NuSMV-Project-01

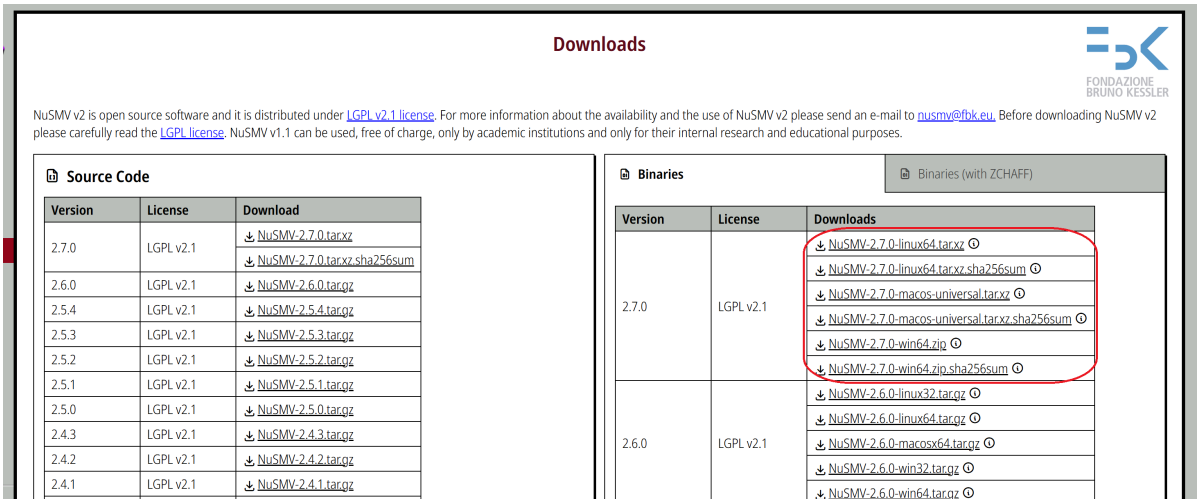
本仓库是《软件工程理论基础》项目一代码和报告。

`./proj1.pdf` 是项目要求，本仓库完成了项目要求中的三个题目。

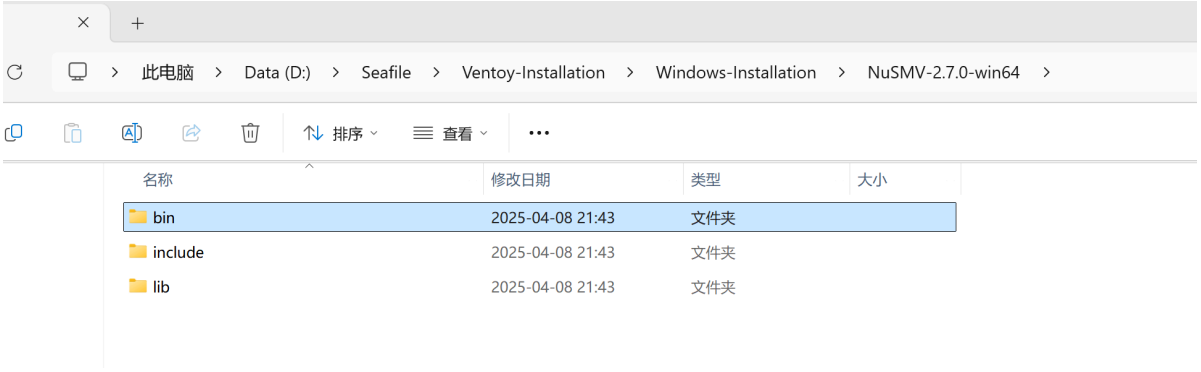
【广告】除了 NuSMV 之外，软件工程有另一个形式化验证工具 TLA+。该工具由图灵奖获得者 Leslie Lamport 开发（$\LaTeX$ 中的 La 就是指 Lamport）。我的 b 站主页搬运了 Lamport 本人录制的教程 The TLA+ Video Course，仅供学习。

1. NuSMV 的安装和配置

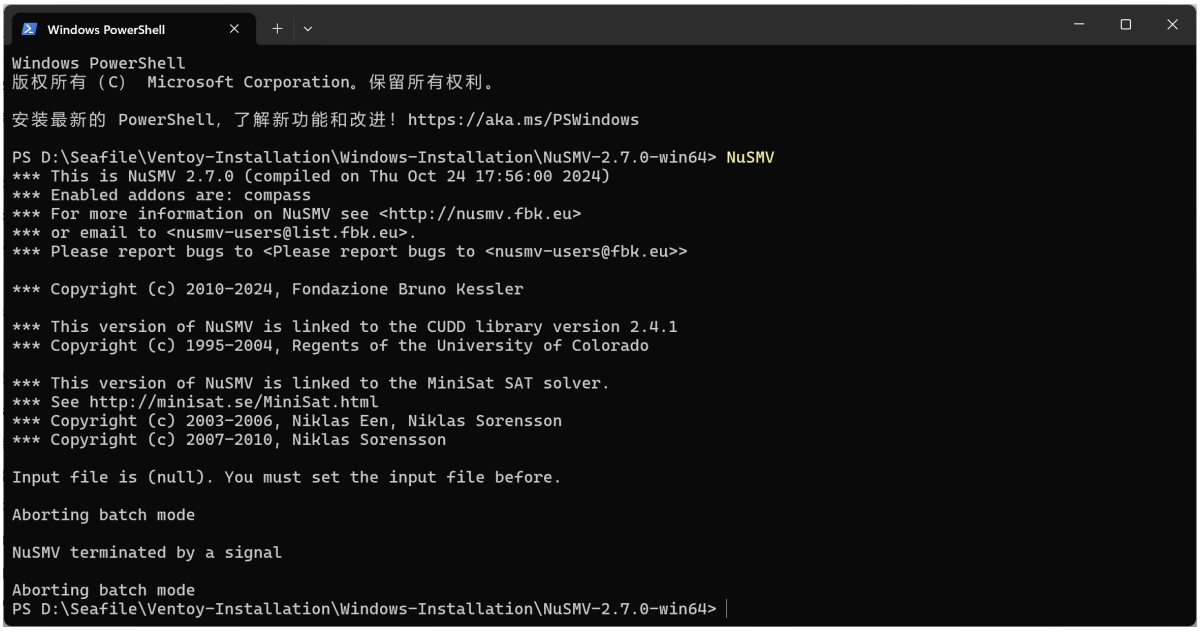
在 NuSMV - Downloads 下载对应版本，推荐直接下载右侧红框中对应的二进制文件。



以 Windows 系统为例，下载到本地后解压到任意目录。



将 `NuSMV-2.7.0-win64/bin` 的路径添加到系统环境变量后，即可在命令行使用 NuSMV 命令。



使用下列命令可以运行所需的 NuSMV 源文件。

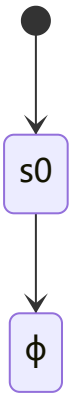
```
NuSMV -df filename.smv
```

在 VSCode 中安装 coderunner 插件，并在配置文件中添加 `".smv": "cd $dir && NuSMV -df $fileName"` 即可使用快捷键 `Ctrl+Alt+N` 运行 `.smv` 文件。

2. 公式等价性判断

结论：a, c 和 d 中两个公式不等价，b 中两个公式等价。

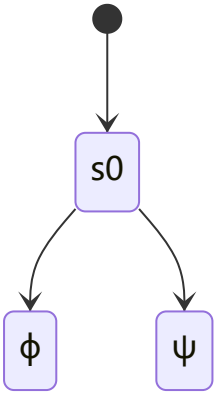
`./formula/fomula_a.smv` 给出了 a 的一个反例：



在上述反例中：

```
CTLSPEC EF(phi);  -- 结果为 TRUE
CTLSPEC EG(phi);  -- 结果为 FALSE
```

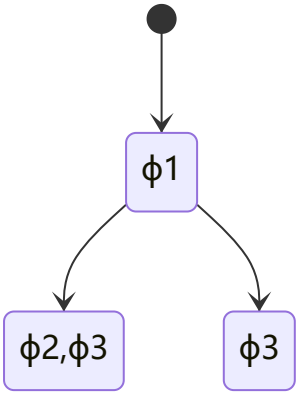
`./formula/fomula_c.smv` 给出了 c 的一个反例：



在上述反例中：

```
CTLSPEC AF(phi);  -- 结果为 TRUE
CTLSPEC AF(psi);  -- 结果为 FALSE
```

`./formula/fomula_d.smv` 给出了 d 的一个反例：



在上述反例中：

```
CTLSPEC A[phi1 U A[phi2 U phi3]];  -- 结果为 TRUE
CTLSPEC A[A[phi1 U phi2] U phi3];  -- 结果为 FALSE
```

3. 冒泡排序建模

`./bubble_sort/bubble_sort.smv` 使用 SMV 代码建模冒泡排序算法，并验证是否完成排序。

基本思路是，根据 `j` 的取值来实现交换逻辑，例如下面代码片段。值得注意，NuSMV 是同步赋值，因此 `a0` 和 `a1` 的更新基于当前状态的值，下面写法中的变量交换是原子的。

```
-- 交换逻辑
next(a0) := case
  -- 比较 a[j] 和 a[j+1]，若需要交换则更新
  j = 0 & a0 > a1 : a1;  -- 交换 a0 和 a1
  TRUE : a0;
esac;

next(a1) := case
  j = 0 & a0 > a1 : a0;  -- 交换 a0 和 a1
  j = 1 & a1 > a2 : a2;  -- 交换 a1 和 a2
  TRUE : a1;
esac;

next(a2) := case
  j = 1 & a1 > a2 : a1;  -- 交换 a1 和 a2
  TRUE : a2;
esac;
```

使用以下规约验证排序结果：

```
CTLSPEC AG( (i = 2 & j = 0 & !swapped) -> (a0 <= a1 & a1 <= a2) ) -- 结果为 TRUE
```

`./bubble_sort/bubble_sort_c.smv` 是改进的算法，使用枚举避免整数域运算，并验证了算法的正确性。由于题目限制了变量的取值范围为 `0..7`，因此可以用三个比特（布尔值）来表达一个整数，进而用布尔（逻辑）运算来代替整数运算。

```
DEFINE
  -- 定义比较逻辑（使用位比较代替数值比较）
  greater_a0_a1 := (a0_b2 & !a1_b2) |
                   ((a0_b2 = a1_b2) & (a0_b1 & !a1_b1)) |
                   ((a0_b2 = a1_b2) & (a0_b1 = a1_b1) & (a0_b0 & !a1_b0));

  greater_a1_a2 := (a1_b2 & !a2_b2) |
                   ((a1_b2 = a2_b2) & (a1_b1 & !a2_b1)) |
                   ((a1_b2 = a2_b2) & (a1_b1 = a2_b1) & (a1_b0 & !a2_b0));
```

使用以下规约验证排序结果：

```
CTLSPEC AG( (i = 2 & j = 0 & !swapped) -> (!greater_a0_a1 & !greater_a1_a2) ) -- 结果为 TRUE
```

4. Raft 算法建模

`./raft/raft.smv` 使用 NuSMV 对 Raft 算法进行建模，并验证了其正确性。

对每个定时器制定相同的计时规则：

```
-- 定时器更新逻辑，每次增加 1，模拟时间流逝
next(timer1) := case
  node1 = Leader: 0;  -- 如果当前是 Leader，定时器归零
  node1 = Follower: min(timer1 + 1, 10);  -- Follower 每次增加 1，但不超过 10
  node1 = Candidate: min(timer1 + 1, 10);  -- Candidate 每次增加 1，但不超过 10
esac;

next(timer2) := case
  node2 = Leader: 0;  -- 如果当前是 Leader，定时器归零
  node2 = Follower: min(timer2 + 1, 10);  -- Follower 每次增加 1，但不超过 10
  node2 = Candidate: min(timer2 + 1, 10);  -- Candidate 每次增加 1，但不超过 10
esac;

next(timer3) := case
  node3 = Leader: 0;  -- 如果当前是 Leader，定时器归零
  node3 = Follower: min(timer3 + 1, 10);  -- Follower 每次增加 1，但不超过 10
  node3 = Candidate: min(timer3 + 1, 10);  -- Candidate 每次增加 1，但不超过 10
esac;
```

根据 Raft 协议规定，描述节点之间的状态转移关系：

```
-- 状态转移：若定时器超时，则成为 Candidate，若选举成功，则成为 Leader
next(node1) := case
  timer1 >= 5: Candidate;  -- 如果定时器超过 5，则成为 Candidate
  node1 = Candidate & timer1 < 5: Follower;  -- 如果处于 Candidate 状态，未超时则变回 Follower
  node1 = Leader: Leader;  -- 如果是 Leader，则保持 Leader
  TRUE: node1;  -- 否则保持当前状态
esac;

next(node2) := case
  timer2 >= 5: Candidate;  -- 如果定时器超过 5，则成为 Candidate
  node2 = Candidate & timer2 < 5: Follower;  -- 如果处于 Candidate 状态，未超时则变回 Follower
  node2 = Leader: Leader;  -- 如果是 Leader，则保持 Leader
  TRUE: node2;  -- 否则保持当前状态
esac;

next(node3) := case
  timer3 >= 5: Candidate;  -- 如果定时器超过 5，则成为 Candidate
  node3 = Candidate & timer3 < 5: Follower;  -- 如果处于 Candidate 状态，未超时则变回 Follower
  node3 = Leader: Leader;  -- 如果是 Leader，则保持 Leader
  TRUE: node3;  -- 否则保持当前状态
esac;
```

本项目实现了三个节点的 Raft 协议，可以轻松扩展到多个节点。

使用以下规约验证协议一致性和安全性：

```
-- 规约 1：三个节点都可能成为 Leader
CTLSPEC AG (node1 = Leader -> EF (node2 = Leader | node3 = Leader)) -- 结果为 TRUE

-- 规约 2：不会出现多个 Leader
CTLSPEC AG (node1 = Leader & node2 = Leader -> FALSE) -- 结果为 TRUE
CTLSPEC AG (node1 = Leader & node3 = Leader -> FALSE) -- 结果为 TRUE
CTLSPEC AG (node2 = Leader & node3 = Leader -> FALSE) -- 结果为 TRUE

-- 规约 3：可能有多个节点同时想成为 Leader
CTLSPEC EF (node1 = Candidate & node2 = Candidate) -- 结果为 TRUE
CTLSPEC EF (node2 = Candidate & node3 = Candidate) -- 结果为 TRUE
CTLSPEC EF (node1 = Candidate & node3 = Candidate) -- 结果为 TRUE
CTLSPEC EF (node1 = Candidate & node2 = Candidate & node3 = Candidate) -- 结果为 TRUE
```