词法分析实验报告

09021220 周天逸

实验目的

根据FA构造词法分析程序

1. 输入字符串
2. 输出对应的token序列

实验内容

本实验做了两个词法分析器，一个对应的是输出token序列，另一个判断是否是科学计数法

一、输出对应的token

1. 思路
    i. 定义REs
    ii. 将RE转换为NFA
    iii. 将NFA转化为最小状态的DFA
    iv. 基于DFA编程
2. 标识符划分
    i. 基本字

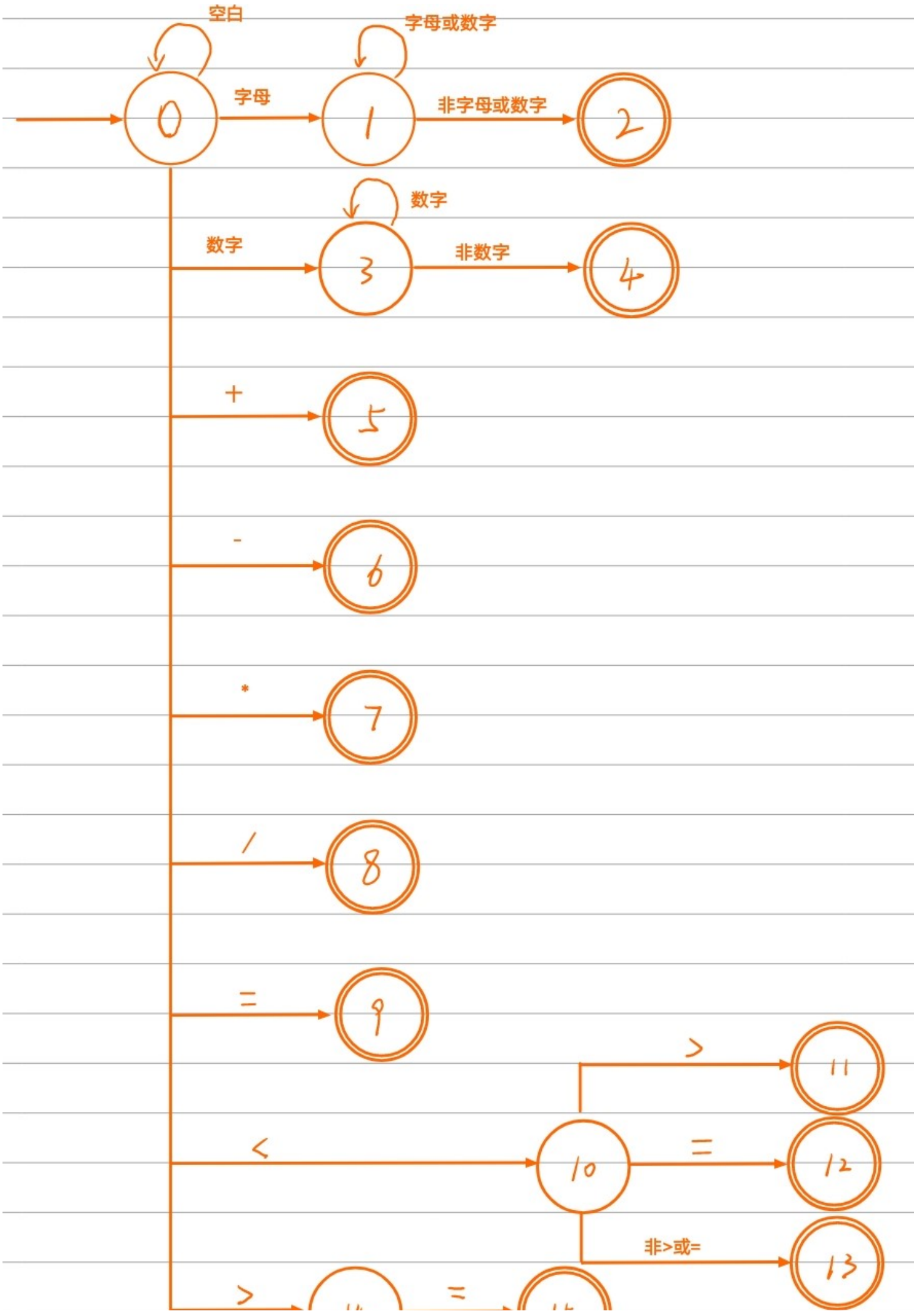
token	Value	RE
intsym	int	int
floatsym	float	float
doublesym	double	double
charsym	char	char
constsym	const	const
dosym	do	do
whilesym	while	while
ifsym	if	if

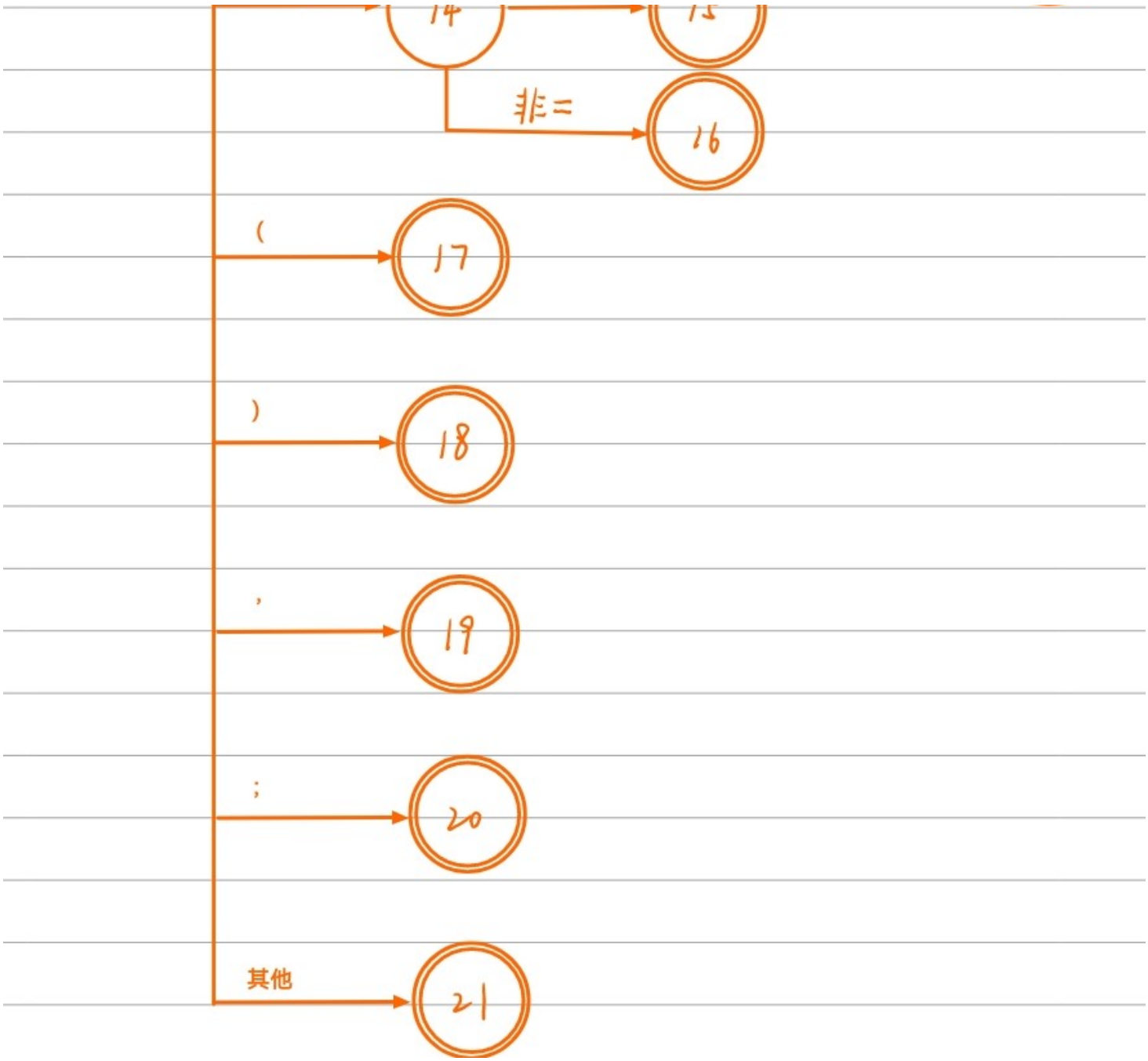
elsesym	else	else
switchsym	switch	switch
casesym	case	case
continuesym	continue	continue
breaksym	break	break
returnsym	return	return

ii. 标识符,运算符，常数

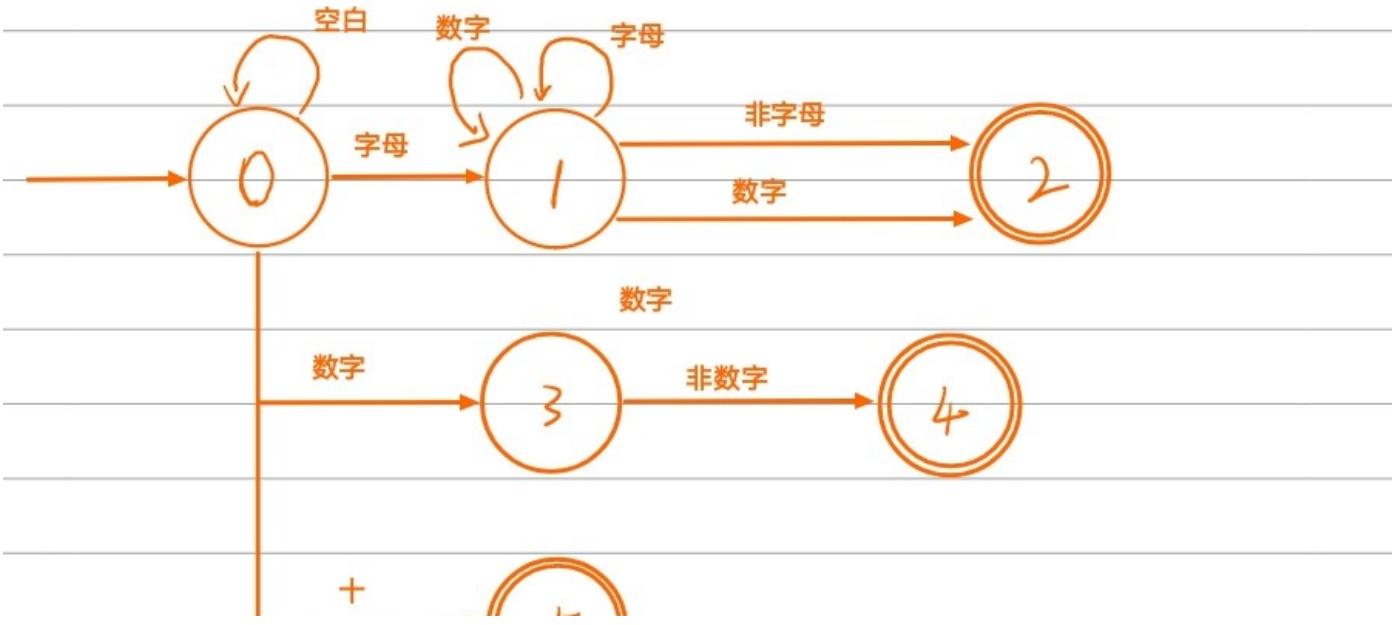
token	Value	RE
id	标识	(字母)(字母
num	常数	(数字)(数字)*
add	+	+
minus	-	-
times	*	*
divide	/	/
equal	=	=
less than	<	<
less equal	<=	<=
NE	<>	<>
greater than	>	>
greater equal	>=	>=
lparen	(	(
rparen	)	)
comma	,	,
semicolon	;	;

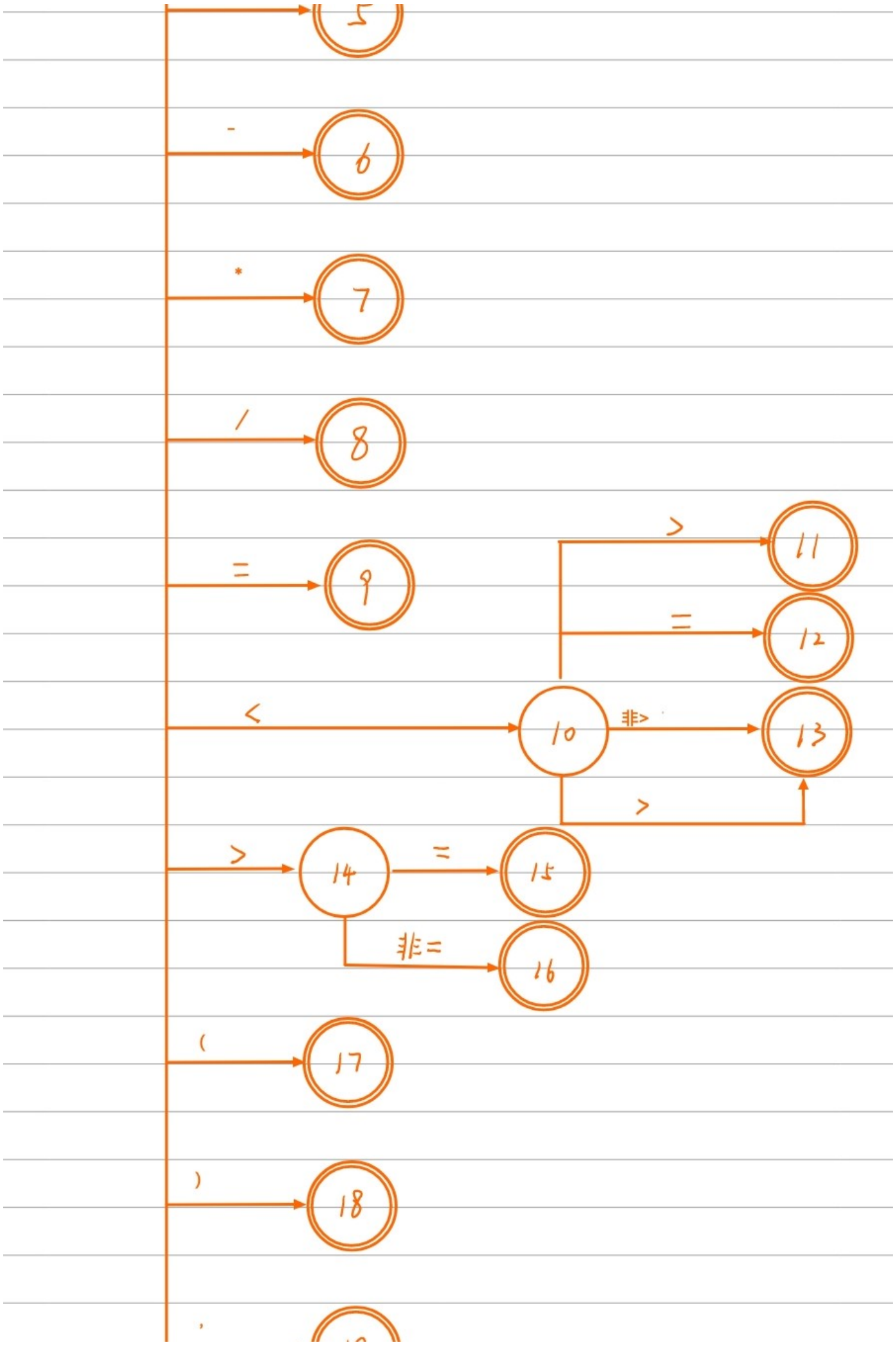
1. 构造NFA

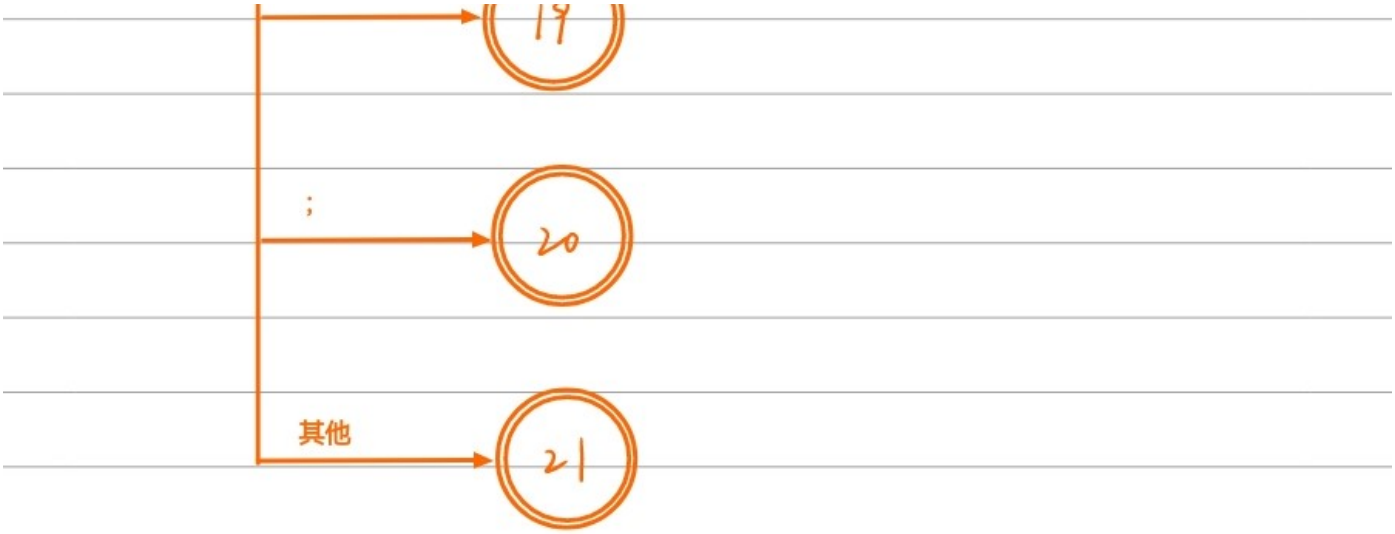

空白
字母或数字
0
字母
1
非字母或数字
2
数字
数字
3
非数字
4
+
5
-
6
*
7
/
8
=
9
>
11
<
10
=
12
非>或=
13
>
=



1. 优化为DFA




5
-
6
*
7
/
8
=
9
>
11
=
12
<
10
非>
13
>
>
14
=
15
非=
16
(
17
)
18



2. 重要数据结构描述
    i. 本词法分析器主要使用if-else和switch-case结构
3. 核心算法
    i. 读入字符串，遍历字符，判断是否结束(EOF)，是则结束；否则进入2）
    ii. 判断是否空格或换行，是则跳过，返回1）；否则根据DFA对字符类型进行判断并输出token序列，返回1）
4. 运行样例代码

```
int b = 1;
a = a + b;
a = a - b;
a = a * (b + a);
while(a >= b) b=b + a, break;
EOF
```

1. 运行结果

```
(intsym,int)
(id,b)
(equal,=)
(num,1)
(semicolon,;)
(id,a)
(equal,=)
(id,a)
(plus,+)
(id,b)
(semicolon,;)
(id,a)
(equal,=)
(id,a)
(minus,-)
```

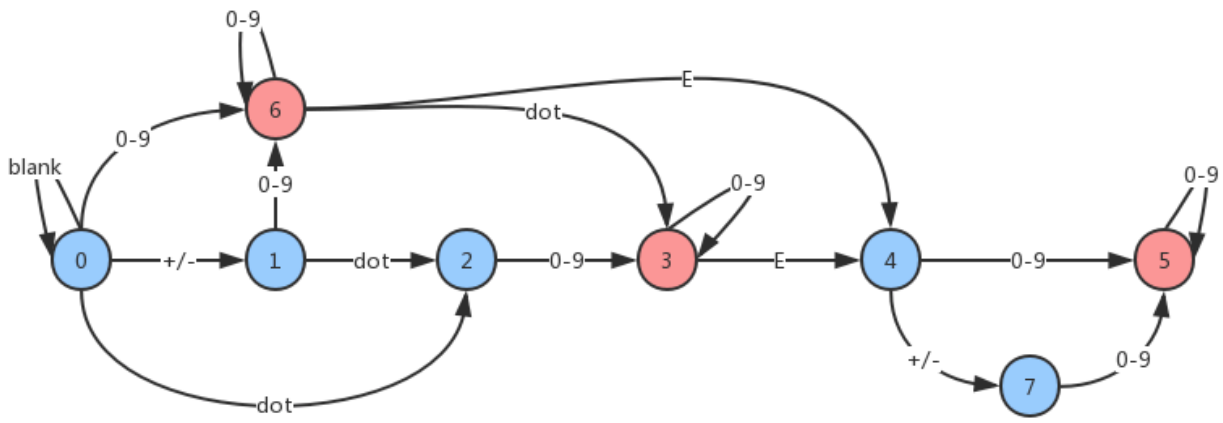
```
(id,b)
(semicolon,;)
(id,a)
(equal,=)
(id,a)
(times,*)
(lparen,()
(id,b)
(plus,+)
(id,a)
(rparen,))
(semicolon,;)
(whilesym,while)
(lparen,()
(id,a)
(greater equal,>=)
(id,b)
(rparen,))
(id,b)
(equal,=)
(id,b)
(plus,+)
(id,a)
(comma,,)
(breaksym,break)
(semicolon,;)
```

1. 实验分析
   i. 本次实验先通过确定要实现的REs，画出NFA，组合成一个大的NFA，确定化得到DFA，再最小化得到最小化DFA
   ii. 之后便根据DFA逐步开始编程，运用string库对输入字符串进行控制。每一个状态对应一个if-else语句或switch-case语句，结构比较清晰。

二 输入字符串判断是否满足科学计数法

1. 输入一个字符串，通过词法分析器判断是否满足一个正确的科学计数法要求
2. 一个 小数 或者 整数（可选）一个 'e' 或 'E' ，后面跟着一个 整数，小数（按顺序）可以分成以下几个部分：
   i. （可选）一个符号字符（'+' 或 '-'）
   ii. 下述格式之一：
      a. 至少一位数字，后面跟着一个点 '.'
      b. 至少一位数字，后面跟着一个点 '.' ，后面再跟着至少一位数字
      c. 一个点 '.' ，后面跟着至少一位数字

3. 状态图构建



4. 状态转移代码
   i. 我使用的是unordered_map这个数据结构，每一个状态都存放可以转移的到的状态。然后根据输入的字符来决定怎么转移
   ii. 在当前状态读入需要判断的字符后，map里没有找到对应的状态说明，字符串不是科学计数法的表示直接返回false。

```
unordered_map<State, unordered_map<CharType, State>> transfer{
            {
                STATE_INITIAL, {
                    {CHAR_NUMBER, STATE_INTEGER},
                    {CHAR_POINT, STATE_POINT_WITHOUT_INT},
                    {CHAR_SIGN, STATE_INT_SIGN}
                }
            }, {
                STATE_INT_SIGN, {
                    {CHAR_NUMBER, STATE_INTEGER},
                    {CHAR_POINT, STATE_POINT_WITHOUT_INT}
                }
            }, {
                STATE_INTEGER, {
                    {CHAR_NUMBER, STATE_INTEGER},
                    {CHAR_EXP, STATE_EXP},
                    {CHAR_POINT, STATE_POINT}
                }
            }, {
                STATE_POINT, {
                    {CHAR_NUMBER, STATE_FRACTION},
                    {CHAR_EXP, STATE_EXP}
                }
```

```
        }, {
            STATE_POINT_WITHOUT_INT, {
                {CHAR_NUMBER, STATE_FRACTION}
            }
        }, {
            STATE_FRACTION,
            {
                {CHAR_NUMBER, STATE_FRACTION},
                {CHAR_EXP, STATE_EXP}
            }
        }, {
            STATE_EXP,
            {
                {CHAR_NUMBER, STATE_EXP_NUMBER},
                {CHAR_SIGN, STATE_EXP_SIGN}
            }
        }, {
            STATE_EXP_SIGN, {
                {CHAR_NUMBER, STATE_EXP_NUMBER}
            }
        }, {
            STATE_EXP_NUMBER, {
                {CHAR_NUMBER, STATE_EXP_NUMBER}
            }
        }
    };
```