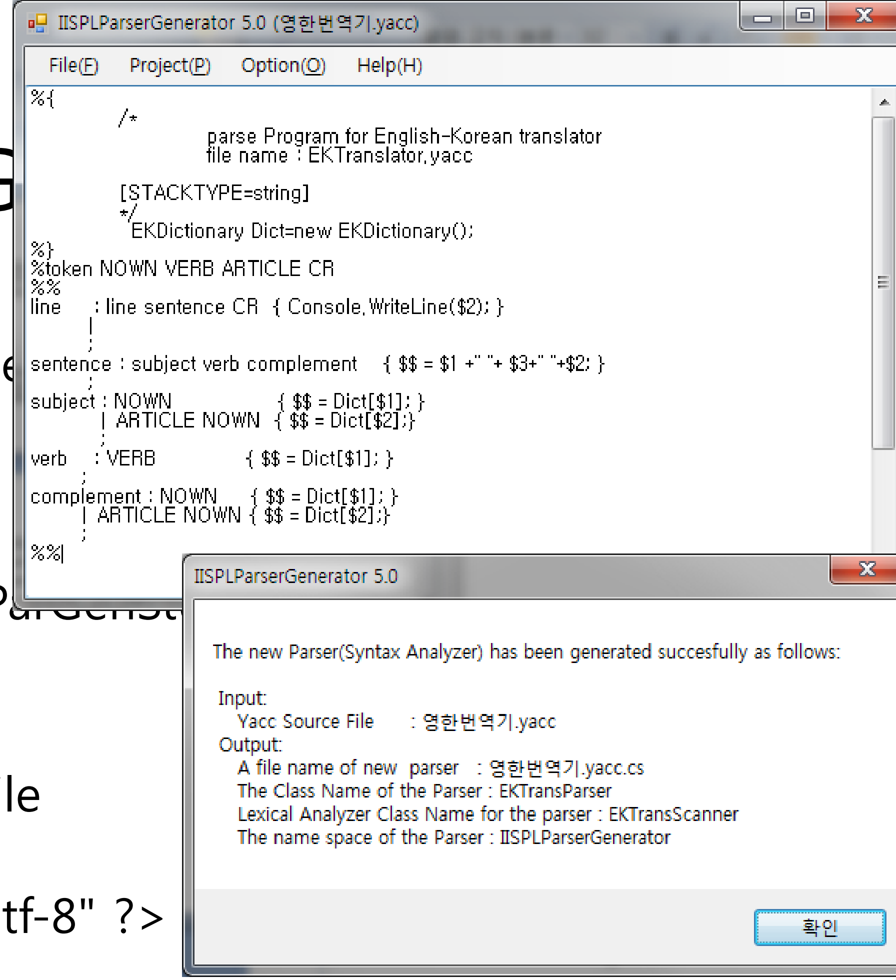
Dept. of Computer Engineering, Hanbat National University
Kim, Yoon Joong

4.9 Parser Generator

- Yacc program structure
- declarations
%%
translation rules
%%
supporting c routines
- Structure of the IISPLParGenStudio5.0
  - xxx.lex ⇒ LEX Generator ⇒ xxx.lex.cs
  - xxx.yacc ⇒ YACC Generator ⇒ xxx.yacc.cs
  - Source Text ⇒ xxx.yacc.cs ⇒ Target Text
(xxx.lex.cs)

4.9 Parser G

- How to use IISPLParserGenerator Syste
- Software
  - Library : IISPLParGenDll5_0.dll
  - Parser Scanner Generator : IISPLParGenSt
- IISPLParGenStudio5_0
  - File : New, Open, Save, Save As
  - Project : Lex Compile, Yacc Compile
- Config.xml
  - <?xml version="1.0" encoding="utf-8" ?>
    <config>
    <Language>C#</Language>
    <NameSpace>IISPLParserGenerator</NameSpace>
    <ScannerClassName>CalculatorScanner</ScannerClassName>
    <ParserClassName>CalculatorParser</ParserClassName>
    </config>



4 projects

1. A Simple Desktop Calculator
2. A Simple Desktop Calculator using a class object
3. A Simple Desktop Calculator with ambiguous grammars
4. An Infix-to-postfix converter
5. An English-Korean Translator

1. A Simple Desktop Calculator

1. Describe the languages : source and target languages
2. Define tokens
3. Describe a grammar for the source language
4. Design a SDD for translator of simple ㅇdesktop calculator
5. Simulate the desktop calculator: aprsing and evaluation
6. Generate a Lex source code “Calculator.lex” for the tokens and test
7. Generate a Yacc source code “Calculator.lex.yacc for the calculator
8. Test the calculator and discuss and show the steps of translation.

1. A Simple Desktop Calculator

1. Describe the languages
   - Source Language : simple expressions
     - 1+2 ₩n
       2*(22+1) ₩n
   - Target Language : result of computation
     - 3 ₩n
       46 ₩n

2. Define tokens for the source language
   -

Token	val	samples
+		
-		
(		
)		
NUM	int	1, 2, 33

3. Describe a grammar for the source language

```
line -> line exp ₩n
      | ε
exp -> exp + term
      | term
term -> term * factor
      | factor
factor ->( exp )
      | NUMBER
```

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

4. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

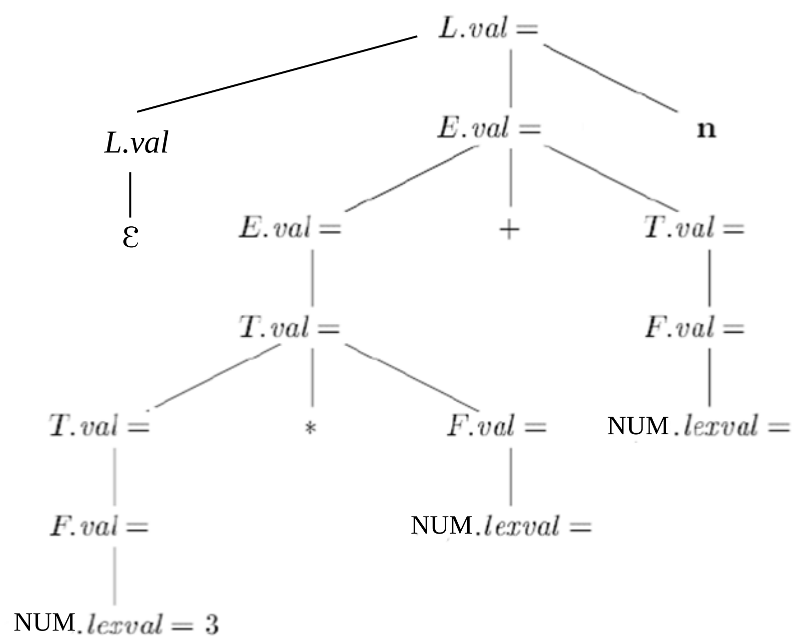


Figure 5.3: Annotated parse tree for $3 * 5 + 4$ **n**

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

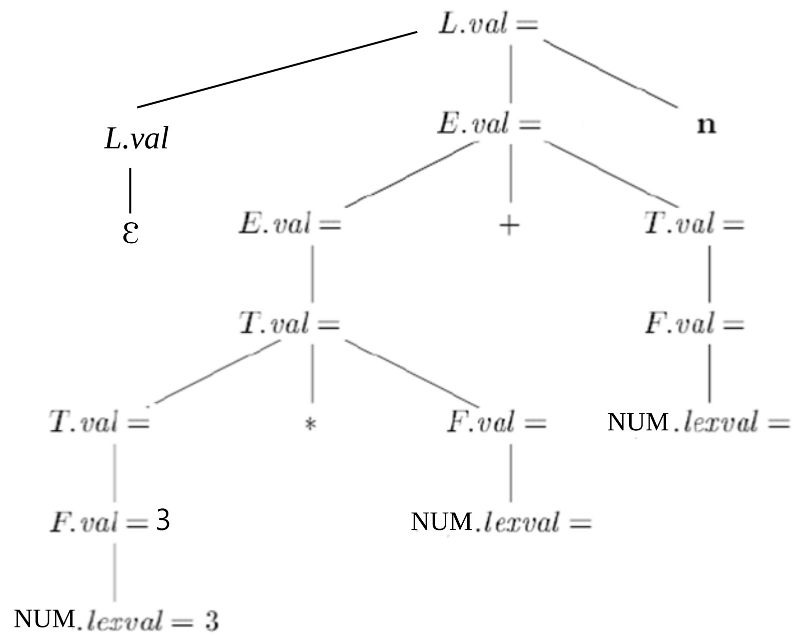


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

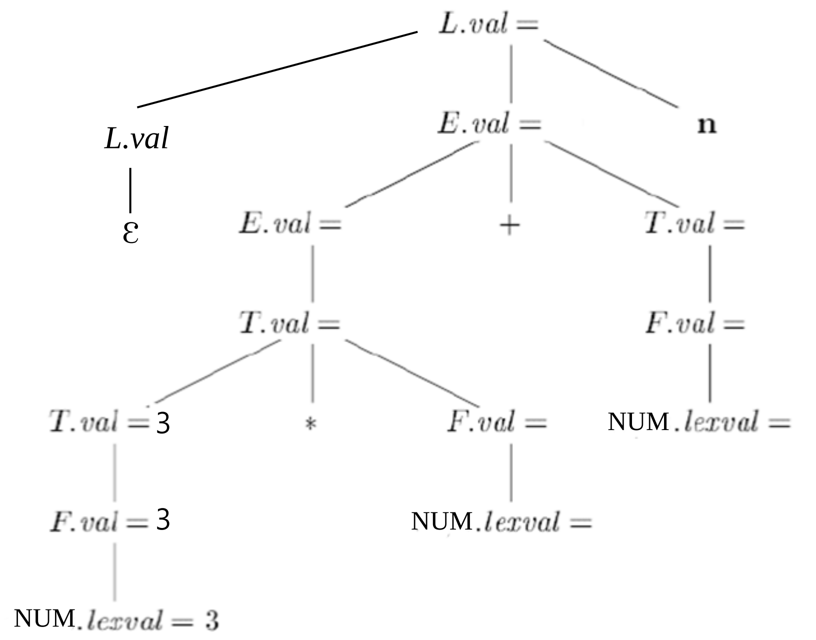


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val ;
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

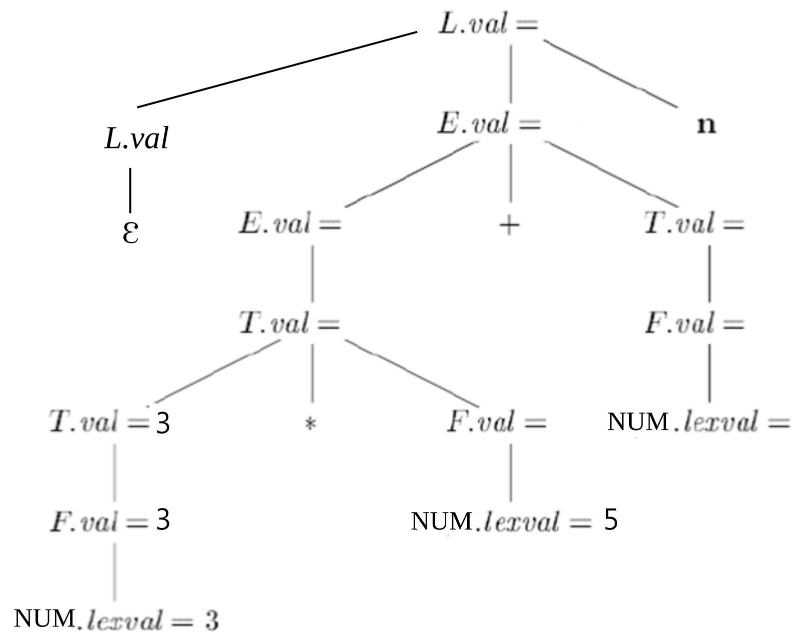


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val ;
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   - 4.1 Make a parse and produce an annotated parse tree
   - 4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

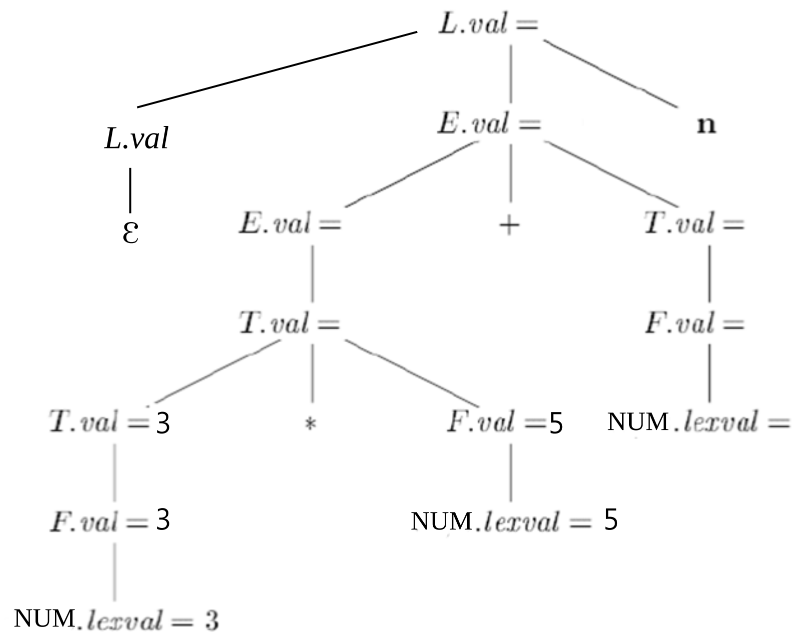


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   - 4.1 Make a parse and produce an annotated parse tree
   - 4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

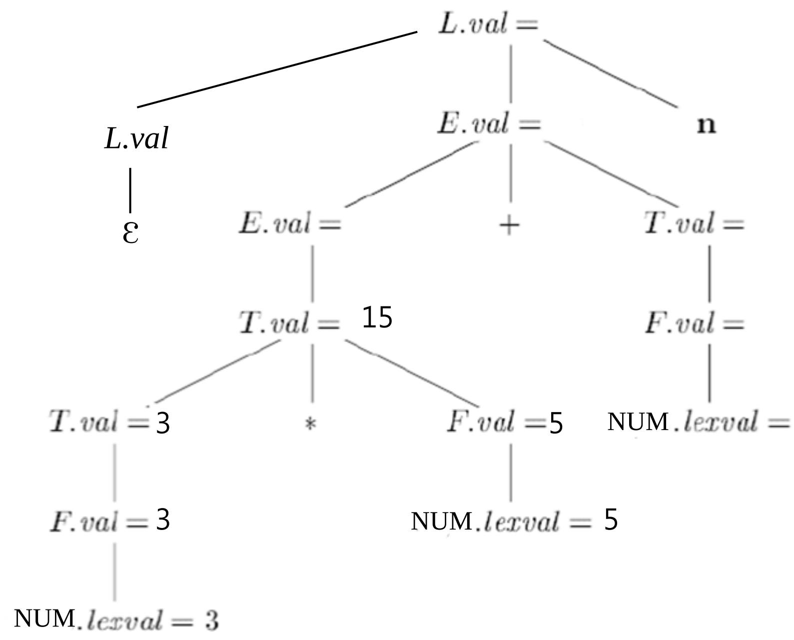


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

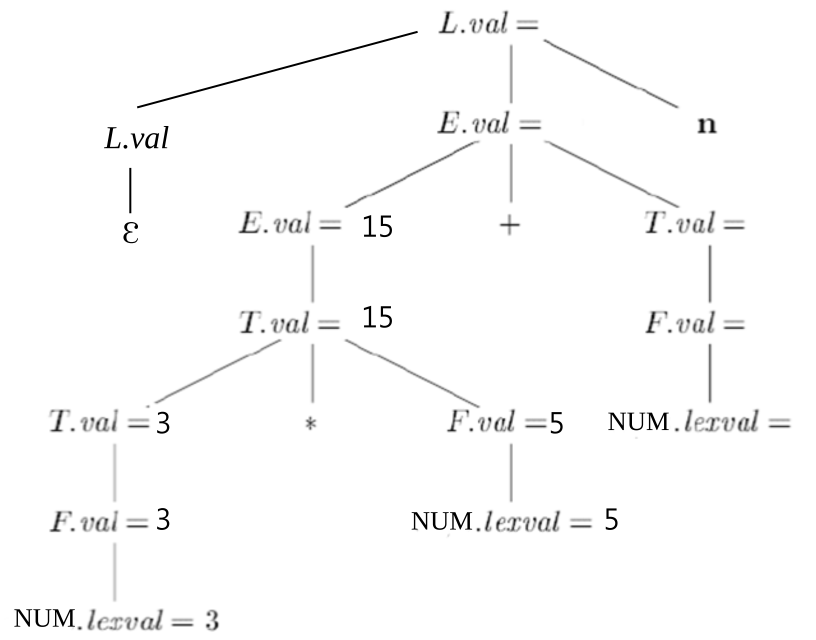


Figure 5.3: Annotated parse tree for $3 * 5 + 4$ **n**

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

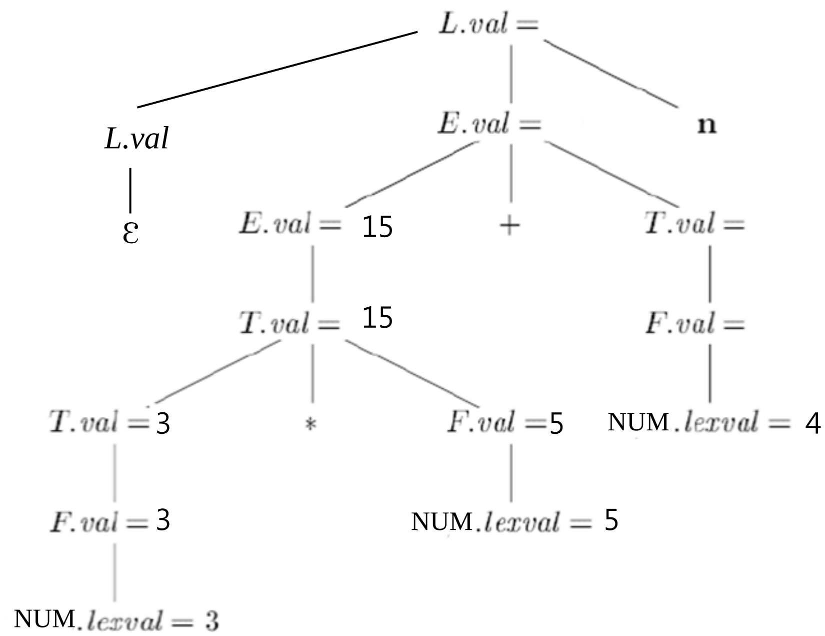


Figure 5.3: Annotated parse tree for $3 * 5 + 4$ **n**

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

4. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

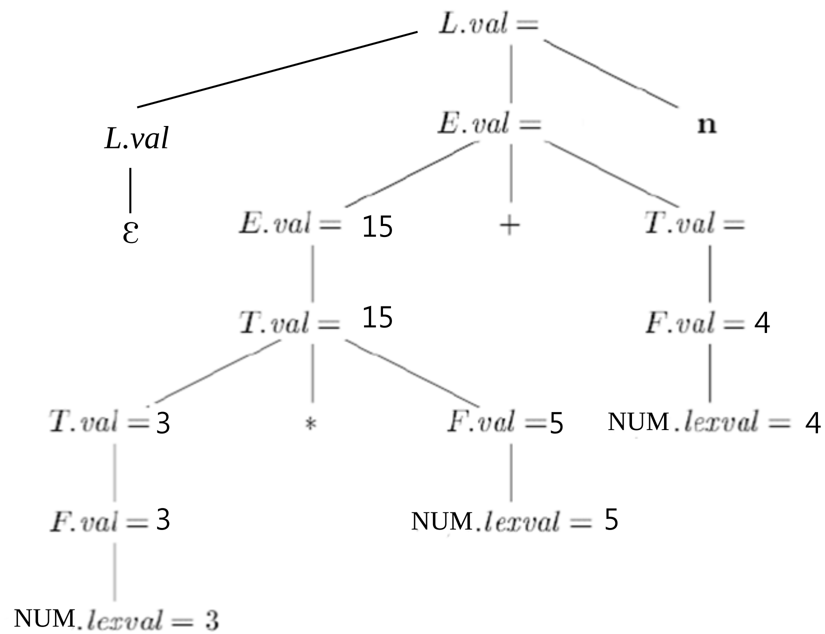


Figure 5.3: Annotated parse tree for $3 * 5 + 4$ **n**

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

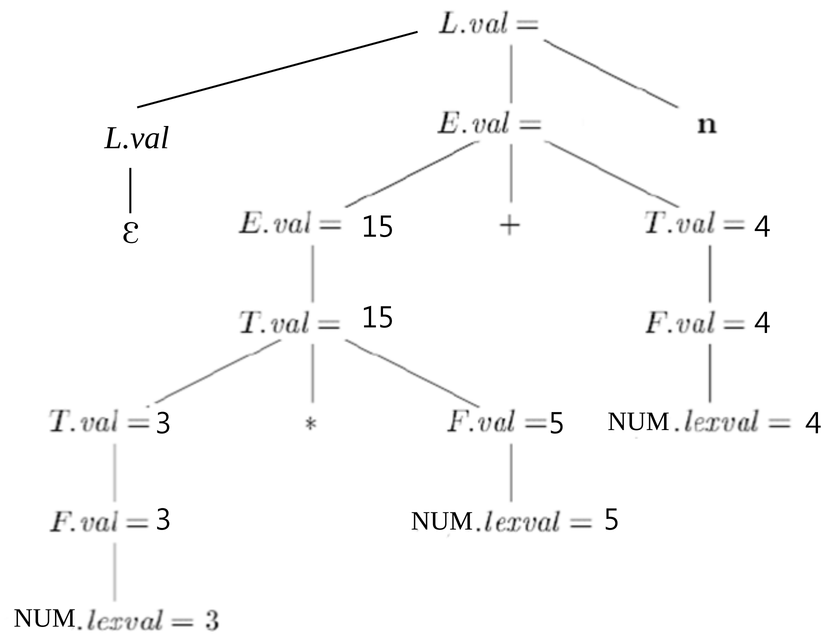


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   4.1 Make a parse and produce an annotated parse tree
   4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

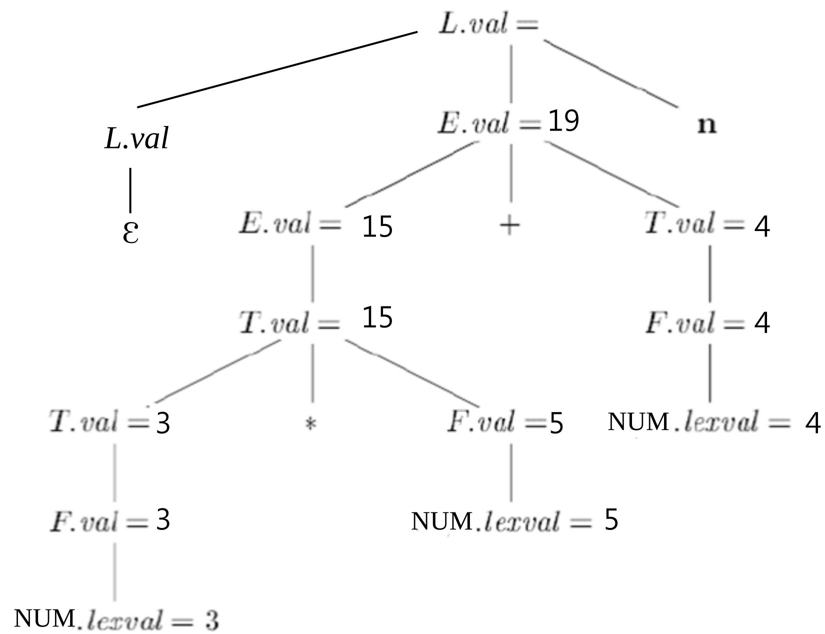


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

4. Design a SDD for translator of simple calculator

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

5. Simulate the desk calculator for a input expression 3*5+4 ₩n
   - 4.1 Make a parse and produce an annotated parse tree
   - 4.2 Evaluate the annotated parser tree for 3*5+4 ₩n

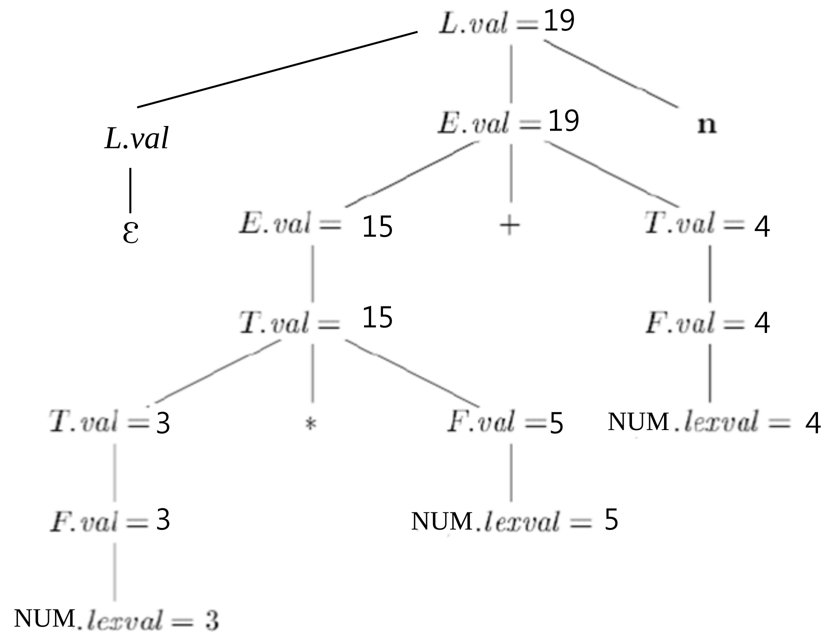


Figure 5.3: Annotated parse tree for $3 * 5 + 4\ \mathbf{n}$

1. A Simple Desktop Calculator

6. Generate and test a Lex source code for the tokens “Calculator.lex”

```
%{
  /*
            Lex source program for a simple desktop calculator .
            file name : Claculator.lex
            Tokens:  NUM  (  )  +  *  CR
  */
  %}
  delim      [ ₩t₩r]
  ws         {delim}+
  digit      [0-9]
  num        {digit}+(₩.{digit}+)?(E[+₩-]?{digit}+)?
  CR         [₩n]
  op         [()+*]
  %%
  {ws}       { return "WS"; }
  {num}      { yyoval=int.Parse(lexbuf);  return "NUM";}     //
  {CR}       { return "CR"; }
  {op}       { return lexbuf; }
  .          { return "WS";  }
  %%
  public void Test(string fn)
  {
    install(fn);
    do {
            string t=(string)lexan();
            Console.WriteLine("["+t+"]"+lexbuf);
    } while(!IsDone());
  }
```

Global Variables

string lexbuf :
has a string of lexeme for the extracted token.

object yyoval :
The value of yyoval will be used by parser

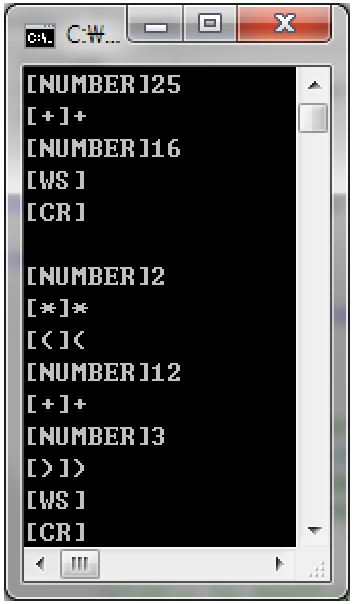
1. A Simple Desktop Calculator

- Config.xml 시스템구성변수 설정 in ParGenStudio

```
<?xml version="1.0" encoding="utf-8" ?>
   <config>
  <Language>C#</Language>
  <NameSpace>IISPLParserGenerator</NameSpace>
  <ScannerClassName>CalculatorScanner</ScannerClassName>
  <ParserClassName>CalculatorParser</ParserClassName>
  </config>
```

- IISPLParGenStudio로 Calculator.lex.cs 생성
  - Project / LEX Compile

- Test Calculator.Lex.cs
  - (new CalculatorScanner()).Test(“calculator.txt");
  - Calculator.txt
    25+16
    2*(12+3)
  - Result :



1. A Simple Desktop Ca

Production	Semantic rules
L -> L E ₩n	L.val = E.val
L -> ε	
E -> E_1 + T	E.val = E_1.val+ T.val
E -> T	E.val = T.val
T-> T * F	T.val = T_1.val * F.val
T -> F	T.val = F.val
F -> (E)	F.val = E.val
F -> NUM	F.val = NUM.lexval

7. Generate and test a Yacc source code for the tokens "Calcul

```
%{
/*
        Yacc source program for a simple desktop calculator Calculator .
        file name : Claculator.lex.yacc   Fig 4.56 A simple desk calculator,  p259
        [STACKTYPE=int]   //parser의 stack에 저장되는 데이터 형식을 int로 지정
*/
%}
%token + * ( ) NUM CR   //토큰의 정의
%%
line   : line expr CR  { Console.WriteLine($2); }
       |
       ;
expr   : expr + term   { $$ = $1 + $3; }
       | term
       ;
term   : term * factor { $$ = $1 * $3; }
       | factor
       ;
factor : ( expr )      {  $$ = $2; }
       | NUM           {  $$ = $1; }   //$1<=yyoval에 의하여
                                        저장된 정수 값을 사용한다.
       ;
%%
%{
public void Test()
{
        lexan.install("../../calculator/calculator.txt");
        Parse();"
}
%}
```

5. Generate a Lex source code for the tokens "Calculator.lex"

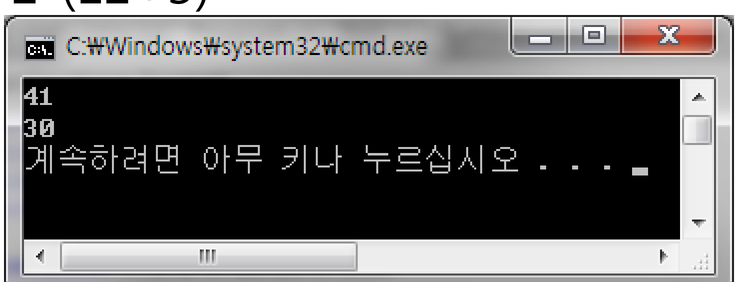
```
%{
 /*
        Lex source program for a
             simple desktop calculator .
        file name : Claculator.lex
        Tokens:  NUM  (  ) + * CR
 */
 %}
 delim     [ ₩t₩r]
 ws        {delim}+
 digit     [0-9]
 num       {digit}+(₩.{digit}+)?(E[+₩-]?{digit}+)?
 CR        [₩n]
 op        [()+*]
 %%
 {ws}      { return "WS"; }
 {num}     { yyoval=int.Parse(lexbuf);  return "NUM";}
 {CR}      { return "CR"; }
 {op}      { return lexbuf; }
 .         { return "WS";  }
 %%
 ...
```

1. A Simple Desktop Calculator

- IISPLParGenStudio로 Calculator.lex.cs 생성
  - Project / YACC Compile

8. Test the calculator and discuss and show the steps of translation.

-Test Calculator.yacc.cs

- (new CalculatorParser()).Test(“calculator.txt");
- Calculator.txt
  25+16
  2*(12+3)
- 

2. A Simple Desktop Calculator using a class object

- Concept
  - The instance of this class is created with attributes of the token whose token is extracted in lexical analysis.
  - And save in the global variable yyoval.

    ```
    {num}  {yyoval=new  myItem("NUMBER",lexbuf,int.Parse(lexbuf)); return "NUM";
    ```

  - In parsing, the value of the yyoval will be shifted into the stack of LR parser automatically .
  - Is able to be used in SDT;

    ```
     factor : ( expr )      {  $$ = $2; }
           | NUM           {  $$ = ((myItem)$1).val; }   //저장된 정수 값을 사용한다.
           ;
    ```

- A item class : myItem.cs
  - namespace IISPLParserGenerator

    ```
    {
        public  class myItem
        {
            public string token;
            public string lexeme;
            public object val;
            public Item(string token, string lexeme, object val)
            {
                    this.token=token;
                    this.lexeme=lexeme;
                    this.val=val;
             }
         }
    }
    ```

2. A Simple Desktop Calculator using a class object

```
%{
/*
        File name : Claculator1.lex
        Lex source program for a simple desktop calculator .
        Tokens:  NUM  (  ) + * CR
*/
%}
Delim [ ₩t₩r]
ws      {delim}+
digit   [0-9]
num     {digit}+(₩.{digit}+)?(E[+₩-]?{digit}+)?
CR      [₩n]
op      [()+*]
%%
{ws}    { return "WS"; }
{num}   {yyoval=new  myItem("NUMBER",lexbuf,int.Parse(lexbuf)); return "NUM";}
{CR}    { return "CR"; }
{op}    { return lexbuf; }
.       { return "WS";  }
%%
public void Test(string fn)
{
  install(fn);
  do {
        string t=(string)lexan();
        Console.WriteLine("["+t+"]"+lexbuf);
  } while(!IsDone());
}
```

```
%{
/*
      file name : Claculator1.lex.yacc
     Yacc source program for a simple desktop calculator.
      Fig 4.56 A simple desk calculator,  p259
      [STACKTYPE=int]
*/
%}
%token + * ( ) NUM CR   //토큰의 정의
%%
line    : line expr CR  { Console.WriteLine($2); }
        |
        ;
expr    : expr + term   { $$ = $1 + $3; }
        | term
        ;
term    : term * factor { $$ = $1 * $3; }
        | factor
        ;
factor : ( expr )    {  $$ = $2; }
        | NUM        {  $$ = ((myItem)$1).val; ; }
        ;
%%
%{
public void Test()
{
            lexan.install("../../calculator/calculator.txt");
            Parse();"
}
%}
```

3. A desktop calculator with ambiguous grammars

- Fig. 4.59 in page 292.
- How to use the Associativity and priority in Yacc source
- ```
  %{
  %}                          //associativity priority
  %token NUM
  %left + -                   //left          0
  %left * /                   //left          1
  %right UMINUS               //right         2
  %%
  E : E+E { $$ = $1 + $2;}
       ...
     | ( E ) { $$=$1; }
     | - E  %prec UMINUS { $$ = -$1; }
     ;
  %%
  ...
  ```
  ```

4. An Infix-to-postfix converter(IPConverter)

1. Describe expressions for the source and target language.
2. Define tokens and grammar
3. Define SDD
4. Construct project using IIPSLParGenDll5_0.dll and IISPLParGenStudio5_0.exe
5. Define converter system “config.xml”
6. Generate an test IPScanner.lex and IPScanner.lex.cs
7. Generate and test IPParser.yacc and IPParser.yacc.cs
8. Test and Show scanning and parsing for the infix expressions.

5. An English-to-Korean Translator(EKTranslator)

1. Describe sentences for the source(English) and target language(Korean).
2. Define tokens and grammar
3. Define SDD
4. Construct project using IIPSLParGenDll5_0.dll and IISPLParGenStudio5_0.exe
5. Define converter system “config.xml”
6. Construct electronic dictionary class Dictionary
7. Generate EKScanner.lex and EKScanner.lex.cs
8. Generate EKParser.yacc and EKParser.yacc.cs
9. Test and Show scanning for some English sentences.
10. Test and Show parsing for the EKTranslator.

5. An English-to-Korean Translator(EKTranslator)

1. Define set of sentence for English and Korean language
   - I am a boy 나는 소년입니다
   - You are a girl 너는 소녀입니다

2. Define tokens
Tokens	sample lexeme
NOWN	I, You, boy, girl
VERB	am, are
ARTICLE	a

Grammar

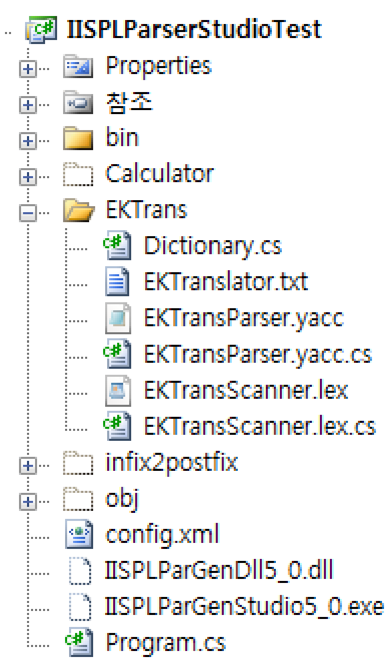
St -> S V C
S -> NOWN
V -> VERB
C -> ARTICLE NOWN

5. An English-to-Korean Translator(EKTranslator)

3. Define a SDD for the EKTranslator.
   - Productions / Semantic Rules

Productions	Semantic Rules
sentence -> subject verb comp	sentence.t = subject.t + " "+comp.t verb.t
subject -> NOWN	subject.t = Dict(NOWN.lexval)
verb -> VERB	verb.t = Dict(VERB.lexval)
comp -> ARTICLE NOWN	comp.t = Dict(NOWN.lexval)

4. Construct project using IIPSLParGenDll5_0.dll and IISPLParGenStudio5_0.exe



5. Define converter system "config.xml"

```
<?xml version="1.0" encoding="utf-8" ?>
- <config>
    <Language>C#</Language>
    <NameSpace>IISPLParserGenerator</NameSpace>
    <ScannerClassName>EKTransScanner</ScannerClassName>
    <ParserClassName>EKTransParser</ParserClassName>
  </config>
```

5. An English-to-Korean Translator(EKTranslator)

6. Construct electronic dictionary class Dictionary

```
class EKDictionary
 {
        Dictionary<string, string> Dict = new Dictionary<string, string>();
        public EKDictionary() {
            Dict["I"] = "나";
            Dict["am"] = "입니다";
            Dict["a"] = "";
            Dict["boy"] = "소년";
            Dict["girl"] = "소녀";
            Dict["You"] = "너";
            Dict["are"] = "입니다";
        }
        public string this[string word]  {
            get {  return  (Dict.ContainsKey(word) ? Dict[word] : "["+word+"]"); }
            set {  Dict[word] = value;  }
        }
}
```

5. An English-to-Korean Translator(EKTranslator)

7. Generate EKScanner.lex and EKScanner.lex.cs

```
%{
        /*
          EKTransScanner.lex
           LEX Program for a simple English-Korean translator
          token : NOWN,VERB,ARTICLE, CR
        */
%}
Delim   [ ₩t₩r]
Ws      {delim}+
Nown    I|You|boy|girl
verb    am|are
Article   a
CR      [₩n]
%%
{ws}      { return "WS"; }
{nown}  {yyoval=lexbuf;  return "NOWN"; }
{verb}    {yyoval=lexbuf;  return "VERB"; }
{article}  { return "ARTICLE";}
{CR}      { return "CR";}
.           { return "ANY";}
%%
public void Test(string fn) {
    install(fn);  string t;
    do  {
                t=(string)lexan();
                Console.WriteLine("["+t+"]"+lexbuf);
        } while(!IsDone());
}
```

8. Generate EKParser.yacc and EKParser.yacc.cs

```
%{
    /*
          EKTransParser.yacc
          Yacc Program for a simple English-Korean translator
   */
     EKDictionary Dict=new EKDictionary();
%}
%token NOWN VERB ARTICLE CR //
%%
line   : line sentence CR  { Console.WriteLine($2); }
        |
        ;
sentence : subject verb complement    { $$ = $1 +" "+ $3 + $2; }
         ;
subject :ARTICLE NOWN  { $$ = Dict[(string)$2];}
       | NOWN               { $$ =  Dict[(string)$1]; }
       ;
verb    : VERB               { $$ = Dict[(string)$1]; }
        ;
complement :ARTICLE NOWN  { $$ = Dict[(string)$2];}
       | NOWN               { $$=  Dict[(string)$1]; }
       ;
%%
%{
public void Test(string fn)
{
            lexan.install(fn);
            Parse();
}
%}
```

5. An English-to-Korean Translator(EKTra

9. Test and Show scanning and parsing for some English sentences.

- EKTranslator.txt
  - I am a boy
  - I am a girl
  - you are a boy
  - I am a girl
  -

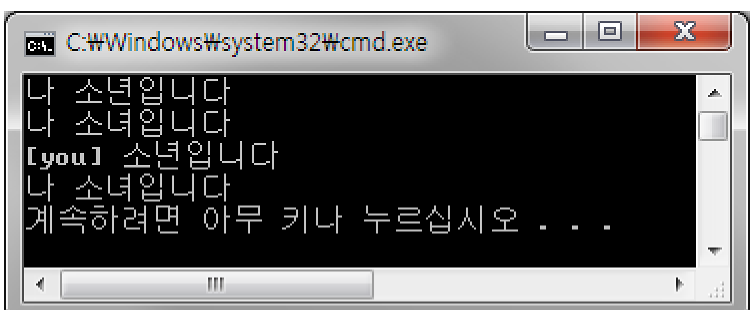
- (new EKTranslatorScanner()).Test("EKTranslator.txt");
  result:
  [NOWN] I
  [WS]
  [VERB] am
  [WS]
  [ARTICLE] a
  [WS]
  [NOWN] boy
  [WS]
  [NOWN] you
  …
  ….

5. An English-to-Korean Translator(EKTranslator)

10. Test and Show parsing for the EKTranslator.

- EKTranslator.txt
  - I am a boy
    I am a girl
    you are a boy
    I am a girl

- (new EKTranslatorParser()).Test("EKTranslator.txt");
  result:



5. An English-to-Korean Translator(EKTranslator)