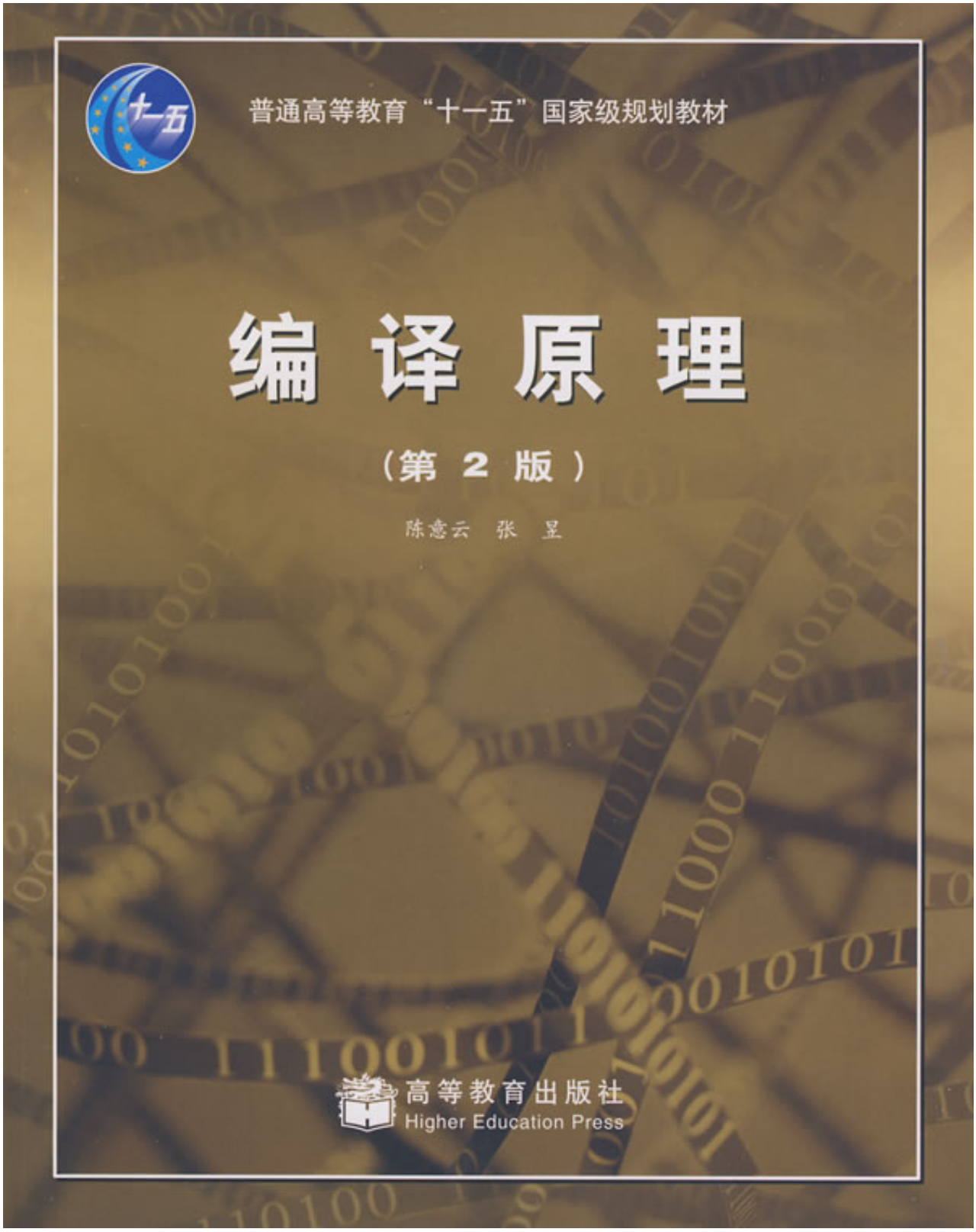
编译原理：知识点整理



第2章：词法分析

词法分析器的任务

把构成源程序的字符流翻译成词法记号流

2.1 词法记号及属性

词法分析的主要任务

扫描输入字符流，产生用于语法分析的词法记号序列

词法分析器能完成的任务

1. 剥去源程序的注解和有空格、制表或换行符等引起的空白
2. 把来自编译器各个阶段的错误信息和源程序联系起来
3. 预处理宏定义

词法记号定义

由记号名和属性值（不必要）构成的二元组，记号名是代表一类词法单元的抽象名字。

模式（规则）定义

描述属于该记号的词法单元的形式。在一个关键字作为一个记号的情况下，它的模式是构成该关键字的字符序列。

词法单元定义

是源程序中匹配一个记号模式的字符序列，它由词法分析器标识为该记号的实例。

标准标识符

预先确定了含义的标识符，但程序可以重新声明它的含义。在该声明的作用域内，程序声明的含义起作用，而预先确定的含义小时，在其他地方则都是预先确定的含义起作用。

词法记号的属性

记号名影响语法分析的决策，属性影响记号的翻译。

词法错误

发现错误	错误恢复
几乎发现不了源程序的错误，因为词法分析器对源程序采取非常局部的观点	紧急恢复（最简单的）：删掉输入指针当前指向的若干个字符（剩余输入的前缀），直到词法分析器能发现一个正确的记号为止
	错误修补尝试：删除一个多余的字符、插入一个遗漏的字符、用一个正确的字符代替一个不正确的字符、交换两个相邻的字符

2.2 词法记号的描述与识别

正规式

正规式是表示模式的规则的一种重要方法。

字母表、串、长度、语言

概念	描述
字母表	表示符号的有限集合
串	某一字母表符号的有穷序列
串的长度	出现在该串中符号的个数
语言	表示字母表上的一个串集，属于该语言的串称为该语言的句子或字

语言运算

运算	描述	符号
和	两个集合的并集	∪
连接	两个集合的笛卡尔积	无符号连接
闭包	表示另个或多个集合连接的并集	*
正闭包	表示一个或多个集合连接的并集	+

正规式的定义

按照一组定义规则，由较简单地正规式构成的，每个正规式表示一个语言***L(r)***。这些定义规则说明***L(r)***是怎样从***r***的子正规式所表示的语言中构造出来的。

正规集的定义

正规式表示的语言叫做正规集。

正规集的运算约定

运算	优先级	结合性
*	最高	左结合
连接	次之	左结合
|	再次之	左结合

正规式的代数定律

概念	描述
r	s = s	r	| 可交换
r	(s | t) = (r | s)	t	| 可结合
(rs) t = r (st)	连接可结合
r (s | t) = rs	rt; (s | t) r = sr	tr	连接对 | 可分配
εr = r; rε = r	ε连接恒等元素
r* = (r | ε)*	ε肯定出现在一个闭包中

C语言正规定义

```
letter_->A|B|...|Z|a|b|c|...|z|_
digit->0|1|...|9
id->letter_(letter_|digit)*
```

无符号数正规定义

无符号数是由整数部分、小数部分、指数部分三部分组成，其中小数部分和指数部分都是可能出现或可能不出现的。指数部分如果出现，是E及可能有的+或-号，再跟上一个或多个数字。小数点后必然有一个数字。

```
digit->0|1|...|9
digits->digit digit*
optional_fraction->.digits|ε
optional_exponent->(E(+|-|ε)digits)|ε
number->digits optional_fraction optional_exponent
```

正规式缩写

缩写	符号	描述
一个或多个实例	+	和*拥有同样的结合性和优先级。
零个或一个实例	?	r|ε
字符组	[abc]	a|b|c
字符组	[a-z]	a|b|c|...|z

状态转换图定义

状态转换图描绘语法分析器为得到下一个记号而调用词法分析器时，词法分析器所做的动作。

正规式	记号名	属性值
ws	---	---
while	while	---
do	do	---
id	id	符号表条目的指针
number	number	数表条目的指针
relop	relop	LT、LE、EQ、NE、GT或GE

转换图属性

图形	描述
圆圈	状态
箭头	边
标记	连接可结合
两个圆圈	接受状态
*	输入指针回退

installID( )

1. 看关键字表
2. 如果当前词法单元构成关键字，则返回相应的记号，结束；否则跳到3
3. 该词法单元是标识符，看标识符表
4. 如果在标识符表发现发现该词法单元则返回相应的条目指针；否则该词法填入

installNum( )

把词法单元置入数表，并返回建立的条目的指针

2.3 有限自动机

语言的识别器

它是一个程序，它取串x作为输入，当x是语言的句子时，它回答“是”，否则回答“不是”。可以通过构造称为有限自动机的更一般的转换图，把正规式翻译成识别器。

有限自动机分类

类型	描述
不确定的	存在这样的状态，对于某个输入符号，它存在不止一种转换
确定的	所有状态输入某个符号都只存在一种转换

不确定的有限自动机模型

内容	表示
有限的状态集合	*S*
输入符号的集合	∑
转换函数	*move*
惟一的开始状态	*s0*
接收状态集合	*F*

转换表

每个状态一行，每个输入符号和ε（如果存在的话）各占一行，表的第i行中符号a的条目是一个状态集合（说的更实际一些，是状态集合的指针），这是NFA在输入是a时，状态i所能到达的状态集合。

优点	缺点
快速访问给定状态和字符的状态集	当输入字母表较大并且大多数转换是空集时，浪费空间

由NFA定义的语言时它接受的输入串集合。

确定的有限自动机模型

是不确定有限自动机的特殊情况。确定的有限自动机从任何状态出发，对于任何输入符号，最多只有一个转换。

特殊内容	说明
任何状态都没有ε转换	任何状态都必须进行输入符号的匹配才能进入下一个状态
对任何状态s和任何输入符号a，最多只有一条标记为a的边离开	转换函数move：S ×∑->S 可以是一个部分函数

有限转换表	有限转换图
表中的每个栏目最多只有一个状态。	从开始状态起，最多只有一条到达某个终态的路径可以由这个串标记。

模拟DFA

```
s = s0;
c = nextChar();
while(c!=eof)
{
  if(move(s,c)未定义)
    return "no";
  else s = move(s,c);
  c= nextChar();
}
if(s 属于 F)
  return "yes";
else return "no";
```

子集构造法定义

从NFA构造识别同样语言的DFA所用到的算法。

子集构造法思想

在NFA的转换表中，每个条目是一个状态集；在DFA的转换表中，每个条目只有一个状态。如果让新构造的DFA的每个状态代表该NFA的一个状态集，这个DFA在输入

$$a_1a_2...a_n$$

后到达的状态对应该NFA从开始状态沿着那些标有

$$a_1a_2...a_n$$

路径能到达的所有状态集合。

```
算法：从NFA到DFA的子集构造法
输入 一个NFA N
输出 一个接受同样语言的DFA D
方法 为D构造转换表Dtran，表中的每个状态是N的状态集合，D"并行"的模拟N面对输入串的所有可能的移动
```

DFA化简重要结论

重要结论：每一个正规集都可以由一个状态数最少的DFA识别，这个DFA是惟一的。

DFA化简基于转换函数是全函数

如果一个DFA的转换函数不是全函数，可以引入一个死状态s_d，s_d对所有输入符号都转换到s_d本身。

区别状态的方法

对于DFA M，如果从状态s出发，在面临输入w时，最终停在一个接收状态，而从状态t出发，面临w时，它停留在一个非接收状态，或者反过来，则串w可以用来区别状态。

极小化DFA状态数算法

把状态分成一些不想交的子集，每一子集的状态都是不可区别的，不同子集的状态都是可区别的，每个子集合并成一个状态。

$$start$$

$$1.划分包含两个子集：接受状态子集and非接受状态子集;$$

$$2.检查每一个子集，看其中的状态是否还可区别。$$

$$3.重复1、2过程，对当前划分进一步细分，直到没有任何一个子集再需细分为止$$

$$end$$

使用这个算法时，其输入DFA的状态转换函数必须是全函数，否则有可能得到的新DFA和原来的接收不同的语言。所以我们引入死状态，其目的就是把转换函数变成全函数。

2.4 从正规式到有限自动机

从正规式构造NFA的一个算法

该算法是语法制导的，它用正规式语法结构来指导构造过程。

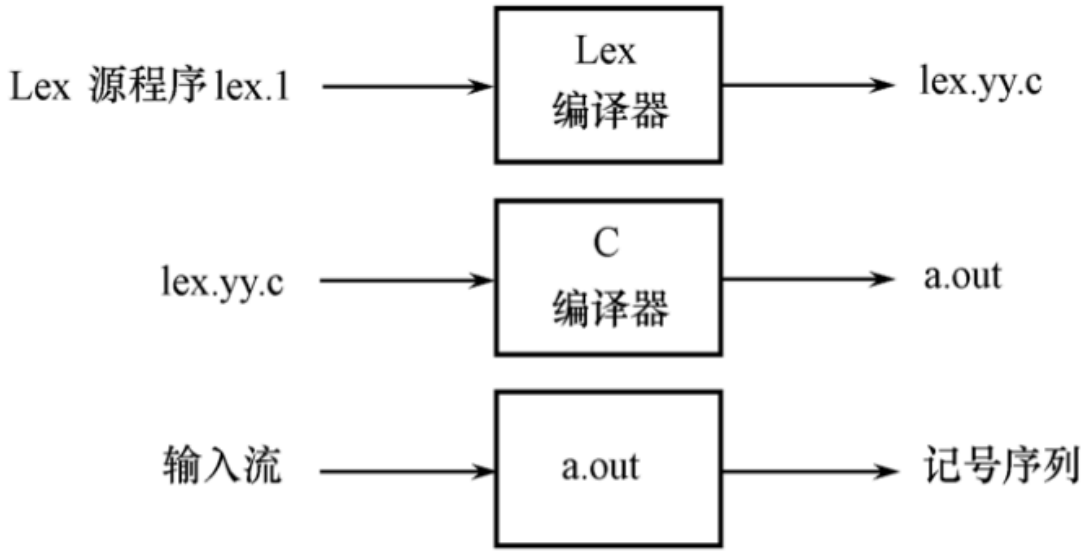
1. 构造识别字母ε和字母表中的一个字符的自动机
2. 构造识别主算符为选择、连接或闭包的正规式的自动机

在构造过程中，每部最多引入两个新的状态。

2.5 词法分析器的生成器

Lex

从基于正规式的描述来构造词法分析器。



1. 词法分析器的说明使用Lex语言表达在程序lex.1中，然后lex.1通过Lex编译器，产生C语言程序lex.yy.c
2. lex.yy.c被编译成目标程序a.out，即是把输入串识别成记号序列的词法分析器

Lex程序的三个部分

部分	说明
声明	包括变量声明、常量定义和正规定义
翻译	翻译规则的形式模式 {动作}；每个模式是一个正规式。每个动作描述该模式匹配词法单元时，词法分析器执行的程序段
规则	包括了动作所用到的附加函数

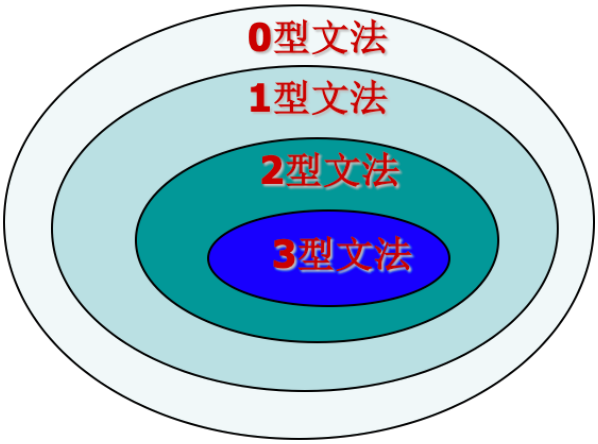
词法分析器的应用

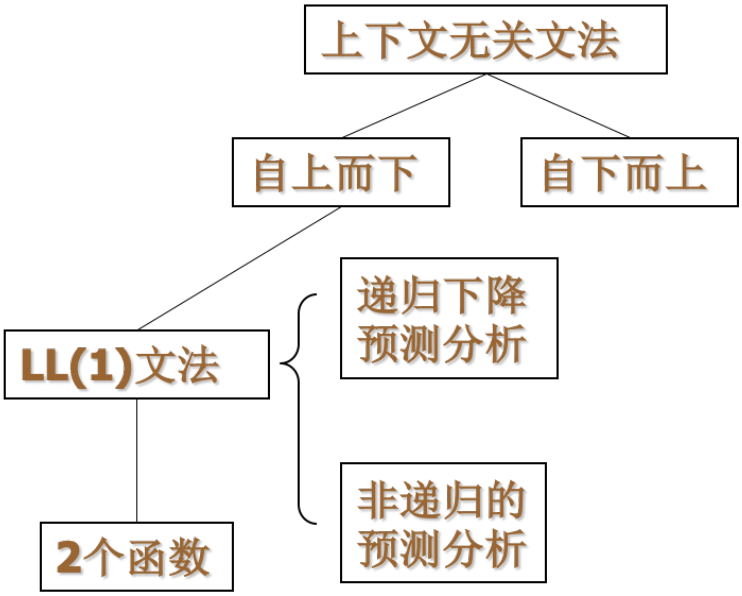
由Lex建立的词法分析器通常作为语法分析器的一个子程序。

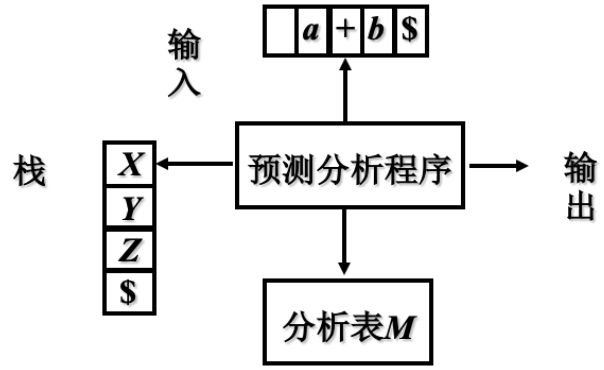
1. 被语法分析器激活
2. 逐个字符的读它的生育输入，直到它在剩余输入中发现能和某个模式P匹配的最长前缀为止。
3. 执行和P对应的动作A
4. 如P描述空白或注释，词法分析器继续寻找后记得词法单元，直到有一个动作引起控制回到语法分析器为止
5. 词法分析器返回一个值（记号名）给语法分析器，记号的属性值通过共享变量yylval传递

第3章：语法分析

总览







语法分析方法

都是从左向右的扫描输入，每次一个符号

方法	说明
自上而下	分析器从根节点到叶节点的次序来建立分析树
自下而上	分析器从叶节点到根节点的次序来建立分析树

最有效的自上而下和自下而上的分析法都只能处理上下文无关的子类。

3.1 上下文无关文法

语法分析器的位置

1. 分析器读取词法分析器提供的记号流，检查它是否能由源语言的文法产生，输出分析树的某种表示。
2. 分析器能以以理解的方式报告语法错误，博那个从发现语法错误的状态中恢复过来，使对剩余程序的分析能继续进行
3. 前端的其余部分：各种记号的信息收入符号表，完成类型检查和其他语义检查，产生中间代码。

正规式的局限

正规式只能表示给定结构的固定次数的重复或者没有指定次数的重复。

上下文无关文法的形式定义

上下文无关文法G是一个四元组

$$(V_r, V_N, S, P)$$

符号	说明
Vr	非空有限集合，其元素称为终结符
VN	非空有限集合，其元素称为非终结符，Vr和VN没有交集
S	非终结符，开始符号，它定义的终结符串集就是文法定义的语言
P	产生式的有限集合，产生式指明了终结符和非终结符组成串的方式（类似于模式）

终结符

类别	实例
字母表前面的小写字母	a,b,c
黑体串	**id** or **while**
数字	0，1，2，……，9
标点符号	分号、逗号
运算符号	+、-

非终结符

类别	实例
字母表前面的大写字母	A,B,C
字母S	通常表示开始符号
小写字母名字	expr or stmt

文法符号

字母表后面的大写字母，如X，Y，Z

终结字符串

字母表后面的小写字母，如u，v，...,z

文法的符号串

小写希腊字母

选择

以A为左部的产生式

推导

把产生式看成重写规则，把符号串中的非终结符用其产生式右部的串来代替。

$$E => E + E$$

产生式

$$E- > (E)$$

表示E的任何出现都可以用文法符号串（E)来代替。

符号“=>”表示“一步推导，符号=>*表示”零步或多步推导。于是，

1. α =>* α对于任何串成立
2. 如果α=>*β*， *β*=>*γ*,*那么α*=>γ

符号=>+表示“一步或多步推导”

等价

如果两个文法能产生相同的语言，那么称这两个文法等价。

句型

如果

$$S =>^* \alpha$$

α可能是非终结符，则把α叫做G的句型。

句子

句子是只含终结符的句型。

最左推导

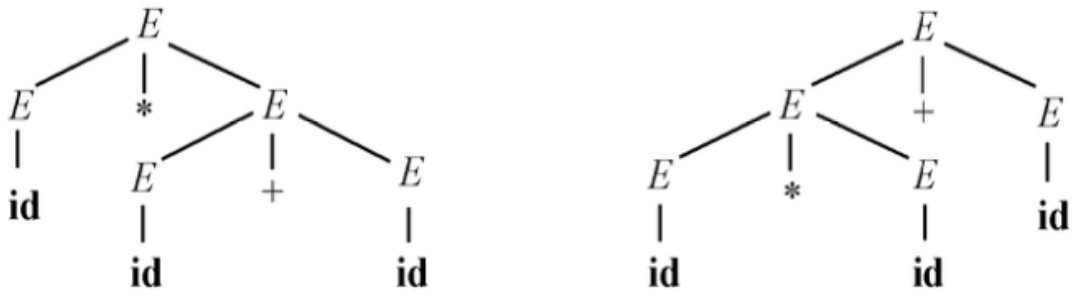
为了理解一些分析器时怎样工作的，需要考虑每一步都是代换句型中最左边非终结符的推导，这样的推导叫做最左推导。

类似的，每步都代换最右侧非终结符的推导称为最右推导，又叫规范推导。

分析树

分析树是推导的图形表示。

数据结构	说明
内部节点	非终结符标记
子节点	非终结符本次推导所用产生式的右部各符号从左到右标记
叶节点	非终结符或终结符标记，所有这些标记从左到右构成一个句型



每棵分析树都有它对应的最左推导和最右推导。

上图中，右边的分析树反映了+和*通常的有限关系，而左边的分析树却不是。这就产生了二义性。

二义性

有些文法的一些句子存在不止一棵分析树，或者说这些句子存在不止一种最左（最右）推导。

文法的二义性

一个文法如果存在某个句子不止一个最右（最左）推导，即某个句子不止由一颗分析树与之对应的话，就称这个文法是二义的。

文法的二义性并不代表语言一定是二义的。只有当产生一个语言的所有文法都是二义的，这个语言才称为二义的。

3.2 语法和文法

文法的优点

1. 文法为语言给出了精确地，易于理解的语法说明
2. 对于某些问法雷，可以自动产生搞笑的分析器
3. 分析器的自动构造过程可以揭示出文法的二义性和难以分析的构造，而这些问题在语言及编译器的最初设计阶段可能没有发现。
4. 如果文法设计的得当，则它赋予语言的结构对于把源程序翻译成正确的目标代码和对于错误诊断都是有用的
5. 语言也是逐渐完善的，增加新构造以完成新任务的情况时有发生

正规式和上下文无关文法

正规式可以描述的语言都能用上下文无关文法描述。

方法	说明
编译器的效率会改进	正规式是描述注入标识符、常数和关键字等词法结构的最有力武器。
上下文无关文法	描述括号配对、begin end配对、语句嵌套、表达式嵌套等结构的最有力武器

NFA和上下文无关文法

可以机械的把一个NFA变换成一个上下文无关文法，它产生的语言和这个自动机识别的语言相同。步骤如下：

1. 确定终结符号集合
2. 为NFA的每个状态引入非终结符
3. 如果状态i有一个a转换到状态j，则引入对应的产生式

$$A_i -> aA_j$$

4. 如果是ε则引入

$$A_i -> A_j$$

5. 如果i是接受状态，再引入

$$A_i -> \varepsilon$$

为什么用正规式定义语言的词法

1. 语言的词法规则非常简单，不必用功能更强的上下文无关文法描述
2. 对于词法记号，正规式给出的描述比上下文无关文法给出的更简洁和易于理解
3. 从正规式构造出E的词法分析器比从上下文无关文法构造出的更有效

词法分析器和语法分析器的分离原因

原因	说明
编译器的效率会改进	词法分析的分离可以简化词法分析器的设计，允许构造专门和更有效的词法分析器
编译器的可移植性加强	输入字符集的特殊性和其他与设备有关的不规则性可以限制在词法分析器中处理
便于编译器前端模块划分	非终结符或终结符标记，所有这些标记从左到右构成一个句型

适当的表达式文法

表达式文法有二义性，一个句子的不同分析树体现不同的算符优先级和算符结核性。

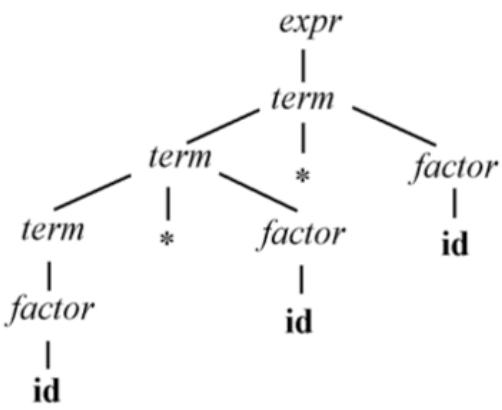
非终结符	说明
expr	开始符号，表示不同层次的表达式和子表达式
term	表示不同层次的表达式和子表达式
factor	产生表达式的基本单位

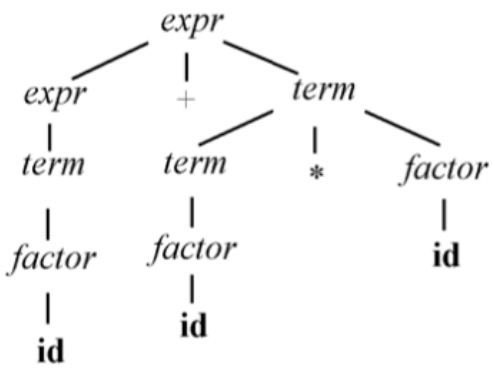
$$factor \rightarrow id|(expr)$$

$$term \rightarrow term * factor|factor$$

$$expr \rightarrow expr + term|term$$

*是一个拥有较高优先级的左结合算符。





二义性实例

悬空else的二义性：

```
stmt → if expr then stmt
      |if expr then stmt else stmt
      |other
```

```
其中，
        if expr then if expr then stmt else stmt
的嵌套条件语句有两个最左推导：
stmt=>if expr then stmt=> if expr then if expr then stmt else stmt
stmt=>if expr then stmt else stmt=> if expr then if expr then stmt else stmt
// 每个else和左边最接近的还没有配对的then相配对
修改为以下文法：
stmt->matched_stmt
      |unmatched_stmt
matched_stmt->if expr then matched_stmt else matched_stmt
      |other
unmatched_stmt->if expr then stmt
      |if expr then matched_stmt else unmatched_stmt
```

定义语言语法的文法有二义性并不可怕，只要消除二义性的规则就行了

左递归

如果一个文法，它有非终结符A，对某个串α，存在推导：

$$A =>^{+} A\alpha$$

则它是左递归的。

自上而下的分析方法不能用于左递归文法，因此要消除左递归。

直接左递归

$$A \to A\alpha$$

由上面形式的产生式引起的左递归称为直接左递归。

消除左递归实例

A->Aα|β可以由下面来代替

$$A \to \beta A'$$

$$A' \to \alpha A'|\varepsilon$$

消除左递归的技术

不管有多少A产生式，都可以用下面的技术来消除直接左递归。

1. A产生式组合在一起

$$A->A\alpha_1|A\alpha_2|...|\beta_1|\beta_2|...|\beta_n$$

2. 用

$$A->\beta_1 A'|\beta_2 A'|...|\beta_n A'$$

$$A'->\alpha_1 A'|\alpha_2 A'|...|\alpha_m A'|\varepsilon$$

代替A产生式。这些产生式和前面的产生式产生一样的串集，但不再有左递归。

这个过程可以删除直接左递归，但不能消除两部或多部推导形成的左递归。

形式语言鸟瞰

文法分类

0、1、2、3, 其差别仅在于对产生式施加不同的限制

描述能力：0>1>2>3

分类	说明
0	短语文法，递归可枚举
1	上下文有关文法，对非终结符的替换需要考虑上下文，并一般不允许ε串
2	上下文无关文法，对非终结符的替换不需要考虑上下文
3	正规文法，等价于正规式

3.3 自上而下分析

宗旨

对任何输入串，使用一切可能的方法，从文法开始符号，自上而下，从左到右的为输入串建立分析树。

前提

使用自上而下分析法时，文法应该没有左递归。

消除回溯

为了构造不带回溯的自上而下分析算法，首先要消除文法的左递归，并找出克服回溯的充分必要条件

如果对文法的任何非终结符，当要用它去匹配输入串时，能够根据所面临的输入符号准确的指派它的一个选择去执行人额度，那么回溯肯定能消除。

准确的含义

若此选择匹配成功，那么这种匹配绝不是虚假的；若次选择无法完成匹配任务，则任何其他的选择也肯定无法完成。

预测分析

能根据当前的输入符号为非终结符确定用那一个选择。

递归下降的预测分析

为每一个非终结符写一个分析过程。由于文法的定义实地贵的，这些过程也是递归的。

在处理输入串时，首先执行的是对应开始符号的过程，然后根据产生式又不出现的非终结符，依次调用相应的过程，这种逐步下降的过程调用序列隐含的定义了输入的分析树。

```
type→simple
      |id
      |array[simple] of type
simple->integer
        |char
        |num dotdot num

void match(terminal t)
{
  if(lookahead == t) lookahead = nextToken();
  else error();
}

void type()
{
  if((lookahead == integer)|| (lookahead == char)||(lookahead == num)) simple();
  else if(lookahead == '*'){match('*'); match(id);}
  else if(lookahead == array){match(array);match("
[");simple();match(']');match(of);type();}
  else error();
}
```

非递归的预测分析

如果显示的维持一个栈，而不是隐式的通过递归调用，那么可以构造非递归的预测法分析器。

表驱动的预测分析器

分类	包含
输入缓冲区	被分析的串，$作为输入串的结束标记
栈	存放文法的符号串，栈底符号$
分析表	二维数组M[A,a], A是非终结符，a是终结符或$
输出流	

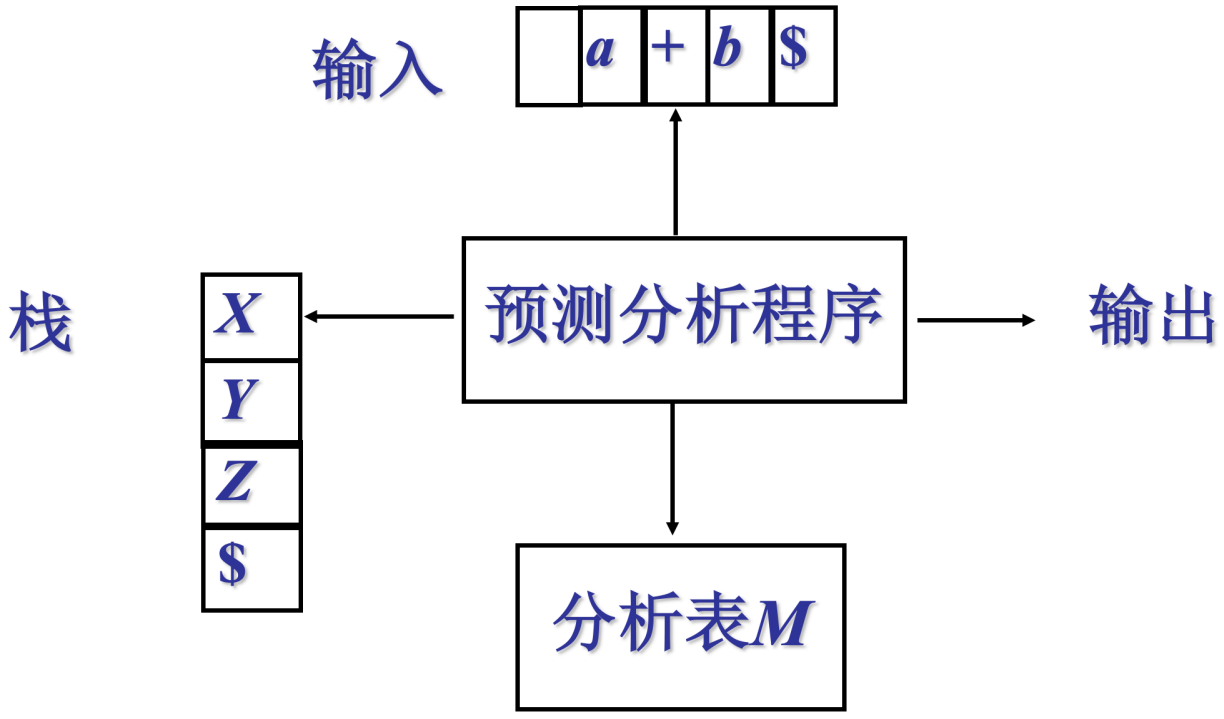
预测分析器工作过程

预测分析器根据当前的栈顶符号X和输入符号a决定分析器的动作

X	a	动作
$	$	分析完全成功，停机
！$	X	弹出栈顶符号X，推进输入值真，使之指向下一个符号
终结符！a	~	发现语法错误，调用错误恢复例程
非终结符	~	访问M，若M[X,a]是X的产生式，打印所用的产生式；若M指示出错，分析器调用错误恢复例程

非递归预测分析算法

过程	说明
输入	串w和文法G的分析表M
输出	如果w属于L（G），则输出w的最左推导，否则报告错误
方法	初始时分析器的格局是：$S在栈里，其中S是开始符号并且在栈顶；w$在输入缓冲区中



程序：

```
让ip指向w$的第一个符号;
令X等于栈顶符号;
while(X!=$)//栈非空
```

```
{
  if(X == a) 把X从栈顶弹出并推进ip;
  else if(M[X,a]是出错入口) error();
  else if(M[X,a] = X->Y1Y2...Yk)
  {
    输出产生式X->Y1Y2...Yk;
    从栈中弹出X;
    把Yk,Yk-1,..,Y1,依次压入栈，Y1在栈顶;
  }
  令X等于栈顶符号;
}
```

构造预测分析表

可以证明，一个文法的预测分析表没有多重定义的条目，当且仅当该文法是LL(1)的。

预测分析的错误恢复

层次	说明
词法错误	标识符、关键字的拼写错，遗漏字符串两端的引号
语法错误	算术表达式的括号不配对，case语句没有外围的Switch语句
语义错误	算符和运算对象之间类型不匹配。在java方法中，return语句的类型是void类型
逻辑错误	C程序中用”=“代替比较算符”==“

分析器对错误处理的基本目标

1. 清楚而准确的报告错误的出现
2. 迅速的从每个错误中恢复过来，以便诊断剩余程序的错误
3. 它不应该是正确程序的处理速度降低太多

非递归分析在什么场合下发现错误

1. 栈顶的终结符和下一个输入符号不匹配
2. 栈顶是非终结符A，输入符号是a，而M[A , a]是空白

采用紧急方式的错误恢复

发现错误时，分析器每次抛弃一个输入记号，直到输入记号属于某个指定的同步记号集合为止。

同步

词法分析器当前提供的记号流能构成的语法结构，正是语法分析器所期望的。

同步记号集合的选择

1. 把FOLLOW(A)的所有终结符放入非终结符A的同步记号集合。

```
if  expr   then
    （then是expr的一个同步记号）
```

2. 把高层结构的开始符号加到低层结构的同步记号集合中。

   ```
   a := expr  ;  if …
   （语句的开始符号作为表达式的同步符号，以免遗漏分号时忽略一大段程序。）
   ```

3. 把FIRST(A)的终结符加入A的同步记号集合。
4. 如果非终结符可以产生空串，若出错时栈顶是这样的非终结符，则可以使用产生空串的产生式。
   如果终结符在栈顶而不能匹配，弹出此终结符。

出错的一般处理

1：栈顶不动；2、3：指针不动

1. 查表，当前表项空白，指向记号流的指针后移；
2. 查表，当前表项中含有同步记号synch，将当前栈中的非终结符弹出栈；
3. 栈顶终结符和当前指针指向的终结符不匹配，将栈顶终结符弹出栈。

3.4 自上而下分析

编译器常用的递进-规约分析方法叫做LR分析。