成绩：

课程名称：系统分析与验证

项目名称：一种拓展模型检查算法（EMC）的 C++ 实现

项目成员：

姓名	项目分工
岳泽龙（组长） 51205902148	进行需求分析，实现部分标记算法，实现加标记和检测标记算法；将三人代码耦合；撰写文档；
李凯旋 52205902008	实现读取输入文件；查找强连通分量算法的实现；根据 Fairness Constraints 找到公平路径，并对公平路径上的结点进行标记；项目图表绘制，文档撰写；
孙衍超 51194501064	实现 CTL 子公式生成，读取 CTL 公式并解析操作符，根据子公式实现由短到长进行模型检查；项目文档撰写；

时间：2021 年 1 月 4 日

目　录

一、 项目需求分析 3
二、 概要设计 5
三、 详细设计 7
3.1 输入文件的处理 7
3.2 求解强连通分量 7
3.3 查找公平路径上节点 7
3.4 CTL 公式预处理 7
3.5 CTL 公式解析 7
3.6 计算 CTL 子公式 7
四、 实现 9
4.1 求解强连通分量(Tarjan 算法) 9
4.2 求解满足 fairness constraints 的强连通分量 10
4.3 求解公平路径，并对其中节点标号 Q 10
4.4 根据 CTL 公式生成子公式 12
4.5 标记算法伪代码 13
五、 测试结果 15
5.1 互斥问题测试 15
5.2 Alternating Bit Protocol 模型验证 18
六、 项目总结与分析 20

一、 项目需求分析

本项目需要实现一种拓展模型检查算法（EMC），其在普通模型检查算法的基础上，增加了公平性的约束条件（Fairness Constraints）。该算法需要实现的功能如下：

[1] 接收 Kripke Structure、CTL 公式、Fairness Constraints。

[2] 解析 Kripke Structure，将结点、迁移关系和属性以某种数据结构进行存储。

[3] 解析 CTL 公式，并根据 CTL 公式得到所有 CTL 子公式。

[4] 找到 Kripke Structure 中的强连通分量。同时根据 Fairness Constraints 找到公平路径，并对公平路径上的结点进行标记。

[5] 从短到长，把所有 CTL 子公式在结点上进行检测，并对满足 CTL 公式的结点进行标记。

[6] 对所有公平路径上的结点检查是否被标记目标 CTL 公式，如有未被标记的结点，则称 CTL 公式不被满足。如所有结点都被标记，则称 CTL 公式被满足。

针对上述功能，我们需要设计输入输出格式，设计算法解析输入的信息，设计算法解析 CTL 公式并生成子公式，设计算法根据运算符来标记结点，设计算法找到强连通分量，最后检测目标 CTL 公式是否在需求结点上被标记，输出结果。

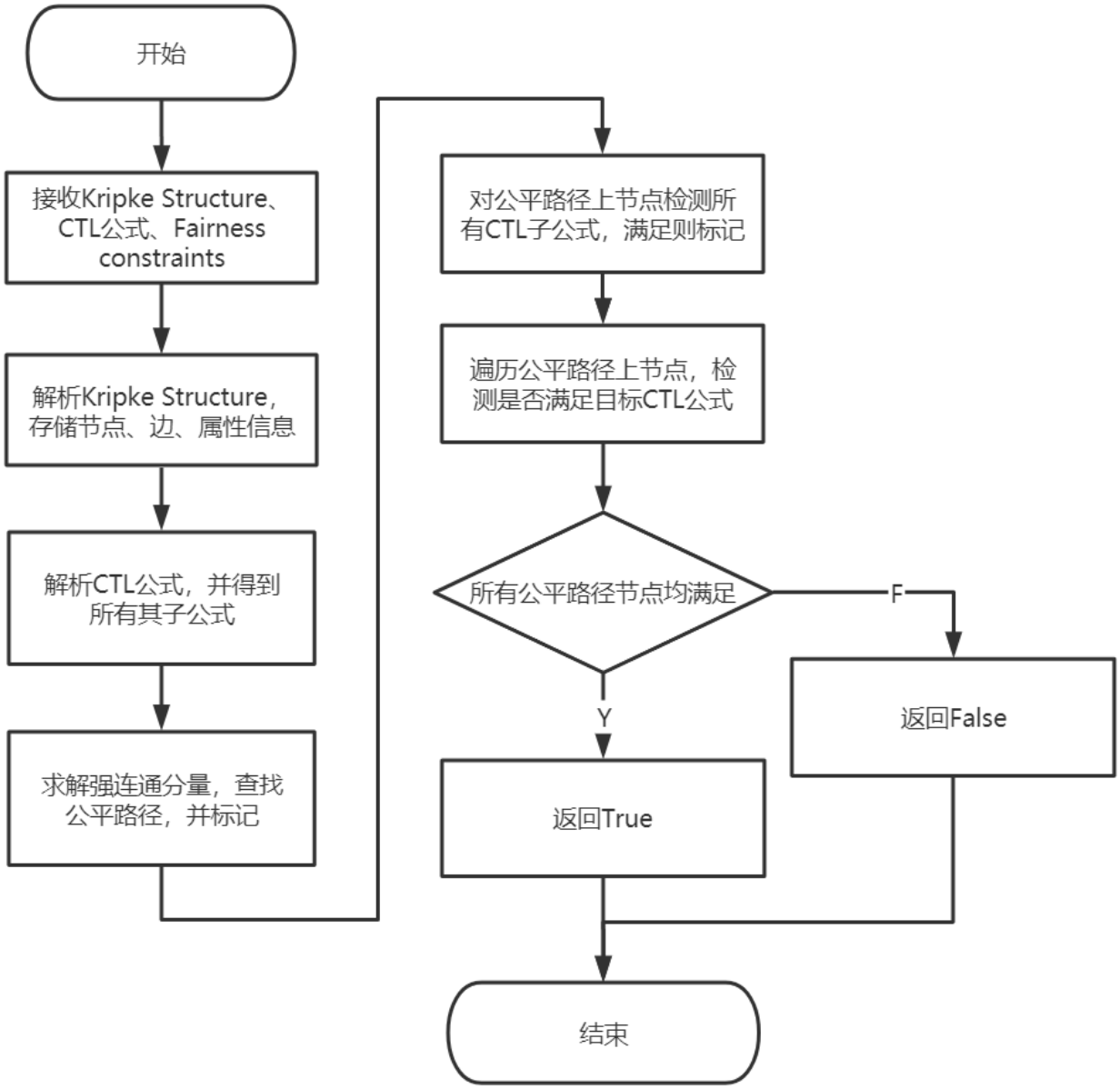

图 1.1.1 项目整体实现思路流程图

二、 概要设计

本项目计划使用 C++ 实现拓展模型检查算法（EMC），该算法将公平性引入了 CTL 模型检查中。首先读取输入文件中的节点状态，节点和边信息；其次，通过查找强连通分量,筛选符合公平性属性的强连通分量;进一步查找公平路径：使用 BFS 算法查找可达满足公平属性强连通分量节点的节点，并将其标号 Q;存储公平路径上的节点；最终结合公平路径，按照从短到长标记 CTL 子公式的方法，实现模型检查。

输入格式如下：

NODES ARCS

// NODES 表示结点个数，ARCS 表示边数。

0 1

1 2

// 空格左侧代表起始结点，右侧代表目的结点。有多少行取决于上面的 ARCS。

CTL FORMULA

AU(NOT(X))(OR(Y)(Z))

// 输入 CTL 公式，操作符位于公式前面，且所有公式需要使用括号括起来。

FAIRNESS

// Fairness 代表 Fairness Constraints 的个数。

1 2

// 输入 Fairness Constraints 结点，用空格分隔。

输出格式如下：

This is properties of staten:

X1 X2

// 输出状态 n 的属性

Here is SCC:

1 2 3 4 5

// 输出强连通分量里的状态

This is fair&scc:

1 2 3 4 5

// 输出满足公平约束的强连通分量

This is set_Q:

1 2 3 4 5

// 输出公平路径中的结点（标记为 Q）的结点

NOT(X1)

// 输出原 CTL 公式

expressions:

NOT(X1):0

X1:1

// 输出由原公式生成的子公式

subformula_index:

X1: -1

NOT(X1): 1

// 输出该公式生成的子公式在数组中保存的序号，若为 -1 则表示无法继续生成子公式。

The CTL formula is FALSE

// 输出模型检查结果

三、 详细设计

本小组在项目实现时，使用的编程语言为：C++；实现的环境为 Windows 10 下使用 GCC 9.2.0 + CLion 2020.03。

整个项目使用了多种数据结构，包括：set、stack、vector、map、unordered_map、queue 等。

3.1 输入文件的处理

使用 string，调用 getline 函数对输入文件内容循环读取并处理；使用 C++ 中的 vector 存储 Kripke Structure，使用 string 读取 CTL 公式；

3.2 求解强连通分量

使用 stack 实现 Tarjan 算法；通过 Tarjan 算法的实现来查找强连通分量，并使用 vector 存储强连通分量的节点；

3.3 查找公平路径上节点

通过 map、queue 和 vector 实现查找满足公平性约束的强连通分量；借助 BFS 算法，使用 queue 的数据结构寻找公平路径，并输出；

3.4 CTL 公式预处理

算法思想：先将表达式中的空格去掉，然后分析表达式的前缀，若为 AF、EF、AG 或 EG 中的其中之一，则将表达式转换成其他容易处理的操作符。

3.5 CTL 公式解析

该部分使用 vector 存储表达式及其各子表达式，vector 结构的索引表示各表达式的编号。

算法思想：将表达式加入 vector 容器，判断表达式操作符，对于一元操作符，找到表达式字符串的子串中子表达式，继续解析该子表达式；对于二元操作符，通过分析括号的位置，找到该表达式对应的两个子表达式，依次解析这两个子表达式。执行该算法，直到表达式为原子表达式。

3.6 计算 CTL 子公式

该部分使用 map 数据结构存储表达式与其子表达式编号之间的映射关系，所有的子表达式编号依次存储在 vector 数据结构中。

算法思想：输入表达式，分析表达式的操作符，对于一元操作符，首先找到其子表达式在上述 CTL 解析中对应的编号，将该表达式与此编号形成映射；对于二元操作符，通过分析括号位置，找到该表达式对应的两个子表达式，分别得到它们对应的编号，同样与该表达式形成映射。对于原子表达式，与之对应的子表

达式编号设为-1 。

具体实现细节请参考第四节实现部分内容。

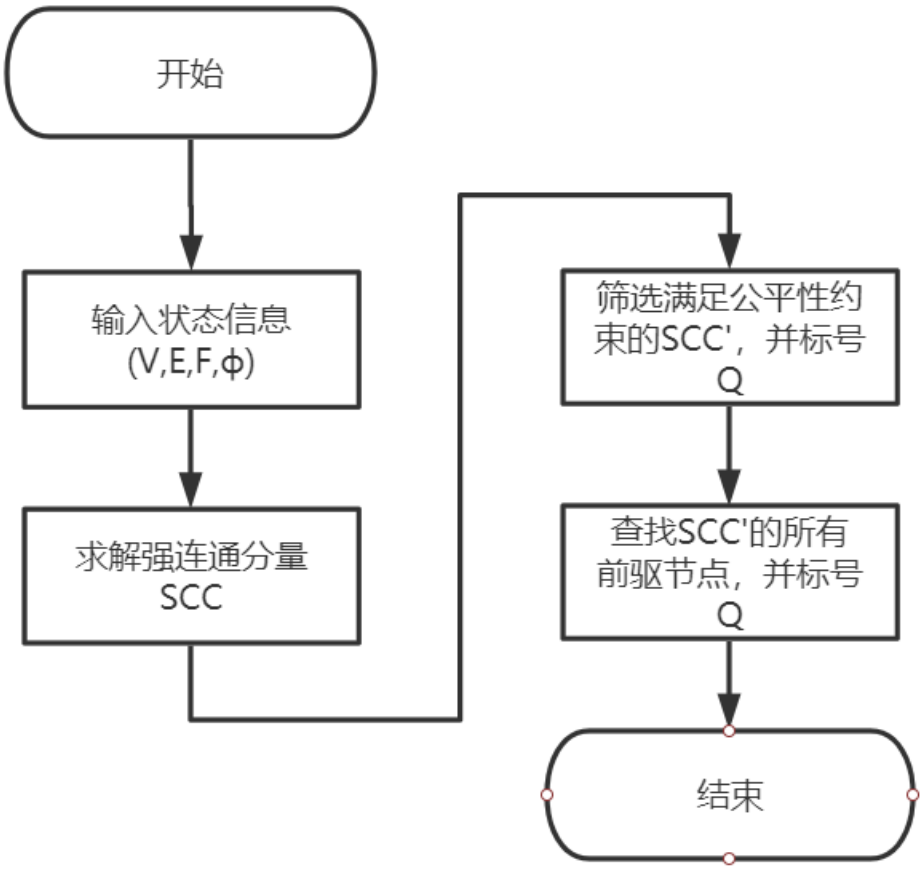

图 3.1.1 求解公平路径节点算法流程图

四、 实现

本节给出了核心算法的伪代码：

4.1 求解强连通分量(Tarjan 算法)

Input:节点；

Output:有向图中的强连通分量；

```
4.1 Tarjan(u)
4.2 {
4.3     DFN[u]=Low[u]=++Index//为节点 u 设定次序编号和 Low 初值
4.4     Stack.push(u)//将节点 u 压入栈中
4.5     for each(u,v) in E//枚举每一条边
4.6         if (v is not visited)//如果节点 v 未被访问过
4.7             tarjan(v)//继续向下找
4.8             Low[u]=min(Low[u],Low[v])
4.9         endif
4.10         else if (vinS)//如果节点 v 还在栈内
4.11                 Low[u]=min(Low[u],DFN[v])
4.12     endfor
4.13     if (DFN[u]==Low[u])//如果节点 u 是强连通分量的根
4.14     repeat{
4.15         v=S.pop//将 v 退栈，为该强连通分量中一个顶点
4.16         print v
4.17         until(u==v)
4.18     }
4.19     endif
4.20 }
```

4.2 求解满足 fairness constraints 的强连通分量

Input: Fairness constraints vector<int>），强连通分量集合 Scc (vector<int>)；

Output: 满足 Fairness constraints 的强连通分量 scc (vector<int>);

```
// *****比较 scc 中是否含有子集包含 f 中的所有状态,并返回**********/
vector<int> scc_flag(vector<int>& fairset, vector<vector<int> >& Scc) {
    vector<int> res;
    for each scc in vector<vector<int> >Scc:
        for each node in scc:
            if (node is in fairset)
                continue;
            endif
            else break;
            res=scc;
        endfor
    endfor
    return res;
}
```

4.3 求解公平路径，并对其中节点标号 Q

查找满足 fairness constraints 的强连通分量中节点的前驱，最终返回标记 Q 的集合

Input：强连通分量(vector<int>)；

Output:需要标号 Q 的节点集合(vector<int>)；

```
//************查找节点的前驱；用于标记 Q;*******************/
vector<int> Find_Fathers(vector<int>& v1) {
    queue<int> qe;
    for each node n in v1:
        qe.push(n);
        label node n as checked;
```

```
7.       Endfor
8.      while (!qe.empty()) do{
9.          int now = qe.front();
10.         vector<int> now_fa = father[now];
11.         qe.pop();
12.         for each m in now_fa:
13.             if (m is unchecked) continue;
14.             else {
15.                 v1.push_back(m);
16.                 label node n as checked;
17.             }
18.         Endfor
19.     }
20.     return v1;
21. }
```

4.4 根据 CTL 公式生成子公式

Input: CTL 公式（string）

Output: 子公式集合（vector<string>）

```
parseExpression:
    add expression to vector expressions;//容器 expressions 存放所有表达式
    if(op is unary) then
        subExpression <- expression.substr() without "()";
        parseExpression(subExpression);
    else if(op is binary) then
        subExpression_1,subExpression_2 <- getSubExpression(expression);
        parseExpression(subExpression_1);
        parseExpression(subExpression_2);

getSubExpression:
    index = 0;//索引
    count = 0;//用于括号计数，"(" : count++ ; ")" : count--;
    while index < expression.length() do
        if(expression[index] == '(' ) then
            count ++;
        else if(expression[index] == ')') then
            cout --;
        if(cout == 0) then
            break;
    end
    subExpression_1 <- expression.substr(1, index - 1);
    subExpression_2 <- expression.substr(index + 2, expression.length()-
index-3);

computeSubformularNumber:
    if(op is unary) then
        subformular_number[expression] <- subExpression'index in expressions
 ;//subformular_number 存放表达式与子表达式编号的映射
        computeSubformularNumber(subExpression);
    else if(op is binary) then
        subExpression_1,subExpression_2 <- getSubExpression(expression);
        subformular_number[expression][0] <- subExpression_1'index in expres
sions ;
        subformular_number[expression][1] <- subExpression_2'index in expres
sions ;
        computeSubformularNumber(subExpression_1);
        computeSubformularNumber(subExpression_2);
    else then //原子表达式
        subformular_number[expression] <- -1;
```

4.5 标记算法伪代码

标记算法一共 6 种，分别处理：$\neg p$、$p \wedge q$、$p \vee q$、$E(pUq)$、$A(pUq)$、EXp；伪代码如下 Algorithm 1、Algorithm 2、Algorithm 3、Algorithm 4、Algorithm 5、Algorithm 6 所示。

Algorithm 1: Label algorithm for ¬p

```
function NOT (graph_n, fi);
  Input: State graph graph_n and CTL formula fi (like ¬p)
foreach state S in state graph graph_n do
    if there is prop p in S then
        continue
    else
        label S with ¬p
    end
end
```

Algorithm 2: Label algorithm for $p \wedge q$

```
function AND (graph_n, fi);
  Input: State graph graph_n and CTL formula fi (like p ∧ q)
foreach state S in state graph graph_n do
    if there is prop p and q in S then
        label S with p ∧ q
    else
        continue
    end
end
```

Algorithm 3: Label algorithm for $p \vee q$

```
function OR (graph_n, fi);
  Input: State graph graph_n and CTL formula fi (like p ∨ q)
foreach state S in state graph graph_n do
    if there is prop p or q in S then
        label S with p ∨ q
    else
        continue
    end
end
```

Algorithm 4: Label algorithm for $E(pUq)$

```
function EU (graph_n, fi);
  Input: State graph graph_n and CTL formula fi (like E(pUq))
foreach state S in state graph graph_n do
    if there is prop q in S then
        find paths that can reach S, (in such paths) if the states before S all have p as a prop, label them with
         E(pUq)
    else
        continue
    end
end
```

Algorithm 5: Label algorithm for $A(pUq)$

```
1  function AU (graph_n, fi);
   Input: State graph graph_n and CTL formula fi (like A(pUq))
2  foreach state S in state graph graph_n do
3      if state S is marked as visited then
4          if if S is labelled with A(pUq) then
5              flag = true; return
6          else
7              flag = false; return
8          end
9      mark S as visited
10     if S is labelled with q then
11         label S with A(pUq); flag = true; return
12     else if S is not labelled with p then
13         flag = flase; return
14 end
```

Algorithm 6: Label algorithm for EXp

```
1  function EX (graph_n, fi);
   Input: State graph graph_n and CTL formula fi (like EXp)
2  foreach state S in state graph graph_n do
3      if there is prop p in S then
4          forall S1 ∈ pre(S) do
5              label S1 with EXp
6          end
7      else
8          continue
9      end
10 end
```

五、 测试结果

使用方法：在 cmd 或者 PowerShell 中以 \path\to\main.exe 的形式打开 main.exe 文件，测试用例命名为 TestModel1.txt，与 main.exe 置于同一目录。

5.1 互斥问题测试

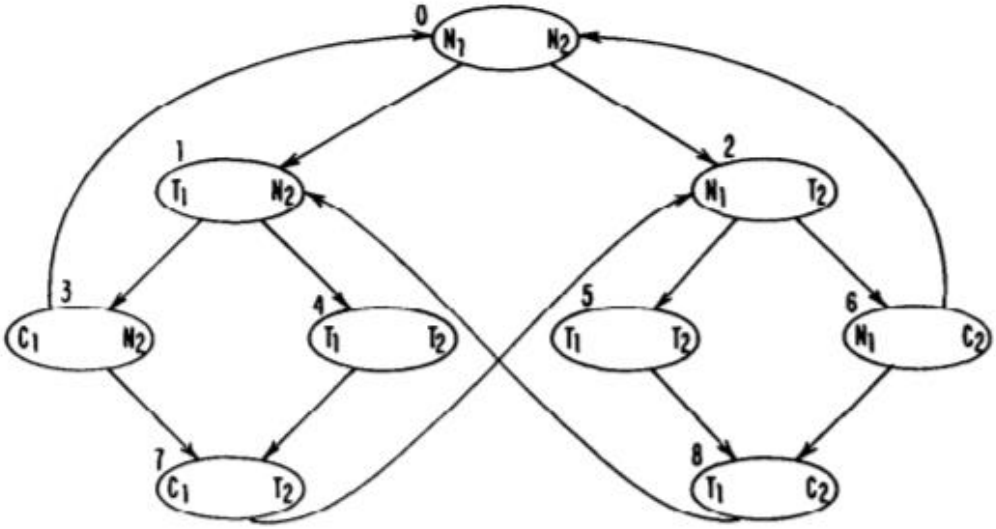

图 5.1.1 互斥问题模型

使用上图互斥算法作为测试用例，测试文件于测试用 CTL 公式如下：

p1 := EX(N1)；易知该样例为真，截图如下：

```
→ .\main.exe
This is properties of state0:
N1 N2
This is properties of state1:
T1 N2
This is properties of state2:
N1 T2
This is properties of state3:
C1 N2
This is properties of state4:
T1 T2
This is properties of state5:
T1 T2
This is properties of state6:
N1 C2
This is properties of state7:
C1 T2
This is properties of state8:
T1 C2
Here is SCC:
4 6 8 5 2 7 3 1 0
This is fair&scc:
4 6 8 5 2 7 3 1 0
This is set_Q:
4 6 8 5 2 7 3 1 0
EX(N1)
expressions:
EX(N1):0
N1:1
subformula_index:
EX(N1) : 1
N1 : -1
The CTL formula is TRUE
```

图 5.1.2 互斥问题测试结果（1）

p2 := NOT(N1)；易知该样例为假，截图如下：

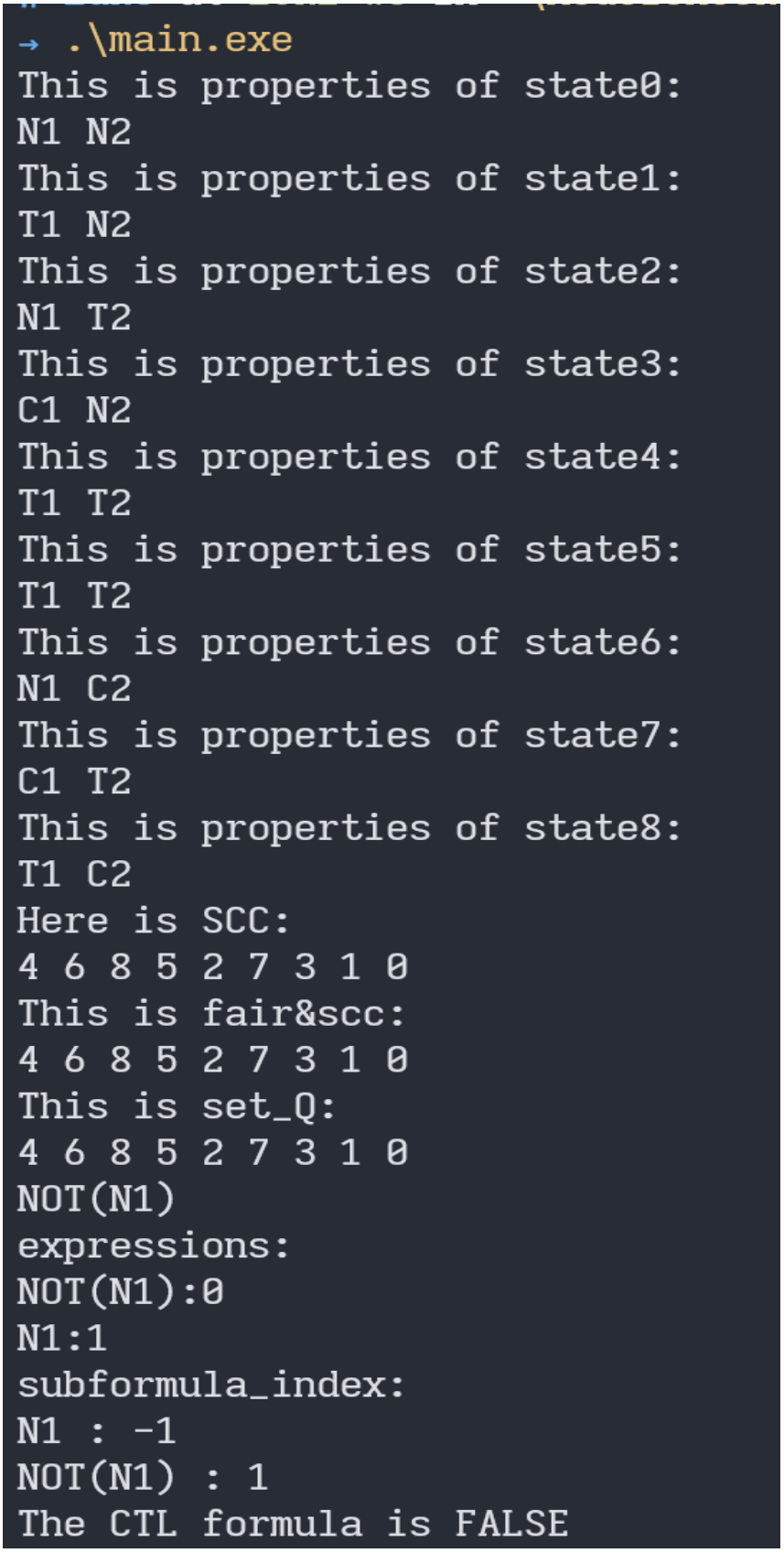

图 5.1.3 互斥问题测试结果（2）

5.2 Alternating Bit Protocol 模型验证

使用 ABP 模型进行验证，验证文中的第一个 CTL 公式，验证模型和验证结果如下:

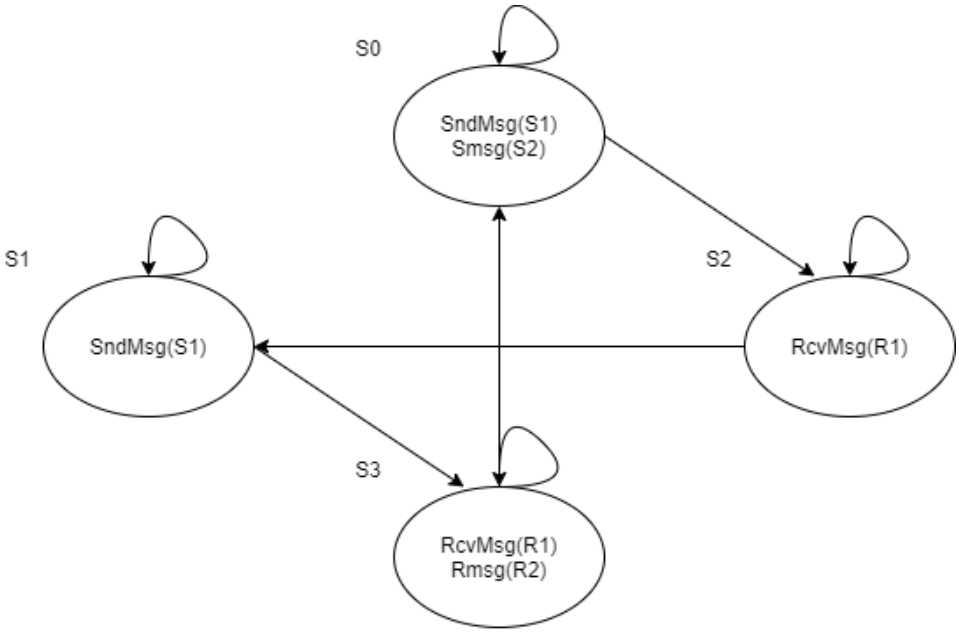

图 5.2.1 Alternating Bit Protocol 验证模型

```
→ .\main.exe
This is properties of state0:
S1 S2
This is properties of state1:
S1
This is properties of state2:
R1 R2
This is properties of state3:
R1
Here is SCC:
3 1 2 0
This is fair&scc:
3 1 2 0
This is set_Q:
3 1 2 0
NOT(EU(TRUE)(NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1)))))))
expressions:
NOT(EU(TRUE)(NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1)))))))):0
EU(TRUE)(NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1))))))):1
TRUE:2
NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1))))):3
OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1)))):4
NOT(R1):5
R1:6
AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1))):7
R1:8
AND(NOT(R1))(AU(NOT(R1))(S1)):9
NOT(R1):10
R1:11
AU(NOT(R1))(S1):12
NOT(R1):13
R1:14
S1:15
subformula_index:
AND(NOT(R1))(AU(NOT(R1))(S1)) : 5  12
AU(NOT(R1))(S1) : 5  15
AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1))) : 6  9
EU(TRUE)(NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1)))))) : 2  3
NOT(EU(TRUE)(NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1))))))) : 1
NOT(OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1))))) : 4
NOT(R1) : 6
OR(NOT(R1))(AU(R1)(AND(NOT(R1))(AU(NOT(R1))(S1)))) : 5  7
R1 : -1
S1 : -1
TRUE : -1
The CTL formula is TRUE
```

图 5.2.2 ABP 模型验证结果

六、 项目总结与分析

本项目使用 C++ 实现了 Edmund Clarke 等人提出的拓展模型检查算法（EMC），该算法将公平性引入了 CTL 模型检查中。按照从短到长标记 CTL 子公式的方法，实现了模型检查，并通过找到强连通分量的方法找到公平路径。

标记满足 CTL 公式结点的时间复杂度约为$O(N*F*P)$,其中N为结点个数，F为 CTL 公式长度,P为子公式个数;找到强连通分量的时间复杂度为$O(M+N)$，M为弧的个数，N为结点的个数；求解公平路径的算法使用了 BFS 的思想，实践复杂度仍为$O(M+N)$，M为弧的个数，N为结点的个数。

对于标记满足 CTL 公式算法，我们可以设计更优的算法来进行优化，进一步缩短时间复杂度。

本小组在三天内经历了项目论文阅读、项目内容讨论、项目需求分析、实现思路梳理和项目代码实现的过程，通过一个基于 C++语言的拓展模型检测工具的设计和实现，经历了整个软件开发的过程，包括系统分析、概要设计、详细设计、编码、测试。

值得一提的是，在确定实现思路之后，小组内分工明确，思路清晰，各自进行负责模块的编程实现。整体编程实现过程仅使用了 36 小时。这也使我们更为深刻地理解了软件开发的过程中，明确需求分析和设计思路的重要性。

放眼于生活，生活也如软件开发的流程一般，只有确立好明确的目标，才能使得前行的脚步更加坚定与轻快。不然，方向不对，努力白费！

最后，感谢郭建老师给予我们小组如此宝贵的机会，在 2020 这么特殊的一年结束之际，让我们得以锻炼阅读文献、动手编程和团队协作的能力，迎来崭新的 2021。