

形式化方法导引

第 5 章 模型检测

5.1 应用 – 5.1.4 NuSMV

黄文超

<http://staff.ustc.edu.cn/~huangwc/fm.html>

1. 应用

1.4 Verification by NuSMV | Introduction

NuSMV (sometimes called simply SMV): A language for

- describing models
- specifying LTL / CTL formulas, etc.
- check the validity of the formulas on the models

The output of model checking

- True, if the specifications holds
- a trace, otherwise

└ 1. 应用

NuSMV (sometimes called simply SMV): A language for

- describing models
- specifying LTL / CTL formulas, etc.
- check the validity of the formulas on the models

The output of model checking

- True, if the specifications holds
- a trace, otherwise

NuSMV (sometimes called simply SMV) provides a language for describing the models we have been drawing as diagrams and it directly checks the validity of LTL (and also CTL) formulas on those models. SMV takes as input a text consisting of a program describing a model and some specifications (temporal logic formulas). It produces as output either the word ‘true’ if the specifications hold, or a trace showing why the specification is false for the model represented by our program.

1. 应用

1.4 Verification by NuSMV | Introduction

```
MODULE main
  VAR
    request : boolean;
    status : {ready,busy};
  ASSIGN
    init(status) := ready;
    next(status) := case
      request : busy;
      TRUE : {ready,busy};
    esac;

  LTLSPEC
    G(request -> F status=busy)
```

NuSMV v.s. C, Java...

- In Common
 - one, or more modules
 - main
- Differences
 - state transition
 - specification
 - non-deterministic
 - abstraction

└ 1. 应用

1. 应用

1.4 Verification by NuSMV | Introduction

```

MODULE main
VAR
  request : boolean;
  status : {ready,busy};
ASSIGN
  init(status) := ready;
  next(status) := case
    request : busy;
    TRUE : {ready,busy};
  esac;
LTLSPEC
  G(request -> F status=busy)

```

NuSMV v.s. C, Java, ...

- In Common
 - one, or more modules
 - main
- Differences
 - state transition
 - specification
 - non-determinism
 - abstraction

SMV programs consist of one or more modules. As in the programming language C, or Java, one of the modules must be called main. Modules can declare variables and assign to them. Assignments usually give the initial value of a variable and its next value as an expression in terms of the current values of variables. This expression can be non-deterministic (denoted by several expressions in braces, or no assignment at all). Non-determinism is used to model the environment and for abstraction.

The SMV consists of a program and a specification. The program has two variables, request of type boolean and status of enumeration type ready, busy: 0 denotes “false” and 1 represents “true”. The initial and subsequent values of variable request are not determined within this program; this conservatively models that these values are determined by an external environment. This under-specification of request implies that the value of variable status is partially determined: initially, it is ready; and it becomes busy whenever request is true. If request is false, the next value of status is not determined.

1. 应用

1.4 Verification by NuSMV | Introduction

```
MODULE main
```

```
VAR
```

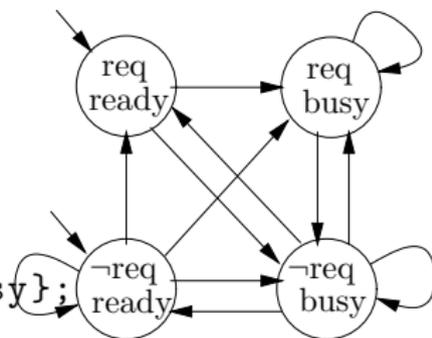
```
  request : boolean;  
  status  : {ready, busy};
```

```
ASSIGN
```

```
  init(status) := ready;  
  next(status) := case  
    request : busy;  
    TRUE   : {ready, busy};  
  esac;
```

```
LTLSPEC
```

```
  G(request -> F status=busy)
```



运行结果: \$./NuSMV c-sample1.smv

- specification G (request -> F status = busy) is true

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

例: Mutual exclusion (互斥, 操作系统经典问题)

- 回顾: 关键词: critical sections (临界区)
- 回顾: 需求: Only one process can be in its critical section at a time
- 问题: to *find a protocol* for determining which process is allowed to enter its critical section at which time

如何利用 NuSMV 求解上述问题:

- ① specify the properties of the protocol using NuSMV
- ② design a protocol
- ③ model the protocol using NuSMV
- ④ check the output of NuSMV
 - if true, problem solved
 - if not, goto step 2

└ 1. 应用

例: Mutual exclusion (互斥, 操作系统经典问题)

- ◆ 回顾: 关键词: critical sections (临界区)
- ◆ 回顾: 需求: Only one process can be in its critical section at a time
- ◆ 问题: to find a protocol for determining which process is allowed to enter its critical section at which time

如何利用 NuSMV 求解上述问题:

- 1 specify the properties of the protocol using NuSMV
- 2 design a protocol
- 3 model the protocol using NuSMV
- 4 check the output of NuSMV
 - ◆ if true: problem solved
 - ◆ if not: goto step 2

Let us now look at a larger example of verification using LTL, having to do with *mutual exclusion*. When concurrent processes share a resource (such as a file on a disk or a database entry), it may be necessary to ensure that they do not have access to it at the same time. Several processes simultaneously editing the same file would not be desirable.

We therefore identify certain *critical sections* of each process' code and arrange that only one process can be in its critical section at a time. The critical section should include all the access to the shared resource (though it should be as small as possible so that no unnecessary exclusion takes place). The problem we are faced with is to find a protocol for determining which process is allowed to enter its critical section at which time. Once we have found one which we think works, we verify our solution by checking that it has some expected properties, such as the following ones.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

1. Specify the properties of the protocol using NuSMV

- *Safety*: Only *one process* is in its *critical section* at any time.
- *Liveness*: Whenever any process *requests* to enter its critical section, it will *eventually be permitted* to do so.
- *Non-blocking*: A process can *always* request to *enter* its critical section.
- *No strict sequencing*: Processes need not enter their critical section in strict sequence.

└ 1. 应用

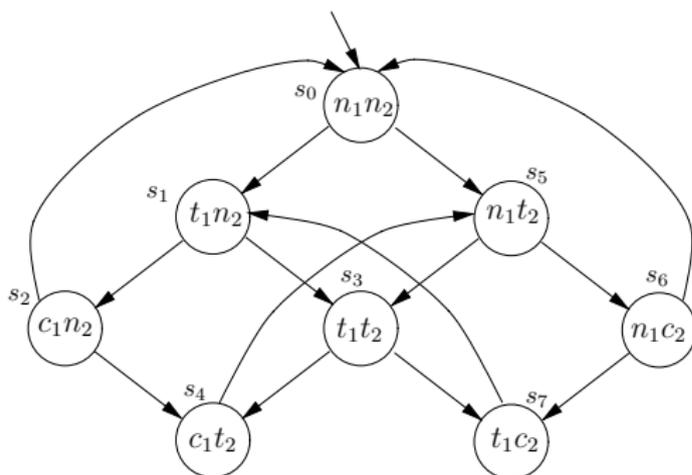
Some rather crude protocols might work on the basis that they cycle through the processes, making each one in turn enter its critical section. Since it might be naturally the case that some of them request access to the shared resource more often than others, we should make sure our protocol has the property

1. Specify the properties of the protocol using NuSMV
 - ◆ **Safety**: Only *one process* is in its *critical section* at any time.
 - ◆ **Liveness**: Whenever any process *requests* to enter its critical section, it will eventually *be permitted* to do so.
 - ◆ **Non-blocking**: A process can *always* request to *enter* its critical section.
 - ◆ **No strict sequencing**: Processes need not enter their critical section in strict sequence.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

A first-attempt model:



processes

- 1,2

states

- n : in its *non-critical* state
- t : *trying* to enter its *critical* state
- c : in its *critical* state

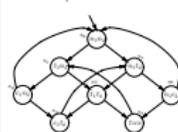
state transitions

- $n_i \rightarrow t_i \rightarrow c_i \rightarrow n_i \dots$

问题: Is the model correct?

└ 1. 应用

A first-attempt model:



processes

• p1,2

states

• n: in its non-critical state

• t: trying to enter its critical state

• c: in its critical state

state transitions

• $n_i \rightarrow t_i \rightarrow c_i \rightarrow n_i \dots$

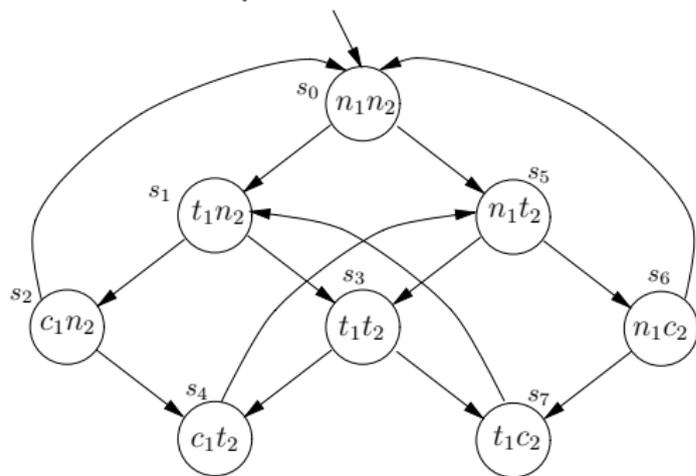
问题: Is the model correct?

State s_0 is the only initial state, indicated by the incoming edge with no source. Either of them may now move to its trying state, but only one of them can ever make a transition at a time (*asynchronous interleaving*). At each step, an (unspecified) scheduler determines which process may run. So there is a transition arrow from s_0 to s_1 and s_5 . From s_1 (i.e., process 1 trying, process 2 non-critical) again two things can happen: either process 1 moves again (we go to s_2), or process 2 moves (we go to s_3). Notice that not every process can move in every state. For example, process 1 cannot move in state s_7 , since it cannot go into its critical section until process 2 comes out of its critical section.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

A first-attempt model:



Safety: Only *one process* is in its *critical section* at any time.

- LTL specification:

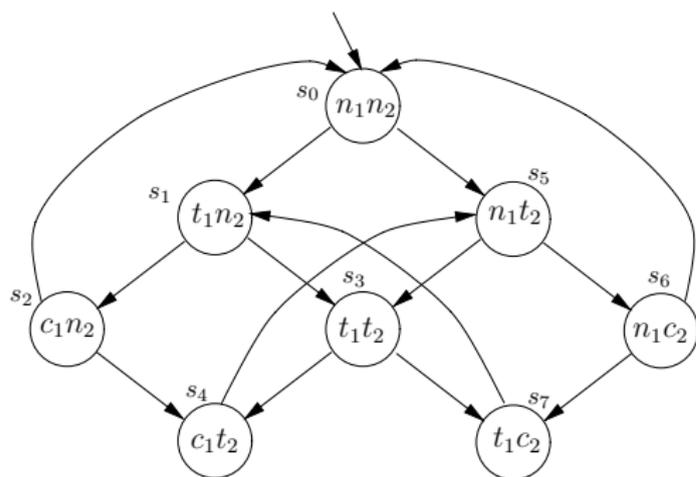
$$G\neg(c_1 \wedge c_2)$$

- Satisfied

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

A first-attempt model:



Liveness: Whenever any process *requests* to enter its critical section, it will *eventually be permitted* to do so.

- LTL specification:

$$G (t_1 \rightarrow F c_1)$$

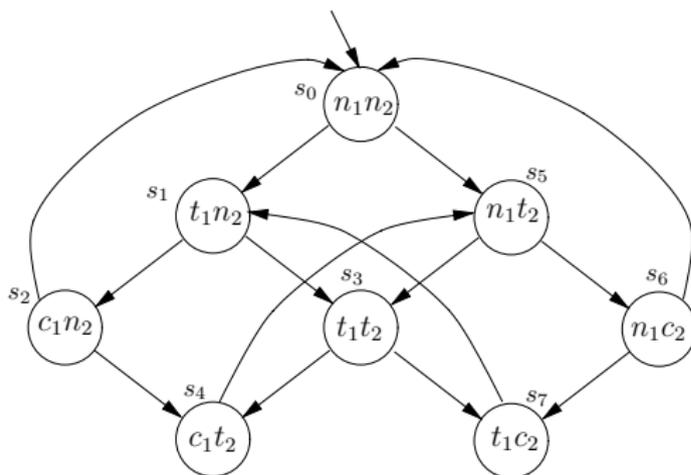
- Not Satisfied

$$s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \dots$$

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

A first-attempt model:



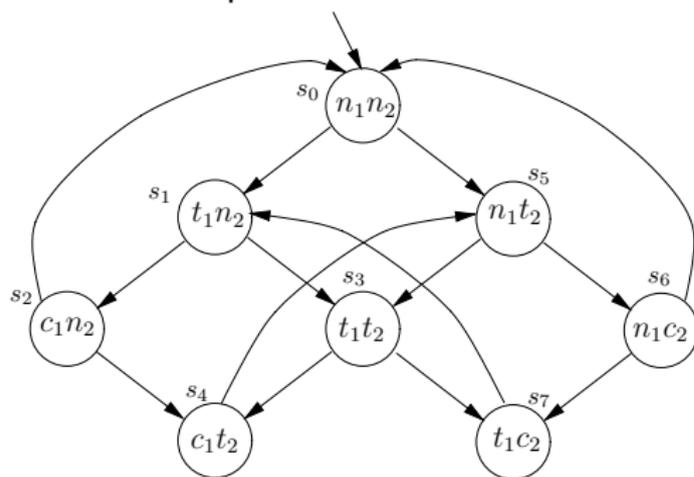
Non-blocking: A process can *always* request to *enter* its critical section.

- In other words, for every state satisfying n_1 , there is a successor satisfying t_1 .
- LTL specification? No
- CTL specification: ??
实验小作业, 见 PPT 尾页
- Satisfied

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

A first-attempt model:



$$G (c_1 \rightarrow c_1 W (\neg c_1 \wedge \neg c_1 W c_2))$$

The compliment LTL is false: $s_0 \rightarrow s_5 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_3 \rightarrow s_4 \rightarrow \dots$

So the original property is satisfied

How to design CTL spec? (同前, 实验小作业)

No strict sequencing: Processes need not enter their critical section in strict sequence

- there is a path with two distinct states satisfying c_1 such that no state in between them has that property
- LTL spec? Also no
- A compliment LTL? OK...

└ 1. 应用

1. 应用

1.4 Verification by NuSMV / Example / Mutual exclusion

A first-attempt model:

No strict sequencing: Processes need not enter their critical section in strict sequence

- there is a path with two distinct states satisfying c_1 such that no state in between them has that property

• LTL spec? Also no

• A complement LTL? OK.

$G (c_1 \rightarrow c_1 W (\neg c_1 \wedge \neg c_1 W c_2))$

The complement LTL is false: $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5 \rightarrow s_6 \rightarrow s_7 \rightarrow s_8 \rightarrow \dots$

So the original property is satisfied

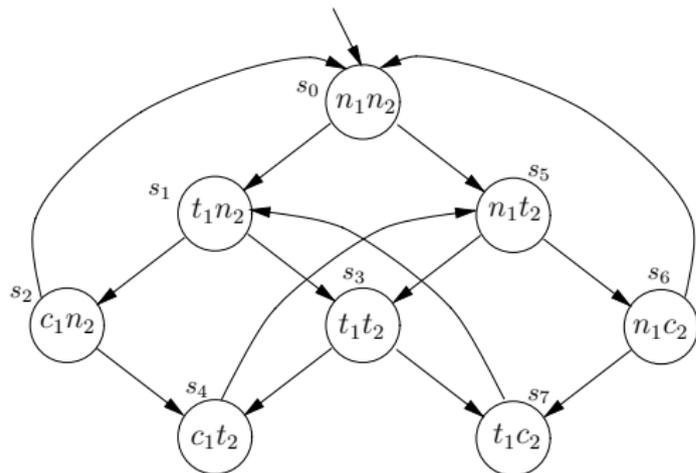
How to design CTL spec? (同前, 实验小作业)

This says that anytime we get into a c_1 state, either that condition persists indefinitely, or it ends with a non- c_1 state and in that case there is no further c_1 state unless and until we obtain a c_2 state.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion

A first-attempt model:



原因: The problem is that the state s_3 does not distinguish between which of the processes first went into its trying state.

解决方法: We can solve this by splitting s_3 into two states. (见下页)

回顾: *Liveness is not satisfied*:
Liveness: Whenever any process *requests* to enter its critical section, it will *eventually be permitted* to do so.

- LTL specification:

$$G (t_1 \rightarrow F c_1)$$

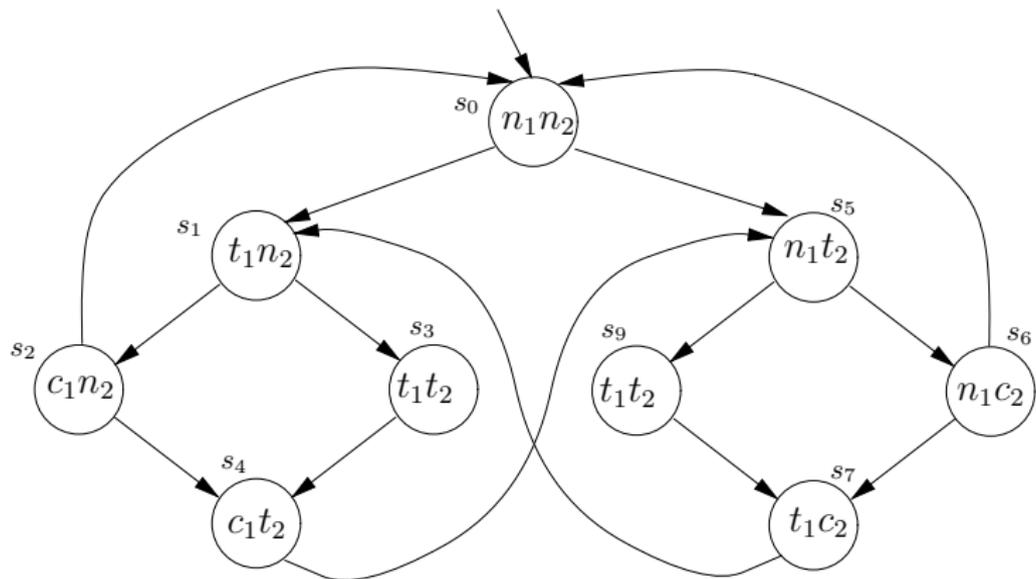
- Not Satisfied

$$s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \dots$$

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt

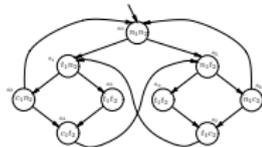
The second modeling attempt:



- $s_3 \implies s_3, s_9$

└ 1. 应用

The second modeling attempt:



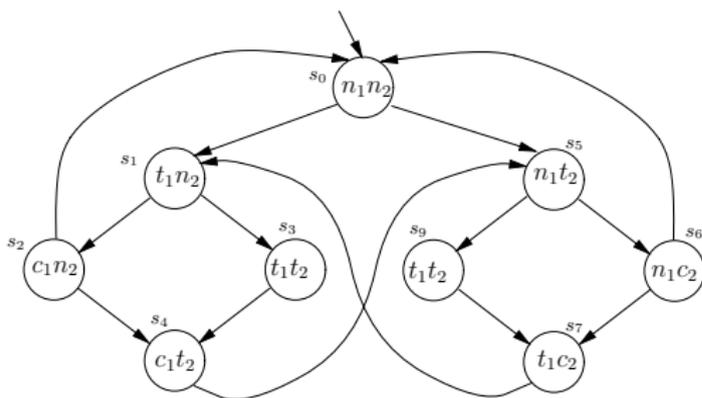
- $s_3 \mapsto s_4, s_9$

The two states s_3 and s_9 both correspond to the state s_3 in our first modelling attempt. They both record that the two processes are in their trying states, but in s_3 it is implicitly recorded that it is process 1's turn, whereas in s_9 it is process 2's turn. Note that states s_3 and s_9 both have the labelling t_1t_2 ; the definition of transition systems does not preclude this. We can think of there being some other, hidden, variables which are not part of the initial labelling, which distinguish s_3 and s_9 .

In this second modelling attempt, our transition system is still slightly over-simplified, because we are assuming that it will move to a different state on every tick of the clock (there are no transitions to the same state). We may wish to model that a process can stay in its critical state for several ticks, *but if we include an arrow from s_4 , or s_7 , to itself*, we will again violate liveness. This problem will be solved later in this chapter when we consider 'fairness constraints'.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt



Safety: Only *one process* is in its *critical section* at any time.

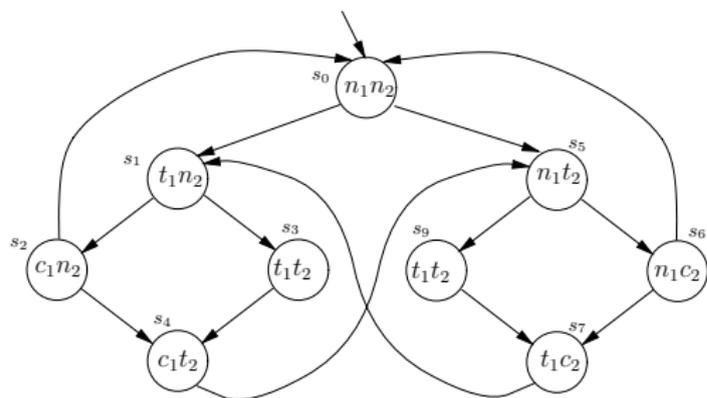
- LTL specification:

$$G \neg (c_1 \wedge c_2)$$

- Satisfied

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt



Liveness: Whenever any process *requests* to enter its critical section, it will *eventually be permitted* to do so.

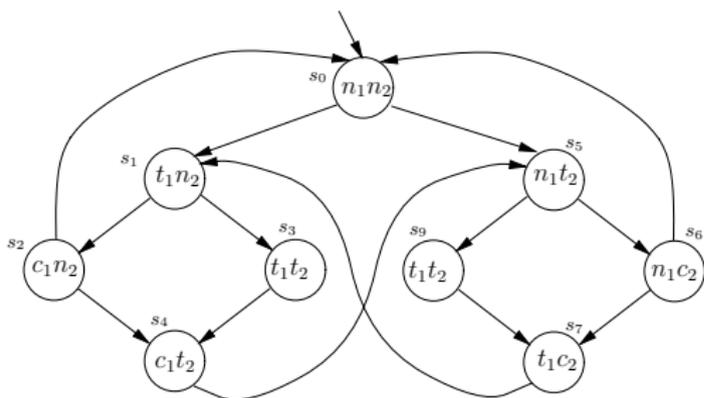
- LTL specification:

$$G (t_1 \rightarrow F c_1)$$

- *Now* Satisfied

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt

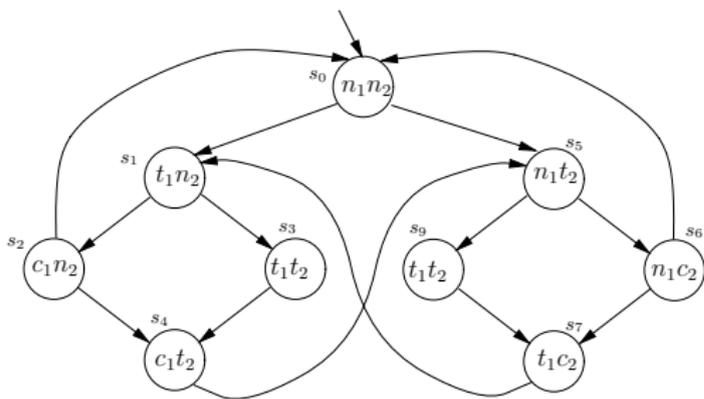


Non-blocking: A process can *always* request to *enter* its critical section.

- In other words, for every state satisfying n_1 , there is a successor satisfying t_1 .
- LTL specification? No
- CTL specification: ??
- Satisfied

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt



No strict sequencing: Processes need not enter their critical section in strict sequence

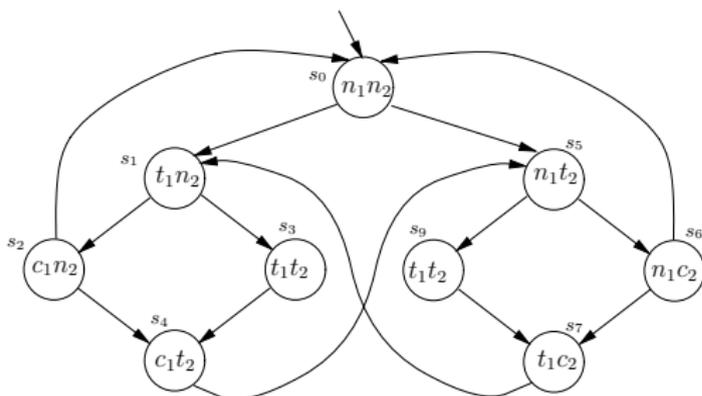
- there is a path with two distinct states satisfying c_1 such that no state in between has that property
- LTL spec? Also no
- A compliment LTL? OK...

$$G (c_1 \rightarrow c_1 \text{ W } (\neg c_1 \wedge \neg c_1 \text{ W } c_2))$$

The compliment LTL is false: $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_0 \rightarrow \dots$
So the original property is satisfied

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt



难道问题都解决了么? No

新问题 1: What if a process can stay in its critical state for several ticks?, i.e.,

- we include an arrow from s_4 , or s_7 , to itself

Liveness:

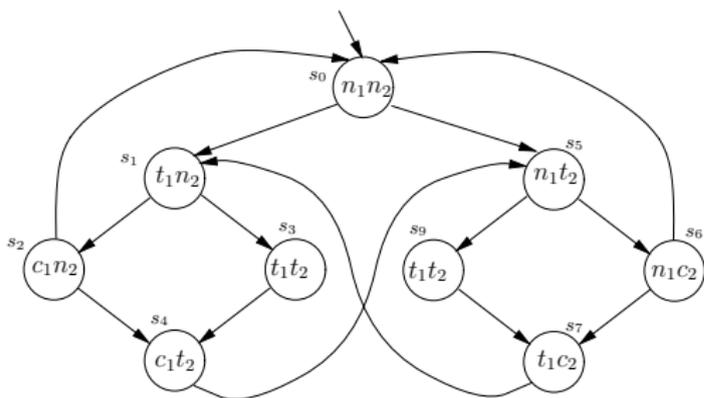
$$G (t_1 \rightarrow F c_1)$$

Not Satisfied again

方法: Consider “fairness constraints”, 见后

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 2nd Attempt



新问题 2: How to distinguish between states s_3 and s_9 in NuSMV?

方法: introduce a new variable, named turn, 见后

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

3rd attempt

```
MODULE main
```

```
VAR
```

```
  pr1: process prc(pr2.st, turn, FALSE);
```

```
  pr2: process prc(pr1.st, turn, TRUE);
```

```
  turn: boolean;
```

```
ASSIGN
```

```
  init(turn) := FALSE;
```

```
  ...
```

问题 2 的解决方法: introduce a new variable, named **turn**

└ 1. 应用

```
3rd attempt
-----
MODULE main
VAR
  pr1 : process prc(pr2.st, turn, FALSE);
  pr2 : process prc(pr1.st, turn, TRUE);
  turn : boolean;
ASSIGN
  init(turn) := FALSE;
  ...
```

问题 2 的解决方法: introduce a new variable, named **turn**

This code consists of two modules, `main` and `prc`. The module `main` has the variable `turn`, which determines whose turn it is to enter the critical section if both are trying to enter

The module `main` also has two instantiations of `prc`. In each of these instantiations, `st` is the status of a process (saying whether it is in its critical section, or not, or trying) and `other-st` is the status of the other process (notice how this is passed as a parameter in the third and fourth lines of `main`).

Because the boolean variable **turn** has been explicitly introduced to distinguish between states `s3` and `s9`, we now distinguish between certain states (for example, `ct0` and `ct1`) which were identical before. However, these states are not distinguished if you look just at the transitions from them. Therefore, they satisfy the same LTL formulas which don't mention `turn`. Those states are distinguished only by the way they can arise.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

```
MODULE prc(other-st, turn, myturn)
  VAR
st: {n, t, c};
  ASSIGN
    init(st) := n;
  ...
```

So, the variables are: st of prc1, st of prc2, turn

└ 1. 应用

The value of `st` evolves in the way described in a previous section: when it is `n`, it may stay as `n` or move to `t`. When it is `t`, if the other one is `n`, it will go straight to `c`, but if the other one is `t`, it will check whose turn it is before going to `c`. Then, when it is `c`, it may move back to `n`. Each instantiation of `prc` gives the turn to the other one when it gets to its critical section.

```
MODULE prc(other-st, turn, myturn)
  VAR
  st: {n, t, c};
  ASSIGN
  stst(st) := n;
  ...
```

So, the variables are: `st` of `prc1`, `st` of `prc2`, `turn`

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

```
MODULE prc(other-st, turn, myturn)
...
next(st) :=
  case
    (st = n)      : {t,n};
    (st = t) & (other-st = n)  : c;
    (st = t) & (other-st = t) & (turn = myturn): c;
    (st = c)      : {c,n};
    TRUE         : st;
  esac;
...
```

└ 1. 应用

```
MODULE prc(other=st, turn, myturn)
...
next(st) :=
  case
    (st = m) : {t,m};
    (st = t) & (other=st = m) : c;
    (st = t) & (other=st = t) & (turn = myturn): c;
    (st = c) : {c,m};
  TRUE : st;
***ac;
...
```

We have eliminated an over-simplification made in the model of 2nd attempt. Recall that we assumed the system would move to a different state on every tick of the clock (there were no transitions from a state to itself).

Now we allow transitions from each state to itself, representing that a process was chosen for execution and did some private computation, but did not move in or out of its critical section. Of course, by doing this we have introduced paths in which one process gets stuck in its critical section, whence the need to invoke a fairness constraint to eliminate such paths.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

```
MODULE prc(other-st, turn, myturn)
...
next(turn) :=
  case
    turn = myturn & st = c : !turn;
    TRUE                    : turn;
  esac;
...
```

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

```
MODULE prc(other-st, turn, myturn)
...
FAIRNESS running
FAIRNESS ! (st = c)
...
```

问题 1 的解决方法: Consider “fairness constraints”:

- We can restrict its search tree to execution paths along which an arbitrary *boolean formula* about the state ϕ is *true infinitely often*.

└ 1. 应用

Because this is often used to model fair access to resources, it is called a fairness constraint and introduced by the keyword FAIRNESS. Thus, the occurrence of FAIRNESS ϕ means that SMV, when checking a specification ϕ , will ignore any path along which ϕ is not satisfied infinitely often.

```
MODULE proc(other-at, turn, myturn)
...
FAIRNESS running
FAIRNESS ! (at = c)
...
```

问题 1 的解决方法: Consider "fairness constraints":

- We can restrict its search tree to execution paths along which an arbitrary *boolean formula* about the state ϕ is *true infinitely often*.

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

```
// safety
LTLSPEC G!((pr1.st = c) & (pr2.st = c))
// liveness
LTLSPEC G((pr1.st = t) -> F (pr1.st = c))
LTLSPEC G((pr2.st = t) -> F (pr2.st = c))
// negation of strict sequencing (desired to be false)
LTLSPEC G(pr1.st=c -> ( G pr1.st=c | (pr1.st=c U
    ( pr1.st!=c & G pr1.st!=c | (( pr1.st!=c) U pr2.st=c)))
```

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

运行结果: \$./NuSMV c-sample2-mutex.smv

```
-- specification G !(pr1.st = c & pr2.st = c) is true
-- specification G (pr1.st = t -> F pr1.st = c) is true
-- specification G (pr2.st = t -> F pr2.st = c) is true
-- specification G (pr1.st = c -> ( G pr1.st = c | (pr1.st = c
U ((pr1.st != c && G p ppr1.s.ssst ! !!!!!!!=t !=.st !=r1.
st != G pr1.st !=c & G p1r1.st !=!t != c & G 1spr1s.st !=1. c
st c!c=c c & p1G p1r1.
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
    pr1.st = n
    pr2.st = n
    turn = FALSE
```

1. 应用

1.4 Verification by NuSMV | Example | Mutual exclusion | 3rd Attempt

运行结果: \$./NuSMV c-sample2-mutex.smv

```
-> State: 1.1 <-  
  pr1.st = n  
  pr2.st = n  
  turn = FALSE  
-> Input: 1.2 <-  
  _process_selector_ = pr1  
  running = FALSE  
  pr2.running = FALSE  
  pr1.running = TRUE  
-> State: 1.2 <-  
-> Input: 1.3 <-  
  _process_selector_ = pr2  
  pr2.running = TRUE  
  pr1.running = FALSE  
-> State: 1.3 <-  
  pr2.st = t  
-> Input: 1.4 <-  
-> State: 1.4 <-  
  pr2.st = c  
-> Input: 1.5 <-  
  _process_selector_ = pr1  
  pr2.running = FALSE  
  pr1.running = TRUE  
-> State: 1.5 <-  
  pr1.st = t
```

```
-> Input: 1.6 <-  
  _process_selector_ = pr2  
  pr2.running = TRUE  
  pr1.running = FALSE  
-> State: 1.6 <-  
  pr2.st = n  
-> Input: 1.7 <-  
  _process_selector_ = pr1  
  pr2.running = FALSE  
  pr1.running = TRUE  
-> State: 1.7 <-  
  pr1.st = c  
-> Input: 1.8 <-  
-> State: 1.8 <-  
  pr1.st = n  
  turn = TRUE  
-> Input: 1.9 <-  
-> State: 1.9 <-  
  pr1.st = t  
-> Input: 1.10 <-  
-> State: 1.10 <-  
  pr1.st = c  
-> Input: 1.11 <-  
-- Loop starts here
```

```
-> State: 1.11 <-  
  pr1.st = n  
-> Input: 1.12 <-  
  _process_selector_ = main  
  running = TRUE  
  pr1.running = FALSE  
-- Loop starts here  
-> State: 1.12 <-  
-> Input: 1.13 <-  
  _process_selector_ = pr1  
  running = FALSE  
  pr1.running = TRUE  
-- Loop starts here  
-> State: 1.13 <-  
-> Input: 1.14 <-  
  _process_selector_ = pr2  
  pr2.running = TRUE  
  pr1.running = FALSE  
-- Loop starts here  
-> State: 1.14 <-  
-> Input: 1.15 <-  
  _process_selector_ = main  
  running = TRUE  
  pr2.running = FALSE  
-> State: 1.15 <-
```

1. 应用

1.4 Verification by NuSMV | Example | Foxes and Rabbits



例: Foxes and Rabbits
如何用 NuSMV 进行SAT 求解?

- *Three* foxes and *three* rabbits have to cross a river
- There is only one boat that can carry *at most two* animals
- When the boat is *on the river*, at *each* of the *sides* the number of *foxes* should be \leq the number of *rabbits*, otherwise the rabbits will be eaten

Is there a solution? which?

1. 应用

1.4 Verification by NuSMV | Example | Foxes and Rabbits

要点: *Automated Reasoning*: *Do not* think about how to solve it, *only* specify the rules, and let the tool to solve it

Several ways to encode

We prefer not to define the moves in both directions separately

Variables:

- b: a boolean expressing where the boat is
- f: the number of foxes at the side where the boat is
- fb: the number of foxes that goes into the boat
- r: the number of rabbits at the side where the boat is
- rb: the number of rabbits that goes into the boat

1. 应用

1.4 Verification by NuSMV | Example | Foxes and Rabbits

- b: a boolean expressing where the boat is
- f: the number of foxes at the side where the boat is
- fb: the number of foxes that goes into the boat
- r: the number of rabbits at the side where the boat is
- rb: the number of rabbits that goes into the boat

```
MODULE main
  VAR
    r : 0..3;
    rb : 0..2;
    f : 0..3;
    fb : 0..2;
    b : boolean;
```

1. 应用

1.4 Verification by NuSMV | Example | Foxes and Rabbits

- b : a boolean expressing where the boat is
- f : the number of foxes at the side where the boat is
- fb : the number of foxes that goes into the boat
- r : the number of rabbits at the side where the boat is
- rb : the number of rabbits that goes into the boat

INIT

$b \ \& \ f = 3 \ \& \ r = 3$

1. 应用

1.4 Verification by NuSMV | Example | Foxes and Rabbits

- b : a boolean expressing where the boat is
- f : the number of foxes at the side where the boat is
- fb : the number of foxes that goes into the boat
- r : the number of rabbits at the side where the boat is
- rb : the number of rabbits that goes into the boat

TRANS

next(b) = $!b \ \&$

$fb + rb \leq 2 \ \&$

$fb + rb \geq 1 \ \&$

$f - fb \leq r - rb \ \&$

next(f) = $3 - f + fb \ \&$

next(r) = $3 - r + rb$

CTLSPEC **!EF**($!b \ \& \ f = 3 \ \& \ r = 3$)

1. 应用

1.4 Verification by NuSMV | Example | Foxes and Rabbits

运行结果: \$./NuSMV c-sample3-fox.smv

```
-- specification !(EF (!!b & f = 3) & r = 3) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
```

```
-> State: 1.1 <-
  r = 3
  rb = 1
  f = 3
  fb = 1
  b = TRUE
-> State: 1.2 <-
  r = 1
  rb = 0
  f = 1
  b = FALSE
-> State: 1.3 <-
  r = 2
  f = 3
  fb = 2
  b = TRUE
```

```
-> State: 1.4 <-
  r = 1
  f = 2
  fb = 1
  b = FALSE
-> State: 1.5 <-
  r = 2
  rb = 1
  b = TRUE
-> State: 1.6 <-
  rb = 0
  b = FALSE
-> State: 1.7 <-
  r = 1
  fb = 2
  b = TRUE
```

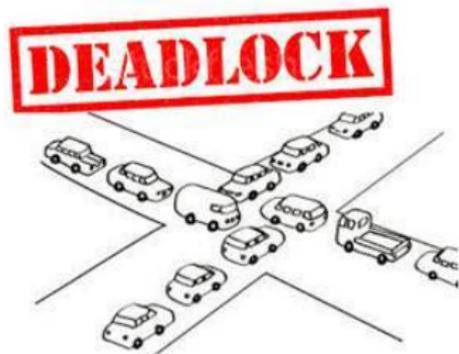
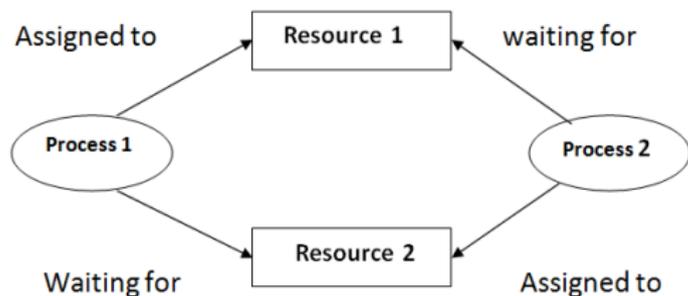
```
-> State: 1.8 <-
  r = 2
  f = 3
  fb = 1
  b = FALSE
-> State: 1.9 <-
  r = 1
  rb = 1
  f = 1
  b = TRUE
-> State: 1.10 <-
  r = 3
  f = 3
  b = FALSE
```

b=FALSE, f=3,
r=3

indeed showing
that the final state
where all animals
reached the other
side

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks



问题: What is a *deadlock* here?

- a *state* that cannot be changed by applying the rules

Occurs very often in hardware, network protocols,...

定义: Deadlock

In a transition system, a state s is a *deadlock state*, if not state $s' \neq s$ exists such that $s \rightarrow s'$

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

定义: Deadlock

In a transition system, a state s is a *deadlock state*, if not state $s' \neq s$ exists such that $s \rightarrow s'$

Typical desired *property* to be verified: *No deadlock state is reachable*

We will elaborate a *network* format in which messages are

- sent
- processed
- received

The network is a *graph*, in which the edges are *channels*.

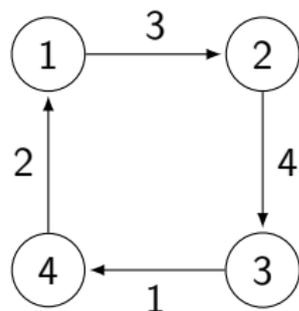
- Such a channel is either empty,
- or is filled by a message and a destination
 - So no two messages can be in the same channel
- 注: 这个 graph 与模型检测中的 transition system 模型的用途不同

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

The *network* is a *graph*, in which the edges are *channels*.

- Such a channel is either empty,
- or is filled by a message and a destination
 - So no two messages can be in the same channel
- 注: 这个 graph 与模型检测中 transition system 模型的用途不同



As the set of *initial states*, we choose the single state where every channel is empty

Then, we wonder *whether* in a *particular network*, a deadlock state is *reachable*

Our computation / processing is *asynchronous*

- That is, it is not controlled by a central clock, but *at any moment* a *send* step, a *processing* step or a *receiving* step *can be done*

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

第 1 步: Define the state space

- *message*: For investigating deadlocks, the *contents of the message* does not play a role: it will be *ignored*
- *node ID*: Number the nodes from 1 to n
- *channel*: So the *contents of a channel* is identified by
 - the *destination node* of the corresponding message, or
 - 0, if it is *empty*
 - so, for every channel c declare
 - $c : 0..n$

This yields state space $\{0, 1, \dots, n\}^k$, for $k =$ number of channels

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

第 2 步: Define $OK(n, m, c)$

For every node n and destination m , it should be *determined which outgoing channel c* from n is *allowed* to be chosen for passing a message to m

定义: Write $OK(n, m, c)$, if this is *allowed*

讨论: *Typically*, $OK(n, m, c)$ yields true, if and only if *c is the first edge of a shortest path* from n to m

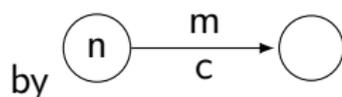
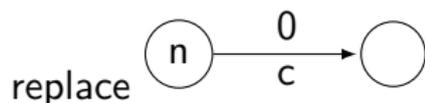
- Then the messages will always follow a shortest path to its destination

1. 应用

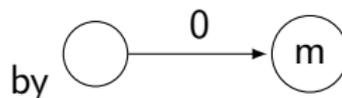
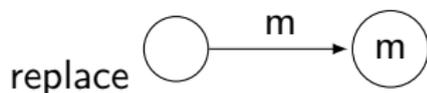
1.4 Verification by NuSMV | Example | Checking deadlocks

第 3 步: Define the transitions of states, i.e., three types of steps:

- *send* steps: replace the value 0 in an empty outgoing channel c from n by the value m , *if* $OK(n, m, c)$



- *receive* steps: if channel c to node m contains the value m , then it may be replaced by 0

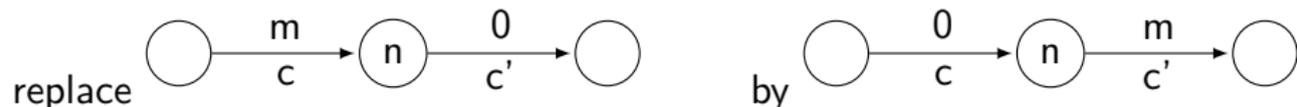


1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

第 3 步: Define the transitions of states, i.e., three types of steps:

- *processing* steps: if channel c to node n contains the value m , and the channel c' starting in n is empty and *satisfies* $OK(n, m, c')$, then the destination m may be passed to c'
 - that is, c gets the value 0 and c' gets the value m



The transition relation is a big *disjunction* of all possible *send*, *receive* and *processing* steps

...

```
| case c = m & c = 0 : next(c) = 0 & next(c') = m;  
   TRUE : next(c) = m & next(c') = 0;  
esac & P
```

...

where P is the conjunction of $\text{next}(x)=x$ for all other channels x

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

The ultimate NuSMV code consists of

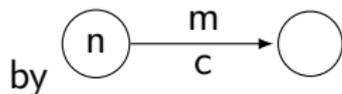
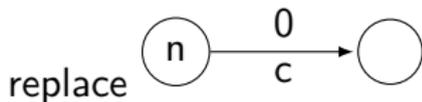
- VAR
 - $c : 0..n$ for all channels c
- INIT
 - $c = 0$ for all channels c
- TRANS
 - the big disjunction of all possible steps, to be *generated by a program*
- CTLSPEC EF D
 - for D describing deadlock
 - Deadlock D is obtained as $!Q$ in which Q is the disjunction of all non-TRUE branches in all these case statements
 - In order to find the path to the deadlock, one should run CTLSPEC !EF D, then the desired path is obtained from the counter example

1. 应用

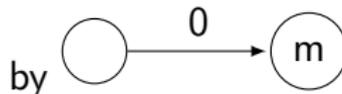
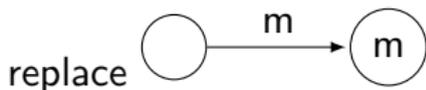
1.4 Verification by NuSMV | Example | Checking deadlocks

回顾: write $OK(n, m, c)$, if it is allowed to pass the message to the outgoing channel c from n , when a message has to be sent from n to m

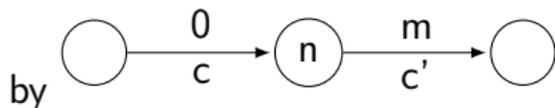
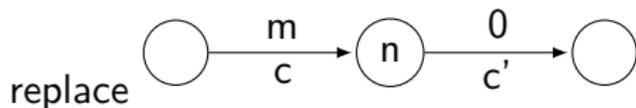
- *send* steps: if $OK(n, m, c)$, then



- *receive* steps:



- *processing* steps: if $OK(n, m, c')$, then

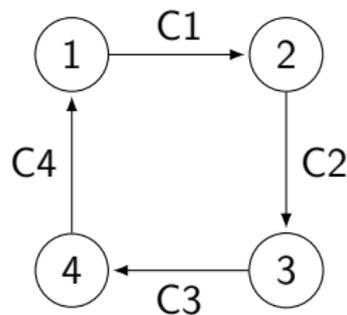


1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

Let M be the set of *main* nodes: nodes that are allowed to send messages, and to which messages can be sent

Choose $M = \{1, 2, 3\}$ in



Then by our approach NuSMV finds a reachable deadlock by the five steps

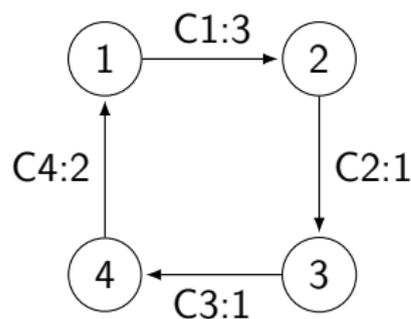
- send(3,2): C3:=2
- process C3: C4:=2; C3:=0
- send(3,1): C3:=1
- send(1,3): C1:=3
- send(2,1): C2:=1

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

Let M be the set of *main* nodes: nodes that are allowed to send messages, and to which messages can be sent

Hence the following deadlock is reached



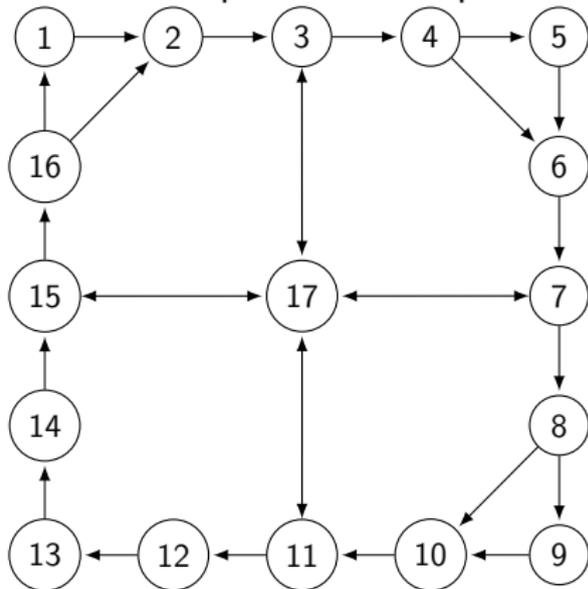
Then by our approach NuSMV finds a reachable deadlock by the five steps

- send(3,2): C3:=2
- process C3: C4:=2; C3:=0
- send(3,1): C3:=1
- send(1,3): C1:=3
- send(2,1): C2:=1

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

A more complicated example:



When taking $M = \{1, 5, 9, 13\}$, no deadlock is reachable

But when taking $M = \{2, 4, 6\}$, a deadlock is reachable

Doing this *by hand* is *not feasible* anymore, just like in many other examples and formats and as occurs in practice

(实验大作业, 可选, 见尾)

1. 应用

1.4 Verification by NuSMV | Example | Checking deadlocks

回顾: 定义: Deadlock

In a transition system, a state s is a *deadlock state*, if not state $s' \neq s$ exists such that $s \rightarrow s'$

Typical desired *property* to be verified: *No deadlock state is reachable*

回顾: CTLSPEC EF D

- Deadlock D is obtained as !Q in which Q is the disjunction of all non-TRUE branches in all these case statements

An interesting *variant* of deadlock: *local deadlock*:

- a particular variable *will never change* in the future
- for checking whether a channel c having value x causes such a local deadlock, we need a nested CTL formula
- CTLSPEC EF(AG $c=x$)

└ 1. 应用

stating that there exists a path to a state (EF) such that from that state for every path on every state (AG), it holds that $c=x$

If we do not know x , we may declare it as a variable, do not initialize it, and put $\text{next}(x)=x$ in TRANS

问题: Deadlock

In a transition system, a state s is a **deadlock state**, if not state $s' \neq s$ exists such that $s \rightarrow s'$

Typical desired **property** to be verified: *No deadlock state is reachable*

问题: CTLSPEC EF D

Deadlock D is obtained as IQ in which Q is the disjunction of all non-TRUE branches in all these case statements

An interesting **variant** of deadlock: **local deadlock**:

- a particular variable **will never change** in the future
- for checking whether a channel c having value x causes such a local deadlock, we need a nested CTL formula
- CTLSPEC EF(AG $c=x$)

总结: NuSMV for model checking, which can be used to

- model and verify a model
- solve an SAT problem

与普通编程语言的区别:

- *Do not* think about functions, think about *states* and transitions of states
- *Do not* think about how to solve it, *only* specify the rules, and let the tool to solve it

实验小作业: 使用 NuSMV 实现 PPT 中 first-attempt model, 要求

- 用 CTL 设计 Non-blocking, No strict sequencing, 并验证所有四个性质
- 给出源码、实验报告

实验大作业 (可选): 选择一篇 CCF A 类论文, 自己用 NuSMV 设计论文中的模型, 并验证, 附完整文档。

(也可选择 NuSMV 中 example 中的 A 类论文, 对已给出的模型进行阅读, 并附完整的阅读、试验报告、以及心得体会)

实验大作业 (可选): 实现 PPT 中的 deadlock 验证, 并针对 complicated example 中所给的结论进行验证。要求

- 附上源码和实验报告