1 测试1

选择正则表达式 `a(a|b)*` 作为测试用例测试结果如下：

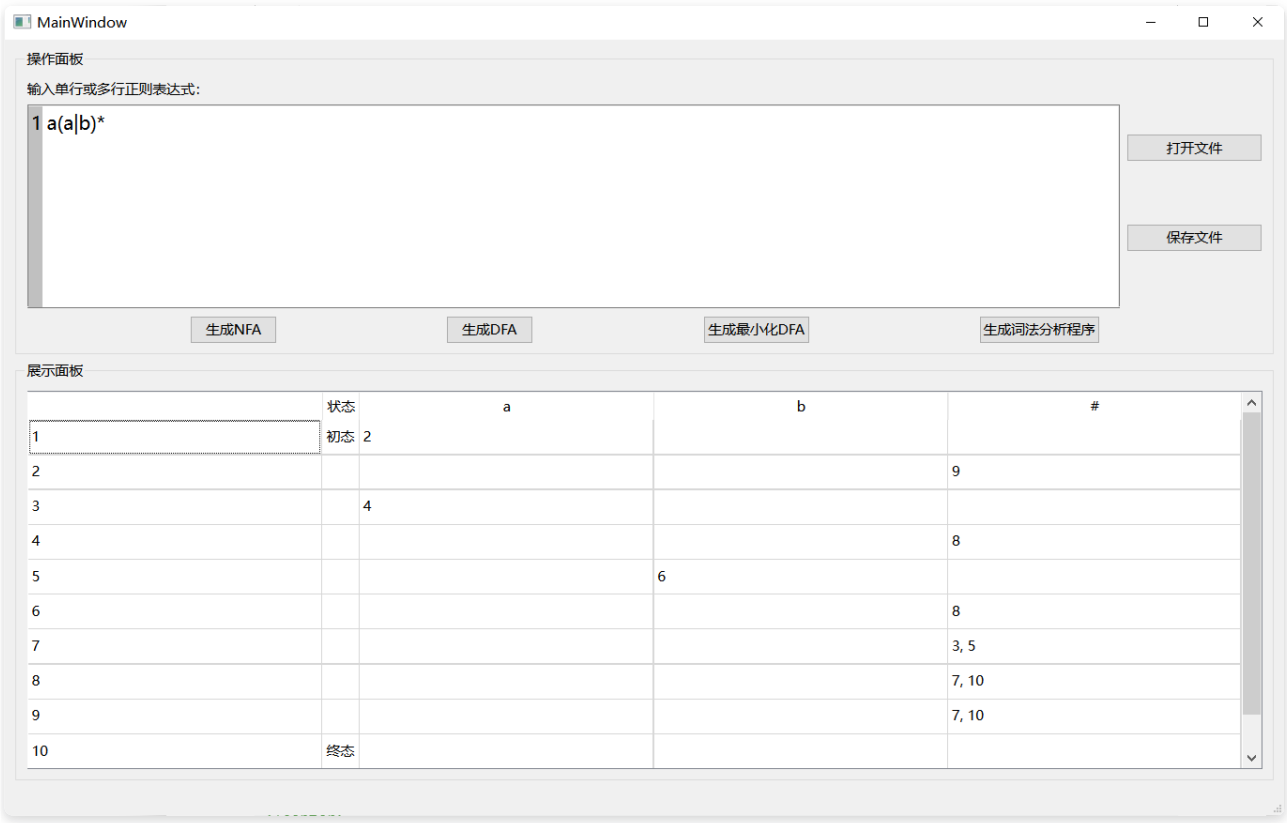


图1. 正则表达式转NFA

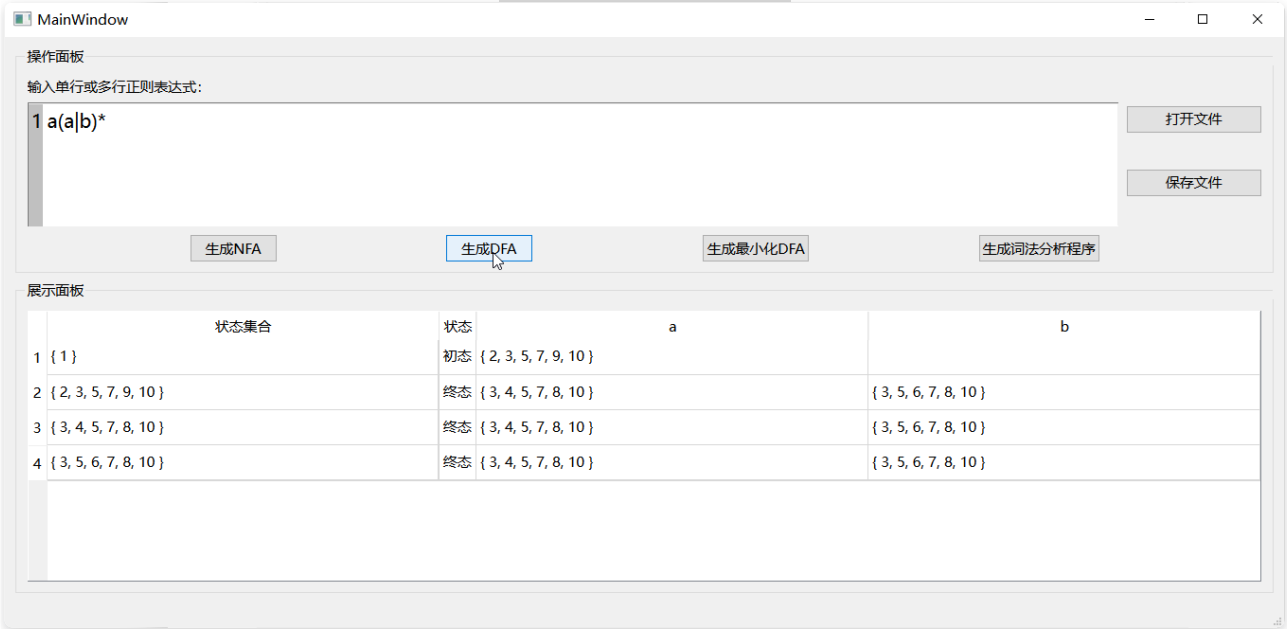


图2. NFA转DFA

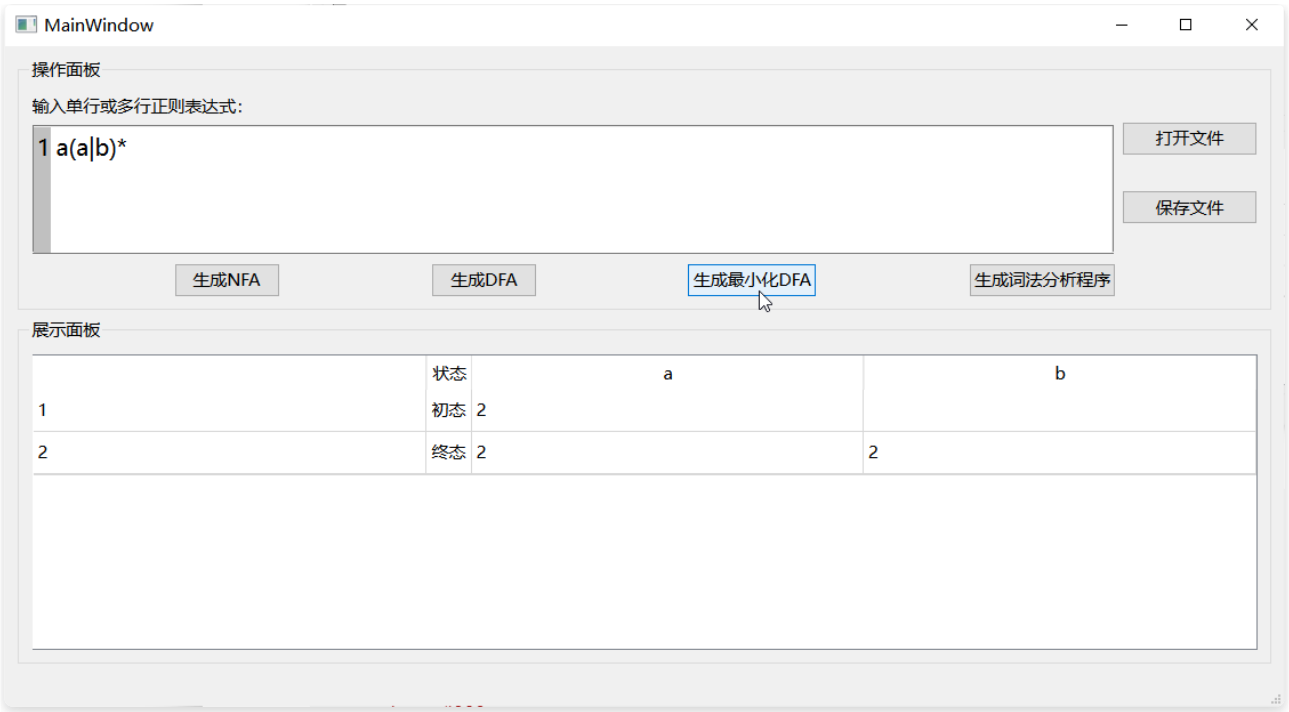


图3. 最小化DFA

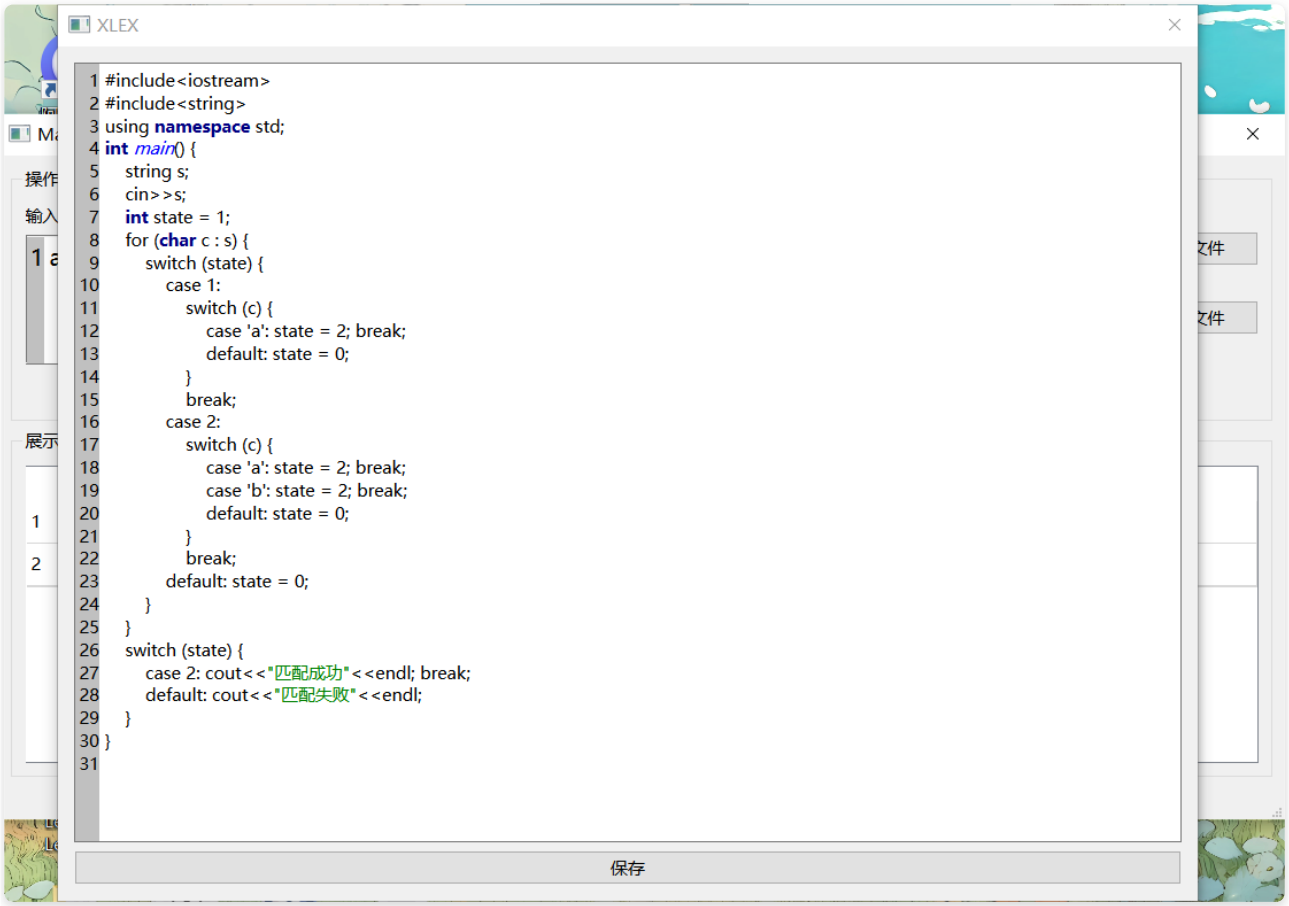


图4. 生成词法分析程序

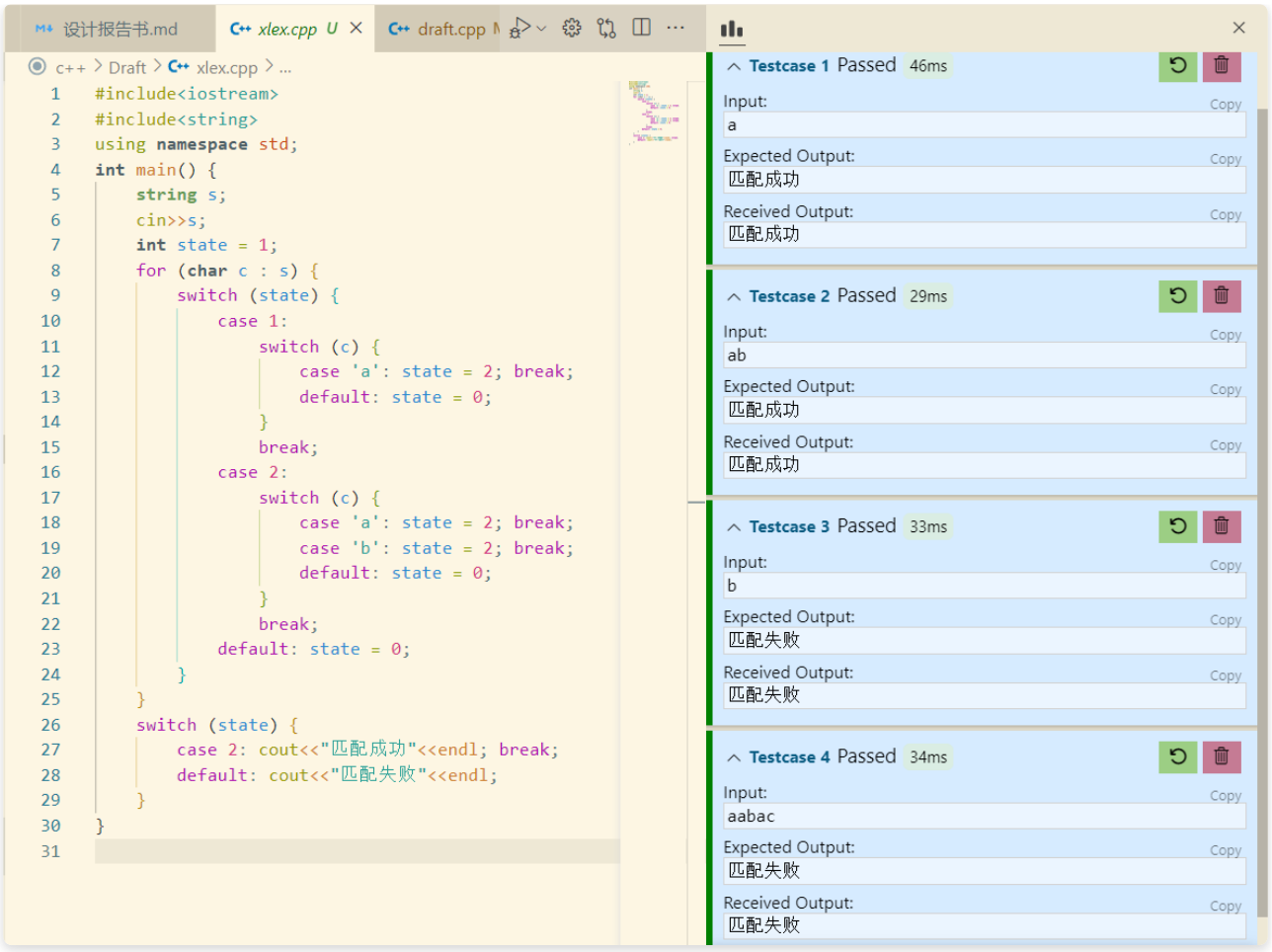


图5. 测试生成的词法分析程序

可看出该程序能生成正确的对正则表达式 `a(a|b)*` 的词法分析程序

2 测试2

选择正则表达式 `(a|b)*(aaa)(a|b)*` 作为测试用例测试结果如下：

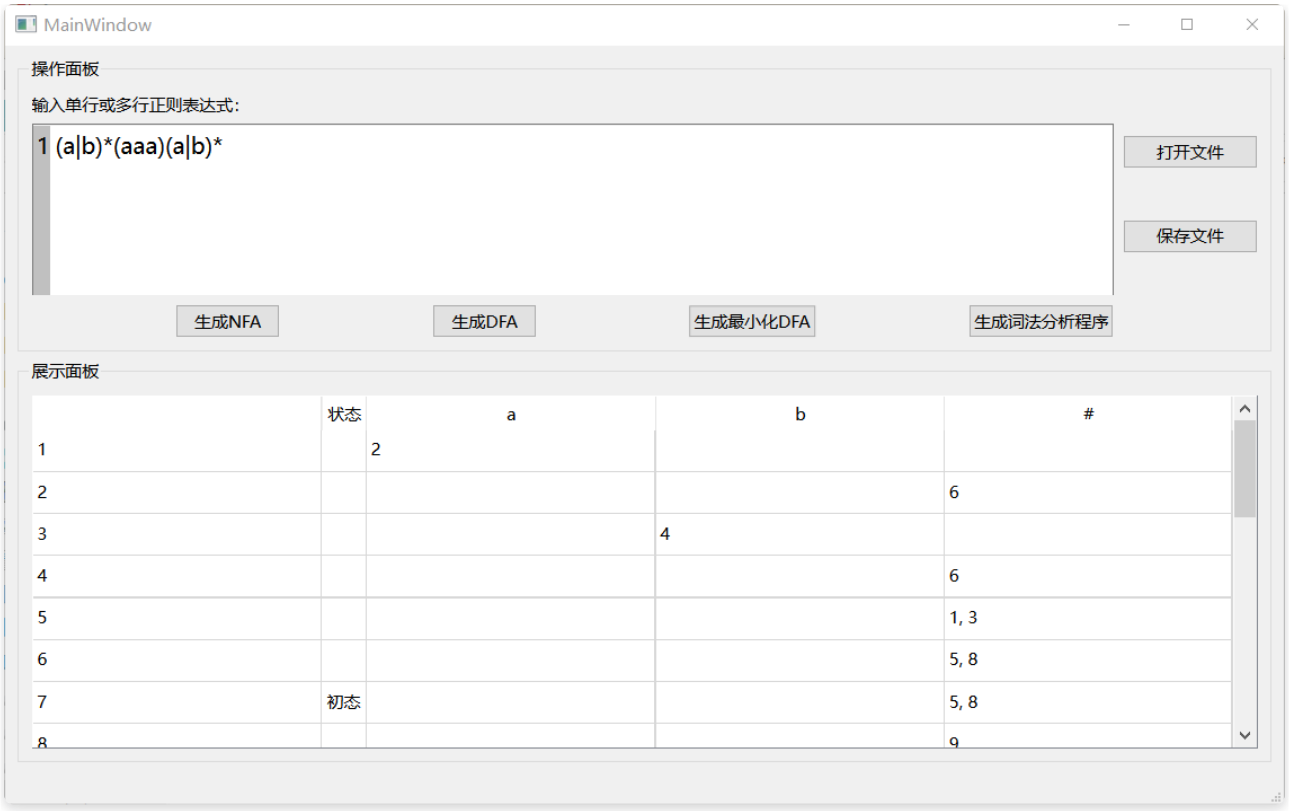


图6(1). 正则表达式转NFA(1)

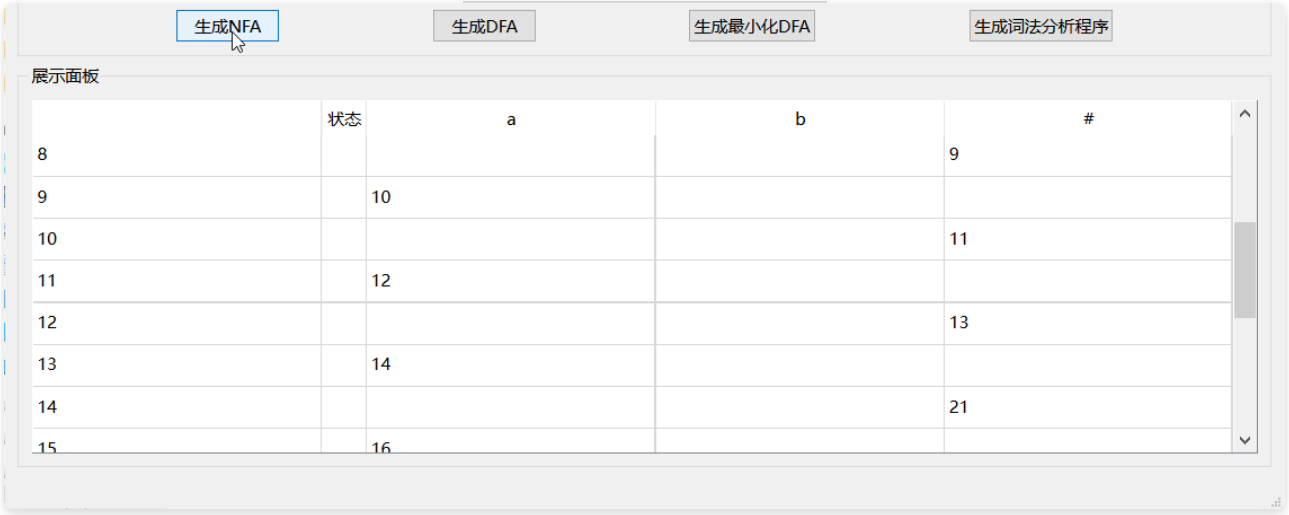


图6(2). 正则表达式转NFA(2)

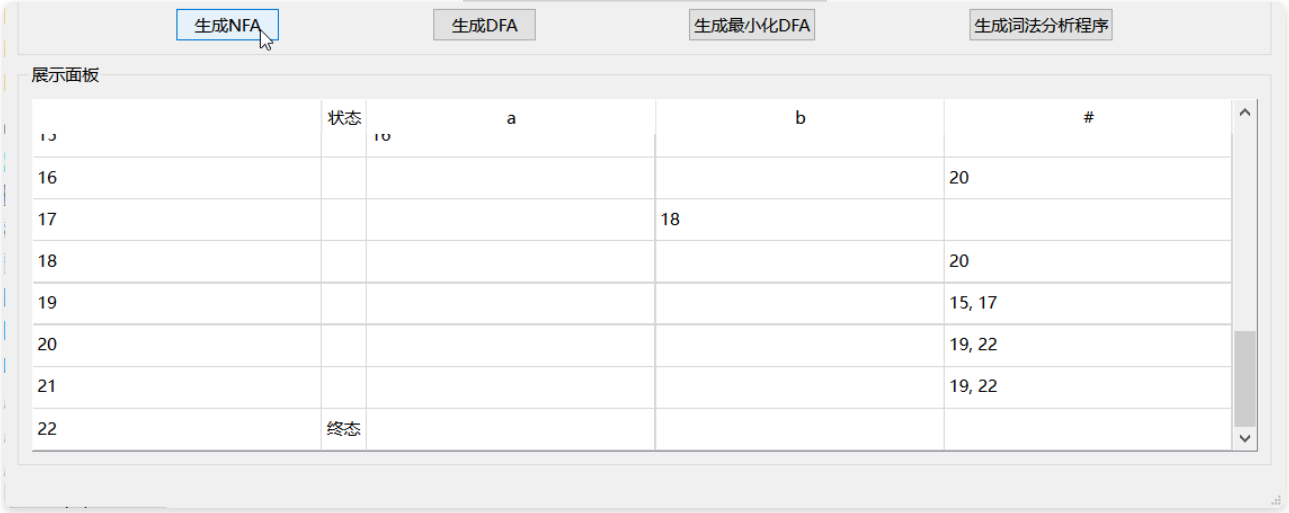
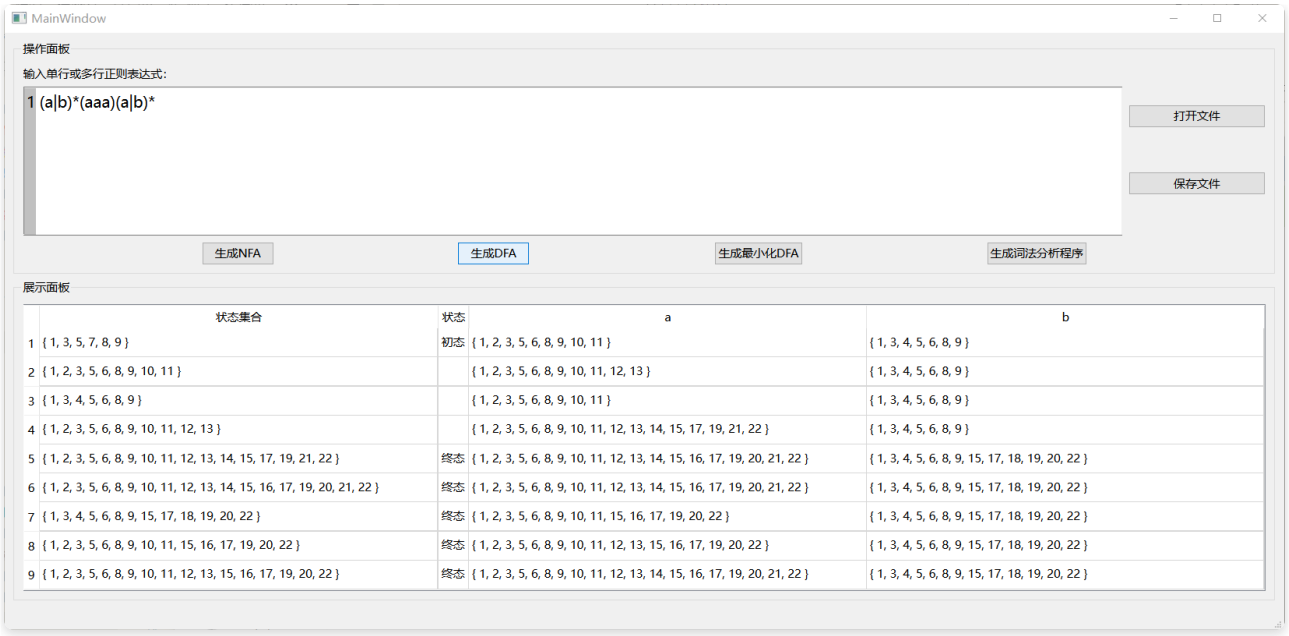
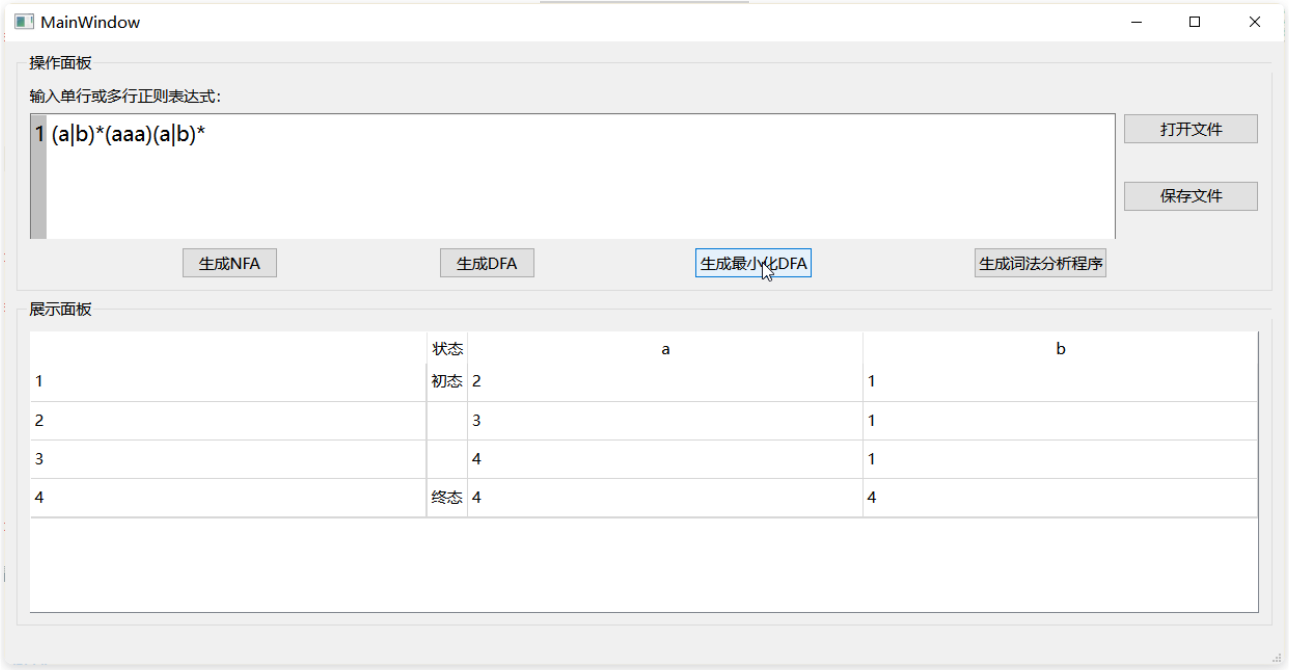


图6(3). 正则表达式转NFA(3)



	状态集合	状态	a	b
1	{ 1, 3, 5, 7, 8, 9 }	初态	{ 1, 2, 3, 5, 6, 8, 9, 10, 11 }	{ 1, 3, 4, 5, 6, 8, 9 }
2	{ 1, 2, 3, 5, 6, 8, 9, 10, 11 }		{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13 }	{ 1, 3, 4, 5, 6, 8, 9 }
3	{ 1, 3, 4, 5, 6, 8, 9 }		{ 1, 2, 3, 5, 6, 8, 9, 10, 11 }	{ 1, 3, 4, 5, 6, 8, 9 }
4	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13 }		{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 21, 22 }	{ 1, 3, 4, 5, 6, 8, 9 }
5	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19, 21, 22 }	终态	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22 }	{ 1, 3, 4, 5, 6, 8, 9, 15, 17, 18, 19, 20, 22 }
6	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22 }	终态	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22 }	{ 1, 3, 4, 5, 6, 8, 9, 15, 17, 18, 19, 20, 22 }
7	{ 1, 3, 4, 5, 6, 8, 9, 15, 17, 18, 19, 20, 22 }	终态	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 15, 16, 17, 19, 20, 22 }	{ 1, 3, 4, 5, 6, 8, 9, 15, 17, 18, 19, 20, 22 }
8	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 15, 16, 17, 19, 20, 22 }	终态	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22 }	{ 1, 3, 4, 5, 6, 8, 9, 15, 17, 18, 19, 20, 22 }
9	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22 }	终态	{ 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22 }	{ 1, 3, 4, 5, 6, 8, 9, 15, 17, 18, 19, 20, 22 }

图7. NFA转DFA



	状态	a	b
1	初态	2	1
2		3	1
3		4	1
4	终态	4	4

图8. 最小化DFA

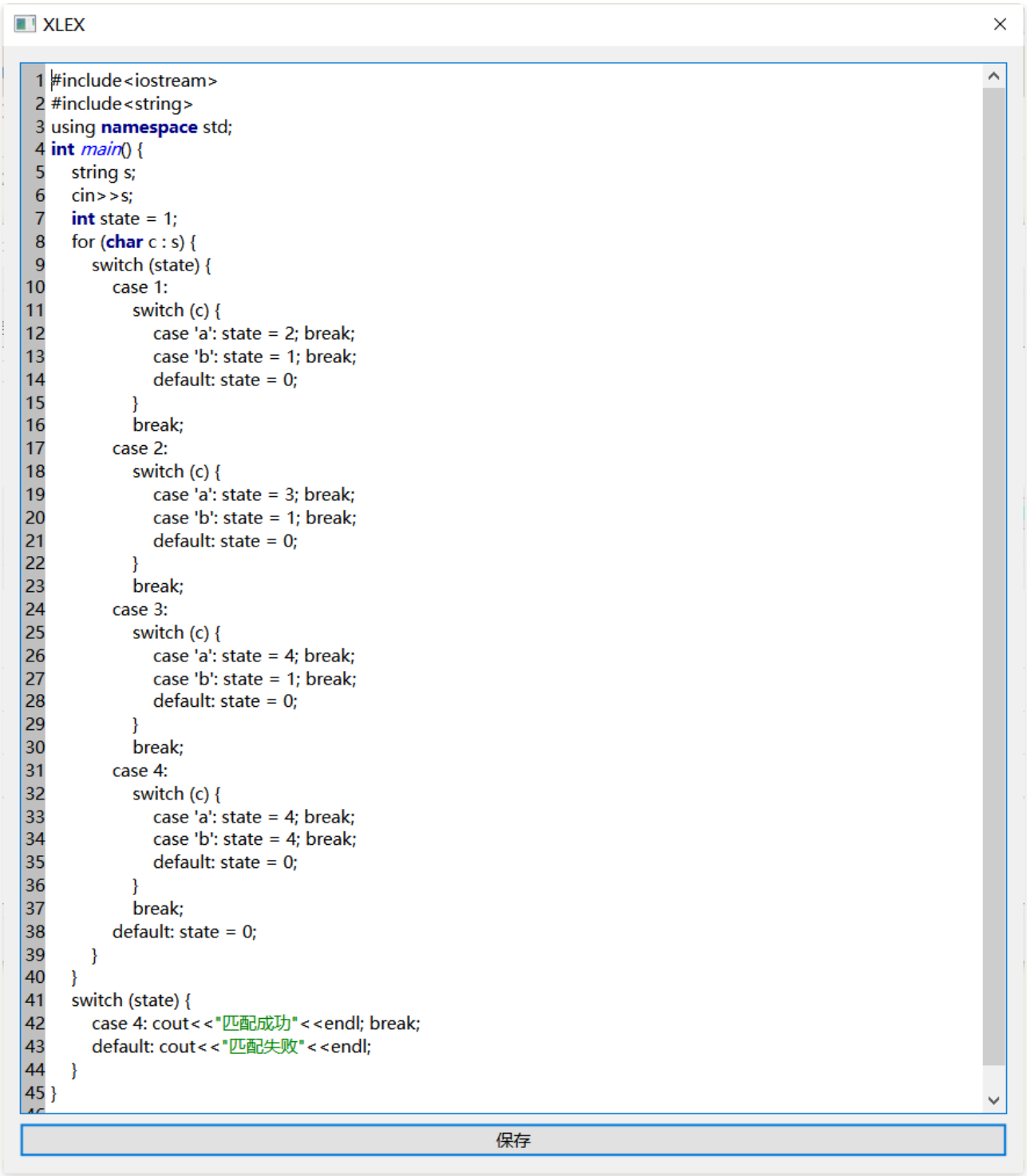

XLEX
#include<iostream>
#include<string>
using namespace std;
int main() {
string s;
cin>>s;
int state = 1;
for (char c : s) {
switch (state) {
case 1:
switch (c) {
case 'a': state = 2; break;
case 'b': state = 1; break;
default: state = 0;
}
break;
case 2:
switch (c) {
case 'a': state = 3; break;
case 'b': state = 1; break;
default: state = 0;
}
break;
case 3:
switch (c) {
case 'a': state = 4; break;
case 'b': state = 1; break;
default: state = 0;
}
break;
case 4:
switch (c) {
case 'a': state = 4; break;
case 'b': state = 4; break;
default: state = 0;
}
break;
default: state = 0;
}
}
switch (state) {
case 4: cout<<"匹配成功"<<endl; break;
default: cout<<"匹配失败"<<endl;
}
}
保存

图9. 生成词法分析程序

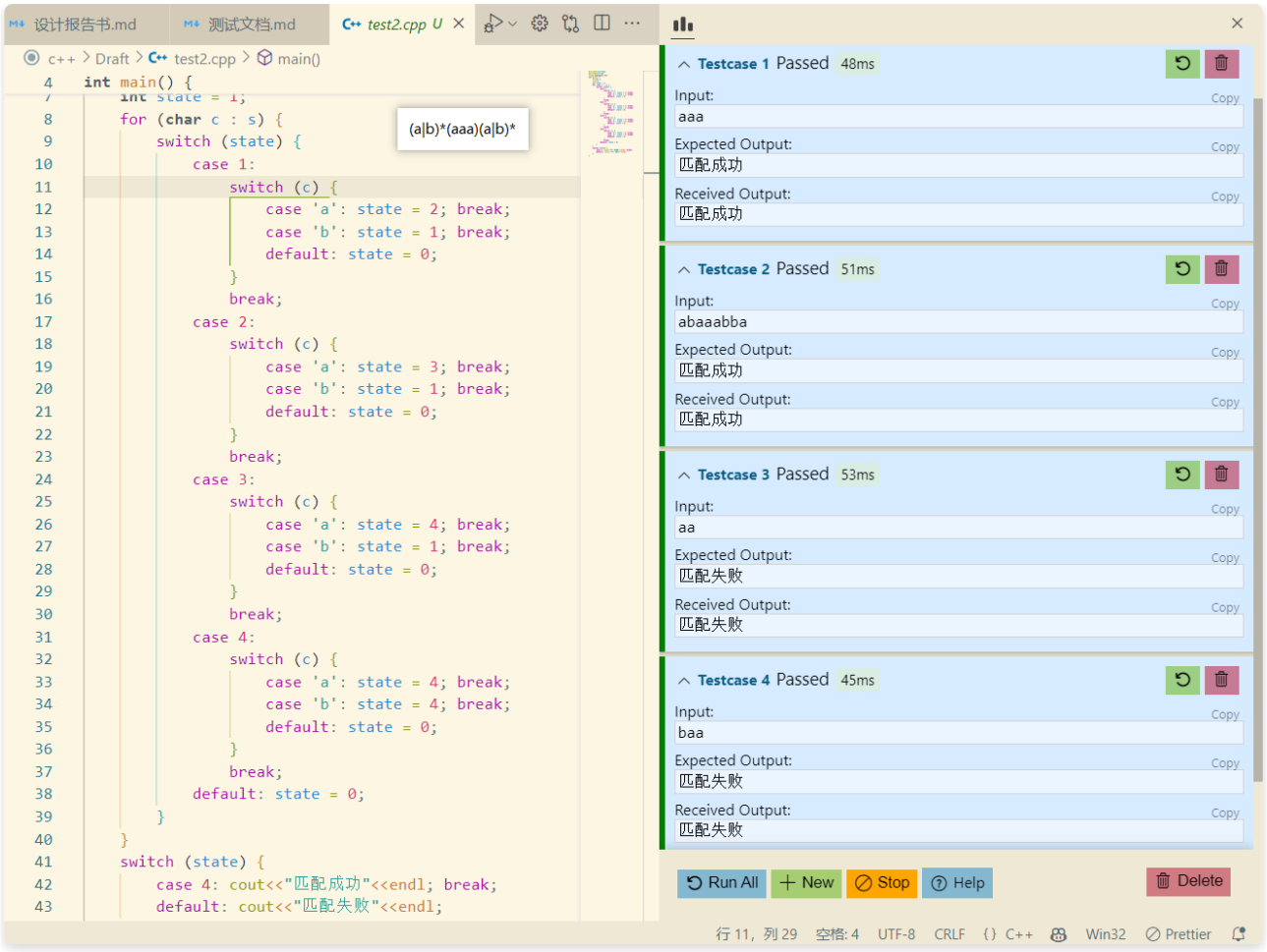


图10(1). 测试生成的词法分析程序(1)

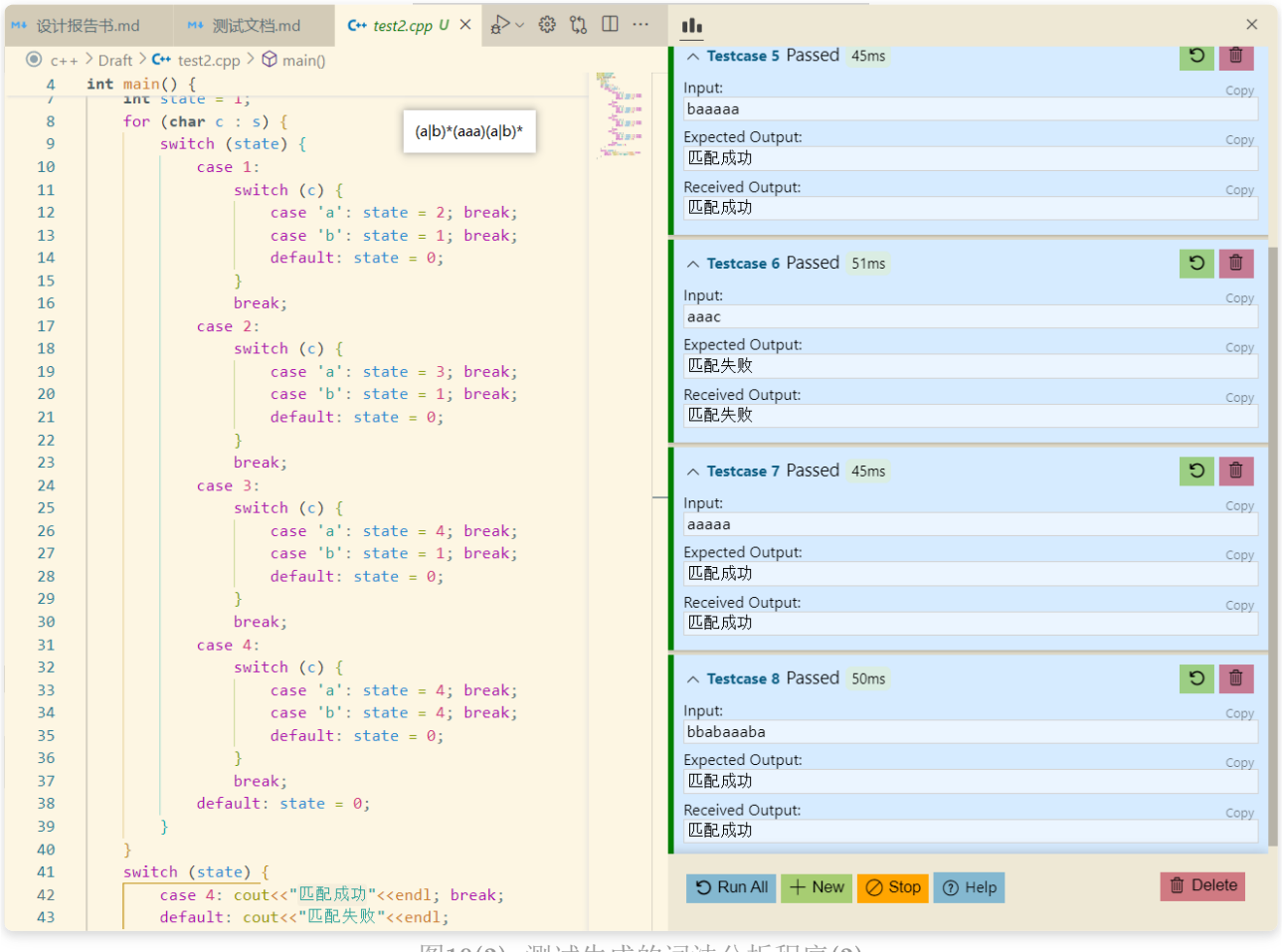
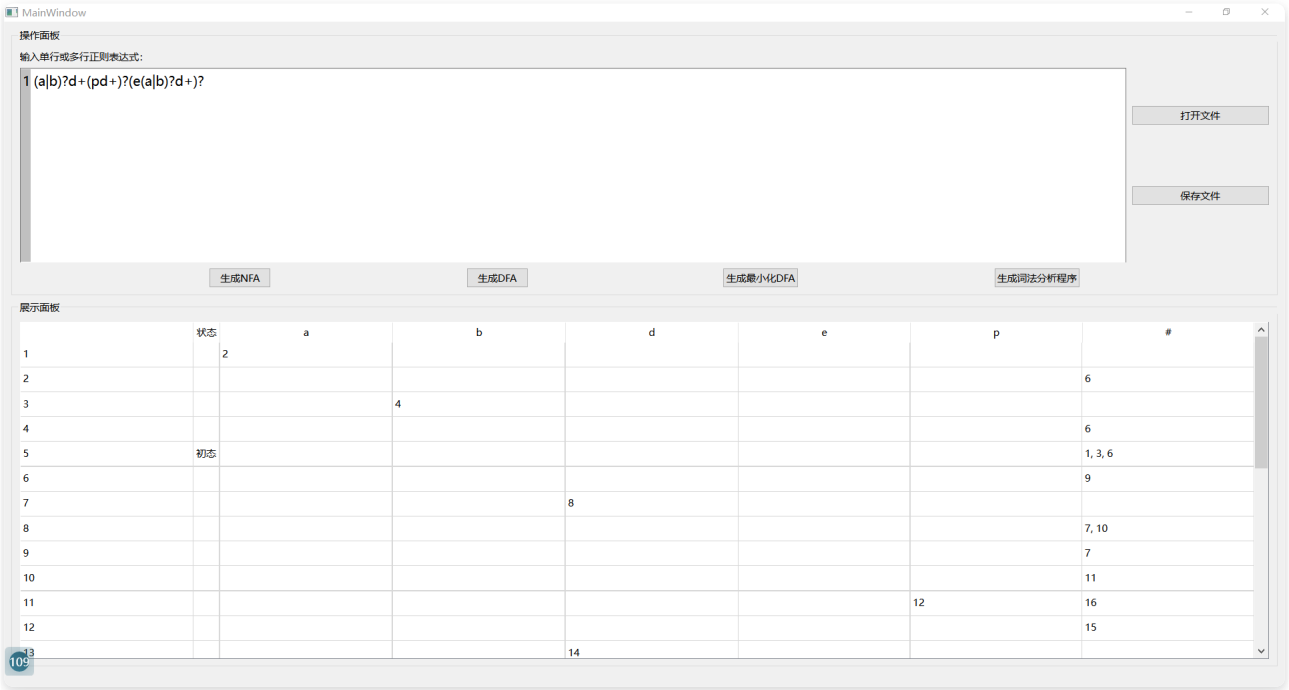


图10(2). 测试生成的词法分析程序(2)

可看出该程序能生成正确的对正则表达式 `(a|b)*(aaa)(a|b)*` 的词法分析程序

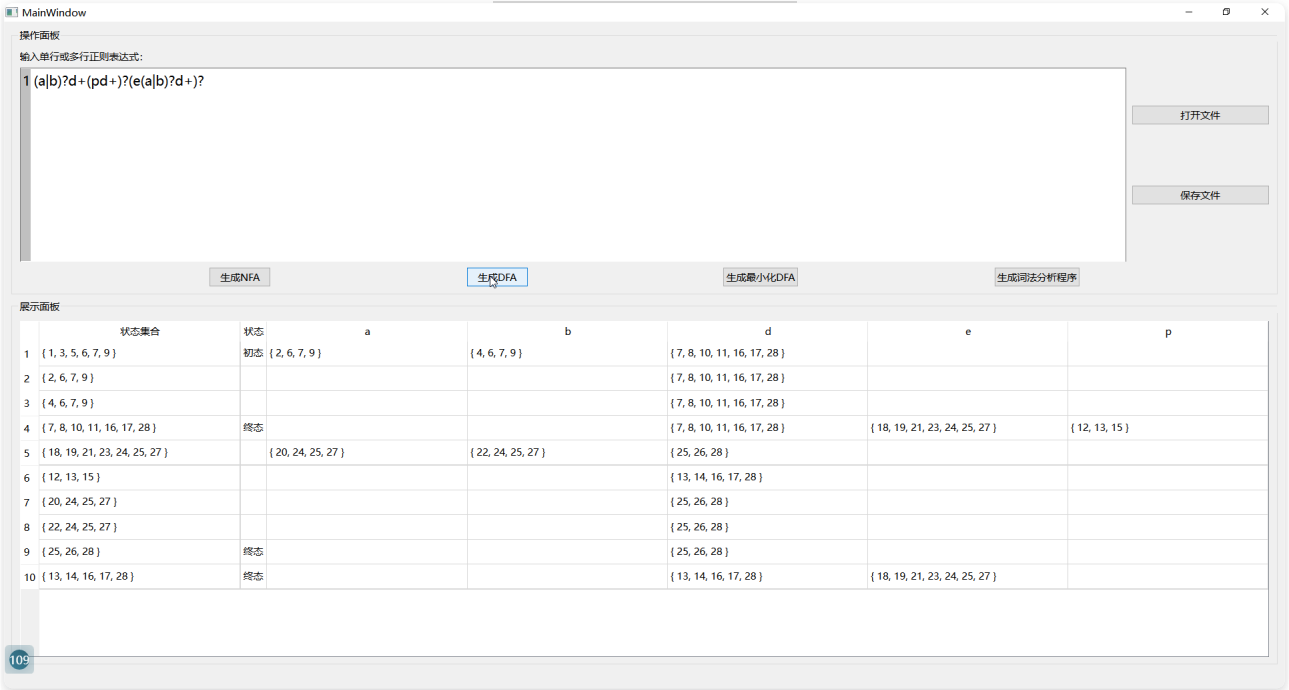
3 测试3

选择正则表达式 `(a|b)?d+(pd+)?(e(a|b)?d+)?` 作为测试用例测试结果如下：



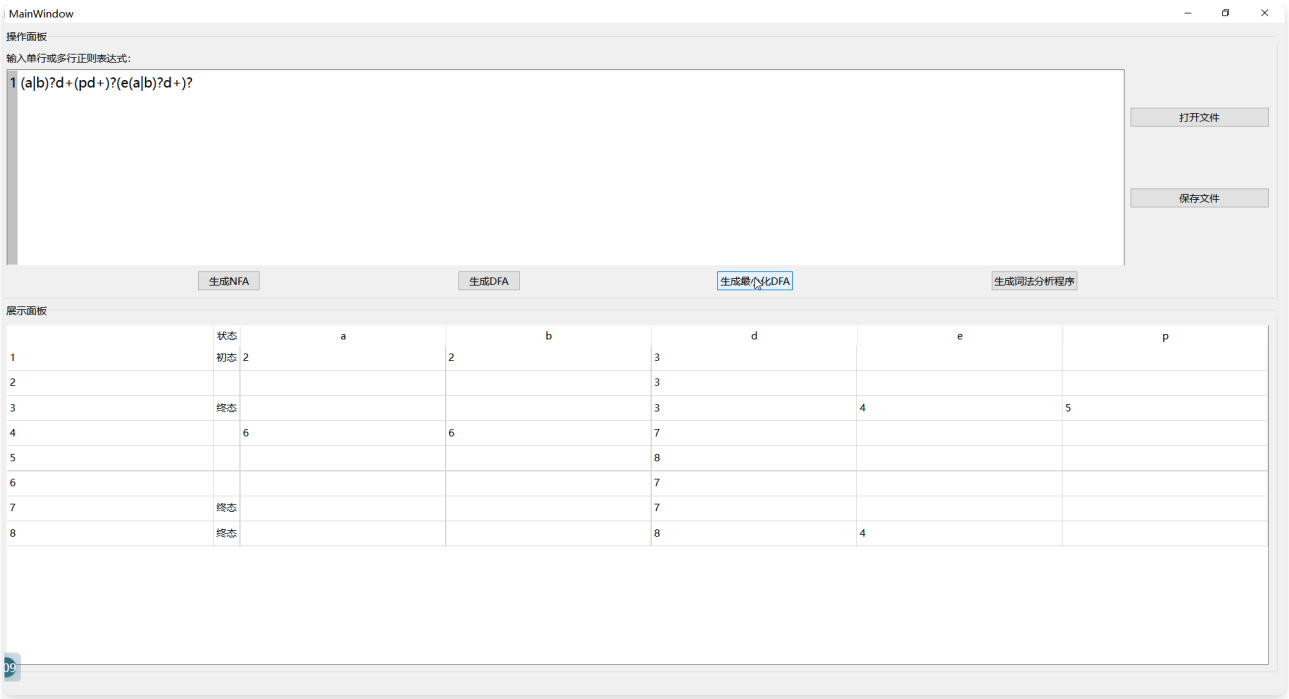
	状态	a	b	d	e	p	#
1		2					
2							6
3			4				
4							6
5	初态						1, 3, 6
6							9
7				8			
8							7, 10
9							7
10							11
11						12	16
12							15
13				14			

图11. 正则表达式转NFA



	状态集合	状态	a	b	d	e	p
1	{ 1, 3, 5, 6, 7, 9 }	初态	{ 2, 6, 7, 9 }	{ 4, 6, 7, 9 }	{ 7, 8, 10, 11, 16, 17, 28 }		
2	{ 2, 6, 7, 9 }				{ 7, 8, 10, 11, 16, 17, 28 }		
3	{ 4, 6, 7, 9 }				{ 7, 8, 10, 11, 16, 17, 28 }		
4	{ 7, 8, 10, 11, 16, 17, 28 }	终态			{ 7, 8, 10, 11, 16, 17, 28 }	{ 18, 19, 21, 23, 24, 25, 27 }	{ 12, 13, 15 }
5	{ 18, 19, 21, 23, 24, 25, 27 }		{ 20, 24, 25, 27 }	{ 22, 24, 25, 27 }	{ 25, 26, 28 }		
6	{ 12, 13, 15 }				{ 13, 14, 16, 17, 28 }		
7	{ 20, 24, 25, 27 }				{ 25, 26, 28 }		
8	{ 22, 24, 25, 27 }				{ 25, 26, 28 }		
9	{ 25, 26, 28 }	终态			{ 25, 26, 28 }		
10	{ 13, 14, 16, 17, 28 }	终态			{ 13, 14, 16, 17, 28 }	{ 18, 19, 21, 23, 24, 25, 27 }	

图12. NFA转DFA



	状态	a	b	d	e	p
1	初态	2	2	3		
2				3		
3	终态			3	4	5
4		6	6	7		
5				8		
6				7		
7	终态			7		
8	终态			8	4	

图13. 最小化DFA

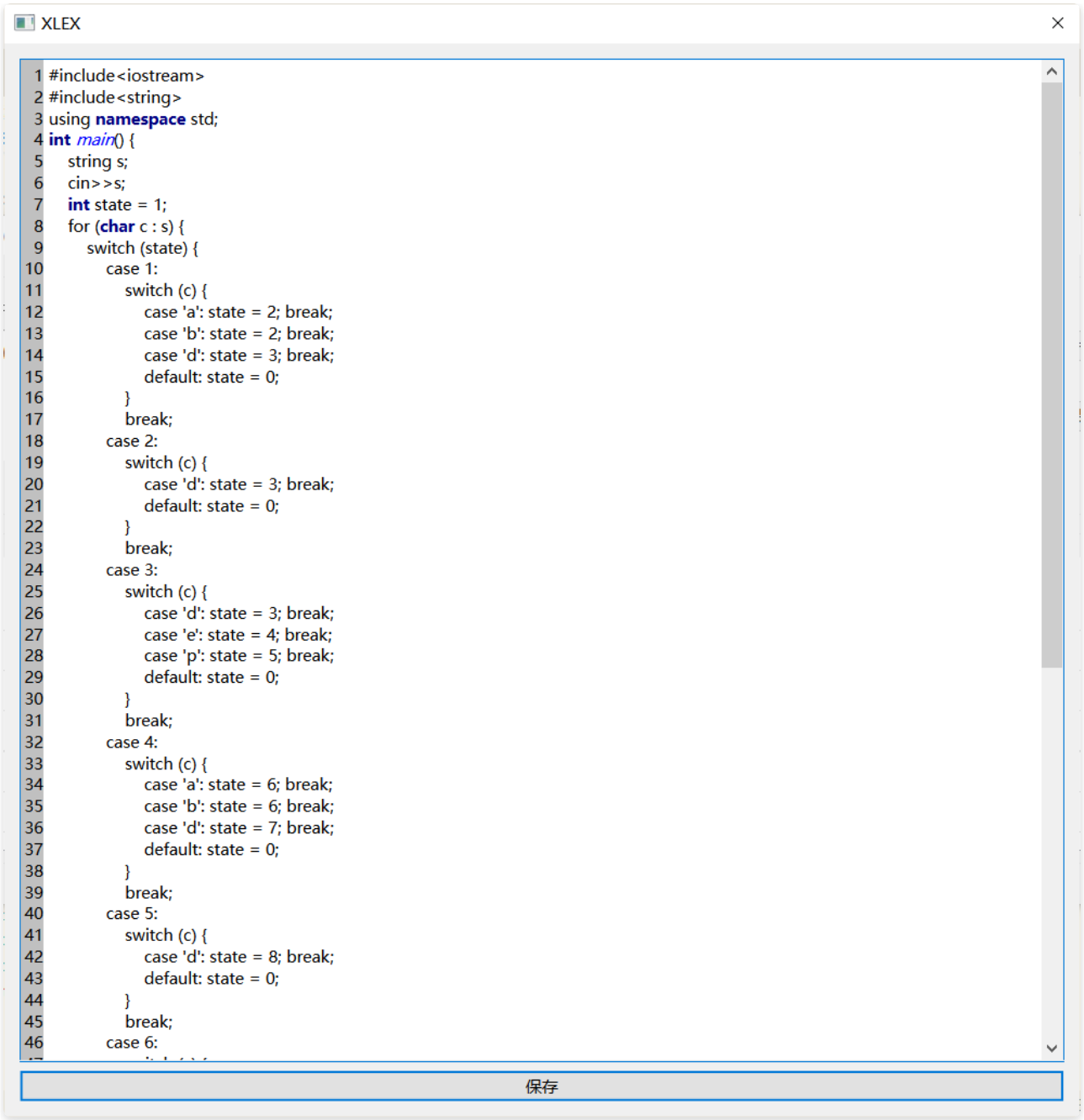


图14. 生成词法分析程序

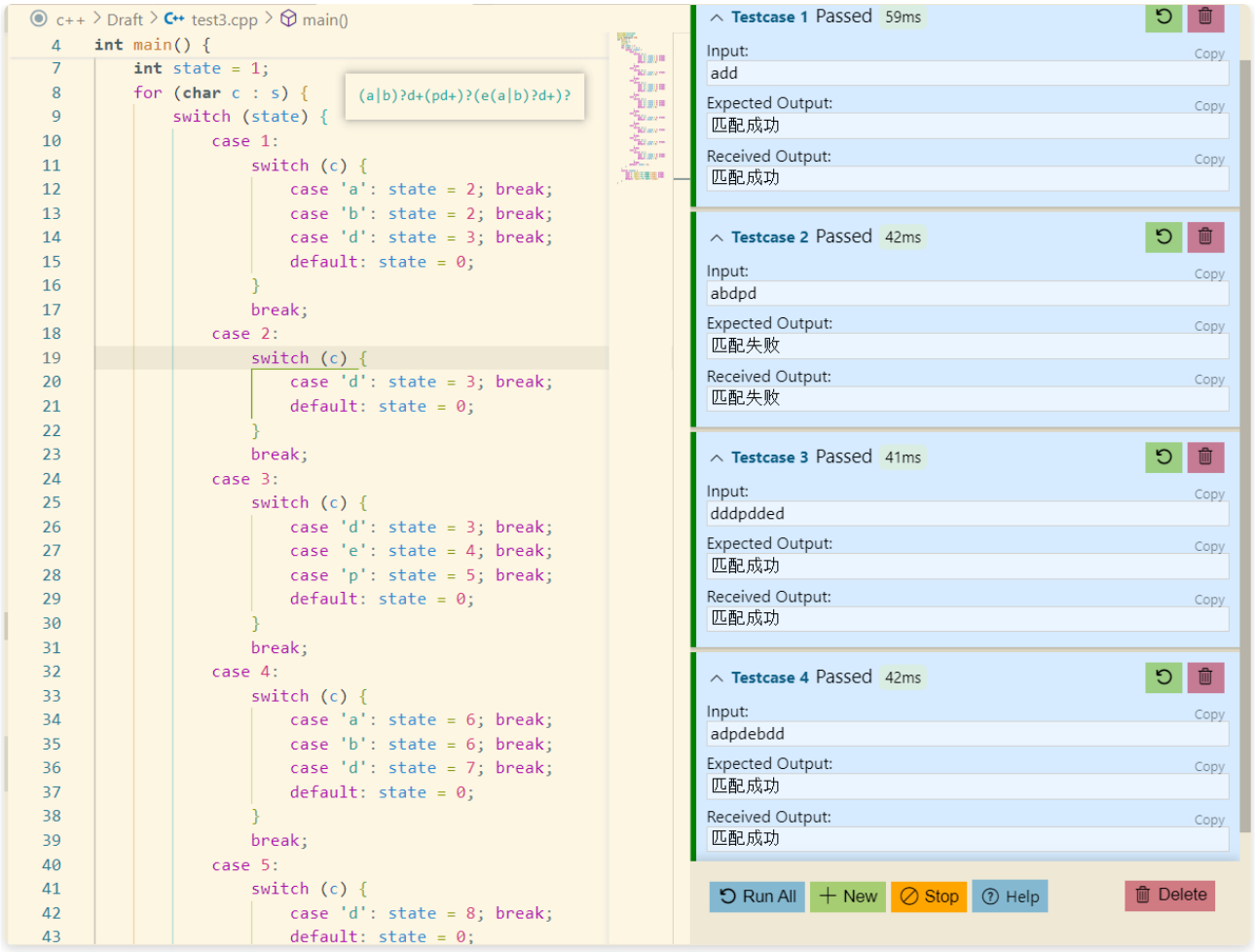


图15. 测试生成的词法分析程序

该正则表达式为类浮点数匹配的正则表达式，浮点数的**DFA**如下图：

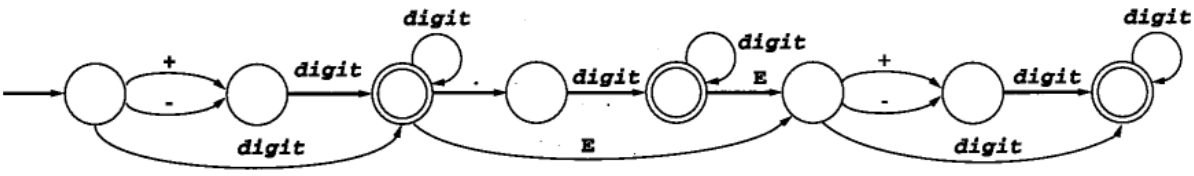
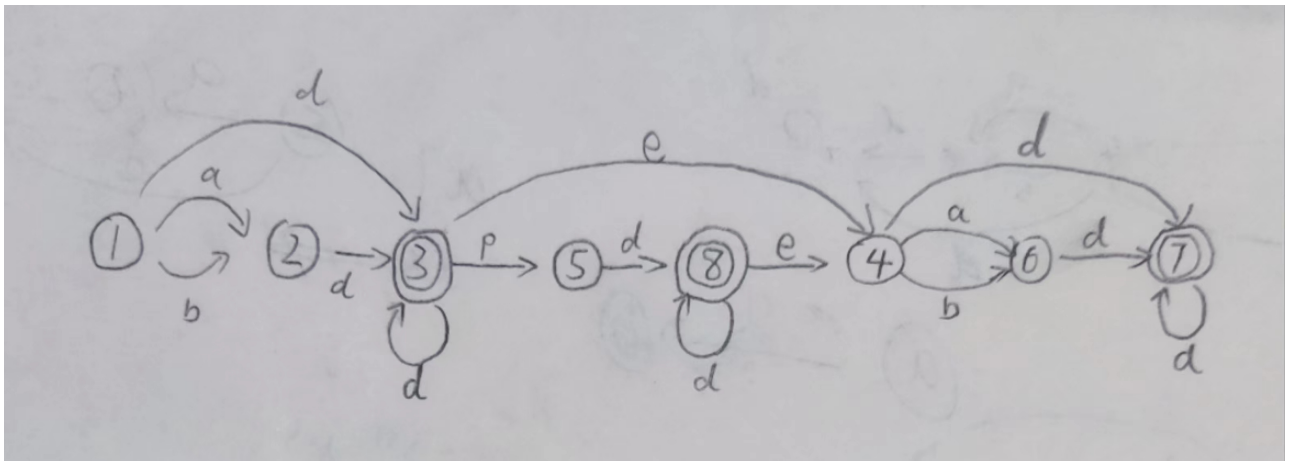


图2-3　浮点数的有穷自动机

根据图**13**生成的最小**DFA**，画出状态转换图，可以发现跟浮点数的**DFA**图高度一致，表明该程序对正则表达式 `(a|b)?d+(pd+)?(e(a|b)?d+)?` 生成了正确的最小**DFA**



根据图**15**可看出该程序能生成正确的对正则表达式 `(a|b)?d+(pd+)?(e(a|b)?d+)?` 的词法分析程序