《软件工程理论基础》项目一报告

1. 等价性判断

a 和 d 中两个公式不等价，b 和 c 中两个公式等价。

1.1 验证 a

```
MODULE main
VAR
  state : {s0, s1};
ASSIGN
  init(state) := s0;
  next(state) := case
    state = s0 : s1;
    state = s1 : s1;
  esac;
DEFINE
  phi := (state = s1);
CTLSPEC EF(phi);  -- 结果为 TRUE
CTLSPEC EG(phi);  -- 结果为 FALSE
```

1.2 验证 d

```
MODULE main
VAR
  state : {s0, s1, s2};
ASSIGN
  init(state) := s0;
  next(state) := case
    state = s0 : {s1, s2};
    state = s1 : s1;
    state = s2 : s2;
  esac;
DEFINE
  phi1 := (state = s0);
  phi2 := (state = s1);
  phi3 := (state = s1) | (state = s2);
CTLSPEC A[phi1 U A[phi2 U phi3]];  -- 结果为 TRUE
CTLSPEC A[A[phi1 U phi2] U phi3];  -- 结果为 FALSE
```

2. 冒泡排序建模

2.1 算法建模和验证

使用 SMV 代码建模冒泡排序算法，并验证是否完成排序：

```
MODULE main
VAR
  a0 : 0..7;  -- 数组元素 0
  a1 : 0..7;  -- 数组元素 1
  a2 : 0..7;  -- 数组元素 2
  i : 0..2;   -- 外层循环变量
  j : 0..2;   -- 内层循环变量
  swapped : boolean;  -- 交换标志

ASSIGN
  -- 初始化数组和循环变量
  init(a0) := 0..7;  -- 初始值任意
  init(a1) := 0..7;
  init(a2) := 0..7;
  init(i) := 0;
  init(j) := 0;
  init(swapped) := FALSE;

  -- 外层循环控制
  next(i) := case
    -- 内层循环完成且未提前退出，递增 i
    j >= (2 - i -1) & swapped : (i < 2 ? i + 1 : i);
    TRUE : i;
  esac;

  -- 内层循环控制
  next(j) := case
    -- 内层循环未完成，递增 j
    j < (2 - i -1) : j + 1;
    -- 重置 j 以开始下一轮外层循环
    j >= (2 - i -1) : 0;
    TRUE : j;
  esac;

  -- 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;

  -- 更新 swapped 标志
  next(swapped) := case
    (j = 0 & a0 > a1) | (j = 1 & a1 > a2) : TRUE;  -- 发生交换
    j >= (2 - i -1) : FALSE;  -- 重置为 FALSE 以开始下一轮
    TRUE : swapped;
  esac;
```

```
59
60 CTLSPEC AG( (i = 2 & j = 0 & !swapped) -> (a0 <= a1 & a1 <= a2) ) --验证是否完成排序，结果为 TRUE
```

2.2 使用枚举避免整数域运算

```
MODULE main
VAR
  a0_b2 : boolean;
  a0_b1 : boolean;
  a0_b0 : boolean;
  a1_b2 : boolean;
  a1_b1 : boolean;
  a1_b0 : boolean;
  a2_b2 : boolean;
  a2_b1 : boolean;
  a2_b0 : boolean;
  i : {0, 1, 2};
  j : {0, 1};
  swapped : boolean;

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));

ASSIGN
  -- 初始化数组元素（非确定性初始状态）
  init(a0_b2) := {TRUE, FALSE};
  init(a0_b1) := {TRUE, FALSE};
  init(a0_b0) := {TRUE, FALSE};
  init(a1_b2) := {TRUE, FALSE};
  init(a1_b1) := {TRUE, FALSE};
  init(a1_b0) := {TRUE, FALSE};
  init(a2_b2) := {TRUE, FALSE};
  init(a2_b1) := {TRUE, FALSE};
  init(a2_b0) := {TRUE, FALSE};

  -- 初始化控制变量
  init(i) := 0;
  init(j) := 0;
  init(swapped) := FALSE;

  -- 外层循环控制逻辑
  next(i) := case
    (i = 0 & j = 1 & swapped) : 1;  -- 完成第一轮冒泡
    (i = 1 & j = 0 & swapped) : 2;  -- 完成第二轮冒泡
    TRUE : i;
  esac;

  -- 内层循环控制逻辑
  next(j) := case
    (i = 0 & j = 0) : 1;  -- 第一轮冒泡比较两次
    (i = 0 & j = 1) : 0;  -- 重置内层循环
    (i = 1 & j = 0) : 0;  -- 第二轮冒泡比较一次
    TRUE : j;
  esac;

  -- 数组元素交换逻辑
```

```
  next(a0_b2) := case
    (j = 0 & greater_a0_a1) : a1_b2;  -- 交换高位
    TRUE : a0_b2;
  esac;
  next(a0_b1) := case
    (j = 0 & greater_a0_a1) : a1_b1;  -- 交换中位
    TRUE : a0_b1;
  esac;
  next(a0_b0) := case
    (j = 0 & greater_a0_a1) : a1_b0;  -- 交换低位
    TRUE : a0_b0;
  esac;

  next(a1_b2) := case
    (j = 0 & greater_a0_a1) : a0_b2;  -- 反向交换高位
    (j = 1 & greater_a1_a2) : a2_b2;  -- 交换高位
    TRUE : a1_b2;
  esac;
  next(a1_b1) := case
    (j = 0 & greater_a0_a1) : a0_b1;  -- 反向交换中位
    (j = 1 & greater_a1_a2) : a2_b1;  -- 交换中位
    TRUE : a1_b1;
  esac;
  next(a1_b0) := case
    (j = 0 & greater_a0_a1) : a0_b0;  -- 反向交换低位
    (j = 1 & greater_a1_a2) : a2_b0;  -- 交换低位
    TRUE : a1_b0;
  esac;

  next(a2_b2) := case
    (j = 1 & greater_a1_a2) : a1_b2;  -- 反向交换高位
    TRUE : a2_b2;
  esac;
  next(a2_b1) := case
    (j = 1 & greater_a1_a2) : a1_b1;  -- 反向交换中位
    TRUE : a2_b1;
  esac;
  next(a2_b0) := case
    (j = 1 & greater_a1_a2) : a1_b0;  -- 反向交换低位
    TRUE : a2_b0;
  esac;

  -- 交换标志更新逻辑
  next(swapped) := case
    (j = 0 & greater_a0_a1) | (j = 1 & greater_a1_a2) : TRUE;
    (i = 0 & j = 1) | (i = 1 & j = 0) : FALSE;  -- 每轮结束重置标志
    TRUE : swapped;
  esac;

-- 验证排序正确性的CTL规范
CTLSPEC AG( (i = 2 & j = 0 & !swapped) -> (!greater_a0_a1 & !greater_a1_a2) ) -- 结果为 TRUE
```

3. Raft 算法建模

Raft 使用 NuSMV 建模结果和运行结果如下：

```
MODULE main
VAR
  node1: {Follower, Candidate, Leader};
  node2: {Follower, Candidate, Leader};
  node3: {Follower, Candidate, Leader};
  timer1: 0..10; -- 模拟超时定时器，超时会变为 Candidate
  timer2: 0..10;
  timer3: 0..10;

ASSIGN
  init(node1) := Follower;  -- 节点1初始状态为 Follower
  init(node2) := Follower;  -- 节点2初始状态为 Follower
  init(node3) := Follower;  -- 节点3初始状态为 Follower
  init(timer1) := 0; -- 初始化定时器为 0
  init(timer2) := 0;
  init(timer3) := 0;

  -- 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;

  -- 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;

```

```
-- 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
```