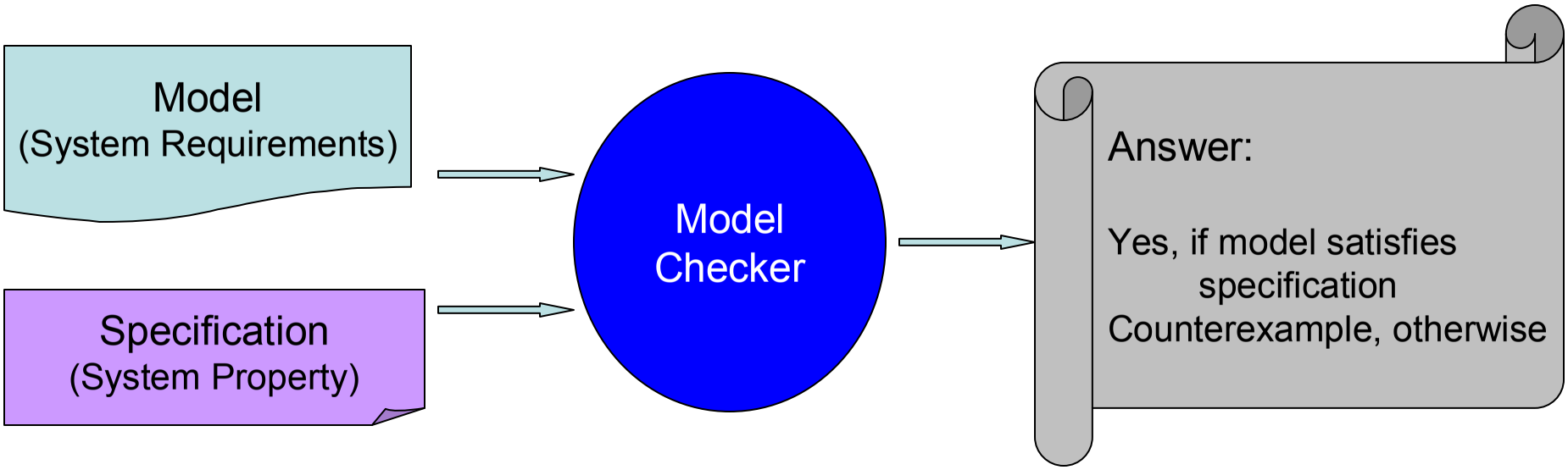
Model Checking

Model Checking Process



Increase our confidence in the correctness of the model:
-The model satisfied enough system properties
- Study counterexamples, pinpoint the source of the error, correct the model, and try again

Mutual Exclusion Example

The Model

(Willem Visser, http://ase.arc.nasa.gov/visser/ASE2002TutSoftwareMC-fonts.ppt)

- Two process mutual exclusion with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available (S_0) or taken (S_1)
- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1\, N_2\, S_0$

$$
\begin{array}{lll||lll}
N_1 & \rightarrow & T_1 & N_2 & \rightarrow & T_2 \\
T_1 \wedge S_0 & \rightarrow & C_1 \wedge S_1 & T_2 \wedge S_0 & \rightarrow & C_2 \wedge S_1 \\
C_1 & \rightarrow & N_1 \wedge S_0 & C_2 & \rightarrow & N_2 \wedge S_0
\end{array}
$$

Mutual Exclusion Example

Specification – Desirable Property

No matter where you are there is always a way to get to the initial state

$$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

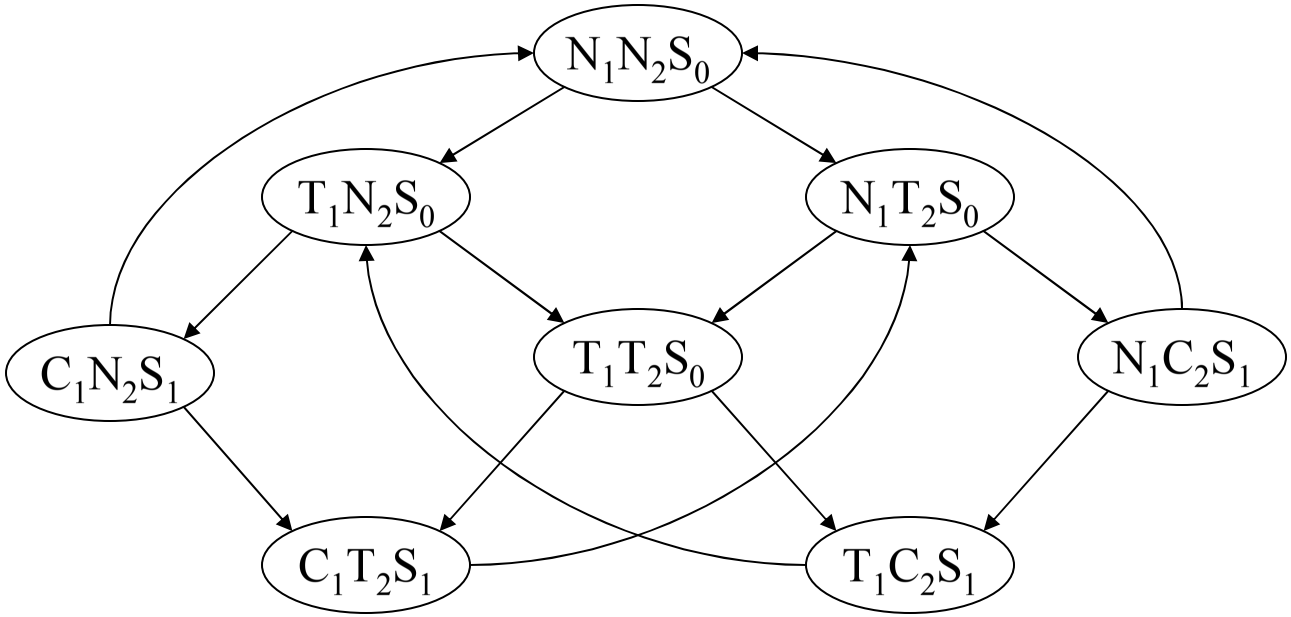
Mutual Exclusion Example

Model Checker

Answer: Yes

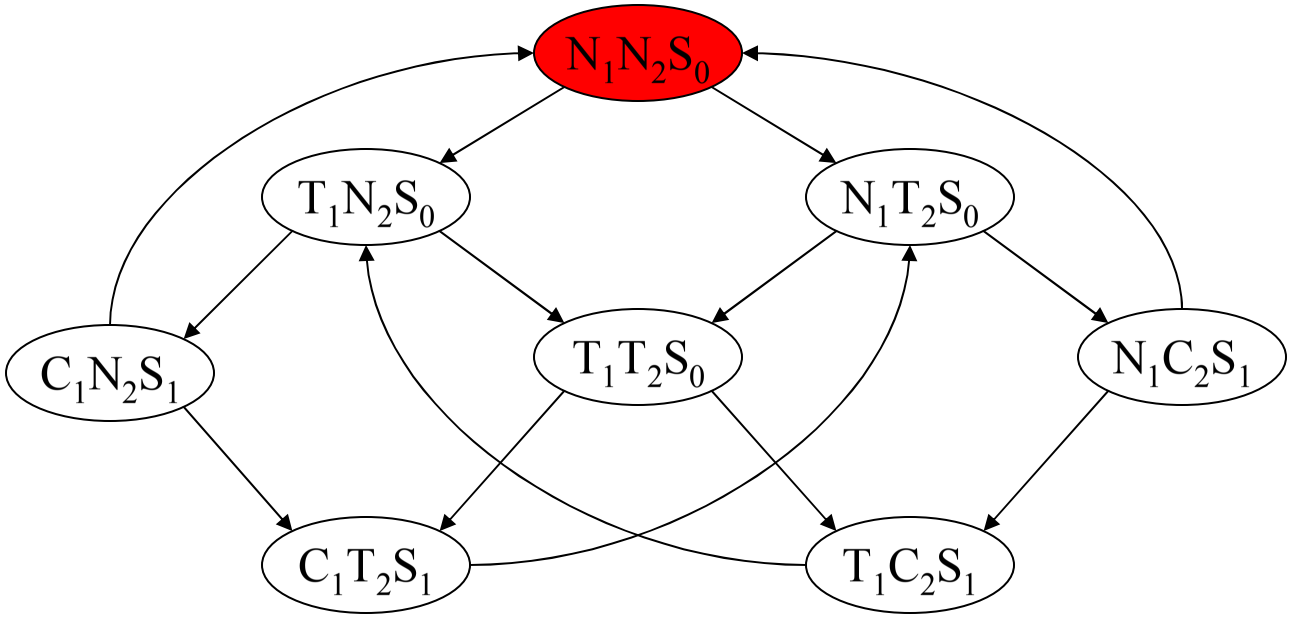
(A Proof:

All possible behaviors



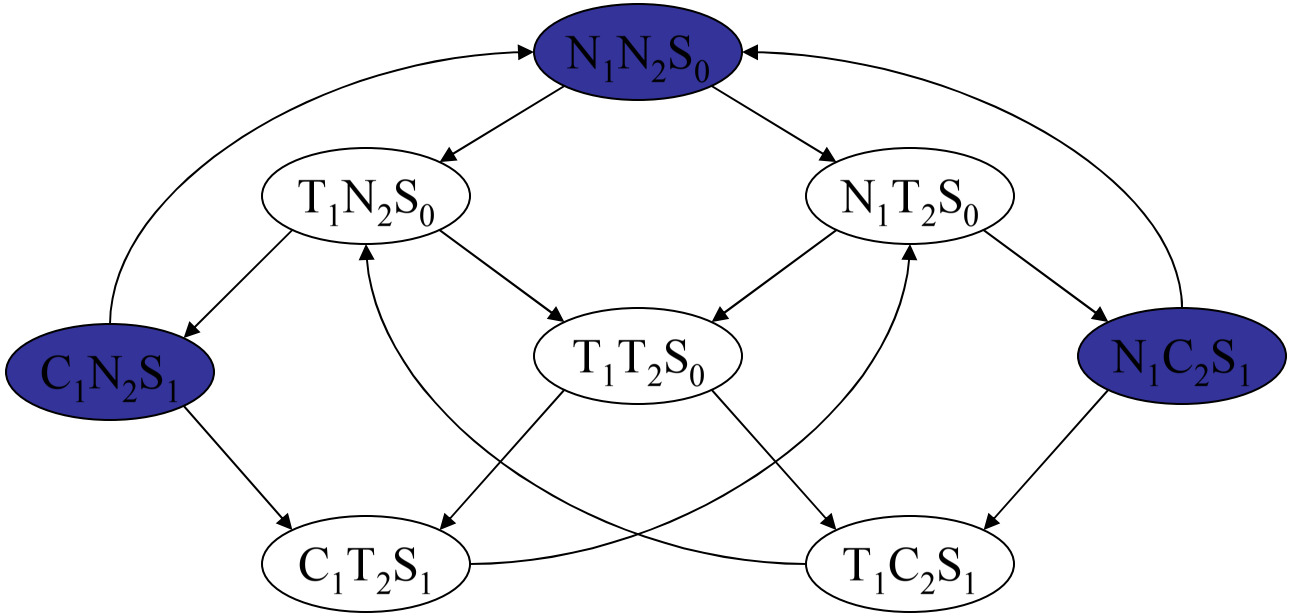
Mutual Exclusion Example

Model Checker



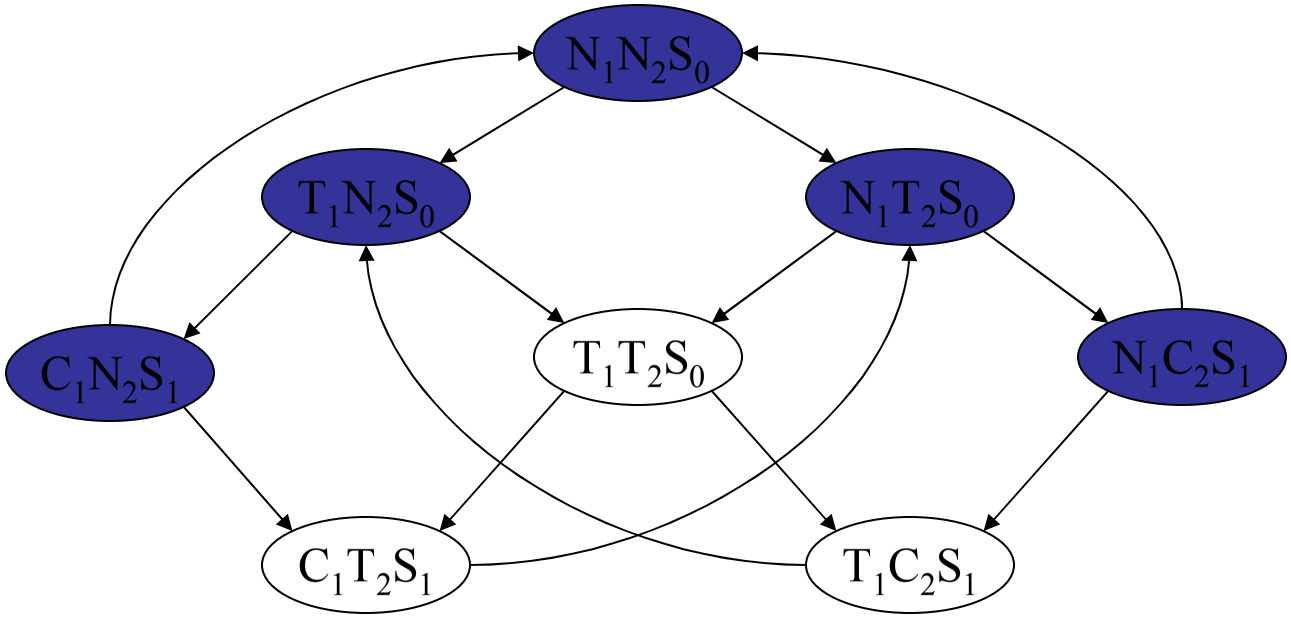
Mutual Exclusion Example

Model Checker



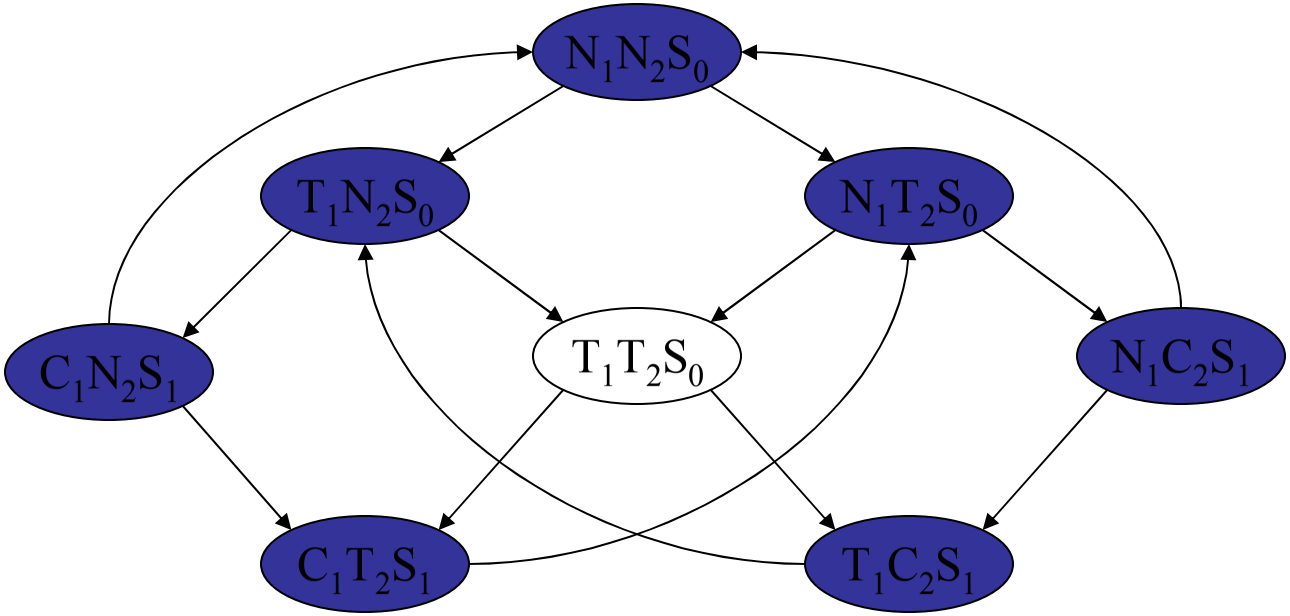
Mutual Exclusion Example

Model Checker



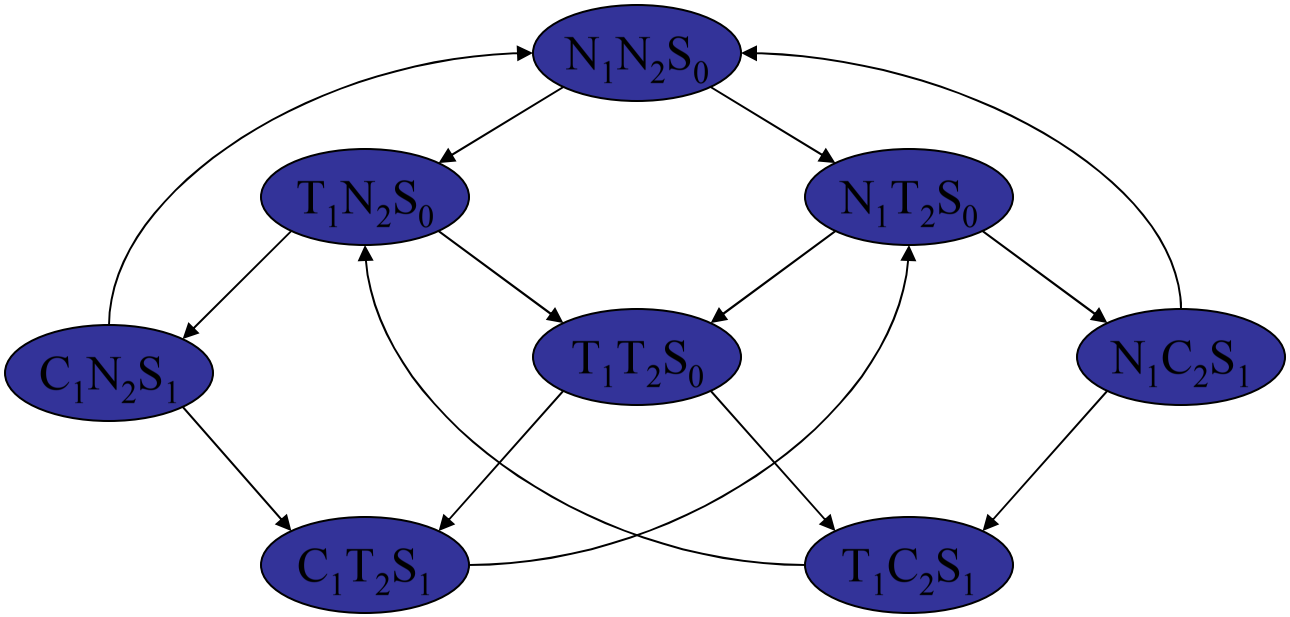
Mutual Exclusion Example

Model Checker



Mutual Exclusion Example

Model Checker



Mutual Exclusion Example

Specification – Desirable Property

No matter where you are there is no way to get to the initial state

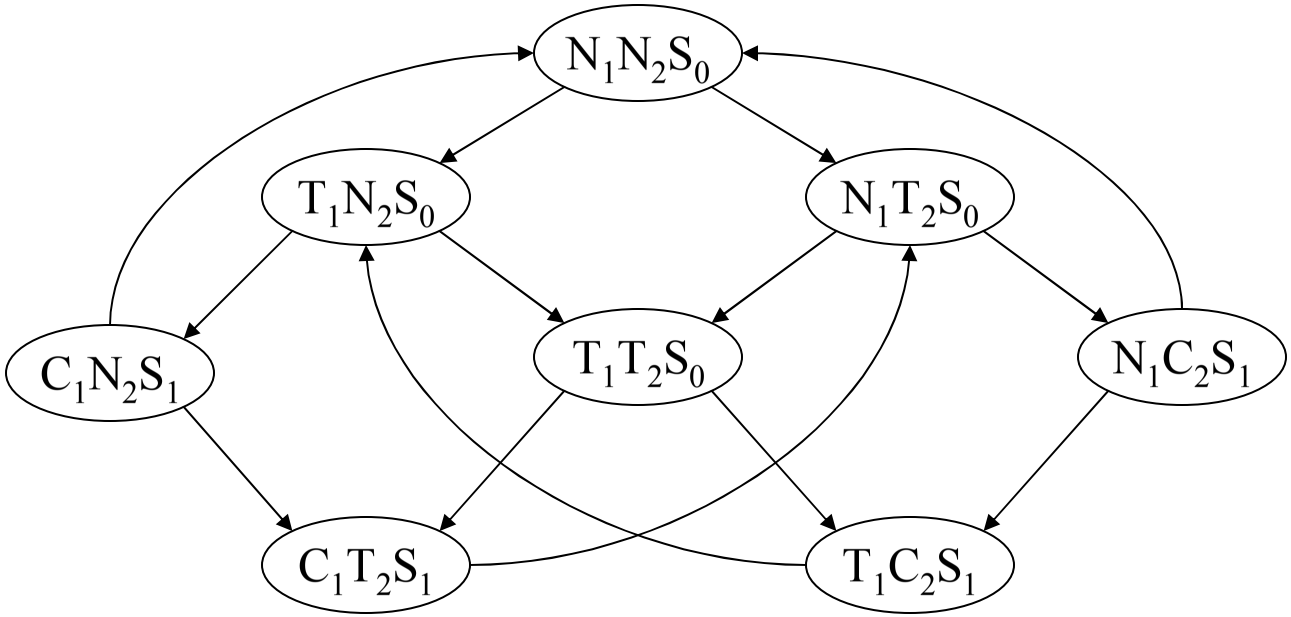
$$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

Mutual Exclusion Example

Model Checker

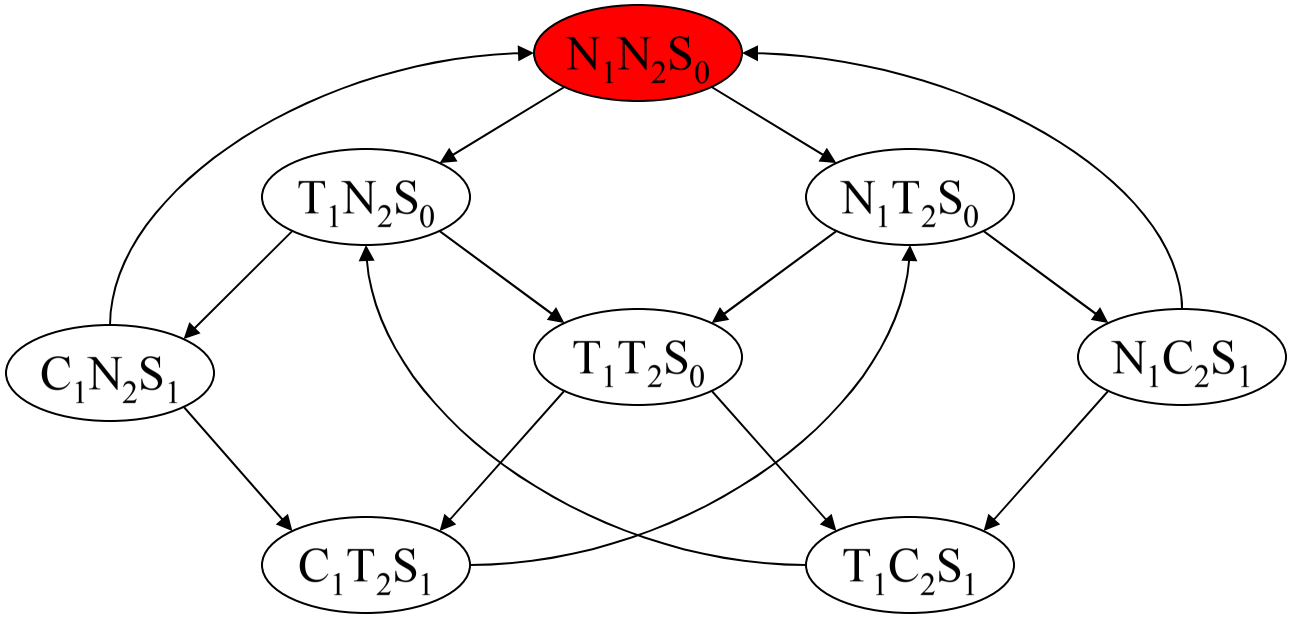
Answer: No

Counterexample: *All* possible behaviors



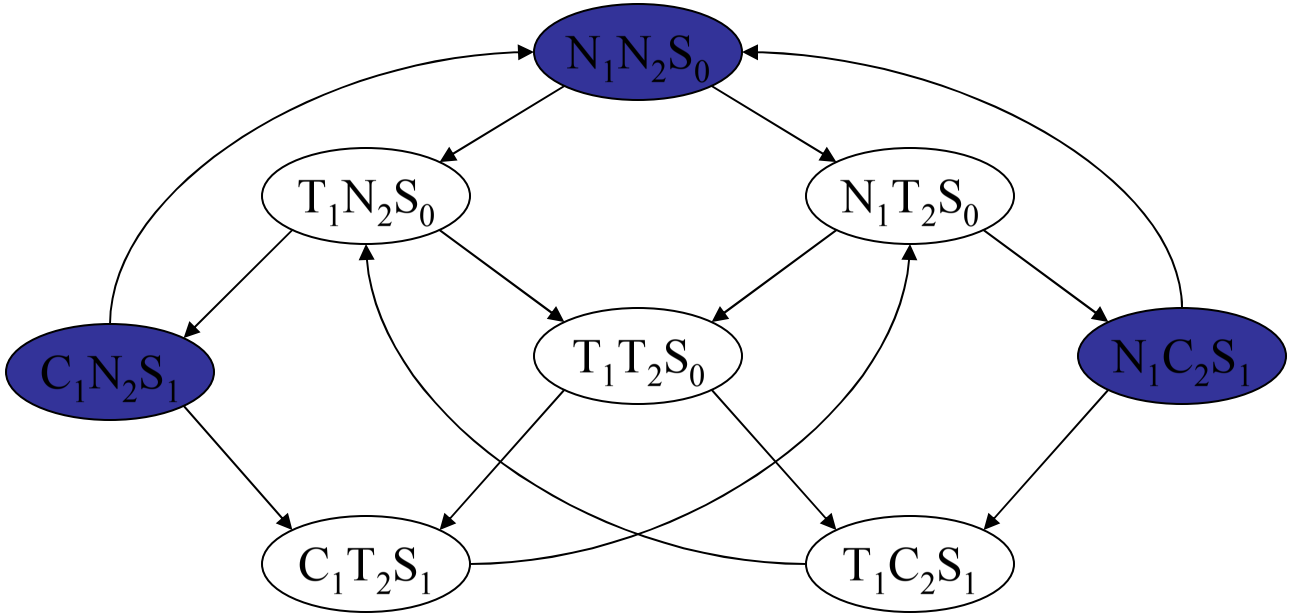
Mutual Exclusion Example

Model Checker



Mutual Exclusion Example

Model Checker



Defining Models

- Kripke Structure

K = < S ,p1, . . ., pk, R >

- S: the set of possible global states
- R(s,s'): the move from s to s' is a possible atomic transition
- Unary relations p1, . . ., pk express properties of the global states, e.g., being an initial state, being an accepting state, or that a particular variable has a special value.

- **Definition:** A *nondeterministic finite state machine* whose states are labeled with boolean variables, which are the evaluations of expressions in that state.

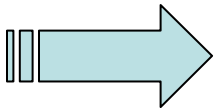
- **State explosion problem:** The size of S is often exponential in |requirements/design

- Model checking problem

K |= phi

Defining Models

- For a complex real-life control systems



Extended Finite State Machine (EFSM)

- FSM with a way to

-modularize the requirements to view them at different levels of detail

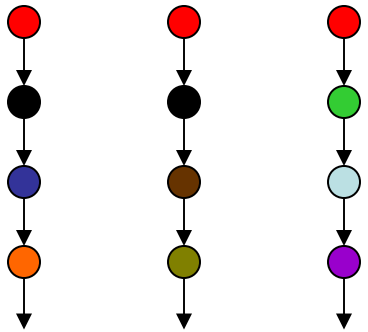
-combine requirements (or design) of components

- state variables and facilities in guards on transitions.

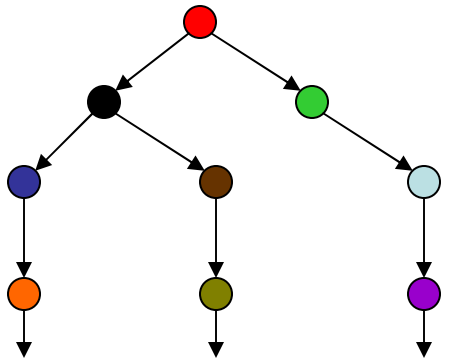
Defining Specifications

- Temporal Logic
  - Express properties of event orderings in time
  - e.g. “Always” when a packet is sent it will “Eventually” be received

- Linear Time
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)



- Branching Time
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)



Safety and Liveness

- Safety properties
  - Invariants, deadlocks, reachability, etc.
  - Can be checked on finite traces
  - “something bad never happens”
- Liveness Properties
  - Fairness, response, etc.
  - Infinite traces
  - “something good will eventually happen”

Linear Temporal Logic (LTL)

- **Connectives of LTL (in addition to not, and, or, implication):**

[]	Always
<>	Eventually
o	Next cycle
(-)	previous cycle
U	Until

- **Examples of LTL**

```
[] p                    // p is always true
<>R -> (P U R)          //  Eventually it is the case that P exists
                        //  before R happens.
```

Linear Temporal Logic (LTL)

- **Safety Examples**
  - **[]({readySignal == 1} -> (){ackSignal == 0})**

- This assertion states "Always, readySignal equals one implies ackSignal equals zero on the next cycle".
  - It makes use of the Always operator (the box) and the Next Cycle operator (the empty parentheses pair).

  - **[]({readySignal == 1} -> (-){ackSignal == 0})**

- This assertion states “Always, readySignal equals one impliesackSignal equals zero on the previous cycle“
  - Makes use of the previous cycle operator (-)

- Liveness Example
  - <>({out1==1} && ()()[]{out2 < 2} && (-){out3==0})
  - This assertion states "Eventually, out1 will equal one, and then two cycles later and always after that, out2 is less than two and on the previous cycle out3 equals zero.
  - This example uses the Eventually operator (the diamond <>), the always operator (the box []), and the Next and Previous Cycle operators (the empty parentheses pair () and the parenthesized minus sign (-), respectively.

SPIN Model Checker

- Kripke structures are described as “programs” in the PROMELA language
  - Kripke structure is generated on-the-fly during model checking
- Automata based model checker
  - Translates LTL formula to Büchi automaton
- By far the most popular model checker
  - 10th SPIN Workshop held with ICSE – May 2003
- Relevant theoretical papers can be found here
  - http://netlib.bell-labs.com/netlib/spin/whatispin.html
- Ideal for software model checking due to expressiveness of the PROMELA language
  - Close to a real programming language
- Gerard Holzmann won the ACM software award for SPIN

Branching Temporal Logic (BTL)

- **Some temporal connectives in Computation Tree Logic (CTL)**

EX φ **true** in current state
if formula φ is **true** in at least one of the next states

EF φ **true** in current state
if there exists some state in some path beginning in current state
that satisfies the formula φ

EG φ **true** in current state
if every state in some path beginning in current state satisfies the formula φ

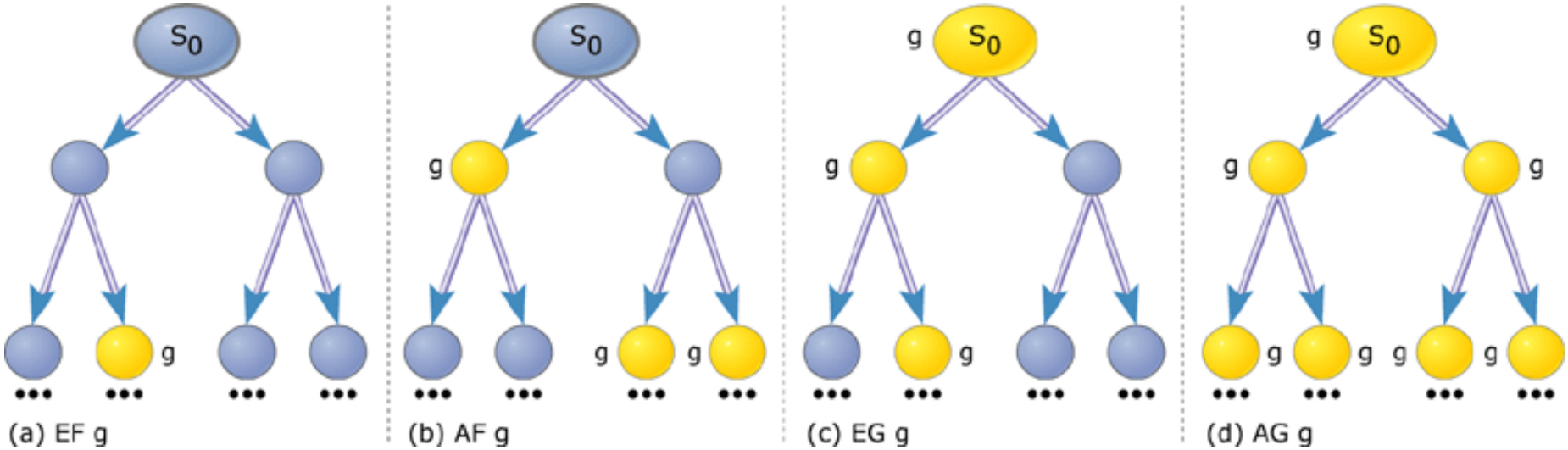
AX φ **true** in current state
if formula φ is **true** in every one of the next states

AF φ **true** in current state
if there exists some state in every path beginning in current state that
satisfies the formula φ

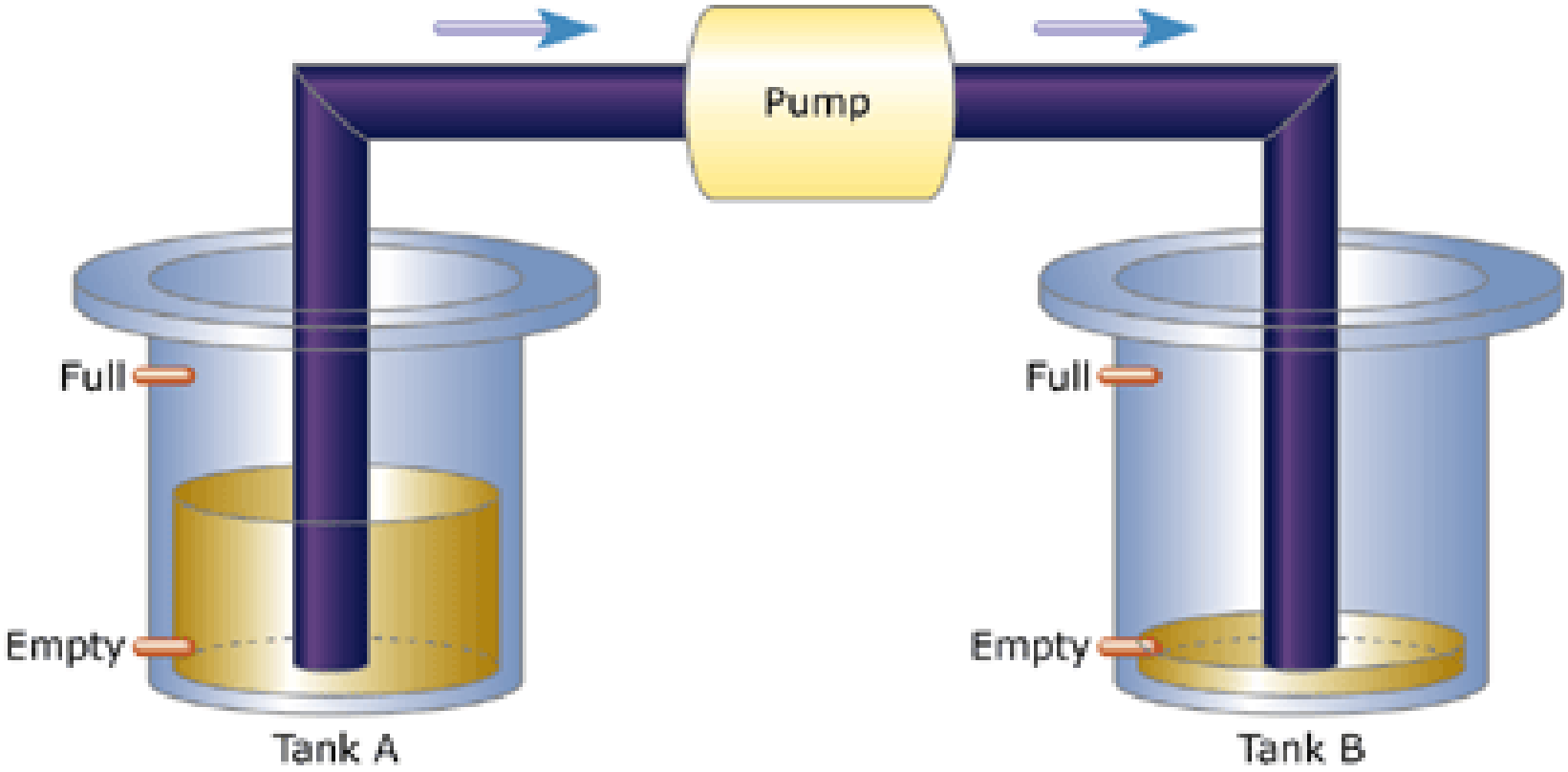
AG φ **true** in current state
if every state in every path beginning in current state satisfies the formula φ

Defining Specifications

- **Intuition for CTL formulae which are satisfied at state s0**



A Simple Two Tank System Example



By Girish Keshav Palshikar, Embedded Systems Programming
http://www.embedded.com/showArticle.jhtml?articleID=17603352

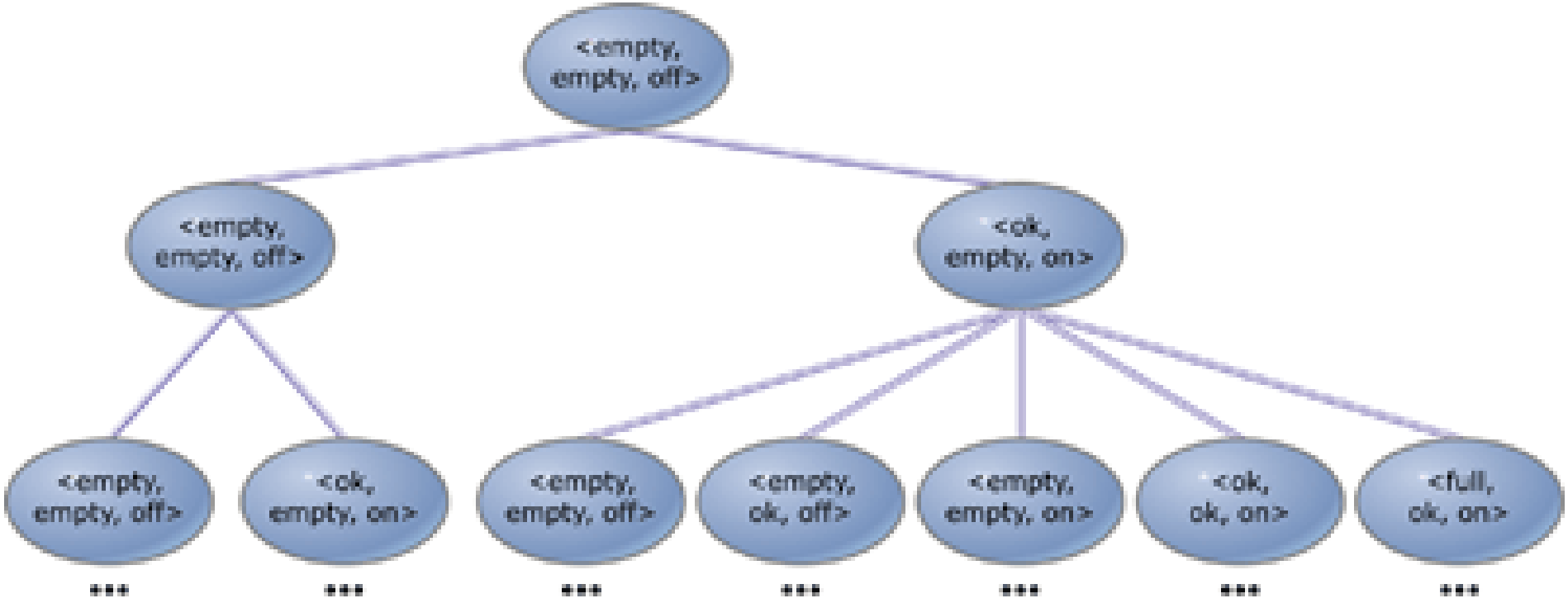
A Simple Two Tank System Example

In symbolic model verifier (SMV) model checking tool, CMU
http://www.cs.cmu.edu/~modelcheck/smv.html

```
MODULE main
VAR
  level_a : {empty, ok, full}; -- lower tank
  level_b : {empty, ok, full}; -- upper tank
  pump : {on, off};
ASSIGN
  next(level_a) := case
     level_a = empty : {empty, ok};
     level_a = ok & pump = off : {ok, full};
     level_a = ok & pump = on : {ok, empty, full};
     level_a = full & pump = off : full;
     level_a = full & pump = on : {ok, full};
     1 : {ok, empty, full};
  esac;
  next(level_b) := case
     level_b = empty & pump = off : empty;
     level_b = empty & pump = on : {empty, ok};
     level_b = ok & pump = off : {ok, empty};
     level_b = ok & pump = on : {ok, empty, full};
     level_b = full & pump = off : {ok, full};
     level_b = full & pump = on : {ok, full};
     1 : {ok, empty, full};
  esac;
  next(pump) := case
       pump = off & (level_a = ok | level_a = full) &
       (level_b = empty | level_b = ok) : on;
       pump = on & (level_a = empty | level_b = full) : off;
       1 : pump; -- keep pump status as it is
    esac;
INIT
  (pump = off)
SPEC
  -- pump if always off if ground tank is empty or up tank is full
  -- AG AF (pump = off -> (level_a = empty | level_b = full))
  -- it is always possible to reach a state when the up tank is ok or full
  AG (EF (level_b = ok | level_b = full))
```

A Simple Two Tank System Example

Initial part of the execution tree for the pump controller system



A Simple Two Tank System Example

Initial part of the execution tree for the pump controller system

-- specification AF pump = on is false

-- as demonstrated by the following execution sequence

-- loop starts here

state 1.1:

level_a = full
level_b = full
pump = off

state 1.2:

A Simple Two Tank System Example

Some other properties

- AF (pump = off)

--- for every path beginning at the initial state, there's a state in that path at which the pump is off.

trivially true at the initial state, since in the initial state itself (which is included in all paths) pump = off is true.

- AG ((pump = off) -> AF (pump = on))

--- it's always the case that if pump is off then it eventually becomes on.

clearly false in the initial state.

- AG AF (pump = off -> (level_a = empty | level_b = full))

---- pump is always off if ground tank is empty or the upper tank is full.

- AG (EF (level_b = ok | level_b = full))

--- it's always possible to reach a state when the upper tank is ok or full.

Issues

- Temporal logic: can be hard to work with
- Translations of requirements models to the input language of model checking engines: cannot be efficiently checked, due to state space explosion
- Counter-examples: do not mean anything to the stakeholders; need to be translated back into the original modeling language.
- Deals only with state-oriented behavioral requirements models