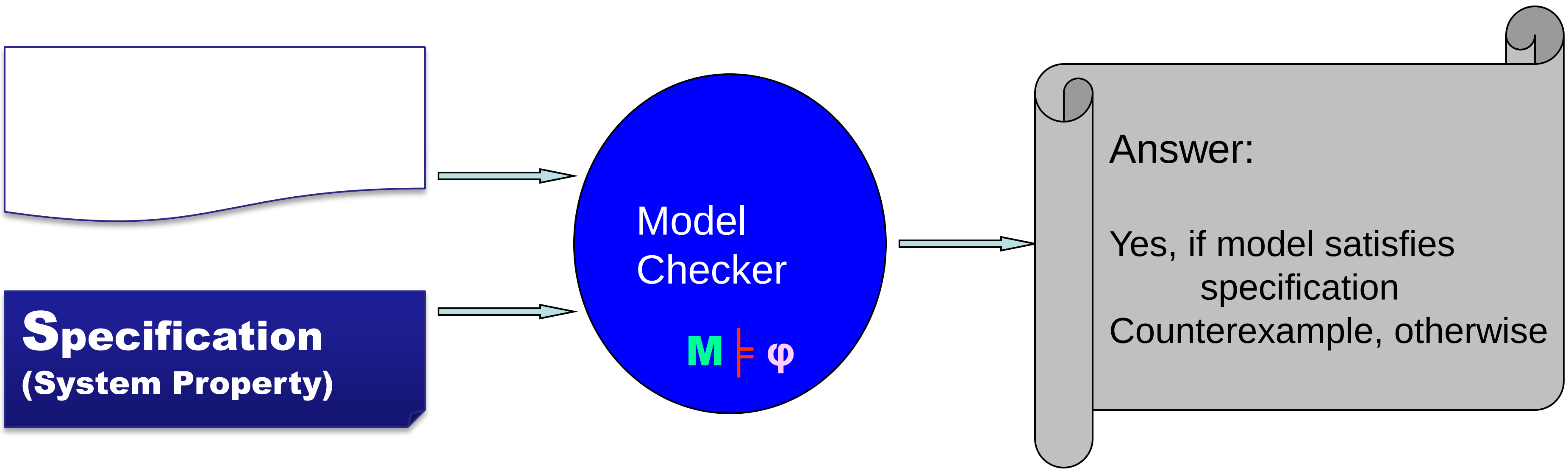
Model Checking

Model Checking Process



For increasing our confidence in the correctness of the model:

- ❑ Verification: The model satisfies important system properties
- ❑ Debugging: Study counter-examples, pinpoint the source of the error, correct the model, and try again

Digicode

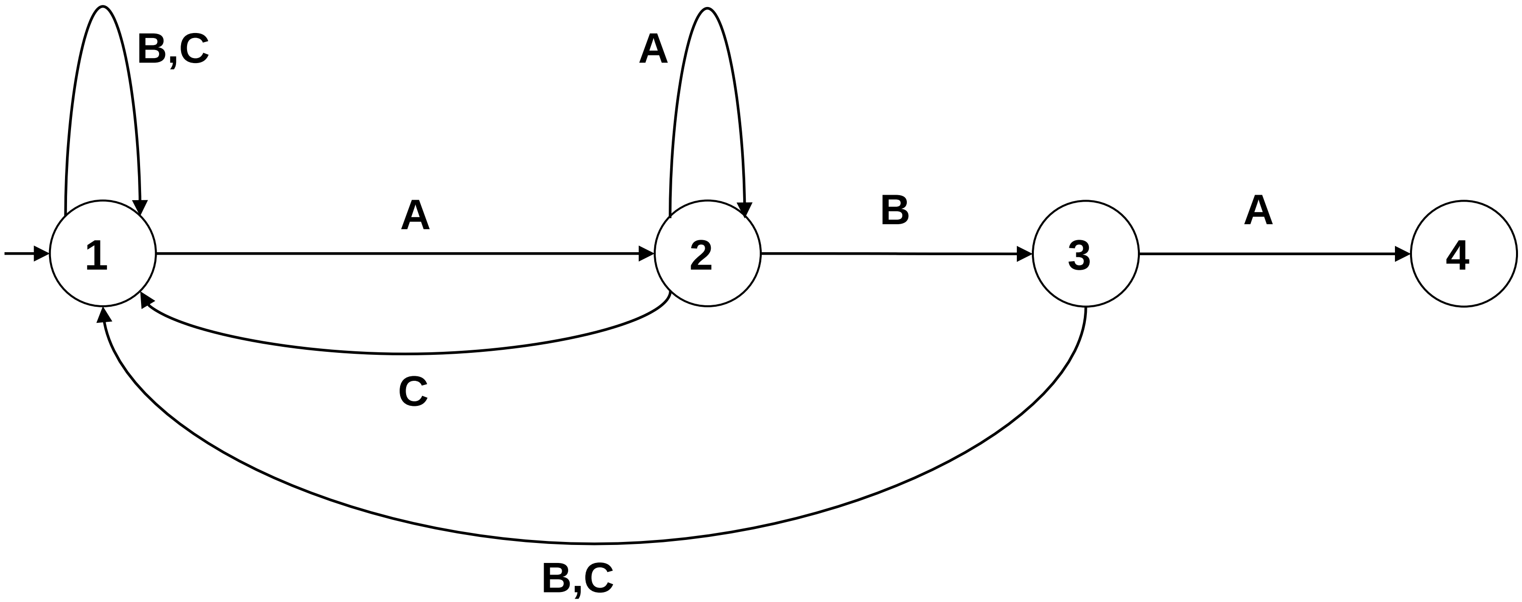
- Consider a program that checks the input given to a bike lock
- Assume we have just three possible entries: A,B,C.
- The bike lock opens only if the combination ABA is digited
- This program can be represented by an automaton with 4 states and 9 transitions

Digicode

```
typedef enum State{s1, s2,  s3, s4} State;
int main(){
		State s=s1;
		while (true){
				read(x);
				switch (s) {
				case s1:
						if (x==A) then s=s2; break;
				case s2:
						if (x==B) then s=s3;
								else if (x!=A) then s=s1;
						break;
				case s3:
						if (x==A) then {s=s4; return 1;}
								else s=s1;
						break;
				default:	break;

				}
		}
}
```

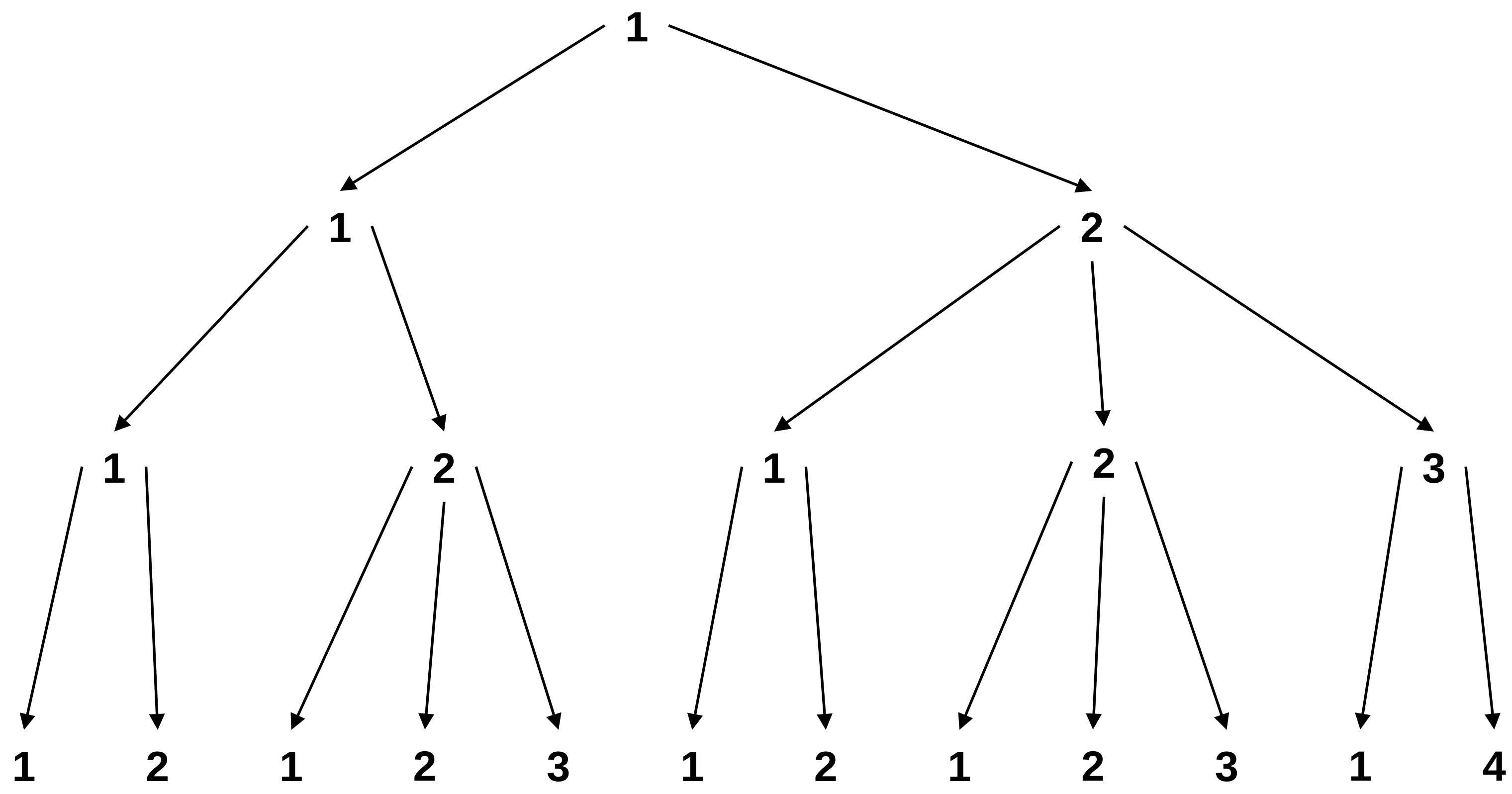
Digicode



Executions

- An execution is a sequence of states that describes a possible evolution of the system.
- For instance, 1121, 12234, 112312234 are possible executions of the digicode
- The possible executions of the digicode are:
  1
  11,12
  111,112,121,122,123
  1111,1112,1121,1122,1123,1211,1212,1221,1222,1223,1231,1234
  …

The execution tree



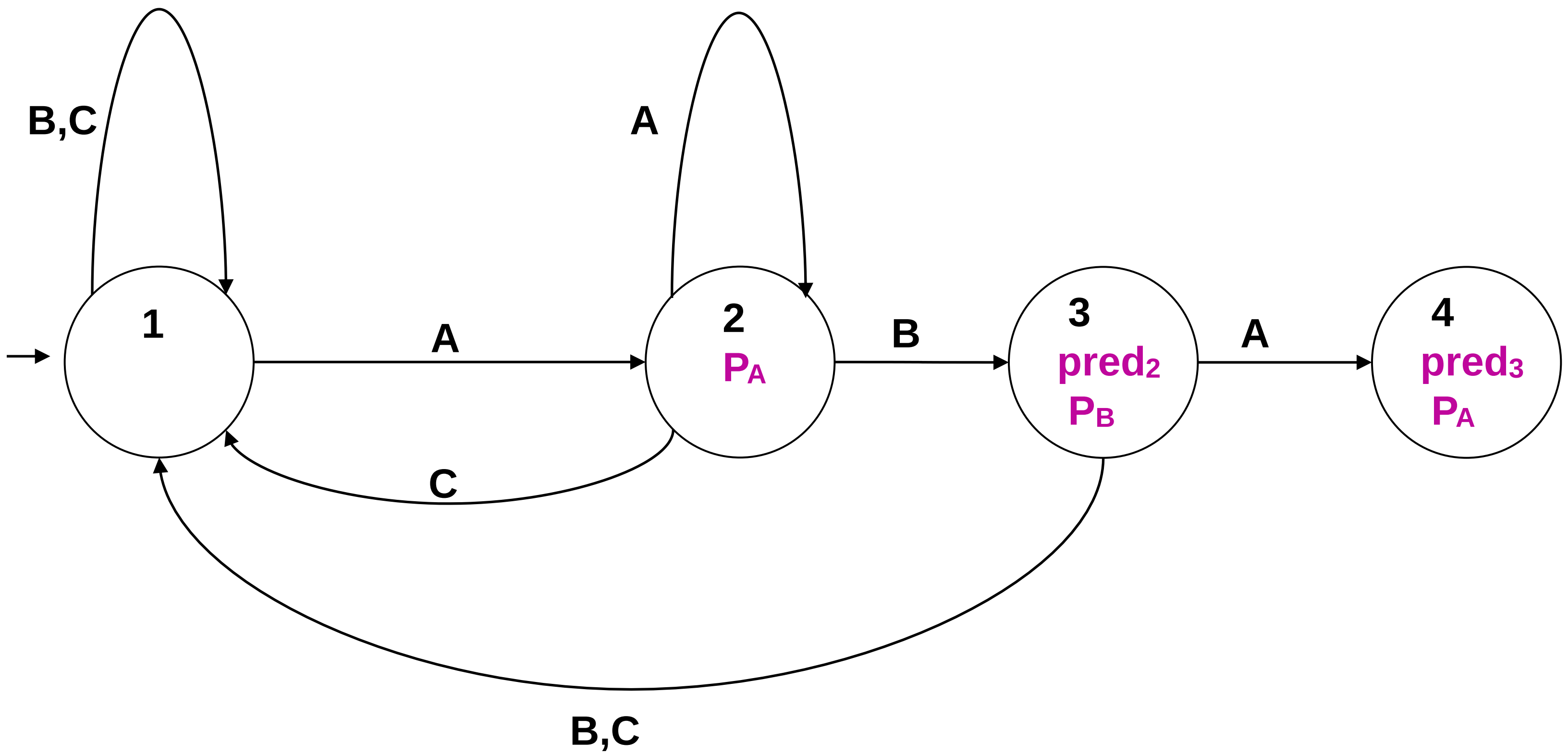
Properties

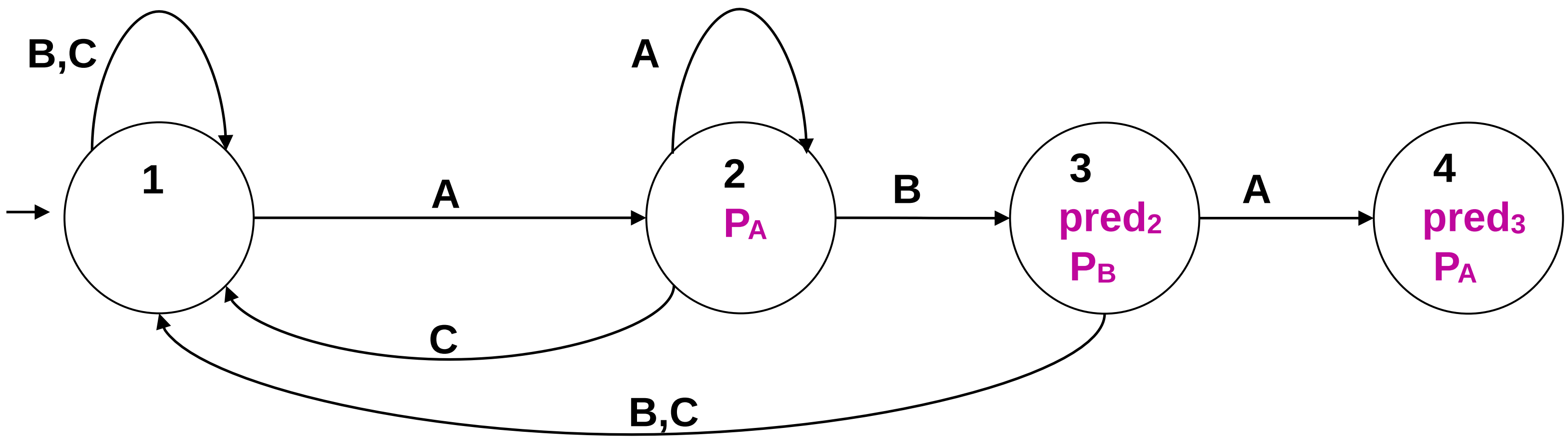
- Each state of the automaton is associated with some elementary properties that are true when the system is in that state.
- For instance, the property “the bike lock is open” holds on state 4, but it does not hold in the states 1,2 and 3.
- We would like to show some properties like
  - If the bike lock opens, then the last three letters that have been digited are ABA, in this order
  - If the input contains a sequence of letters that ends with ABA, the bike lock opens.

Atomic formulas

- In our digicode example, the basic formulas are
  - P_A : the last input digited is A
  - P_B : the last input digited is B
  - P_C : the last input digited is C
  - $pred_1$: the previous state is state 1
  - $pred_2$: the previous state is state 2
  - $pred_3$: the previous state is state 3

Adding atomic formulas to the automaton





- Let us prove that of the bike lock opens, then the last digits inserted were ABA
- Consider an execution that opens the lock, i.e. that ends in state 4
- As in 4 the formula $pred_3$ holds, the execution should end with 34
- But in state 3 the formula $pred_2$ holds. Therefore the execution should end with 234.
- In state 2 and in state 4 the formula P_A holds, and in state 3 the formula P_B holds. Therefore the last three digits inserted should be: ABA.

Defining Models

Model
(System Requirements)

- Kripke Structure

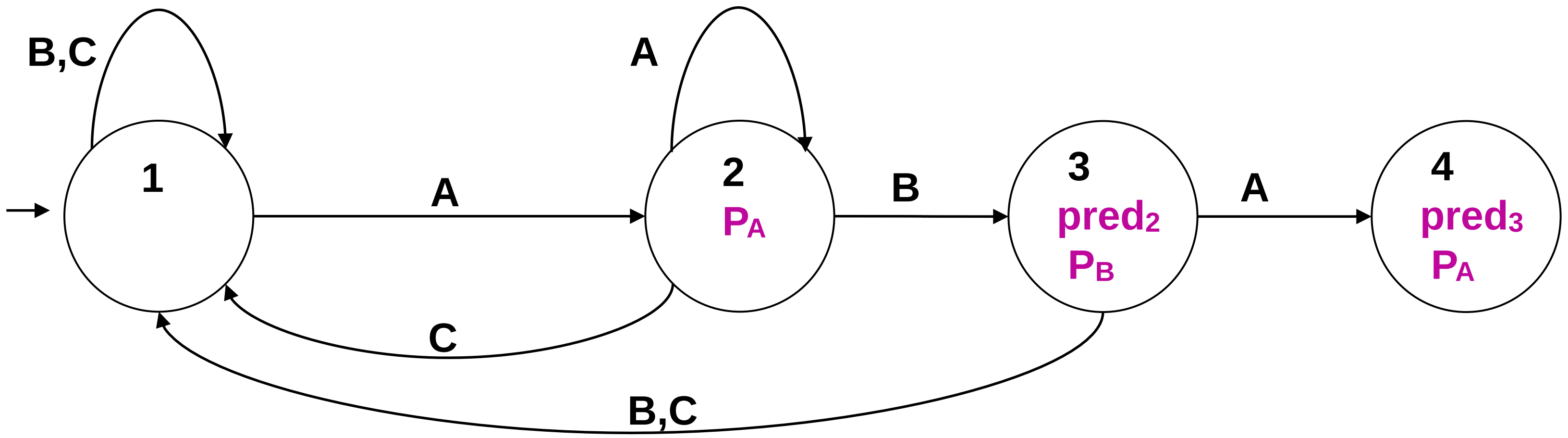
$$K = < S, P, R, L, s_0 >$$

- S: the set of possible global states
- P: a non-empty set of atomic propositions $\{p_1, . . ., p_k\}$ which express atomic properties of the global states, e.g., being an initial state, being an accepting state, or that a particular variable has a special value.
- $R \subseteq S \times S$: a transition relation s.t. R(s,s') if s to s' is a possible atomic transition
- $L: S \rightarrow 2^P$: a labeling function which defines which propositions hold in which states.
- $s_0 \in S$: the initial state

- Model checking : A ***model checker*** **check**s whether a system, interpreted as an automaton, is a (Kripke) **model** of a property expressed as a temporal logic formula.

$$K \models \varphi$$

The digicode automaton



- $S = \{1,2,3,4\}$
- $P = \{P_A, P_B, P_C, pred_1, pred_2, pred_3\}$
- $R = \{ (1,A,2),(1,B,1),(1,C,1),(2,A,2),(2,B,3),(2,C,1),$
  $(3,A,4),(3,B,1),(3,C,1) \}$
- $L = \{ 1 \mapsto \emptyset, 2 \mapsto \{P_A\}, 3 \mapsto \{P_B, pred_2\}, 4 \mapsto \{P_A, pred_3\} \}$
- $s_0 = 1$

Mutual Exclusion Example

Model
(System Requirements

- Two process mutual exclusive 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 --- $N_1 N_2 S_0$

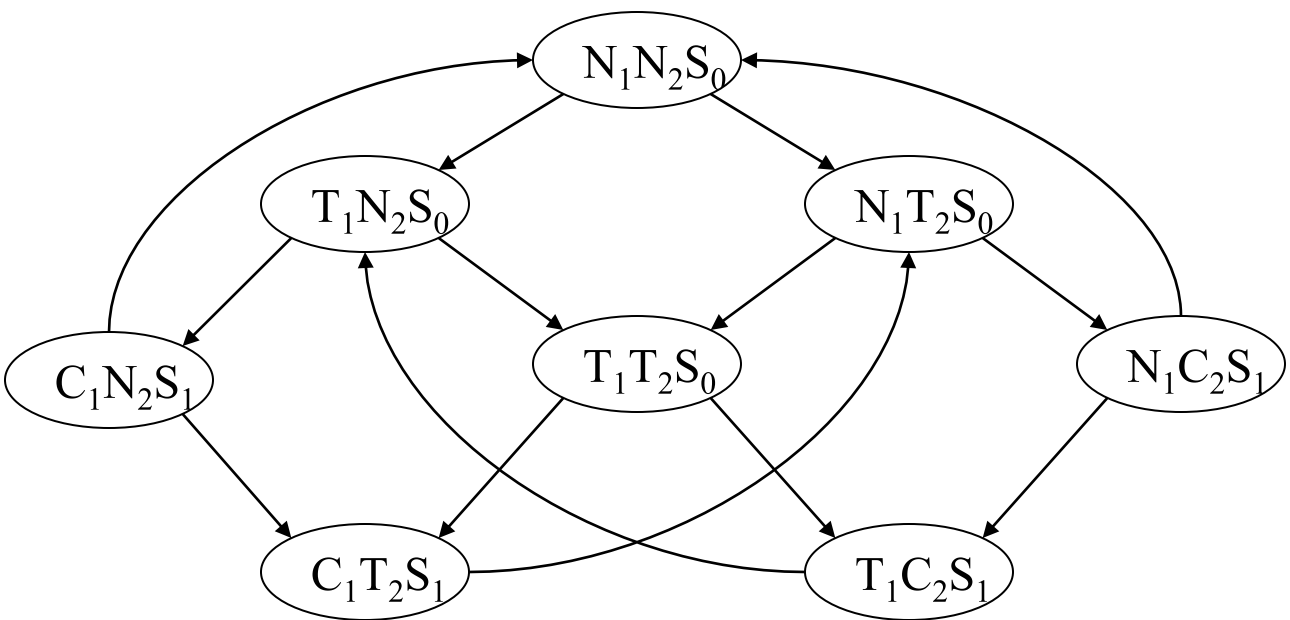
$$
\begin{array}{lcl}
N_1 & \rightarrow & T_1 \\
T_1 \wedge S_0 & \rightarrow & C_1 \wedge S_1 \\
C_1 & \rightarrow & N_1 \wedge S_0
\end{array}
\quad || \quad
\begin{array}{lcl}
N_2 & \rightarrow & T_2 \\
T_2 \wedge S_0 & \rightarrow & C_2 \wedge S_1 \\
C_2 & \rightarrow & N_2 \wedge S_0
\end{array}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

Mutual Exclusion Example

Model
(System Requirements)

•Initially both processes are in the Non-critical state and the semaphore is available --- $N_1\ N_2\ S_0$

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



Mutual Exclusion Example

Specification
(System Property)

Specification – Desirable Property

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

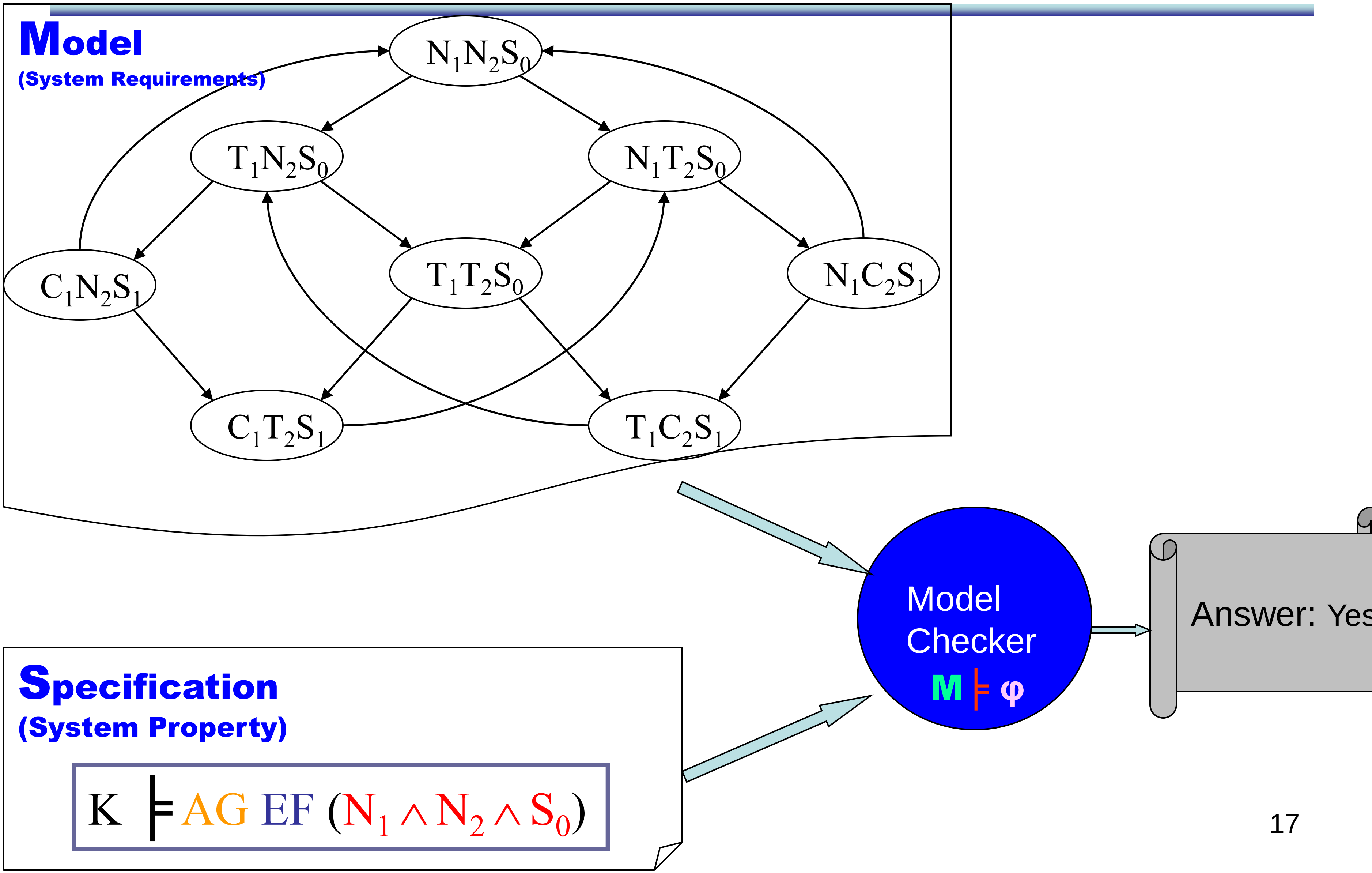
$$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$$

Kripke structure

CTL (Computation Tree Logic)

$M \models \varphi$

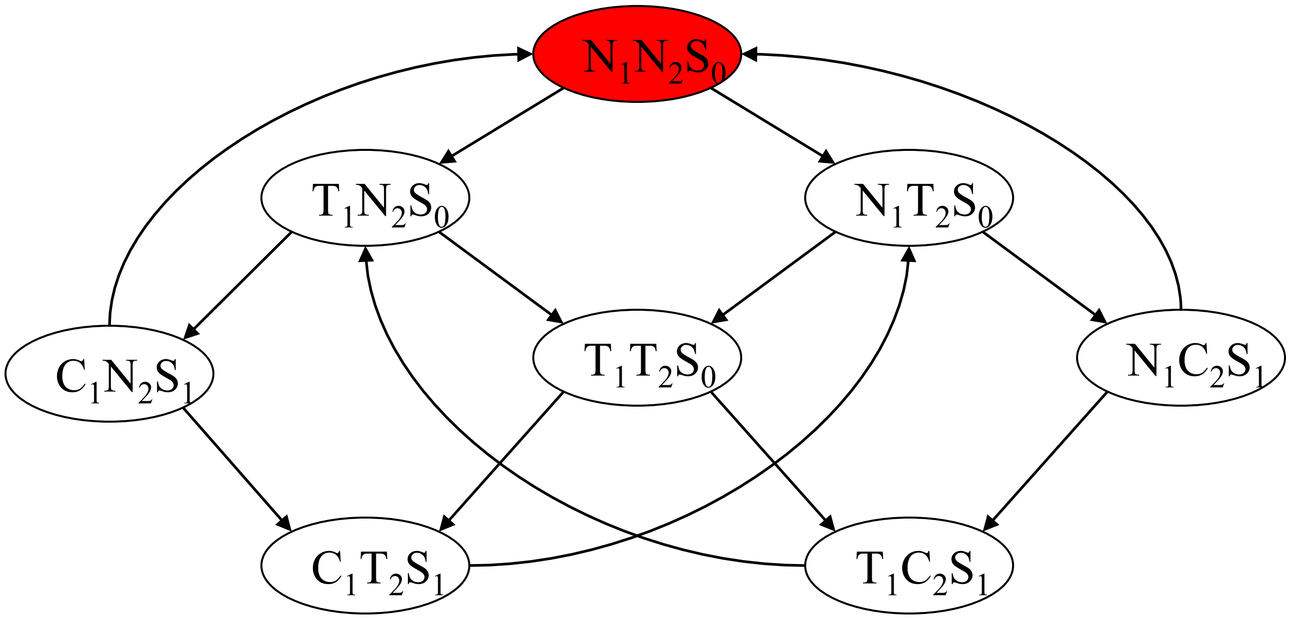
Mutual Exclusion Example



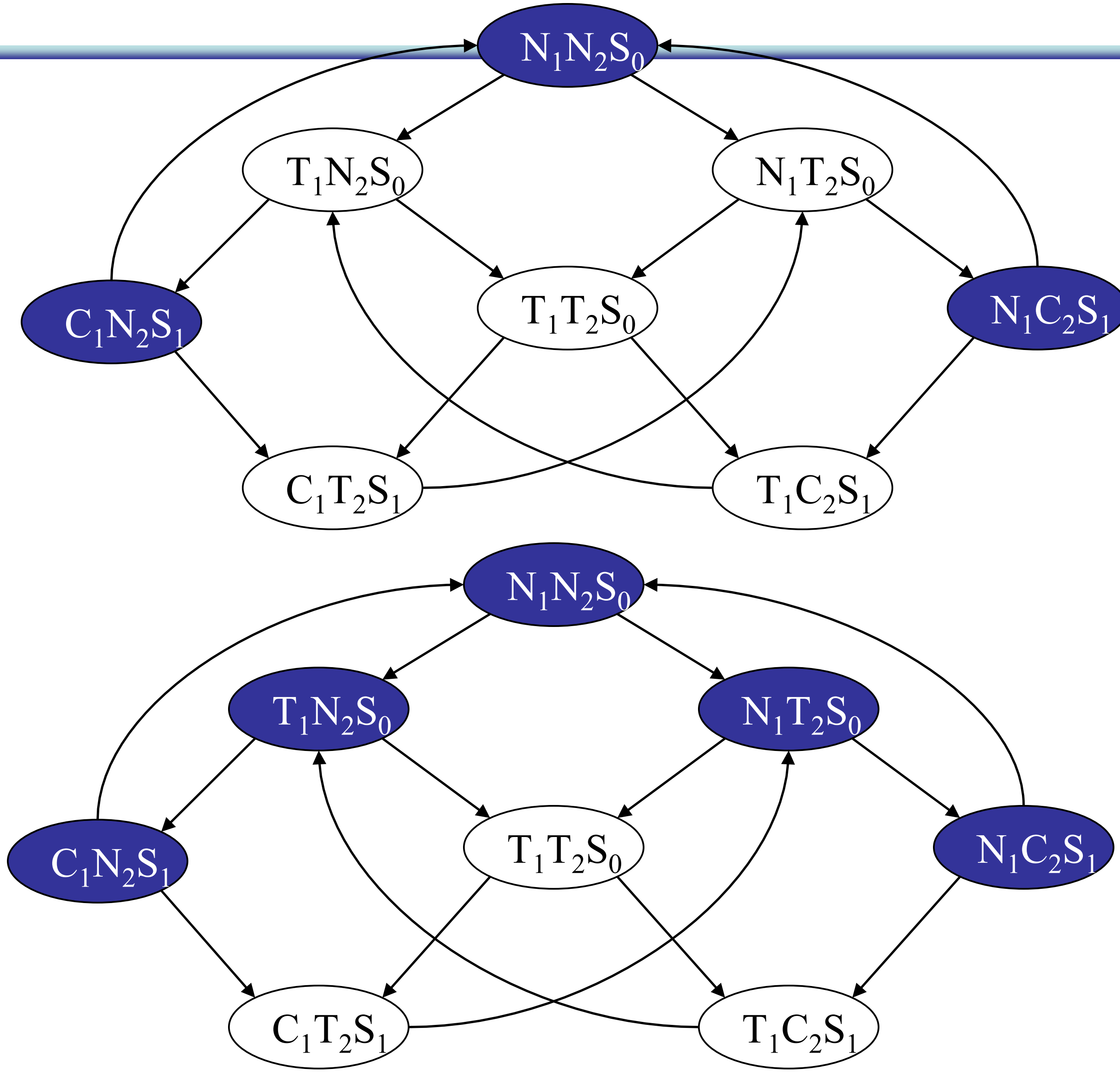
Mutual Exclusion Example

Answer: Yes

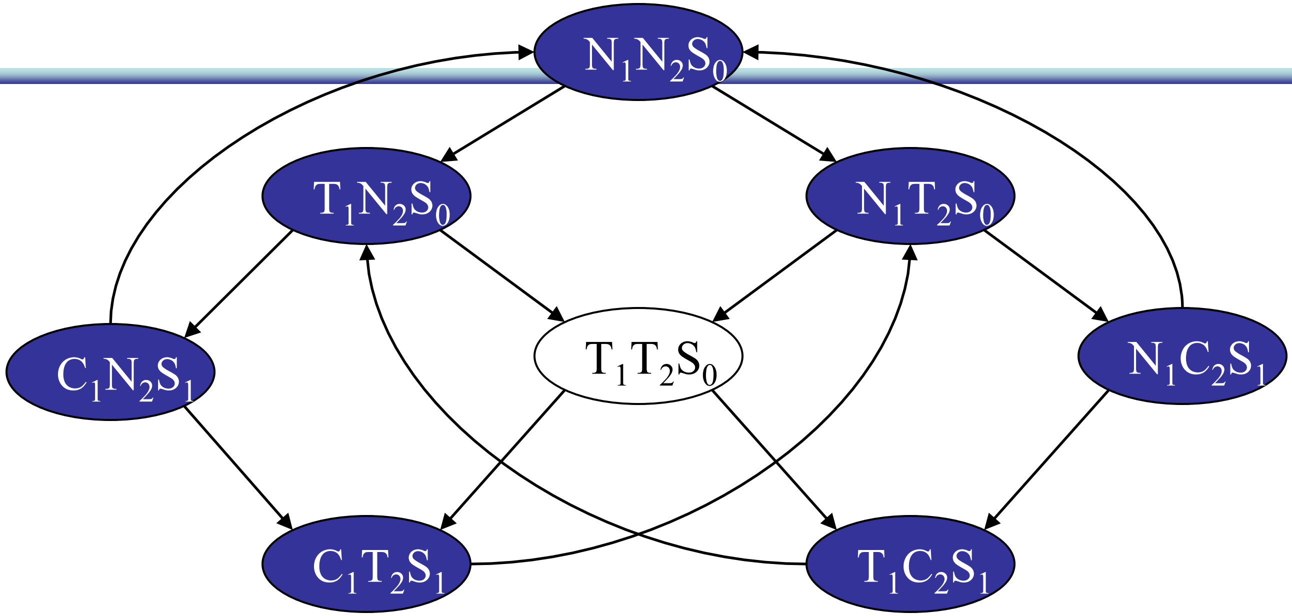
A Proof: *For All* possible behaviors

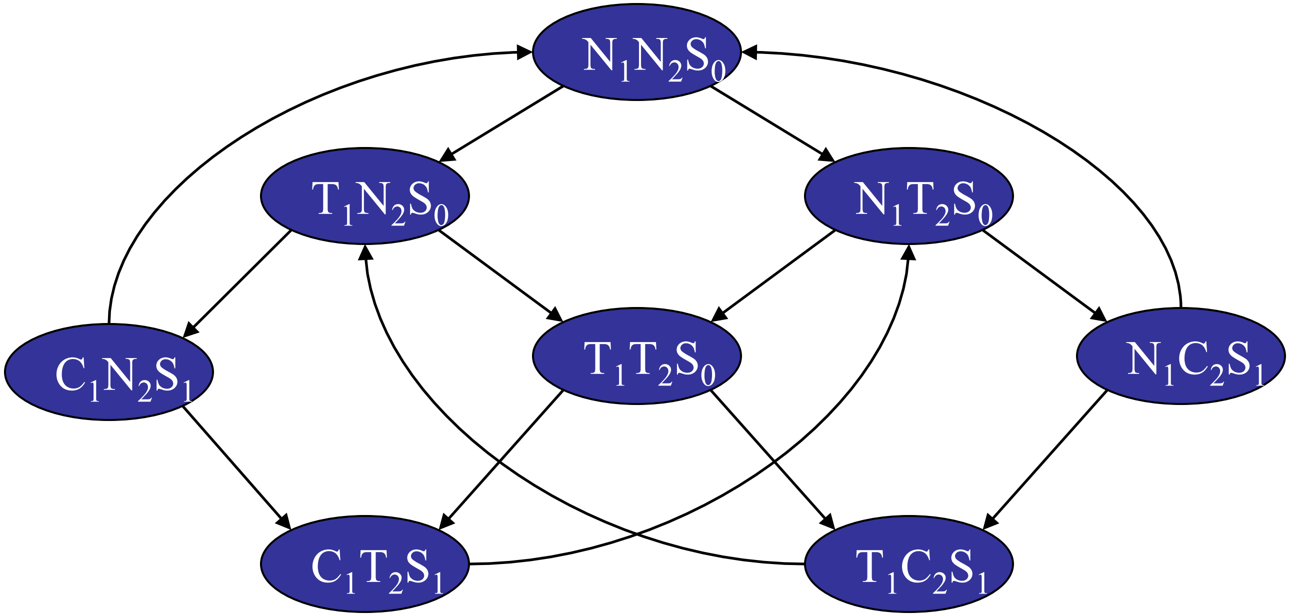


Mutual Exclusion Example



Mutual Exclusion Example





Mutual Exclusion Example

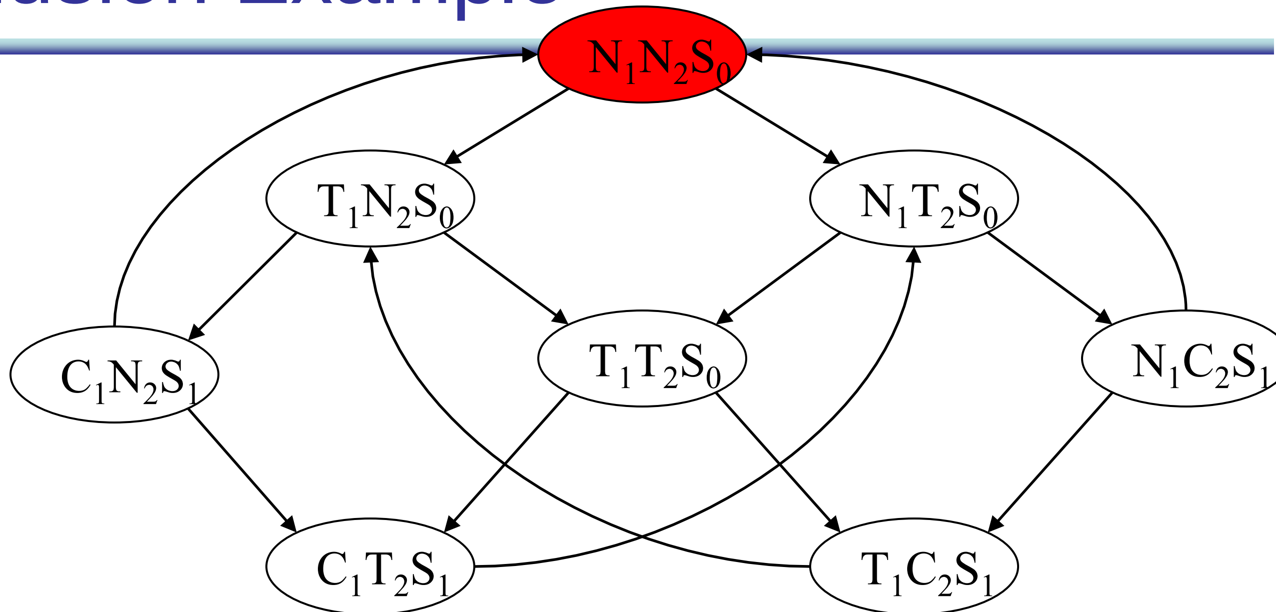
Specification – Desirable Property

No matter where you are there is

no *way to get to the initial state*

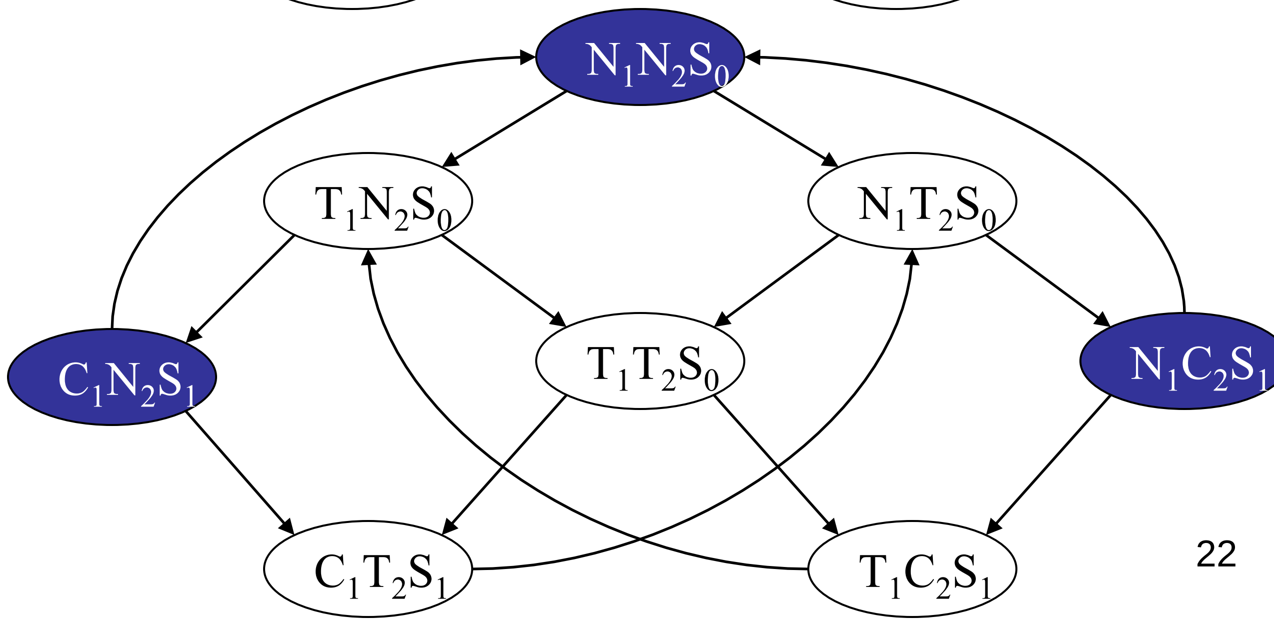
- $\boxed{K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)}$

Mutual Exclusion Example

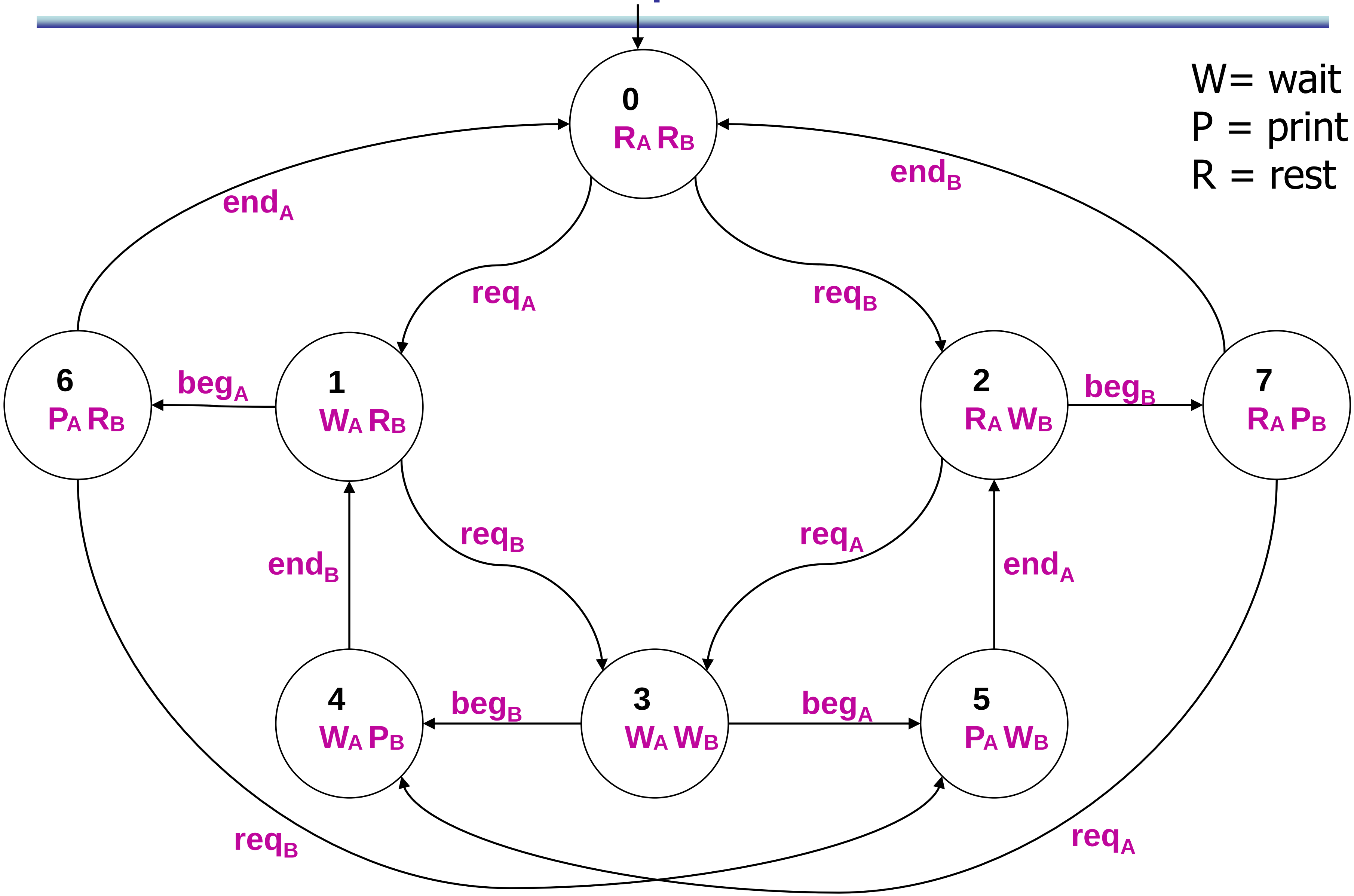


Answer: No

Counterexample



Printer Monitor Example



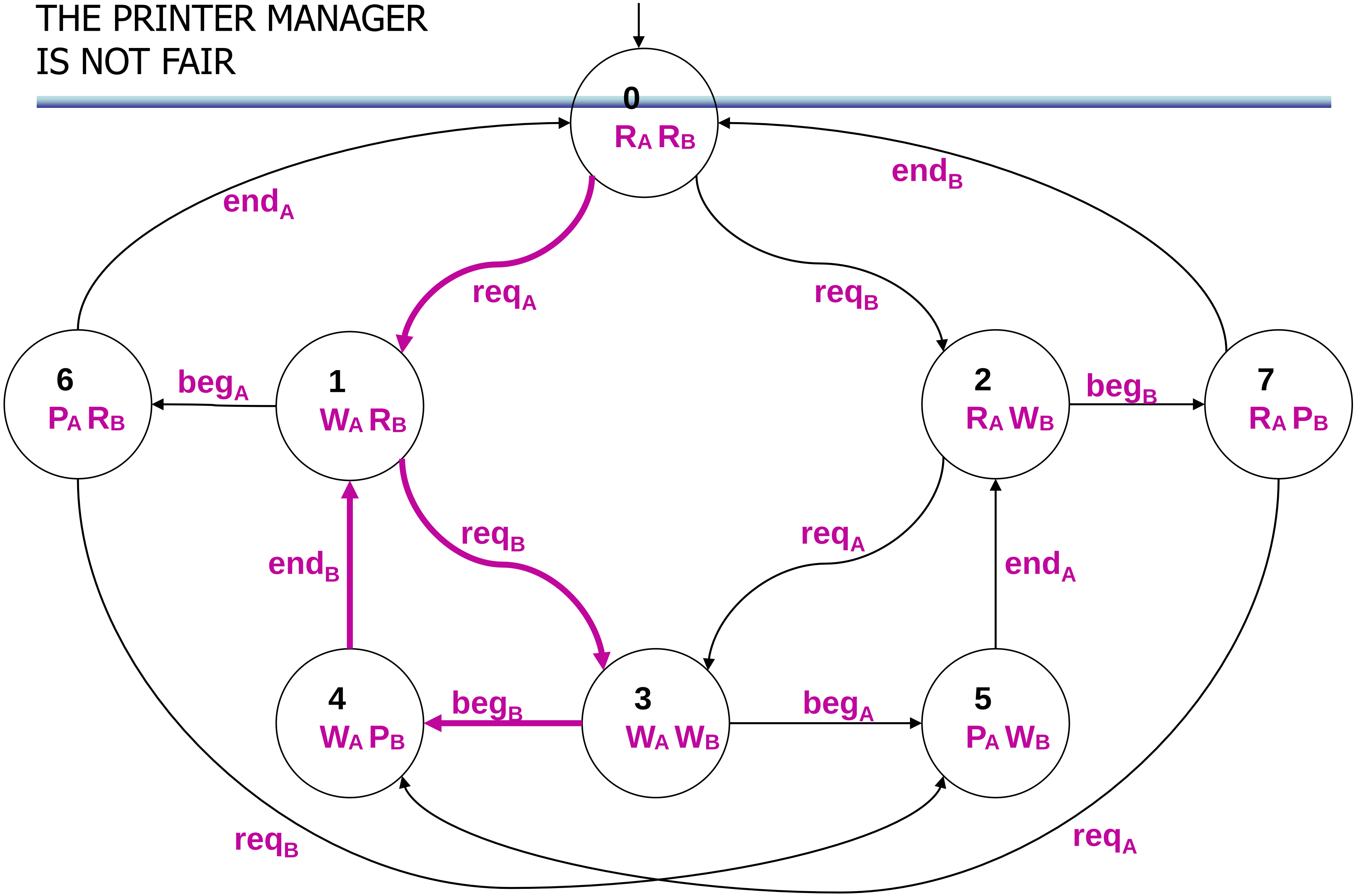
Printer Monitor Example

Desired properties

- In every execution, each state in which P_A holds is preceeded by a state in which W_A holds
  - Easy to verify!

- In every exection every state in which W_A is followed (sooner or later) by a state in which P_A holds.
  - This property does not hold! And the model checker will produce a counterexample.

THE PRINTER MANAGER
IS NOT FAIR



Counterexample: 0 1 3 4 1 3 4 1 3 4 1 3 4 1 3 4 1 3 4 ...

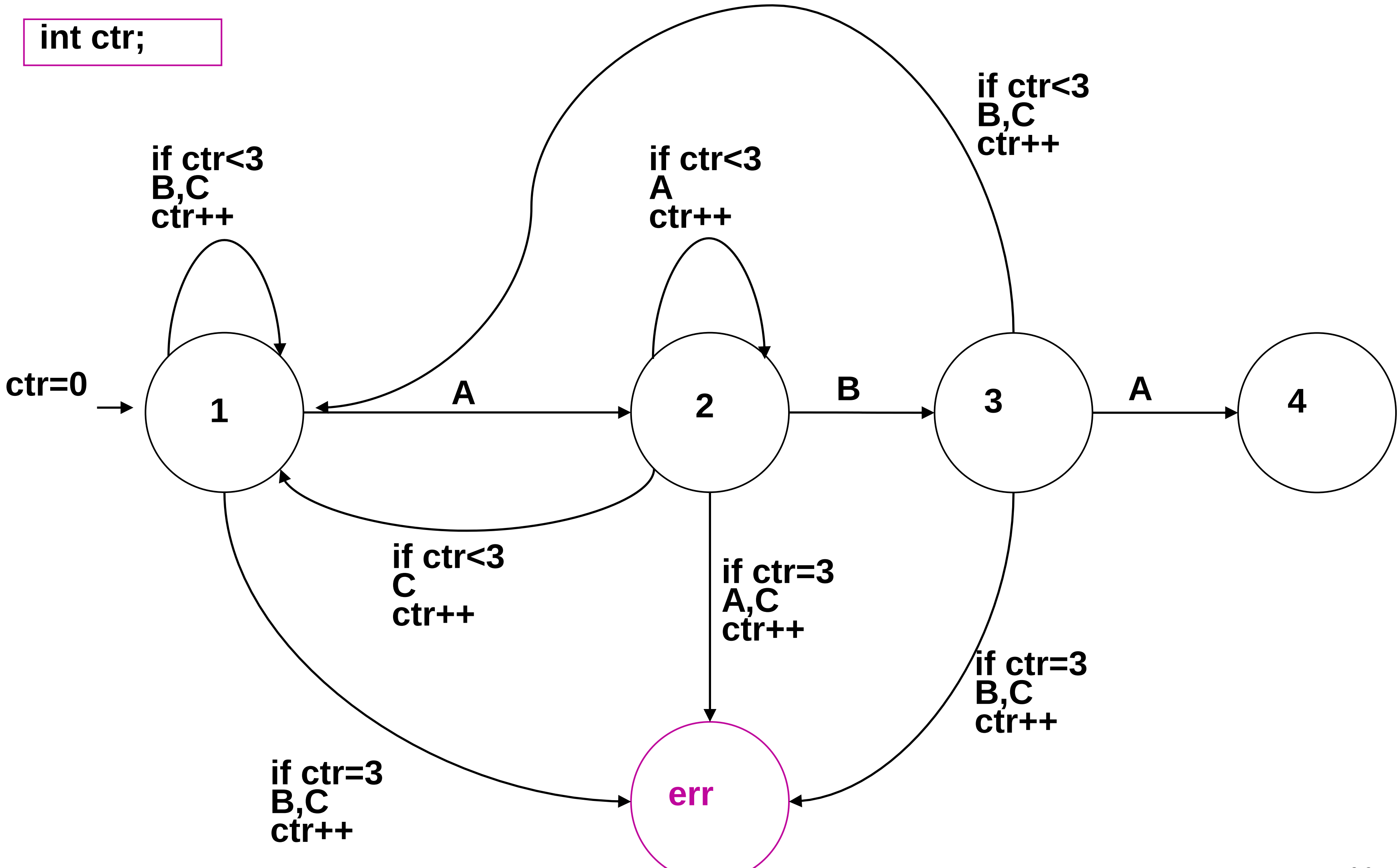
Automata with variables

- When we model a system we would like to represent also variables
- Program = Control + Data
  - The pair <state, transition> represents control
  - Variables represent data
- Example: In the digicode example, if we want to limit the number of attempts (max 3 errors), we need a counter that take count of the errors.

Interaction automaton - variables

- The automaton interacts with the variables in two ways:
  - Assignment: a transition may modify one or more variables
  - Guard: a transition may be constrained by the status of the variables

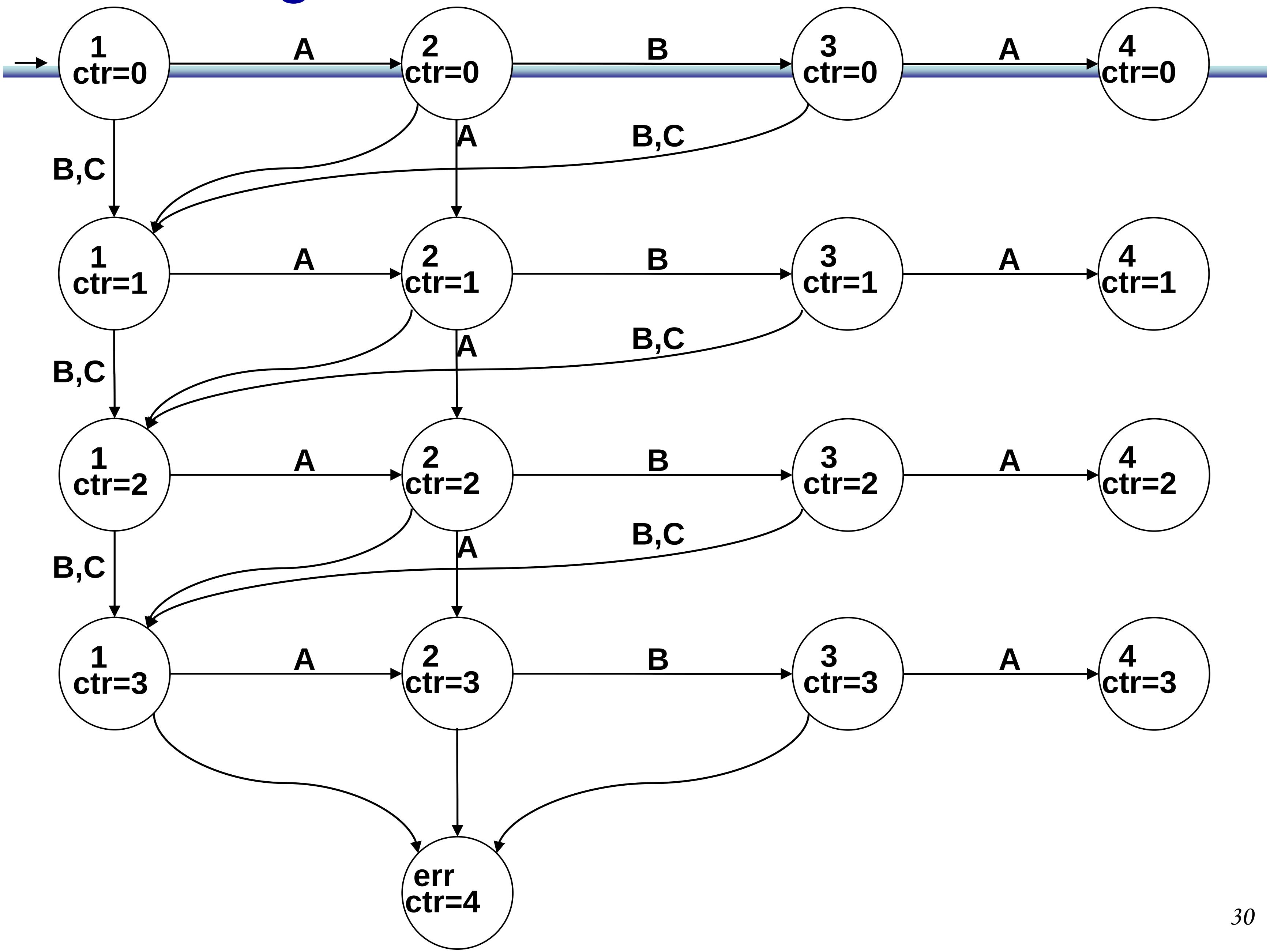
Dealing with variables and control stms



Unfolding

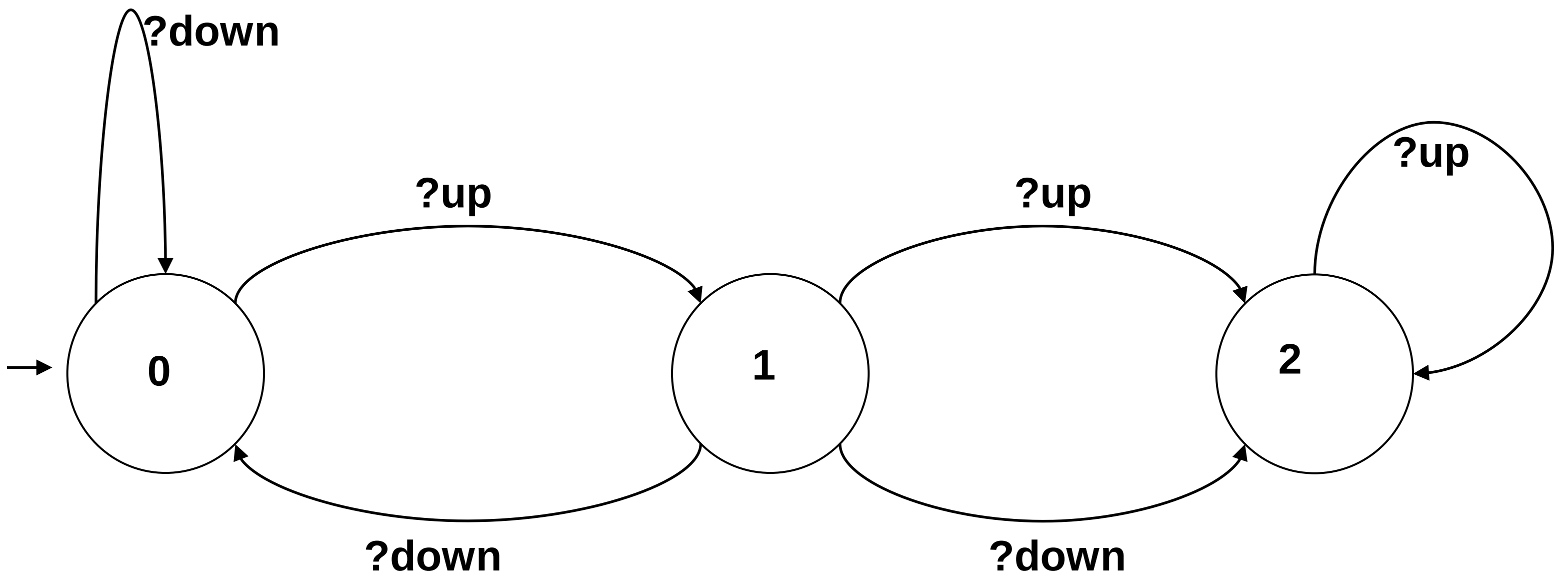
- Automata with variables can be expressed in automata state graph where only state transactions appear
- In this case we spak about a "transition system"
- The states of an unfolded automaton are called global states

Unfolding

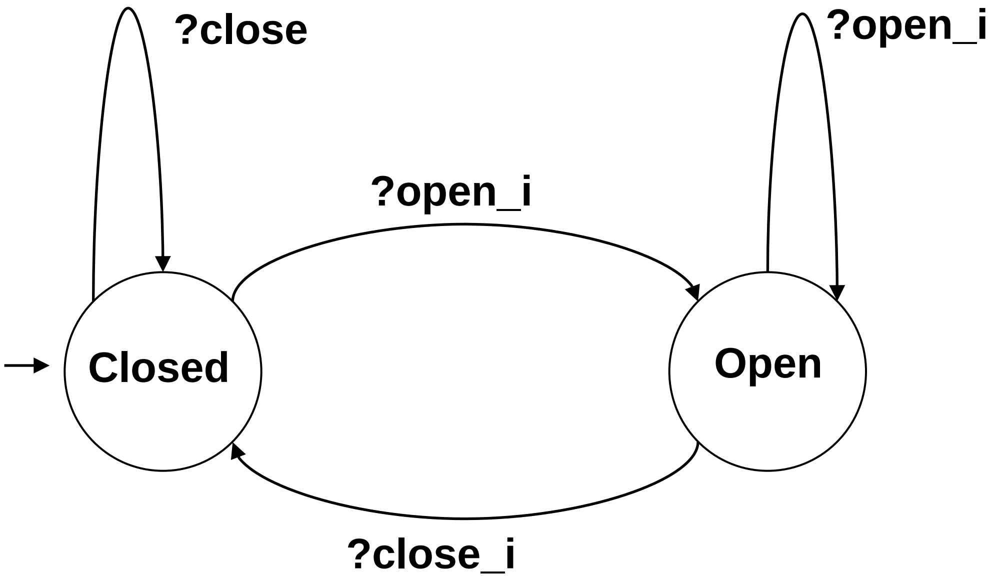


1
ctr=0
A
2
ctr=0
B
3
ctr=0
A
4
ctr=0
B,C
A
B,C
1
ctr=1
A
2
ctr=1
B
3
ctr=1
A
4
ctr=1
B,C
A
B,C
1
ctr=2
A
2
ctr=2
B
3
ctr=2
A
4
ctr=2
B,C
A
B,C
1
ctr=3
A
2
ctr=3
B
3
ctr=3
A
4
ctr=3
err
ctr=4

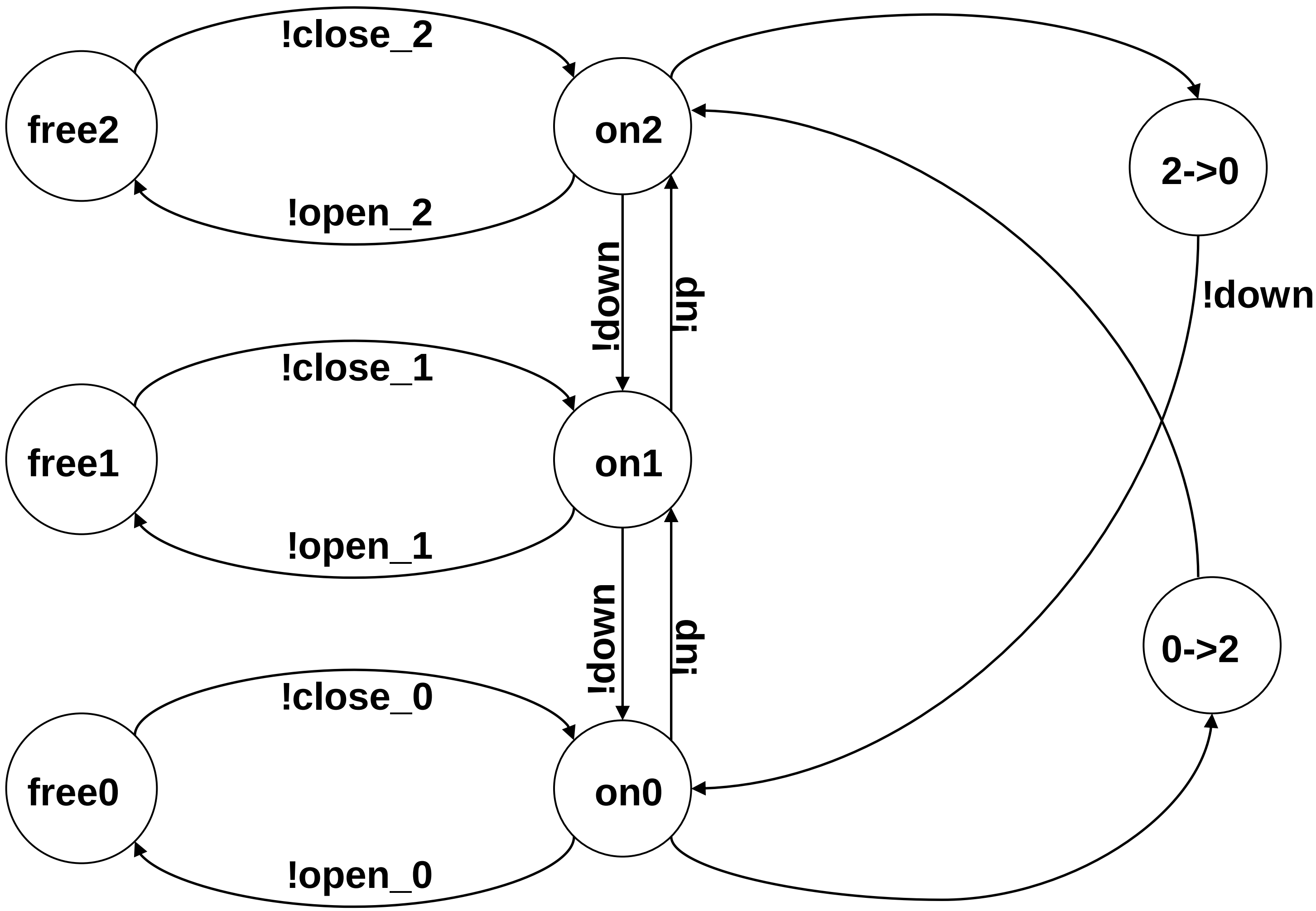
Syncronization: an elevator



The doors at the different floors



The controller



The resulting automaton

- The system is represented by the product of 5 automata (3 doors, the elevator , the controller)
- The constraints are represented by conditions on the tranactions:

Sync={ (?open0,-,-,-,!open0), (?close0,-,-,-,!close0),
(-,?open1,-,-,!open1), (-,?close1,-,-,!close1),
(-,-,?open2,-,!open2), (-,-,?close2,-,!close2),
(-,-,-,?down,!down), (-,-,-,?up,!up) }

Desired properties

- The door at a given floor does not opens if the elevator is at a different floor.
- The elevator does not move if one door is still open