Introduction to Formal Methods

Part I. Principles and Techniques

Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr

Introduction

- Text
  - System and Software Verification : Model-Checking Techniques and Tools

- In this book, you will find enough theory
  - to be able to assess the relevance of the various tools,
  - to understand the reasons behind their limitations and strengths, and
  - to choose the approach currently best suited for your verification task.

- Part I : Principles and Techniques
- Part II : Specifying with Temporal Logic
- Part III : Some Tools

Chapter 1. Automata

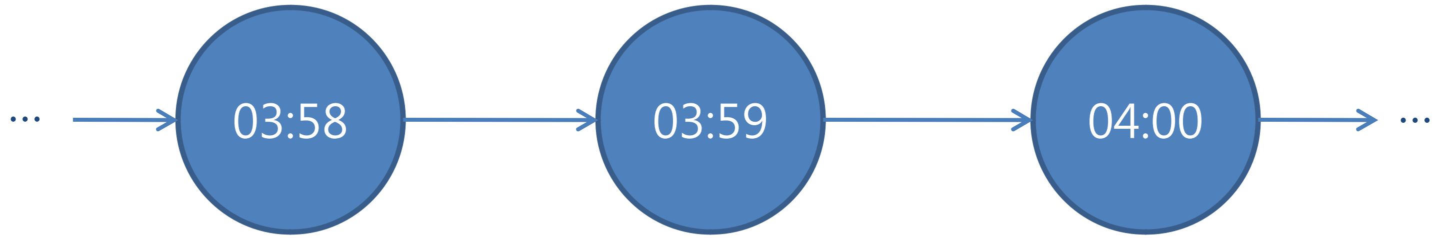
- Model checking consists in verifying some properties of the model of a system.
- Modeling of a system is difficult
  - No universal method exists to model a system
  - Best performed by qualified engineers

- This chapter describes <u>a general model</u> which serves as <u>a basis</u>.

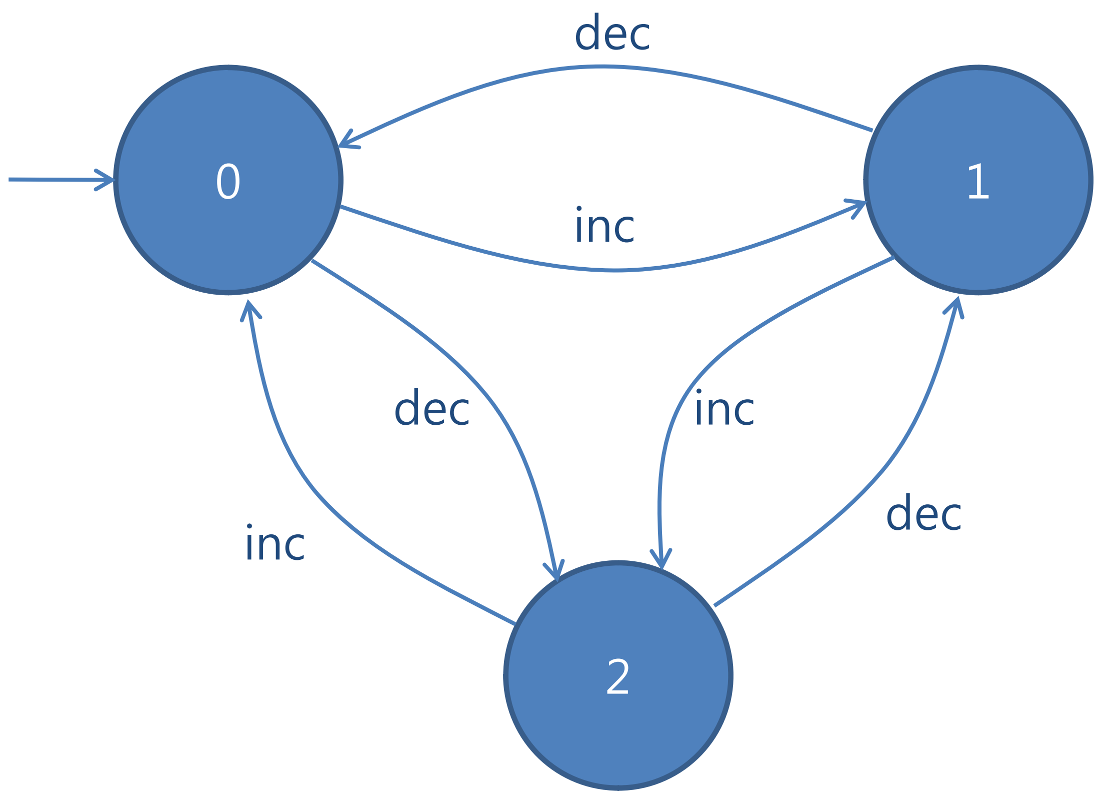
- Organization of Chapter 1
  - Introductory Examples
  - A Few Definitions
  - A Printer Manager
  - A Few More Variables
  - Synchronized Product
  - Synchronization by Messaging Passing
  - Synchronization by Shared Variables

1.1 Introductory Examples

- (Finite) Automata
  - Best suited for verification by model checking techniques
  - A machine evolving from one *state* to another under the action of *transitions*
  - Graphical representation

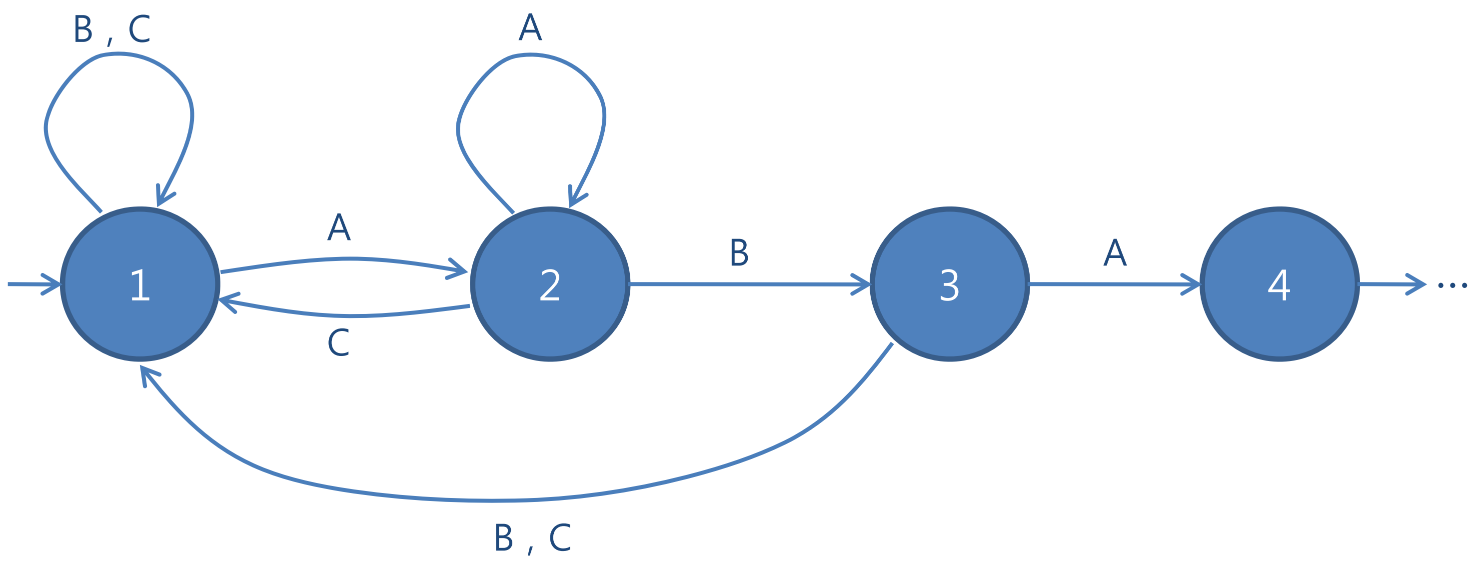


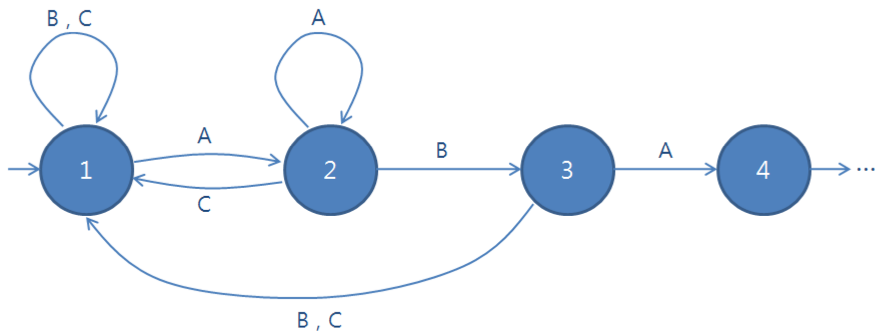
An automate model of a digital watch (24x60=1440 states)



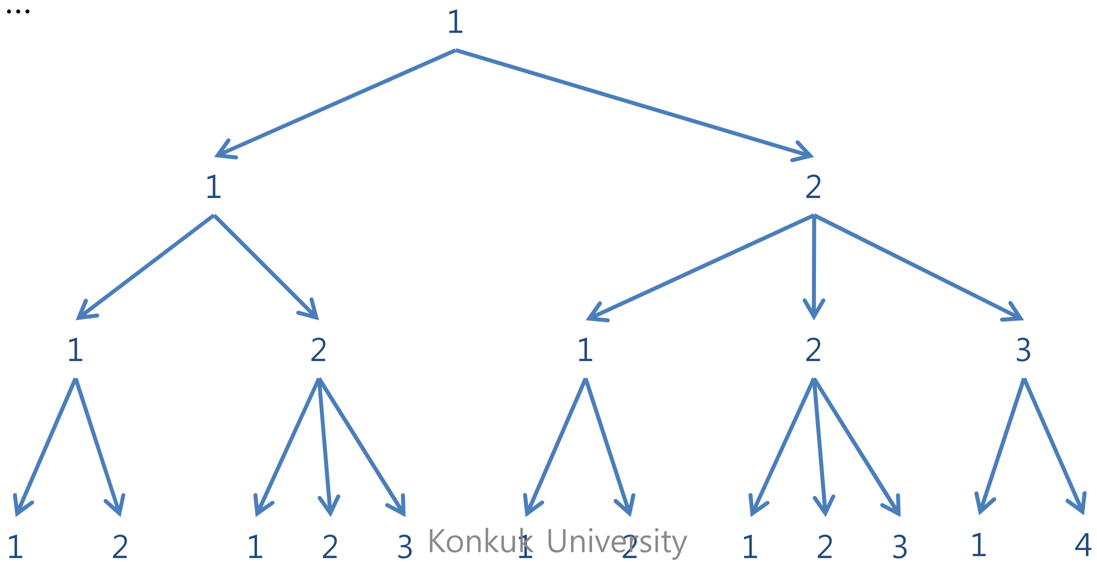
A_{c3} : a module 3 counter

- A digicode door lock example
  - Controls the opening of office doors
  - The door opens upon the keying in of the correct character sequence, irrespective of any possible incorrect initial attempts.
  - Assumes
    - 3 keys A, B, and C
    - Correct key sequence : ABA



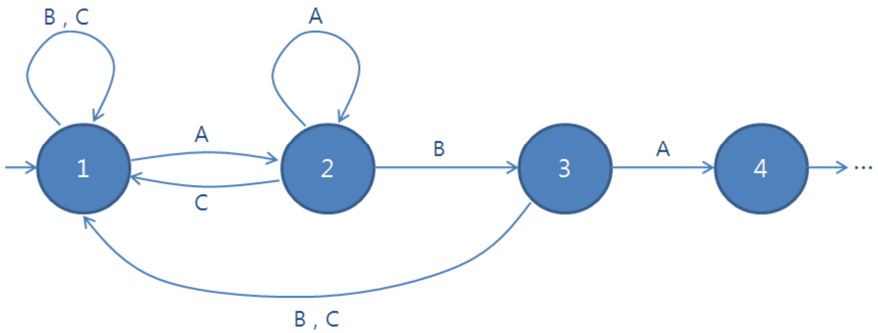


- Two fundamental notations
  - execution
    - A sequence of states describing one possible evolution of the system
    - Ex. 1121 , 12234 , 112312234 ← 3 different executions
  - execution tree
    - A set of all possible executions of the system in the form of a tree
    - Ex. 1
      11, 12
      111, 112, 121, 122, 123
      1111, 1112, 1121, 1122, 1123, 1211, 1212, 1221, 1222, 1223, 1231, 1234
      …



- We associate with each automaton state a number of elementary properties which we know are satisfies, since our goal is to verify system model properties.

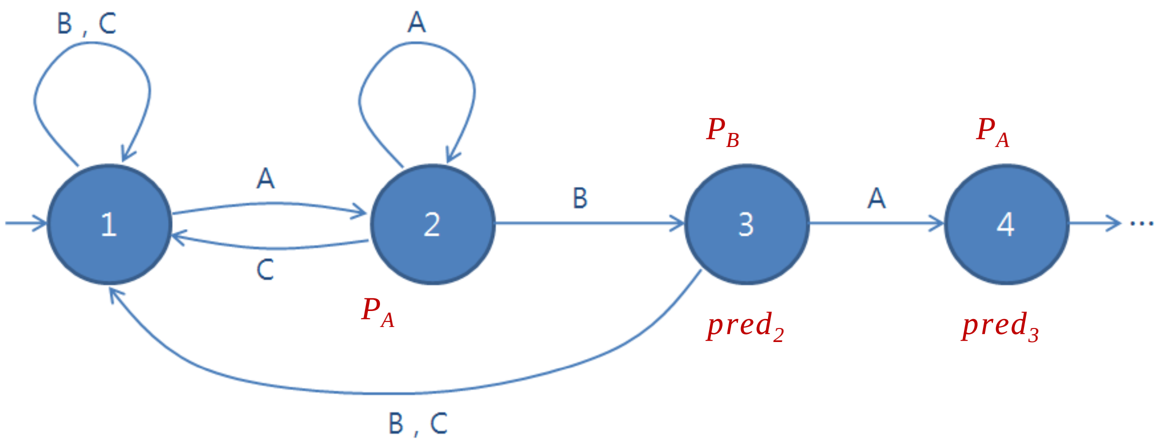
- Properties
  - Elementary property
    - (atomic) Proposition
    - Associated with each state
    - True or False in a given state
  - Complicated property
    - Expressed using elementary properties
    - Depends on the logic we use



- For example,
  - P_A : an A has just been keyed in
  - P_B : an B has just been keyed in
  - P_C : an C has just been keyed in
  - $pred_2$: the proceeding state in an execution is 2
  - $pred_3$: the proceeding state in an execution is 3

  - Properties of the system to verify
    1. If the door opens, then A, B, A were the last three letters keyed in, in that order.
    2. Keying in any sequence of letters ending in ABA opens the door.

  - Let's prove the properties with the propositions



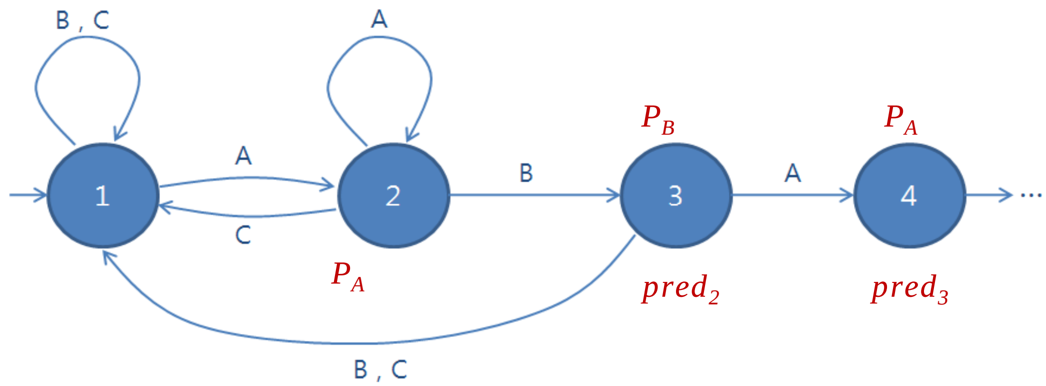
1.2 A Few Definition

- An automaton is a tuple $A = \langle Q, E, T, q_0, l \rangle$ in which
  - Q : a finite set of states
  - E : the finite set of transition labels
  - $T \subseteq Q \times E \times Q$: the set of transitions
  - q_0 : the initial state of the automaton
  - l : the mapping each state with associated sets of properties which hold in it

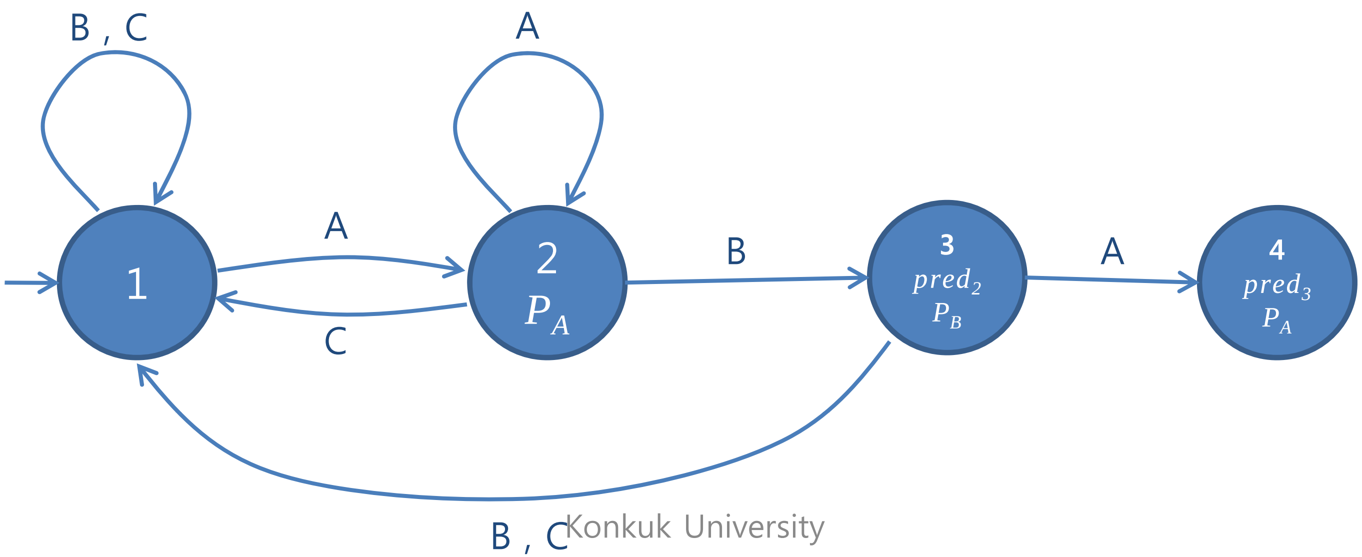
  - $Prop = \{P_1, P_2, \dots\}$: a set of elementary propositions

$A = < Q, E, T, q_0, l >$

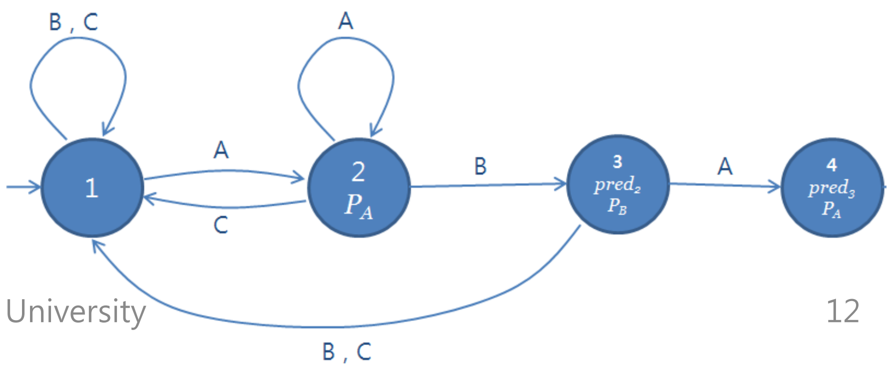
- Q = {1, 2, 3, 4}
- E = {A, B, C}
- T = { (1,A,2), (1,B,1), (1,C,1), (2,A,2), (2,B,3), (2,C,1), (3,A,4), (3,B,1), (3,C,1) }
- q_0 = 1
- $l = \begin{cases} 1 \rightarrow \varnothing \\ 2 \rightarrow \{P_A\} \\ 3 \rightarrow \{P_B, pred_2\} \\ 4 \rightarrow \{P_A, pred_3\} \end{cases}$



The digicode with its atomic propositions

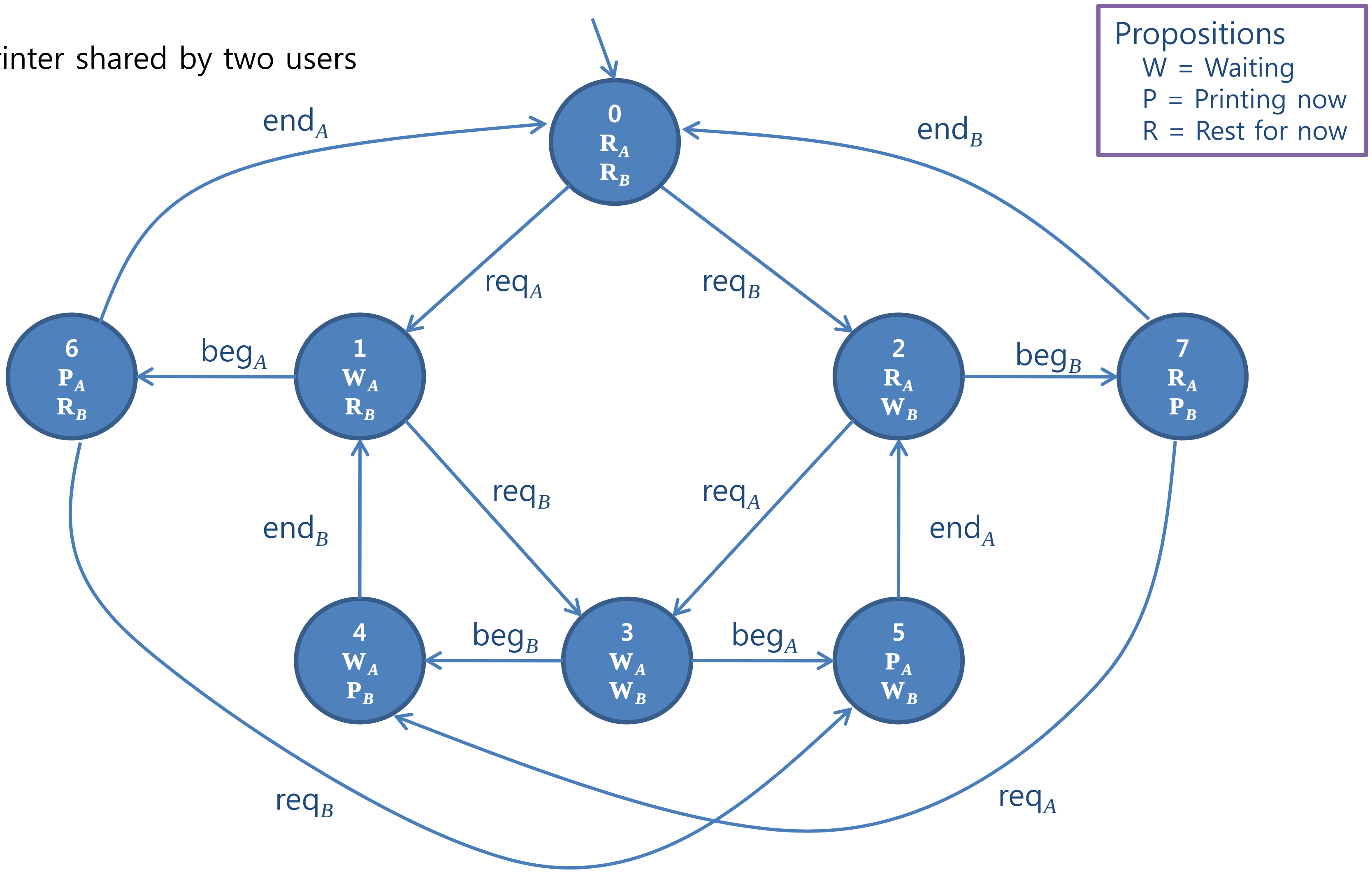


- Formal definitions of automaton's behavior
  - a *path* of automaton A :
    - A sequence σ, finite or infinite, of transitions which follows each other
    - Ex. $3 \xrightarrow{B} 1 \xrightarrow{A} 2 \xrightarrow{A} 2$
  - a *length* of a path σ :
    - $|\sigma|$
    - σ's potentially infinite number of transitions: $|\sigma| \in N \cup \{\omega\}$
  - a *partial execution* of A :
    - A path starting from the initial state q_0
    - Ex. $1 \xrightarrow{A} 2 \xrightarrow{A} 2 \xrightarrow{B} 3$
  - a *complete execution* of A :
    - An execution which is maximal.
    - Infinite or deadlock
  - a *reachable state* :
    - A state is said to be reachable,
      - if a state appears in the execution tree of the automaton, in other words,
      - if there exists at least one execution in which it appears.



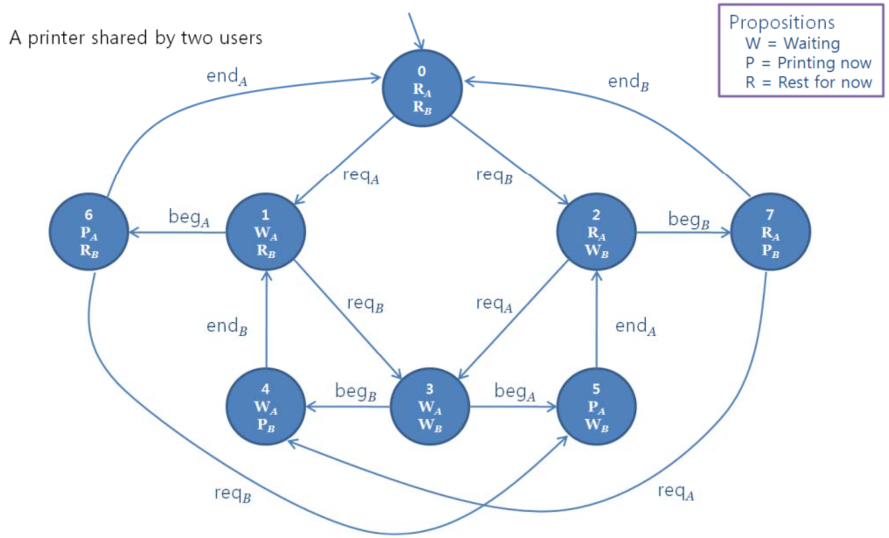
1.3 Printer Manager

A printer shared by two users



$A = < Q, E, T, q_0, l >$

- Q = {0, 1, 2, 3, 4, 5, 6, 7}
- E = {req_A, req_B, beg_A, beg_B, end_A, end_B}
- T = { (0,req_A,1), (0,req_B,2), (1,req_B,3), (1,beg_A,6), (2,req_A,3), (2,beg_B,7), (3,beg_A,5), (3,beg_B,4), (4,end_B,1), (5,end_A,2), (6,end_A,0), (6,req_B,5), (7,end_B,0), (7,req_A,4) }
- $q_0 = 0$
- $l = \begin{cases} 0 \rightarrow \{R_A, R_B\}, & 1 \rightarrow \{W_A, R_B\} \\ 2 \rightarrow \{R_A, W_B\}, & 3 \rightarrow \{W_A, W_B\} \\ 4 \rightarrow \{W_A, P_B\}, & 5 \rightarrow \{P_A, W_B\} \\ 6 \rightarrow \{P_A, R_B\}, & 7 \rightarrow \{R_A, P_B\} \end{cases}$



- Properties of the printer manager to verify

1. We would undoubtedly wish to prove that any printing operation is preceded by a print request.
    - In any execution, any state in which P_A holds is preceded by a state in which the proposition W_A holds.

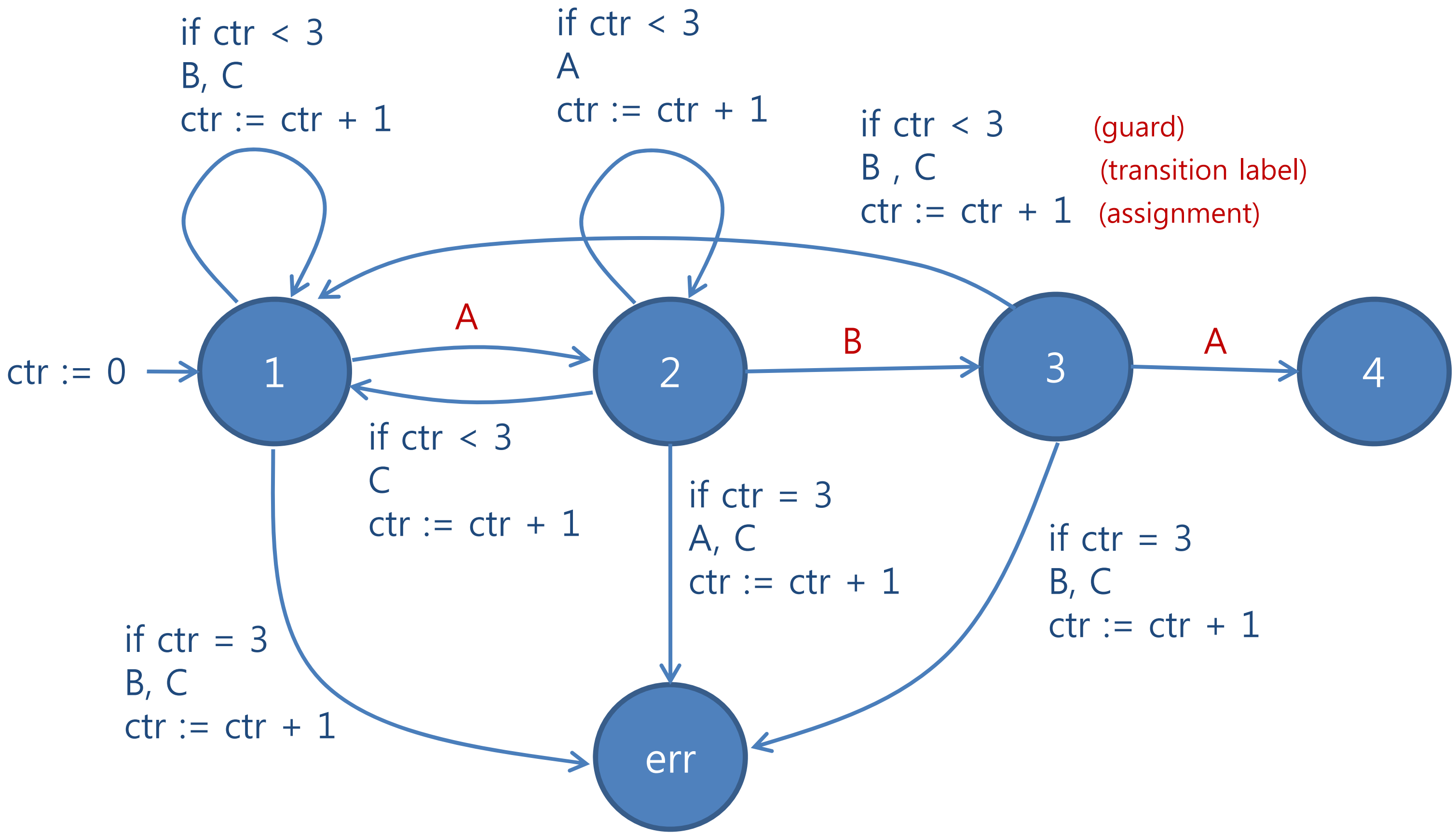
2. Similarly, we would like to check that any print request is ultimately satisfied. (→ fairness property)
    - In any execution, any state in which W_A holds is followed by a state in which the proposition P_A holds.

- Model checking techniques allow us to prove automatically that
    - Property 1 is TRUE, and
    - Property 2 is FALSE, for example 0 1 3 4 1 3 4 1 3 4 1 3 4 1 … (counterexample)

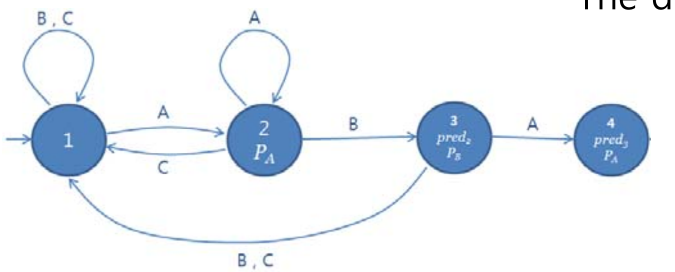
1.4 Few More Variables

- It is often convenient to let automata manipulate ***state variables***.
  - Control : states + transitions
  - Data : variables (assumes finite number of values)

- An automaton interacts with variables in two ways:
  - Assignments
  - Guards



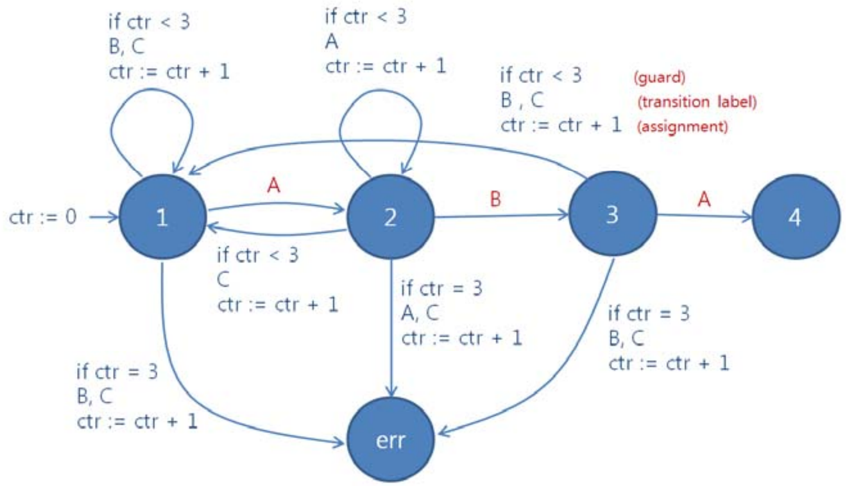
The digicode with guarded transitions



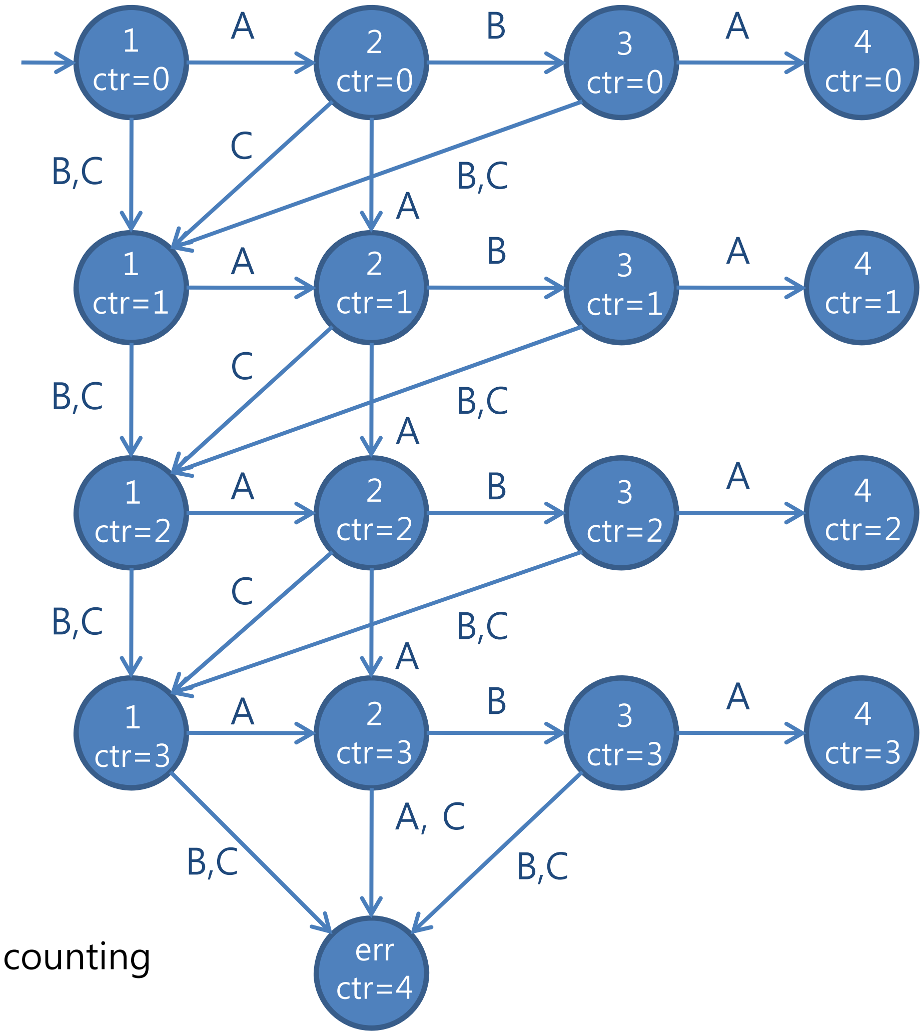
No more than 3 mistakes !!!

- It is often necessary, in order to apply model checking methods,
  - to *unfold* the behaviors of an automaton with variables
  - into a state graph
  - in which the possible transitions appear and the configurations are clear marked.

- Unfolded automaton = Transition system
  - has global states
  - transitions are no longer guarded
  - no assignments on the transitions



Unfolding



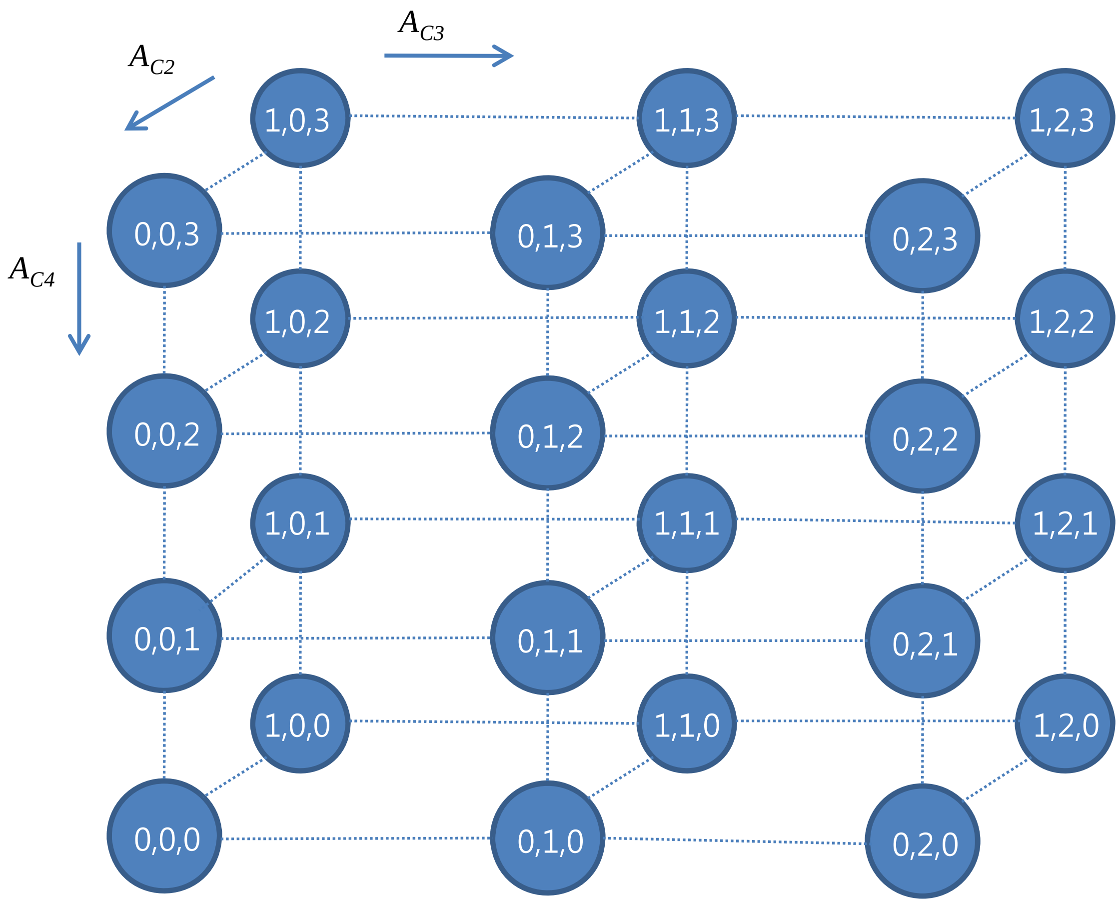
The digicode with error counting
(Unfolded automaton)

1.5 Synchronized Product

- Real-life programs or systems are often composed of modules or subsystems.
  - Modules/Components → (composition) → Overall system
  - Component automata → (synchronization) → Global automaton

- Automata for an overall system
  - Often has so many global states
  - Impossible to construct it directly (State explosion problem)
  - Two composition ways
    - With synchronization
    - Without synchronization

- An example <u>without synchronization</u>
  - A system made up of three counters (modulo 2, 3, 4)
  - They do not interact with each other
  - Global automaton = Cartesian product of three independent automata



2*3*4 = 24 states
3*3*3 - 1 = 26 transitions per a state
(Inc, Dec, -)

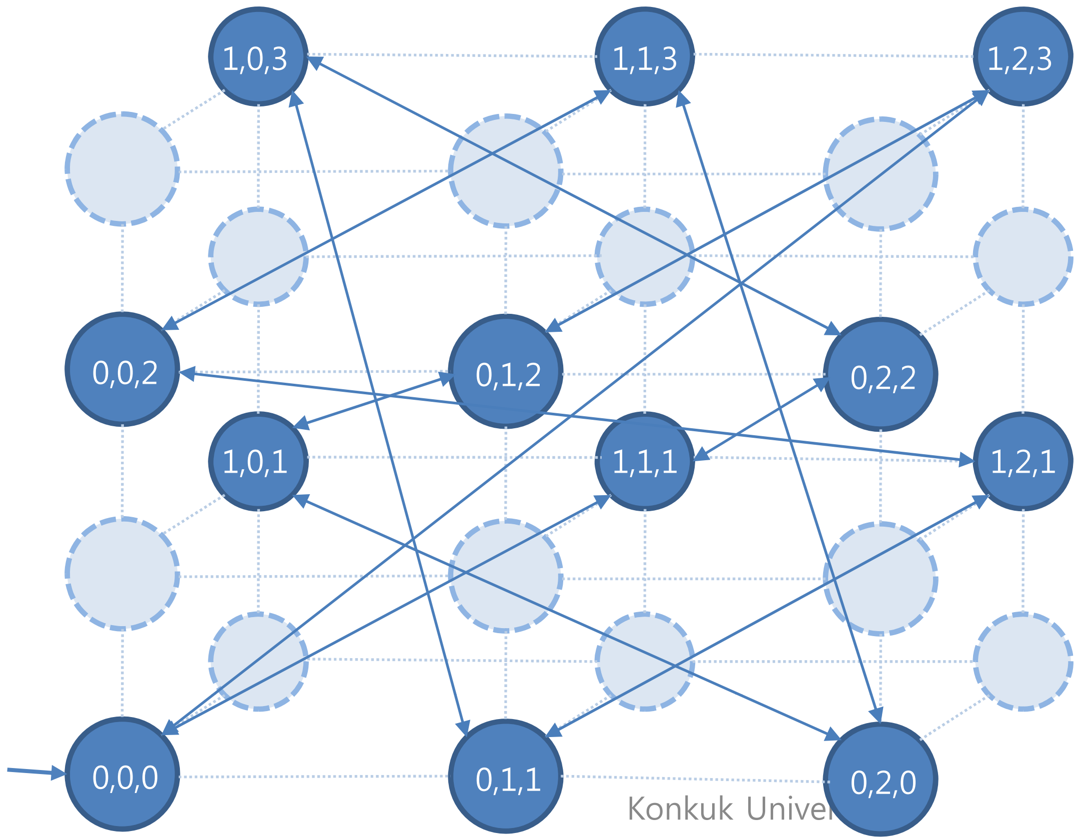
→ 24 * 26 = 624 transitions

- An example <u>with synchronization</u>
  - A number of ways depending on the nature of the problem
  - Ex. Allowing only “inc, inc, inc” and “dec, dec, dec” (24*2=48 transitions)
  - Ex. Allowing updates in only one counter at a time (24*3*2=144 transitions)

- Synchronized product
  - A way to formally express synchronizing options
  - Synchronized product = Component automata + Synchronized set
  - $A_1 \times A_2 \times \dots \times A_n$: Component automata
  - $A = < Q, E, T, q_0, l >$
    - $Q = Q_1 \times Q_2 \times \dots \times Q_n$
    - $E = \prod_{1 \le i \le n} (E_i \cup \{-\})$
    - $T = \left\{ \begin{array}{l} ((q_1, \dots, q_n), (e_1, \dots, e_n), (q'_1, \dots, q'_n)) \mid \text{for all } i, \\ (e_i = \text{‘-’ and } q'_i = q_i) \text{ or } (e_i \neq \text{‘-’ and } (q_i, e_i, q'_i) \in T_i) \end{array} \right\}$
    - $q_0 = (q_{0,1}, \dots, q_{0,n})$
    - $l((q_1, \dots, q_n)) = \bigcup_{1 \le i \le n} l_i(q_i)$
  - $Sync \subseteq \prod_{1 \le i \le n} (E_i \cup \{-\})$: Synchronized set

- An example with synchronization
  - Ex. Allowing only “inc, inc, inc” and “dec, dec, dec” (24*2=48 transitions) → Strongly coupled version of modular counters
  - $Sync$ = { (inc, inc, inc), (dec, dec, dec) }
  - $T = \left\{ \begin{array}{l} ((q_1, \ldots, q_n), (e_1, \ldots, e_n), (q'_1, \ldots, q'_n)) \mid (e_1, \ldots, e_n) \in Sync \\ (e_i = \text{‘-’ and } q'_i = q_i) \text{ or } (e_i \neq \text{‘-’ and } (q_i, e_i, q'_i) \in T_i) \end{array} \right\}$

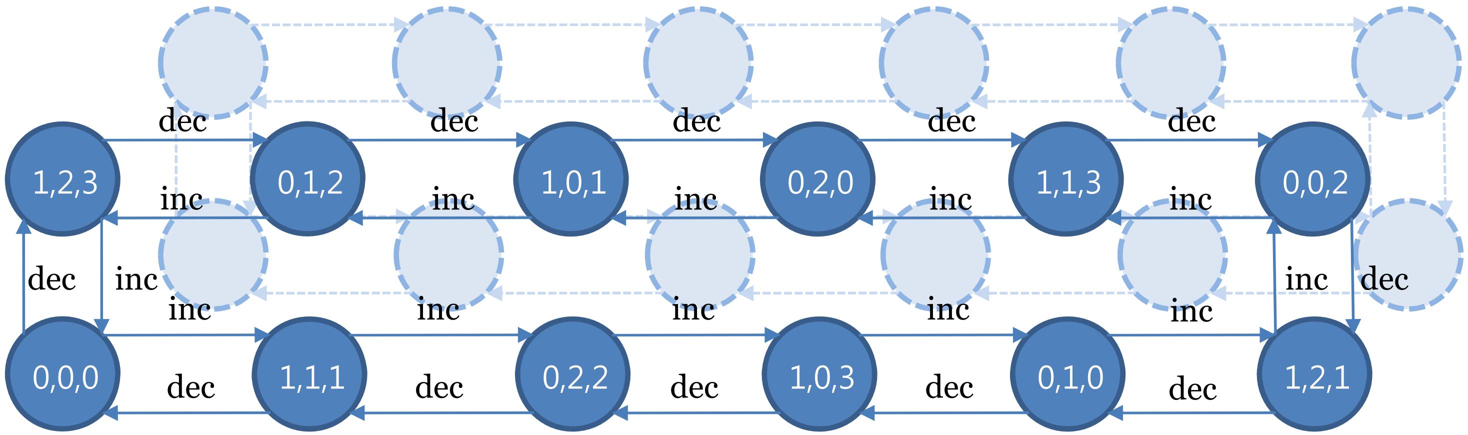
A_{ccc}^{coupl}



12 states

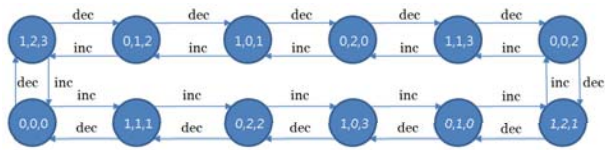
12 transitions
(inc, inc, inc) (dec, dec, dec)

- Reachable states
  - Reachability depends on the synchronization constraints



Rearranged automaton A_{ccc}^{coupl} $\rightarrow$ modulo 12 counter

- Reachability graph
  - Obtained by deleting non-reachable states
  - Many tools to construct R.G. of synchronized product of automata
  - Reachability is a difficult problem
    - <u>State explosion problem</u>



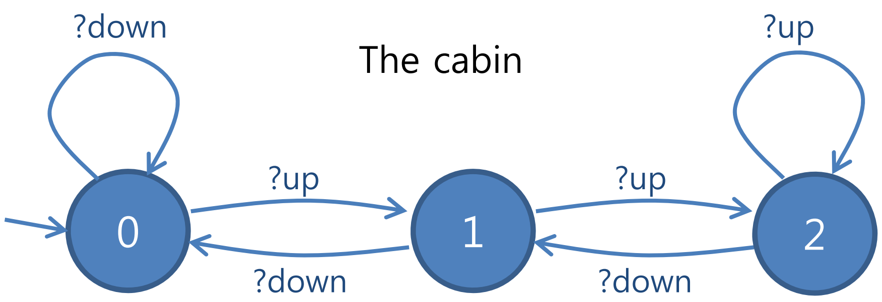
1.6 Synchronization with Message Passing

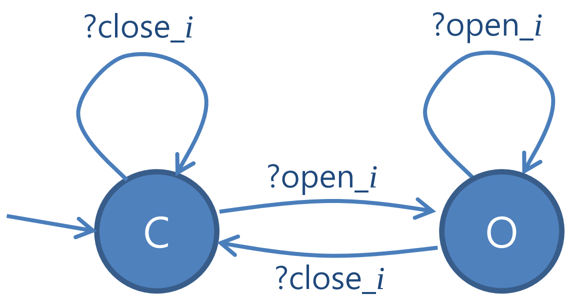
- Message passing framework
  - A special case of synchronized product
  - $!m$: Emitting a message
  - $?m$: Reception of the message

  - Only the transition in which !m and ?m pairs are executed simultaneously is permitted.

  - Synchronous communication
    - Control/command system
  - Asynchronous communication
    - Communication protocol (using channel/buffer)

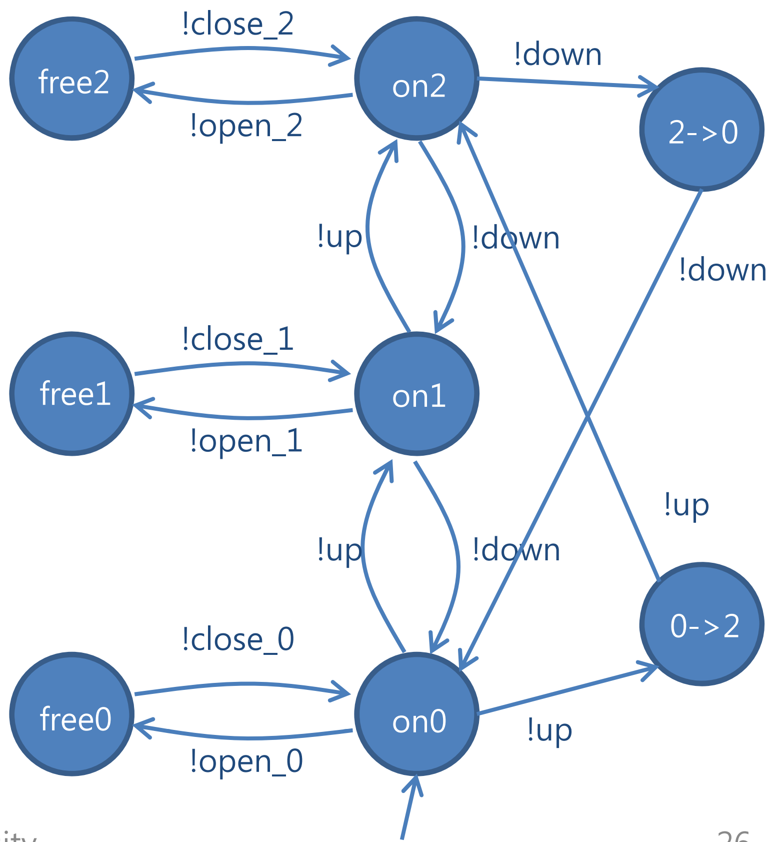
- Smallish elevator
  - Synchronous communication (message passing)
  - One cabin
  - Three doors (one per floor)
  - One controller
  - No requests from the three floors





The i^{th} door

The controller



- An automaton for the smallish elevator example
  – Obtained as the synchronized product of the five automata

  – (door 0, door 1, door 2, cabin, controller)
  – $Sync$ = { (?open_1, -, -, -, !open_1), (?close_1, -, -, -, !close_1),
    (-, ?open_2, -, -, !open_2), (-, ?close_2, -, -, !close_2),
    (-, -, ?open_3, -, !open_3), (-, -, ?close_3, -, !clsoe_3),
    (-, -, -, ?down, !down), (-, -, -, ?up, !up) }

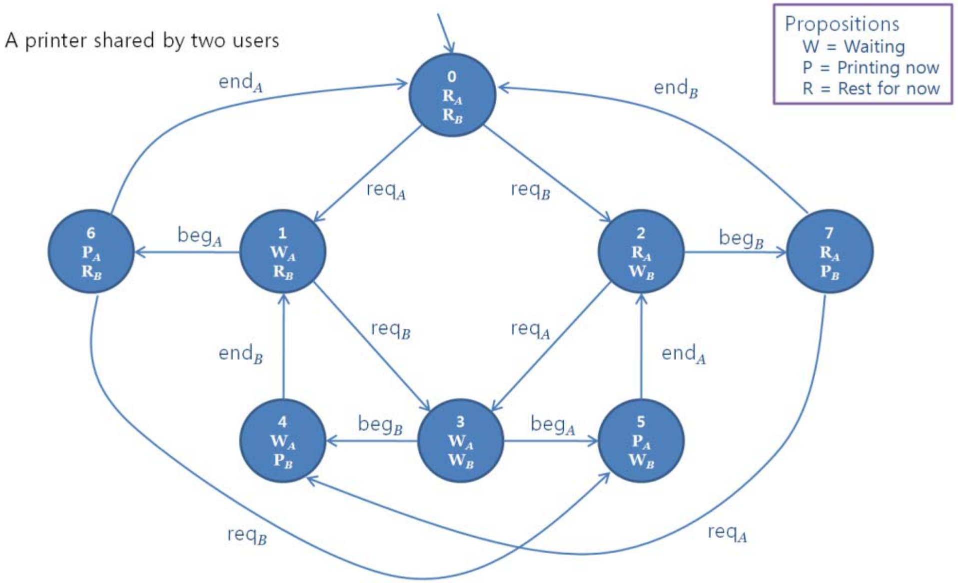
- Properties to check
  - (P1) The door on a given floor cannot open while the cabin is on a different floor.
  - (P2) The cabin cannot move while one of the door is open.

- Model checker
  - Can build the synchronized product of the 5 automata.
  - Can check automatically whether properties hold or not.

1.7 Synchronization by Shared Variables

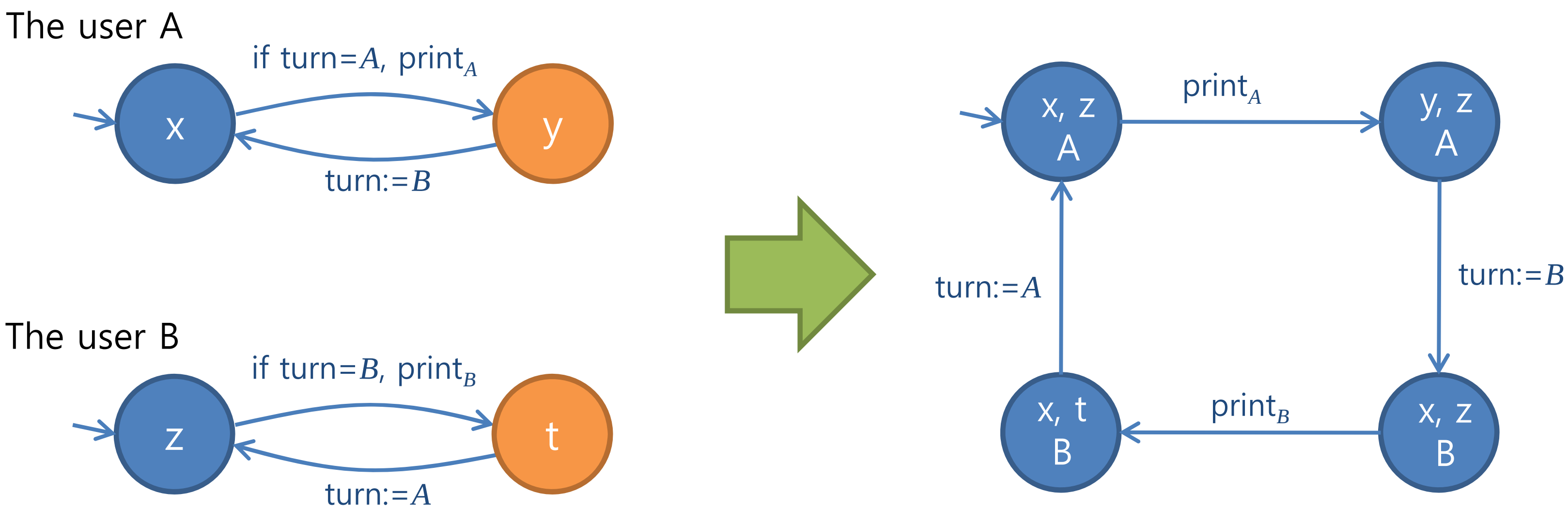
- Another way to have components communicate with each other
- Share a certain number of variables
- Allow variables to be shared by several automata

- Ex. The printer manager in Chapter 1.3
  - Problem: fairness property is not satisfied

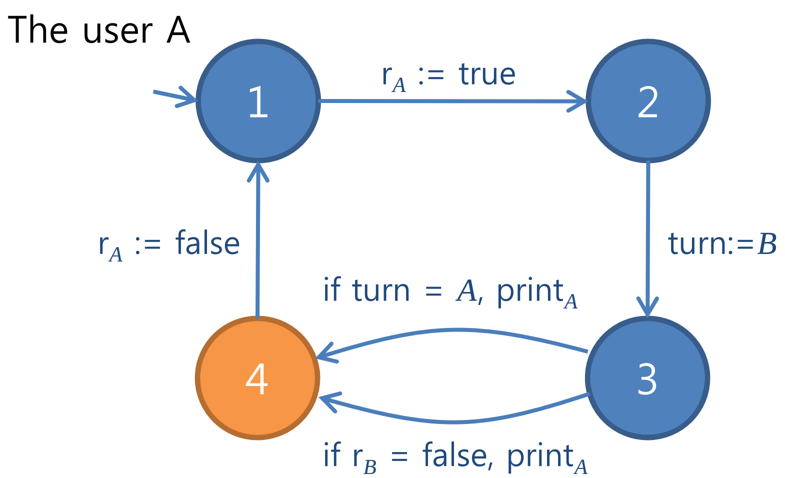


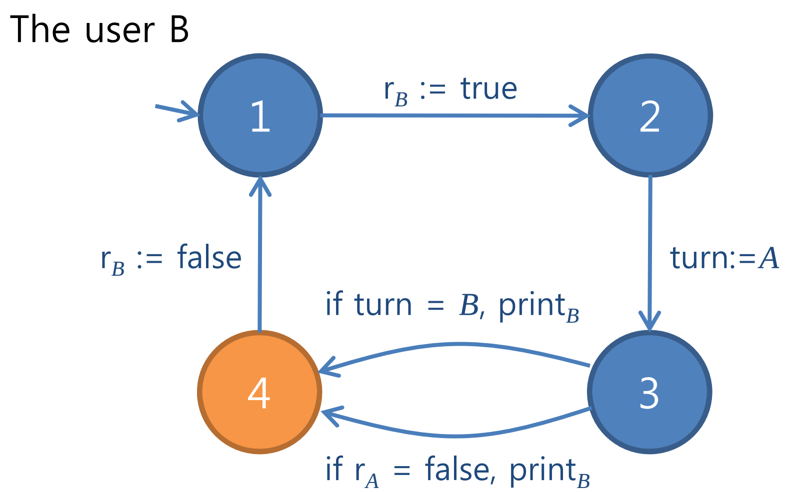
- The printer manager synchronized with a shared variable
  - Shared variable: turn

- Fairness property: Any print request is ultimately satisfied.
  → No state of the form (y, t, -) is reachable.
  → TRUE in the model.
  → But, this model forbids either user from printing twice in a row.



- Printer manager : A complete version with 3 variables [by Peterson]
  - r_A : a request from user A
  - r_B : a request from user B
  - turn : to settle conflicts
  - Satisfies all our properties





$A_{A \times B} = < Q, E, T, q_0, l >$

- $Q = A \times B \times r_A \times r_B \times turn$
  4 x 4 x 2 x 2 x 2 = 128 states
  (only 128 reachable states)