

学生学号	0121710880415	实验课成绩	
------	---------------	-------	--

武汉理工大学

学 生 实 验 报 告 书

实验课程名称	编译原理
开 课 学 院	计算机科学与技术
指导教师姓名	王云华
学 生 姓 名	颜道江
学生专业班级	软件 1704

2019 -- 2030 学 年 第 一 学 期

实验课程名称： 编译原理

实验项目名称	词法分析			实验成绩	
实验者	颜道江	专业班级	软件 1704	组别	
同组者				实验日期	2019/11/30

一、词法分析——简化的 fortran 分析

1. 设计题目

简化的 fortran 语言词法分析程序设计。程序能够从用户输入的源程序中，识别出的单词符号，并用二元式表示，显示输出或输出到文件中。

2. 设计目的及设计要求

自己定义一个简单程序设计语言的单词集及其编码规则，并把词法分析器构造出来。保留字的识别按标识符一样识别，通过查找保留字表区分是保留字还是标识符。程序能够从用户输入的源程序中，识别出的单词符号，并用二元式表示，显示输出或输出到文件中。

3. 设计内容

3.1 词法分析设计

- 关键字：begin、if、then、while、do
- 运算符和界符： "+", "-", "*", "/", ":", ":", "<", ">", "<>", "<=", ">", ">=", "=", ":", "(", ")", "#";
- 标识符 ID 和整形常数 NUM

$$ID = \text{letter}(\text{letter}|\text{digit})^*$$

$$NUM = \text{digit}(\text{digit})^*$$
- 各种单词符号对用的种别编码如图 1 所示

单词符号	种别码	单词符号	种别码
begin	1	:	17
if	2	:=	18
then	3	<	20
while	4	<>	21
do	5	<=	22
end	6	>	23
letter(letter digit)*	10	>=	24
digitdigit*	11	=	25
+	13	;	26
-	14	(27
*	15)	28
/	16	#	0

图 1 单词符号与种别码

3.2 分析器构造

1. 从源代码文本中读入源代码，去掉每行末尾的换行符、空格符和制表符，得到源代码字符列表 `sourceCode`，后续分析基于此进行；
2. 首先根据输入的字符列表进行逐个字符的扫描判断，设计的算法如下：

Algorithm1 take(sourceCode)

Input: 源代码字符列表 `sourceCode`

1. 初始化指针, `num = 0`, 指向 `sourceCode` 第一个字符
 2. while (`num != len(sourceCode)`)
 - { 读取当前指向的字符 `str`;
 - if `str` is 字母:
 - 进入保留字与标识符识别程序;
 - 根据返回值设置二元组
 - else if `str` is 数字:
 - 进入数字识别程序;
 - 根据返回值设置二元组
 - else:
 - 进入运算符和界符识别程序;
 - 根据返回值设置二元组
 - }
-

3. 保留字与标识符分析:
保留字与标识符的识别算法如下:

Algorithm2 indetfier(sourcecode,s,num)

Input: `sourceCode,s,num`

Output: `s, num`

1. if `num == len(sourceCode)-1`
 - return `s, num`
 2. `curNum = num + 1, flag = True`
 3. while `flag`:
 - { if `sourceCode[curNum]`是数字 or `sourceCode[curNum]`是字母
 - `s = s + sourceCode[curNum]`
 - if `s` is 关键字:
 - `num = curNum = curNum+1`
 - return `s, num`
 - `curNum = curNum+1`
 - else:
 - `flag = False`
 - `num = curNum`
 - return `s, num`
 - }
-

4. 数字识别

数字识别流程图如下图 2:

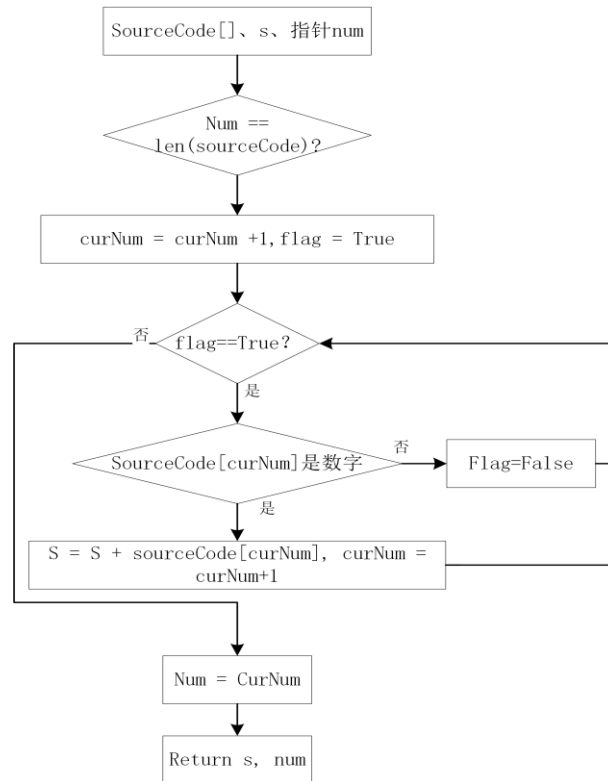


图 2 数字识别流程

5. 标识符识别

标识符的识别流程图如下图 3:

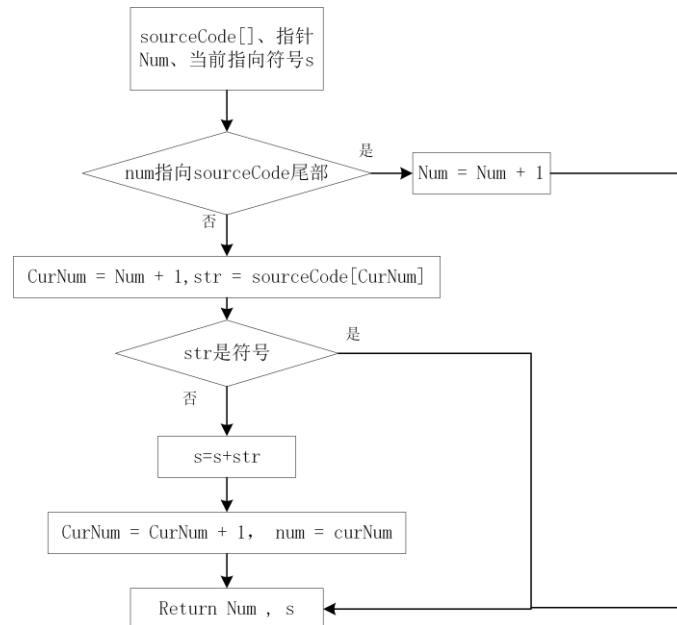


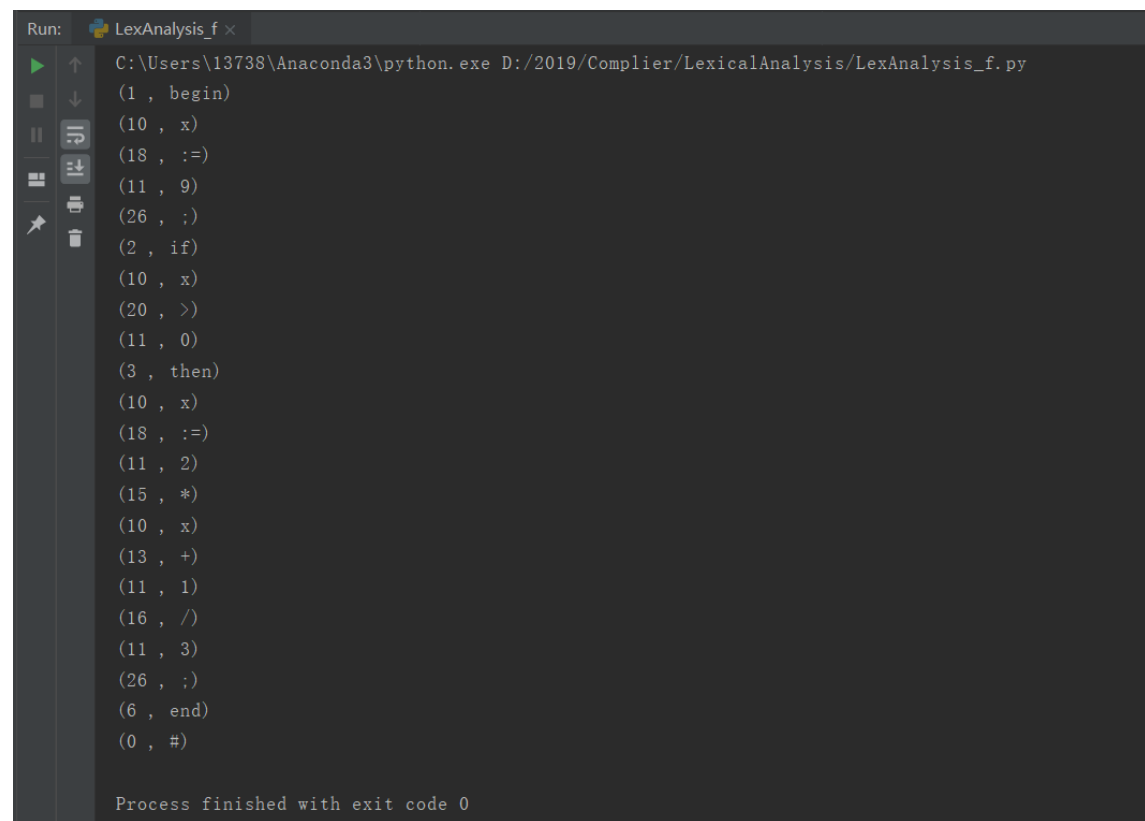
图 3 标识符识别流程

4. 输入输出设计

程序的从文本文件中读取进行输入，并通过附录的预处理程序删除换行符，同时程序的输出形式为二元式，直接将输出的内容写到控制台。

5. 运行结果

程序运行，分析得到的结果如下：



```
Run: LexAnalysis_f x
C:\Users\13738\Anaconda3\python.exe D:/2019/Compiler/LexicalAnalysis/LexAnalysis_f.py
(1 , begin)
(10 , x)
(18 , :=)
(11 , 9)
(26 , ;)
(2 , if)
(10 , x)
(20 , >)
(11 , 0)
(3 , then)
(10 , x)
(18 , :=)
(11 , 2)
(15 , *)
(10 , x)
(13 , +)
(11 , 1)
(16 , /)
(11 , 3)
(26 , ;)
(6 , end)
(0 , #)

Process finished with exit code 0
```

图 4 运行结果图

二、词法分析——NFA 确定化为 DFA

1. 设计题目

把 NFA 确定化为 DFA 的算法实现。在又穷自动机的理论中的定理：设 L 为一个由不确定的有穷自动机接受的集合，则存在一个接受 L 的确定的有穷自动机。本次的实验中对实现上述定理的算法——子集法进行设计实现。

2. 设计目的及设计要求

构造一程序，实现：将给定的 NFA M (其状态转换矩阵及初态、终态信息保存在指定文件中)，确定化为 DFA M 。

要先实现 ε -CLOSURE 函数和 I_a 函数。输出 DFA M' (其状态转换矩阵及初态、终态信息保存在指定文件中)。

3. 设计内容

3.1 ε -CLOSURE 函数设计

令 M 是一自动机， I 是 M 的状态子集，定义 ε -CLOSURE (I) 如下：

- (1) 若 $q \in I$ ，则 $q \in \varepsilon$ -CLOSURE(I)；
- (2) 若 $q \in I$ ，那么从 q 出发经任意条 ε 弧到达的状态都属于 ε -CLOSURE(I)。

根据上述的定义，设计求解算法如下：

Algorithm1 ε -CLOSURE

Input: 状态集 I ，词法规则 f

Output: ε -CLOSURE(I)

5. 初始化 I 中所有状态 i 的求解标志 Closure_flag[i] 为 False
 6. While(I 中存在状态 i 的求解标志 Closure_flag[i] 为 False) do
 - { if ((Closure_flag[i] == False) && 存在状态 i 经过 ε 弧到达的状态)
 - 遍历 f 找到所有 i 经过 ε 弧到达的状态, 并设求解标志为 False;
 - 更新当前求解标志 Closure_flag[i] == True
 - }
-

3.2 I_a 函数设计

状态结合 I 的 α 弧转换，表示为 $\text{move}(I, \alpha)$ ，定义为状态集合 J ，其中 J 是所有那些可从 I 中某一状态经过一条 α 弧到达的状态全体。其中我们有如下的关系：

$$I_a = \varepsilon - \text{Closure}(J)$$

Algorithm2 Move(I,arc),求解状态集 I 中转态经过一条 arc 弧度到达的边

Input: 状态集 I, 词法规则 f, 转换边 arc

Output: 求解得到的状态集 state

1. 初始化结果状态集 state
 2. for i in I:
 - { if (存在和 i 连接的 arc 弧)
遍历与 i 有弧相连的所有状态 new_state
if (相连的弧为 arc)
将 new_state 加入集合 state
-

3.3 状态子集算法设计

M 的状态集 S 由 K 的一些子集组成, 具体的子集构成算法 subset (f, E, K_0) 如下所示:

Algorithm3 Subset(f,E,K_0),构造 K 的子集

Input: 状态集 K, 词法规则 f, 初始状态 K_0

Output: DFA M 的规则和转换关系

1. 开始, 令 ϵ -CLOSURE(K_0)为 C 中唯一成员,状态为未标记的
 2. While(C 中存在尚未被标记的子集 T)do
 - { 标记 T;
for 每个输入字母 a do
 - {U:= ϵ -CLOSURE(Move(T,a))
if U 不在 C 中 then
将 U 作为未标记的子集加在 C 中
 - if (前一次循环 C 大小 == 后一次循环 C 大小)
break;
-

4. 输入输出设计

4.1 输入设计

本次实验中使用 Python 中的 json 包辅助进行输入输出的处理, 其中输入的 nfa.json 文件如图 5 所示。其中关于输入文件的详细说明如下:

1. K 是一个有穷集, 每个元素称为一个状态;
 2. E 是一个有穷字母表, 它的每个元素称为一个状态;
 3. f 转换规则;
 4. S 是一个非空初态集;
 5. Z 是一个终态集;
- 其中#用于表示 ϵ 。

```

{"K": ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"],
 "E": ["a", "b"],
 "S": "0",

 "f": {
   "0": {"#": ["1", "7"]},
   "1": {"#": ["2", "4"]},
   "2": {"a": ["3"]},
   "3": {"#": ["6"]},
   "4": {"b": ["5"]},
   "5": {"#": ["6"]},
   "6": {"#": ["1", "7"]},
   "7": {"a": ["8"]},
   "8": {"b": ["9"]},
   "9": {"b": ["10"]},
   "10": {}
 },
 "Z": ["10"]
}

```

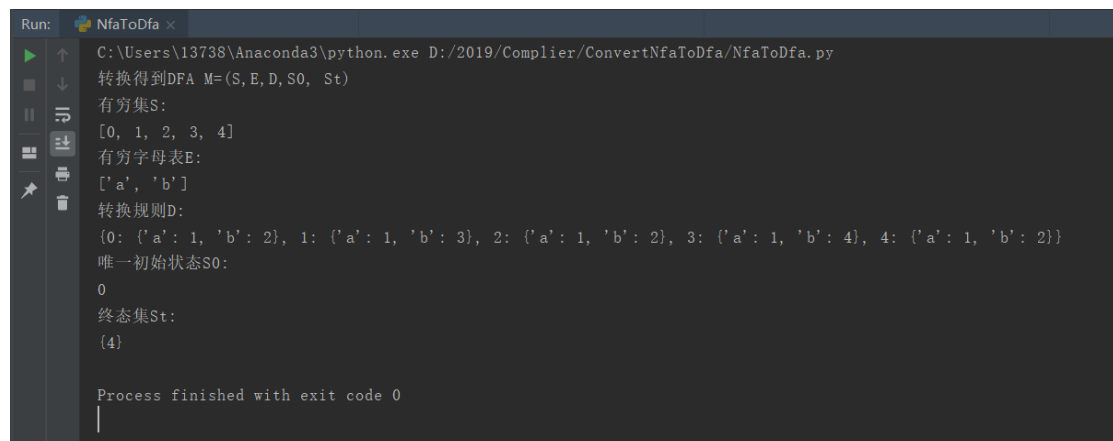
图 5 输入文件图

4.2 输出设计

与输入设计类似，程序的输出也同样采用的是写入 json 文件，同时 json 文件的结构和 NFA 类似，此处不进行详细的展示说明。

5. 运行结果

程序运行后将自动生成结果的 dfa.json 文件，同时在控制台进行显示，本次的实验中采用的是教材 50 所示的示例进行的验证，输出的结果如下图 6 所示：



```

Run: NfaToDfa x
C:\Users\13738\Anaconda3\python.exe D:/2019/Compiler/ConvertNfaToDfa/NfaToDfa.py
转换得到DFA M=(S, E, D, S0, St)
有穷集S:
[0, 1, 2, 3, 4]
有穷字母表E:
['a', 'b']
转换规则D:
{0: {'a': 1, 'b': 2}, 1: {'a': 1, 'b': 3}, 2: {'a': 1, 'b': 2}, 3: {'a': 1, 'b': 4}, 4: {'a': 1, 'b': 2}}
唯一初始状态S0:
0
终态集St:
{4}

Process finished with exit code 0

```

图 6 生成 DFA M 图

三、总结

本次的实验主要是分为两个小的任务，对于实验一是进行简化版的 Fortran 语言的词法分析，实验二是实现 NFA 到 DFA 的转化。

对于实验一，实验的算法思想比较的简单，主要就是逐个字符的进行扫描判断，直到扫描到文件的结尾。在扫描的过程中设计到符号，标识符和数字等的判断。

对于实验二，我花了比较多的时间并且查阅的相关的资料并借鉴了别人的一些好的设计才实现，在实验二的实现过程中主要涉及到了我们学习的各种算法的设计与实现，包括 ϵ -CLOSURE 函数设计，move 函数的设计实现以及状态子集的实现。

通过本次的实验主要有以下几个方面的收获：

1. 通过本次的实验将词法分析的整个流程进行了一遍梳理，由于我负责的是将 NFA 确定化，因此之前的各个算法也必须要实现才能达到最后的目的；
2. 通过本次的实现算法中的一些思想有了更加深刻的理解，特别是通过程序设计这个过程，比如在设计实现 ϵ -CLOSURE 的过程中会用到递归的思想等；
3. 通过本次的实验也算是对词法分析这部分的内容进行了一个详细的复习。

四、附录

1.2 词法分析——简化的 fortran 分析

1. 源代码输入预处理

```
1. def getSourceCode(path):
2.     """按行从文件中读取代码（忽略每行末尾的换行符），返回源代码"""
3.     line = ''
4.     for s in open(path):
5.         line = line + s.strip('\n')
6.     return line
7.
8.
9. def removeSpace(sourceCode):
10.    """去除源代码中的空格和制表符
11.    返回一个字符流形式的代码列表"""
12.    sourceCode = [' '.join([i.strip() for i in code.strip().split('\t')]) for c
13.                   ode in sourceCode] # 处理
14.    while ' ' in sourceCode:
15.        sourceCode.remove(' ')
16.    return sourceCode
```

2. 分析器构造

```
1. def take(sourceCode):
2.     """从字符流列表开始进行逐一分析"""
3.     num = 0
4.     while num != len(sourceCode):
5.         str = sourceCode[num]
6.         k = identifyChar(str)
7.         if k == 1:
8.             str1, num = identifier(sourceCode, str, num)
9.             if isKeyword(str1):
10.                 printf(str1, isKeyword(str1))
11.             else:
12.                 printf(str1, 10)
13.             continue
14.         if k == 2:
15.             str1, num = number(sourceCode, str, num)
16.             printf(str1, 11)
17.             continue
18.         if k == 3:
19.             str1, num = symbolStr(sourceCode, str, num)
```

```
20.         printf(str1, isSymbol(str1))
21.         continue
22.
23.
24. def idetifier(sourceCode, s, num):
25.     """标识符和保留字处理"""
26.     if num == len(sourceCode) - 1:
27.         return s, num + 1
28.     curNum = num + 1
29.     flag = True
30.     while flag:
31.         if isNum(sourceCode[curNum]) or isLetter(sourceCode[curNum]):
32.             s = s + sourceCode[curNum]
33.             if isKeyWord(s):
34.                 curNum = curNum + 1
35.                 num = curNum
36.                 return s, num
37.                 curNum = curNum + 1
38.         else:
39.             flag = False
40.             num = curNum
41.     return s, num
42.
43.
44. def symbolStr(sourceCode, s, num):
45.     """符号处理"""
46.     if num == len(sourceCode) - 1:
47.         return s, num + 1
48.
49.     curNum = num + 1
50.     str = sourceCode[curNum]
51.     if str in [ ">", "<", "=", ":" ]:
52.         s = s + sourceCode[curNum]
53.         curNum = curNum + 1
54.     num = curNum
55.     return s, num
56.
57.
58. def number(sourceCode, s, num):
59.     """对数字进行处理"""
60.     if num == len(sourceCode) - 1:
61.         return s, num + 1
62.     curNum = num + 1
63.     flag = True
```

```
64.     while flag:
65.         if isNum(sourceCode[curNum]):
66.             s = s + sourceCode[curNum]
67.             curNum = curNum + 1
68.         else:
69.             flag = False
70.     num = curNum
71.     return s, num
72.
73.
74. def isSymbol(s):
75.     """对运算符和界符进行判断, 如果是C子集中的运算符和界符返回相应的种别码"""
76.     symbol = ["+", "-", "*", "/", ":", ":", "<", ">", "<>", "<=", ">", ">=", "
77. =, ";", "(", ")", "#"]
78.     symbolNum = [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
79. , 0]
80.     if s in symbol:
81.         return symbolNum[symbol.index(s)]
82.     else:
83.         return 0
84.
85. def isNum(s):
86.     """判断是否是数字"""
87.     if s in '0123456789':
88.         return True
89.     else:
90.         return False
91.
92. def isLetter(s):
93.     """判断是否是字母"""
94.     letters = 'qwertyuiopasdfghjklzxcvbnm'
95.     if s in letters:
96.         return True
97.     else:
98.         return False
99.
100.
101. def isKeyWord(s):
102.     """判断是否是关键字, 如果是就返回种别码"""
103.     key = ["begin", "if", "then", "while", "do", "end"]
104.     keyNum = [1, 2, 3, 4, 5, 6]
105.     if s in key:
```

```

106.         return keyNum[key.index(s)]
107.     else:
108.         return 0
109.
110.
111. def identifyChar(c):
112.     """对单个字符进行识别"""
113.     if c in 'qwertyuiopasdfghjklzxcvbnm':
114.         return 1
115.     if c in '0123456789':
116.         return 2
117.     if c in ["+", "-", "*", "/", ":", ":", "<", ">", "<>", "<=", ">=",
        "=", ";", "(", ") ", "#"]:
118.         return 3

```

1.2 NFA 确定化为 DFA 核心代码

1. ϵ -CLOSURE (I) 求解核心代码

```

1. def epsilon_closure(f, I):
2.     """求状态集合 I: $\epsilon$ -closure(I)"""
3.     I = set(I)
4.     # 设置每个状态的标志
5.     closure_flag = dict()
6.     for i in I:
7.         closure_flag[i] = False # 初始状态设置为 False
8.     return opt_closure(f, closure_flag)
9.
10.
11. def opt_closure(f, closure_flag):
12.     """被  $\epsilon$ -closure(I)函数进行调用, 进行递归求解, 直到集合不再增大"""
13.     for i in list(closure_flag.keys()):
14.         if "#" in f[i].keys() and closure_flag[i] == False:
15.             for new_state in f[i]["#"]:
16.                 closure_flag[new_state] = False # 添加新的状态, 并将操作标志置为
                False
17.                 # 更改当前的标志位
18.                 closure_flag[i] = True
19.                 # 对新加入的状态进行递归求解
20.                 opt_closure(f, closure_flag)
21.     # 没有新的状态加入就返回
22.     return set(closure_flag.keys())

```

2. move(I, α)求解核心代码

```
1. def move(f, I, arc):
2.     """move 函数的实现, 从集合 I 中的某个状态出发, 经过一条 arc 弧到达的状态"""
3.     new_states = set() # 将转移状态集合初始化为空集
4.     for i in I:
5.         if arc in f[i].keys():
6.             for new_state in f[i][arc]:
7.                 new_states.add(new_state)
8.     return new_states
```

3. 子集求解算法

```
1. def subSet(f, E, K_0):
2.     """子集构造算法, 传入参数为 NFA 状态转换规则, 字母表和 NFA 初始状态"""
3.     T = {} # 保存构造出的所有子集, 键为子集的下标, 值为对应的状态集合
4.     flags = {} # 每个子集设置一个标志, 表明是否被标记
5.     relations = {} # 子集间的转换关系
6.     index = 0
7.
8.     # 开始, 令  $\epsilon$ -closure( $K_0$ ), 并且将其设置为未标记状态
9.     # 即求  $T_0$  并标记
10.    T[index] = epsilon_closure(f, K_0)
11.    flags[index] = False
12.
13.    while True:
14.        # 将子集状态族 C 初始化, 其中  $T_0$  为子集状态族唯一成员, 之后循环更新
15.        #  $C = (T_1, T_2, \dots)$  这里用列表存储下标, 下标作为字典的键可进行定位
16.        C = list(T.keys())
17.        beforeSize = len(C) # 通过子集状态族前后变化设置循环退出条件
18.        for i in C:
19.            if flags[i] == False:
20.                flags[i] = True # 标记 T
21.                # 构造两个状态之间的转换关系
22.                relations[i] = {}
23.                for arc in E:
24.                    U = epsilon_closure(f, move(f, T[i], arc))
25.
26.                # 产生的 U 不在子集状态族中
27.                if U not in T.values():
28.                    index += 1
29.                    T[index] = U
30.                    # 并将新状态的标记为设置为 False
31.                    flags[index] = False
```

```
32.             relations[i][arc] = index
33.             # 已经存在就添加转换关系
34.         else:
35.             # 字典中根据 value 获得 key （因为是一一对应的关系）
36.             relations[i][arc] = list(T.keys())[list(T.values()).index(
ex(U))]
37.         # 添加新的子集并更新 C
38.         C = list(T.keys())
39.         afterSize = len(C)
40.         # 判断子集构造是否结束
41.         if beforeSize == afterSize: # 已经没有新的状态需要加入
42.             break
43.
44.     return T, relations
```