Temporal Logic
- Model Checking

Moonzoo Kim
CS Dept. KAIST

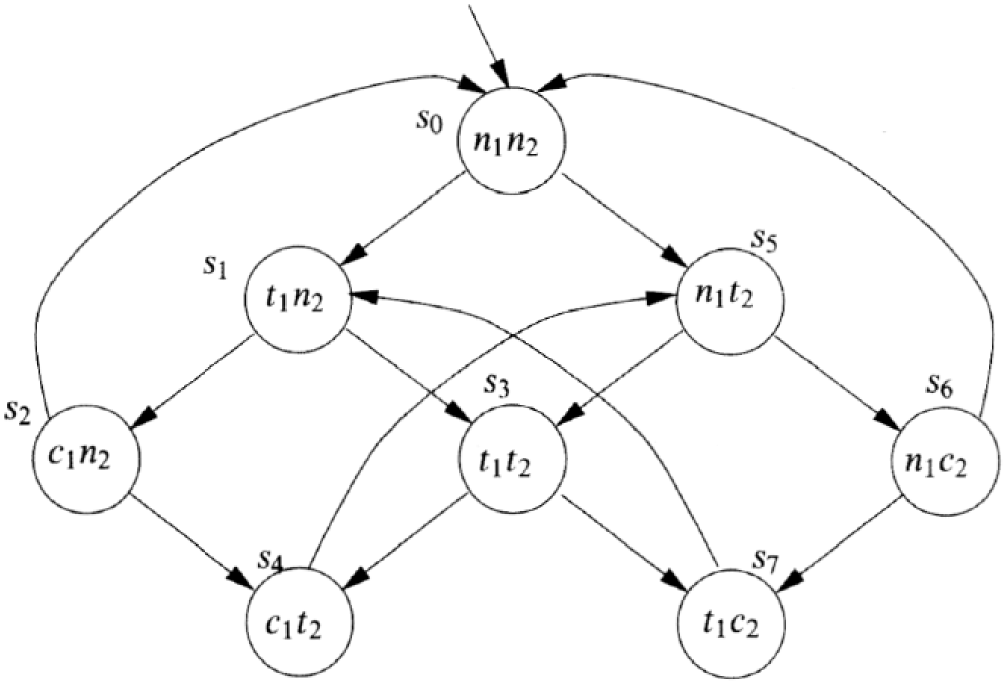
moonzoo@cs.kaist.ac.kr

Mutual exclusion example

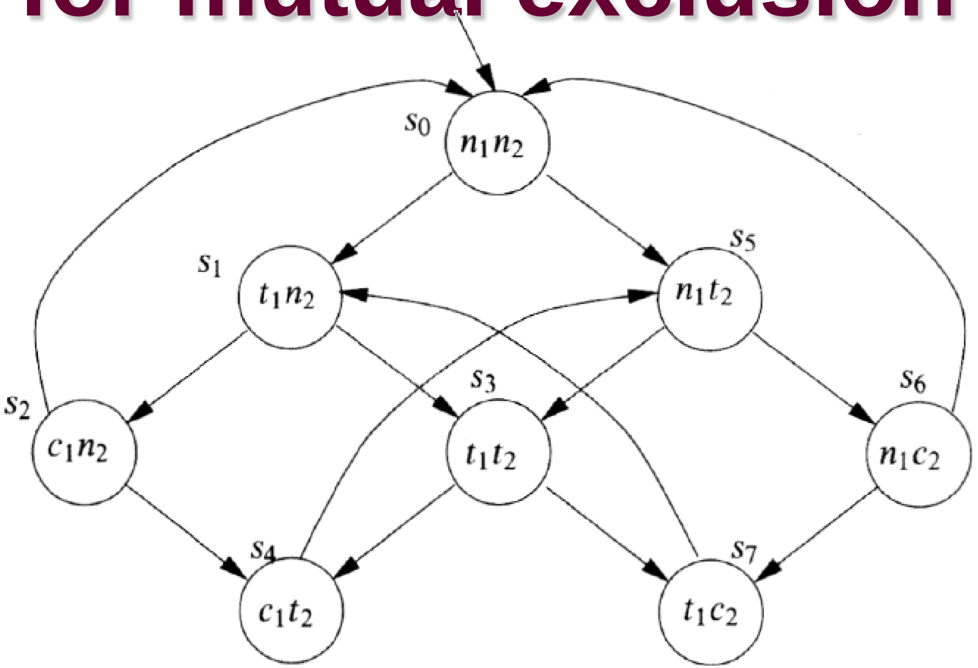
- When concurrent processes share a resource, it may be necessary to ensure that they do not have access to the common resource at the same time
  - We need to build a protocol which allows only one process to enter critical section
- Requirement properties
  - Safety:
    - Only one process is in its critical section at anytime
  - 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^{st} model

- We model two processes
  - each of which is in
    - non-critical state (n) or
    - trying to enter its critical state (t) or
    - critical section (c)
  - No self edges
- each process executes like n→ t → c → n → …
  - but the two processes interleave with each other
    - only one of the two processes can make a transition at a time (asynchronous interleaving)



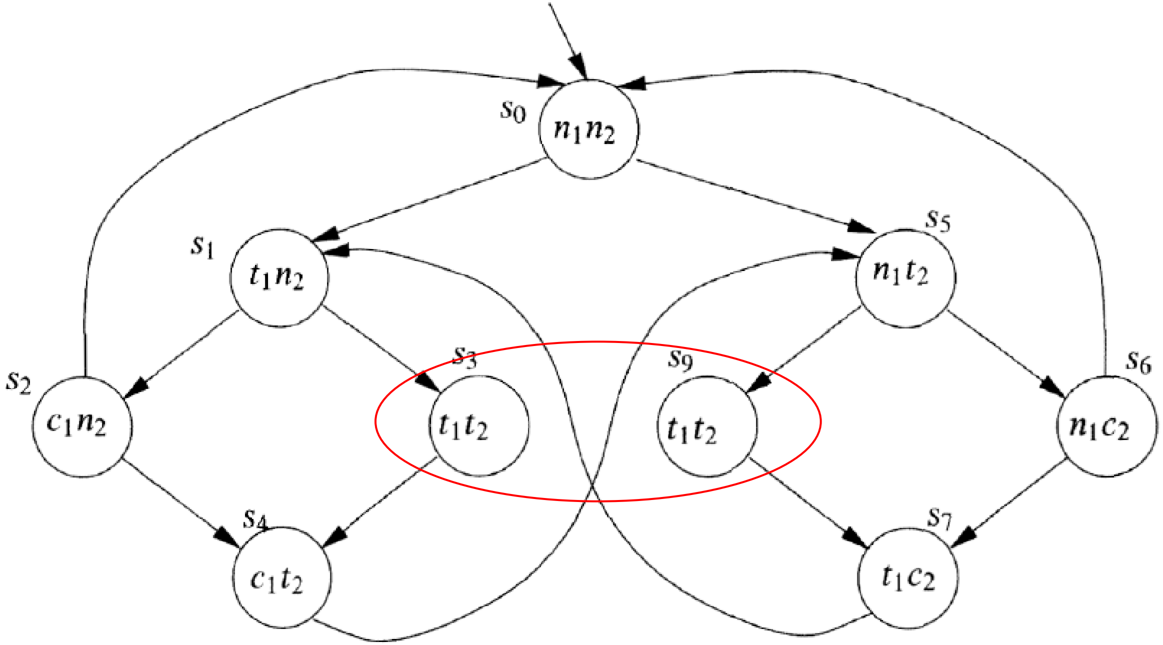
1st model for mutual exclusion



- Safety: $s_0 \vDash G \neg (c_1 \wedge c_2)$
- Liveness $s_0 \nvDash G(t_1 \rightarrow F c_1)$
  - see $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \rightarrow s_1 \rightarrow s_3 \rightarrow s_7 \ldots$
- Non-blocking
  - for every state satisfying n_i, there is a successor satisfying t_i
    - s_0 satisfies this property
  - We cannot express this property in LTL but in CTL
    - Note that LTL specifies that ϕ is satisfied for all paths
- No strict ordering
  - there is a path where c_1 and c_2 do not occur in strict order
  - Complement of this is
    - $G(c_1 \rightarrow c_1 \; W \; (\neg c_1 \wedge \underline{\neg c_1 \; W \; c_2}))$
    - 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

2nd model for mutual exclusion

- All 4 properties are satisfied
  - Safety
  - Liveness
  - Non-blocking
  - No strict sequencing



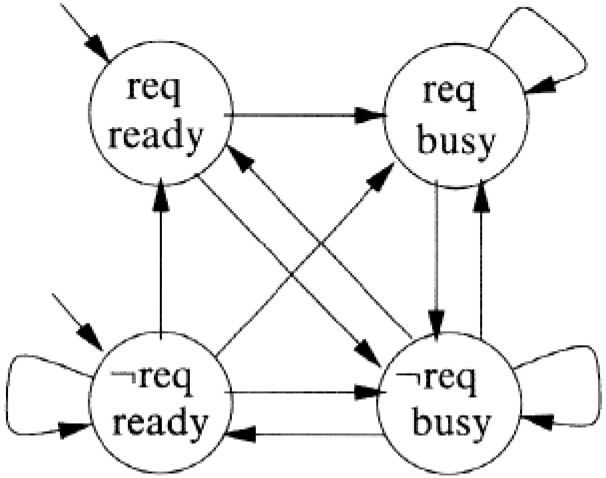
NuSMV model checker

- NuSMV programs consist of one or more modules.
  - 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 x (init(x)) and its next value (next(x)) 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

Example

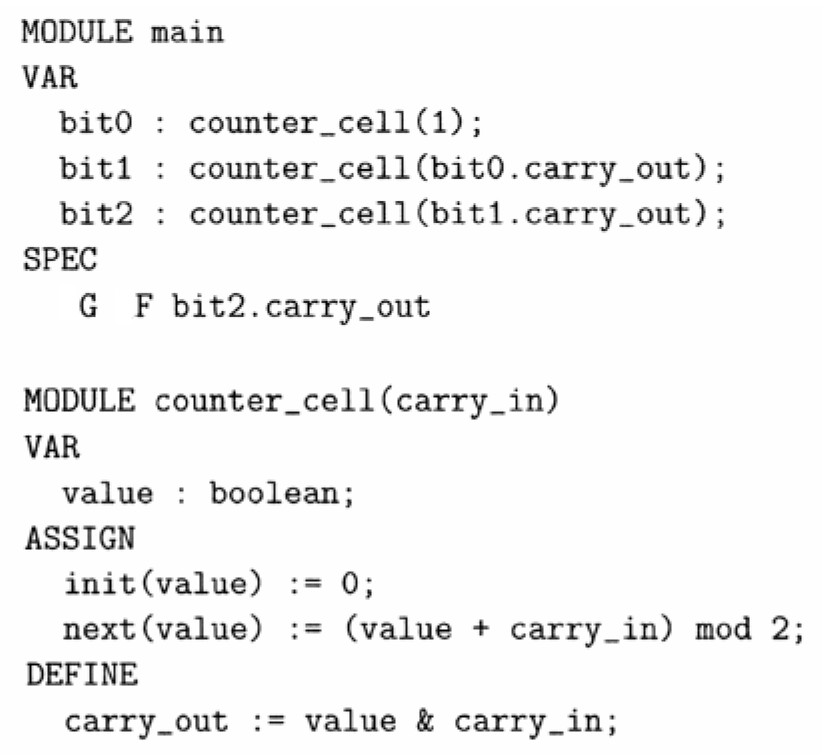
```
MODULE main
VAR
 request: boolean;
 status: {ready,busy};
ASSIGN
 init(status) := ready;
 next(status) := case
                    request : busy;
                    1: {ready,busy};
                    esac;

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

- request is under-specified, i.e., not controlled by the program
  - request is determined (randomly) by external environment
  - thus, whole program works non-deterministically
- Case statement is evaluated top-to-bottom



Modules in NuSMV

- A module is instantiated when a variable having that module name as its type is declared.
- A 3 bit counter increases from 000 to 111 repeatedly
  - Req. property
    - infinitely setting carry-out of most significant bit as 1
- By default, modules in NuSMV are composed synchronously
  - there is a global clock and, each time it ticks, each of the modules executes in parallel
  - By use of the ‘process’ keyword, it is possible to compose the modules asynchronously



```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
   G  F bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE
  carry_out := value & carry_in;
```