编译原理-Chapter2

2.1

a、 $C^*a(a|c)^*b(a|b|c)^*$

b、 $(b|c)^*(a(b|c)^*a)^*(b|c)^*$

c、 （这里限制接受字符串为非负二进制整数） $(\varepsilon|(0|1)^*0)\cdot 0$ 这里接受输入含前导0

或 $(\varepsilon|(1\cdot(0|1)^*\cdot 0))\cdot 0$ 这里不接受输入含前导0

d、（同c，输入非负）

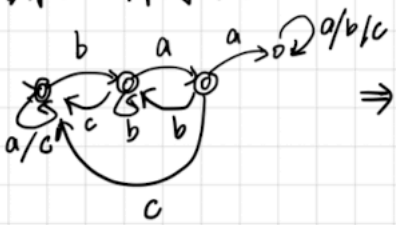
$$((0^*1\cdot 0^*)\cdot(0|1)\cdot(0|1)\cdot(0|1)\cdot(0|1)\cdot(0|1)\cdot(0|1)) \quad \geq 1000000$$

$$(0^*1)\cdot((1\cdot(0|1)\cdot(0|1)\cdot(0|1)\cdot(0|1)) \quad \geq 11000$$

$$(0\cdot 1\cdot((01|10|11)\cdot(0|1)) \quad \geq 1001$$

$$= 0^*1\cdot(((0^*(0|1)(0|1))|1)\cdot(0|1)\cdot(0|1)\cdot(0|1))|(0\cdot 1\cdot(01|10|11))(0|1)$$

e、不含有'baa'子串的DFA



$\Rightarrow (a|c)^*(bb^*(a|\varepsilon)(c(a|c)^*|\varepsilon))^*$

即 b^+

f、 $0([1-7][0-7]^*|\varepsilon)|[1-9][0-9]^*$

8进制或0　　10进制

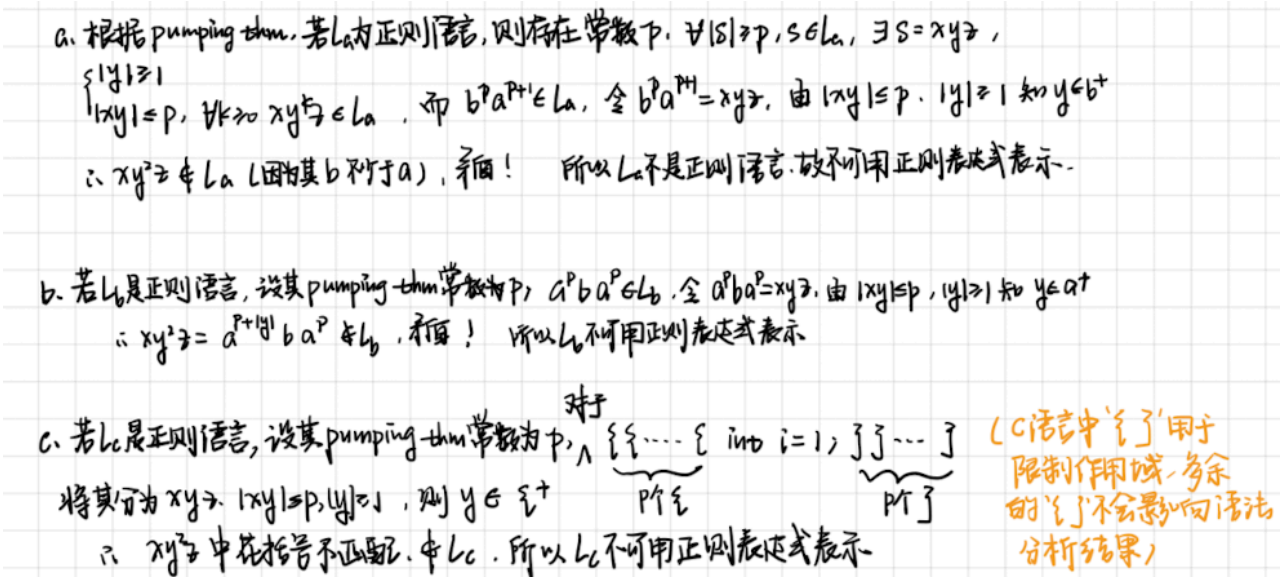
g、限制a,b,c≥1， $0^*1(0|\varepsilon)$ 即0或1

2.2

在《理论计算机科学导引》课程中，我学习到了正则语言所满足的泵引理的性质，由此，可以推出，有限自动机没有办法进行计数，（或者说，因为有限自动机的状态有限，而表示计数信息的状态却有无限个）。

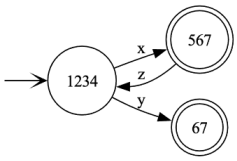
> 对于**a**，由于需要比较 **a**与 **b** 的个数，需要对 **a** 和 **b** 的数量进行计数，所以没有办法实现；对于**b**，因为回文串需要找到其中心位置才能进行左右比较，因此需要对回文串的长度进行模拟，无法实现；对于**c**，一个很常见的例子是合法的括号表达式的集合构成的语言是上下文无关语言，它不是正则语言，（有限自动机无法保存括号的个数信息），而括号表达式（带上变量和运算符）常常会出现在 **C** 语言的语法中，因此有限自动机无法实现。

以下为根据泵引理作出的严谨的证明：



2.5

1 a题



2 b题

首先研究转移：

- 状态1输入a，可以转移到状态1或状态2；输入b转移到状态1；
- 状态 i $(=2,3,4,5)$ 输入a或b，转移到状态 $i+1$。

根据「当前可能处于 NFA 上的状态的集合」作为 DFA 的状态，可以写出如下代码，进行求解转移（我们用一个二进制数表示一个状态集合，比如 $11=(1011)_2$ 表示可以到达 $1,2,4$ 集合），最后可能出现在 DFA 上的状态应该是状态 1 可以到达的状态，终态为包含了状态6的集合所处的状态：

```
#include <bits/stdc++.h>
using namespace std;
int G[1<<6][2];
bool vis[1<<6];
int mska(int i) {
    switch (i)
    {
    case 1:
        return 3;
        break;
    case 6:
        return 0;
        break;
    default:
        return (1<<i);
```

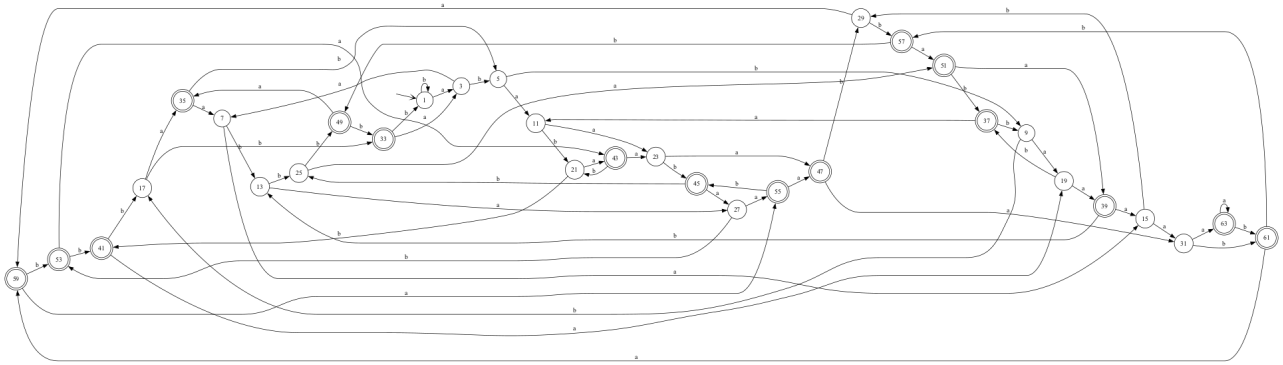
```
        break;
    }
}
int mskb(int i) {
    switch (i)
    {
    case 1: return 1;
        break;
    case 6:
        return 0;
        break;
    default:
        return (1<<i);
        break;
    }
}
void dfs(int x) {
    if(vis[x]) return ;
    vis[x] = 1;
    dfs(G[x][0]);
    dfs(G[x][1]);
}
string Tostring(int x) {
    return to_string(x);
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    for(int i = 1; i < (1<<6); ++i) {
        int _a = 0, _b = 0;
        for(int j = 1; j <= 6; ++j) {
            if(i >> (j-1) & 1) {
                _a |= mska(j);
                _b |= mskb(j);
            }
        }
        G[i][0] = _a; G[i][1] = _b;
    }
    dfs(1);
    cout << "states = {";
    for(int i = 1; i < (1<<6); ++i) if(vis[i]) {
        cout << "\'" << Tostring(i) << "\',";
    }
    cout << "}\n";
    cout << "transitions = {\n";
    for(int i = 1; i < (1<<6); ++i) if(vis[i]) {
        cout << "\'" << Tostring(i) << "\':{";
        cout << "\'a\': \'" << Tostring(G[i][0]) << "\', \'b\': \'" << Tostring(G[i][1])
<< '\'';
        cout << "},\n";
    }
    cout << "}\naccept_states = {";
    for(int i = 1; i < (1<<6); ++i) if(vis[i] && (i>>5&1)) {
        cout << "\'" << Tostring(i) << "\',";
    }
```

```
    cout << "}\nstart_state = '1'";
}
```

得到的输出如下：

```
states =
{'1','3','5','7','9','11','13','15','17','19','21','23','25','27','29','31','33','35','37'
,'39','41','43','45','47','49','51','53','55','57','59','61','63',}
transitions = {
'1':{'a': '3', 'b': '1'},
'3':{'a': '7', 'b': '5'},
'5':{'a': '11', 'b': '9'},
'7':{'a': '15', 'b': '13'},
'9':{'a': '19', 'b': '17'},
'11':{'a': '23', 'b': '21'},
'13':{'a': '27', 'b': '25'},
'15':{'a': '31', 'b': '29'},
'17':{'a': '35', 'b': '33'},
'19':{'a': '39', 'b': '37'},
'21':{'a': '43', 'b': '41'},
'23':{'a': '47', 'b': '45'},
'25':{'a': '51', 'b': '49'},
'27':{'a': '55', 'b': '53'},
'29':{'a': '59', 'b': '57'},
'31':{'a': '63', 'b': '61'},
'33':{'a': '3', 'b': '1'},
'35':{'a': '7', 'b': '5'},
'37':{'a': '11', 'b': '9'},
'39':{'a': '15', 'b': '13'},
'41':{'a': '19', 'b': '17'},
'43':{'a': '23', 'b': '21'},
'45':{'a': '27', 'b': '25'},
'47':{'a': '31', 'b': '29'},
'49':{'a': '35', 'b': '33'},
'51':{'a': '39', 'b': '37'},
'53':{'a': '43', 'b': '41'},
'55':{'a': '47', 'b': '45'},
'57':{'a': '51', 'b': '49'},
'59':{'a': '55', 'b': '53'},
'61':{'a': '59', 'b': '57'},
'63':{'a': '63', 'b': '61'},
}
accept_states =
{'33','35','37','39','41','43','45','47','49','51','53','55','57','59','61','63',}
start_state = '1'
```

使用 **python - graphvix** 库绘制 **DFA** 图如下：



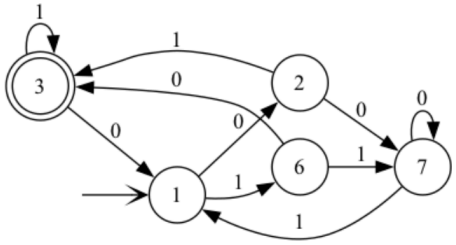
2.6

划分不等价状态：

- **3** 与 **1**， **2**， **4**， **5**， **6**， **7**， **8** 不等价；
- **2**，**8** 与 **1**，**4**，**5**，**6**，**7** 不等价（考虑**1**转移）
- **4**，**6** 与 **1**，**5**，**7** 不等价（考虑**0**转移）
- **1**，**5** 与 **7** 不等价（考虑**0**转移）
- **4** 冗余（没有状态可以转移到 **4**）

可以划分为 **{1,5}, {2,8} {3} {6} {7}**

合并等价的状态后：



可以验证此时没有等价状态，因此 **DFA** 已经最小化。