Bài 9. Phương pháp đệ quy trên xuống

Đặc điểm của phương pháp

- Sử dụng để phân tích cú pháp cho các văn phạm LL(1)
- Có thể mở rộng cho văn phạm LL(k), nhưng việc tính toán phức tạp
- Sử dụng để phân tích văn phạm khác có thể dẫn đến lặp vô hạn

Bộ phân tích cú pháp

- Bao gồm một tập thủ tục, mỗi thủ tục ứng với một sơ đồ cú pháp (một ký hiệu không kết thúc)
- Các thủ tục đệ quy : khi triển khai một ký hiệu không kết thúc có thể gặp các ký hiệu không kết thúc khác, dẫn đến các thủ tục gọi lẫn nhau, và có thể gọi trực tiếp hoặc gián tiếp đến chính nó.

Mô tả chức năng

- Giả sử mỗi thủ tục hướng tới một đích ứng với một sơ đồ cú pháp
- Tại mỗi thời điểm luôn có một đích được triển khai, kiểm tra cú pháp hết một đoạn nào đó trong văn bản nguồn

Thủ tục triển khai một đích

- Đối chiếu văn bản nguồn với một đường trên sơ đồ cú pháp
- Đọc từ tố tiếp
- Đối chiếu với nút tiếp theo trên sơ đồ
- Nếu là nút tròn (ký hiệu kết thúc)thì từ tố vừa đọc phải phù hợp với từ tố trong nút
- Nếu là nút chữ nhật nhãn A (ký hiệu không kết thúc), từ tố vừa đọc phải thuộc FIRST (A) => tiếp tục triển khai đích A
- Ngược lại, thông báo một lỗi cú pháp tại điểm đang xét

Từ sơ đồ thành thủ tục

- Mỗi nút trên sơ đồ ứng với một thủ tục
- Các nút xuất hiện tuần tự chuyển thành các câu lệnh kế tiếp nhau.
- Các điểm rẽ nhánh chuyển thành câu lệnh lựa chọn (if, case)
- Chu trình chuyển thành câu lệnh lặp (while, do while, repeat. . .)
- Nút tròn chuyển thành đoạn đối chiếu từ tố
- Nút chữ nhật chuyển thành lời gọi tới thủ tục khác

Chú ý

- Bộ phân tích cú pháp luôn đọc trước một từ tố
- Xem trước một từ tố cho phép chọn đúng đường đi khi gặp điểm rẽ nhánh trên sơ đồ cú pháp
- Khi thoát khỏi thủ tục triển khai một đích, có một từ tố đã được đọc dôi ra

Bộ phân tích cú pháp KPL

- void error (const char msg[]);
- int accept(symbol s); // kiểm tra s có phải là symbol không?
- int expect(symbol s); // kiểm tra s có phải là symbol cần đọc không?
- void factor(void);//phân tích nhân tử
- void term(void);//phân tích số hạng
- void expression(void); // phân tích biểu thức
- void condition(void); // phân tích điều kiện
- void statement(void); // phân tích câu lệnh
- void block(void); // phân tích các khối câu lệnh
- void basictype(void); // các kiểu biến cơ bản
- void program();

Hàm accept

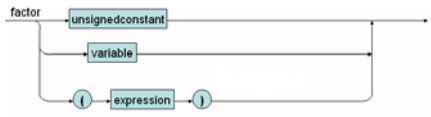
```
int accept(symbol s)
{
   if (sym == s)
   {
        getsym();
        return 1;
   }
   return 0;
}
```

Hàm expect

```
int expect(symbol s)
{
   if(accept(s))
        return 1;
   error("expect: unexpected symbol");
   return 0;
}
```

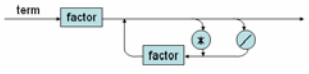
Phân tích factor

```
void factor(void)
{if(accept(ident){}
else
     if(accept(number)) {}
          else if(accept(lparen))
     {
          expression();
          expect(rparen);
     }
     else
     {
          error("factor: syntax error");
          getsym();
     }
}
```



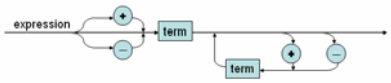
Phân tích term

```
void term(void)
{
   factor();
   while(sym == times || sym == slash)
   {
        getsym();
        factor();
   }
}
```



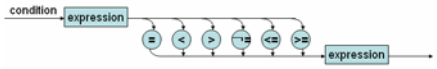
Phân tích expression

```
void expression(void)
{
    if(sym == plus || sym == minus)
       getsym();
    term();
    while(sym == plus || sym == minus)
    {
       getsym();
       term();
    }
}
```

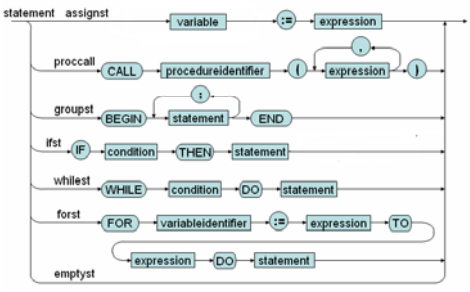


Phân tích condition

```
void condition(void)
{
    expression();
    if(sym == eql || sym == neq || sym == lss || sym == leq || sym ==
    grt || sym == geq)
    {
       getsym();
       expression();
    }
    else
    {
       error("condition: syntax error");
    }
}
```



Sơ đồ cú pháp của lệnh KPL



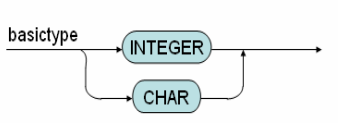
Phân tích statement

```
void statement(void)
{
        if(accept(ident))
        {
                expect(becomes);
                expression();
                // variable :=
                        }
        else if(accept(callsym))
        {
                expect(ident);
                expect(lparen);
                expression();
                while (sym == comma)
                {
                                getsym();
                                expression();
                }
```

```
expect(rparen);
        }
        else if(accept(beginsym))
        {
                statement();
                while(sym == semicolon)
                {
                                getsym();
                                statement();
                }
                expect(endsym);
        }
        else if(accept(ifsym))
        {
                condition();
                expect(thensym);
                statement();
                if (accept(elsesym))
                                statement();
        }
        else if(accept(whilesym))
        {
                condition();
                expect(dosym);
                statement();
        }
        else if (accept(forsym))
        {
                expect(ident);
                expect(becomes);
                expression();
                expect(tosym);
                expression();
                expect(dosym);
                statement();
        }
        else
        {
                getsym();
        }
}
```

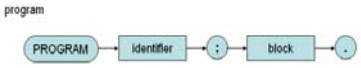
Phân tích basic type

```
void basictype()
{
    if(accept(integersym)){}
    else
      expect(charsym);
}
```



Phân tích program

```
void program()
{
    expect(programsym);
    expect(ident);
    expect(semicolon);
    block();
    if(sym == period)
    {
      printf("No error!");
      return;
    }
    else
    {
      error("Syntax error.");
    }
}
```



Phân tích block

```
void block(void)
{
    if(accept(constsym)) // const
    {
      while (accept(ident))
      {
          expect(eql);
          constant_decl();
          expect(semicolon);
      }
    }
    if (accept(typesym)) // type
    {
      while (accept(ident))
      {
          expect(eql);
          type();
          expect(semicolon);
      }
    }
    if(accept(varsym))
    {
      while (accept(ident))
      {
          expect(colon);
          type();
          expect(semicolon);
      }
    }
    while(sym == procsym)
    {
      getsym();
      expect(ident);
      if (accept(lparen))
      {
          paramlist();
          expect(rparen);
      }
      expect(semicolon);
      block();
      expect(semicolon);
    }
      expect(beginsym);
    statement();
    while(accept(semicolon))
      statement();
    expect(endsym);
}
```

Khối