**北京邮电大**

**计算机学院(国家示范性软件学院)**

**2022-2023 学年第 2 学期**

**实验项目总结报告**

**课程名称： 形式语言与自动机**

**项目名称：实验二: 设计上下文无关文法的变换算法**

**项目完成人：**

**负责人姓名：**郭栩源 **学号：**2021213586

**姓名：**姜杨皓 **学号：**2021213650

**姓名：**胡栋康 **学号：**2021210166

**姓名：**邢智恺 **学号：**2021213585

**指导教师：杨正球 助教：杨惠元**

**日 期： 2023年 6月8日**

## **1. 项目要求及分工**

**1.1 实验目的**

编程实现上下文无关文法的变换算法，用于消除文法中的ε产生式、单产生式、以及无用符号。程序的输入是一个上下文无关文法，输出是与该文法等价的没有ε产生式、单产生式、无用符号的上下文无关文法。

**1.2 实验要求**

1. 采用分组实验
2. 编程语言要求使用C/C++，需要使用头歌教学云平台进行测试验证代码
3. 要求程序运行正确，设计风格好，文档描述清晰

**1.3 项目分工**

姜杨皓负责代码框架，整体算法思路以及伪代码

郭栩源和邢智恺负责代码编写，代码测试

胡栋康负责实验报告总结

## **2. 项目开发环境**

1. 语言：C++
2. 操作系统：macOS，Windows
3. 集成编译环境：在visual studio code中使用mingw编译和gdb调试。
4. 测试环境：使用头歌教学云平台进行测试验证代码。

## **3. 需求分析**

**3.1 项目需求：**

上下文无关文法对产生式的右部没有限制，这种完全自由的形式有时会对文法分析带来不良影响，而对文法的某些限制形式在应用中更方便。通过上下文无关文法的变换，在不改变文法的语言生成能力的前提下，可以消除文法中的ε产生式、单产生式、以及无用符号。

本实验要求编程实现消除上下文无关文法中的ε产生式、单产生式、以及无用符号的算法。输入是一个上下文无关文法，输出是与该文法等价的没有ε产生式、单产生式、无用符号的上下文无关文法。

**3.2 输入输出格式要求：**

**1.输入要求**

输入的第一行:非终结符集合。

输入的第二行:终结符集合。

接下来的若干行:每一行代表一个产生式,第一个字符表示产生式左侧的非终结符,ε用N表示。

**2.详细输入格式示例如下，按顺序写文法的N，T，P，S**

N={A,B,C,D,S}

T={a,b,c,d}

P:

S-->B|a|bA|ccD

A-->N|abB

B-->aA

C-->ddC

D-->ddd

S=S

**3.输出要求：**

排序统一按照ascii码由小到大排序（产生式左部和右部都按照这个顺序），每一个产生式左边有一个’\t’做分割，如果原输入的开始符号S能推导出空产生式导致需要生成新的非终结符，则生成S1。

**4.对应详细输出格式如下：**

N={A,B,D,S}

T={a,b,c,d}

P：

A-->abB

B-->a|aA

D-->ddd

S-->a|aA|b|bA|ccD

S=S

## **4.概要设计**

**4.1 核心思想**

核心思想是实现上下文无关文法的变换算法，用于消除文法中的ε产生式、单产生式、以及无用符号。

具体来说，该程序的主要步骤包括：

**1.消除ε产生式。**

消除ε产生式的步骤可以通过计算所有可以推导出空串的非终结符的集合，并将原始文法中的每个产生式转化为多个产生式来实现。

**2.消除单一产生式。**

消除单一产生式的步骤可以通过对于每个非终结符，构造一个集合，该集合包含所有可以从该非终结符推导出的单一非终结符，然后根据集合中的信息，构造一个新的文法来实现。

**3.消除无用产生式。**

消除无用产生式的步骤可以先使用消除ε产生式的算法消除所有无法推导出终结符的非终结符和ε产生式，得到一个新的文法，然后使用消除单一产生式的算法消除所有不可达的非终结符，得到最终的简化文法。

**4.2 设计思路**

该代码的设计思路可以分为以下几个步骤：

（1）定义文法类Grammar和产生式类Production，其中Grammar类包含一组产生式和起始符号，而Production类包含产生式的左部和右部。

（2）定义输入函数input和输出函数output，分别用于读取输入文法和输出简化后的文法。

（3）定义algorithm1() 和 algorithm2() 函数都是用于计算一些辅助信息。algorithm1() 函数用于计算所有可以推导出空串的非终结符的集合；algorithm2() 函数用于计算所有可达的非终结符的集合。

（4）定义消除ε产生式的函数eliminate\_epsilon\_productions，使用迭代的方式来找到所有可以推导出ε的非终结符，并将其从产生式中删除，然后返回一个新的文法。

（5）定义消除单一非终结符产生式的函数eliminate\_unit\_productions，使用迭代的方式来找到所有可以到达的非终结符，并将单一非终结符的产生式转换为多个不包含单一非终结符的产生式，然后返回一个新的文法。

（6）定义消除无用产生式的函数eliminate\_useless\_productions，使用算法1和算法2来找到所有无用的产生式，并将其从产生式中删除，然后返回一个新的文法。

（7）在主函数中调用上述函数，依次对输入文法进行简化，并输出最终的简化文法。

**4.3 核心算法**

该代码的核心算法包括以下几个部分：

**4.3.1 消除ε产生式**

主要思路是计算出所有可以推导出空串的非终结符的集合，然后将原始文法中的每个产生式转化为多个产生式，其中每个新的产生式都是将右部中的某个可以推导出空串的非终结符替换为 ε。

具体算法思路如下：

（1）计算所有可以推导出空串的非终结符的集合 A。具体来说，对于每个非终结符，构造一个空集合，然后遍历所有产生式，对于每个产生式，如果其右部中所有非终结符都在集合 A 中，则将左部的非终结符加入到集合 A 中。不断重复这一过程，直到集合 A 不再变化。

（2）对于每个产生式，如果其右部中包含可以推导出空串的非终结符，则将其拆分成多个产生式。具体来说，对于每个右部中包含可以推导出空串的非终结符的产生式，将其拆分成多个产生式，其中每个新的产生式都是将右部中的该非终结符替换为 ε。

（3）如果起始符号可以推导出空串，则需要添加一个新的起始符号，并添加一个新的产生式，将新的起始符号推导为原始文法的起始符号和 ε。

**4.3.2 消除单一产生式**

主要思路是对于每个非终结符，构造一个集合，该集合包含所有可以从该非终结符推导出的单一非终结符。然后，根据集合中的信息，构造一个新的文法，该文法不包含单一产生式。

具体算法思路如下：

（1）对于每个非终结符，构造一个集合 S，该集合包含所有可以从该非终结符推导出的单一非终结符。

（2）对于每个非终结符 A，遍历其集合 S 中的所有非终结符 B，将所有以 B 为左部的产生式加入到 A 的产生式集合中。这样，就得到了一个新的文法，其中不包含单一产生式。

（3）对于每个非终结符 A，如果它的产生式集合中包含某个非终结符 B 的产生式，则将 B 的集合 S 中的所有非终结符加入到 A 的集合 S 中。不断重复这一过程，直到集合 S 不再变化。

（4）如果起始符号可以推导出单一非终结符，则需要添加一个新的起始符号，并添加一个新的产生式，将新的起始符号推导为原始文法的起始符号和单一非终结符。

**4.3.3 消除无用产生式**

主要思路是先使用消除ε产生式的算法消除所有无法推导出终结符的非终结符和ε产生式，得到一个新的文法。然后使用消除单一产生式的算法消除所有不可达的非终结符，得到最终的简化文法。

具体算法思路如下：

（1）计算所有可达的非终结符的集合 V。具体来说，从起始符号开始，不断遍历所有产生式，对于每个产生式，如果其右部中所有非终结符都在已知的可达非终结符集合中，则将左部的非终结符加入到可达非终结符集合中。不断重复这一过程，直到集合 V 不再变化。

（2）计算所有可以推导出终结符的非终结符的集合 T。具体来说，对于每个终结符，将其所在的产生式的左部非终结符加入到集合 T 中。然后，对于每个非终结符，遍历其产生式集合中的所有产生式，如果该产生式的右部中所有的非终结符都在集合 T 中，则将该产生式的左部非终结符加入到集合 T 中。不断重复这一过程，直到集合 T 不再变化。

（3）构造一个新的文法，其中只包含左部非终结符在集合 V 中且可以推导出终结符的产生式。具体来说，遍历原始文法中的所有产生式，如果该产生式的左部非终结符在集合 V 中且可以推导出终结符，则将该产生式加入到新的文法中。

（4）如果起始符号不在集合 V 中，则需要添加一个新的起始符号，并添加一个新的产生式，将新的起始符号推导为原始文法的起始符号。

## **5. 实际代码**

**5.1 数据结构**

*// 定义文法*

struct Grammar

{

    set<char> nonterminal; *// 非终结符，是一个char类型的集合*

    set<char> terminal; *// 终结符，是一个char类型的集合*

    map<char, set<string>> production; *// 生成式，是从char到一个string集合的映射*

    char start; *// 初始非终结符*

};

**5.2 函数接口**

*//函数接口*

void input(Grammar &*G*);

void output(Grammar *G1*);

Grammar algorithm1(Grammar *G*, set<char> *terminal*); *算法一：消除无用终结符*

Grammar algorithm2(Grammar *G*); *// 算法二：消除无用产生式*

Grammar eliminate\_epsilon\_productions(Grammar *G*);

Grammar eliminate\_unit\_productions(Grammar *G*);

Grammar eliminate\_useless\_productions(Grammar *G*);

**5.3 主函数**

int main()

{

    Grammar G, G1, G2, G3;

    input(G);

    set<char> temp;

    temp.insert('N');

    G1 = eliminate\_epsilon\_productions(G);

    G2 = eliminate\_unit\_productions(G1);

    G3 = eliminate\_useless\_productions(G2);//

    output(G3);

}

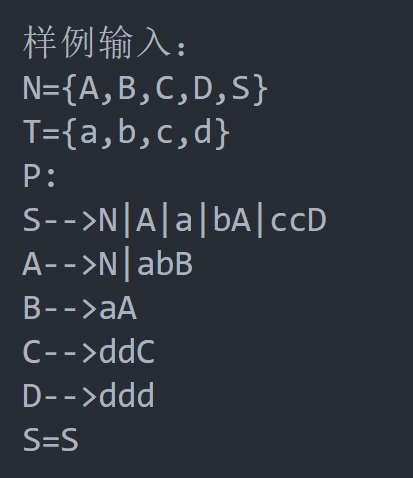
**5.4函数实现**

详见源程序代码。

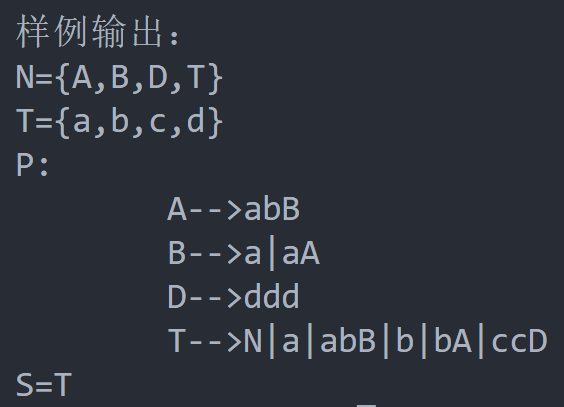
## **6.程序测试及上机执行效果**

* 1. **测试用例**

**（1）样例输入**



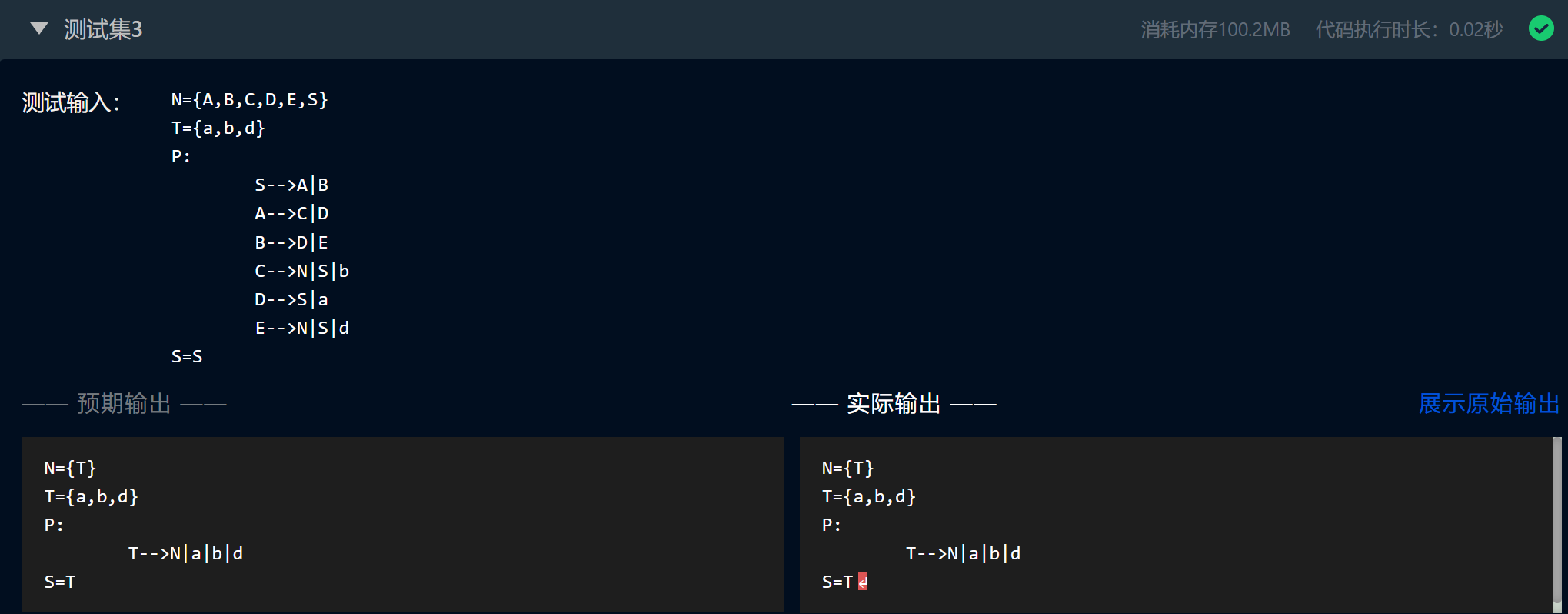
**（2）样例输出**

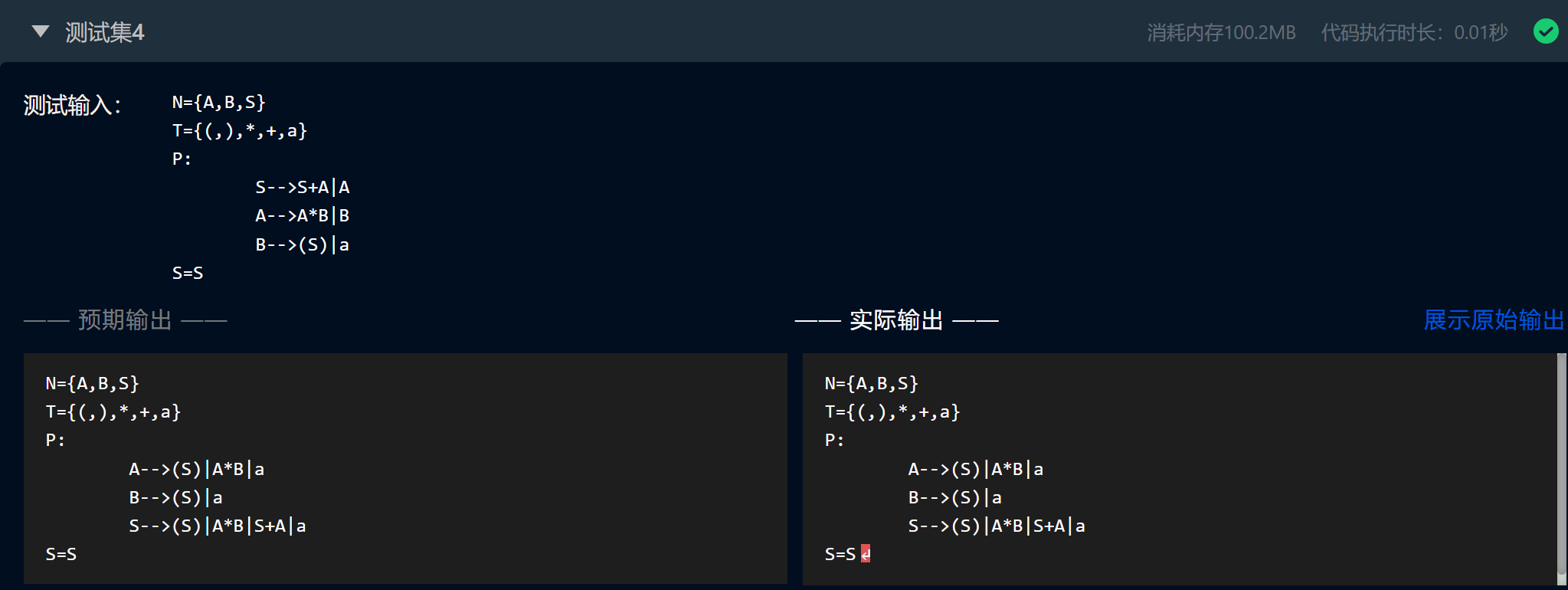


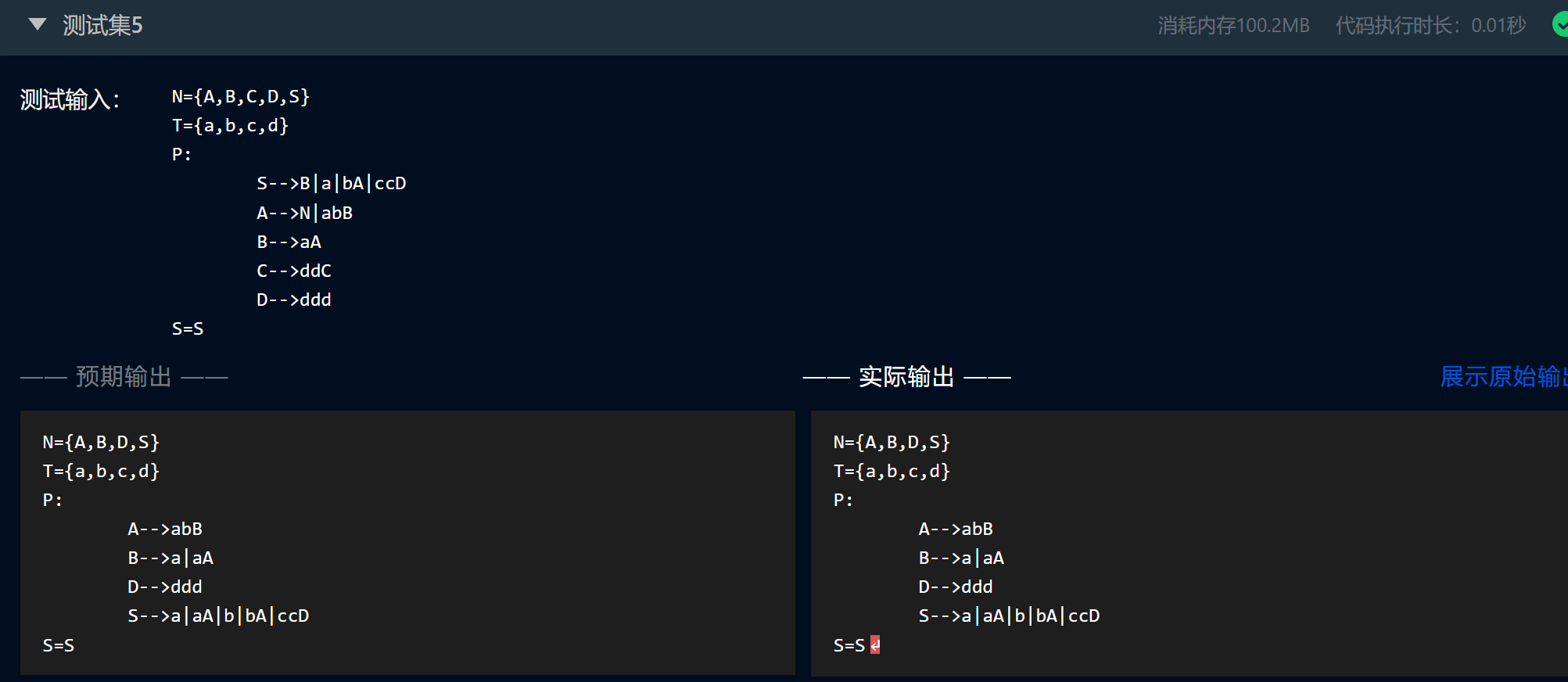
**6.2 执行效果**

****







****

## **7. 改进思路和方法**

**1.添加异常处理**

目前代码中没有进行异常处理，如果输入的文法不符合规范，程序可能会出现错误。可以考虑添加异常处理机制，如输入格式不正确等情况，及时给出提示和错误信息。

**2.增加用户交互性**

目前代码只是简单地读入文法并输出简化后的结果，可以考虑增加用户交互性，如通过命令行界面或图形界面与用户进行交互，提供更加友好的使用体验。

**3.优化算法实现**

该代码中的算法实现已经比较简单和直观，但是可能存在一些效率上的问题，可以考虑优化算法实现，提高代码的运行效率和性能。例如，可以使用更快速的数据结构或算法来实现集合运算或文法遍历等操作。