语法分析

一、语法分析分为自顶向下和自底向上

二、首先分析自顶向下

①在自顶向下中可能会遇到二义性问题、左递归引起的无限推导回溯问题

二义性问题：多个可选产生式，无法确定用哪个产生式进行推导

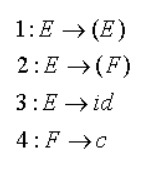
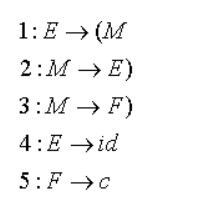
回溯问题：**如果对于非终结符E有多个候选式存在公共前缀，则自顶向下的语法分析程序便不能准确地选择产生式进行推导，只能进行试探，出现错误要回溯到上一个分支点，再选择其它的产生式进行尝试。**

**这些问题的部分解决方法：**

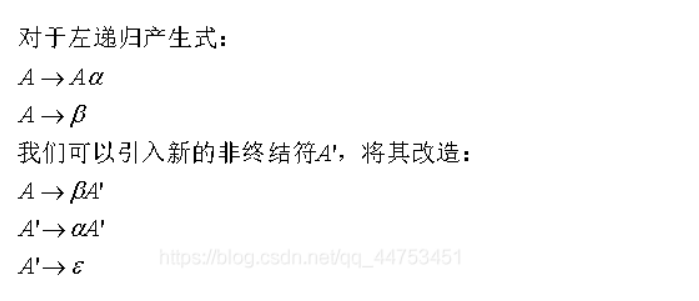
**1.提取左公因子**

**引入一个新的非终结符，这个非终结符定义为公共部分之后的部分，然后原来的非终结符定义为公共部分加上这个新的非终结符**

**例子：**

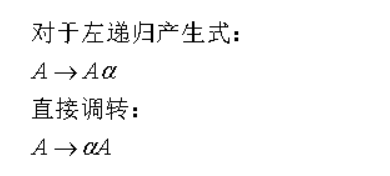
————————————>

2.消除左递归

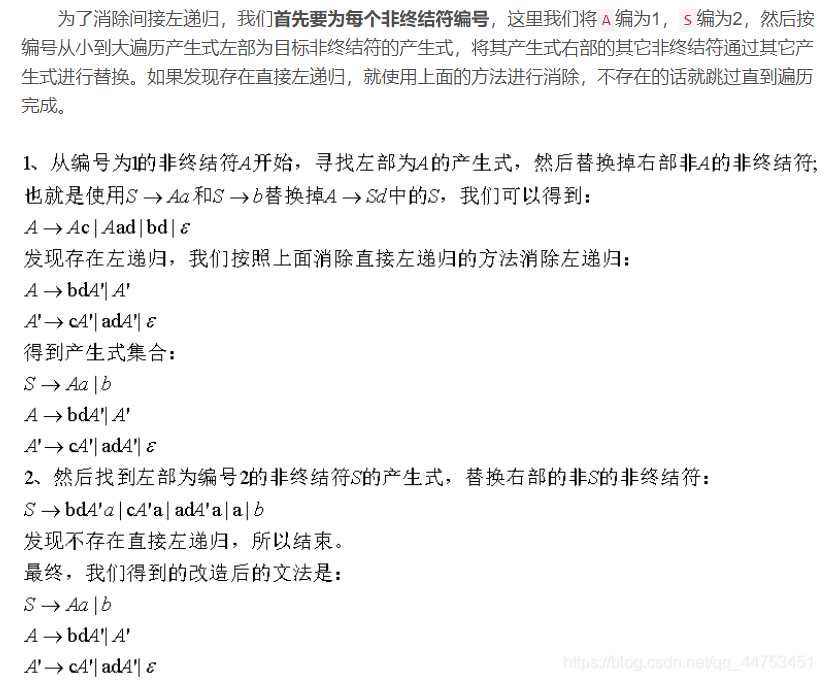


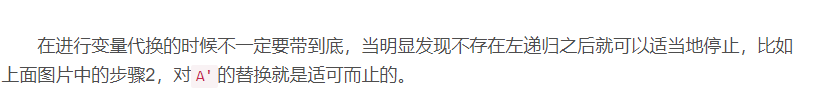
引入一个新的非终结符，然后是原来的到原来的能不左递归的加新的，新的到左递归之后的额加新的，新的到空

如果其中的β为空（）则直接转换：



若存在间接左递归：

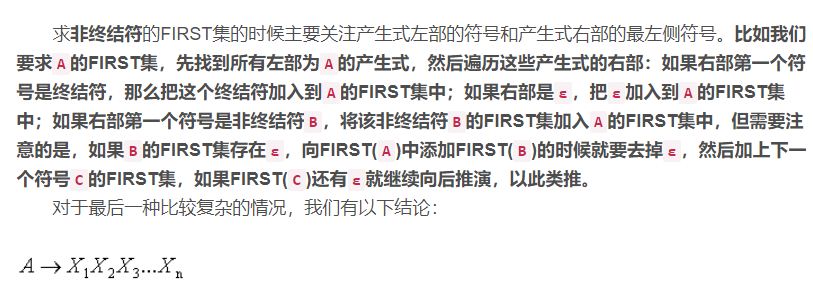




② LL(1)文法：（因为存在回溯问题，所以又叫不确定的自顶向下分析）

LL(1)文法的要求：无二义性，无左递归，任意一个非终结符的各个候选式所能推导出的第一个终结符必须各不相同

于是需要FIRST集（其实就是某个串能推导出的第一个终结符的集合）



另外，终结符的FIRST集是它本身

以及FOLLOW集：（某个非终结符之后可能出现的第一个终结符）



第三个条件：非终结符号A的所有候选式的FIRST集必须两两不相交

如上，可以判断是否是LL(1)文法（其实就是做了改写文法的操作）

然后，下面的步骤

首先，对每一个非终结符号A，做如下操作：

创建一个初始状态和一个终结状态（按0,1,2,顺序开始标号），然后对每一个产生式创建一条从初态到终态的路径。

然后对状态转换图进行化简（利用带入的方法）

然后也即可以实现程序，在程序之中，如果需要别的非终结符参与，使用procX（）代替

在使用proc的东西之后不用移动指针（因为在proc里面会进行移动），如果是终结符，那么要进行判断之后的pointer++

以上是递归调用预测分析。

非递归预测分析中：

通过一张分析表和一个栈联合控制

预测分析表的构造：

先构造每个非终结符的FIRST集和FOLLOW集

然后开始遍历每一个产生式

在每一个产生式A->k中，先看右边的每个终结符号n，如果终结符号在k的FIRST集中，则表中的（A,n）空格填入A->k这个产生式

如果k的FIRST集合中有空，则对于A的FOLLOW集中的元素x，向空格（A,x）中填入这个产生式A->k

如果一个预测分析表中不含多重定义的项，则这个文法是LL(1)文法

构造完预测分析表，接下来看分析过程

第一列是栈，第二列是输入，第三列是输出

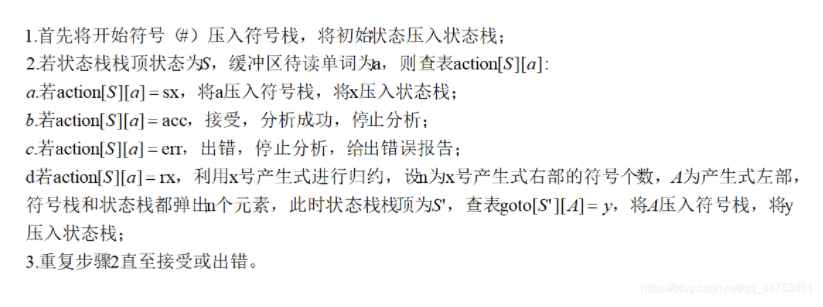
栈这一列起始状态是$和起始符号，然后看栈的最后一个符号和输入的第一个符号，对应分析表里的项，然后把栈的符号退栈，将对应格子的产生式倒序入栈，输出里就是写这个格子的产生式，一直到栈顶符号和现在正在分析的符号一致，这时候这两个直接消掉，没有输出，直到最后两列都只剩下一个$，则分析完毕，接受。

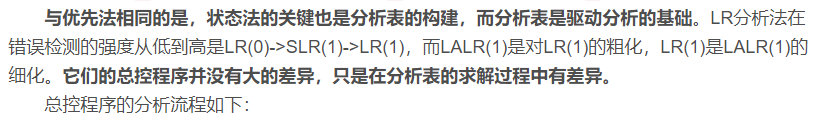
三、自底向上分析方法（移进-归约分析法）

“移进-归约方法”的规范归约过程：每移动一次看当前串中是否有句柄，有的话就进行归约，直到左边是$S，右边是$

**LR分析方法：**

这种方法不需要消除左递归，L表示从左至右扫描符号串，R表示为输入符号串构造一个最右推导的逆过程，k表示为了作出分析而向前看的输入符号的个数





SLR(1)分析表的构造：

首先进行拓广文法，新加入一个起始符号S’，然后S’->S

构造识别文法所有活前缀的DFA：

从S’->·S开始，先构造这个的闭包，这就是I0

然后看第一个的闭包里的点后边有多少符号，就有多少个下一层转移。下一层转移的第一句就是对应转移的前一个闭包里的那一句的点向后移动一个，然后再根据这新产生的这一句进行新的闭包的构造，循环下去，直到状态不再增多（In不再增多）

在构造中可能存在移进-归约冲突和归约-归约冲突

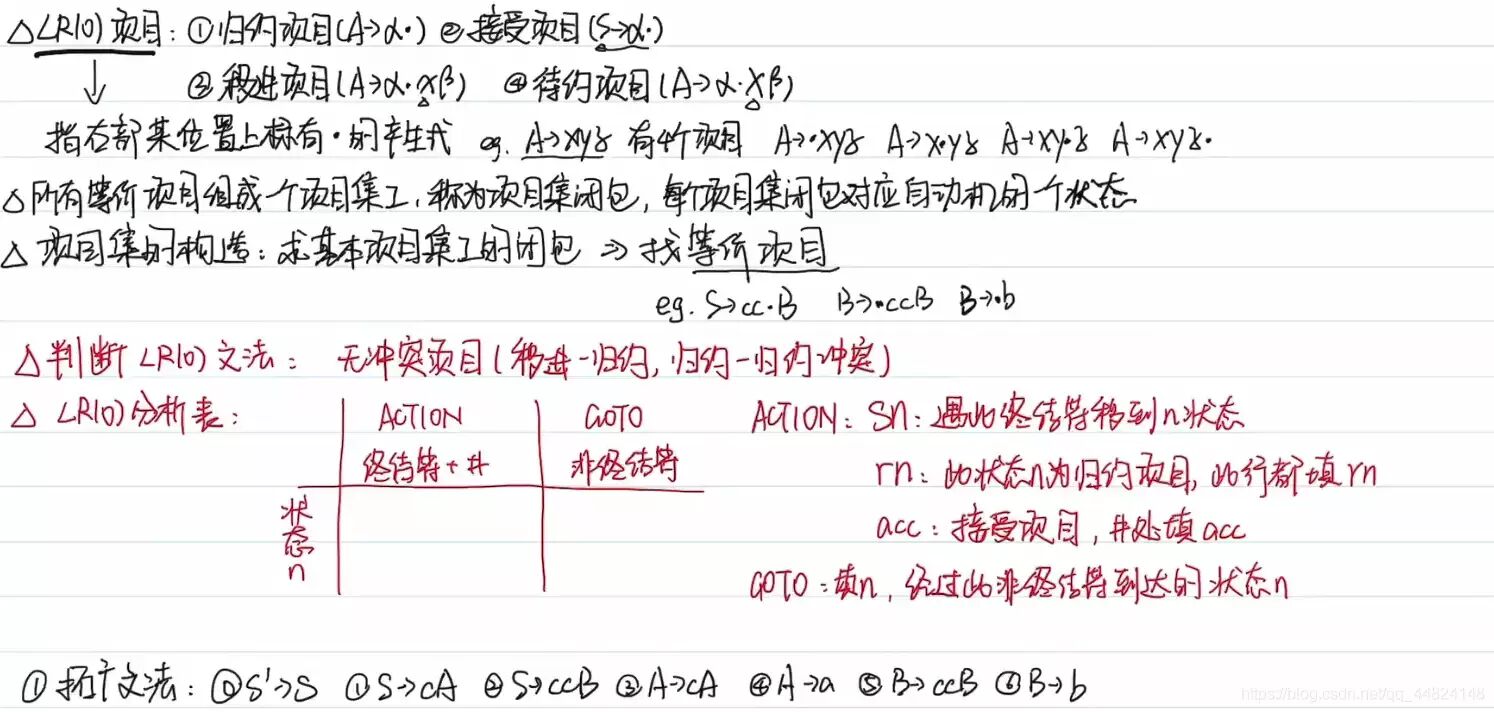
解决冲突要看移进项目的FOLLOW集合，看看交集是否为空，如果为空那么可以作出决策，不为空的话其实依旧解决不了

构造SLR(1)分析表，这个分析表包含一个goto和一个action，action是根据当前状态和非终结符来的，goto是根据当前状态和终结符来的

比如，在状态3下，有个c，过c之后状态是8，则action（3，c）= 8，但是这个要写成S8。非终结符则是goto（3，X） = k，这个直接写数字。如果一个状态是归约项目，则对这个归约项目左部分的FOLLOW集中的所有符号，在action（左部符号，FOLLOW集中符号）填入Rn，其中n是产生式的编号，即用哪个式子进行归约。若是接受项目，则写acc

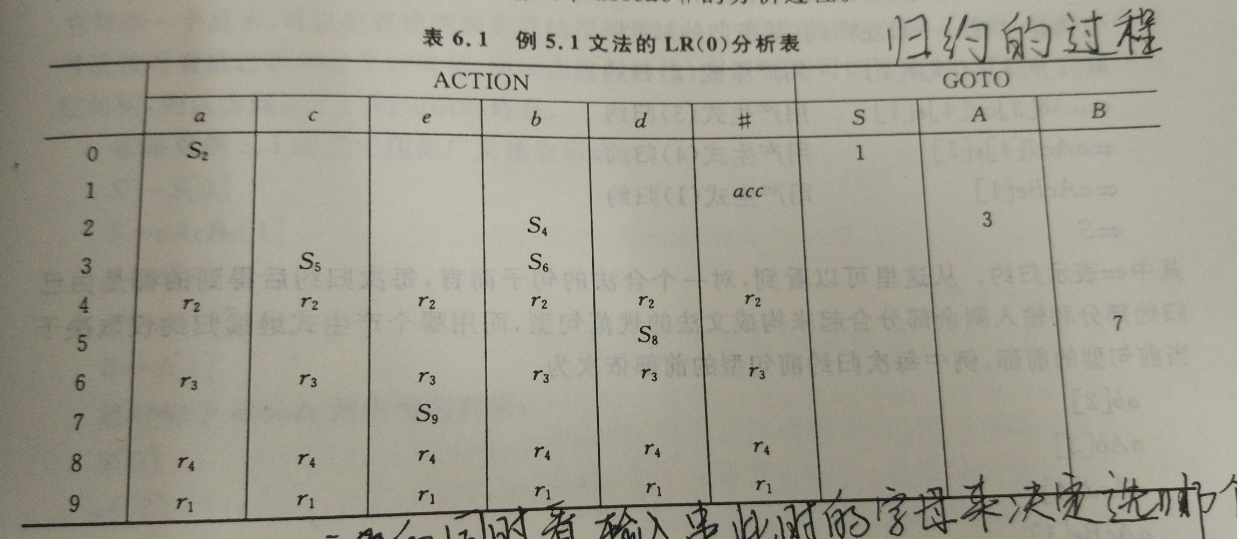
如果在执行上述方法的过程中，始终没有向前看任何输入符号

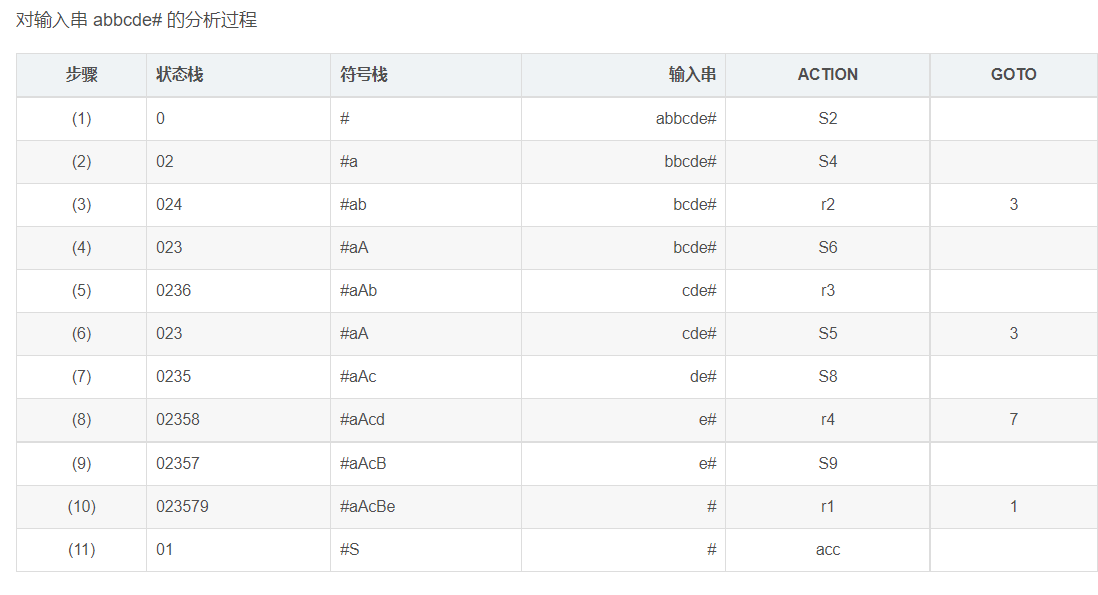
要么所有元素都是移进-待约项目，要么只含有唯一的归约项目



\*\*\*\*\*\*\*\*\*\*LR(0)文法中好像归约的那一行都是归约（？）\*\*\*\*\*\*\*\*

对于分析过程：

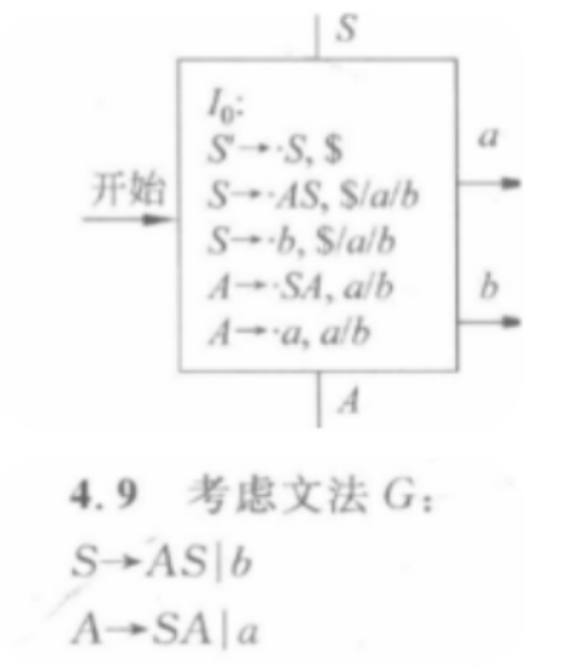




状态栈，符号栈，输入串先确定，然后action根据当前状态栈栈顶和输入串第一个确定，然后将输入串第一个移进符号栈，如果是归约，那么先把状态栈栈顶符号退掉，把符号栈栈顶进行归约，然后看状态栈栈顶和符号栈栈顶来确定goto（为什么这么看呢，因为其实在进行归约之后，相当于是现在的状态栈栈顶（退掉之后的）经过了一个终结符，所以说用他俩看goto），此时就确定好了这次的归约的状态，在下一行再开始写goto完之后的（注意上一步未进行移进，故输入符号串暂未改变）

LR(1)分析表的构造：

LR(1)中的向前看符号，看·之后的第二个符号能推出来的第一个终结符，若是空的则直接把前一个式子的先前看符号挪下来，但是有一种特殊情况，需要多加考虑，例如下图：



从上至下看的时候S应该只有$才对，a和b是哪来的呢，其实是下面推出来的A的产生式里出来的，意思就是A的式子里的圆点后边是S，可以看成这个式子还能推出S，也就是说S还能被A这个推出来，所以说S的向前看符号就可以加上这个圆点后面的第二个符号，也就是A的向前看符号

分析表的构造：

S和goto的构造和上面差不多，在归约项目上，如果一个状态里有归约项目，那么看这个项目的向前看符号，并在这个空格之下填归约，其实是做了细化，，然后分析过程差不多

LALR(1)分析表的构造：

合并LR(1)项目集规范族中的同心集

同心集的合并，不会产生新的移进归约冲突，但可能会出现新的归约-归约冲突

在合并同心集时，把向前看符号合并，分配给每个项目，I6和I7合并之后记为I67，分析表与LR(1)分析表构造方法类似，分析方法类似。

四、LR(1)>LALR(1)>SLR(1)>LR(0)

不论是lr还是ll都是数越大分析的越精确，这个也形成一种包含关系

LR（0）