学号 E21714049 专业 计算机科学与技术 姓名 梅世祺

实验日期 **2019.11.29** 教师签字 成绩

实验报告

【实验名称】  **Chomsky文法类型判断**

【实验目的】

1. 深入理解4种文法类型的概念和区别；
2. 设计一个Chomsky文法类型判别器，判断0型文法、1型文法、2型文法和3型文法（为简单起见，可不做0型文法的判断）。

【实验要求】

­ 输入：一组任意的规则

输出：相应的Chomsky文法的类型

1. 文法的输入应简便；
2. 指明是哪一类Chomsky文法，并给出相应的四元组形式G(VN,VT,P,S)；
3. 说明：简单起见可以不考虑0型文法。

【实验原理】

设文法G(VN,VT，P,S)，其中VT是一个非空有限集,包含所有的终结符；VN是一个非空有限集，包含所有的非终结符；S是一个非终结符，称为开始符号；P是一个有限的产生式集合，每个产生式的形式为α→β；开始符号S必须在某个产生式的左部出现一次。

乔姆斯基（Chomsky）将文法分为4种类型，对于一个文法G：

1. 若产生式集合P中每一个产生式均满足：α∈(VN∪VT)\*且至少包含一个非终结符，β∈(VN∪VT)\*，则G是一个0型文法；
2. 在满足1）的前提下，若P中的每一个产生式均满足：|β|≥|α|（S→ε除外），则称G是1型文法或上下文有关文法；
3. 在满足1）、2）的前提下，若P中的每个产生式均满足，α是一个非终结符，β∈(VN∪VT)\*，则称G是2型文法或上下文无关文法；
4. 在满足1）、2）和3）的前提下，若P中每个产生式的形式都是A→aB|a（右线性文法）或A→Ba|a(左线性文法)，其中A和B都是非终结符，α∈VT\*，则G是3型文法或正规文法（注意：混合左右线性文法不是3型文法）。

上述4种类型的文法可以由下面的Veen图表示：



[来源: Chomsky Hierarchy in Theory of Computation](https://www.geeksforgeeks.org/chomsky-hierarchy-in-theory-of-computation/)

可知，4种类型的文法的定义是逐渐增加限制的，因此可采用自顶向下的算法实现。

【实验心得】

在正式开始实验前，参考了以下文章，复习了一些基本概念：

1. [无限制文法(0型文法) - 百度百科](https://baike.baidu.com/item/%E6%97%A0%E9%99%90%E5%88%B6%E6%96%87%E6%B3%95/8353653?fromtitle=0%E5%9E%8B%E6%96%87%E6%B3%95&fromid=19132381&fr=aladdin)
2. [0型文法、1型文法、2型文法和3型文法 – Elaine博客园](https://www.cnblogs.com/ITGirl00/p/3391211.html)
3. [Chomsky Hierarchy in Theory of Computation](https://www.geeksforgeeks.org/chomsky-hierarchy-in-theory-of-computation/)

由于老师的要求是将待测试的文法的写入一个文件中，再通过读取文件分析产生式得出文法类型，分析*文法定义源文件*中所有的产生式，这个可以尝试用正则表达式去匹配，然后得到形如 [ [left, right], [left, right], [left, right] … ] 二维数组，其中 left、right分别是产生式的左部和右部。最后根据【实验原理】中各类文法的定义，把定义翻译成相应的JavaScript代码即可。

【实验代码与结果】

实验代码：

[核心代码：](https://github.com/lolimay/Compiling-Principle-Experiment/blob/b7138f5bb4d012e280515eb5cf4e5c3ba067ea66/src/main.ts#L118)

// 若所有产生式满足：左部至少包含一个非终结符, 则是0型文法

*const* isType0 = productions.every(

([*left*]) *=>* left.split('').some(*symbol* *=>* isUpperCase(symbol))

);

// 所有产生式的左部符号个数小于等于右部符号个数(S➜ε除外), 满足以上条件则是1型文法

*const* isType1 = isType0 && productions.every(

([*left*, *right*]) *=>* left.length <= right.length

);

// 在满足1型文法的基础上，左部有且仅有一个非终结符，则是2型文法

*const* isType2 = isType1 && productions.every(

([*left*]) *=>* left.length === 1 && isUpperCase(left)

);

// 判断是否是左线性文法

*const* isLeftLinear = isType2 && productions.every(([, *right*]) *=>* {

switch (right.length) {

case 1:

return true;

case 2:

return isUpperCase(right[0]) && isLowerCase(right[1]);

default:

return false;

}

});

// 判断是否是右线性文法

*const* isRightLinear = isType2 && productions.every(([, *right*]) *=>* {

switch (right.length) {

case 1:

return true;

case 2:

return isLowerCase(right[0]) && isUpperCase(right[1]);

default:

return false;

}

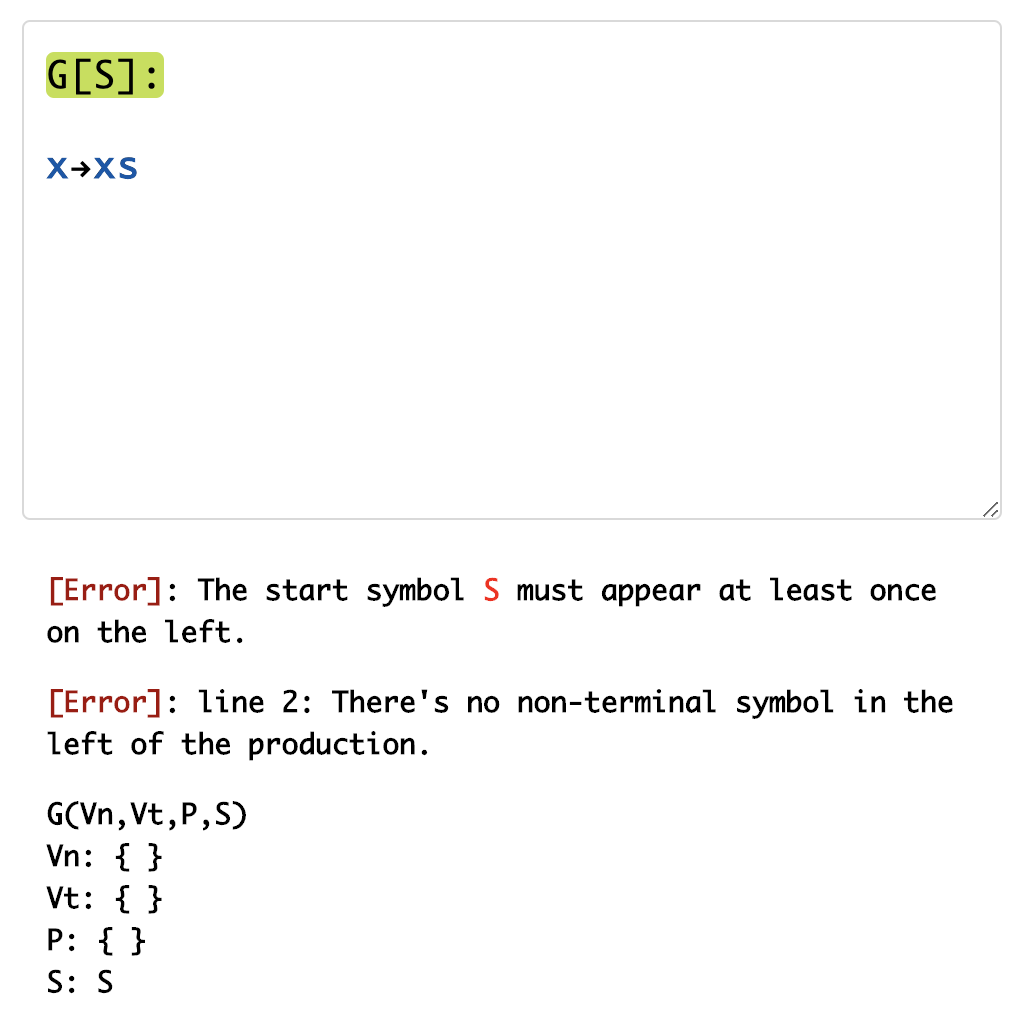
});

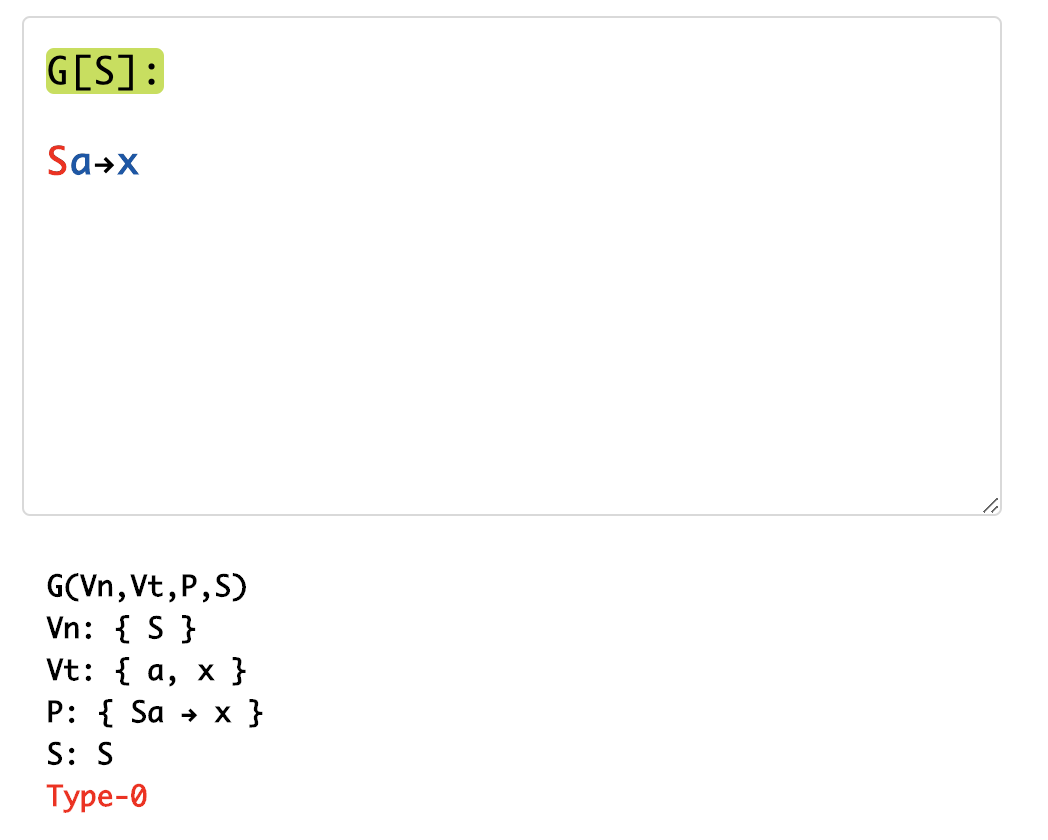
// 3型文法是左线性文法或右线性文法（混用左右则不是）

*const* isType3 = isLeftLinear || isRightLinear;

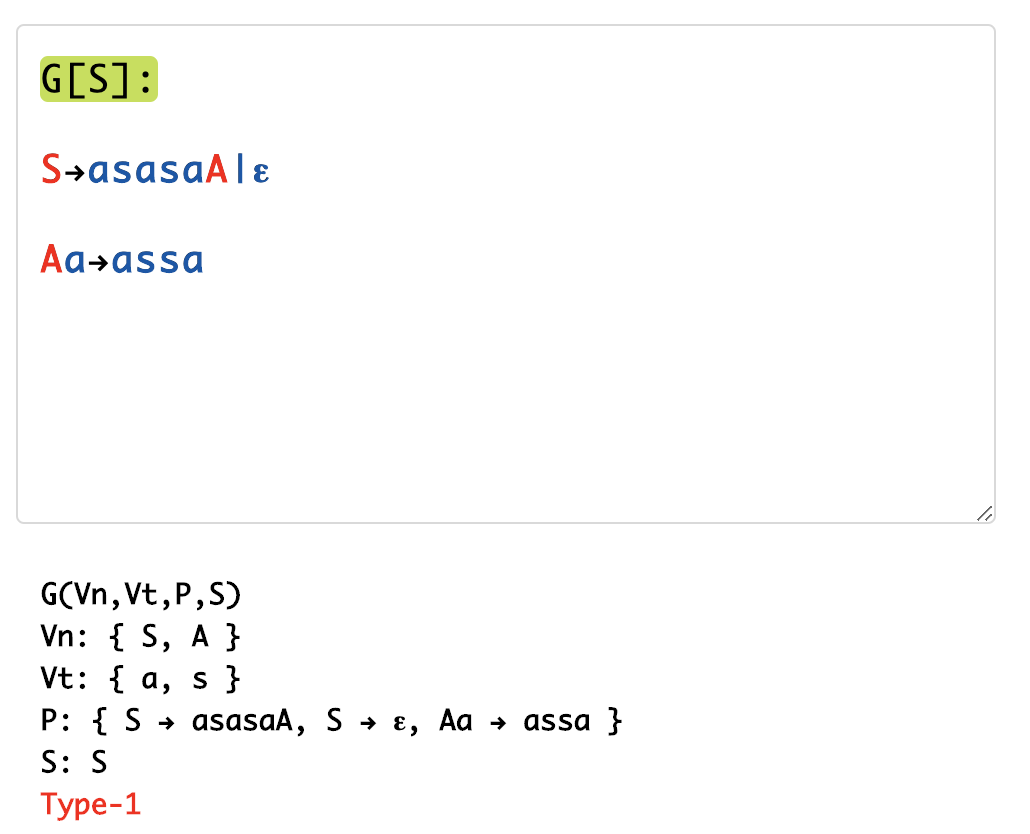
实验结果：

不满足开始符号至少在一条产生式左部出现的情况或者不满足产生式左部至少有一个非终结符的情况：

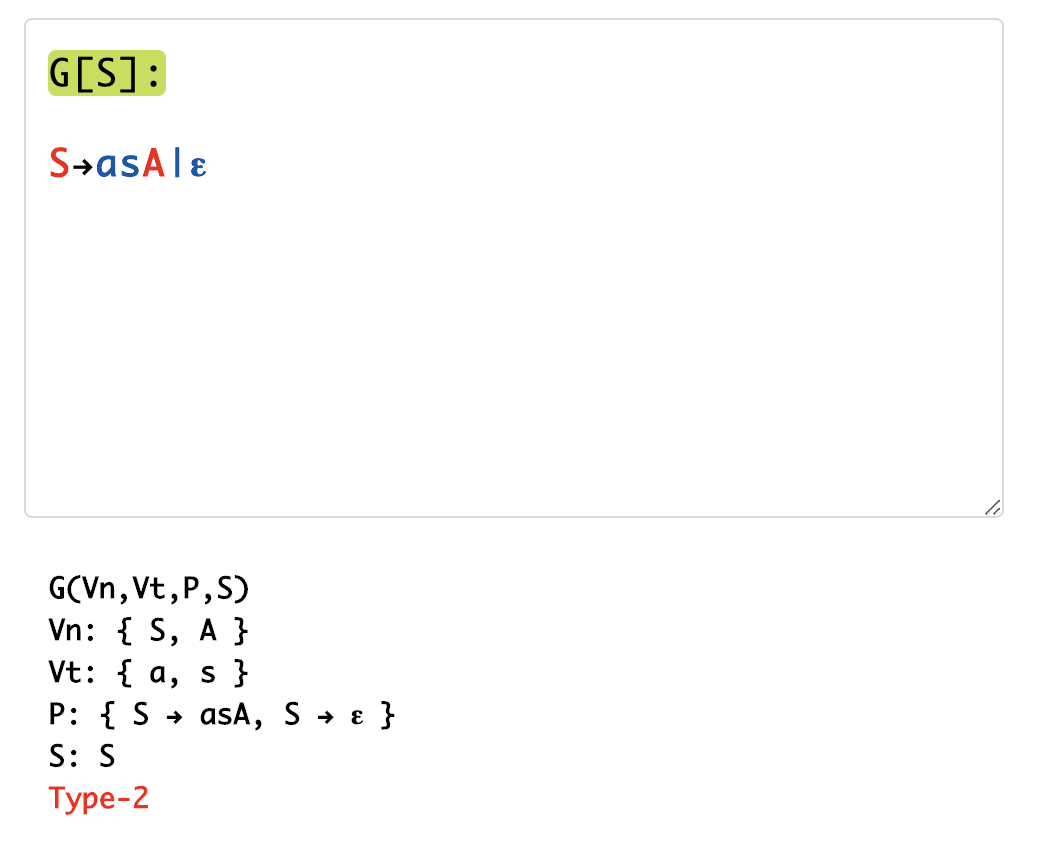




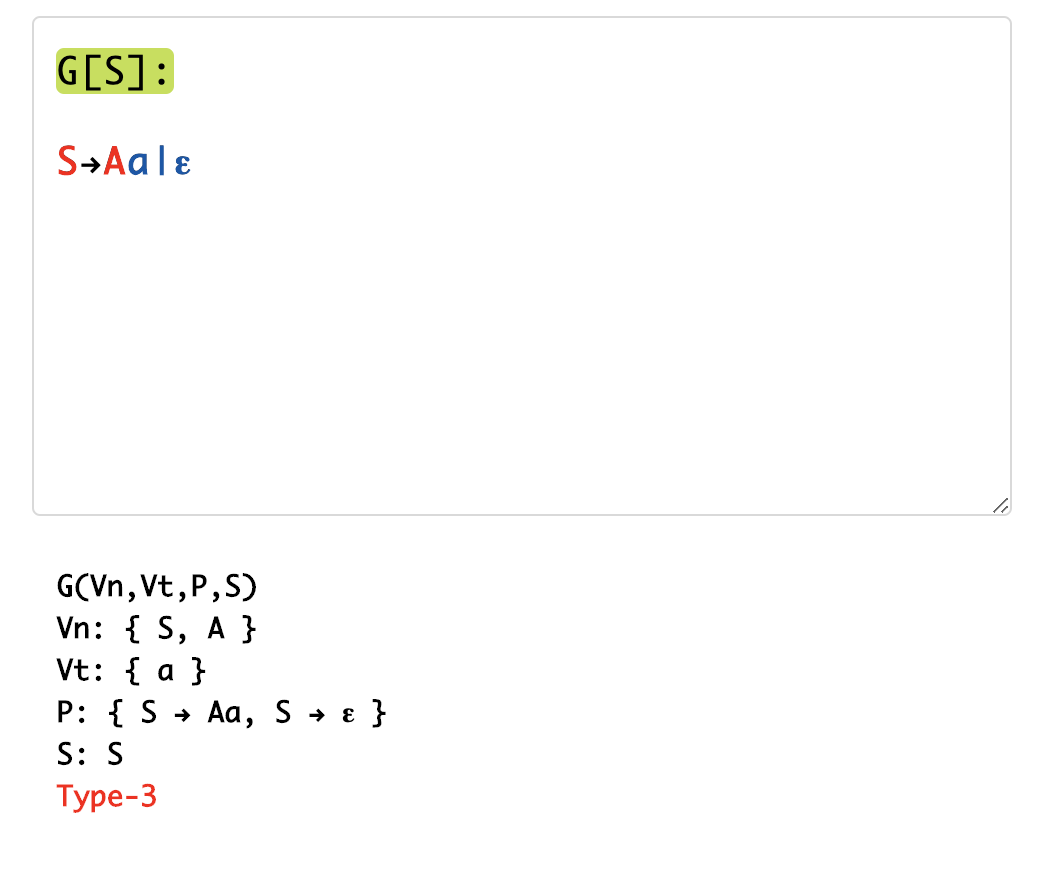
一个 Type-0 文法的例子



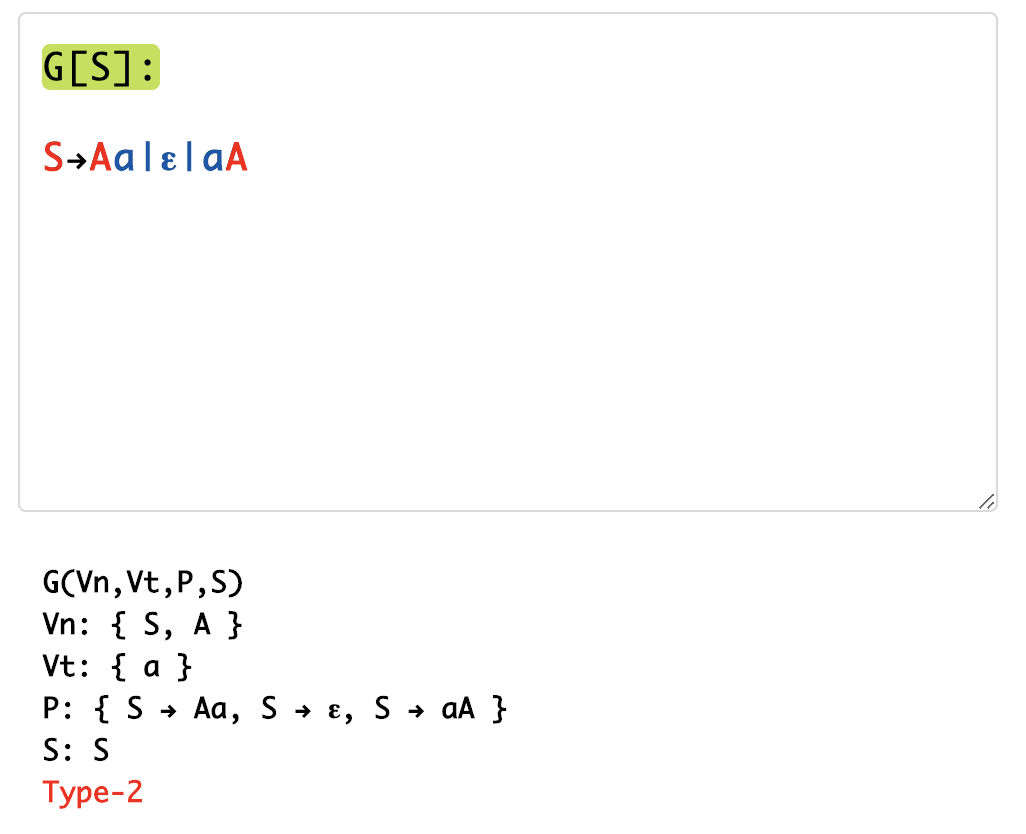
一个 Type-1 文法的例子



一个 Type-2 文法的例子



一个 Type-3 （正规文法）的例子



左线性文法和右线性文法混用则不是正规文法

在线演示地址👉 <https://chomsky.lolimay.cn/>