北京郵電大學

形式语言与自动机课程实验（二）

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

小组：第七小组

成员：张晨阳 2022211683

廖轩毅 2022211637

梁维熙 2022211124

金建名 2022211130

2024 年 6 月 2 号

目录

1 实验目的和要求

1.1 实验目的

本次实验分为两个部分，第一部分为设计上下文无关文法的变换算法，第二部分为构造与下推自动机等价的上下文无关文法。

1.1.1 设计上下文无关文法的变换算法

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。

1.1.2 构造与下推自动机等价的上下文无关文法

编程实现由下推自动机构造等价的上下文无关文法的算法。

1.2 实验要求

（1）采用分组实验，每组学生 3~4 人，本小班内自由组合，培养学生团队合作能力。

（2）编程语言不限，进行测试验证。

（3）要求程序运行正确，设计风格好，文档描述清晰，并且按期提交实验报告，源代码，及可执行程序。文件命名方式：组长班级+组长姓名+文件类型（报告/代码/程序）。三个文件打包提交，命名方式：实验二+组长班级+组长姓名。

（4）实验报告至少包含以下内容：

① 小组成员，班级，姓名，学号；成员分工。

② 实验环境描述：所使用的语言等。

③ 程序的设计思路及核心算法。

④ 程序的输入格式，输出格式。

⑤ 程序的测试用例，输入，输出，以及执行效果（可截图）。

⑥ 改进思路和方法（可选）。

2 实验环境

操作系统：Windows11。

IDE：Visual Studio Code。

编程语言：C++17。

3 实验需求

3.1 设计上下文无关文法的变换算法

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。

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。

至少使用如下文法中的产生式进行程序的正确性验证。

```
S→a|bA|B|ccD
A→abB|ε
B→aA
C→ddC
D→ddd
```

3.2 构造与下推自动机等价的上下文无关文法

下推自动机和上下文无关文法，是用于描述上下文无关语言的两种方式。对于给定的一个下推自动机，必存在一个上下文无关文法，使得该文法产生的语言与下推自动机接受的语言等价。

要求如下：

（1）编程实现由下推自动机构造等价的上下文无关文法的算法。输入为一个下推自动机，输出为与该下推自动机等价的上下文无关文法。

（2）将输出的等价上下文无关文法作为输入，利用所实现的上下文无关文法变换算法，输出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。

至少使用如下的下推自动机进行程序的正确性验证。

设 PDA M＝({q0,q1},{a,b},{B,z0},δ,q0,z0,Φ)

δ 定义为：

```
δ(q0,b,z0) = {(q0,Bz0)}
δ(q0,b,B) = {(q0,BB)}
δ(q0,a,B) = {(q1,ε)}
δ(q1,a,B) = {(q1,ε)}
δ(q1,ε,B) = {(q1,ε)}
δ(q1,ε,z0) = {(q1,ε)}
```

4 程序设计思路及核心算法

4.1 设计上下文无关文法的变换算法

4.1.1 程序设计思路

4.1.1.1 输入

输入内容是一个上下文无关文法，按顺序输入文法的 N，T，P，S。

空产生式用 # 表示。

第 1 行：非终结符集合: SABCD。

第 2 行：终结符集合: abc。

第 3 行：从第三行开始，开始输入 P，P 中有几个转换就输入几行，转化的这几行均以换行为结尾，当 P 中的转换式输入完成后，输入符号@作为结尾。

4.1.1.2 对上下文无关文法进行变换

消除 ε 产生式的过程如下：

（1）找到所有可以直接推出 ε 的非终结符；

（2）不断找到新的可以推出 ε 的非终结符，直到集合不再变化；

（3）根据新的产生式集合生成新的产生式集合 P1。

消除单产生式的过程如下：

（1）遍历原始产生式集合 P 中的每个非终结符 A；

（2）找出所有可以直接推出单个非终结符的产生式，并将这些非终结符加入 canEpsilon 集合中；

（3）不断扩展 canEpsilon，直到 canEpsilon 不再变化为止；

（4）构建新的产生式列表；

（5）去除重复的产生式，如果新的产生式列表为空，则添加一个产生式 A→#。

消除无用符号的过程如下：

（1）找出所有能直接推导出终结符的非终结符，并将其加入 N1 集合中；

（2）不断扩展 N1 集合，直到 N1 不再发生变化；

（3）从原始产生式集合 P 中删除所有左部非终结符不在 N1 集合中的产生式；

（4）反向遍历所有的“可达性”非终结符，并将它们加入到新的非终结符集合 canEpsilion 中；

（5）同样从原始产生式集合 P 中删除所有左部非终结符不在 canEpsilion 集合中的产生式。

4.1.1.3 输出

输出是通过 printout()函数实现的。

首先输出非终结符集合 N 和终结符集合 T，并通过循环遍历输出各个元素，其中使用了迭代器并判断是否为最后一个元素来避免末尾多余的逗号。然后按照 key 对产生式集合 P 中的元素进行排序，并依次输出左部符号和右部符号，其中右部符号按照字典序排序，同样使用了迭代器并判断是否为最后一个元素来避免末尾多余的竖线。最后输出开始符号 S 即可。

需要注意的是，这里使用了一些 C++11 中新增的语法特性，比如 auto 用于获取迭代器类型和范围 for 循环，还有 lambda 表达式和 map 排序等。

4.1.2 核心算法

4.1.2.1 消除 ε 产生式

eli_epsilon()函数用于消除生成 ε 的产生式。

它通过以下步骤实现：

（1）首先遍历产生式集合 P 中的每个产生式，找出所有可以直接推出空串的非终结符，加入集合 canEpsilon。具体方法是：对每个非终结符 key，遍历它对应的产生式列表 value，如果 value 中有产生式右部只包含一个'#'的话，即能直接推出空串，则将 key 加入 canEpsilon 集合。

（2）然后不断扩展 canEpsilon 集合，直到不再有新的可以推出空串的非终结符加入。扩展方法是：对每个非终结符 key，遍历它对应的产生式列表 value，如果 value 中所有的符号都在 canEpsilon 集合中，则 key 也可以推出空串，将其加入 canEpsilon。

（3）根据 canEpsilon 集合构建新的产生式集合 P1。对每个非终结符 key，遍历它对应的产生式列表 value，对 value 中的每个产生式：

①计算其中可以推出空串的非终结符的个数 canEpsilon.size()。

②枚举所有这些非终结符的组合。

③对于每个组合：

1）生成新的不含空串的产生式。

2）如果组合中包含该非终结符，则保留该非终结符。

3）否则舍去该非终结符。

④加入 P1。

（4）如果原始起始符号 S 可以推出空串，则选择 T 作为新的起始符号，同时为 P1 添加一个新的产生式 T→S。

（5）P1 代替原始的产生式集合 P。

4.1.2.2 消除单产生式

eli_single()函数用于消除每个非终结符只有一个产生式的产生式。

它通过以下步骤实现：

（1）定义新的非终结符集合 new_N、新的产生式集合 new_P 和新的起始符号 new_start。

（2）遍历每个非终结符 A，找出所有可以从 A 直接推出单个非终结符的产生式，将这些非终结符加入 new_N 集合。具体是：遍历 A 对应的产生式列表，如果存在产生式的右部只有一个非终结符 B，则将 B 加入 new_N。

（3）不断扩展 new_N 集合，直到不再有新的非终结符加入。扩展方法是：对 new_N 中的每个非终结符 B，找出从 B 可以直接推出单个非终结符的产生式，并将该非终结符加入 new_N。

（4）构建新的产生式列表 new_P[A]。对 new_N 中的每个非终结符 B，将从 B 推出的非单产生式的产生式加入 new_P[A]。如果 new_P[A]为空，添加 A→N 作为默认产生式，以免删除 A 后影响文法的语言。

（5）如果 A 是原始起始符号 S，则更新 new_start 为 S（如果 S 在 new_N 集合中或 new_P 的第一个 key），该步保证新的文法有唯一的起始符号。

（6）将新的产生式集合 new_P 替换原始产生式集合 P。

4.1.2.3 消除无用符号

eli_useless()函数用于消除在文法中不可达的非终结符和终结符。

它分为两个主要步骤：

（1）生成符号集 N1，包含所有可以从起始符号 S 推导出的终结符集合。

（2）创建新的产生式集合 P1，只包含在 N1 中的非终结符的产生式。

（3）更新非终结符集合 N 和终结符集合 T，只保留在 N1 中的符号。

（4）创建新的产生式集合 P2，只包含在新的 N 和 T 中的产生式。

（5）更新产生式集合 P 为 P2。

4.2 构造与下推自动机等价的上下文无关文法

4.2.1 程序设计思路

4.2.1.1 输入

该程序的输入通过 main 函数实现，其中使用如下的数据结构存储 PDA 的状态：

```
// PDA 定义
struct PDA {
    set<State> Q;                                                    // 状态集合
    set<Terminal> T_PDA;                                             // 终结符集合
    set<StackSymbol> Gamma;                                          // 符号栈集合
    map<tuple<State, Terminal, StackSymbol>, set<pair<State, string>>> delta;  // 转换函数
    State q0;                                                        // 初始状态
    StackSymbol z0;                                                  // 初始栈符号
    set<State> Phi;                                                  // 接受状态集合
};
```

首先进行状态集合的读入。通过用户输入的状态数判断应该读入的状态个数，由于使用的是字符串类型存储，所以需要在每个状态之间使用回车进行分隔。终结符集合、符号栈符号以及最终状态的读入也是如此。

接着分别进行初始状态、初始栈符号的读入。

读入完成后，进行转换函数的读入。由于转换函数的形式为(q, a, z)→(newQ, str)，故使用 stringstream()函数进行读入。考虑到字符串读入必须有相应的输入，无法使用空输入作为空字符串，故规定 # 作为空字符串。

当用户输入完成后程序进行相应的转换。

4.2.1.2 从 PDA 生成对应的 CFG

CFG 中有四个相应的元素，分别是 N，T，P，S，从 PDA 生成对应的 CFG 即为构造这四个对应的元素。

首先通过全排列的方式可以得到对应的非终结符集合 N。

由于构造时不会改变终结符集合，所以 T 即为 PDA 中的 T_PDA。

对于 S_CFG，设置 S 作为 CFG 的起始符号，同时将其加入非终结符集合中。

对于构造相应的生成式集合 P，需要分两种情况讨论：当转化前的转换式往

栈中重新压入的状态句子为空句子时，应直接将转移到相应状态的生成式压入 P 中；当转化前的转换式往栈中新压入了栈中符号时，应通过全排列的方式找出相应的全部生成式，再压入 P 中。

这样便得到相应的未化简的 CFG。

4.2.1.3 删去 CFG 中的无用符号

首先遍历所有生成式左边的非终结符，寻找没有出现过的符号。若某个非终结符在生成式左边从未出现过，则可以判定这一非终结符是无用符号，因为生成包含这一符号的句子无法将所有的非终结符转换为终结符。

然后将所有包含有标记为无用的非终结符和终结符的生成式删除，再将这些非终结符和终结符从 N 与 T 中删去，即可完成初步的无用符号的删除。

4.2.1.4 完成化简

将生成的 CFG 输入到第一部分的程序中即可完成化简。

4.2.1.5 输出

输出是通过 printout()函数实现的，将 N、T、P、S 按照顺序打印即可。

5 测试及执行效果

5.1 设计上下文无关文法的变换算法

5.1.1 测试用例

5.1.1.1 输入

```
SABCD
abcd
S a|bA|B|ccD
A abB|#
B aA
C ddC
D ddd
@
```

5.1.1.2 输出

```
The result after simplify:
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
```

5.1.2 执行效果

```
请输入非终结符(以S开头):
SABCD
请输入终结符:
abcd
请输入产生式(以@结束):
示例: S --> a|aB|ε
应输入:
S a|aB|#
@
S a|bA|B|ccD
A abB|#
B aA
C ddC
D ddd
@
The result after simplify:
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
```

5.2 构造与下推自动机等价的上下文无关文法

5.2.1 测试用例

5.2.1.1 输入

```
2
q0
q1
2
a
b
2
A
z0
q0
z0
0
6
q0 a z0 q0 Az0
q0 a A q0 AA
q0 b A q1 #
q1 b A q1 #
q1 # A q1 #
q1 # z0 q1 #
```

5.2.1.2 输出

```
Non-terminals (N): S [q0,A,q1] [q0,z0,q1] [q1,A,q1] [q1,z0,q1]
Terminals (T): a b
Start Symbol (S): S
Productions (P):
S -> [q0,z0,q1]
[q0,A,q1] -> a[q0,A,q1][q1,A,q1]
[q0,A,q1] -> b
[q0,z0,q1] -> a[q0,A,q1][q1,z0,q1]
[q1,A,q1] -> epsilon
[q1,A,q1] -> b
[q1,z0,q1] -> epsilon
[q0,A,q1]renamed to:A
[q0,z0,q1]renamed to:B
[q1,A,q1]renamed to:C
[q1,z0,q1]renamed to:D
The result after simplify:
```

```
N={A, C, S}
T={a, b}
P:
        A --> aA | aAC | b
        C --> b
        S --> aA
S = S
```

5.2.2 执行效果

```
Please input the mode of the test:(1.Simplify CFG 2.PDA to CFG)
2
Enter the number of states: 2
Enter the states: q0
Enter the states: q1
Enter the number of terminals: 2
Enter the terminals: a
Enter the terminals: b
Enter the number of stack symbols: 2
Enter the stack symbols: A
Enter the stack symbols: z0
Enter the start state: q0
Enter the start stack symbol: z0
Enter the number of final states: 0
Enter the number of transitions(the function take # as epsilon): 6
Enter the transitions in the format (q, a, z) -> (newQ, str):
q0 a z0 q0 Az0
Enter the transitions in the format (q, a, z) -> (newQ, str):
q0 a A q0 AA
Enter the transitions in the format (q, a, z) -> (newQ, str):
q0 b A q1 #
Enter the transitions in the format (q, a, z) -> (newQ, str):
q1 b A q1 #
Enter the transitions in the format (q, a, z) -> (newQ, str):
q1 # A q1 #
Enter the transitions in the format (q, a, z) -> (newQ, str):
q1 # z0 q1 #

Non-terminals (N): S [q0,A,q1] [q0,z0,q1] [q1,A,q1] [q1,z0,q1]
Terminals (T): a b
Start Symbol (S): S
Productions (P):
S -> [q0,z0,q1]
[q0,A,q1] -> a[q0,A,q1][q1,A,q1]
[q0,A,q1] -> b
[q0,z0,q1] -> a[q0,A,q1][q1,z0,q1]
[q1,A,q1] -> epsilon
[q1,A,q1] -> b
[q1,z0,q1] -> epsilon
[q0,A,q1]renamed to:A
[q0,z0,q1]renamed to:B
[q1,A,q1]renamed to:C
[q1,z0,q1]renamed to:D
The result after simplify:
N={A, C, S}
T={a, b}
P:
        A --> aA | aAC | b
        C --> b
        S --> aA
S = S
```

6 实验小组成员分工

张晨阳：主要负责程序源代码的编写。

廖轩毅：主要负责对实验报告的整体整合和修改。

梁维熙：主要负责撰写构造与下推自动机等价的上下文无关文法部分的实验报告。

金建名：主要负责撰写设计上下文无关文法的变换算法部分的实验报告。

7 改进的思路和方法

7.1 设计上下文无关文法的变换算法

（1）可以完善文法等价性的判断，保证变换后文法的语言生成能力不变。

（2）可以增加其他文法变换，如消除左递归、标准化文法等。

（3）可以对输入的文法进行预处理，过滤无用的产生式和符号。

（4）可以增加可视化的输出，以图形化显示文法变换的过程。

（5）可以增加对更复杂文法的支持，目前程序仅支持简单的右线性文法。

（6）可以增加异常处理，以处理输入文法格式不正确的情况。

7.2 构造与下推自动机等价的上下文无关文法

7.2.1 现存问题

无法识别输入为单个或是大于等于三个栈中符号。由于生成非终结符集合的方法为全排列，其时间复杂度为 O(n!)，当输入为多个栈中符号时会导致生成时间的增长速率会超越指数级增长，所以在本次实验中为防止生成时间过长，我们限制了生成式生成的栈中符号数为 0 个或 2 个。

7.2.2 改进方法

全排列问题：目前只想到基于回溯法的全排列生成算法，实现难度较大且复杂度过高，后续考虑学习一些全排列问题的优化算法。