一个简单的语法制导翻译器

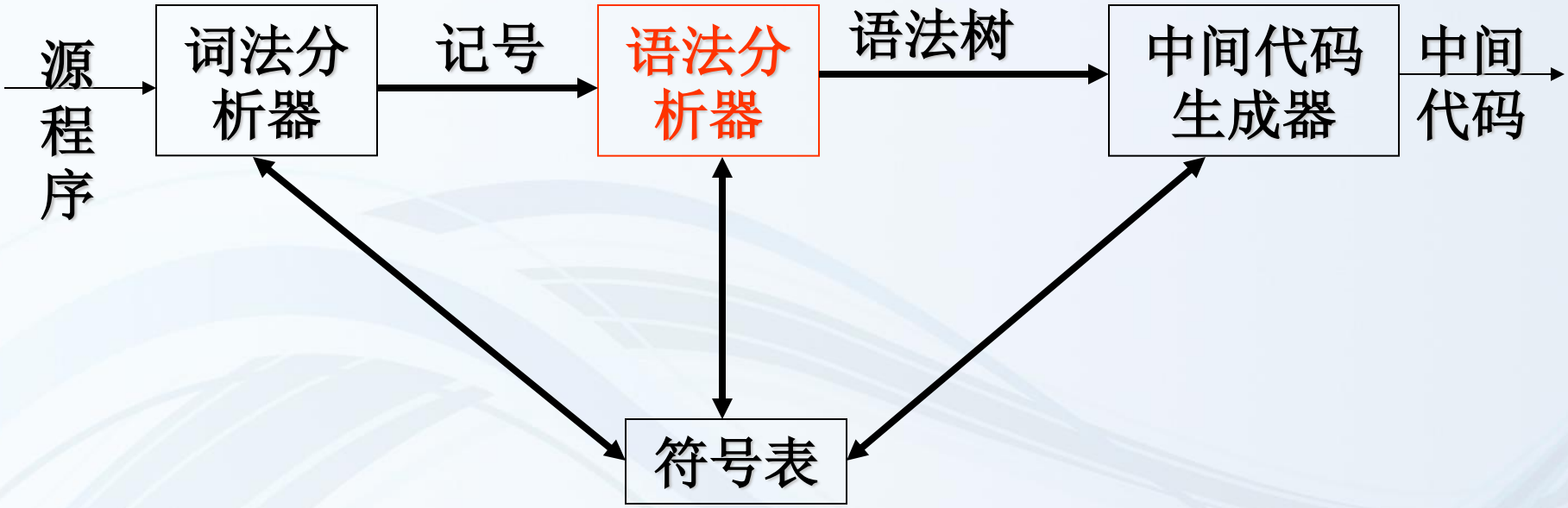
待编译的代码

```
{
    int i , j ;
    float a[100], v,  x;
    while ( true ) {
        do{ i = i + 1 ;  }while ( a[i] < v );
        do{ j = j - 1 ;  }while ( a[j] > v );
        if ( i >= j ) break;
        x=a[i];   a[i]=a[j]; a[j] = x;
    }
}
```

简化的中间代码

```
1:      i=i+1
2:      t1=a[i]
3:      if t1<v goto 1
4:      j=j-1
5:      t2=a[j]
6:      if t2>v goto 4
7:      ifFalse i>=j goto 9
8:      goto 14
9:      x=a[i]
10:     t3=a[j]
11:     a[i]=t3
12:     a[j]=x
13:     goto 1
14:
```

2.1 引言（编译器前端的模型）



2.2 文法的形式化描述

- 一个文法包括若干产生式(production)，产生式也称重写规则(rewriting rule, substitution rule)或规则。

（上下文无关文法）产生式写作：

U ::= u　　或　　U → u

其中：

U是一个文法符号，称为规则的左部或头

u是有穷文法符号串，称为规则的右部或体

- 例：

<sentence> → <subject> <verb>

<sentence> → <subject> <verb> <complement>

文法的形式化描述(II)

- 巴科斯-瑙尔范式(Backus-Naur Form, BNF)

  用尖括号包围非终结符来标记非终结符

  终结符带有下划线（如果很明确，也常省略）

  符号::=表示“推导出”

  符号|表示“还能推导出”

<sentence> ::= <subject> <verb>

| <subject> <verb> <complement>

常常写成

sentence → *subject verb*

| *subject verb complement*

文法例1

<句子> ::= <主语> <谓语> <宾语>

<主语> ::= <代词> | <名词>

<谓语> ::= <动词>

<宾语> ::= <代词> | <名词>

<代词> ::= I | you | he | she | it | we | they

<名词> ::= cat | dog | book | pen | computer

<动词> ::= eat | read | write | use | see

最后一条相当于5条规则

产生式的使用

<句子> ⇒<主语> <谓语> <宾语>

⇒<代词><谓语> <宾语>

⇒I <谓语> <宾语>

⇒I <动词> <宾语>

⇒I use <宾语>

⇒I use <名词>

⇒I use computer

文法例2

<URL> ::= <协议>://<域名>

<协议> ::= http | https | ftp

<域名> ::= <域名段>.<域名段>.<顶级域名>

<域名段> ::= <字母数字串>

<顶级域名> ::= com | cn | org | net | edu

<字母数字串> ::= <字母> | <数字> | <字母> <字母数字串>
| <数字> <字母数字串>

<字母> ::= a | b | ... | z | A | B | ... | Z

<数字> ::= 0 | 1 | ... | 9

推导*Derivation*

❖从开始符出发，不断将某个非终结符替换为该非终结符的某个产生式的右部。

❖推导的符号：⇒

❖语言：从开始符出发，利用推导能得到的所有终结符号串的集合。

推导（例）

$list \Rightarrow list + digit \Rightarrow list - digit + digit$

$\Rightarrow digit - digit + digit \Rightarrow 9 - digit + digit$

$\Rightarrow 9 - 5 + digit \Rightarrow 9 - 5 + 2$

$list \rightarrow list + digit$

$list \rightarrow list - digit$

$list \rightarrow digit$

$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

上下文无关文法Context-Free Grammar

- 也称2型文法，是四元组：
  - 终结符(terminal)集(常记作T或VT)
    - ✓文法所定义语言的基本符号的集合
    - ✓仅仅出现在产生式右部（体）
  - 非终结符(non-terminal)集(常记作N，NT或VN)
    - ✓在某个产生式的左部出现过的文法符号
  - 产生式(production)集
  - 开始符(start symbol)
    - ✓两种说明方式：
      - (1)明确指出, 如G[A]
      - (2)没有明确指出时，第1条产生式规则左部的文法符号

延伸阅读：哈课 2-4 文法的分类

正则文法Regular Grammar

- 也称3型文法，文法的任何产生式均为A→aB或A→a的形式，其中A和B是非终结符，a是终结符或空串
  - 也称右线性文法
  - 3型文法也常定义为左线性文法：文法的任何产生式均为A→Ba或A→a的形式
  - 3型文法是特殊的2型文法

上下文无关文法（例）

$S \to (S)$

$S \to a$

也写作： $S \to (S) \mid a$

终结符集T={(,a,)}，非终结符集N={S}， 开始符是S

本文法描述了语言 $\{(^n a)^n \mid n \geq 0\}$

上下文无关文法生成的语言称为上下文无关语言

串

- 字母表：元素的有穷非空集合，常记作∑。
  - 字母表中的元素称为符号，故字母表又名符号集。
- 串（符号串）：符号构成的有穷序列
  - 空符号串（空串）：不含任何符号的符号串，用ε表示
- 串的运算:
  - 串的长度：组成该串的符号个数，例：x=abca,则$|x|$=4
  - 串的联结：设x和y表示两个串，则xy是它们的联结，表示将串y联结到x之后。
    - ✓ $x\varepsilon=\varepsilon x=x$
    - ✓ x=abca, y=bb，则xy=abcabb
    - ✓串的幂运算：$x^0=\varepsilon$，$x^2=xx$，$x^2y^2=xxyy$

闭包

- 符号串集合的乘积：设A和B为两个符号串集合，则它们的乘积$AB=\{xy|x\in A,y\in B\}$。
  - A={a, ab}, B={b, bb}，则AB={ab, abb, abbb}
  - {ε}A=A{ε}=A
  - 符号串集合的幂运算：A={a,b},则$A^0=\{\varepsilon\}$, $A^2=\{aa,ab,ba,bb\}$
- 符号串集合的闭包：符号串集合的各次方幂之并集
  - 正闭包：不考虑0次幂
  - 串的闭包和正闭包：$a^*=\{\varepsilon, a, aa, aaa, \ldots\}$ $a^+=\{a, aa, aaa, \ldots\}$
  - A={a,b,c}，则A*={ε,a,b,c,aa,ab,ac,ba,bb,bc,ca,cb,cc,aaa,…}
    A^+=?

例子

已知：L = {A, B, C, D }　　M = {1, 2, 3}

$L \cup M$ = {A, B, C, D, 1, 2, 3 }

LM = {A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3}

L^2 = { AA, AB, AC, AD, BA, BB, BC, BD, CA, … DD}

L* = { All possible strings of L plus ε }

$L^+ = L^* - \{\varepsilon\}$

$L (L \cup M) = ??$

$L (L \cup M)^* = ??$

句型和句子

- 推导长度：执行一步推导的次数
  - 一步推导$\Rightarrow$
  - 大于等于1步的推导$\overset{+}{\Rightarrow}$
  - 大于等于0步的推导$\overset{*}{\Rightarrow}$
- 对于上下文无关文法G[S]:
  - 如果$S \overset{*}{\Rightarrow} \alpha$, 则$\alpha$为文法G的句型
  - 如果$S \overset{+}{\Rightarrow} W$ ($W \in T^*$), 则W为文法G的句子。
    - ✓一个文法的句子集合称为语言。
    - ✓文法G对应的语言记为L(G)

例：以下文法会产生多少个不同的句子

(1) A→ BB　　B→CC　　C→1 | 2

(2) A→ BB　　B→CC　　C→1 | 2 | ε

例：以下文法会产生多少个不同的句型

(1) A→ BB　　B→CC　　C→1 | 2

$0+1+1+2*3^2+3^4=101$

(2) A→ BB　　B→CC　　C→1 | 2 | ε

$1+5+4^2+(3^2+3^2+3^3)+3^4=148$

文法和语言

- 文法和语言的关系
  - 给定文法，就唯一地确定语言
  - 给定语言，能给出文法，但文法不唯一
- 等价文法
  - 设G_1和G_2是两个文法，若$L(G_1)=L(G_2)$，则称G_1与G_2为等价文法。

文法和语言(例)

文法确定语言
注意指数的取值范围

(1) 文法A→ aAb | ab对应的语言为：

$L(G[A])=\{a^n b^n | n \geqslant 1\}$

(2) 文法S：S→xSx| xS| y 所识别的语言为：

$L(G[S])=\{x^m y x^n | m \geqslant n \geqslant 0\}$

(3) 文法G：S → xxS | y 所识别的语言为：

$L(G[S])=\{x^{2n} y | n \geqslant 0\}$

也可以写成 $L(G[S])=(xx)^*y$

文法和语言(例2)

多个文法可以产生同一语言

(1) 语言$L=\{ab^na|n\geqslant 1\}$对应的文法可以为：

$A\rightarrow aBa$　　$B\rightarrow b|bB$　或

$A\rightarrow aB$　　$B\rightarrow ba|bB$　或……

(2) 语言$L=\{a^{2n}b|n\geqslant 1\}$对应的文法可以为：

$A\rightarrow Bb$　　$B\rightarrow aa|aaB$　或

$A\rightarrow aab \mid aaA$　　或……

(3) 语言$L=\{a^mb^nc^k|m=n或n=k\}$对应的文法可以为：

$S\rightarrow AB \mid DE$　　$A\rightarrow aAb \mid \varepsilon$　　$B\rightarrow cB \mid \varepsilon$

$D\rightarrow aD \mid \varepsilon$　　$E\rightarrow bEc \mid \varepsilon$

语法分析树

- 以图形方式描述推导过程
- 语法分析树的构成：
  - 根节点是开始符
  - 叶节点是终结符（*token*）和 ε
  - 内部节点（非叶结点）是非终结符
  - 如果应用了规则$A \rightarrow x_1x_2\ldots x_n$, 则A是内部结点；$x_1$，$x_2$，…，$x_n$是子结点

使用分析树来描述推导过程

$list \Rightarrow list + digit$

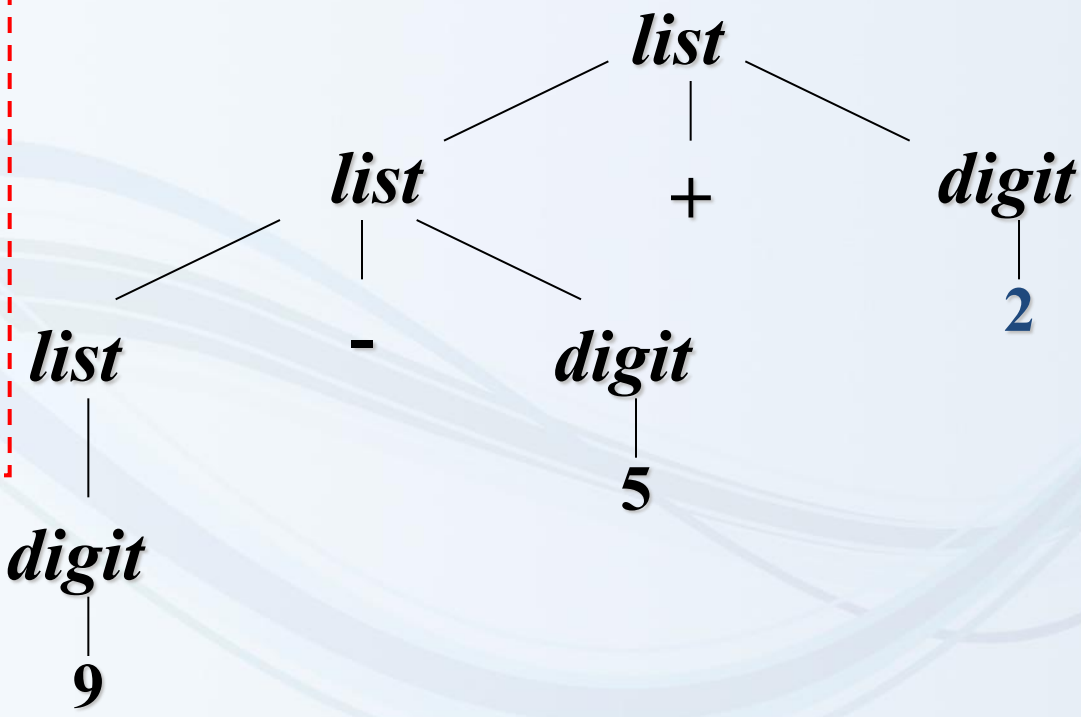
$\Rightarrow list - digit + digit$

$\Rightarrow digit - digit + digit$

$\Rightarrow 9 - digit + digit$

$\Rightarrow 9 - 5 + digit$

$\Rightarrow 9 - 5 + 2$



综合例

考虑文法：S→ SS+ | SS* | a

1）试推导出串aa+a*

$S \Rightarrow SS^* \Rightarrow SS+S^* \Rightarrow aS+S^* \Rightarrow aa+S^* \Rightarrow aa+a^*$

2）试为这个串构造一棵语法分析树

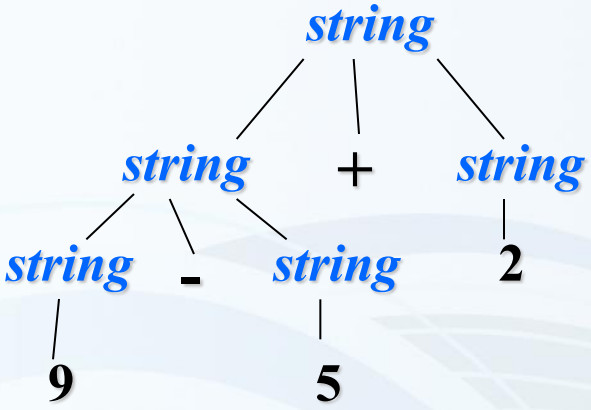
3）该文法生成的语言是什么？

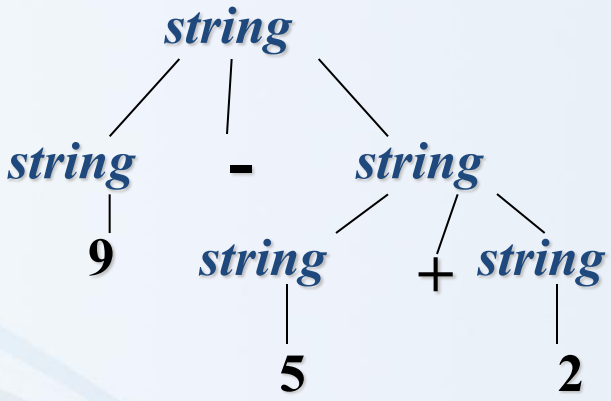
后缀表达式

二义性*Ambiguous*

- 基于一个文法，若有多棵分析树生成同一个终结符号串，则此文法具有二义性。
- 本课程只讨论文法二义，不涉及语义二义。
  - 语义二义例：**Jack said Tom left *his* assignment at home.**

一个二义的文法



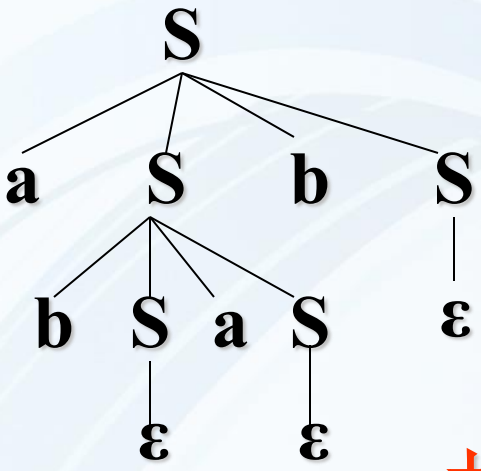


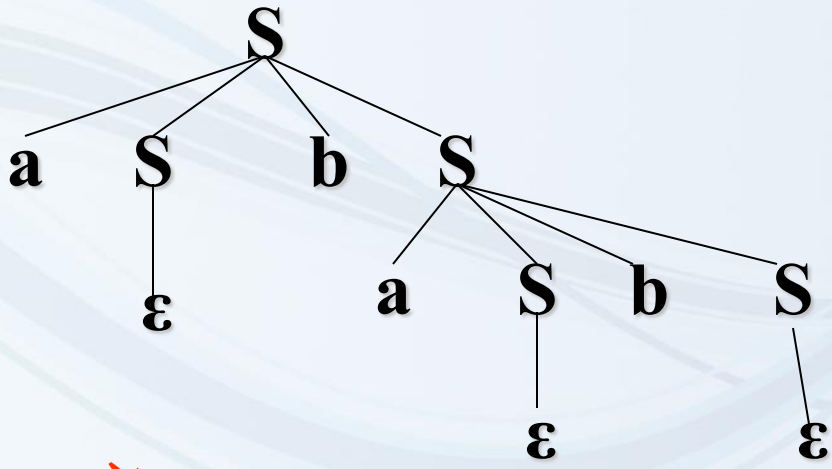
string → *string* + *string* | *string* – *string* | 0 | 1 | …| 9

证明以下文法二义。

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

证：对于串abab存在两棵不同的分析树：





故本文法二义

Which of the following grammars are ambiguous?

(A) $S \rightarrow SS \mid a \mid b$

(B) $E \rightarrow E + E \mid id$

(C) $S \rightarrow Sa \mid Sb$

(D) $E \rightarrow E' \mid E' + E$

$E' \rightarrow -E' \mid id \mid (E)$

可能基于优先级和结合性来构建无二义的文法

- 四则运算存在两个优先级，所以可引入两个非终结符expr和term来对应不同的抽象层次。
  - 抽象层次较低的非终结符对应较高优先级
- 四则运算均为左结合，因此每条规则中，更为抽象的非终结符应位于左边

expr→ expr + term | expr - term | term

term→ term * factor | term / factor | factor

factor→ digit | (expr)

digit → 0 | 1 | 2 |…| 9

最左推导与最右推导

- 最左推导（*Leftmost derivation*）：总是替换当前句型中最左边的非终结符
- 最右推导（*rightmost derivation*）：总是替换当前句型中最右边的非终结符

E	$\longrightarrow$	$E+T$
E	$\longrightarrow$	T
T	$\longrightarrow$	id

Derivations for id + id:

LEFTMOST	RIGHTMOST
$E \Longrightarrow E+T$	$E \Longrightarrow E+T$
$\Longrightarrow T+T$	$\Longrightarrow E+\text{id}$
$\Longrightarrow \text{id}+T$	$\Longrightarrow T+\text{id}$
$\Longrightarrow \text{id}+\text{id}$	$\Longrightarrow \text{id}+\text{id}$

最左推导与最右推导（2）

- 最左（右）推导与分析树一一对应
  - 可以通过说明对于某一个串存在两个不同的最左推导来证明文法二义
  - 可以通过说明对于某一个串存在两个不同的最右推导来证明文法二义

证明以下文法二义：（第二种证法）

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

证：对于串abab存在两个不同的最左推导：

S⇒aSbS⇒abSaSbS⇒abaSbS⇒ababS⇒abab

S⇒aSbS⇒abS⇒abaSbS⇒ababS⇒abab

故本文法二义