编译原理复习 #3 Top-Down Parsing

**Scanning vs. Parsing**

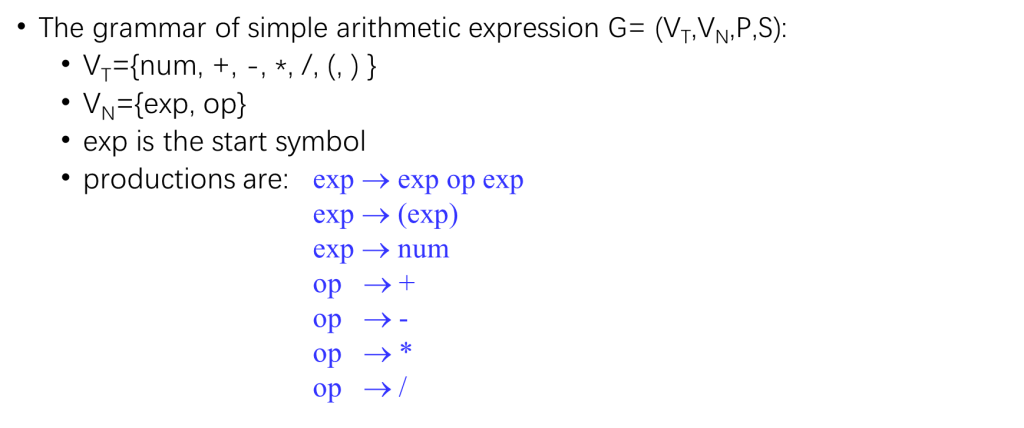
|  | **Scanning** | **Parsing** |
| --- | --- | --- |
| Task | determining the structure of tokens | determining the syntax or structure of a program |
| Describing Tools | regular expression | context-free grammar |
| Algorithmic Method | represent by DFA | top-down parsing bottom-up parsing |
| Result Data Structure | linear structure | parser tree or syntax tree, they are recursive |

**上下文无关文法**

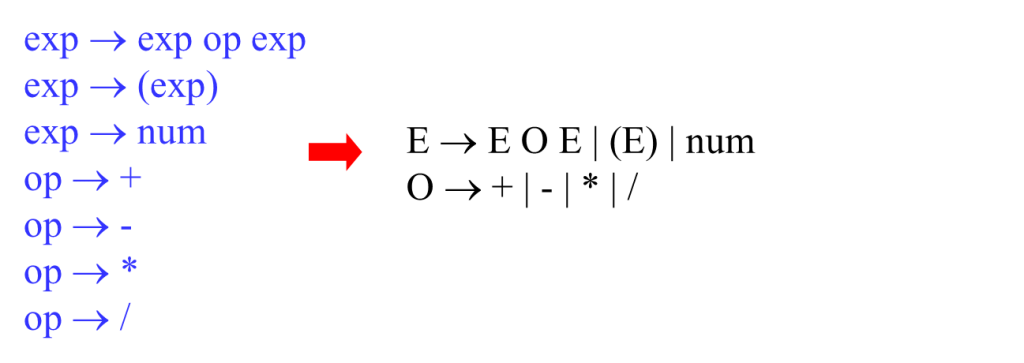
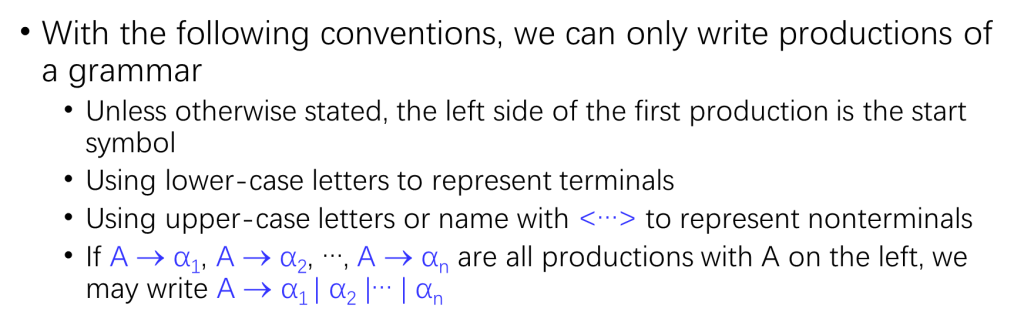
上下文无关文法（简称文法）由**终结符号、非终结符号、一个开始符号和一组产生式组成**

* 终结符号是组成串的基本符号。术语“词法单元名字”是“终结符号”的同义词
* 非终结符号是表示串的集合的语法变量
* 一个文法中，某个非终结符号被指定为开始符号。这个符号表示的串集合就是这个文法生成的语言。按照惯例首先列出开始符号的产生式
* 一个文法的产生式描述了将终结符号和非终结符号组合成串的方法

**例子**



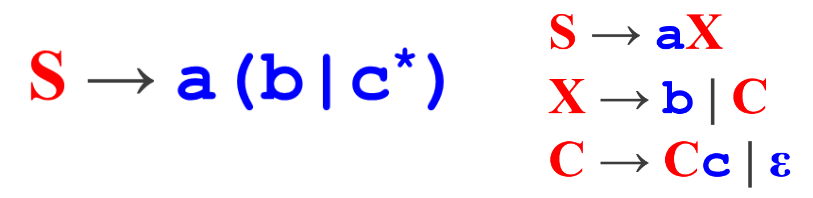
**补充性写法规范**



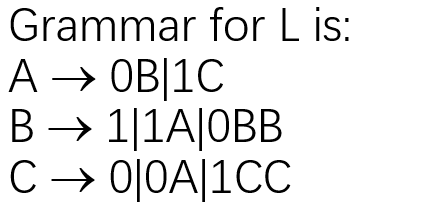
**R.E. vs. CFG\***

* The rules of context-free grammar are **recursive**
* The power of representation of context-free grammar is increased significantly over that of regular expression

**改写正则表达式**



**A language consists of 0 and 1, every string of the language has the same number of 0 and 1**



**树和推导（Derivation）**

将一个非终结符号替换为其某一个产生式右边的动作称为**推导**



* 文法的**句型：**S 开始符号在文法推导路上的**任意一步**
* 文法的**句子：只能包含非终结符**
* 文法生成的**语言：**其所有**句子的集合**



**语法分析树**

*最左推导对应于先序遍历语法树，最右推导对应后序遍历语法树的倒转（指访问到非终结符的顺序对应于推导变换终结符的顺序）*



**二义性**

一个文法可以为某个句子生成多棵语法分析树那就是**二义性（ambiguous）**，亦或者是对同一个句子有多个最左推导或多个最右推导的文法**（二义性和你使用哪种推导是没有关系的）**

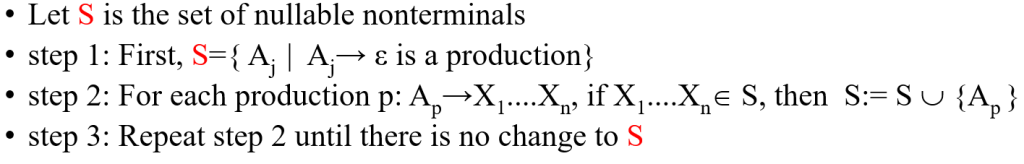
**树的查找：Summary of Leftmost BFS/DFS**

* **Leftmost BFS**works on  all grammars
* Worst-case runtime is exponential
* Worst-case memory usage is exponential
* Rarely used in practice
* **Leftmost DFS** works on grammars without left recursion
* Worst-case runtime is exponential
* Worst-case memory usage is linear
* Often used in a limited form as **recursive descent**

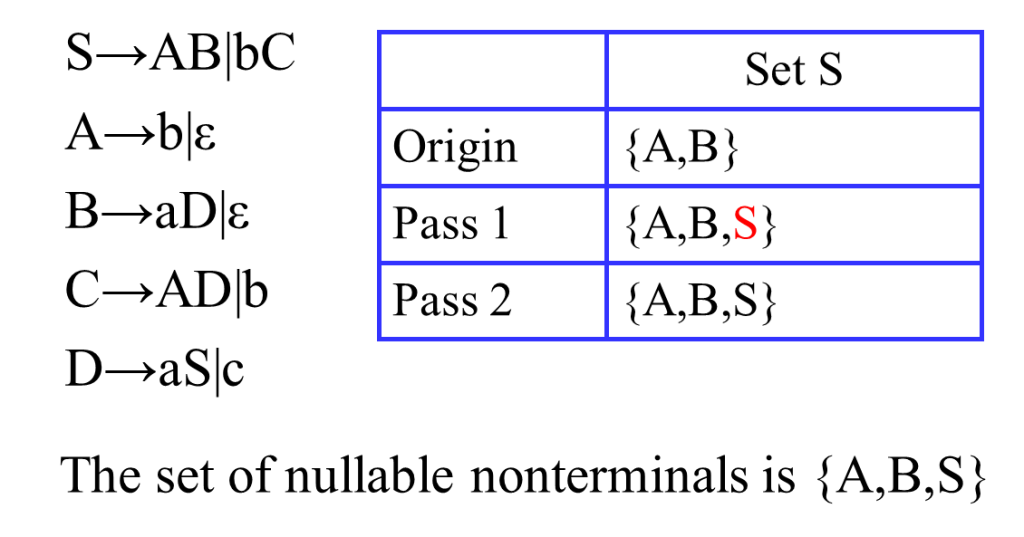
**LL(1) 文法**

**Compute the set of nullable nontermianls**

**可以推导为空串**的就是 **nullable nontermianls**

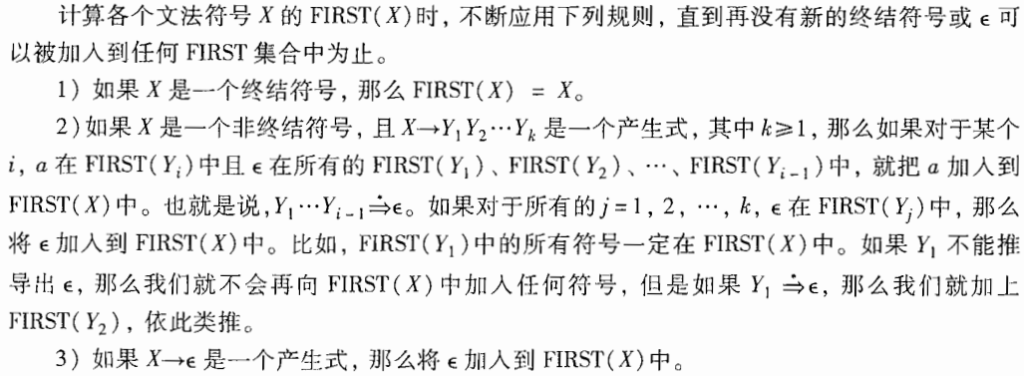


**例子**

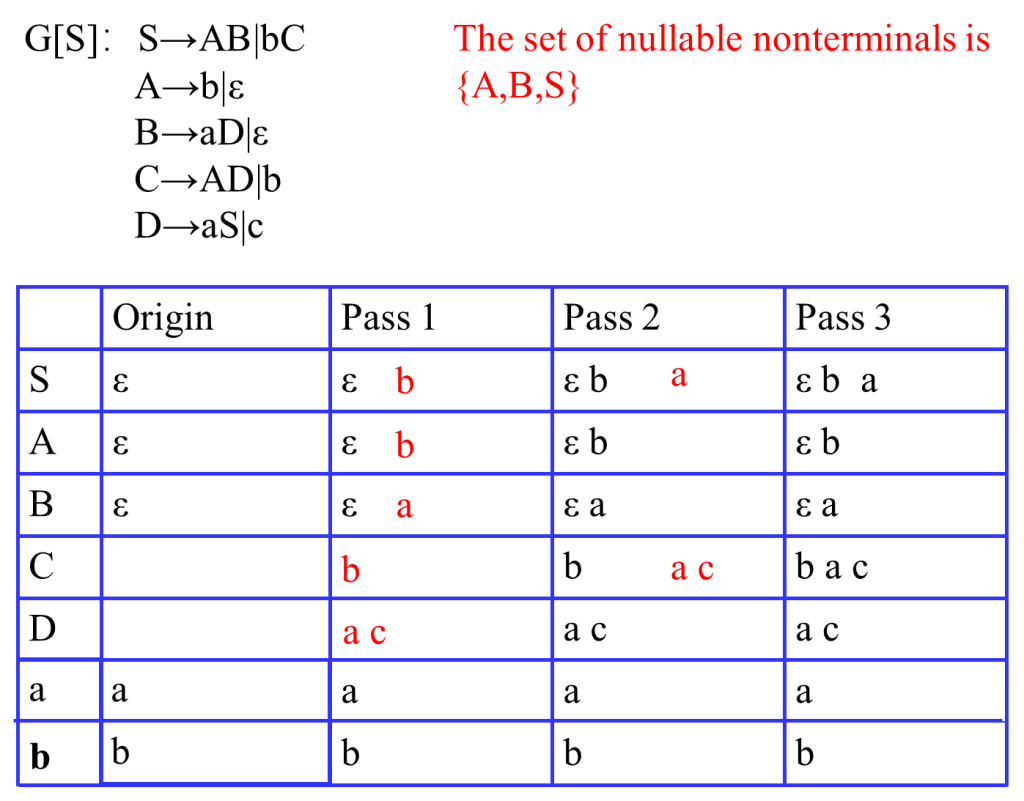


**First & Follow**

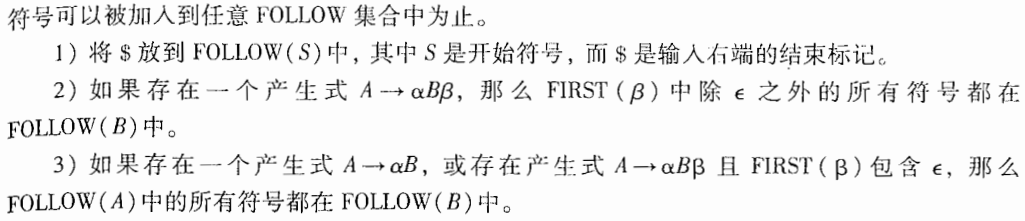
**FIRST 集合**被定义为可从文法符号串推导得到的串的**首符号**的**集合**



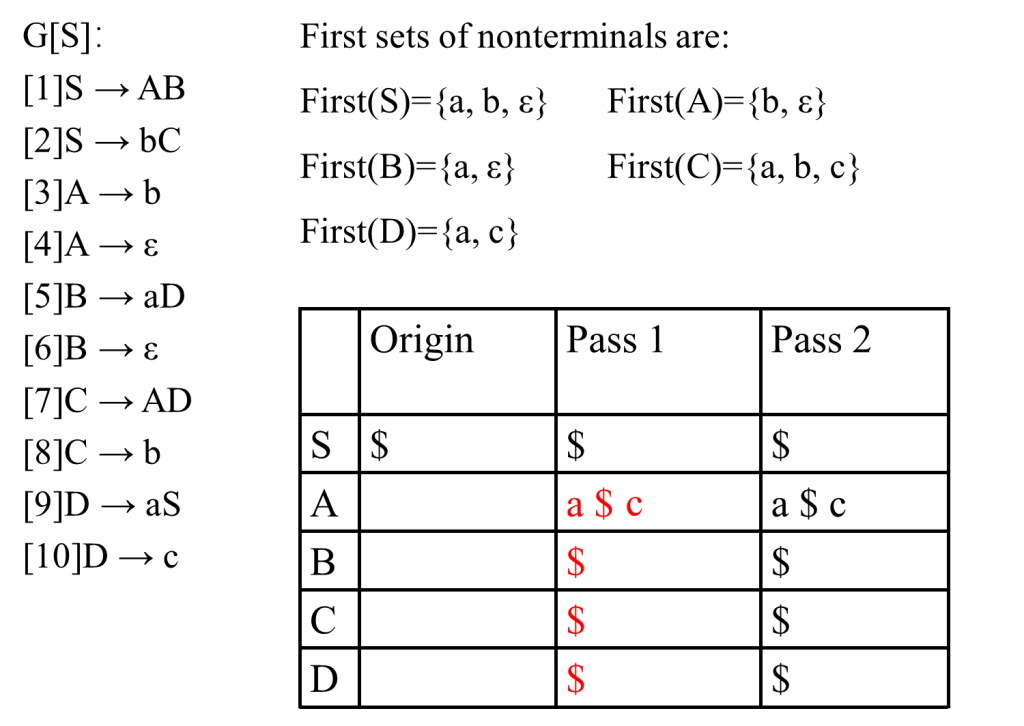
**例子**



**FOLLOW 集合**被定义为可能在某些句型中**紧跟在该符号右边的终结符号的集合**

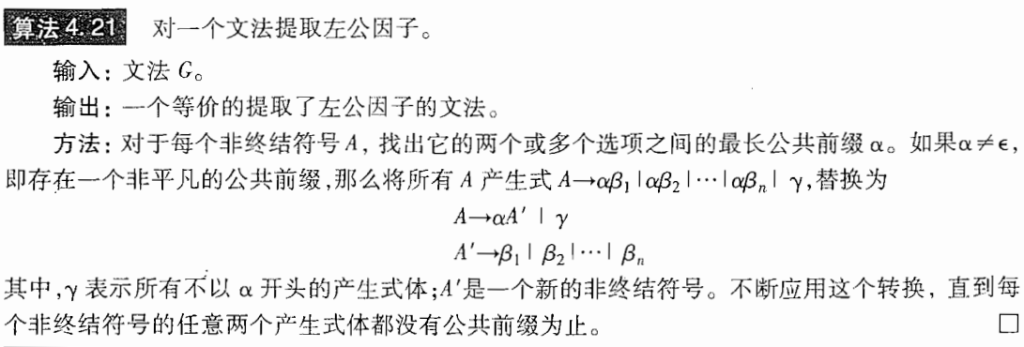


**例子**



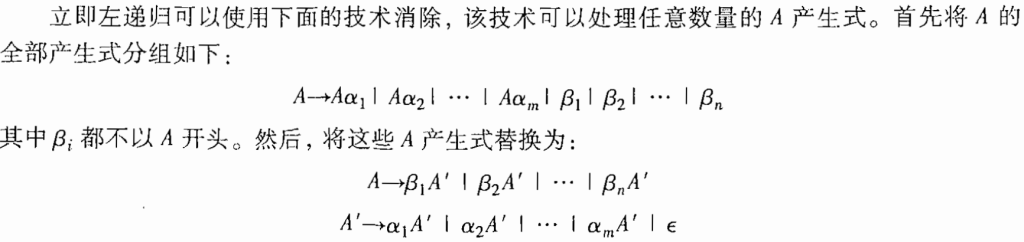
* 求 Nullable 是竖着走的，其他两个是横着排的
* **First 集合不**会出现**结束符可能**会出现**空符**，而 **Follow 集合不**会出现**空符一定会**出现**结束符**
* **First Origin 放空符，而 Follow Origin 放结束符**

**提取左公因子**



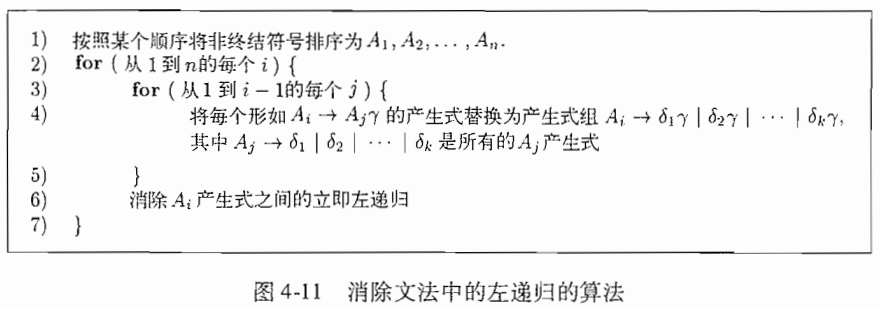
**左递归的消除**

**立即左递归的消除**



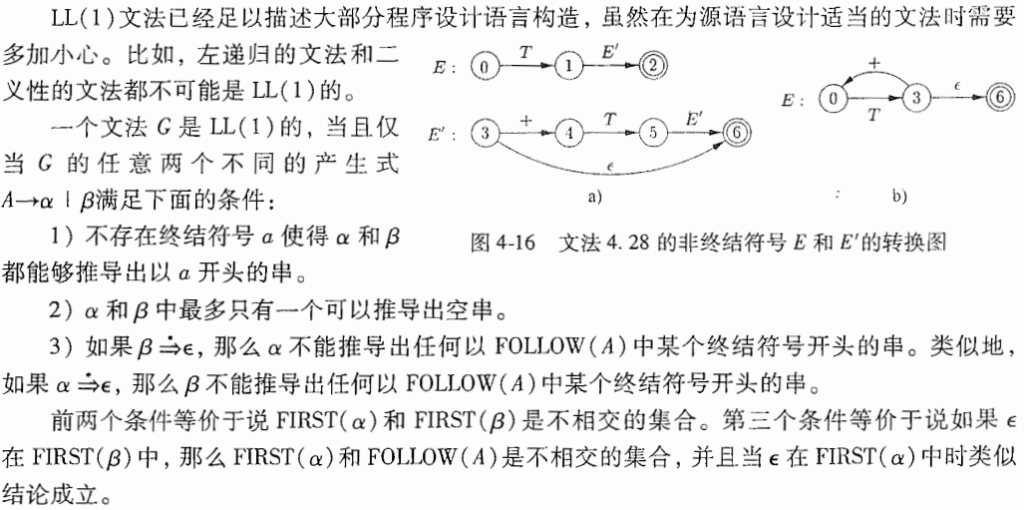
*A’ 一定放在后面，而且 A’ 产生空串，A 产生式的前面就是没有立即左递归的部分，A’ 产生式前面是 A 立即左递归的后面*

**消除全部左递归**



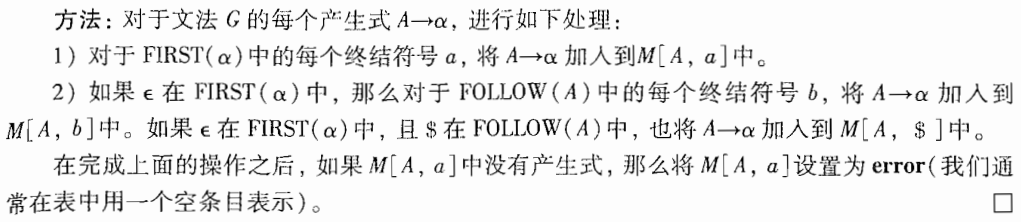
将产生式按顺序排开，从**第一个开始依次往后**，把**当前的产生式右侧第一个符号是前面产生式左侧的进行替换（后面的后侧被替换为前面的前侧）**，然后**消除立即左递归**

**LL(1) 文法判断**



*有左递归和左因子的都不是 LL(1) 文法*

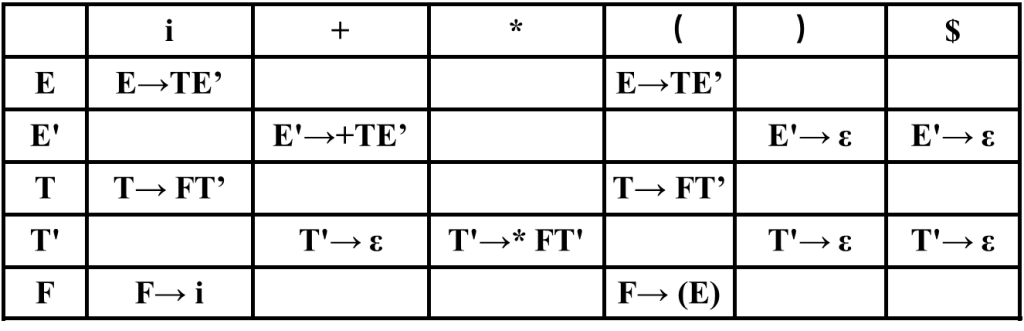
**预测分析表（parsing table）**



*就是想判断当前句型下收到的终结符号是 A 如何产生的（First & Follow）*

**示例：通过下面的表格求出 Parsing Table**

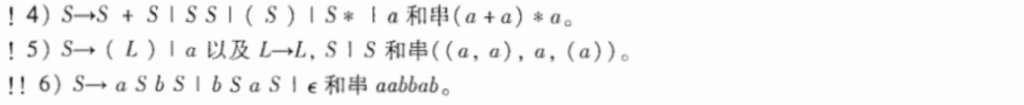
|  |  |  |
| --- | --- | --- |
| **Production** | **FirstSet** | **FollowSet** |
| **E→TE’** | {(,i} |  |
| **E’→+TE’** | {+} |  |
| **E’→ε** | {ε} | {),$} |
| **T→FT’** | {(,i} |  |
| **T’→\*FT’** | {\*} |  |
| **T’→ε** | {ε} | {+,),$} |
| **F→(E)** | {(} |  |
| **F→i** | {i} |  |



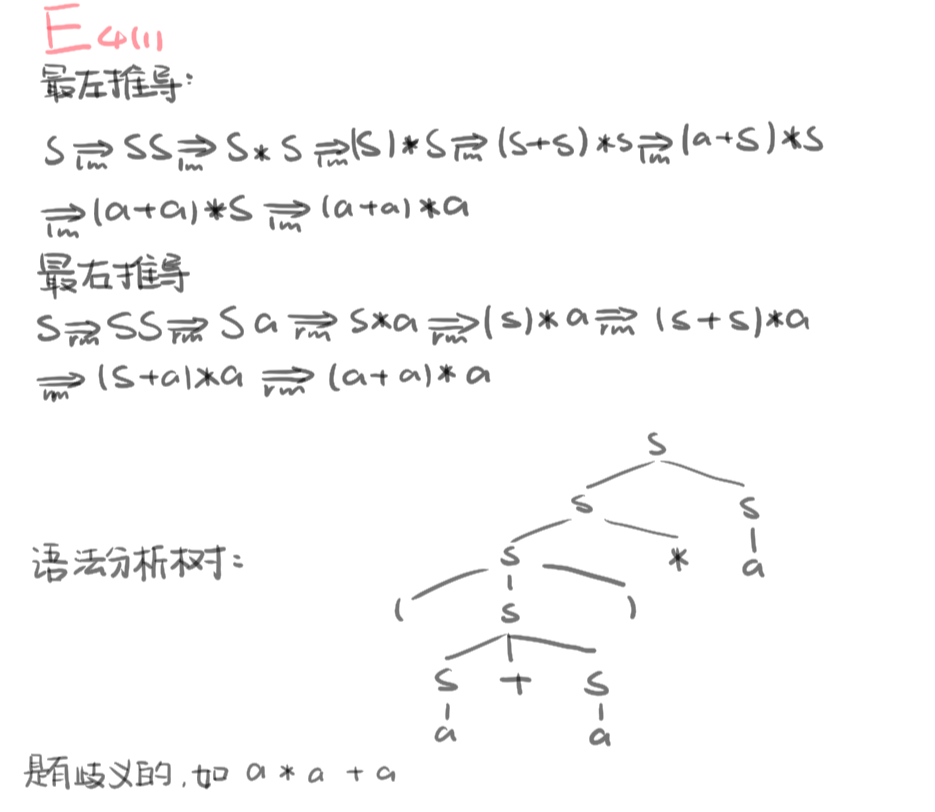
**分析匹配（Parsing）**

***注意 stack 需要反向装入\****

**练习题：对于第四题和第六题，完成下列题目**

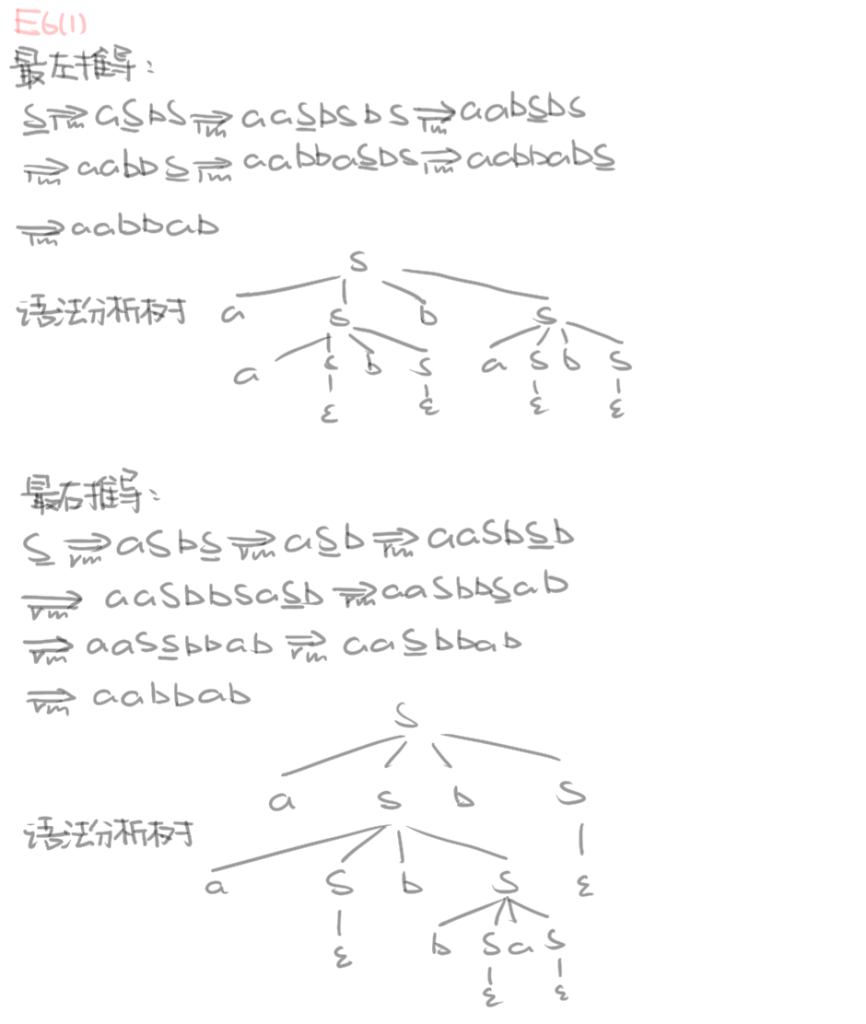


* **完成最左推导，最右推导，画语法树，判定是否有歧义**

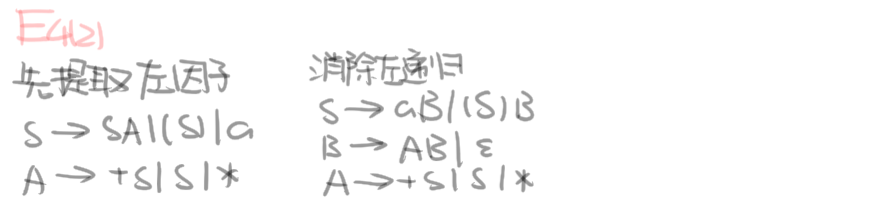


对于本题这个句子是没有歧义的，左右推导树是相同的，但是这个语法是有歧义

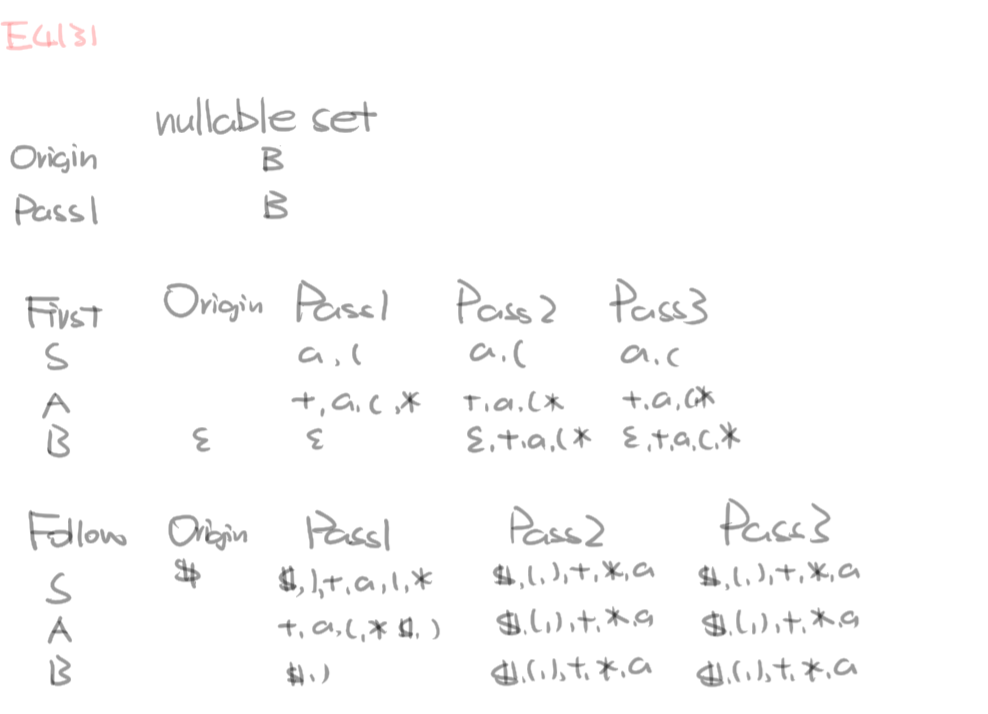
*如果一个文法可以****为某个句子生成多棵语法分析树****那就是****二义性（ambiguous）***



* **判定是否存在左因子，左递归情况，进行文法改写**



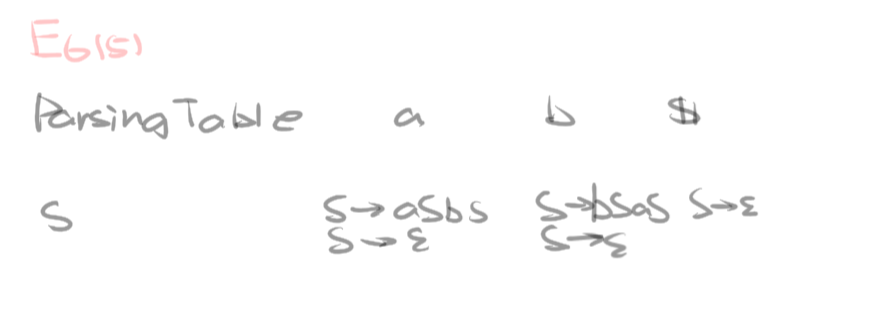
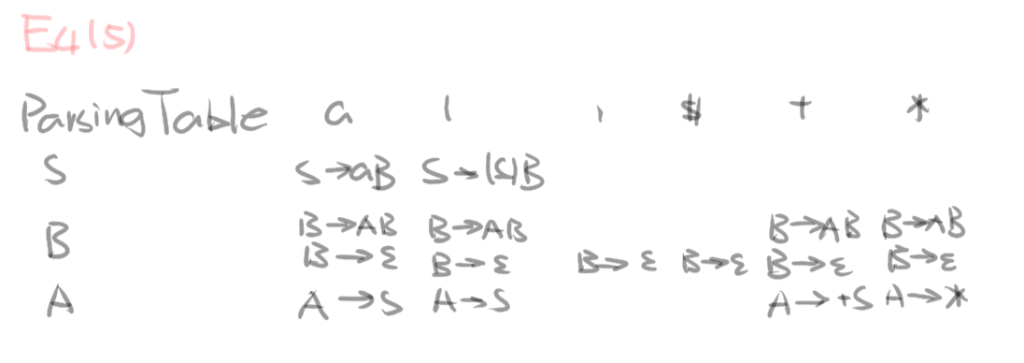
* **完成 First set / Follow set求解**



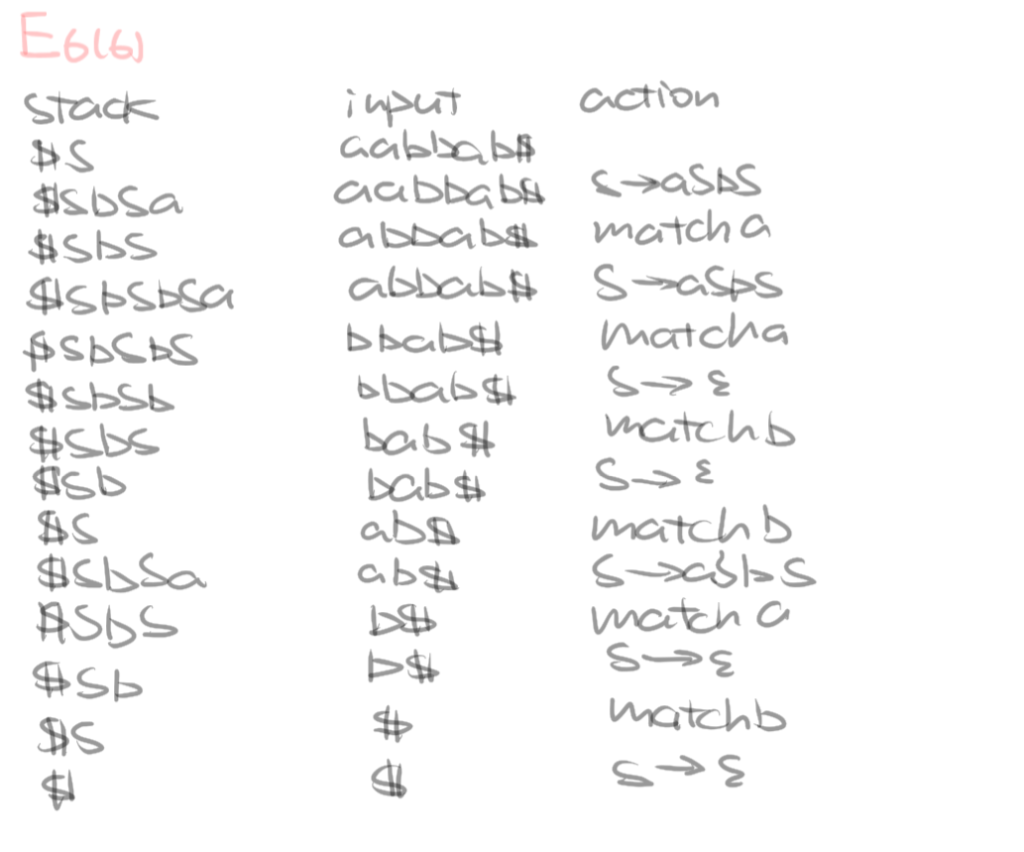
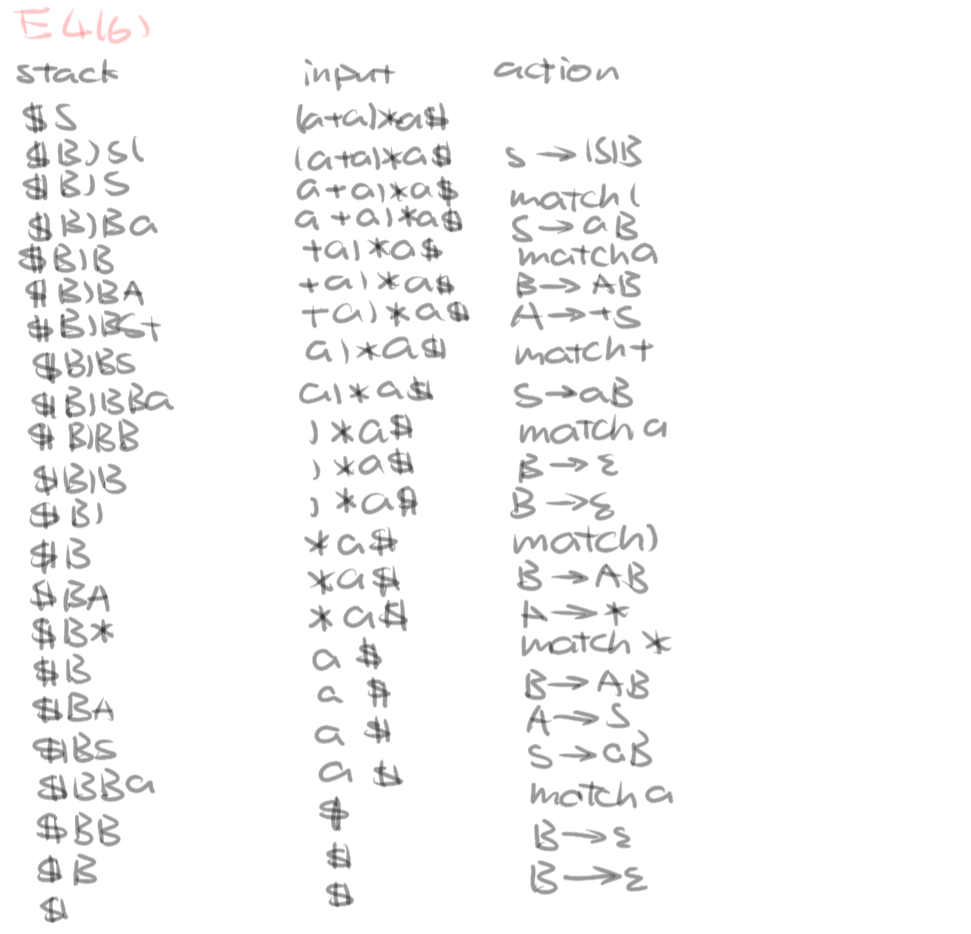
* **判定是否满足 LL(1) 条件**



* **构建 LL(1) parsing table**

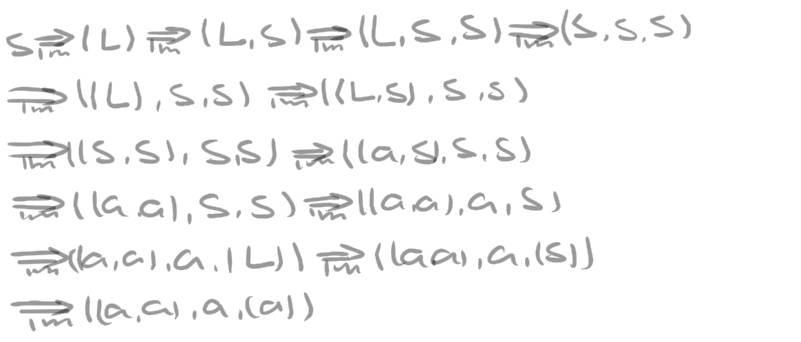


* **对所给字符串，完成基于 parsing table 的 LL(1) 语法解析**

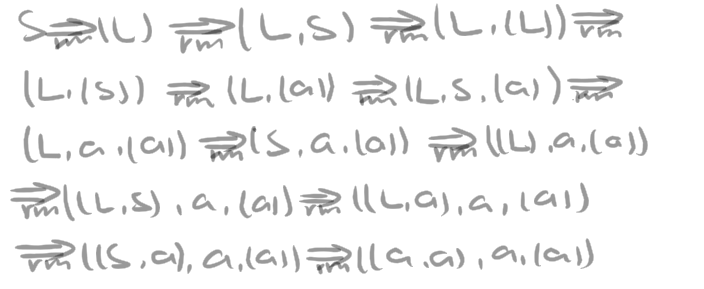


**关于 S->(L) I a and L-> L,S I S with string ((a, a), a, (a))**

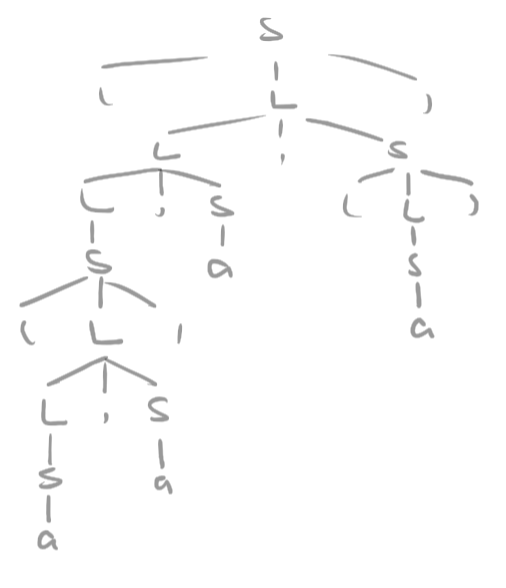
* **Leftmost derivation**



* **Rightmost derivation**



* **Give a parse tree for the string**



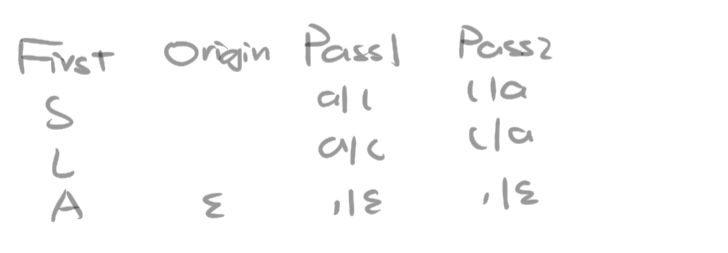
* **Is the grammar ambiguous or unambiguous? Justify your answer**



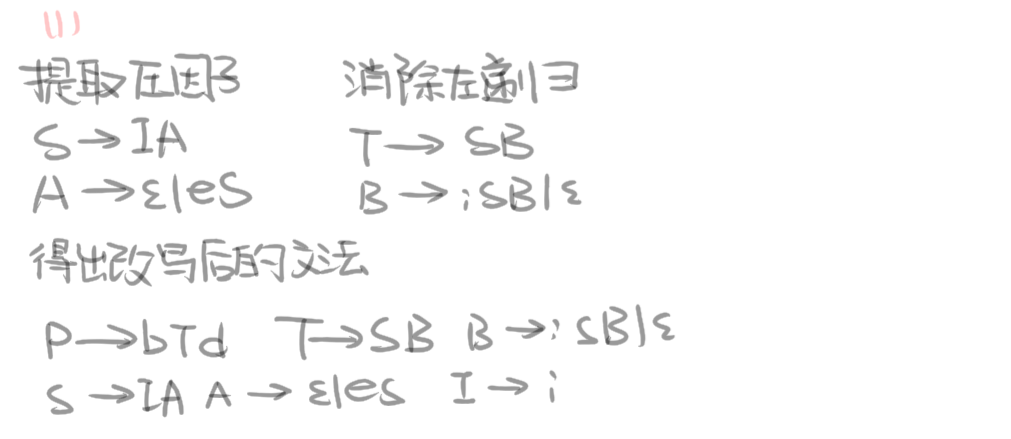
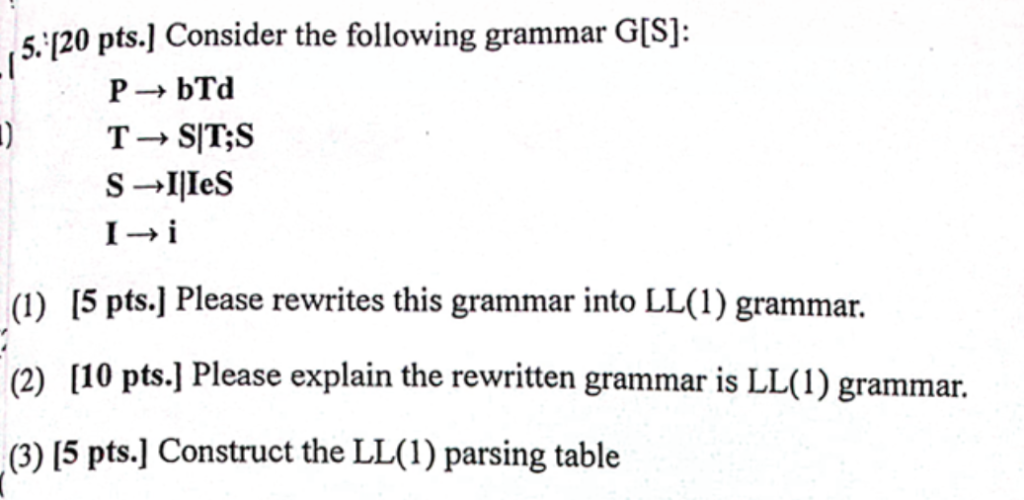
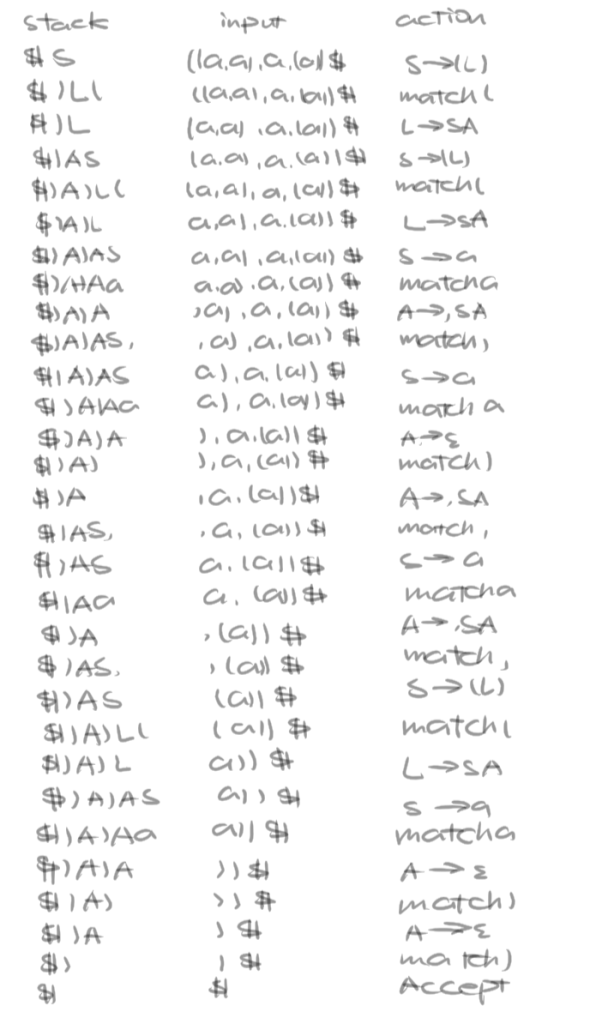
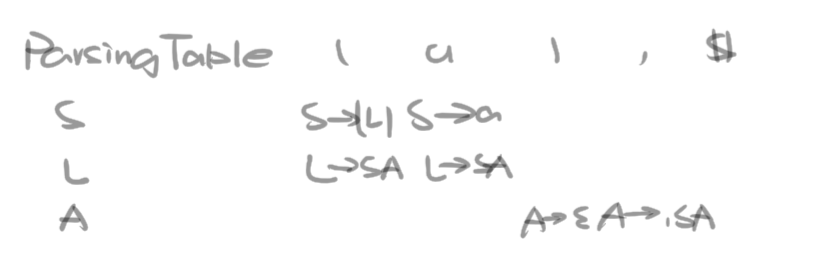
* **In addition to left factoring, eliminate left recursion from the original** **grammar**



* **Compute FIRST and FOLLOW**



* **求 Parsing Table 并且做解析**



窗体顶端