> 开发环境
> python == 3.9
> (需要准备 `graphviz` 库支持，并在操作系统安装https://graphviz.org/download/)

成员设定

姓名	学号	班级	分工
刘亮	2020211318	2020211322	顶层设计
王祥龙	2020211415	2020211322	可视化设计
马紫薇	2020211392	2020211322	代码实现
金耘石	2019211328	2020211322	代码实现

结构设计

按照类似于邻接矩阵的图存储方式，存储状态机，设计类如下。

（以比较抽象的封装，方便获取可视化属性的传递）

- 状态：因为存在“子集”，采用列表
- 转移函数：`transfer`
- 状态机：状态和转移函数的集合

```
class State(object):
    def __init__(self):
        self.state = []
    def __str__(self):
        return str(self.state)

class transfer(object):
    def __init__(self, start, end, cond):  # 这里传入State类型
        self.start = start
        self.end = end
        self.cond = cond
    def __str__(self):
        return str(self.start) + "---" + str(self.cond) + "-->" + str(self.end)

class FA(object):
    def __init__(self):
        self.states = []
        self.trans = []
        self.startstates = []
        self.endstates = []
    def addstate(self):
        pass
    def addtrans(self):#传入Transfer类型
```

```
        pass
    def addstartstate(self):
        pass
    def addendstate(self):
        pass
    def __str__(self):
        str = ""
        for i in self.trans:
            str += i.str + "\n"
        return str

```

设计思路

- 输入状态机：输入状态集合 -> 输入状态转移集合 -> 设置始末状态

> 可见，略为繁琐，选择从文件读取解析的方式

- 输出状态机：打印出所有转移函数。

具体实现

程序的算法：通过BFS控制连通分量，实现子集构造法。核心代码如下，完整代码请阅读文件。

```
def toDFA(nfa: FA) -> FA:
    # 初始状态同NFA，创建副本tmp，每次更新，直到其不再变化（已得到所有子集）
    index = 0
    while tmp != dfa.states:
        dfa.states = tmp.copy()
        state = tmp[index]
        # 对每个state，求对于所有输入符ipt的状态转移 nfa.δ(state,ipt)
        for ipt in dfa.inputs:
            end = []
            # nfa.δ(state,ipt) = state中每一个状态 element 的转移 δ(element,ipt)的并集
            for element in state:
                sub_end = [t.end for _, t in enumerate(nfa.trans)
                           if t.start == element and ipt == t.cond][0]
                for sub_element in sub_end:
                    # 不加入重复元素，同时将'NoS'替换为'Ø'
                    if sub_element not in end:
                        if sub_element == 'NoS':
                            end.append('Ø')
                        else:
                            end.append(sub_element)
            # 保证 state 内部按字典序排列，便于检验重复的state
            end = sorted(end)
            # 删除非空转移中的'Ø'
            while 'Ø' in end and len(end) != 1:
                del end[end.index('Ø')]
            # 加入DFA状态转移表。若转移后状态不为空且不存在于 tmp 中，将状态加入
            dfa.trans.append(Transfer(state, end, ipt))
            if end not in tmp and 'Ø' not in end:
                tmp.append(end)
        # 继续遍历 tmp
        index += 1
```

```
    # 将包含NFA终态的状态加入DFA的终态中
    for state in dfa.states:
        for end in nfa.endstates:
            if end in state and state not in dfa.endstates:
                dfa.endstates.append(state)
    return dfa
```

使用方式及效果

1. 按照格式编辑 `_nfa.in` 文件（与 `.py` 同目录）。文件格式如下:

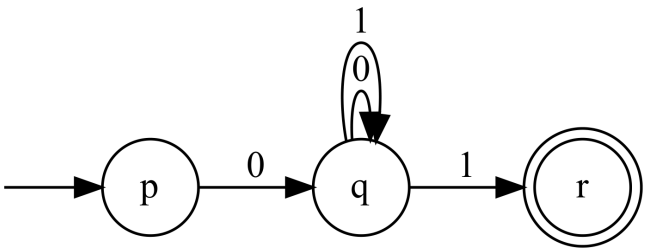
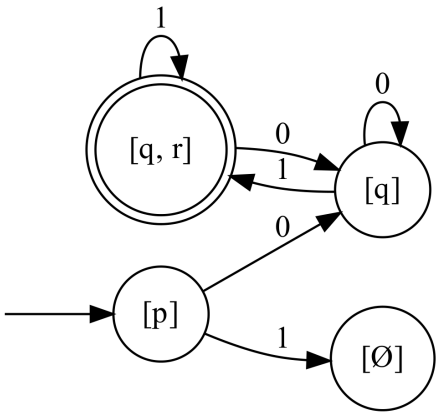
```
StartState: #初态
p
EndState:   #终态
r
States:     #所有非终结符，用空格分隔
p q r
Inputs:     #所有非终结符，用空格分隔
0 1
Transfer:   #状态转移表，格式为：
-           #各个状态以分隔符 "-"
p           #状态
0-q         #输入符1-转移状态1-转移状态2-...
1-NoS       #若转移状态为空，则为NoS(No State).不可省略
-
q
0-q
1-q-r
-
r
0-NoS
1-NoS
-           #分隔符 "-" 作为文件结尾
```

2. 在合适的 `python` 环境下运行 `main.py` ，如果正常会在命令行输出状态机信息、自动打开输入的NFA、输出的DFA的可视化PDF（对应运行路径下的 `NFA.gv.pdf` 、 `DFA.gv.pdf` ）。

示例结果

如上方的示例输入的NFA（同时也是老师PPT中的例子）：

NFA	DFA

NFA	DFA
	

命令行输出如下：

```
开始状态:
p
结束状态:
[[q, r]]
输入字符:
[0, 1]
状态转移:
[p]---0-->[q]
[p]---1-->[Ø]
[q]---0-->[q]
[q]---1-->[q, r]
[q, r]---0-->[q]
[q, r]---1-->[q, r]
```