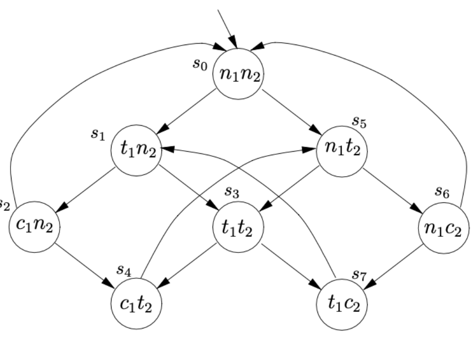
形式化方法导引 Lab2

PB20111686 黄瑞轩

需要实现的模型

A first-attempt model:



建模尝试

这个问题中有两个实体（进程），分别命名为 $p1, p2$，其状态空间为 $\{n, t, c\}$，以 $p1$ 为例，其初始化及转移逻辑应如下所示：

```
ASSIGN
    init(p1.s) := n;
    next(p1.s) :=
        case
            p1.s = n : t;
            p1.s = t & p2.s != c : c;
            p1.s = t & p2.s = c : t;
            p1.s = c : n;
        esac;
```

但如果直接按此逻辑声明两个进程模块，则会导致并发问题，即某一时刻二者同时行动，比如都为 t，此时双方都满足 $pi.s = t \: \& \: pj.s \neq c$，二者就可能进入 $p1.s = p2.s = c$ 的状态，导致 safety 不满足。

```
-- specification AG !(p1 = c & p2 = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    p1 = n
    p2 = n
  -> State: 1.2 <-
    p1 = t
    p2 = t
  -> State: 1.3 <-
    p1 = c
    p2 = c
```

经搜寻，发现将变量声明成 process 可以保证每次只有一个变量活跃，避免上面的问题。并且需要给将被声明成 process 的 proc 模块添加 `FAIRNESS running` 保证不卡死。整个模块的结构如下所示：

```
MODULE proc(another_s)
    VAR
        s : { n, t, c };
    ASSIGN
        -- init & next definition
    FAIRNESS
        running

MODULE main
    VAR
        p1 : process proc(p2.s);
        p2 : process proc(p1.s);

    -- CTLSPECs
```

这样写后，发现 process 产生了自环，很难消除，最终决定使用翻译状态的方法：

```
MODULE main
    VAR
        s : { s0, s1, s2, s3, s4, s5, s6, s7 };

    ASSIGN
        init(s) := s0;
        next(s) := case
            s = s0 : { s1, s5 };
            s = s1 : { s2, s3 };
            s = s2 : { s0, s4 };
            s = s3 : { s4, s7 };
            s = s4 : s5;
            s = s5 : { s3, s6 };
            s = s6 : { s0, s7 };
            s = s7 : s1;
        esac;

    DEFINE
        n1 := s = s0 | s = s5 | s = s6;
        t1 := s = s1 | s = s3 | s = s7;
        c1 := s = s2 | s = s4;
        n2 := s = s0 | s = s1 | s = s2;
        t2 := s = s5 | s = s3 | s = s4;
        c2 := s = s6 | s = s7;
```

这样可以保证建模与要求完全一致。

四个性质的 CTL 设计及验证

Safety

- $AG(\neg(c_1 \wedge c_2))$
- `AG(!(c1 & c2))`

验证结果:

```
-- specification AG !(c1 & c2)  is true
```

Liveness

- $AG(t_1 \to AF\ c_1)$
- $AG(t_2 \to AF\ c_2)$
- `AG(t1 -> AF(c1))`
- `AG(t2 -> AF(c2))`

验证结果:

- 两个都是 FALSE，为避免冗长这里只展示前一个的结果
- 循环导致不能满足 Liveness

```
-- specification AG (t1 -> AF c1)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    s = s0
    c2 = FALSE
    t2 = FALSE
    n2 = TRUE
    c1 = FALSE
    t1 = FALSE
    n1 = TRUE
  -- Loop starts here
  -> State: 1.2 <-
    s = s1
    t1 = TRUE
    n1 = FALSE
  -> State: 1.3 <-
    s = s3
    t2 = TRUE
    n2 = FALSE
  -> State: 1.4 <-
    s = s7
    c2 = TRUE
    t2 = FALSE
  -> State: 1.5 <-
    s = s1
    c2 = FALSE
    n2 = TRUE
```

Non-blocking

- $AG(n_1 \rightarrow EF\ t_1)$
- $AG(n_2 \rightarrow EF\ t_2)$
- `AG(n1 -> EF(t1))`
- `AG(n2 -> EF(t2))`

验证结果：

```
-- specification AG (n1 -> EF t1)  is true
-- specification AG (n2 -> EF t2)  is true
```

No strict sequencing

- $EF(c_1 \wedge E[c_1 U(\neg c_1 \wedge E[\neg c_2 U c_1])])$
- $EF(c_2 \wedge E[c_2 U(\neg c_2 \wedge E[\neg c_1 U c_2])])$
- `EF(c1 & E[c1 U (!c1 & E[(!c2) U c1])])`
- `EF(c2 & E[c2 U (!c2 & E[(!c1) U c2])])`

验证结果：

```
-- specification EF (c1 & E [ c1 U (!c1 & E [ !c2 U c1 ] ) ] )  is true
-- specification EF (c2 & E [ c2 U (!c2 & E [ !c1 U c2 ] ) ] )  is true
```

建议

- NuSMV 最新版本已经将 process 标记为 *deprecated*，助教和老师可以关注下实验要求的更新

- 建议使用一些支持可视化的工具来做实验，以便验证建模是否与要求相符
- 建议助教在实验文档中提供 NuSMV 的二进制分发