第三章作业

第一次作业:

练习 **3.2.1**，本题中我们把所有的符号归结为 **op**，实际当中是可以再细分的，同学们只要回答的有道理即可

3.1.1：根据3.1.2节中的讨论，将下面的C++程序

```
float  limiteSquare(x) { float x;
      /* returns x-squared, but never more than 100 */
      return (x<=-10.0 || x>=10.0) ? 100:x*x;
}
```

划分成正确的词素序列。那些词素应该有相关联的词法值？应该具有什么值？

<float, >
<id, 指向 limitedSquaare 符号表项指针>
<(, >
<id, 指向 x 符号表指针>
<), >
<{, >
<float, >
<id, 指向 x 符号表指针>
<return, >
<(, >
<id, 指向 x 符号表指针>
<op,"<=">
<num, -10.0>
<op, "||">
<id, 指向 x 符号表指针>
<op, ">=">
<num, 10.0>
<), >
<op, "?">
<num, 100>
<op, ":">
<id, 指向 x 符号表指针>
<op, "*">
<id, 指向 x 符号表指针>
<}, >

练习 3.3.2

3.3.2：是描述下列正则表达式定义的语言（仅1、2、5）

1)a(a|b)*a　　2) ((ε|a)b*) *　　5)(aa|bb) *((ab|ba)(aa|bb) *(ab|ba)(aa|bb) *) *

1)　a(a|b)*a

表示以 a 开头以 a 结尾且至少包含两个字符的由 a 和 b 构成的字符串的集合

2)　((ε|a)b*) *

表示由 a 和 b 构成的字符串的集合

5) (aa|bb) *((ab|ba)(aa|bb) *(ab|ba)(aa|bb) *) *
表示含有偶数个 a 和偶数个 b 的由 a 和 b 构成的字符串的集合

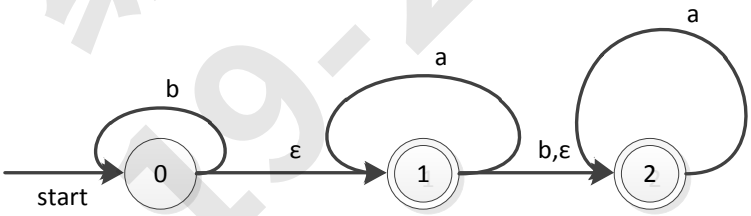
练习 3.3.5

3.3.5：试写出下列语言的正则定义（仅1、8、9）
1) 包含5个元音的所有小写字母串，这些串中的元音按顺序出现
8) 所有由a和b组成且不含子串abb的串
9) 所有由a和b组成且不含子序列abb的串

提示：8、9 可以先画出状态转换图，再根据状态转换图写正则表达式（1、8 的状态图见第二次作业）。8、9 注意“子串”跟“子序列”的区别。“子串”要求连续，“子序列”不要求连续，只需要保证单调性。e.g. “aabab”不包含子串 abb 但是包含子序列 abb
注意正则表达式中的几个特殊符号：
”*”表示出现任意次，包含 0 次，
”+”表示至少出现一次，
”?”表示至多出现一次
1) 包含 5 个元音的所有小写字母串，这些串中的元音按顺序出现
str -> other* a (other|a)* e (other|e)* i (other|i)* o (other|o)* u (other|u)*
other -> [bcdfghjklmnpqrstvwxyz]

8) 所有由 a 和 b 组成且不含子串 abb 的串
str ->b*(a|ab)* 或者 str ->b*(a+b?)*

9) 所有由 a 和 b 组成且不含子序列 abb 的串
ANS:思路如下，不包含子序列 abb，可以尝试着枚举几种典型的情况，状态图如下：



b*a*(b|ε)a* 和 b*a*(a|b)a*|ε

即
str ->b*| b*a+ | b*a+ba*

第二次作业:

1.D FA/NFA 设计

为下面的语言设计一个DFA或NFA

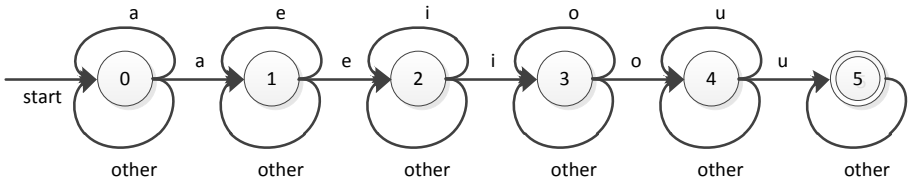
1) 包含5个元音的所有小写字母串，这些串中的元音按顺序出现

2) 所有由a和b组成且不含子串abb的串

为下面的语言设计一个 DFA 或 NFA。

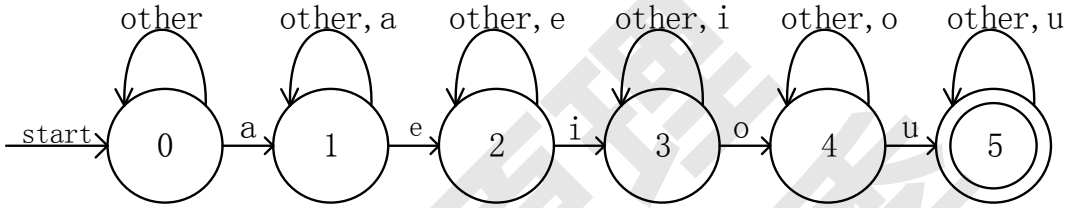
1) 包含 5 个元音的所有小写字母串，这些串中的元音按顺序出现

NFA 的一种:



上图中 other 表示除去元音字母以外剩下的 21 个字母。

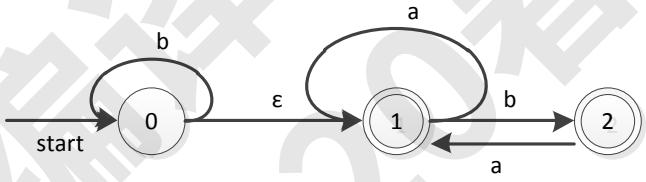
DFA 的一种：



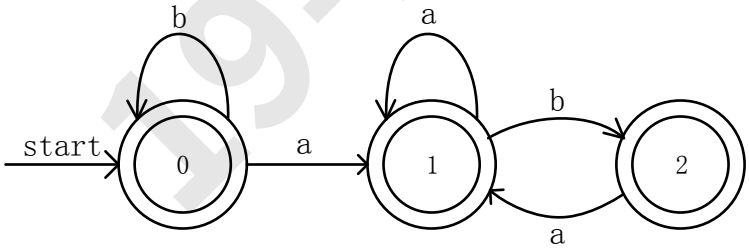
上图中 other 表示除去元音字母以外剩下的 21 个字母。

2) 所有由 a 和 b 组成且不含子串 abb 的串

ANS:枚举几种典型的情况，不难画出如下 NFA：



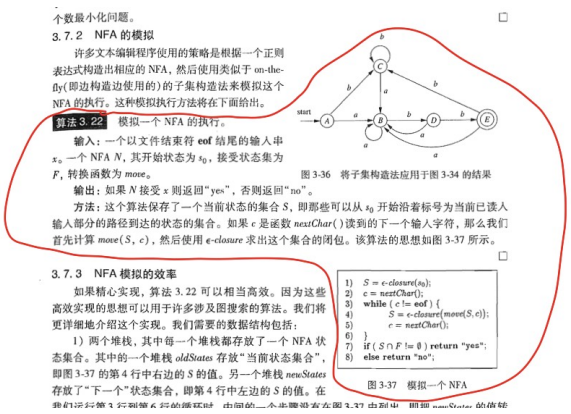
等价的 DFA 如下图所示。



2.NFA 模拟输入

用算法3.22模拟图3-29中的NFA在处理输入aabb时的过程

用算法 3.22 模拟图 3-29 中的 NFA 在处理输入 aabb 时的过程。(算法 3.22 见龙书第三版 P99)



个数最小化问题。

3.7.2 NFA 的模拟

许多文本编辑程序使用的策略是根据一个正则表达式构造出相应的 NFA，然后使用类似于 on-the-fly(即边构造边使用的)的子集构造法来模拟这个 NFA 的执行。这种模拟执行方法将在下面给出。

算法 3.22 模拟一个 NFA 的执行。

输入：一个以文件结束符 **eof** 结尾的输入串 x。一个 NFA N，其开始状态为 s_0，接受状态集为 F，转换函数为 *move*。

输出：如果 N 接受 x 则返回“yes”，否则返回“no”。

方法：这个算法保存了一个当前状态的集合 S，即那些可以从 s_0 开始沿着标号为当前已读入输入部分的路径到达的状态的集合。如果 c 是函数 *nextChar*( ) 读到的下一个输入字符，那么我们首先计算 *move*(S, c)，然后使用 *e-closure* 求出这个集合的闭包。该算法的思想如图 3-37 所示。

图 3-36 将子集构造法应用于图 3-34 的结果

3.7.3 NFA 模拟的效率

如果精心实现，算法 3.22 可以相当高效。因为这些高效实现的思想可以用于许多涉及图搜索的算法。我们将更详细地介绍这个实现。我们需要的数据结构包括：

1）两个堆栈，其中每一个堆栈都存放了一个 NFA 状态集合。其中的一个堆栈 *oldStates* 存放“当前状态集合”，即图 3-37 的第 4 行中右边的 S 的值。另一个堆栈 *newStates* 存放了“下一个”状态集合，即第 4 行中左边的 S 的值。在

```
1)  S = e-closure(s0);
2)  c = nextChar();
3)  while ( c != eof ) {
4)      S = e-closure(move(S, c));
5)      c = nextChar();
6)  }
7)  if ( S ∩ F != ∅ ) return "yes";
8)  else return "no";
```

图 3-37 模拟一个 NFA

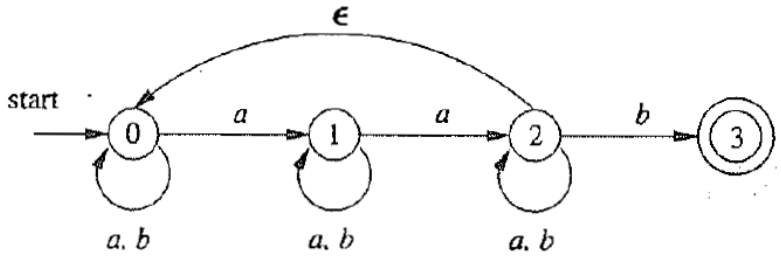


图 3-29　练习 3.6.3 的 NFA

ANS:

F={3},S=ε-closure(0)={0},c='a'

S=ε-closure(move({0},'a'))={0,1},c='a'

S=ε-closure(move({0,1},'a'))={0,1,2},c='b'

S=ε-closure(move({0,1,2},'b'))={0,1,2,3},c='b'

S=ε-closure(move({0,1,2,3},'b'))={0,1,2,3},c=eof

S∩F != null，∴return "yes"

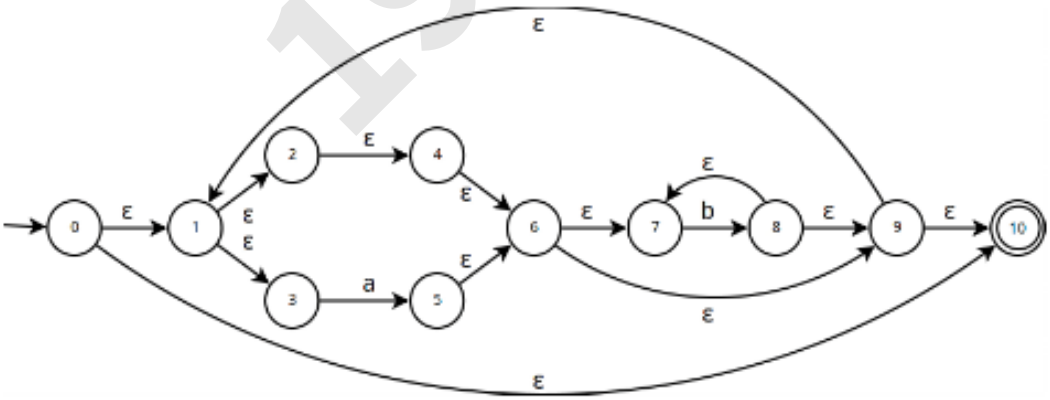
3.DFA 化简

使用算法 3.23 和 3.20 将下述正则表达式转换为 DFA，并尝试化简该 DFA

1)　((ε | a)b*) *

ANS:

NFA 如下：

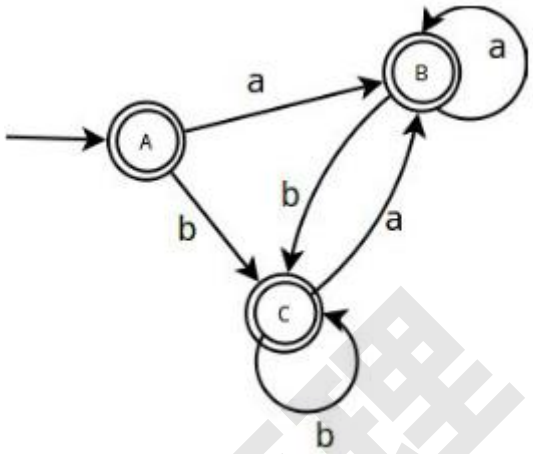


转换表如下：

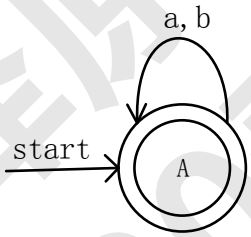
NFA 状态	DFA 状态	输入符号	
		a	b

{0,1,2,3,4,6,7,9,10}	A	B	C
{1,2,3,4,5,6,7,9,10}	B	B	C
{1,2,3,4,6,7,8,9,10}	C	B	C

转换后的 DFA 如下：



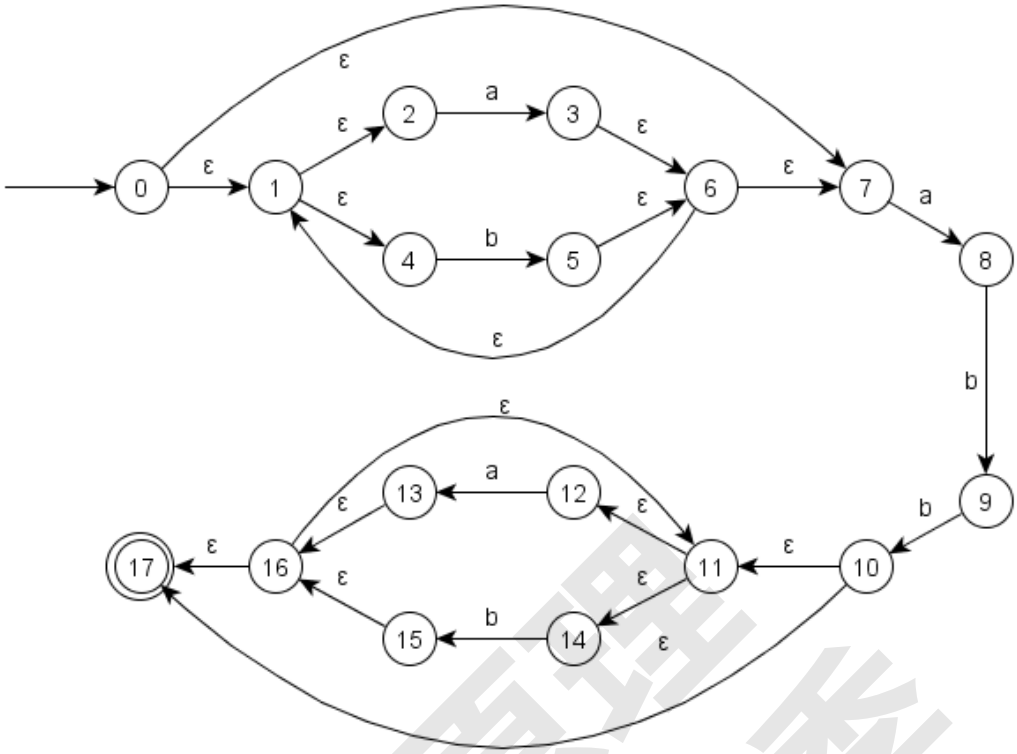
经过化简，最终可得化简后 DFA 如下图所示。



2) (a | b) *abb(a | b) *

ANS:

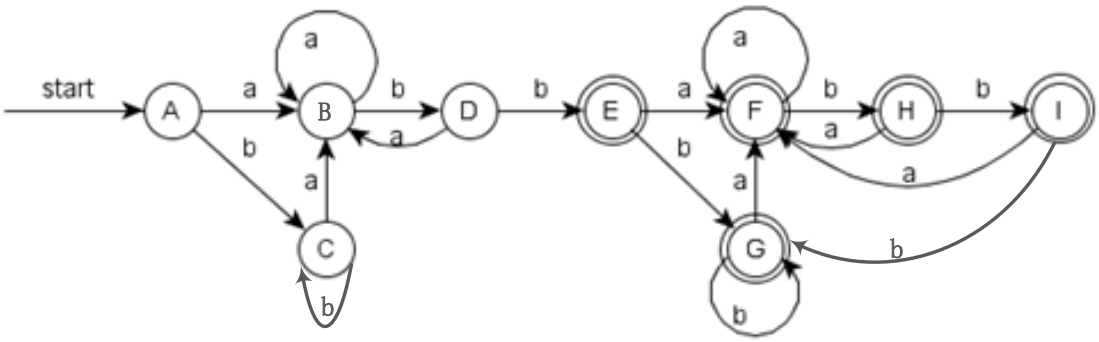
NFA 如下：



转换表如下：

NFA 状态	DFA 状态	输入符号	
		a	**b**
{0,1,2,4,7}	A	B	C
{1,2,3,4,6,7,8}	B	B	D
{1,2,4,5,6,7}	C	B	C
{1,2,4,5,6,7,9}	D	B	E
{1,2,4,5,6,7,10,11,12,14,17}	E	F	G
{1,2,3,4,6,7,8,11,12,13,14,16,17}	F	F	H
{1,2,4,5,6,7,11,12,14,15,16,17}	G	F	G
{1,2,4,5,6,7,9,11,12,14,15,16,17}	H	F	I
{1,2,4,5,6,7,10,11,12,14,15,16,17}	I	F	G

DFA 如下：



经过化简，最终可得化简后 DFA 如下图所示。