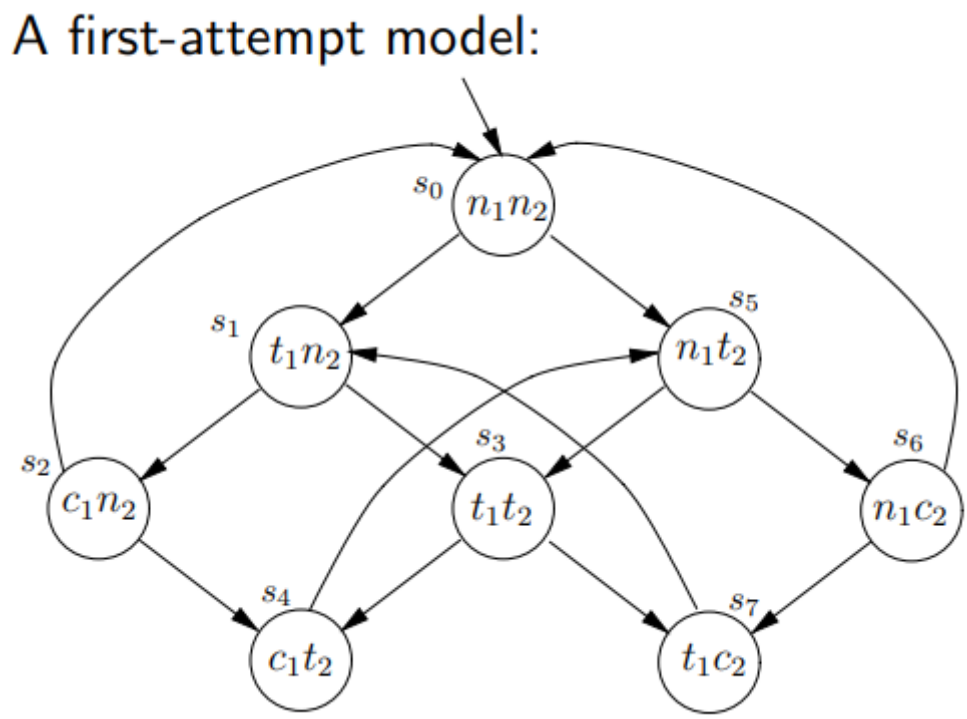
实验小作业2-作业7

梁峻滔 PB19051175

1. NuSMV建立模型

根据 first-attempt 的状态转移图



首先建立两个进程的模型, 进程模型需要且仅需要另一个进程的status作为参数:

```
MODULE prc(other-st)
    VAR
        st: {n, t, c}; --一个进程的status有三种
    ASSIGN
        init(st) := n; --一个进程的初始状态为n
        next(st) :=
            case
                (st = n) : {t};
                (st = t) & ((other-st = n) | (other-st = t)) : {c};
                (st = c) : n;
                TRUE : st;
            esac;
```

需要说明的是, 在 first-attempt 中是不支持状态停留的, 在NuSMV中每次状态转移是选定一个进程, 令其status变化, 此时另一个进程的status是保持不变的, 所以在进程的MODULE定义中就不需要(或者说不能)像 3-rd attempt 那样可以让一个进程的status不变. 具体地说就是, 当一个状态在main中被选中时, 如果它当前status是**n(t, c)**, 那它的下一个status就一定要是**t(c, n)**, 保持status不变的唯一情况是该次状态转移是另一个进程被选中.

有一个地方值得注意的是, 一个进程的 status 可以从 **t** 转移到 **c**, 当且仅当另一个进程的 status 不是 **c**, 这是为了满足 **Safety** 性质.

main模型主要工作是例化两个进程模块和进行LTL/CTL公式的描述:

```
MODULE main
    VAR
        pr1: process prc(pr2.st);
        pr2: process prc(pr1.st);
    -- LTLSPEC
        -- G !(pr1.st = c & pr2.st = c) -- Safety
        -- G (pr1.st = t -> F pr1.st = c) -- Liveness
    CTLSPEC
        -- AG (pr1.st = n -> EX pr1.st = t) -- Non-blocking
        AG (pr1.st = c -> E[ !(pr2.st = c) U pr1.st = c ]) -- No strict sequencing
```

2. 四个性质的LTL/CTL描述及其验证

- Safety

  ```
  LTLSPEC
      G !(pr1.st = c & pr2.st = c)
  ```

  表示从初始状态出发的所有路径上都不会出现进程1和进程2同时进入临界区的状态. 使用 NuSMV 验证结果为true:

  ```
  -- specification  G !(pr1.st = c & pr2.st = c)  is true
  ```

  与课堂上所讨论的结果一致

- Liveness

  ```
  LTLSPEC
      G (pr1.st = t -> F pr1.st = c)
  ```

  表示任意从 pr1.st = t 的状态出发的路径上都会在之后某个时刻满足 pr1.st = c. 使用 NuSMV 验证结果为false:

```
-- specification  G (pr1.st = t ->  F pr1.st = c)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    pr1.st = n
    pr2.st = n
  -> Input: 1.2 <-
    _process_selector_ = pr1
    running = FALSE
    pr2.running = FALSE
    pr1.running = TRUE
  -> State: 1.2 <-
    pr1.st = t
  -> Input: 1.3 <-
    _process_selector_ = pr2
    pr2.running = TRUE
    pr1.running = FALSE
  -- Loop starts here
  -> State: 1.3 <-
    pr2.st = t
  -> Input: 1.4 <-
    _process_selector_ = main
    running = TRUE
    pr2.running = FALSE
  -- Loop starts here
  -> State: 1.4 <-
  -> Input: 1.5 <-
    _process_selector_ = pr2
    running = FALSE
    pr2.running = TRUE
  -> State: 1.5 <-
    pr2.st = c
  -> Input: 1.6 <-
  -> State: 1.6 <-
    pr2.st = n
  -> Input: 1.7 <-
  -> State: 1.7 <-
    pr2.st = t
```

原因是存在一个pr2.st 不断在 **n, t, c** 之间切换而 pr1.st 保持为 **t** 的循环, 与课堂上所讨论的结果一致.

- Non-blocking

```
CTLSPEC
    AG (pr1.st = n -> EX pr1.st = t)
```

该 CTL 描述表示对于 CTL 计算树中所有路径, 路径上所有满足 pr1.st = n 的状态, 都存在一条从该状态出发的路径, 该路径上的下一个状态满足 pr1.st = t. 该描述根据课件`5.1.4f`中"for every state satisfying n1, there is a successor satisfying t1"来设计.

使用 NuSMV 验证结果为true:

```
-- specification AG (pr1.st = n -> EX pr1.st = t)  is true
```

事实上也确实满足.

- No strict sequencing

```
CTLSPEC
    AG (pr1.st = c -> E[ !(pr2.st = c) U pr1.st = c ])
```

该 CTL 描述表示对于 CTL 计算树中所有路径, 路径上所有满足 pr1.st = c 的节点, 都存在这样一条后续路径: 该路径上直到 pr1.st = c 满足前(且pr1.st = c一定会在某个时刻满足), 都不满足 pr2.st = c. 意思就是进程1可以连续两次进入临界状态而不必一定要与进程2轮换着进入临界区.

使用 NuSMV 验证结果为true:

```
-- specification AG (pr1.st = c -> E [ !(pr2.st = c) U pr1.st = c ] )  is true
```

与课堂上使用LTL验证的结果一致.