

# 语法分析

## 1 语法分析的任务

语法分析的主要任务是在词法分析结果的基础上，进一步得到解析树（parsing tree）。例如，下面两个算术表达式：

```
1 + x * y
```

```
(1 + x) * y
```

它们经过词法分析结果如下：

```
TOK_NAT(1) TOK_PLUS TOK_IDENT(x) TOK_MUL TOK_IDENT(y)
```

```
TOK_LEFT_PAREN TOK_NAT(1) TOK_PLUS TOK_IDENT(x)
TOK_RIGHT_PAREN TOK_MUL TOK_IDENT(y)
```

```
      *
     /\
    ( ) y
     |
     +
    /\
   1  x
```

```
      *          *
     /\          /\
    ( ) y      + y
     |          /\
     +          1 x
    /\
   1  x
```

其中，`(1 + x) * y` 这个式子我们给出了两个树结构，一个是直接反应字符串内容的树结构，另一个是省去了括号这一冗余结构之后的结果，这个我们之前介绍过了，我们称其为抽象语法树。

## 2 上下文无关语法与解析树

本课程中将主要介绍基于上下文无关语法（Context-free grammar）的移入规约分析算法（shift-reduce parsing）。上下文无关语法从形式化与抽象语法树的语法规定是类似的，但是它也要处理括号等抽象语法树不关注的源代码信息。一套上下文无关语法包含若干条产生式（production），例如：

#### 顺序执行

```
S -> S ; S
S -> ID := E
S -> PRINT ( L )
```

打印输出

#### 表达式

```
E -> ID
E -> NAT
E -> E + E
E -> ( E )
```

#### 表达式列表

```
L -> E
L -> L , E
```

```
S -> S ; S
-> ID := E ; S
-> ID := E + E ; S
-> ID := ID + E ; S
-> ID := ID + NAT ; S
-> ID := ID + NAT ; PRINT(L)
-> ID := ID + NAT ; PRINT(L, E)
-> ID := ID + NAT ; PRINT(E, E)
-> ID := ID + NAT ; PRINT(ID, E)
-> ID := ID + NAT ; PRINT(ID, ID)
```

在上面例子中，**S** 表示语句，**E** 表示表达式，**L** 表示表达式列表。在这个语言中，**PRINT** 指令可以带多个参数，表达式中只允许出现加法运算，多条语句只允许顺序执行，没有条件分支与循环。

上下文无关语法（定义）

- 一个初始符号，例如上面例子中的：**S**；
- 一个**终结符（terminal symbols）集合**，例如上面例子中的：**ID NAT , ; ( ) + := ;**；
- 一个非终结符（nonterminal symbols）集合，例如上面例子中的：**S E L**，当词法分析器和语法分析器结合使用的时候，这个非终结符集合一般就是词法分析中的标记集合；
- 一系列产生式，每个产生式的左边是一个非终结符，每个产生式的右边是一列（可以为空）终结符或非终结符。

解析树

- 根节点为上下文无关语法的初始符号；
- 每个叶子节点是一个终结符，每个内部节点是一个非终结符；
- 每一个父节点和他的子节点构成一条上下文无关语法中的产生式；

对于所有标志串，至多只有一种解析树可以生成。

### 3 歧义与歧义的消除

**ID + ID \* ID**，我们想要默认先做乘法；但是可能有两种解析树。

可能有歧义：

可能有不止一种解析树

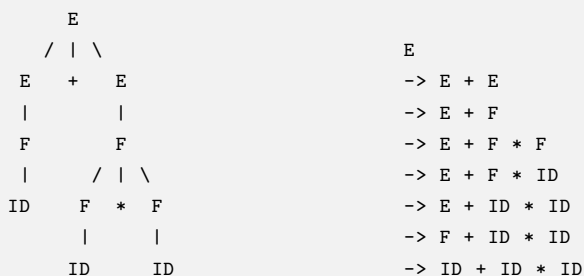
```
E -> ID      E -> E + E      E -> E * E      E -> ( E )
```

下面语法能够消除上面标记串语法分析中的歧义。

```
E -> F      E -> E + E      F -> F * F
F -> ( E )   F -> ID
```

如果考虑 **ID + ID + ID**，还是有歧义

**E - F, E - E + F, F - F \* G, F - G, G - (E), G - ID**



## 4 派生与规约

- 一个派生中,如果每次都展开最左侧的非终结符,那么这个派生就称为一个**最左派生 (left-most derivation)**;
- 一个派生中, 如果每次都展开最右侧的非终结符, 那么这个派生就称为一个**最右派生 (right-most derivation)**;
- 同一棵解析树能够唯一确定一种最左派生, 同一棵解析树能够唯一确定一种最右派生;
- 如果一串标记串没有歧义, 那么只有唯一的最左派生可以生成这一标记串, 也只有唯一的最右派生可以生成这一标记串;
- 规约是派生反向过程:**

$E = ID + (ID + ID) * ID$

reduction规约

左规约对应右派生

$ID + ID + ID$   
 $\rightarrow G + ID + ID$   
 $\rightarrow F + ID + ID$   
 $\rightarrow E + ID + ID$   
 $\rightarrow E + G + ID$   
 $\rightarrow E + F + ID$   
 $\rightarrow E + ID$   
 $\rightarrow E + G$   
 $\rightarrow E + F$   
 $\rightarrow E$

派生的顺序不唯一。

$E \rightarrow E + F$   
 $\rightarrow E + G$   
 $\rightarrow E + ID$   
 $\rightarrow E + F + ID$   
 $\rightarrow E + G + ID$   
 $\rightarrow E + ID + ID$   
 $\rightarrow F + ID + ID$   
 $\rightarrow G + ID + ID$   
 $\rightarrow ID + ID + ID$

$E = E + F$   
 $E = G + F$   
 $E = ID + F$   
 $E = ID + F * G$   
 $E = ID + G * G$   
 $E = ID + (E) * G$   
 $E = ID + (E + F) * G$   
 $E = ID + (F + F) * G$   
 $E = ID + (G + F) * G$   
 .....

请大家注意, 上图中的派生是最右派生, 但是他对应的规约却是每次尽可能地进行左侧规约。这是因为派生与规约的方向是相反的。

## 5 移入规约分析

下面将要介绍的移入规约分析, 是一个计算生成最左规约 (或者说最右派生) 的过程。移入规约分析的主要思想是: 从左向右扫描标记串, **从左向右进行规约**。

- 共有两类操作: 移入、规约;
- 扫描线的右侧全部都是终结符;
- 规约操作都发生在扫描线的左侧紧贴扫描线的区域内。**
- 例子

这里进行规约

```
| ID + ID + ID
-> ID | + ID + ID
-> G | + ID + ID
-> F | + ID + ID
-> E | + ID + ID
-> E + | ID + ID
-> E + ID | + ID
-> E + G | + ID
-> E + F | + ID
-> E | + ID
-> E + | ID
-> E + ID |
-> E + G |
-> E + F |
-> E |
```

- 移入与规约的选择

- 既能移入，又能规约的情形
- 有多种规约方案的情形

```
| ID + ID + ID
-> ID | + ID + ID
-> G | + ID + ID
-> F | + ID + ID
-> E | + ID + ID
-> E + | ID + ID
-> E + ID | + ID
-> E + G | + ID
-> E + F | + ID
```

E + E | + ID

E | + ID

E + F + | ID

三种选择

线性时间算法不允许都扫描一遍！

E + E |

E + F + | 无法完成规约！

扫描线右侧一定是Terminal Symbol (ID，可以得到额外信息)  
左侧成分复杂一些。