# CS342: Software Engineering

Dr. Ismail Hababeh
German Jordanian University

Lecture 12
ANALYSIS WORKFLOW

*Adapted from Software Engineering, by Dr. Paul E. Young*
*& slides by Dr. Mohammad Daoud*

# Finite State Machines State Transitions

**Current State + Event + Predicate $\Rightarrow$ Next State**

- A State is a description of the status of a system waiting to execute a transition.
- A Transition is a set of actions to be executed when a condition is satisfied, or an event received.
- An Event is an occurrence within a particular system or domain; it is something that has happened or is expected to happen in that domain.
- A Predicate is a function or a statement that returns a Boolean value depending on the data passed to the predicate.

# The Power of Finite State Machines (FSM)

- Advantages of using an FSM, a specification is
  - Easy to write it down
  - Easy to validate
  - Easy to convert into a design
  - Easy to convert into code
  - Easy to understand
  - Easy to maintain
  - More precise than graphical methods
- Disadvantages of FSM
  - Timing considerations are not handled

# Difficulty with Specifying Real-Time Systems

- A major difficulty with specifying real-time systems is timing
  - Synchronization problems
  - Race conditions (*perform two or more operations at the same time*)
  - Deadlock
- Often a consequence of poor specifications
- Solution? ⇒ Petri Nets.

# Petri Nets

- A **Petri net** (also known as a place/transition **net** or P/T **net**) is one of several mathematical modeling languages for the description of distributed systems that have timing problems.
- A powerful technique for specifying systems that have potential timing problems.

# Petri Nets Specifications

- A Petri net consists of four parts:
- A set of places P
- A set of transitions T
- An input function I(t)
  - Mapping from transitions to a set of places
- An output function O(t)
  - Mapping from transitions to a set of places

# Petri Nets Mathematical Model

A Petri net is a 4-tuple $C = (P, T, I, O)$

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of *places*, $n \geq 0$
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of *transitions*, $m \geq 0$, with disjoint P and T
- $I : T \rightarrow P^\infty$ is the *input* function, a mapping from set of transitions to a set of places.

# Petri Nets Mathematical Model

- $O : T \rightarrow P^{\infty}$ is the *output* function, a mapping from transitions to a set of places
- A *marking* of a Petri net is an assignment of tokens to that Petri net.
- Any distribution of tokens over the places will represent a configuration of the net called a *marking*.

# Petri Nets Architecture – Example 1

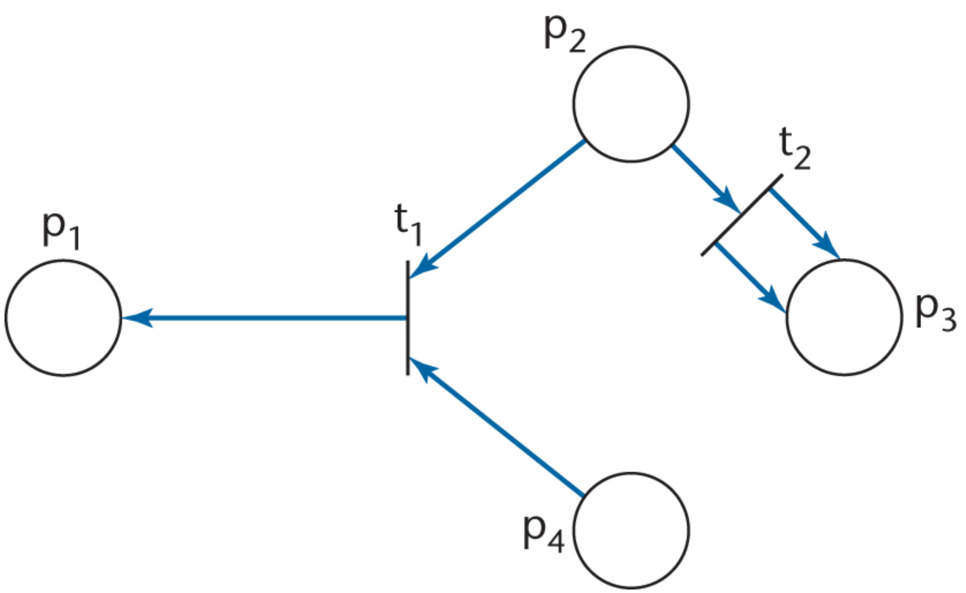
- Set of places P is $\{p_1, p_2, p_3, p_4\}$
- Set of transitions T is $\{t_1, t_2\}$
- Input functions:

  $I(t_1) = \{p_2, p_4\}$

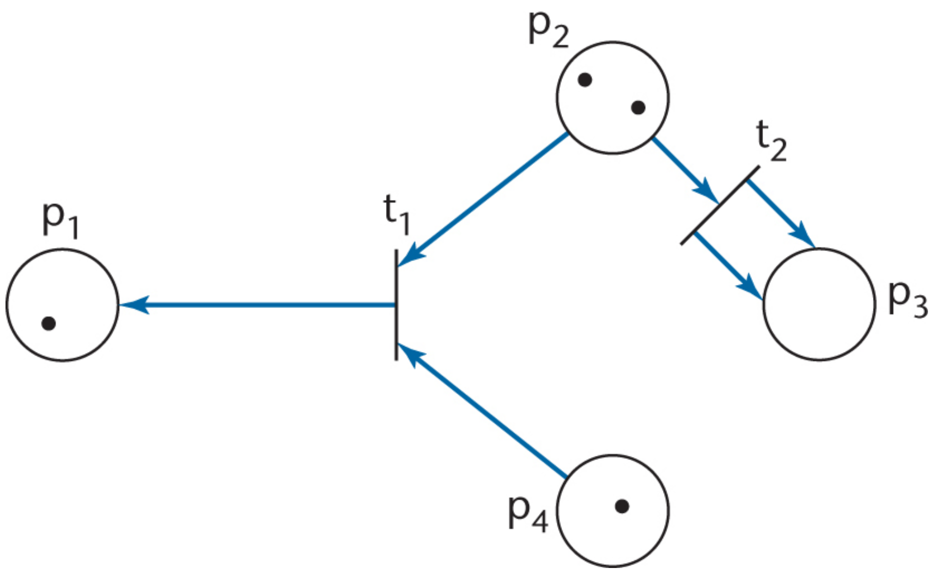
  $I(t_2) = \{p_2\}$
- Output functions:

  $O(t_1) = \{p_1\}$

  $O(t_2) = \{p_3, p_3\}$



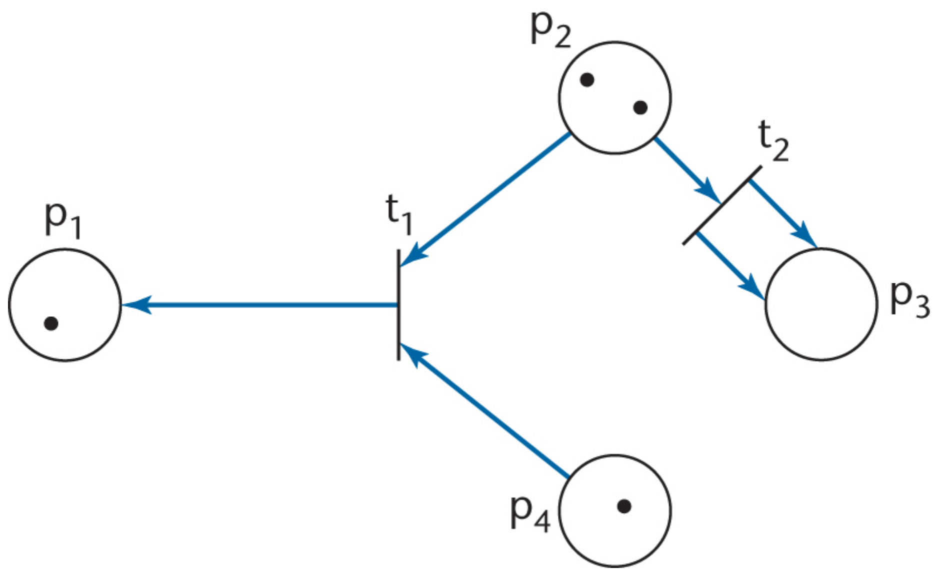
# Petri Nets Marking – Example 2




- Four tokens: one in $p_1$, two in $p_2$, none in $p_3$, and one in $p_4$ are represented by the vector (1, 2, 0, 1)

# Enabled Transitions

- A transition is enabled (ready to fire) if each of its input places has number of tokens greater than or equal to the number of arcs from the input place to that transition. In the following figure, both t1 and t2 are enabled.

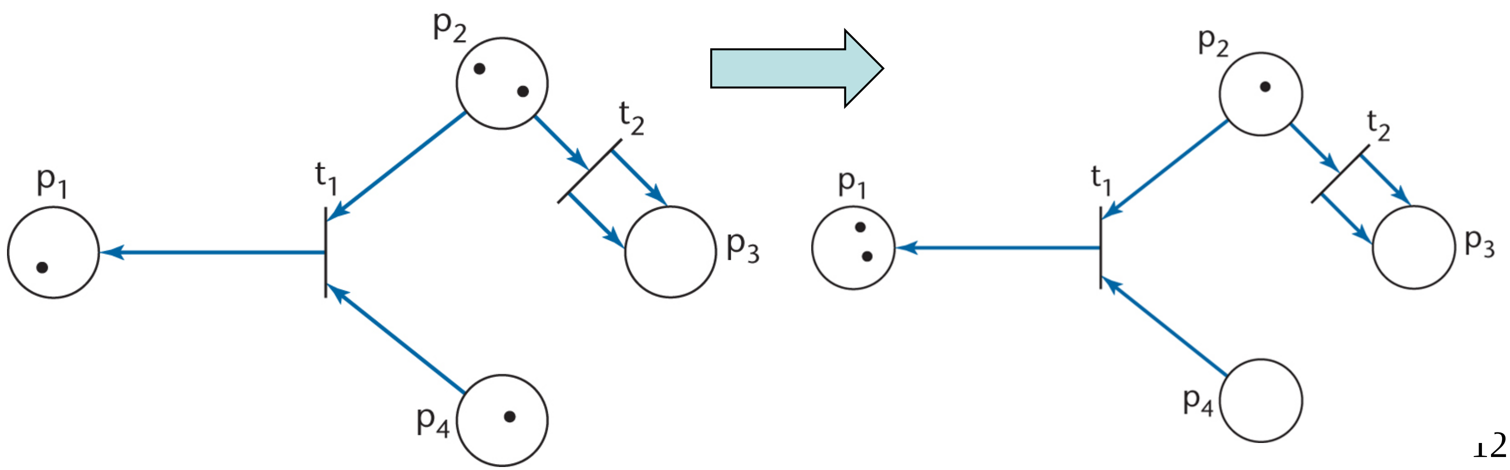


# Petri Nets Tokens

- Transition $t_1$ is enabled (ready to fire)
  - If $t_1$ fires, one token is removed from $p_2$ and one from $p_4$, and one new token is placed in $p_1$

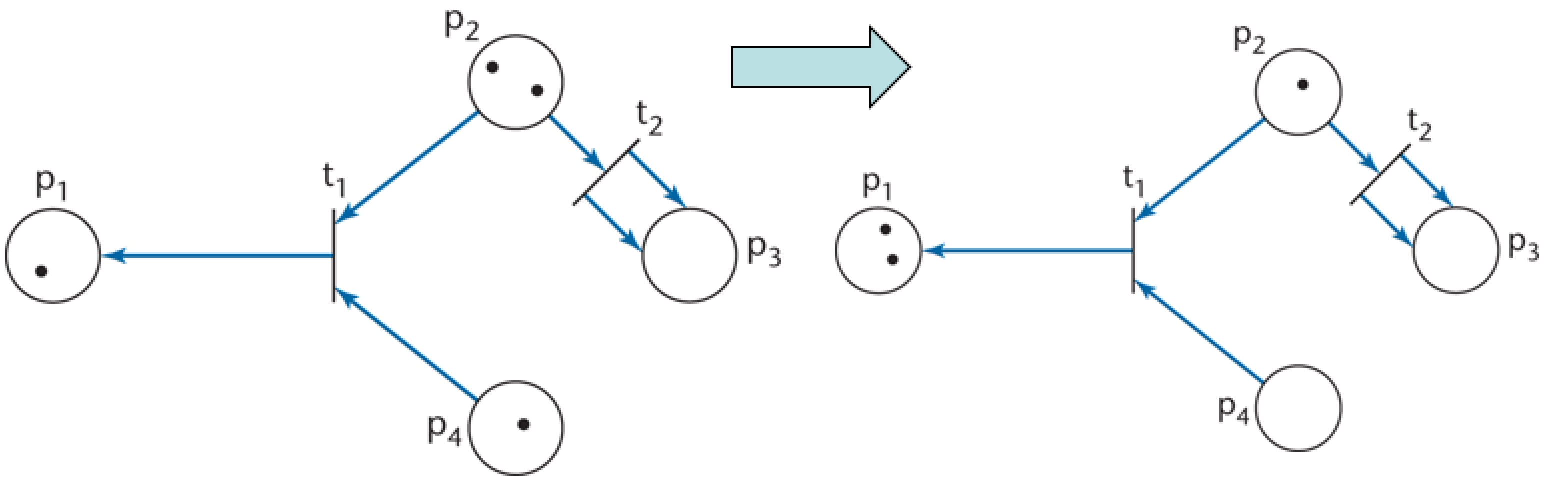
⇨ Number of tokens is not conserved

- Transition $t_2$ is also enabled



# Petri Nets Indeterminate

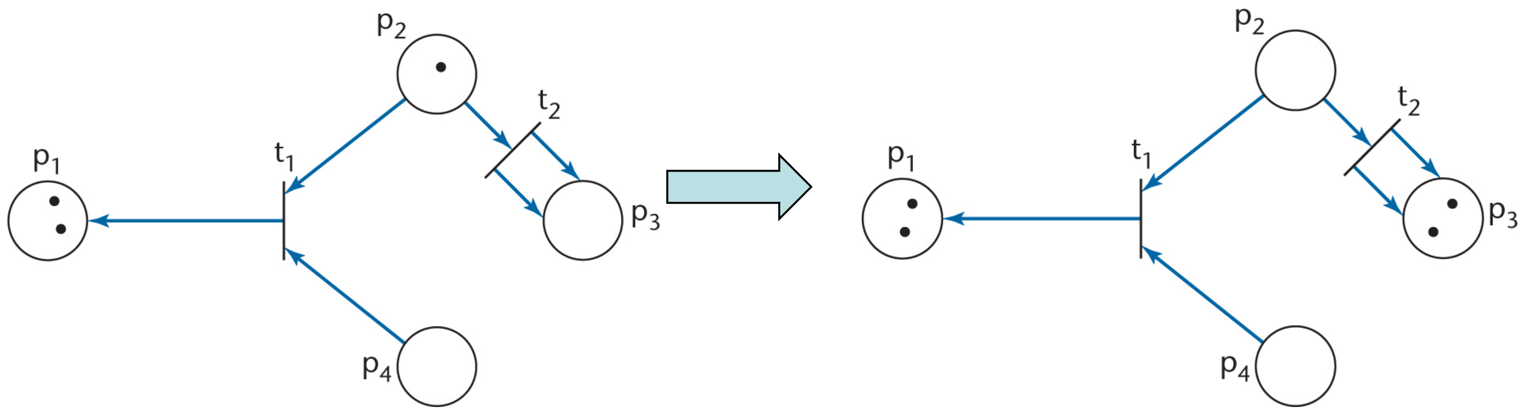
- Petri nets are indeterminate
  - Suppose $t_1$ is fired




- The resulting marking is (2, 1, 0 ,0)

# Petri Nets Transitions

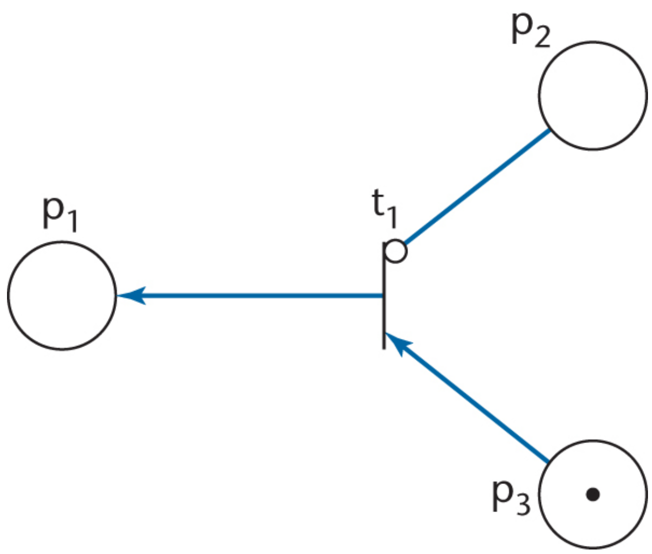
- Now, only $t_2$ is enabled.




- The resulting marking is (2, 0, 2, 0)

# Petri Nets Inhibitor Arc

- Inhibitor Arc: imposes the precondition that the transition may only fire when the place is empty.
- An inhibitor arc is marked by a small circle, not an arrowhead.
- In the following Petri Net, transition $t_1$ is enabled



# Updated Enabled Transitions

- In general, a transition is enabled if there is at least one token on each (normal) input arc places, and no tokens on any  inhibitor input arcs places.

# Petri Nets: Classical Real Time Systems Analysis - Elevator Case Study

- **First Constraint on Elevator Buttons:**
  - Each elevator has a set of m buttons, one for each floor.
  - The button light on when pressed and cause the elevator to visit the corresponding floor.
  - The button light off when the corresponding floor is visited by an elevator.

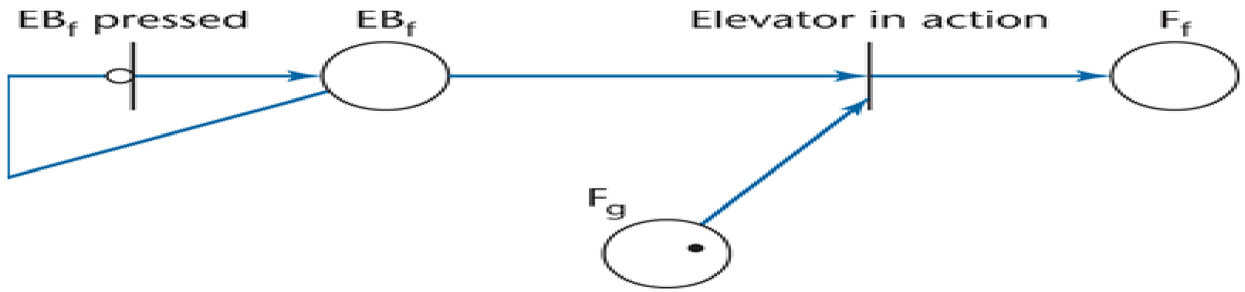
# Petri Nets - Elevator Button Analysis

- The elevator button for floor f is represented by place

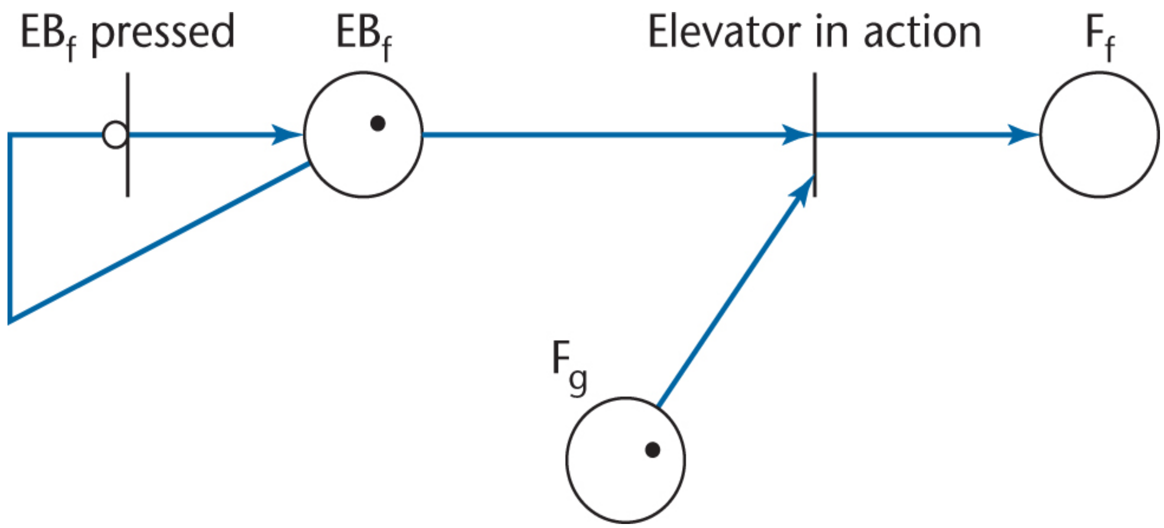
$$EB_f,\ 1 \le f \le m$$

- A token in $EB_f$ denotes that the elevator button for floor f is light on.
- A button must be light on the first time the button is pressed, and subsequent button presses must be ignored.

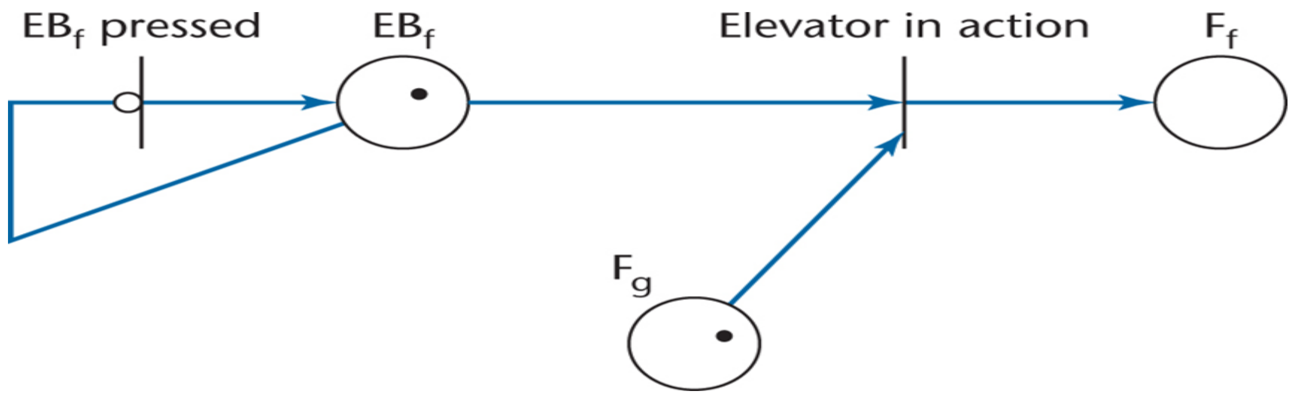
# Petri Nets - Elevator Button Analysis



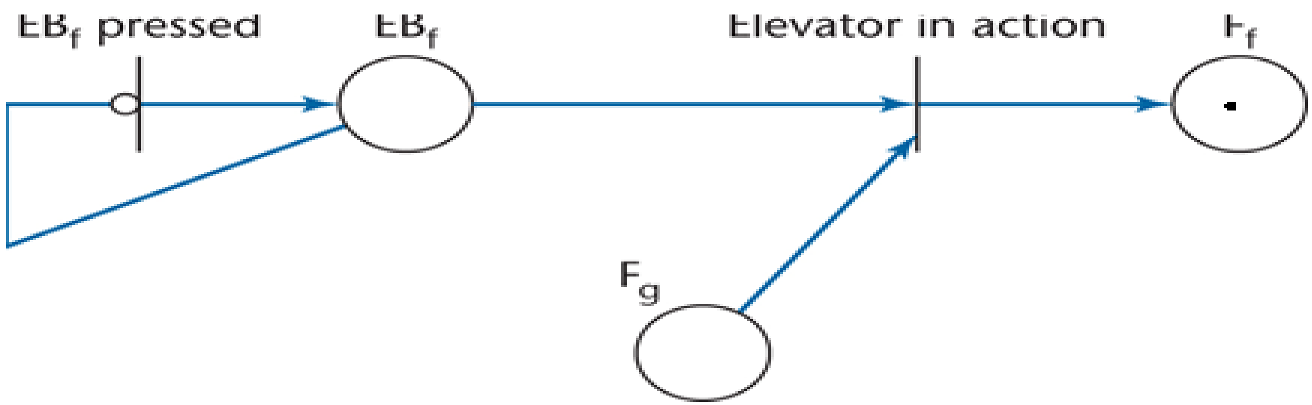

- Initial state, button $EB_f$ is not pressed, not lightened, no token in place $EB_f$ and transition $EB_f$ is not enabled
- Next state, $EB_f$ is pressed, new token is placed in $EB_f$



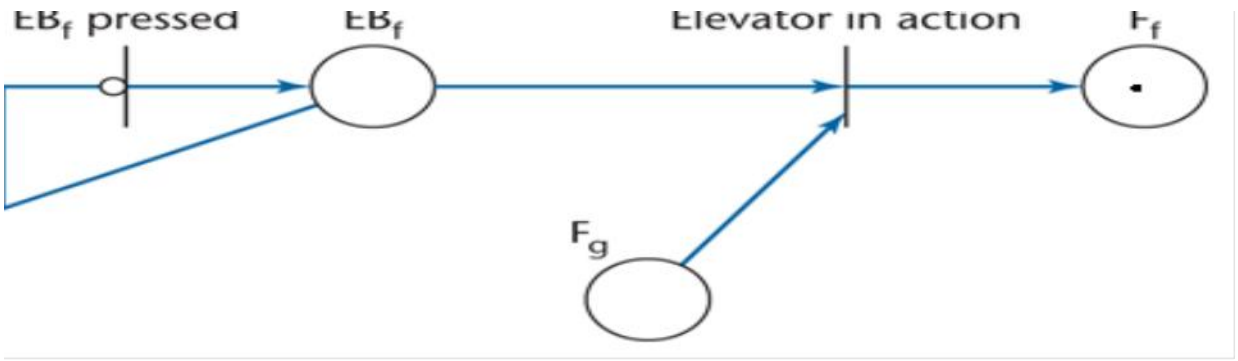
# Petri Nets - Elevator Button Analysis




- Next state, when the elevator reaches floor g
  - The transition Elevator in action is enabled, and then fired
  - The tokens in $EB_f$ and $F_g$ are removed, and then the light in button $EB_f$ is turned off.



# Petri Nets - Elevator Button Analysis



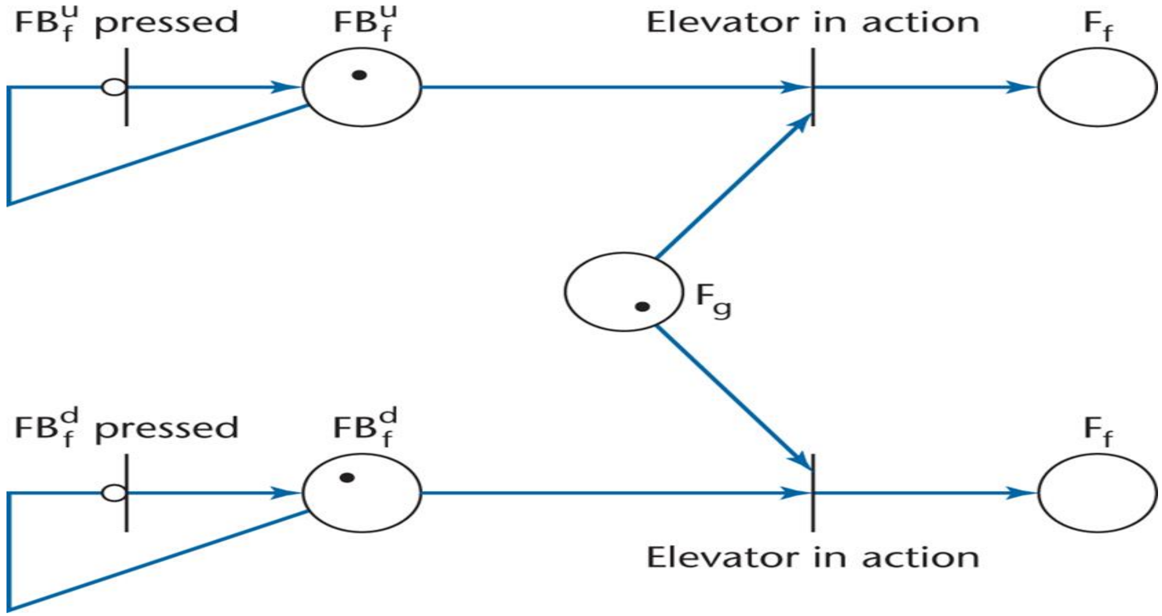

- A new token appears in $F_f$
  - This brings the elevator from floor g to floor f
- Motion from floor g to floor f cannot take place instantaneously
  - We need timed Petri nets

# Petri Nets - Floor Button Analysis

- **Second constraint on Floor Buttons:**
- Each floor, except the first and the top floor, has two buttons, one to request elevator up, and the other to request elevator down.
- These buttons are lightened on upon pressed
- The light is off when the elevator visits the floor, and then moves in desired direction
- Floor buttons are represented by 2 places

$$FB^u_f \text{ and } FB^d_f$$

# Petri Nets – Floor Button Analysis




- The Petri net models the situation when an elevator reaches floor f from floor g with one or both buttons illuminated
- If both buttons are illuminated, only one is turned off (the button that represents the direction of the elevator motion).

# Petri Nets - Elevator Waiting State Analysis

- **Third constraint on the Waiting State:**

If an elevator has no requests

- it remains at its current floor with its doors closed
- no Elevator in action transition is enabled

# Comparison of Classical Analysis Techniques

1. Formal methods are Powerful, but difficult to learn and use. Example: Techniques such as Petri nets.
2. Informal methods have little power but are easy to learn and use; written in a natural language.
3. Semiformal methods are techniques between informal and formal.

   Example: Software Requirements Engineering Method (SREM)

   – Useful for specifying real-time systems

   – Powerful of finite state machine (FSM)

# Comparison of Classical Analysis Techniques

4. New Classical Analysis Techniques
   - Many are untested in practice
   - Involved risks:
     - Training costs
     - Adjustment from the classroom to an actual project
     - CASE tools may not work properly
   - However, possible gains *may* be huge

# Computer Aided Software Engineering (CASE) Tools for Classical Analysis

- A graphical tool is very useful
- Typical tools
  - Analyst/Designer
  - Software through Pictures
  - System Architect

# Comparison of Classical Analysis Techniques

- Which Analysis Technique Should Be Used?
- It depends on the
  - Project
  - Development team
  - Management team
  - Other factors?

# Testing During Classical Analysis

- Specification inspection
  - Aided by fault checklist

# Challenges of Classical Analysis

- A specification document must be
  - Informal enough for the client
  - Formal enough for the development team
- Analysis (“what”) should not cross the boundary into design (“how”).