

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	表达式	表达式列表
S -> S ; S	E -> ID	L -> E
S -> ID := E	E -> NAT	L -> L , E
S -> PRINT (L)	E -> E + E	
顺序执行 打印输出	E -> (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。

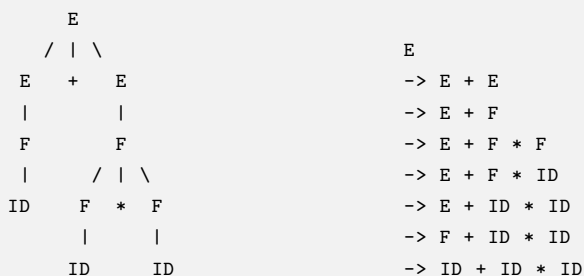
```

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。

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

```
| 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 |
```

移入与规约的选择问题：

- 既能移入，又能规约的情形，应当选择移入还是规约？
- 有多种规约方案的情形，应当如何选择？

例子：

```
| 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
```

6 已移入部分的结构

已移入部分的结构

- 已移入规约的部分可以分为 n 段；
- 每段对应一条产生式；
- 第 i 段拼接上第 $i + 1$ 条产生式的左侧非终结符是第 i 条产生式右侧符号串的一个前缀。

例子

- $F * (E) |$ 中已移入规约的部分可以分为: $F * (E)$
- 分别对应产生式: $F \rightarrow F * . G$ $G \rightarrow (E) .$
- $F * (E + | ID)$ 中已移入规约的部分可以分为: $F * (E +$
- 分别对应产生式: $F \rightarrow F * . G$ $G \rightarrow (. E)$ $E \rightarrow E + . F$

移入与规约对应的扫描线左侧结构变化:

原操作: 移入

$| ID + ID + ID \quad \rightarrow \quad ID | + ID + ID$

带结构的操作:

$(START \rightarrow . E) \quad \rightarrow \quad (START \rightarrow . E)$
 $(E \rightarrow . E + F)$
 $(E \rightarrow . E + F)$
 $(E \rightarrow . F)$
 $(F \rightarrow . G)$
 $(G \rightarrow ID .)$

看点右侧, E可能对应下面这么多情况

原操作: 规约

$ID | + ID + ID \quad \rightarrow \quad G | + ID + ID$

带结构的操作:

$(START \rightarrow . E) \quad \rightarrow \quad (START \rightarrow . E)$
 $(E \rightarrow . E + F)$
 $(E \rightarrow . E + F)$
 $(E \rightarrow . F)$
 $(F \rightarrow . G)$
 $(G \rightarrow ID .)$

原操作: 规约

$G | + ID + ID \quad \rightarrow \quad F | + ID + ID$

带结构的操作:

$(START \rightarrow . E) \quad \rightarrow \quad (START \rightarrow . E)$
 $(E \rightarrow . E + F)$
 $(E \rightarrow . E + F)$
 $(E \rightarrow . F)$
 $(F \rightarrow G .)$

原操作: 规约

$F | + ID + ID \quad \rightarrow \quad E | + ID + ID$

增加结构:

```
(START -> . E)      -> (START -> . E)
(E -> . E + F)      (E -> . E + F)
(E -> . E + F)      (E -> E . + F)
(E -> F .)
```

原操作: 移入

```
E | + ID + ID      -> E + | ID + ID
```

增加结构:

```
(START -> . E)      -> (START -> . E)
(E -> . E + F)      (E -> . E + F)
(E -> E . + F)      (E -> E . + F)
```

原操作: 移入

```
E + | ID + ID      -> E + ID | + ID
```

增加结构:

```
(START -> . E)      -> (START -> . E)
(E -> . E + F)      (E -> . E + F)
(E -> E + . F)      (E -> E + . F)
(F -> . G)
(G -> ID .)
```

7 已移入部分的结构判定

以扫描线左侧的最右段为状态, 构建 NFA。新增一段带来的变化对应一条 ϵ 边, 其他加入符号带来的变化对应一条普通边。该 NFA 中的所有状态都是终止状态, 要判断一串符号串是否是可行的扫描线左侧结构, 只需判断这个符号串能否被这个 NFA 接受。

ϵ : START -> . E 变为 E -> . F

ϵ : START -> . E 变为 E -> . E + F

E: START -> . E 变为 START -> E .

ϵ : E -> . F 变为 F -> . F * G

ϵ : E -> . F 变为 F -> . G

F: E -> . F 变为 E -> F .

ϵ : E -> . E + F 变为 E -> . F

ϵ : E -> . E + F 变为 E -> . E + F

E: E -> . E + F 变为 E -> E . + F

+: $E \rightarrow E . + F$ 变为 $E \rightarrow E + . F$

\in : $E \rightarrow E + . F$ 变为 $F \rightarrow . F * G$

\in : $E \rightarrow E + . F$ 变为 $F \rightarrow . G$

...

判断 $E + F + | \dots$ 是否是可行的扫描线左侧结构:

$E + F + | \dots$

START $\rightarrow . E$

$E \rightarrow . F$

$E \rightarrow . E + F$

$F \rightarrow . G$

$F \rightarrow . F * G$

$G \rightarrow . (E)$

$G \rightarrow . ID$

.

$E + F + | \dots$

START $\rightarrow E .$

$E \rightarrow E . + F$

.

$E + F + | \dots$

$E \rightarrow E + . F$

$F \rightarrow . G$

$F \rightarrow . F * G$

$G \rightarrow . ID$

$G \rightarrow . (E)$

.

$E + F + | \dots$

$E \rightarrow E + F .$

$F \rightarrow F . * G$

.

$E + F + | \dots$

(No possible state)

.

8 基于 follow 集合的判定法

定义

- x 是 $\text{Follow}(y)$ 的元素当且仅当 $\dots y x \dots$ 是可能被完全规约的。
- x 是 $\text{First}(y)$ 的元素当且仅当 $x \dots$ 是可能被规约为 y 的。
- 上述计算只考虑 x 是终结符的情形。

First 集合的计算

- 对任意终结符 x , x 都是 $\text{First}(x)$ 的元素。 可以规约到 z 的都可以规约到 y
- 对任意产生式 $y \rightarrow z \dots$, $\text{First}(z)$ 的元素都是 $\text{First}(y)$ 的元素。

Follow 集合的计算

- 对任意产生式 $u \rightarrow \dots y z \dots$, $\text{First}(z)$ 是 $\text{Follow}(y)$ 的子集。
- 对任意产生式 $z \rightarrow \dots y$, $\text{Follow}(z)$ 是 $\text{Follow}(y)$ 的子集。

9 移入规约分析完整过程实例

$\text{ID} * (\text{ID} + \text{ID})$ 的移入规约分析:

- 初始: $| \text{ID} * (\text{ID} + \text{ID})$
- 移入: $\text{ID} | * (\text{ID} + \text{ID})$
- 规约: $\text{G} | * (\text{ID} + \text{ID})$, 因为 $\text{ID} * | \dots$ 不可行
- 规约: $\text{F} | * (\text{ID} + \text{ID})$, 因为 $\text{G} * | \dots$ 不可行
- 移入: $\text{F} * | (\text{ID} + \text{ID})$, 因为 $*$ 不是 $\text{Follow}(\text{E})$ 中的元素
- 移入: $\text{F} * (| \text{ID} + \text{ID})$
- 移入: $\text{F} * (\text{ID} | + \text{ID})$
- 规约: $\text{F} * (\text{G} | + \text{ID})$, 因为 $\text{F} * (\text{ID} + | \dots)$ 不可行
- 规约: $\text{F} * (\text{F} | + \text{ID})$, 因为 $\text{F} * (\text{G} + | \dots)$ 不可行
- 规约: $\text{F} * (\text{E} | + \text{ID})$, 因为 $\text{F} * (\text{F} + | \dots)$ 不可行
- 移入: $\text{F} * (\text{E} + | \text{ID})$
- 移入: $\text{F} * (\text{E} + \text{ID} |)$
- 规约: $\text{F} * (\text{E} + \text{G} |)$, 因为 $\text{F} * (\text{E} + \text{ID}) | \dots$ 不可行
- 规约: $\text{F} * (\text{E} + \text{F} |)$, 因为 $\text{F} * (\text{E} + \text{G}) | \dots$ 不可行
- 规约: $\text{F} * (\text{E} |)$, 因为另外两种方案扫描线左侧的结构都不可行
- 移入: $\text{F} * (\text{E}) |$
- 规约: $\text{F} * \text{G} |$

- 规约: `F |`
- 规约: `E |` , 亦可看做 `E | EOF`
- 移入: `E EOF |`
- 规约: `START |`
- 语法分析结束