实验报告

课程编号：3132112040　**实践课程名称：形式语言与自动机**　**学年：2021-2022**　**学期：春**

学生姓名	杜嘉骏	学号	2020212257
指导教师姓名	刘咏彬	起止时间	2022 年 4 月 29 日—5 月 2 日
项目名称	RL 模型转换工具		

项目内容

我要实现的是ε-NFA ->DFA 的转换工具，我使用的编程语言是 python。

1、设计方案：

只需要根据给定的ε-NFA 构造一个对应的 DFA 即可完成任务。

（1）首先考虑转换的算法。课本上介绍了ε-NFA 到 NFA 的构造方法和 NFA 到 DFA 的构造方法，如下图：

定理 3-1　NFA 与 DFA 等价。

证明：显然只需证明对于任给的 NFA，存在与之等价的 DFA。为此，设有 NFA：

$M_1=(Q,\Sigma,\delta_1,q_0,F_1)$

(1) 构造与 M_1 等价的 DFA M_2。

取 DFA

$M_2=(Q_2,\Sigma,\delta_2,[q_0],F_2)$

其中，

$Q_2=2^Q$

在表示状态集合时暂时将习惯的“{”和“}”改为用“[”和“]”，表示把集合的元素汇集成整体。

$F_2=\{[p_1,p_2,\cdots,p_m]\mid\{p_1,p_2,\cdots,p_m\}\subseteq Q \text{ 且 } \{p_1,p_2,\cdots,p_m\}\cap F_1\neq\varnothing\}$

对 $\forall\{q_1,q_2,\cdots,q_n\}\subseteq Q, a\in\Sigma,$

$\delta_2([q_1,q_2,\cdots,q_n],a)=[p_1,p_2,\cdots,p_m]\Leftrightarrow\delta_1(\{q_1,q_2,\cdots,q_n\},a)=\{p_1,p_2,\cdots,p_m\}$

定理 3-2　ε-NFA 与 NFA 等价。

证明：显然只需证明对于任给的 ε-NFA，存在与之等价的 NFA。为此，设有ε-NFA

$M_1=(Q,\Sigma,\delta_1,q_0,F)$

(1) 构造与之等价的 NFA M_2。

取 NFA

$M_2=(Q,\Sigma,\delta_2,q_0,F_2)$

其中，

$$F_2=\begin{cases}F\cup\{q_0\} & \text{如果 } F\cap\varepsilon\text{-}CLOSURE(q_0)\neq\varnothing\\ F & \text{如果 } F\cap\varepsilon\text{-}CLOSURE(q_0)=\varnothing\end{cases}$$

对 $\forall(q,a)\in Q\times\Sigma$，**使** $\delta_2(q,a)=\hat{\delta}_1(q,a)$。

可以将两个步骤结合起来：即只需要将开始状态设为 q0 的空转移集合，并且在每次状态转移后都考虑空转移。因为开始状态已经考虑空转移，并且每个状态在生成前也考虑了空转移，所以在考虑当前状态转移到下一个状态前不需要考虑空转移。这样依次将所有的到的状态集合都放到状态转移函数

中，就得到了 DFA。此处，可以发现，如果要将 NFA 转化为 DFA，只需要将输入文件的ε转移列都设为空集即可。

（2）其次要考虑 FA 如何存储的问题：FA 存储的内容有开始状态、结束状态、状态转移函数和字母表。字母表是固定的 0 和 1，所以不必考虑；状态转移函数要求根据当前状态和 input 可以得到下一个状态集合，这样就想到 dict 字典类型，使用当前状态和 input 组成的二元元组当作键，使用下一个状态作为值；同时使用排序后的 tuple 元组作为每个状态的存储方式，这样就变成可 hash 对象了；对于开始和结束状态来说，开始状态只有一个，就是 q0 的空转移集合，结束状态就是含有ε-NFA 结束状态的所有状态集合。

（3）将结果写入文件时，我对写的内容进行了整理，将列与列使用更大的空格隔开，这样更易于阅读。

2、环境配置：

Windows10、python 3.10.0

3、测试用例：

使用了两个测试用例，分别是课本 105 页的 15 题的（1）和（2）：

表 3-11　ε-NFA M_3 的状态转移函数

状态说明	状　态	输入字符		
		0	1	ε
开始状态	q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_2\}$
	q_1	$\{q_3, q_0\}$	$\varnothing$	$\{q_2\}$
	q_2	$\varnothing$	$\{q_3, q_1\}$	$\{q_2, q_1\}$
终止状态	q_3	$\{q_3, q_2\}$	$\{q_3\}$	$\{q_0\}$

表 3-12　ε-NFA M_4 的状态转移函数

状态说明	状　态	输入字符		
		0	1	ε
开始状态	q_0	$\{q_1, q_3\}$	$\{q_1\}$	$\{q_3, q_1\}$
	q_1	$\{q_2\}$	$\{q_2, q_1\}$	$\varnothing$
	q_2	$\{q_3, q_2\}$	$\{q_0\}$	$\{q_2, q_3\}$
终止状态	q_3	$\varnothing$	$\{q_0\}$	$\varnothing$

具体的输入在 codeANDexample 文件夹下的 example1.txt 和 example2.txt 中。

测试者在本机上只需要打开 code.py 的文件夹，不需要考虑解压路径问题，可以直接运行 py 文件，也可以在 cmd 终端运行。运行完毕可以在相同目录下得到两个 DFA 文件，如下图：

DFA1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

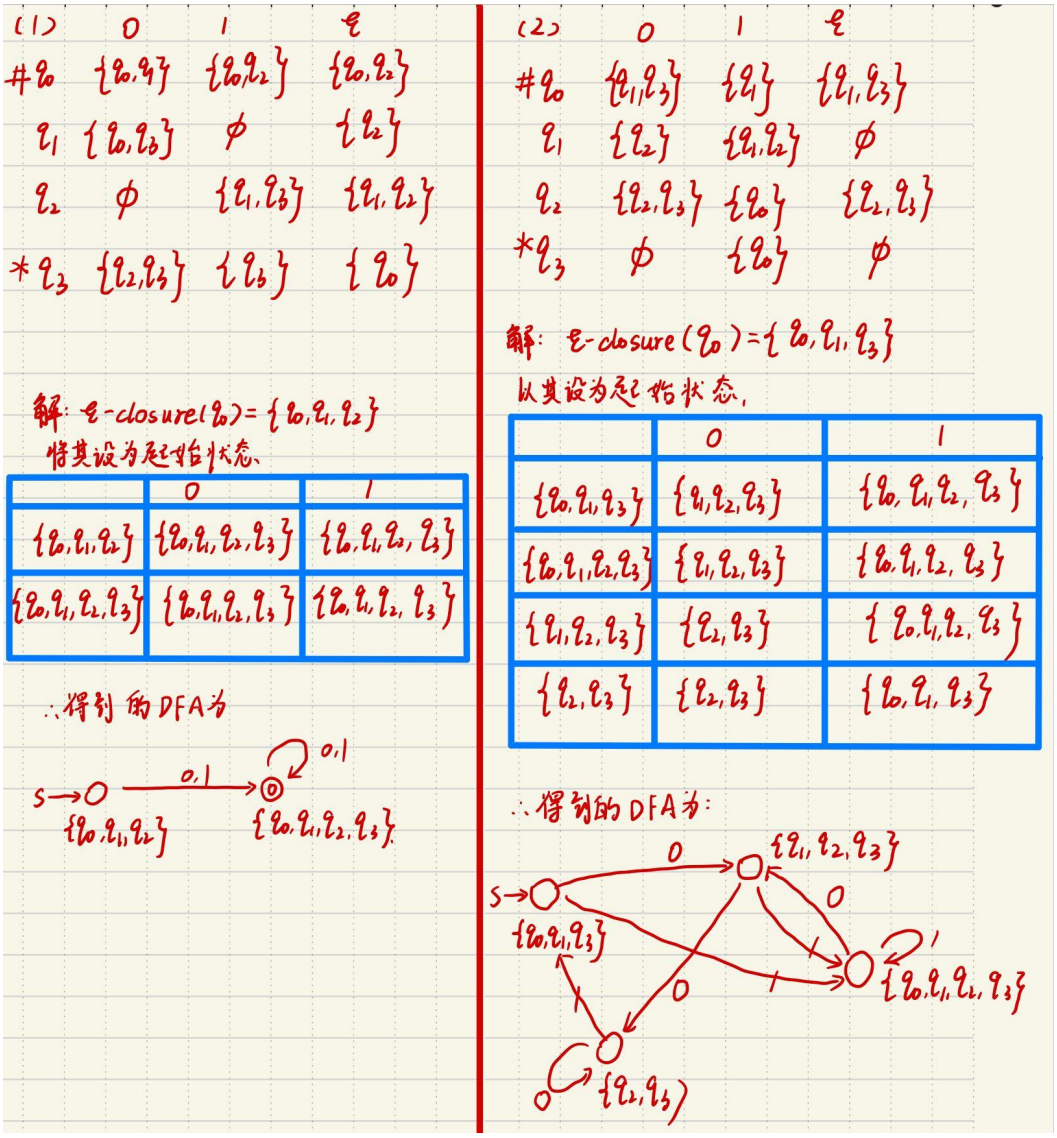
	0	1
#{0, 1, 2}	{0, 1, 2, 3}	{0, 1, 2, 3}
*{0, 1, 2, 3}	{0, 1, 2, 3}	{0, 1, 2, 3}

DFA2.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

	0	1
*{2, 3}	{2, 3}	{0, 1, 3}
*{0, 1, 2, 3}	{1, 2, 3}	{0, 1, 2, 3}
#*{0, 1, 3}	{1, 2, 3}	{0, 1, 2, 3}
*{1, 2, 3}	{2, 3}	{0, 1, 2, 3}

这个结果与人工计算得到的结果相同：



	由此，可以验证代码实现的正确性。
结论	本次实验中我实现了一个ε-NFA->DFA 的转换模型，在转化算法方面，主要使用了广度优先的原则，不断扩展状态集，不断将新的状态放入状态转移函数中，最终得到 DFA。在实现过程中，我加深了对 python 中各种数据结构的认识，让我学会了如何使用正确的结构存储正确的对象，我对ε-NFA 和 DFA 的认识有了更深入的理解。