ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

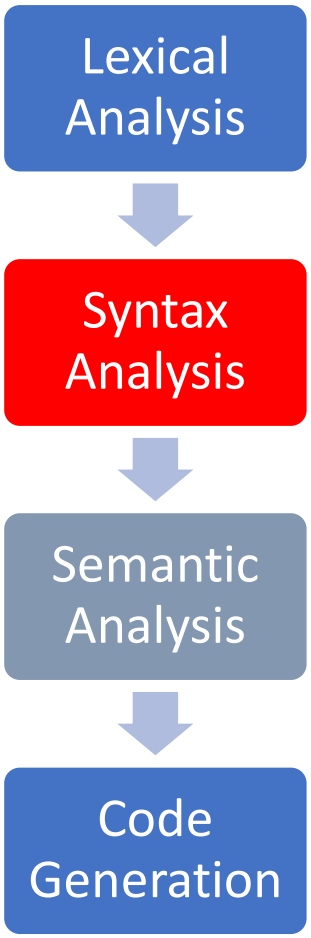
Thực hành
Xây dựng chương trình dịch

Bài 2. Phân tích cú pháp

Nội dung

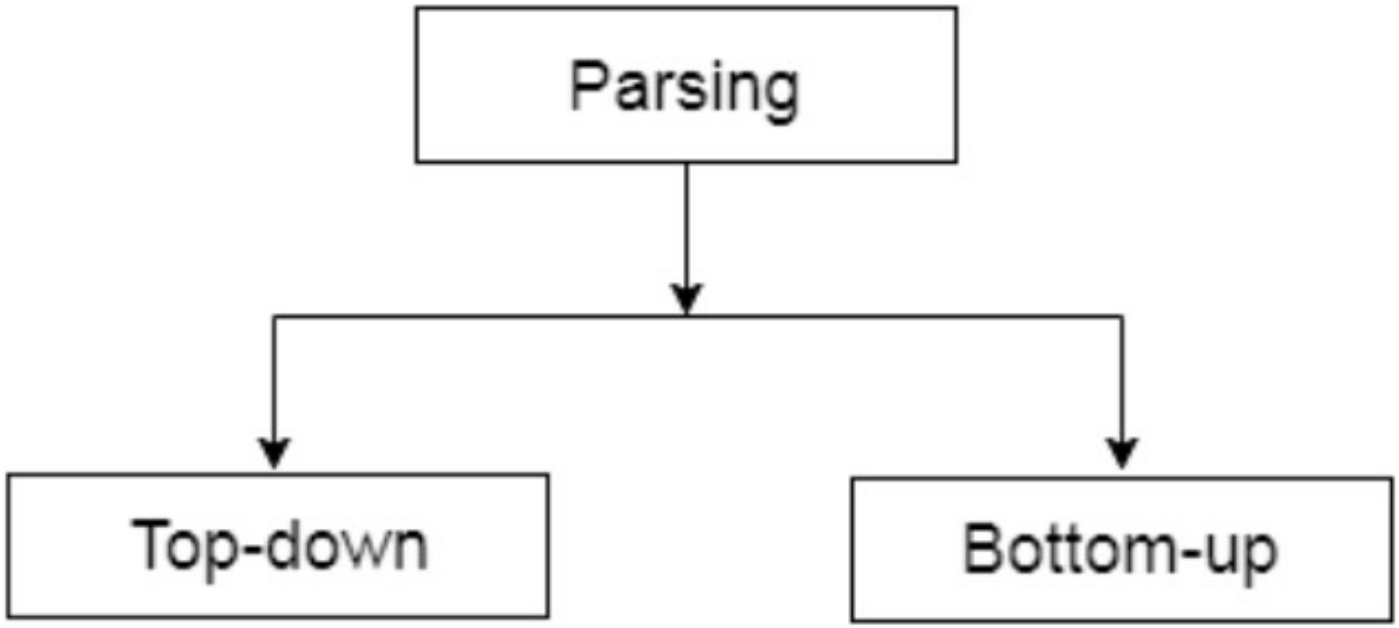
- Tổng quan
- Văn phạm KPL
- Cài đặt bộ phân tích cú pháp

Nhiệm vụ của bộ PT cú pháp



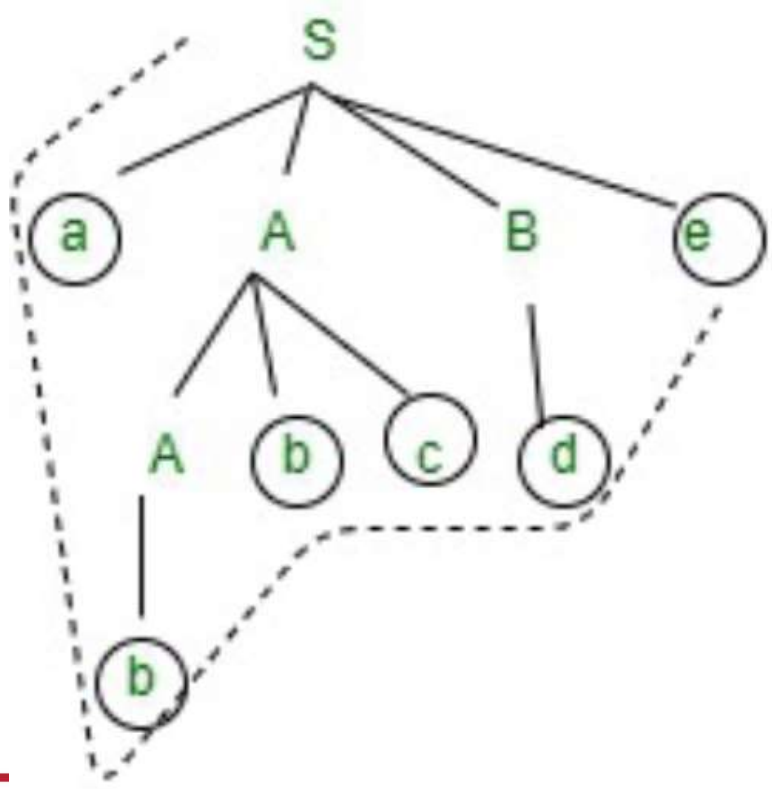
- Kiểm tra cú pháp của chương trình nguồn
  - Cấu trúc cú pháp được biểu diễn bằng văn phạm hoặc sơ đồ cú pháp
- Phục vụ cho các pha sau
  - Compiler làm việc theo tiếp cận tựa cú pháp nên bộ cú pháp rất quan trọng, quyết định tộc độ của compiler

Phân loại các bộ PTCP của NN lập trình



Phân tích trên xuống

- Xây dựng cây phân tích cú pháp từ gốc xuống lá, đọc chương trình nguồn từ trái qua phải
- Việc xây dựng cây dựa vào suy dẫn trái
- Nếu một vế trái có nhiều hơn 1 vế phải, chọn vế phải nào cho các nút cấp dưới?
- Ví dụ: Văn phạm G với các sản xuất
  - G: (1) $S \rightarrow a\ ABe$
    (2, 3) $A \rightarrow Abc|b$
    (4) $B \rightarrow d$
  - Xâu vào: abbcde

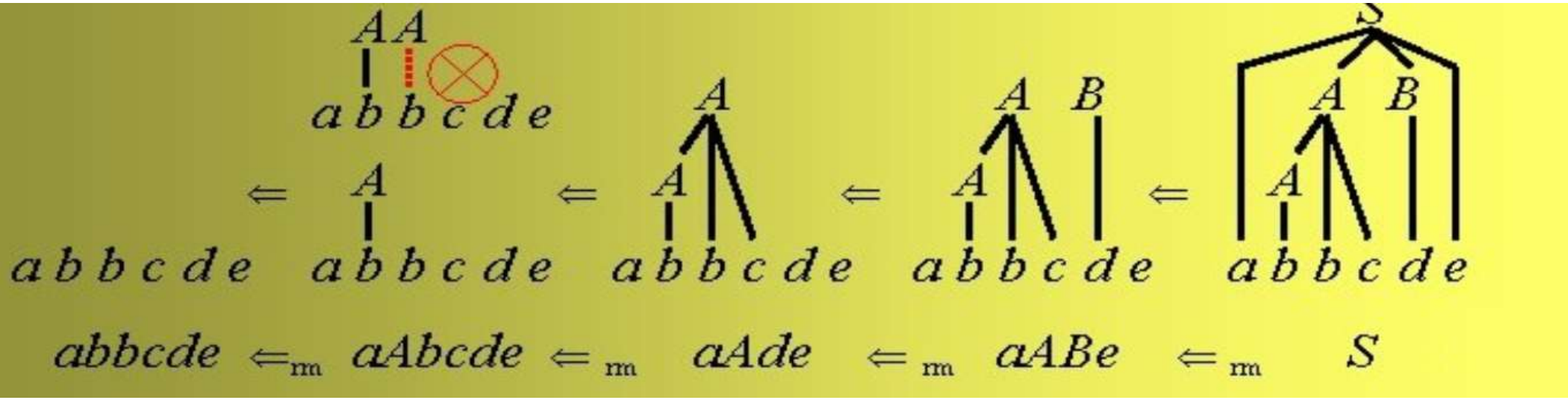


Phân tích cú pháp từ dưới lên

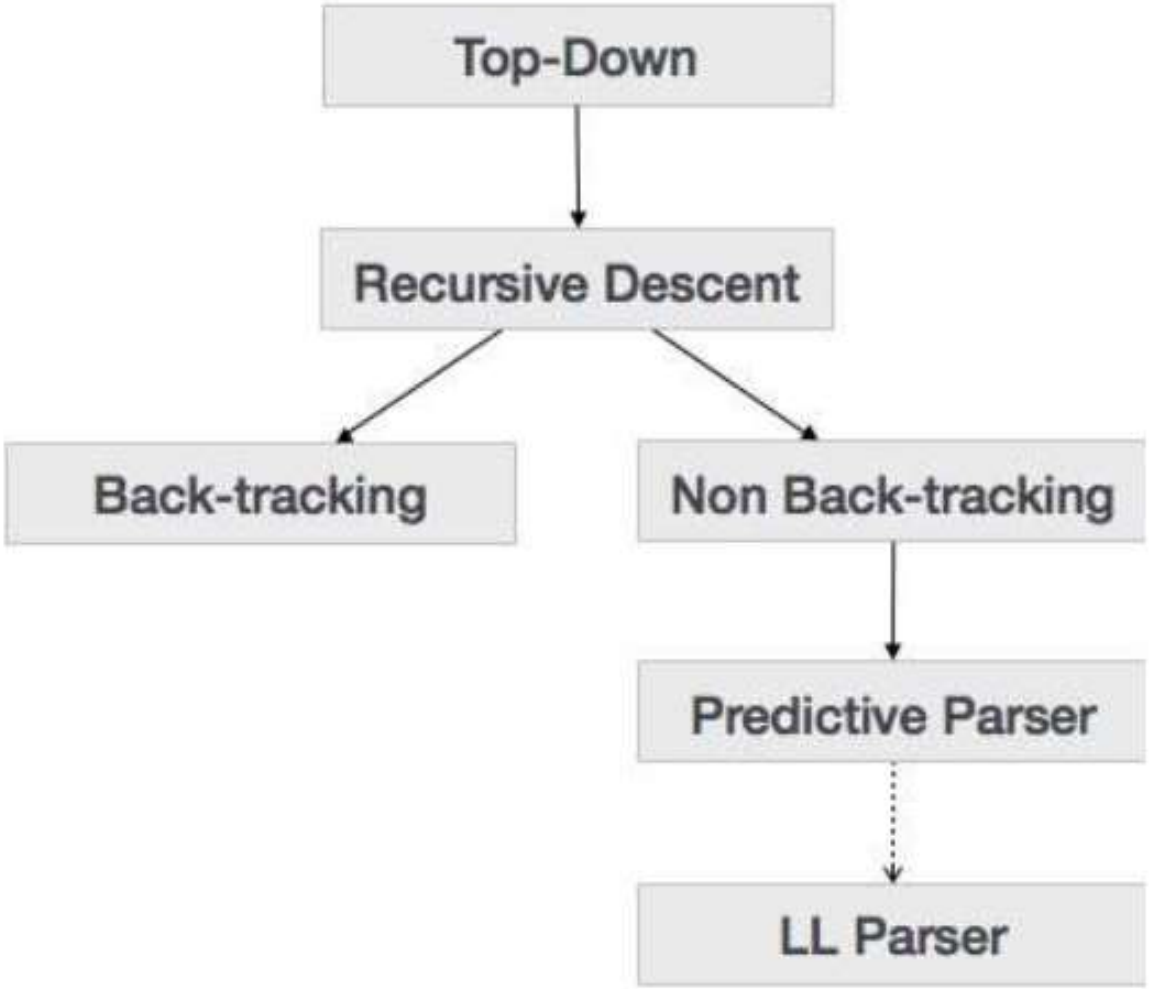
- Xây dựng cây phân tích cú pháp từ lá lên gốc, đọc chương trình nguồn từ trái qua phải
- Sử dụng suy dẫn phảiIt follows the rightmost derivation
- Ví dụ: Cho văn phạm G với các sản xuất:
  - G: (1) $S \rightarrow$ a ABe

    $A \rightarrow$ Abc|b

    $B \rightarrow$ d
  - Xâu vào: abbcde



Phương pháp phân tích trên xuống



Phương pháp đệ quy trên xuống

- Là phương pháp phân tích từ trên xuống, tức là vẽ cây từ gốc xuống lá
- Sử dụng các thủ tục có thể đệ quy, mỗi thủ tục cho một ký hiệu kếtthúc
  - Đầu tiên gọi thủ tục phân tích ký hiệu đầu
  - Khi một ký hiệu không kết thúc có nhiều vế phải, mỗi vế phải là một nhánh trong thủ tục
  - Sử dụng ký hiệu xem trước lookAhead để quyết định vế phải được chọn.

Phương pháp đệ quy trên xuống

- Với mỗi luật (sản xuất)có dạng

  <phrase1> → E

  bộ phân tích cú pháp có một hàm để phân tích luật E

```
void compilePhrase1( )
{  /* phân tích luật E */   }
```

- E là một dãy có thể chứa ký hiệu kết thúc và không kết thúc
- Văn phạm cần không đệ quy trái

Phân tích một luật (một vế phải)

- Một vế phải có thể gồm một dãy ký hiệu $Y_1\ Y_2\ Y_3 \ldots Y_n$
  phải xem xét từng ký hiệu
- Mỗi ký hiệu không kếtthúc tương ứng một hàm

  `compileY`
- Mỗi ký hiệu kết thúc tương ứng với lời gọi hàm

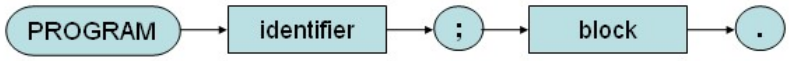
  `eat(y)`

  để kiểm tra xem y có phải là ký hiệu kếtthúc tiếp theo được sinh râ bằng cách áp dụng các luật cú pháp hay không
  - Ký hiệu kết tthúc chính là từ tố do bộ scanner đưaa ra
  - Nếu biến lookAhead chứa ký hiệu tiếp theo đúng với văn phạm thì chuyển xét từ tố mới, ngược lại báo lỗi

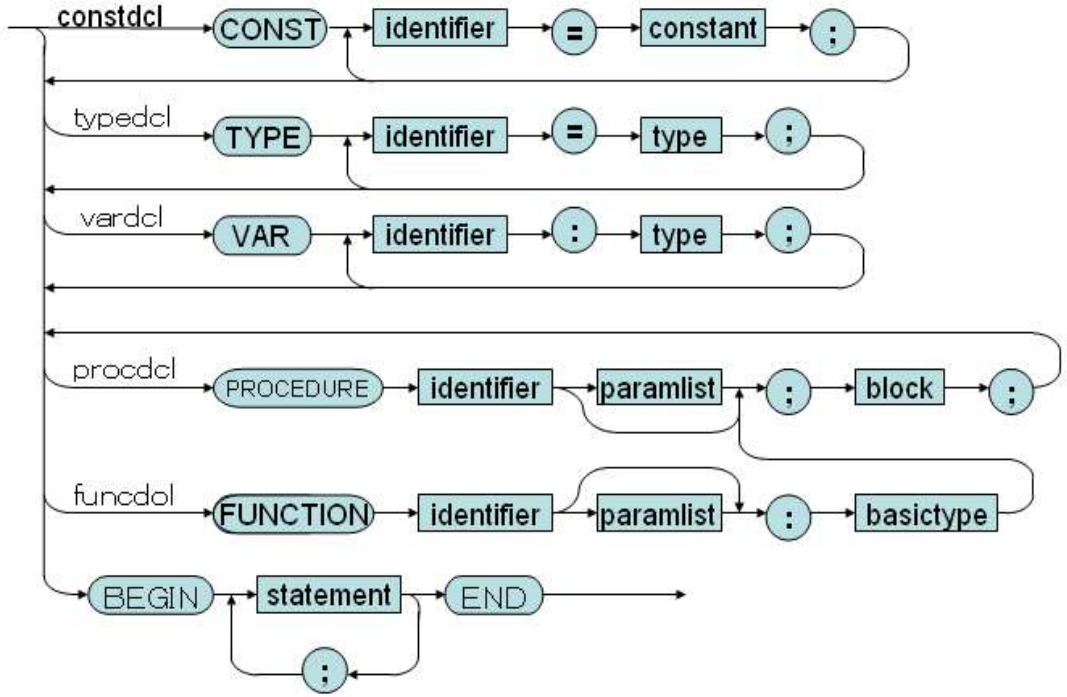
```
eat(y)://giả ngữ
        if (lookAhead == y)
        then getNextToken()
        else SyntaxError()
```

Sơ đồ cú pháp của KPL

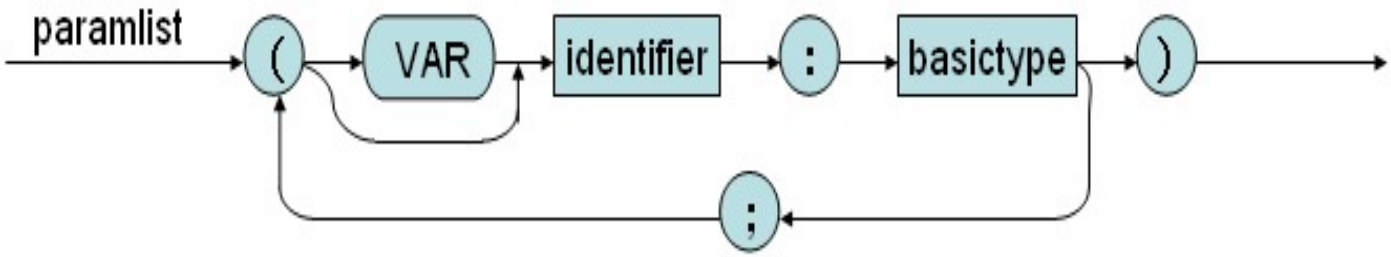
program

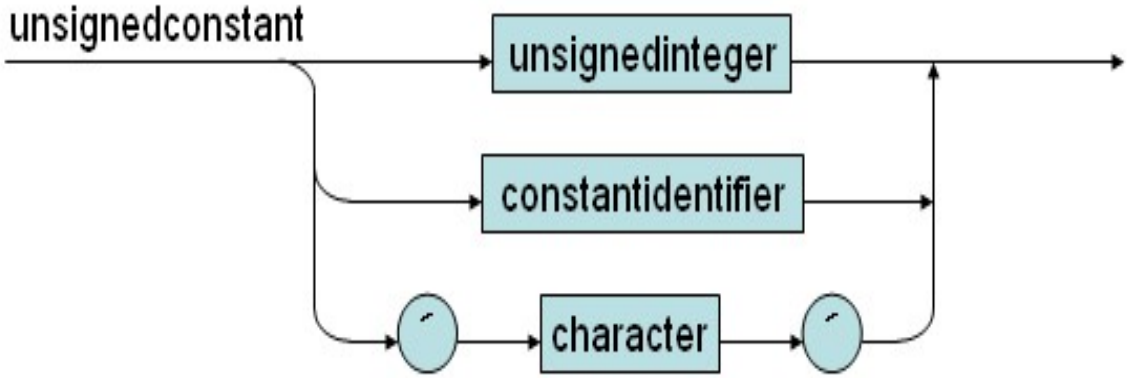


block

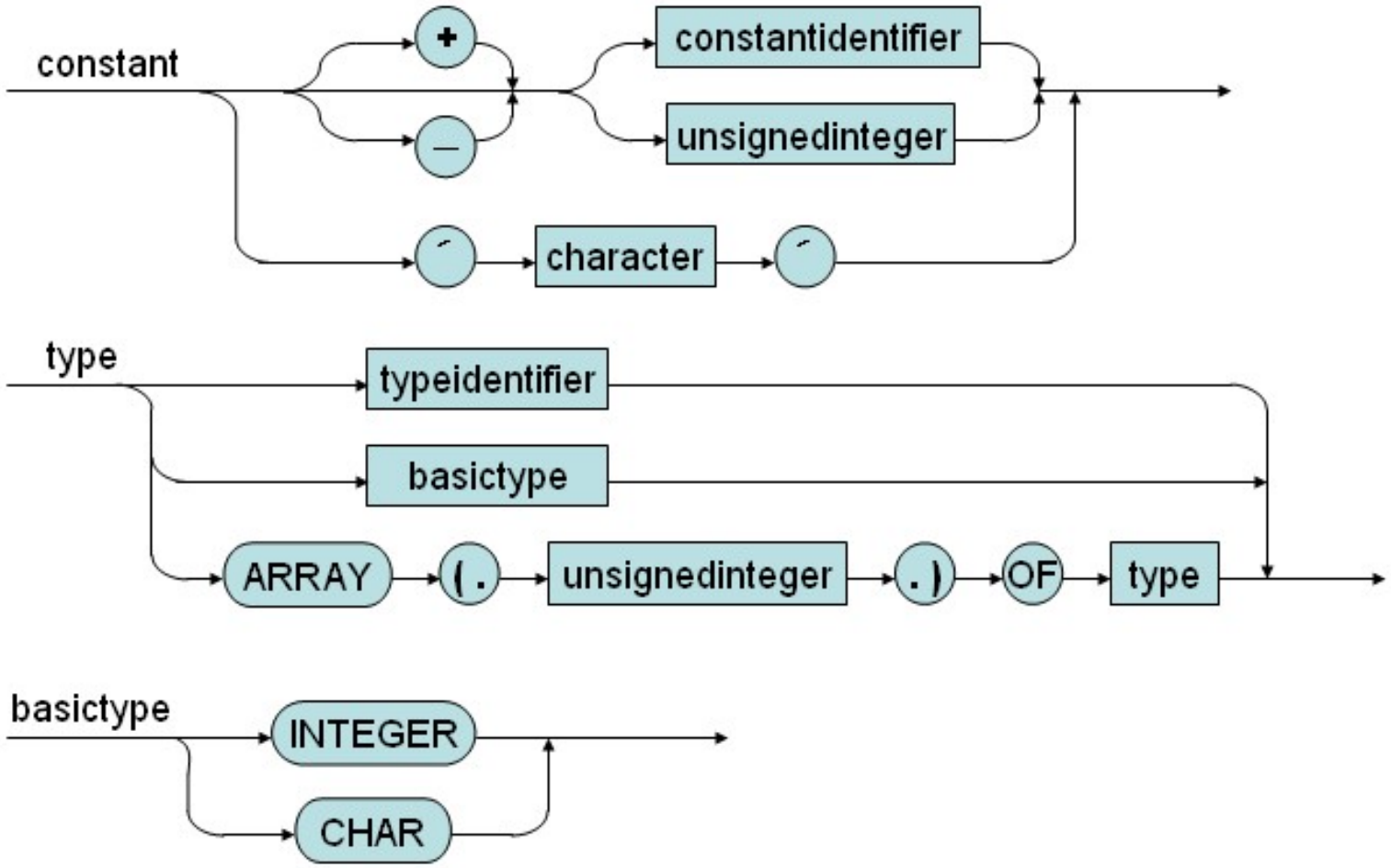


Sơ đồ cú pháp của KPL

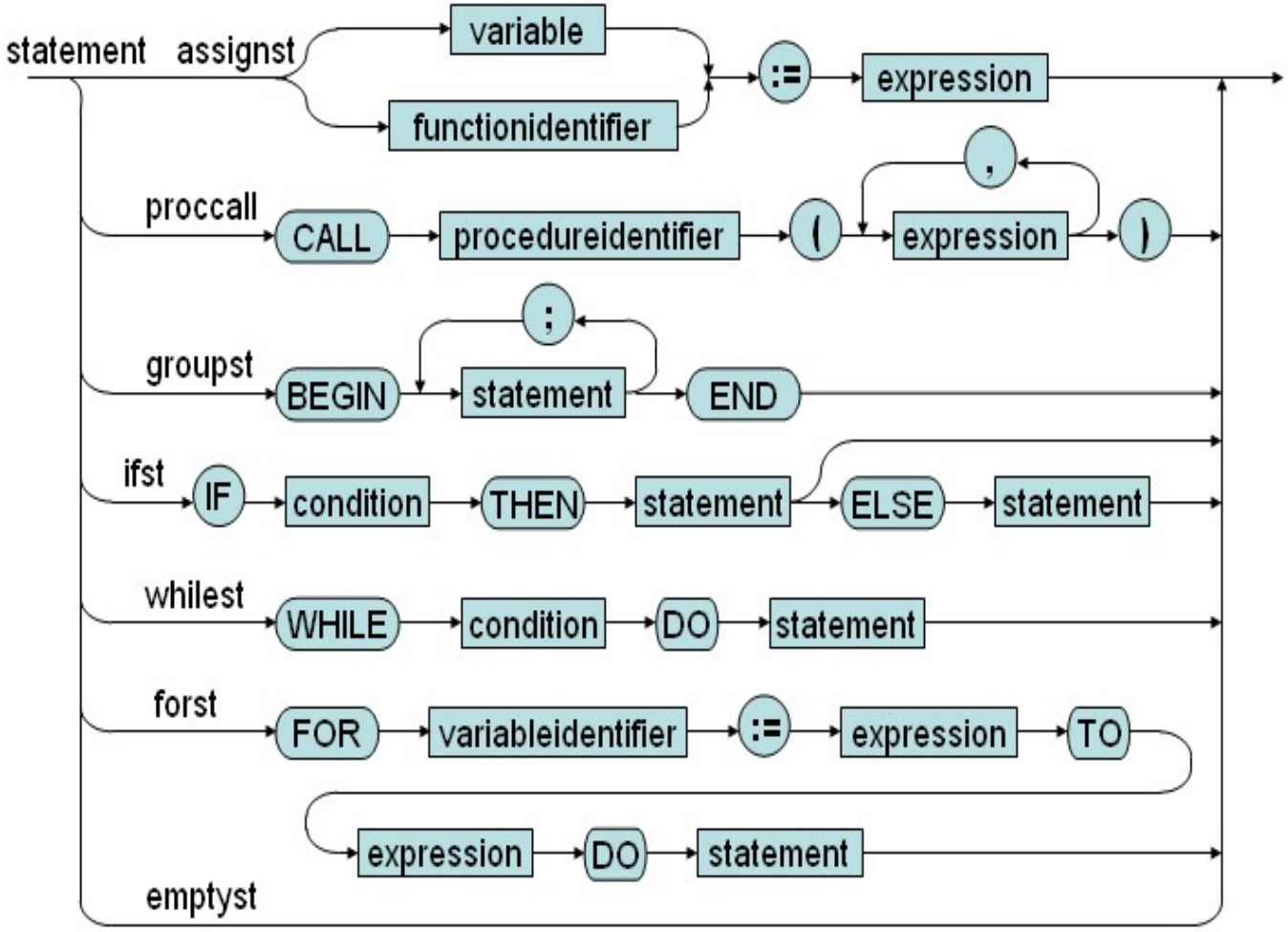




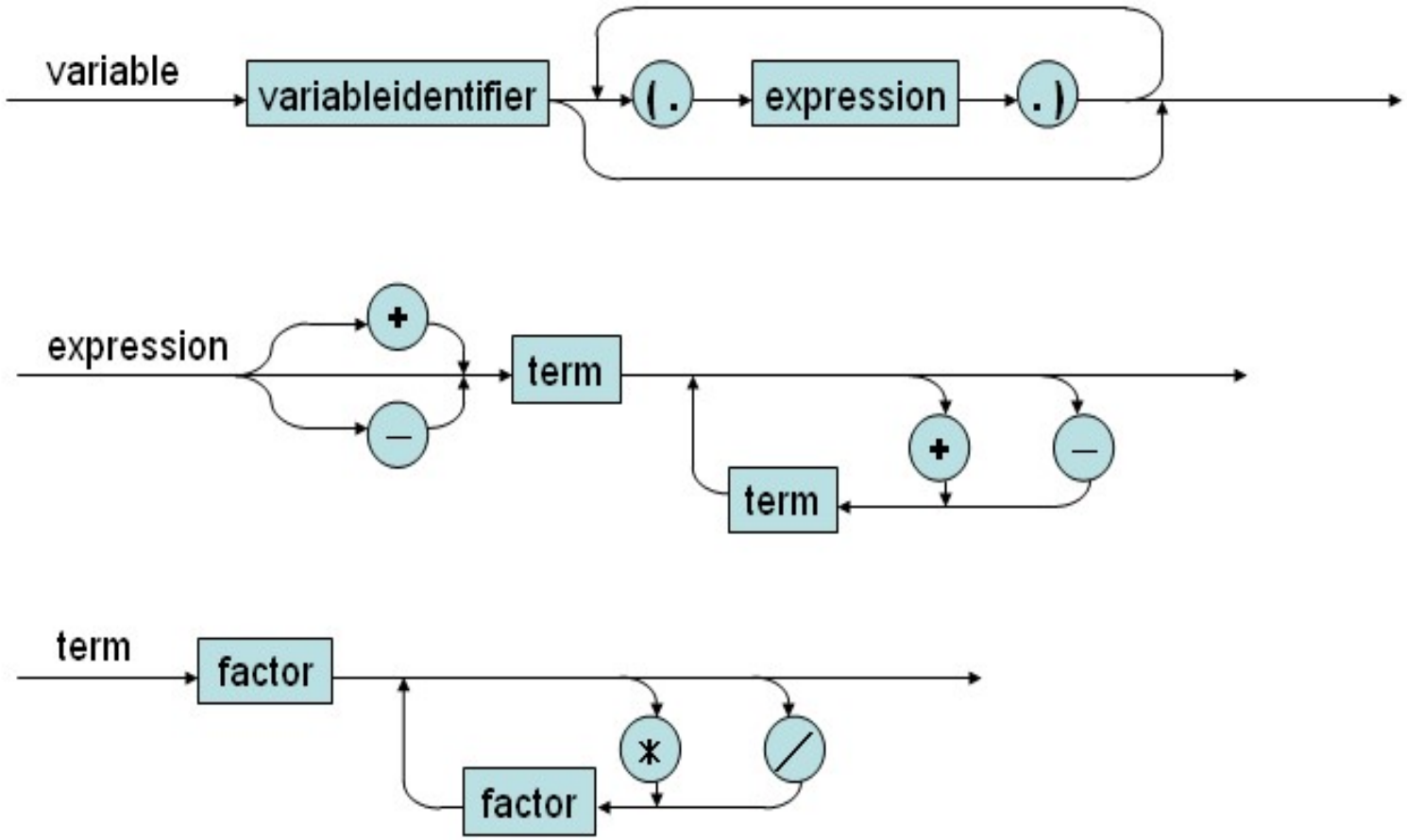
Sơ đồ cú pháp của KPL



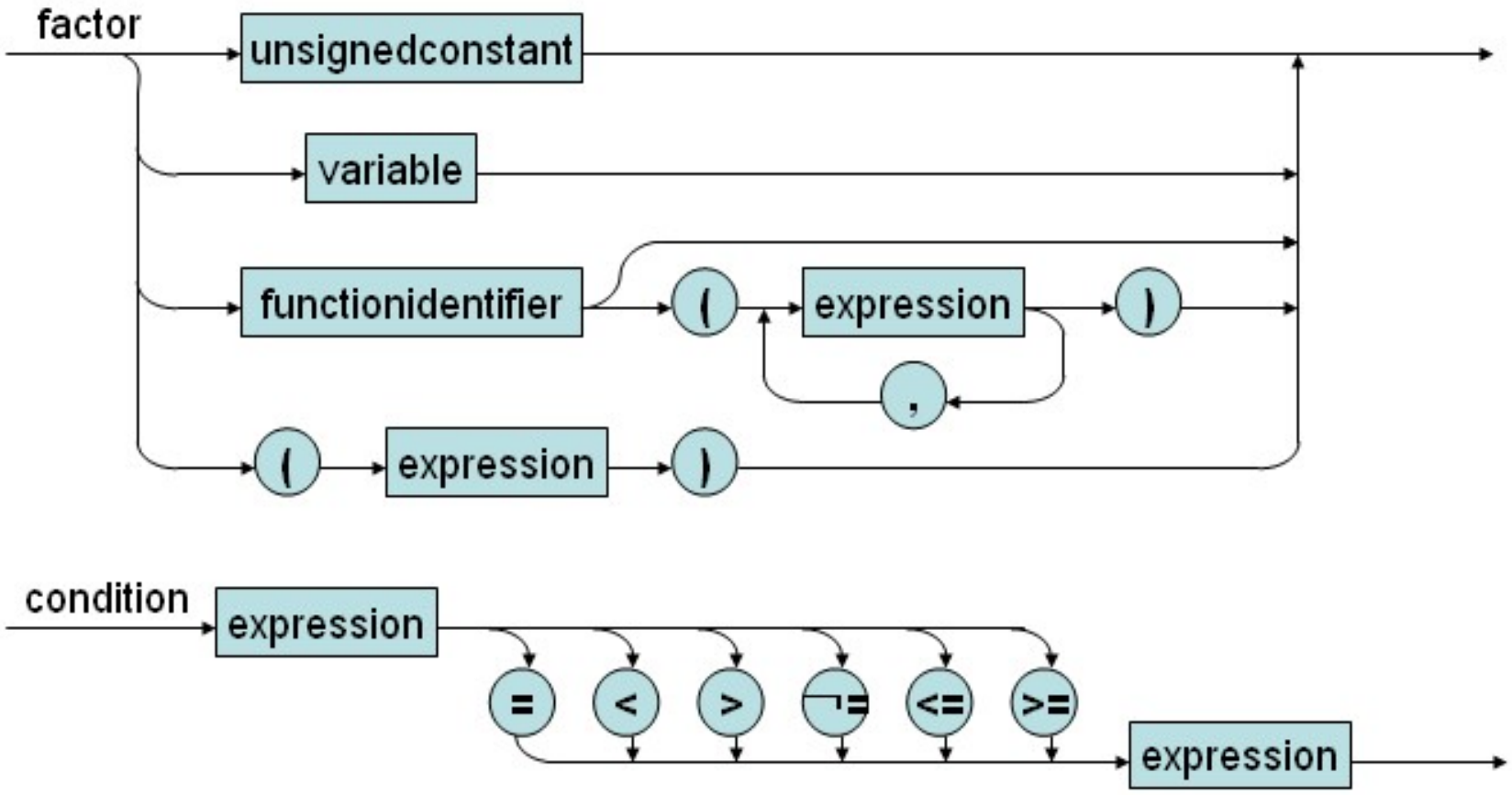
Sơ đồ cú pháp của KPL



Sơ đồ cú pháp của KPL

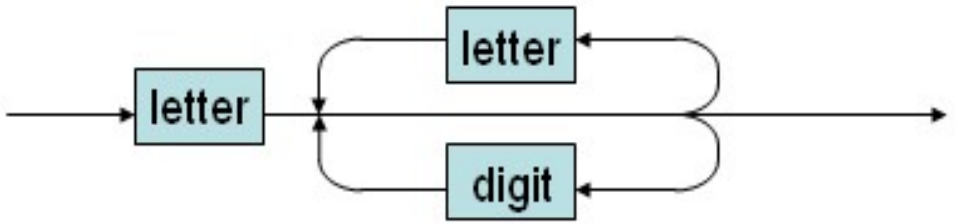


Sơ đồ cú pháp của KPL



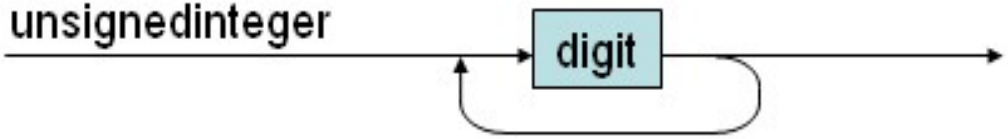
Sơ đồ cú pháp của KPL

identifier
constantidentifier
typeidentifier
procedureidentifier
functionidentifier
variableidentifier



※ digit, letter, character can be expressed with syntax chart.

unsignedinteger



Văn phạm KPL biểu diễn bằng BNF

- Chuyển từ tập sơ đồ cú pháp sang văn phạm
- Khử đệ quy trái (đã thực hiện)
- Nhân tử trái (đã thực hiện)

Văn phạm KPL biểu diễn bằng BNF

```
01) <Prog> ::= KW_PROGRAM TK_IDENT SB_SEMICOLON <Block> SB_PERIOD

02) <Block>  ::= KW_CONST <ConstDecl> <ConstDecls> <Block2>
03) <Block>  ::= <Block2>

04) <Block2> ::= KW_TYPE <TypeDecl> <TypeDecls> <Block3>
05) <Block2> ::= <Block3>

06) <Block3> ::= KW_VAR <VarDecl> <VarDecls><Block4>
07) <Block3> ::= <Block4>

08) <Block4> ::= <SubDecls><Block5>
09) <Block4> ::= <Block5>

10) <Block5> ::= KW_BEGIN <Statements> KW_END
```

Văn phạm KPL biểu diễn bằng BNF

11) <ConstDecls>::= <ConstDecl> <ConstDecls>
12) <ConstDecls>::= ε

13) <ConstDecl> ::= TK_IDENT SB_EQUAL <Constant> SB_SEMICOLON

14) <TypeDecls> ::= <TypeDecl> <TypeDecls>
15) <TypeDecls> ::= ε

16) <TypeDecl> ::= TK_IDENT SB_EQUAL <Type> SB_SEMICOLON

17) <VarDecls>::= <VarDecl> <VarDecls>
18) <VarDecls>::= ε
19) <VarDecl>   ::= TK_IDENT SB_COLON <Type> SB_SEMICOLON

20) <SubDecls> ::= <FunDecl> <SubDecls>
21) <SubDecls> ::= <ProcDecl> <SubDecls>
22) <SubDecls> ::= ε

Văn phạm KPL biểu diễn bằng BNF

```
23) <FunDecl>    ::= KW_FUNCTION TK_IDENT <Params> SB_COLON
                     <BasicType> SB_SEMICOLON <Block> SB_SEMICOLON

24) <ProcDecl>   ::= KW_PROCEDURE TK_IDENT <Params> SB_SEMICOLON
                     <Block> SB_SEMICOLON

25) <Params>     ::= SB_LPAR <Param> <Params2> SB_RPAR
26) <Params>     ::= ε

27) <Params2> ::= SB_SEMICOLON <Param> <Params2>
28) <Params2> ::= ε

29) <Param>    ::= TK_IDENT SB_COLON <BasicType>
30) <Param>    ::= KW_VAR TK_IDENT SB_COLON <BasicType>
```

Văn phạm KPL biểu diễn bằng BNF

```
31) <Type> ::= KW_INTEGER
32) <Type> ::= KW_CHAR
33) <Type> ::= TK_IDENT
34) <Type> ::= KW_ARRAY SB_LSEL TK_NUMBER SB_RSEL KW_OF <Type>

35) <BasicType> ::= KW_INTEGER
36) <BasicType> ::= KW_CHAR

37) <UnsignedConstant> ::= TK_NUMBER
38) <UnsignedConstant> ::= TK_IDENT
39) <UnsignedConstant> ::= TK_CHAR

40) <Constant> ::= SB_PLUS <Constant2>
41) <Constant> ::= SB_MINUS <Constant2>
42) <Constant> ::= <Constant2>
43) <Constant> ::= TK_CHAR

44) <Constant2>::= TK_IDENT
45) <Constant2>::= TK_NUMBER
```

Văn phạm KPL biểu diễn bằng BNF

```
46) <Statements> ::= <Statement> <Statements2>

47) <Statements2> ::= SB_SEMICOLON <Statement> <Statements2>
48) <Statements2> ::= ε

49) <Statement> ::= <AssignSt>
50) <Statement> ::= <CallSt>
51) <Statement> ::= <GroupSt>
52) <Statement> ::= <IfSt>
53) <Statement> ::= <WhileSt>
54) <Statement> ::= <ForSt>
55) <Statement> ::= ε
```

Văn phạm KPL biểu diễn bằng BNF

```
56) <AssignSt> ::= <Variable> SB_ASSIGN <Expression>
57) <AssignSt> ::= TK_IDENT SB_ASSIGN <Expression>

58) <CallSt>   ::= KW_CALL TK_IDENT <Arguments>

59) <GroupSt>  ::= KW_BEGIN <Statements> KW_END

60) <IfSt>     ::= KW_IF <Condition> KW_THEN <Statement> <ElseSt>

61) <ElseSt>   ::= KW_ELSE <Statement>
62) <ElseSt>   ::= ε

63) <WhileSt>  ::= KW_WHILE <Condition> KW_DO <Statement>
64) <ForSt>    ::= KW_FOR TK_IDENT SB_ASSIGN <Expression> KW_TO
                   <Expression> KW_DO <Statement>
```

Văn phạm KPL biểu diễn bằng BNF

```
65) <Arguments> ::= SB_LPAR <Expression> <Arguments2> SB_RPAR
66) <Arguments> ::= ε

67) <Arguments2>::= SB_COMMA <Expression> <Arguments2>
68) <Arguments2>::= ε

68) <Condition> ::= <Expression> <Condition2>

69) <Condition2>::= SB_EQ <Expression>
70) <Condition2>::= SB_NEQ <Expression>
71) <Condition2>::= SB_LE <Expression>
72) <Condition2>::= SB_LT <Expression>
73) <Condition2>::= SB_GE <Expression>
74) <Condition2>::= SB_GT <Expression>
```

Văn phạm KPL biểu diễn bằng BNF

```
75) <Expression> ::= SB_PLUS <Expression2>
76) <Expression> ::= SB_MINUS <Expression2>
77) <Expression> ::= <Expression2>

78) <Expression2> ::= <Term> <Expression3>

79) <Expression3> ::= SB_PLUS <Term> <Expression3>
80) <Expression3> ::= SB_MINUS <Term> <Expression3>
81) <Expression3> ::= ε

82) <Term> ::= <Factor> <Term2>

83) <Term2> ::= SB_TIMES <Factor> <Term2>
84) <Term2> ::= SB_SLASH <Factor> <Term2>
85) <Term2> ::= ε
86) <Factor> ::= <UnsignedConstant>
87) <Factor> ::= <Variable>
88) <Factor> ::= <FunctionApptication>
89) <Factor> ::= SB_LPAR <Expression> SB_RPAR
```

Văn phạm KPL biểu diễn bằng BNF

90) <Variable> ::= TK_IDENT <Indexes>

91) <FunctionApplication> ::= TK_IDENT <Arguments>

92) <Indexes> ::= SB_LSEL <Expression> SB_RSEL <Indexes>
93) <Indexes> ::= ε

Vào ra trong KPL KPL

- Vào: dùng các hàm
  - ReadI: Đọc một số nguyên. Không tham số
  - ReadC: Đọc một ký tự. Không tham số

  Ví dụ

  ```
  var a: integer;
  a:= ReadI;
  ```
- Ra: Dùng thủ tục
  - WriteI: In một số nguyên. 1 tham số
  - WriteC: In một ký tự. 1 tham số
  - WriteLn: In dấu xuống dòng.

  Ví dụ

  ```
  call WriteI(a);
  call WriteLn;
  ```

Ví dụ: chương trình KPL

- Viết hàm tính bình phương của một số nguyên.
- Viết một chương trình tính tổng bình phương của n số tự nhiên đầu tiên. N đọc từ bàn phím.

Lời giải

```
program example5;
(* sum of the squares of the first n natural
numbers *)
var n : integer;i: integer;sum: integer;

function f(k : integer) : integer;
  begin
    f := k * k;
  end;

BEGIN
      n := readI;
      sum :=0;
      for i:=1 to n do
            sum:= sum + f(i);
      call writeln;
      call writeI(sum);
END.  (* example*)
```

Cài đặt bộ phân tích cú pháp KPL

- Nói chung văn phạm KPL là LL(1)
- Thiết kế bộ phân tích cú pháp đệ quy trên xuống
  - *lookAhead*
  - Phân tích ký hiệu kết thsuc
  - Phân tích ký hiệu không kết thúc
    - Xây dựng bảng phân tích
      - Tính các tập FIRST() và FOLLOW()
- Example

```
02) Block  ::= KW_CONST ConstDecl ConstDecls Block2  =>RHS1
03) Block  ::= Block2                                =>RHS2
FIRST(RHS1)={KW_CONST}
FIRST(RHS2)={KW_TYPE, KW_VAR, KW_FUNCTION,
  KW_PROCEDURE,KW_BEGIN}
FIRST(RHS1)∩ FIRST(RHS2)=∅

LookAhead =KW_BEGIN =>RHS2 is chosen =>LL(1)
```

Biến lookAhead

- Đọc từ tố tiếp sau trên chương trình nguồn

```
Token *currentToken;
Token *lookAhead;

void scan(void) {
  Token* tmp = currentToken;
  currentToken = lookAhead;
  lookAhead = getValidToken();
  free(tmp);
}
```

Hàm eat

```
void eat(TokenType tokenType) {
  if (lookAhead->tokenType == tokenType) {
    printToken(lookAhead);
    scan();
  } else
   missingToken(tokenType, lookAhead->lineNo, lookAhead->colNo);
}
```

Khởi tạo bộ PTCP

```
int compile(char *fileName) {
  if (openInputStream(fileName) == IO_ERROR)
    return IO_ERROR;

  currentToken = NULL;
  lookAhead = getValidToken();

  compileProgram();

  free(currentToken);
  free(lookAhead);
  closeInputStream();
  return IO_SUCCESS;
}
```

Phân tích một ký hiệu KKT, có 1 vế phải

Example: **Program**

```
1)<Prog>::= KW_PROGRAM TK_IDENT SB_SEMICOLON <Block> SB_PERIOD

void compileProgram(void) {
  assert("Parsing a Program ....");
  eat(KW_PROGRAM);
  eat(TK_IDENT);
  eat(SB_SEMICOLON);
  compileBlock();
  eat(SB_PERIOD);
  assert("Program parsed!");
}
```

Vế trái có nhiều hơn 1 vế phải

Luật cho Basic Type

35) <BasicType> ::= KW_INTEGER
36) <BasicType> ::= KW_CHAR

```
void compileBasicType(void) {
  switch (lookAhead->tokenType) {
  case KW_INTEGER:
    eat(KW_INTEGER);
    break;
  case KW_CHAR:
    eat(KW_CHAR);
    break;
  default:
    error(ERR_INVALIDBASICTYPE, lookAhead->lineNo,
lookAhead->colNo);
    break;
  }
}
```

Phân tích statement

Statement: 7 vế phải trong đó có 1 vế phải là xâu rỗng

FIRST(<Statement>) = {TK_IDENT, KW_CALL, KW_BEGIN, KW_IF, KW_WHILE, KW_FOR, ε}

FOLLOW(<Statement>) = {SB_SEMICOLON, KW_END, KW_ELSE}

/* Predict parse table for Expression */

Input	Production
TK_IDENT	49) <Statement> ::= <AssignSt>
KW_CALL	50) <Statement> ::= <CallSt>
KW_BEGIN	51) <Statement> ::= <GroupSt>
KW_IF	52) <Statement> ::= <IfSt>
KW_WHILE	53) <Statement> ::= <Whilst?
KW_FOR	54) <Statement> ::= <ForSt?
SB_SEMICOLON	55) ε
KW_END	55) ε
KW_ELSE	55) ε
Khác	Error

Hàm phân tích statement

```
void compileStatement(void) {
  switch (lookAhead->tokenType)
{
  case TK_IDENT:
    compileAssignSt();
    break;
  case KW_CALL:
    compileCallSt();
    break;
  case KW_BEGIN:
    compileGroupSt();
    break;
  case KW_IF:
    compileIfSt();
    break;
  case KW_WHILE:
    compileWhileSt();
    break;
  case KW_FOR:
    compileForSt();
    break;
    // check FOLLOW tokens
  case SB_SEMICOLON:
  case KW_END:
  case KW_ELSE:
    break;
    // Error occurs
  default:
    error(ERR_INVALIDSTATEMENT,
lookAhead->lineNo, lookAhead-
>colNo);
    break;
  }
}
```

Xủ lý chu trình

Có thể dung đệ quy và tính FOLLOW. Tuy nhiên có thể viết ngắn gọn hơn khi biết đây thực chất là một chu trình trong sơ đồ cú pháp

10) <ConstDecls>::= <ConstDecl> <ConstDecls>

11) <ConstDecls>::= ε

```
void compileConstDecls(void) {
  while (lookAhead->tokenType == TK_IDENT)
    compileConstDecl();
}
```

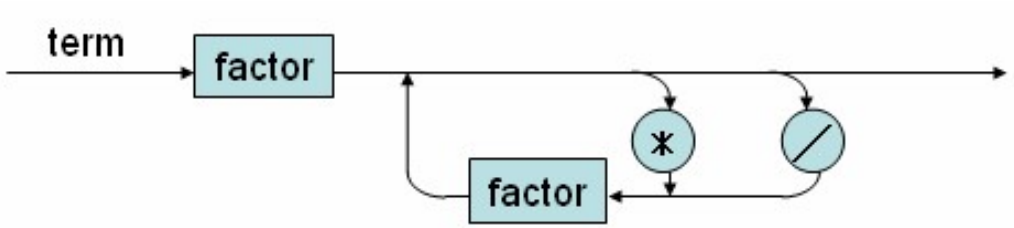
Có thể kết hợp sơ đồ cú pháp

Syntax of Term (using BNF)

```
82) <Term> ::= <Factor> <Term2>

83) <Term2> ::= SB_TIMES <Factor> <Term2>
84) <Term2> ::= SB_SLASH <Factor> <Term2>
85) <Term2> ::= ε
```

Syntax of Term (using Syntax Diagram)

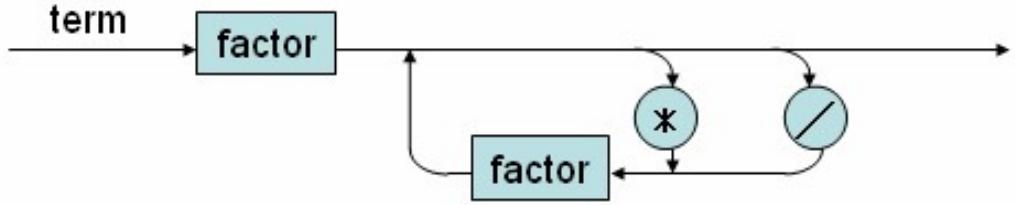


Xử lý term và term 2 theo luật, tính Follow

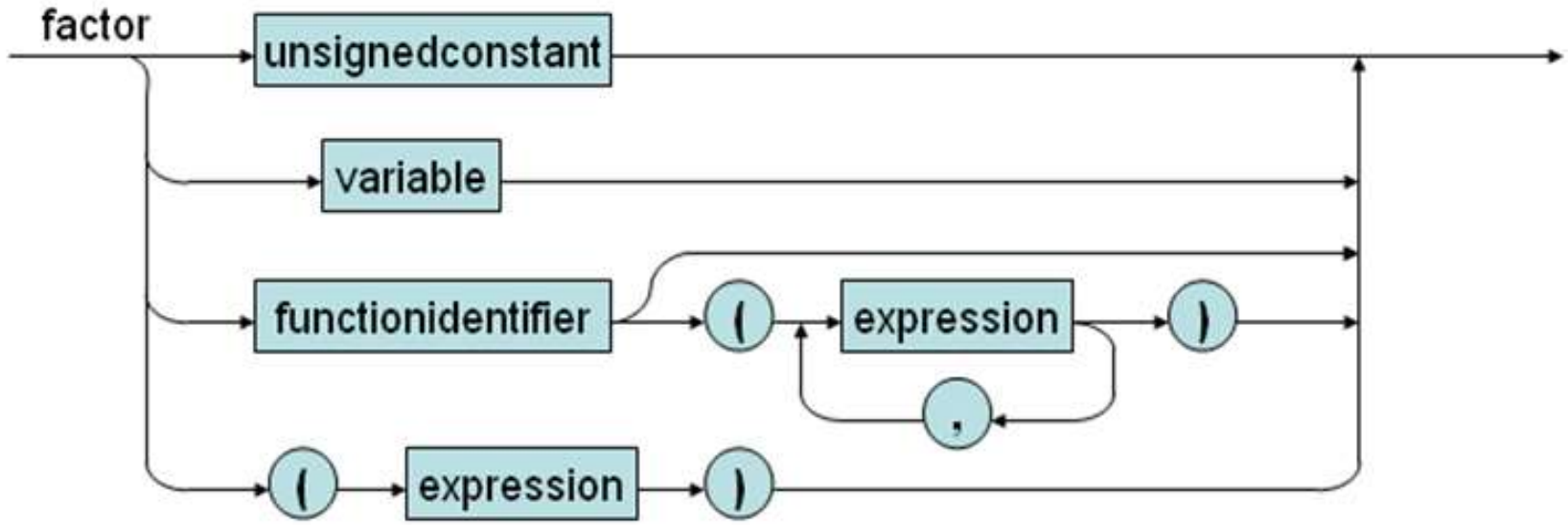
```
void compileTerm(void)
{ compileFactor();
  compileTerm2();
}
void compileTerm2(void) {
  switch (lookAhead->tokenType) {
  case SB_TIMES:
    eat(SB_TIMES);
    compileFactor();
    compileTerm2();
    break;
  case SB_SLASH:
    eat(SB_SLASH);
    compileFactor();
    compileTerm2();
    break;
// check the FOLLOW set
  case SB_PLUS:
  case SB_MINUS:
  case KW_TO:
  case KW_DO:
  case SB_RPAR:
  case SB_COMMA:
  case SB_EQ:
  case SB_NEQ:
  case SB_LE:
  case SB_LT:
  case SB_GE:
  case SB_GT:
  case SB_RSEL:
  case SB_SEMICOLON:
  case KW_END:
  case KW_ELSE:
  case KW_THEN:
    break;
  default:
    error(ERR_INVALIDTERM, lookAhead->lineNo,
lookAhead->colNo);
  }
}
```

Xử lý term theo sơ đồ cú pháp

```
void compileTerm(void)
{compileFactor();
  while(lookAhead->tokenType== SB_TIMES ||
  lookAhead->tokenType == SB_SLASH)
{switch (lookAhead->tokenType)
{
  case SB_TIMES:
    eat(SB_TIMES);
    compileFactor();
    break;
  case SB_SLASH:
    eat(SB_SLASH);
    compileFactor();
    break;
}
}
}
```



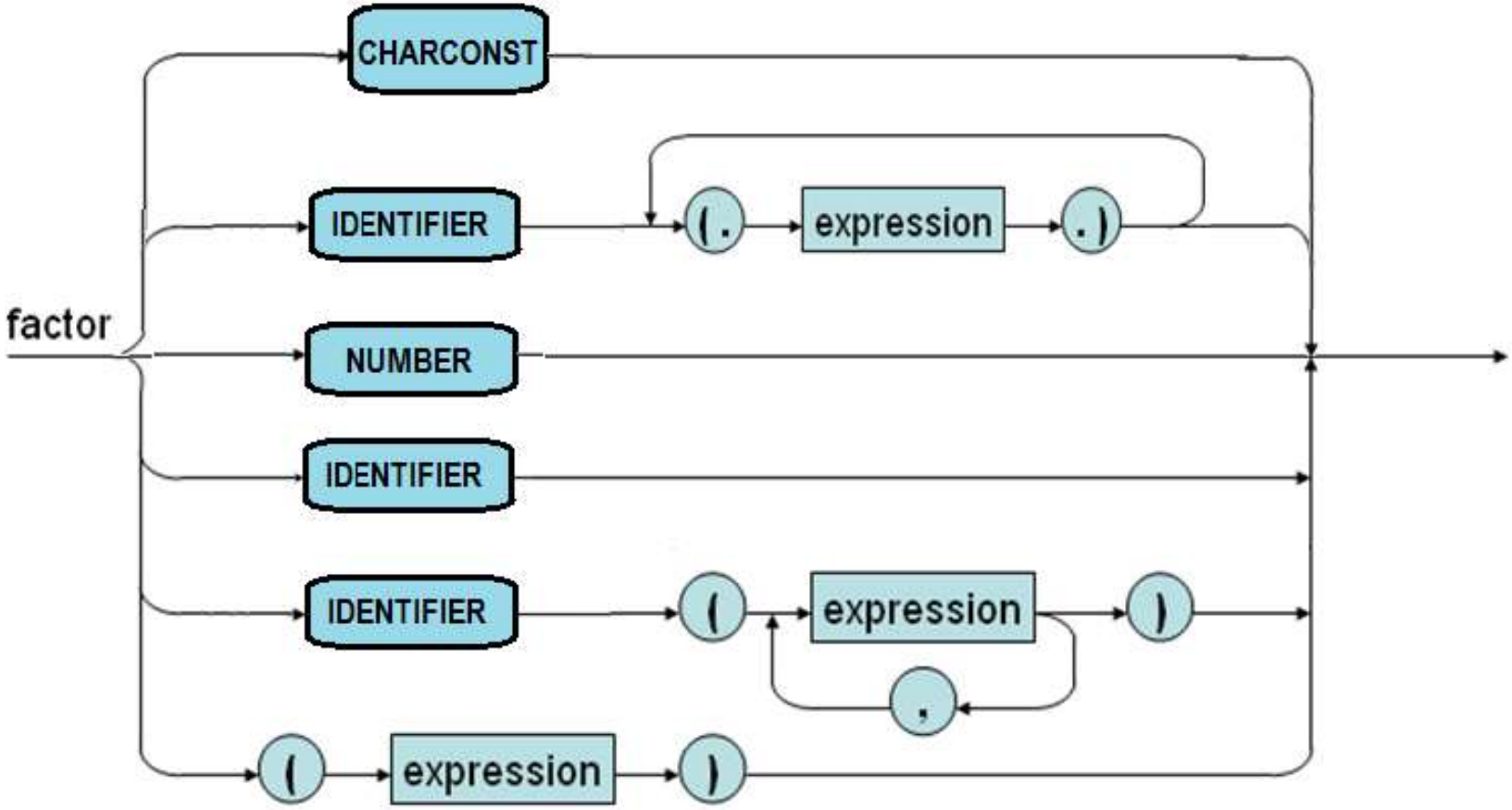
Factor: Vi phạm điều kiện LL(1)



FIRST(unsignedconstant) = {TK_NUMBER, TK_IDENT, TK_CHAR)
FIRST(variable) = {TK_IDENT}
FIRST(functioncall) = {TK_IDENT}
FIRST(unsignedconstant) ∩ FIRST(functioncall) = {TK_IDENT)
FIRST(variable) ∩ FIRST(functioncall) = {TK_IDENT}
FIRST(variable) ∩ FIRST(unsignedconstant)}= {TK_IDENT}

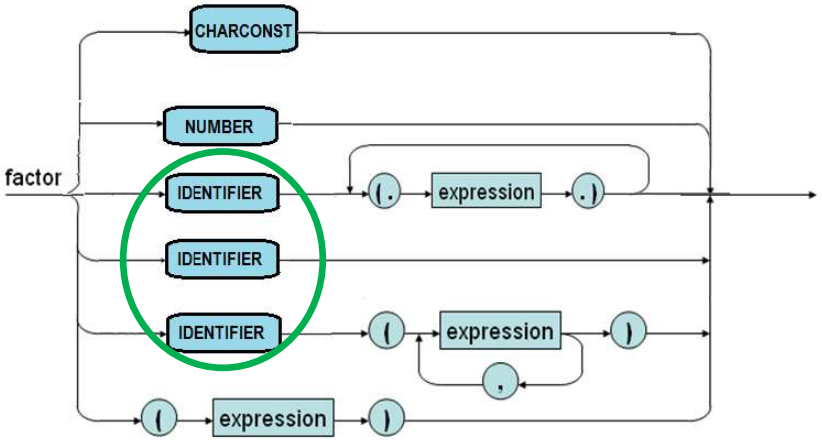
=>violation of LL(1) condition

Sau khi sắp xếp lại



Phân tích factor

```
void compileFactor(void) {
  switch (lookAhead->tokenType) {
  case TK_NUMBER:
    eat(TK_NUMBER);
    break;
  case TK_CHAR:
    eat(TK_CHAR);
    break;
  case TK_IDENT:
    eat(TK_IDENT);
    switch (lookAhead->tokenType) {
    case SB_LSEL:
      compileIndexes();
      break;
    case SB_LPAR:
      compileArguments();
      break;
    default: break;
    }
    break;
```




```
  case SB_LPAR:
      eat(SB_LPAR);
      compileExpression();
      eat(SB_RPAR);
      break;
    default:
      error(ERR_INVALIDFACTOR,
lookAhead->lineNo, lookAhead->colNo);
    }
}
```