Embedded Systems Design and Modeling

Petri Nets

And

Their Timed Version

Outline

- Background
- Basic concepts
- Formal definition
- General properties
- Timed Petri nets

Background

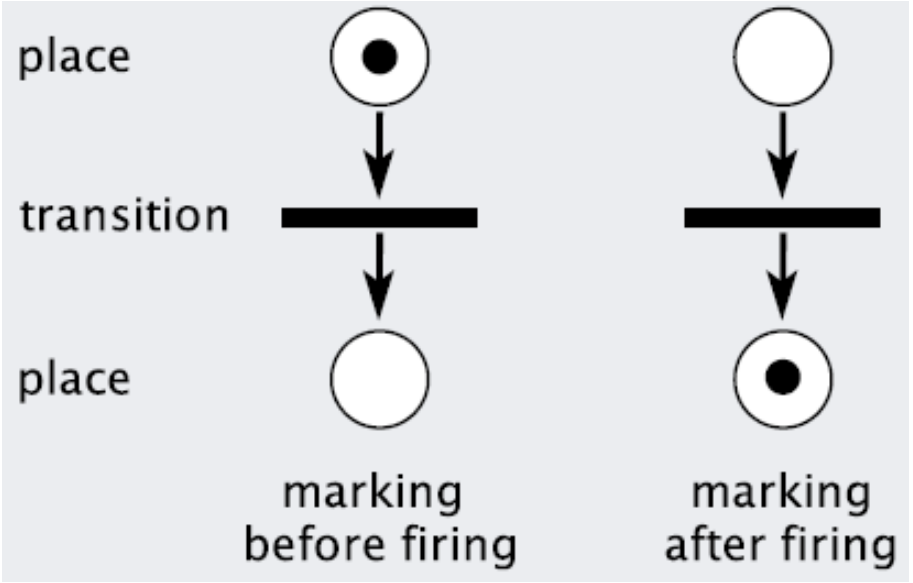
- Petri nets:
  - A formalism developed in the '60s by C. A. Petri to model concurrent systems
  - Popular formalism in many fields (computer science, control systems, production systems, ...)
  - Simple, intuitive, but at the same time powerful

Basic Concepts

- Two types of elements:
  - Places (depicted as white circles):
    - Can contain arbitrary number of tokens
  - Transitions (depicted as rectangles):
    - Are enabled if all its input places have at least one token
    - Once enabled, can fire consuming one token at each input, producing one at each output
- Marking: the condition of the overall network, similar to a state in FSM

Simple Example

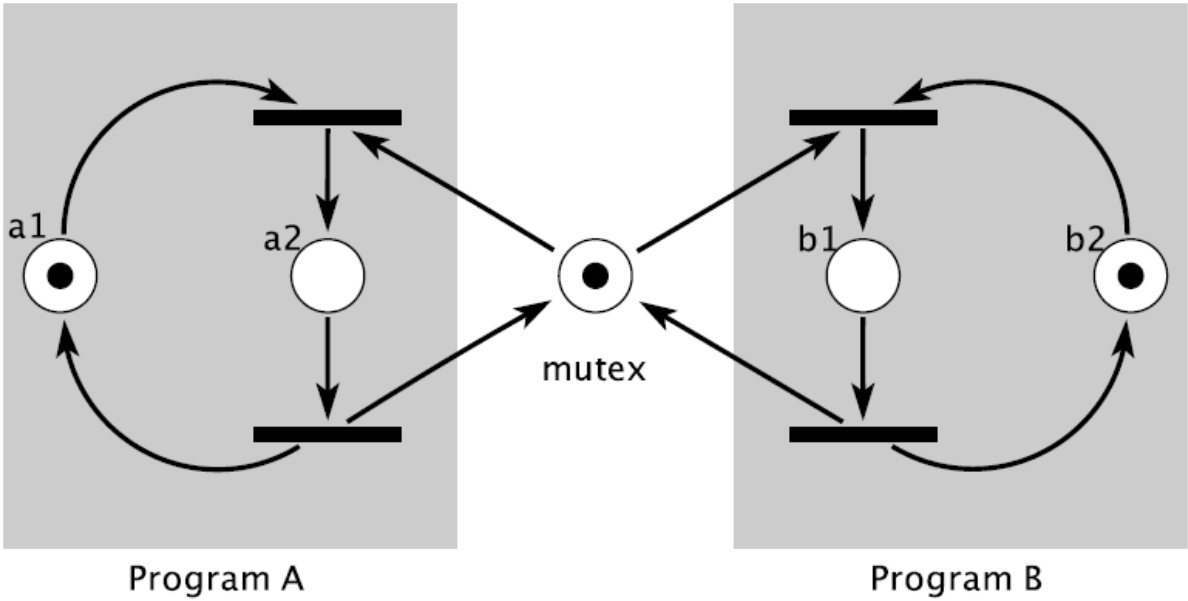
- In general, a transition can have any number of inputs and outputs
- Similarly, a place can go to more than one transition



- In that case, the net is nondeterministic:
  - A token in that place may trigger either of the transitions

Mutual Exclusion (mutex) Example

- Example: when 2 programs can't be in one critical section at the same time



- It can be shown that a Petri net with a finite number of markings can be represented with an FSM and vice versa

Formal Definition

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:

$P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
$W: F \to \{1, 2, 3, \cdots\}$ is a weight function,
$M_0: P \to \{0, 1, 2, 3, \cdots\}$ is the initial marking,
$P \cap T = \varnothing$ and $P \cup T \neq \varnothing$.

A Petri net structure $N = (P, T, F, W)$ without any specific initial marking is denoted by N.

A Petri net with the given initial marking is denoted by (N, M_0).

Petri Net Properties

1. Marking-independent or structural properties
2. Marking-dependent or behavioral:
    1. Reachability
    2. Boundedness
    3. Liveness
    4. Reversibility
    5. Coverability, persistence, ….

Reachability

- Consider a firing sequence of M0, M1, …, Mn
- Mn is said to be reachable from M0
- Reachability problem: determine if an arbitrary marking Mi can be reached from an arbitrary initial marking of M0

Boundedness

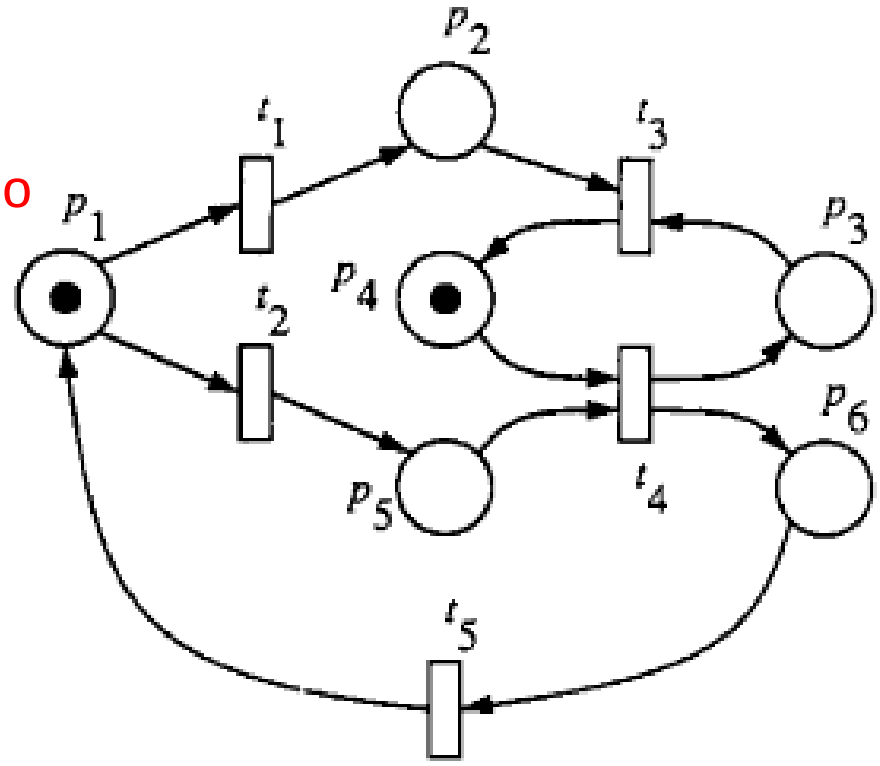
- A Petri net is said to be bounded if the number of tokens in each place doesn't exceed a finite number for any reachable marking
- If the finite number can be specifically found (say k), then it is called k-bounded
- Importance: a buffer or register with finite size can be used to implement a bounded Petri net

Liveness

- Liveness comes from lack of deadlocks
- A Petri net is said to be live if no matter what marking has been reached from an arbitrary marking, the firing can be continued
- Example in the next slide

Nonlive Example

- Transitions t1 and t2 may fire nondeterministically
- If t1 fires first, there will be a deadlock
  - Intuitive observation:
    - A system branches out into two processes nondeterministically
    - But the two processes depend on each other
    - Recipe for deadlock!

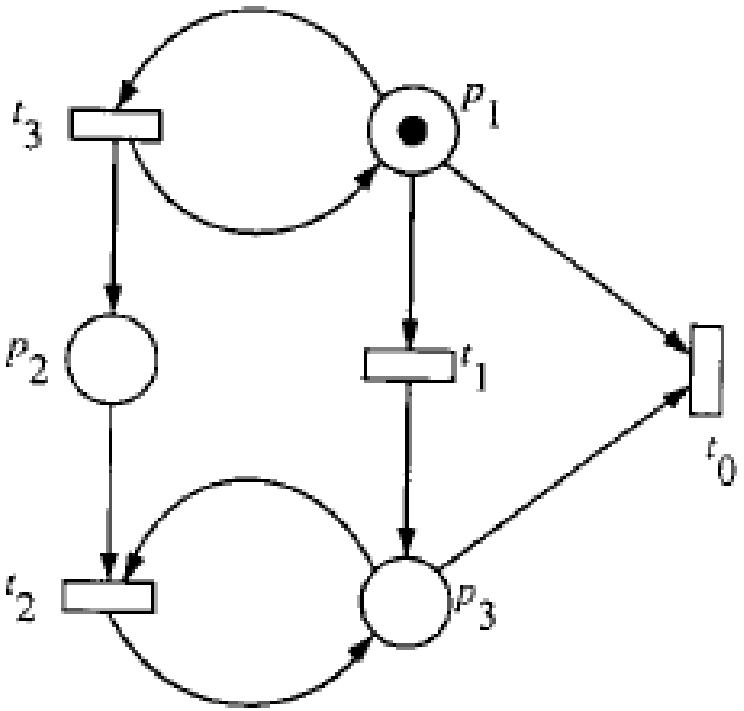


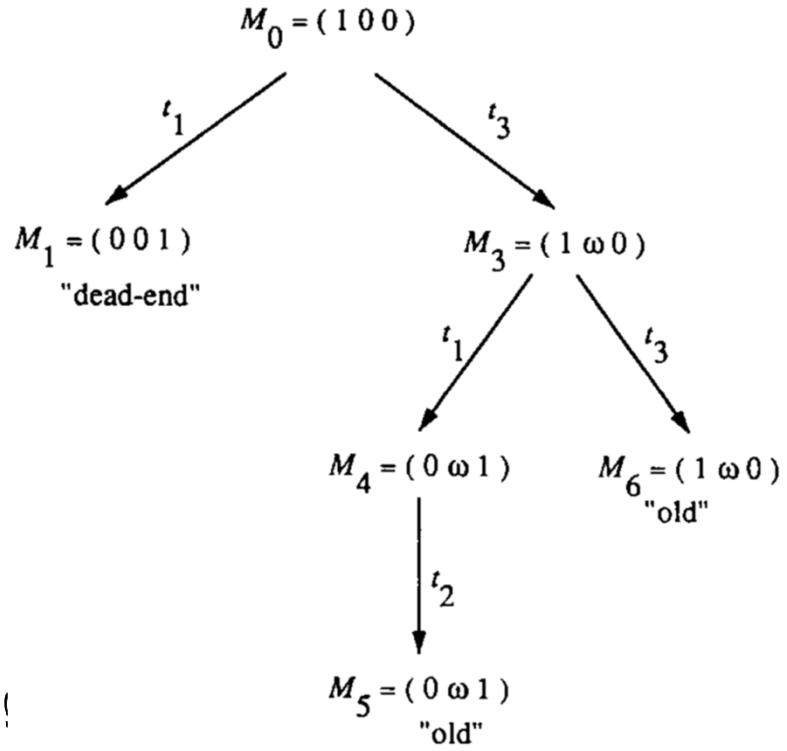
Reversibility

- A Petri net is reversible if for a marking M reached from M0, one can reach M0 from M
- May or may not be necessary
- But it is helpful if we have it
- If not, define a “home state”:
  - A marking M’ if from any arbitrary marking one can reach M’

Analysis Methods

- The coverability tree:
  - A tree that represents ALL possible markings
  - Leaves: markings generated from initial
  - Branches: a transition firing





Tree Formation Algorithm

Step 1) Label the initial marking M_0 as the root and tag it "new."

Step 2) While "new" markings exist, do the following:

Step 2.1) Select a new marking M.

Step 2.2) If M is identical to a marking on the path from the root to M, then tag M "old" and go to another new marking.

Step 2.3) If no transitions are enabled at M, tag M "dead-end."

Step 2.4) While there exist enabled transitions at M, do the following for each enabled transition t at M:

Step 2.4.1) Obtain the marking M' that results from firing t at M.

Step 2.4.2) On the path from the root to M if there exists a marking M'' such that $M'(p) \geq M''(p)$ for each place p and $M' \neq M''$, i.e., M'' is coverable, then replace $M'(p)$ by ω for each p such that $M'(p) > M''(p)$.

Step 2.4.3) *Introduce* M' as a node, draw an arc with label t from M to M', and tag M' "new."

Coverability Tree Results

1) A net (N, M_0) is *bounded* and thus $R(M_0)$ is finite *iff* (if and only if) ω does not appear in any node labels in T.
2) A net (N, M_0) is *safe iff* only 0's and 1's appear in node labels in T.
3) A transition t is *dead iff* it does not appear as an arc label in T.
4) If M is reachable from M_0, then there exists a node labeled M' such that $M \leq M'$.

Coverability Tree Observations

- For unbounded Petri nets:
  - The tree will be infinitely large
- For a bounded Petri net:

  coverability tree = reachability tree
- Usually have to consider another graph:
  - Coverability graph
- Not practical!
- Mathematical solution (incidence matrix)

Time and Petri Nets

- Previous examples do not involve time
- In many cyber-physical systems a quantitative notion of time is essential
- The idea of adding time to Petri nets is to introduce temporal constraints on its elements:
  - A transition must fire within 10 time units
  - A token must remain in a certain place for at least 2 time units
  - ...

Time and Petri Nets (Cont'd)

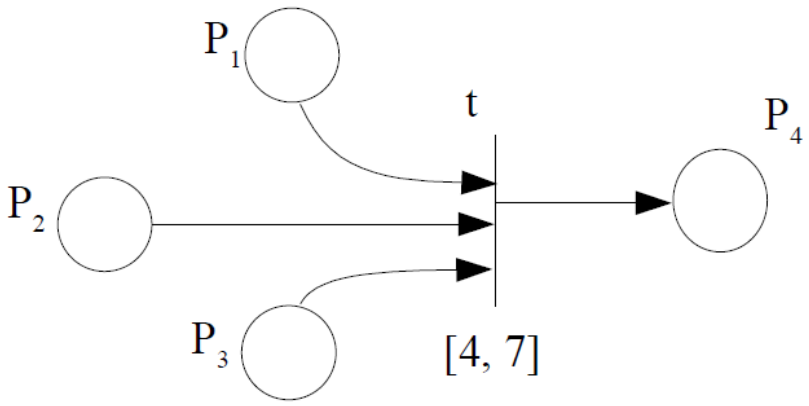
- Note that we talk about "time units", the actual unit is not our concern here
- Time has been added to Petri nets in many different ways:
  - Adding temporal constraints only to places, only to transitions, only to arcs, or to any combination of them
  - Almost all possible approaches have been studied in literature
  - All of them are called Timed Petri net. So beware of the differences!

Timed Petri Net

- Our focus: introducing the notion of temporal constraints on transitions:
  - Add a lower bound d and an upper bound D to transitions:
    - If transition t is associated with constraint [d, D] (with $d \leq D$), then after t is enabled:
    - it must fire no less than d and no more than D time units after it is enabled (unless it is disabled before)
    - D and d do not have to be integers, they can be any non-negative real numbers
  - Example in the next slide

Timed Petri Net Example

- Assume: a token arrives in place P1 at time 3, one in P2 at time 5, and one in P3 at time 1
- Transition t fires nondeterministically between times 9 and 12
- If d=D, transition has to happen exactly at d
- If a transition has no explicit temporal constraint, then [0, +∞] is assumed by default

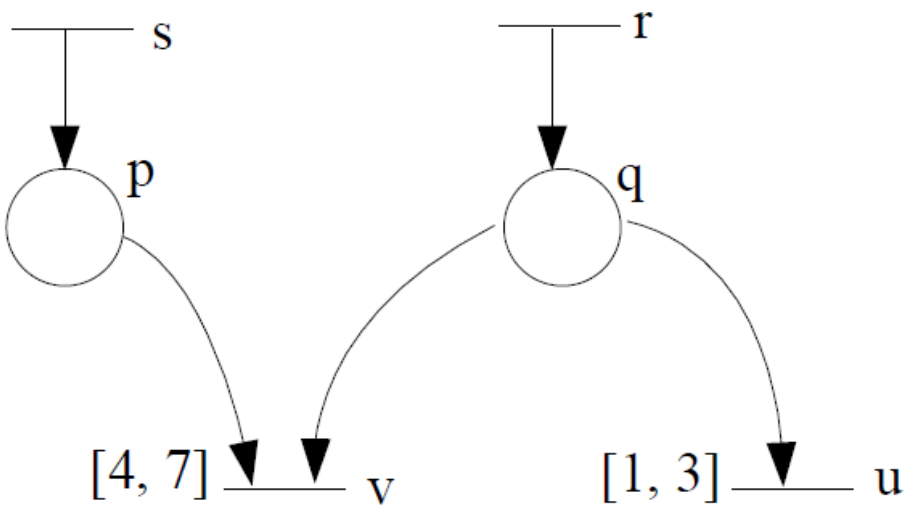


Time Semantics

- There are two ways to look at the time constraints:
  1. A transition must fire if it is still enabled when the upper bound on its constraint is reached (called Strong Time Semantics or STS)
     - With STS, a transition is forced to fire by its temporal upper bound (unless it is disabled first)
     - The most widely adopted semantics
  2. A transition is not forced to fire if it is still enabled when its upper bound is reached (called Weak Time Semantics or WTS)
     - If it fires, then the firing occurs within its bound

STS Example

- Assuming STS semantics:
  - If p and q get their tokens at time 0
  - u gets enabled and has to fire by time 3
  - v cannot fire before time 4
  - So v never gets a chance to fire

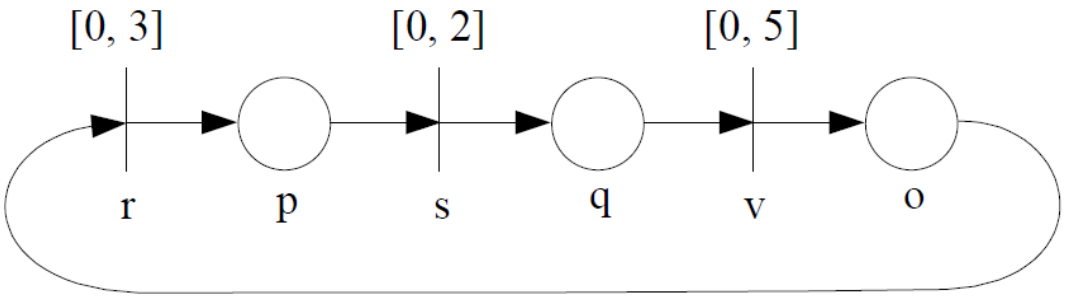


STS vs. WTS Question

- Which semantics is more useful?
  - WTS: closer to the original untimed Petri nets
    - Reminder: in untimed Petri nets a transition is not forced to fire if it is enabled, it could stay enabled and never fire
  - It has been shown that Petri nets with STS are "more powerful" than with WTS
  - Choosing the right semantics can greatly affect the system characteristics and performance
  - When building the model of a system through a TPN, one must specify the semantics with which the model should be interpreted

Zero-Time Transitions

- Transitions in which the lower bound is 0 are called zero-time transitions
- Since they can occur at the same time they are enabled, zero-time transitions may give rise to Zeno behavior
- The following sequence of firings in this example is admissible: <s, T>, <v, T>, <r, T>, <s, T>, <v, T>, <r, T>, <s, T>, ...
- T is when place p contains a token
- Seems that time is not advancing!

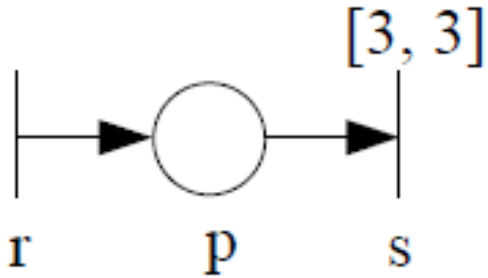


Avoiding Zeno Behavior

- Zeno behavior is physically impossible
- So it should be avoided
- Various solutions have been proposed to avoid this phenomenon
  - For instance, prohibiting zero-time transitions
  - But zero-time transitions have interesting practical applications
- The most common solution: disallow cycles with only zero-time transitions
  - There has to be at least one non-zero-time transition in any cycle

Lower Bound Semantics

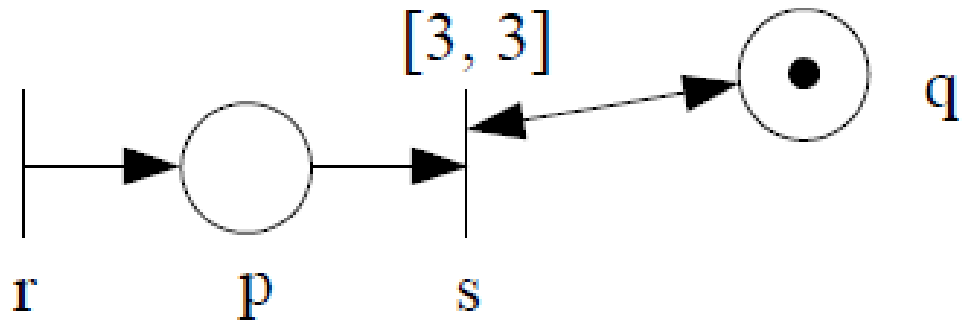
- Some have interpreted the lower bound as the minimum delay between consecutive firings of a transition (i.e., recharge time)
- Consider this example:
  - If transition r fires at times 1 and 2, when does s fire?



  - Traditional interpretation: at times 4 and 5
  - Considering recharge time: at times 4 and 7

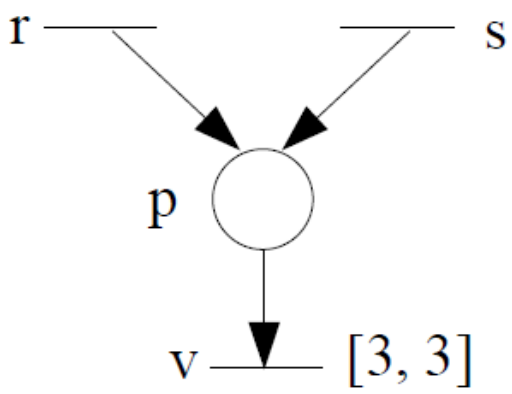
Which One to Use?

- Both semantics have been used
  - 1st one: to allow simultaneous firing of a transition
  - 2nd one: to allow recharge time and empty presets
- Recharge time can still be implemented using the first semantics (this example)
- Notice the double arc between q and s



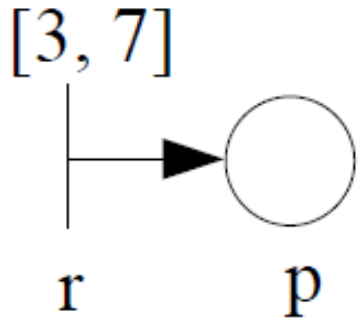
Simultaneous Firing of a Transition

- The first interpretation can lead to simultaneous firing of one transition
- In this example if r and s fire at the same time T, then v fires twice at time T+3, so the following firing sequence is valid:
  - <r, T>, <s, T>, <v, T+3>, <v, T+3>
- Again, this may be good or bad depending on what is to be achieved



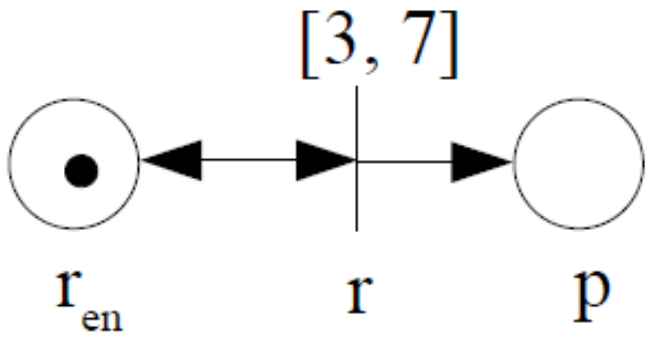
Empty Presets

- Untimed Petri nets allow transitions to have empty presets:
  - No input places means a transition is always enabled
- But with timed Petri nets, how can this model be interpreted?



Empty Presets (Continued)

- Common interpretation:
  - A transition with an empty preset is the same as a transition that has a place that is both an input and an output of the transition and
  - the initial marking contains a token
  - as shown below

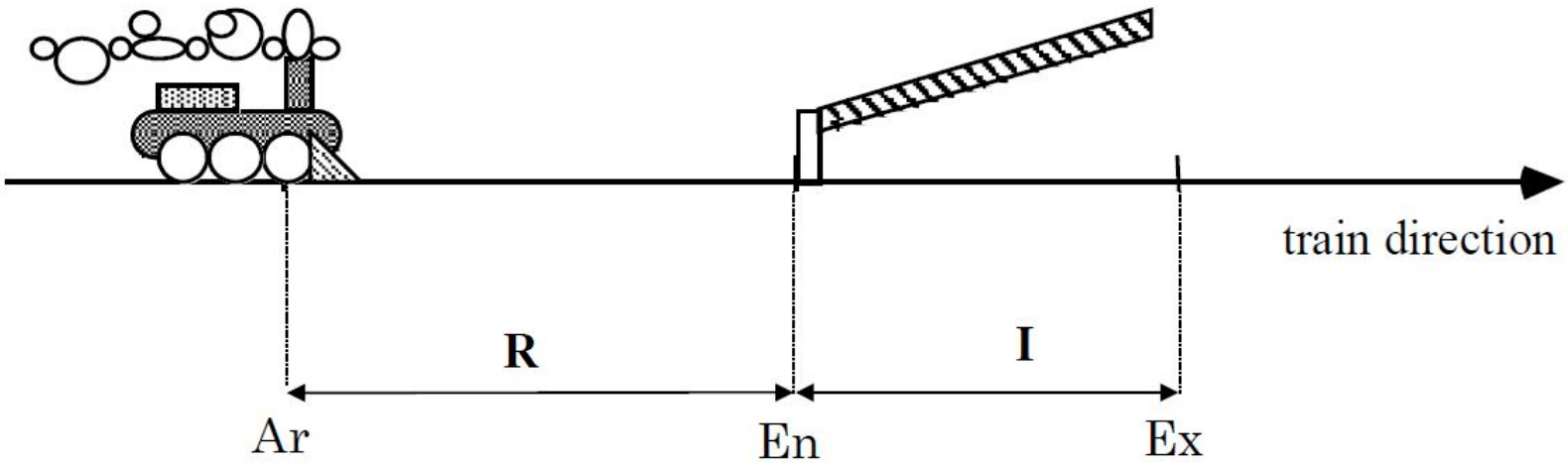


Infinite Upper Bound

- An infinite upper bound means that the transition may never fire
- Consistent with the traditional idea of Petri nets: an enabled transition may never fire
- Another intuitive interpretation: if a transition has infinite upper bound, then it must eventually fire, unless it is disabled
- The first one is consistent with WTS
- The second one is consistent with STS

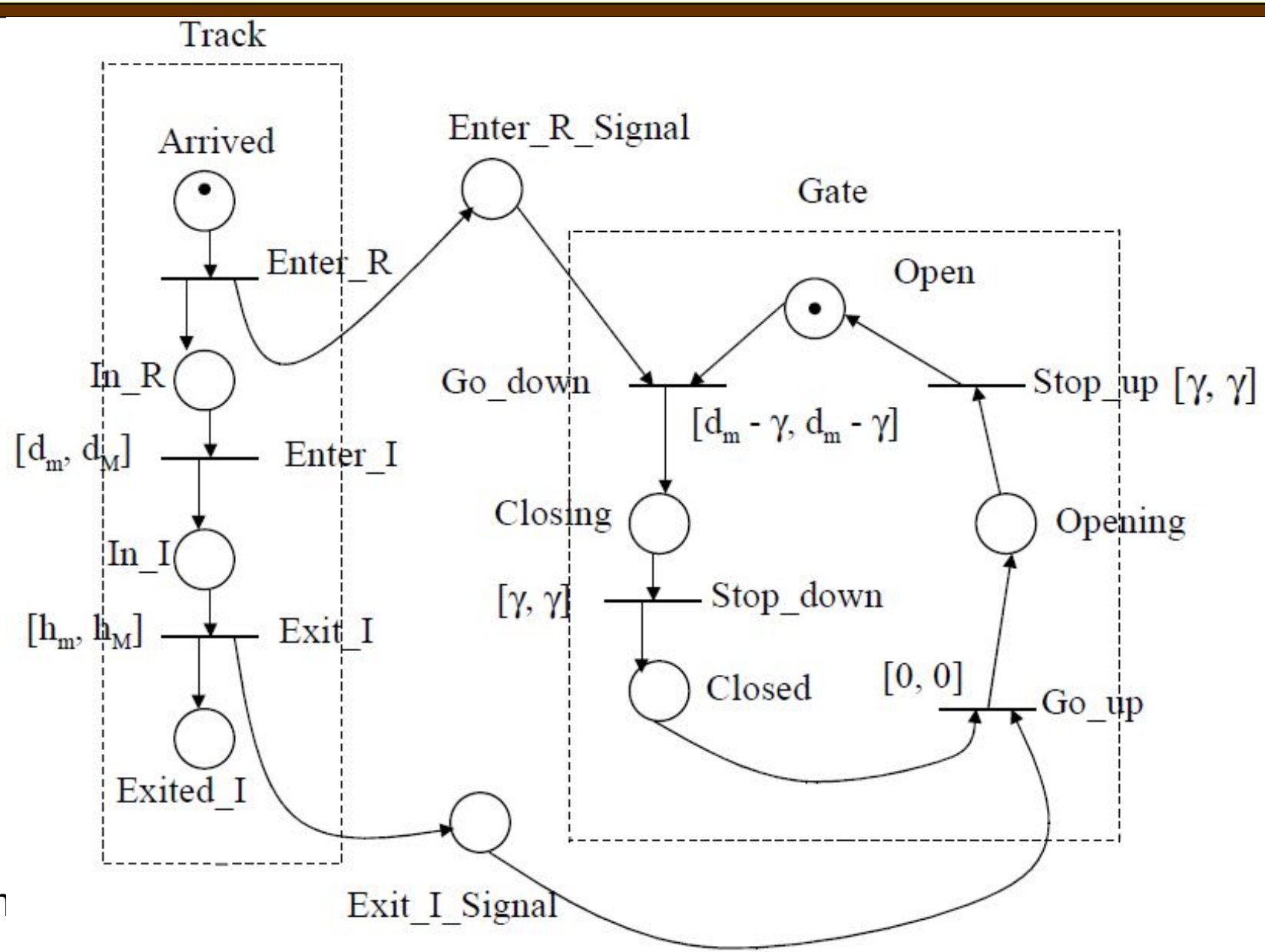
A Real-Time Example

- Kernel (simplified) railroad crossing example assumptions:
  - There is only one train
  - d_m and d_M are, respectively, the minimum and maximum time to go from the beginning of section R to the beginning of section I
  - h_m and h_M are, respectively, the minimum and maximum time to go through I
  - The gate can be open or closed but also moving up and down
  - The moving of the gate takes gamma time units and cannot be interrupted



En

Petri Net Model of KRC



Next Example: Elevator System

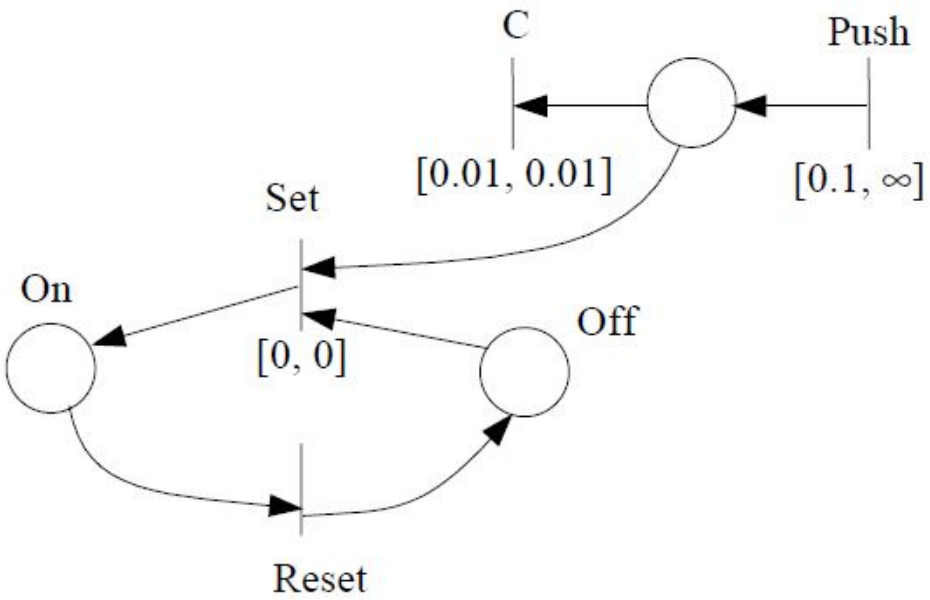
- An elevator system is to be installed in a building with m floors.
- The elevator cabin has a set of buttons, one for each floor. The buttons light up when pressed and cause the elevator to visit the corresponding floor. The lights switch off when the elevator visits that floor.
- Each floor has one button to request the elevator. This button lights up when pressed. The light switches off when the elevator visits the floor.
- When an elevator has no requests to service, it should remain at its last destination and await further requests.
- All requests for elevators from floors must be serviced eventually with all floors given equal priority.
- All requests for floors within cabin must be serviced eventually with floors being serviced sequentially in the direction of the cabin's travel.

System Decomposition

- The system is composed of a certain number of parts that evolve concurrently:
  - The elevator cabin
  - The buttons (both internal and external)
- Each component has a dynamics of its own, which can be described through a timed Petri net
  - The components interact through shared events (e.g., a cabin arriving to a floor switches the corresponding button off)
  - The shared events are modeled in TPNs as transitions whose firings are controlled by places belonging to subnets modeling the dynamics of different components
  - Parts of the complete diagram are in the next slides

Button Model

- All buttons have the same dynamics
  - Transition Reset is shared with other parts of the system
  - So other incoming arcs will be added later
  - The light is switched off only if the cabin arrives to the desired floor
  - Transition C is a token consumer, it prevents the accumulation of tokens in place P if a request is still pending



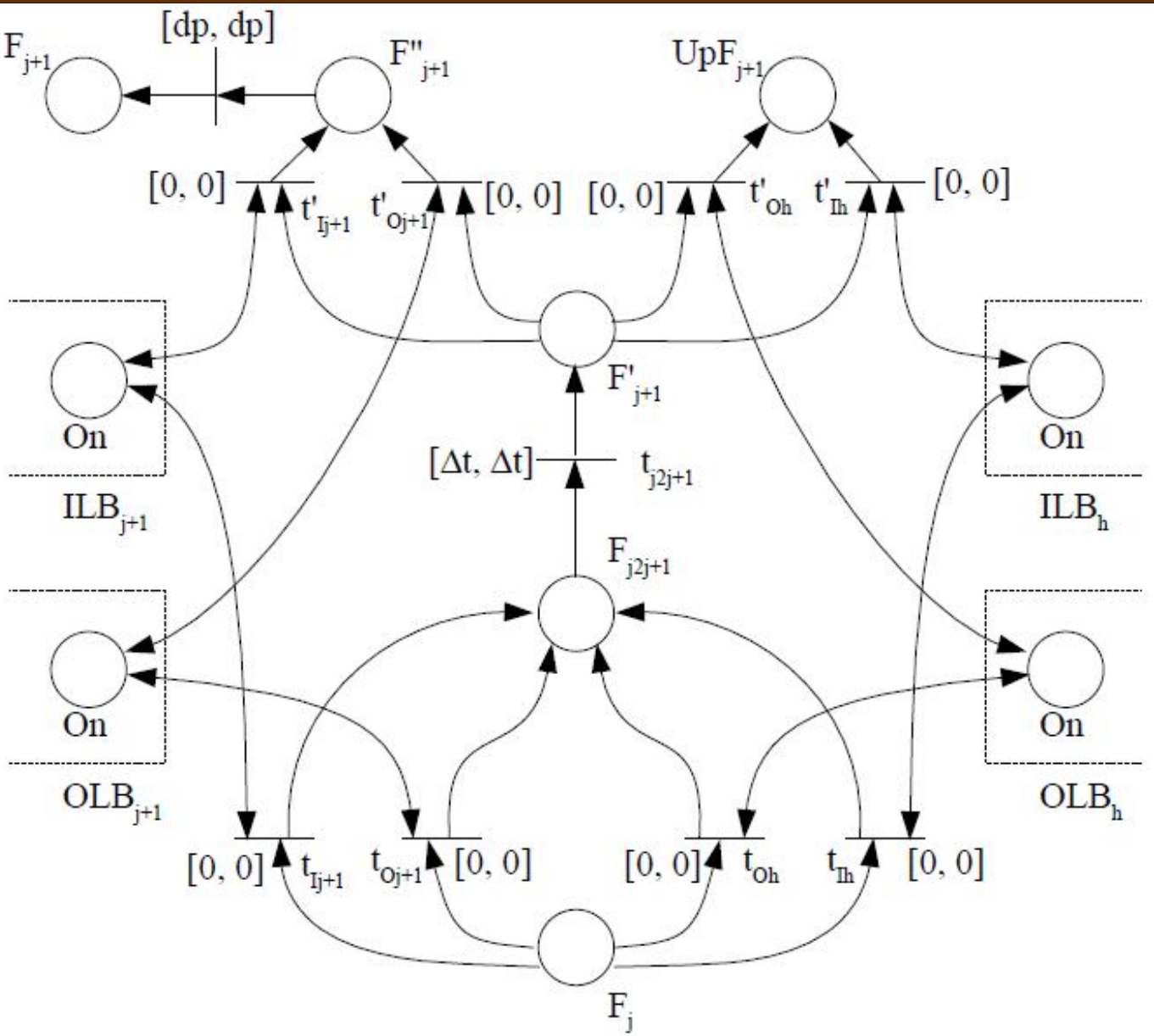
Cabin Model

- The cabin interacts with all of the buttons.
- Every button is modeled through an instance of the TPN described before.
- Only parts of the TPNs representing the various buttons are shown.
- The dynamics of the elevator cabin going up is symmetric to the one of the cabin going down => only the part concerning the upward movement is shown.
- Only a generic cabin movement from floor j to floor j+1 is shown. The complete upward dynamics is made of m-1 of these diagrams cascaded together.
- The last floor (floor m) should be modified accordingly as the cabin cannot go further up.

Cabin Model (Continued)

- In the subdiagram representing the movement upwards from the j^{th} to the $(j+1)^{th}$ floor, h represents any floor above $(j+1)^{th}$ floor
- ILB_{j+1} is the TPN representing the button inside the cabin for floor $j+1$
- ILB_h is the TPN representing the buttons inside the cabin for floors above $(j+1)^{th}$
- OLB_{j+1} is the TPN representing the button on $(j+1)^{th}$ floor
- OLB_h is the TPN representing the buttons on floors above $(j+1)^{th}$

Cabin Petri Net



Embe

Cabin Petri Net Description

- Place F_j represents the cabin being stopped at floor j.
- Place F_{j2j+1} represents the cabin moving from floor j to floor j+1.
- Place F'_{j+1} represents the cabin having just arrived at floor j+1.
- The transition between F'_{j+1} and F''_{j+1} represents the cabin remaining at the floor at least dp time units (where dp is the time for a person to get off the cabin).
- When the cabin arrives at the floor j+1, there are two possible options: either it stops at the floor (place F_{j+1}) or it keeps moving to floor j+2 without stopping (place UpF_{j+1}); if there are requests pending both for floor j+1 and for floors above j+1, the choice is entirely nondeterministic (that is, the cabin might not stop even if a request for floor j+1 is active).
- The cabin moves upward from floor j only if there is at least one pending request from/for a floor above j.
- Δt is the fixed time to move from one floor to the next.