CS 4072 - Topics in CS
Process Mining

Lecture # 04

February 22, 2022

Spring 2022

FAST - NUCES, CFD Campus

Dr. Rabia Maqsood

rabia.maqsood@nu.edu.pk

Today’s Topics

- Process modeling and analysis
  - Different techniques and limitations
- Petri-net: basics

Process Modeling and Analysis

Process Modeling

- A variety of notations are available to model (business or operational) processes.
  - Petri nets
  - Business Process Modeling Notation (BPMN)
  - C-nets (variant of Petri nets)
  - EPCs
  - YAWL
  - Process trees
  - …

Process Modeling and Analysis

- Models are used to reason *about processes* (redesign) and to make decisions *inside processes* (planning and control).
- Models used in operations management are typically tailored towards a particular analysis technique and only used for answering a specific question.
- However, process models in BPM typically serve *multiple* purposes.
  - E.g., discuss responsibilities, analyze compliance, predict performance using simulation, and configure a workforce management (WFM) system.

Process Modeling and Analysis

- Thus, making a good model is “an art rather than a science”.

- Creating models by-hand is difficult and error-prone. Typical errors include:
  - Model describes an idealized version of reality
  - Inability to adequately capture human behavior
  - Model is at the wrong abstraction level

- Only experienced designers and analysts can make models that have a good predictive value and can be used as a starting point for (re)implementation and redesign.

Process Modeling and Analysis via Process Mining

- Process mining allows for the extraction of models based on *facts*.
- Provides various views on the same reality at different abstraction levels.
- Process mining can also reveal that people in organizations do not functions as “machines”.

Process Modeling and Analysis via Process Mining

- Our focus is on control-flow perspective of processes. That is, we assume there is a set of ***activity labels*** $\mathcal{A}$.

- The goal of process model is to decide *which activities* need to be executed and in *what order*.
  - Activities can be executed sequentially, activities can be optional or concurrent, and can be repeated multiple times.

Transition Systems

- The most basic process modeling notation is a *transition system*.

Definition 3.1 (Transition system) A *transition system* is a triplet $TS = (S, A, T)$ where S is the set of *states*, $A \subseteq \mathcal{A}$ is the set of *activities* (often referred to as *actions*), and $T \subseteq S \times A \times S$ is the set of *transitions*. $S^{start} \subseteq S$ is the set of *initial states* (sometimes referred to as “start” states) and $S^{end} \subseteq S$ is the set of *final states* (sometimes referred to as “accept” states).

Transition Systems

If the state space is finite, then a transition system is also referred to as a Finite-State Machine (FSM) or finite-state automaton.

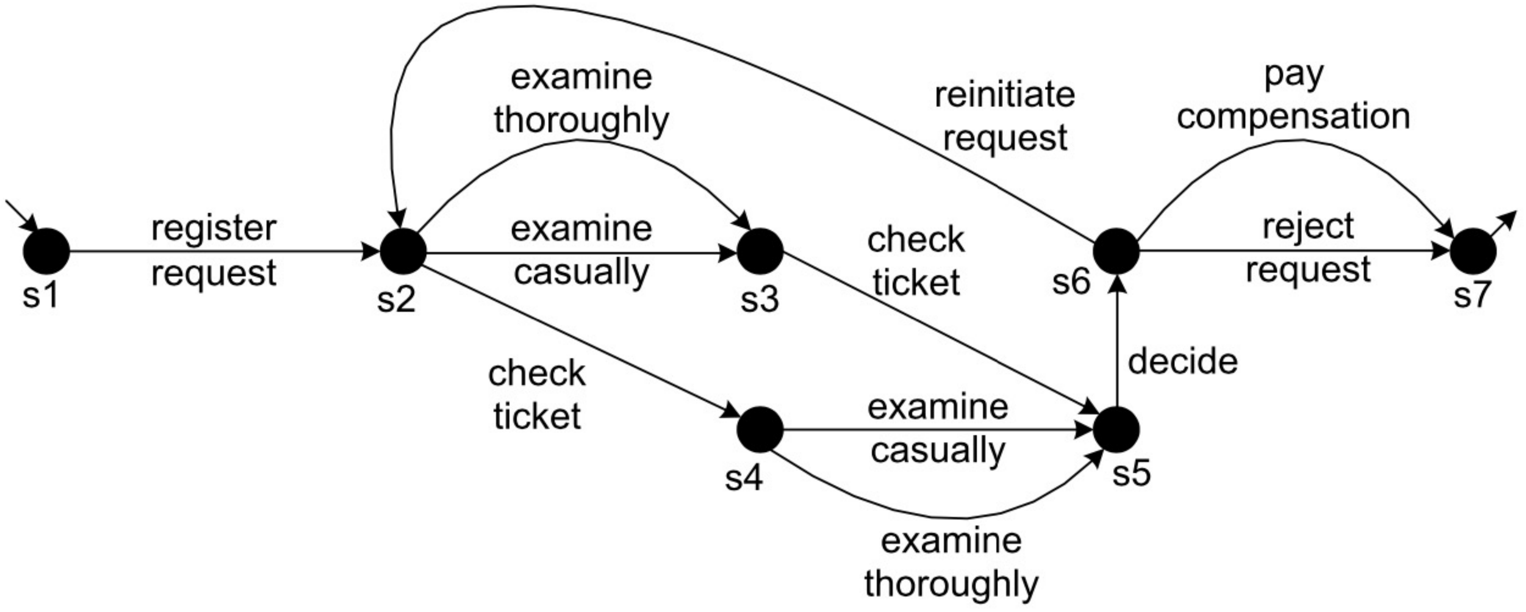


Fig. 3.1 A transition system having one initial state and one final state

S = {s1, s2, s3, s4, s5, s6, s7}, S^{start} = {s1}, S^{end} = {s7}, A = {*register request*, *ex- amine thoroughly*, *examine casually*, *check ticket*, *decide*, *reinitiate request*, *reject request*, *pay compensation*}, and T = {(s1, *register request*, s2), (s2, *examine casually*, s3), (s2, *examine thoroughly*, s3), (s2, *check ticket*, s4), (s3, *check ticket*, s5), (s4, *examine casually*, s5), (s4, *examine thoroughly*, s5), (s5, *decide*, s6), (s6, *reinitiate request*, s2), (s6, *pay compensation*, s7), (s6, *reject request*, s7)}

Transition Systems: limitations

- Transition systems are simple but have problems expressing concurrency succinctly.

- For *n* parallel activities, there are *n!* possible execution sequences.
  - The transition system requires 2^n states and $n \times 2^{n-1}$ transitions.
  - Consider for example 10 parallel activities. The number of possible execution sequences is 10! = 3,628,800, the number of reachable states is $2^{10} = 1024$, and the number of transitions is $10 \times 2^{10-1} = 5120$.

The corresponding Petri net is much more compact and needs only 10 transitions and 10 places to model the 10 parallel activities.

Business Process Modeling Notation (BPMN)

- BPMN has become popular recently to model business processes.
- Activities are connected using gateways. There should be one input and output line from each activity.

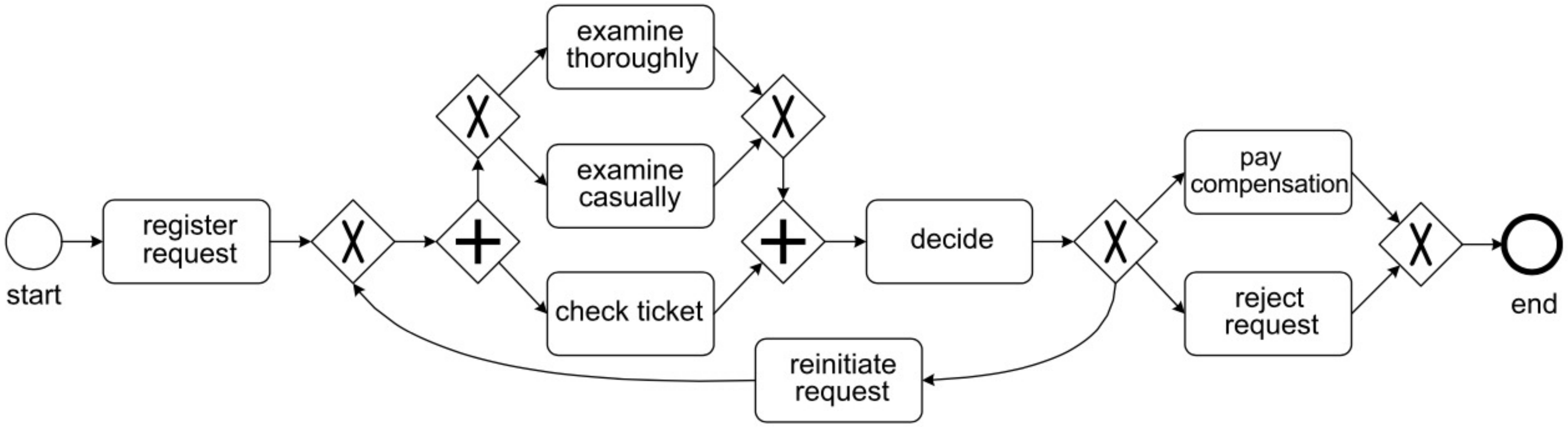
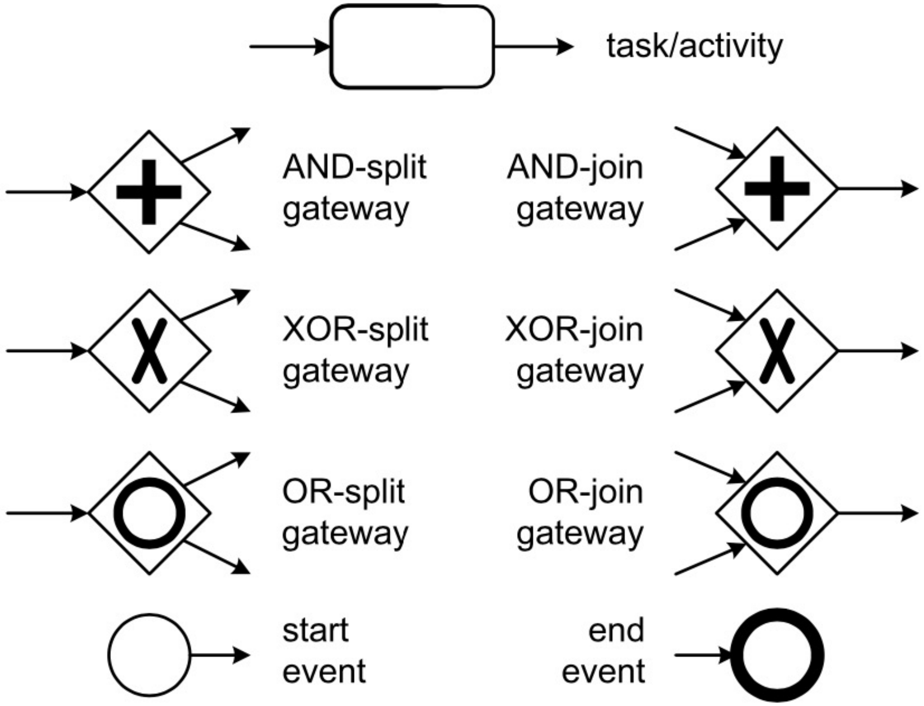


Fig. 3.7 Process model using the BPMN notation

BPMN Symbols



Petri Nets

- Petri nets are the oldest and best investigated process modeling language allowing for the modeling of concurrency.
- Petri nets are executable, and many analysis techniques can be used to analyze them.
- Petri nets were introduced by C.A. Petri in his Ph.D. Dissertation: ”Kommunikation mit Automaten.”, Institut für Instrumentelle Mathematik, Bonn, 1962.
- They are particularly useful form modeling systems with concurrent and asynchronous processing .

Why concurrency is a problem?

- Concurrency and asynchronous processing is typical in real world.
- It can pose a problem when many entities (people, machines, processing threads) which use (share) the same resource (or a limited number of resources).
- A trivial example is that of an elevator – the cabin is single resource that many people want to use. The problem is how to control the elevator to minimise waiting time?

Why concurrency is a problem?

- The elevator '"scheduling" control is not crucial, at worst improper algorithm may result in long waiting times.
- There are situations where handling concurrency is crucial for correct behaviour of the system.
- Unwanted results of concurrency include:
  - Race conditions
  - Resource starvation
  - Deadlocks

Race conditions

- When race conditions occur, the result of the system behaviour may be unexpected and is dependent on the sequence of other events.
- Race conditions were first described in electronics (logic circuits), where parallelism is typical.
- An example of race conditions may be poorly implemented ATM.

ATM example

- Let's imagine the ATM operation is following:
  - Authorise client
  - Get account balance
  - Get client request
  - Update account
  - Dispense cash

- This can translate to following situation:

Time = t_0	Authorisation OK
Time = t_0+2s	Balance is 1000
Time = t_0+10s	Client requested 800, 800<1000
Time = t_0+12s	Account updated by -800, 200 left
Time = t_0+14s	Cash dispensed

- Everything worked fine.

ATM example

- Now, lets imagine that there are two cards attached to the same account (wife and husband, corporate account etc.)
- Two persons at nearly the same time want to withdraw money. What may happen?

ATM example

	Client 1	Client 2
Time = t_0	Authorisation OK	
Time = t_0+1s		Authorisation OK
Time = t_0+2s	Balance is 1000	
Time = t_0+3s		Balance is 1000
Time = t_0+5s		Client requested 800, 800<1000
Time = t_0+7s		Account updated by -800, 200 left
Time = t_0+9s		Cash dispensed
Time = t_0+10s	Client requested 800, 800<1000	
Time = t_0+12s	Account updated by -800, **-600 left**	
Time = t_0+14s	Cash dispensed	

Race conditions

- Race conditions may be resolved by resource locking

	Client 1	Client 2
Time = t_0+2s	Balance is 1000, lock account	
Time = t_0+3s		account locked - cannot proceed
Time = t_0+10s	Client requested 800, 800<1000	
Time = t_0+12s	Account updated by -800, 200 left, unlock account	
Time = t_0+14s	Cash dispensed	

Resource starvation

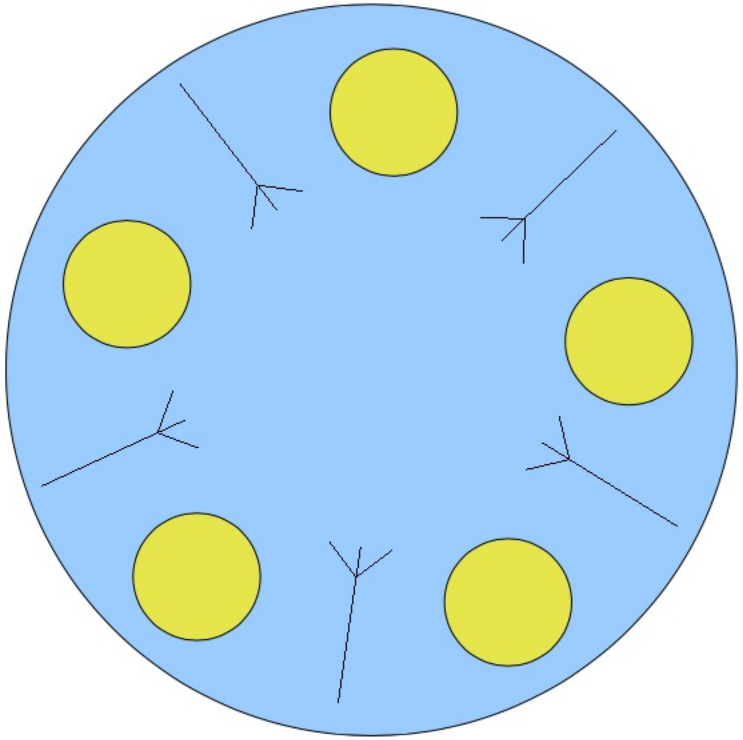
- The process (person, system) is denied the resource it needs, because other process is not freeing them.

Deadlock

- Deadlock occurs when a number (at least 2) processes are waiting for the other to finish (or release resources) and therefore none progresses.

- This can be illustrated by “dining philosophers” problem (invented by Edsger Dijkstra).

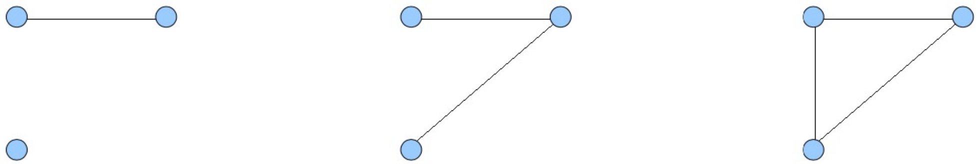
“Dining Philosophers” problem



- The philosopher is either thinking, or eating
- The philosophers do not talk (communicate)
- The philosopher can pick a fork to her/his right or left, one at a time
- The philosopher needs both forks to eat

Graphs

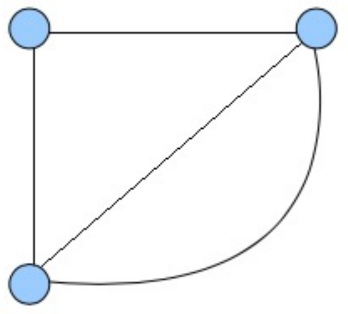
- Petri nets are graphs.
- Graph is an ordered pair G:=(V,E), where V is a set of nodes (vertices), E is a set of pairs of distinct vertices – edges (lines).
  - If E is a set of unordered vertices, the graph is undirected
  - If E is a set of ordered vertices, the graph is directed (digraph)



- Petri nets are digraphs.

Multigraph

- A multigraph is a graph that may contain more than one edge connecting the same vertices.

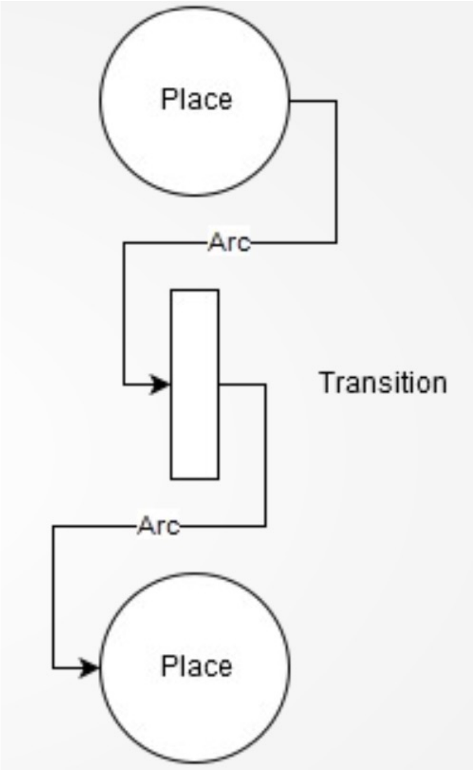


Bipartite graph

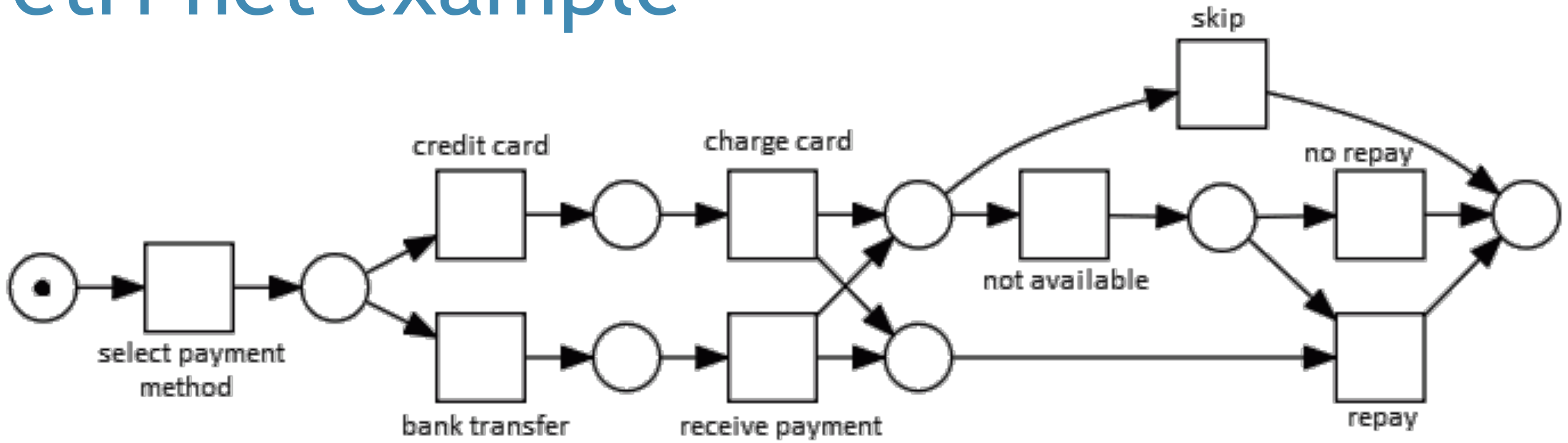
- Petri nets are bipartite graphs.
- In bipartite graphs the set of vertices V can be divided into two disjoint subsets V1 and V2 such that any edge always connects vertices from different subsets.
- The graph is sometimes denoted as G:=(V1+V2, E).

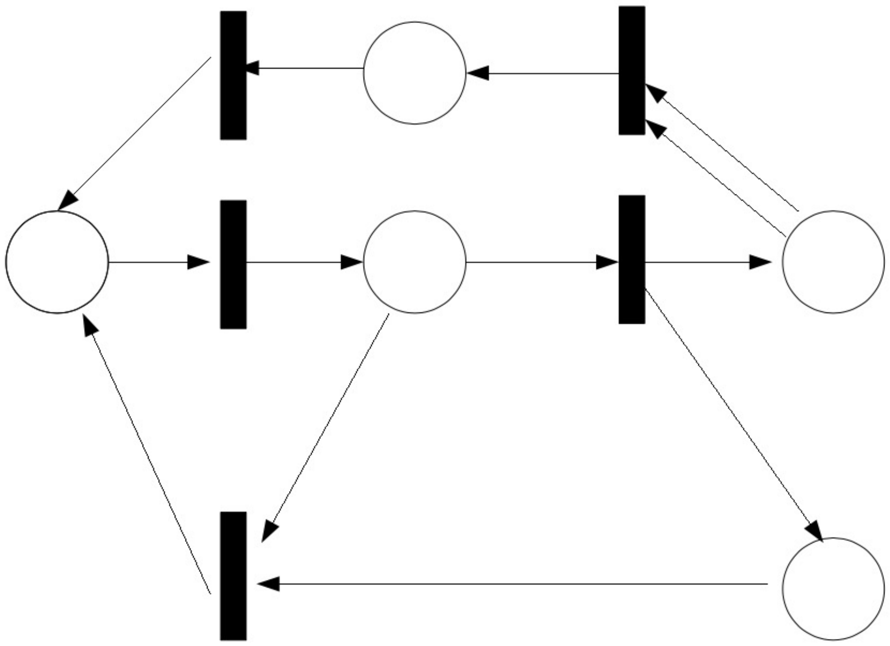
Petri Nets as graphs

- In Petri nets nodes of the first subset of vertices are called **places**, nodes of the second – **transitions**.
- The symbol of a place is a circle or an ellipse.
- The symbol of transition is a solid bar or a rectangle.
- The edges of the graph are called arcs.



Petri net example





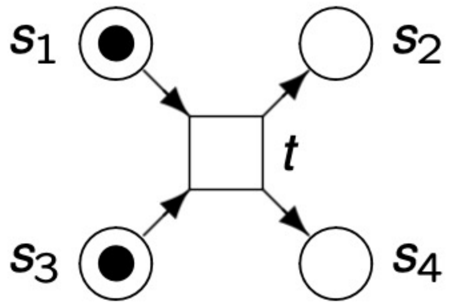
Tokens

- In order to describe dynamics of Petri nets (and being able to "execute" them) another concept is introduced – that of a **token**.
- The tokens are denoted by a solid dot and can be placed inside the place symbol.
- They indicate presence or absence of, for example, resource.
- Places can hold any number of tokens or only a limited number (capacitated places).

Behaviour of Petri nets

- Tokens are used to describe enabling of transitions.
- If an arc is drawn from a place to a transition, it indicates that a token in the place is required to enable the transition.
- If many arcs are drawn (multigraph!), its number indicates the number of required tokens (also referred to as **weight** of an arc).
- The transition is enabled iff for all arcs coming to the transition the condition of the required tokens are met.
- If a transition is enabled, it can **fire**.

Behaviour of Petri nets



The number of the tokens removed and inserted may be different!

Firing sequence is a sequence of transitions firing.

In the example, transition ***t*** may “fire” if there are tokens on places *s*1 and *s*3. Firing ***t*** will remove those tokens and place new tokens on *s*2 and *s*4.

<u>Transitions</u> symbolise **actions**; <u>places</u> symbolise **states** or **conditions** that need to be met before an action can be carried out.

Input and output sets

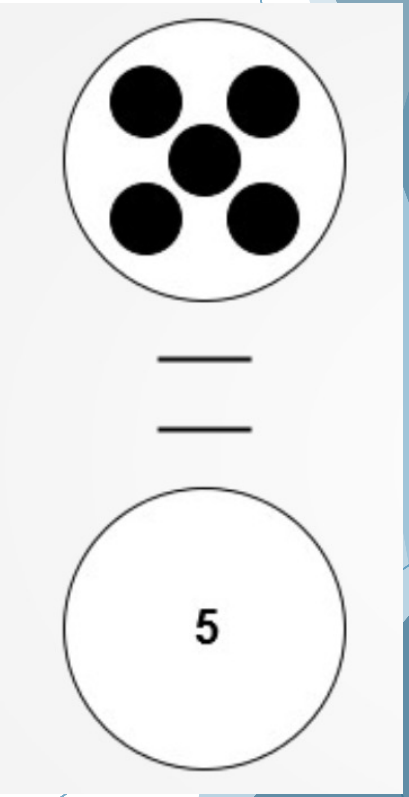
- The input set (preset) of a transition t, denoted •t, is a set of all places for which there are arcs going from these places to the transition t.

- The output set (postset) of a transition t, denoted t•, is a set of all places for which there are arcs going from transition t to these places.

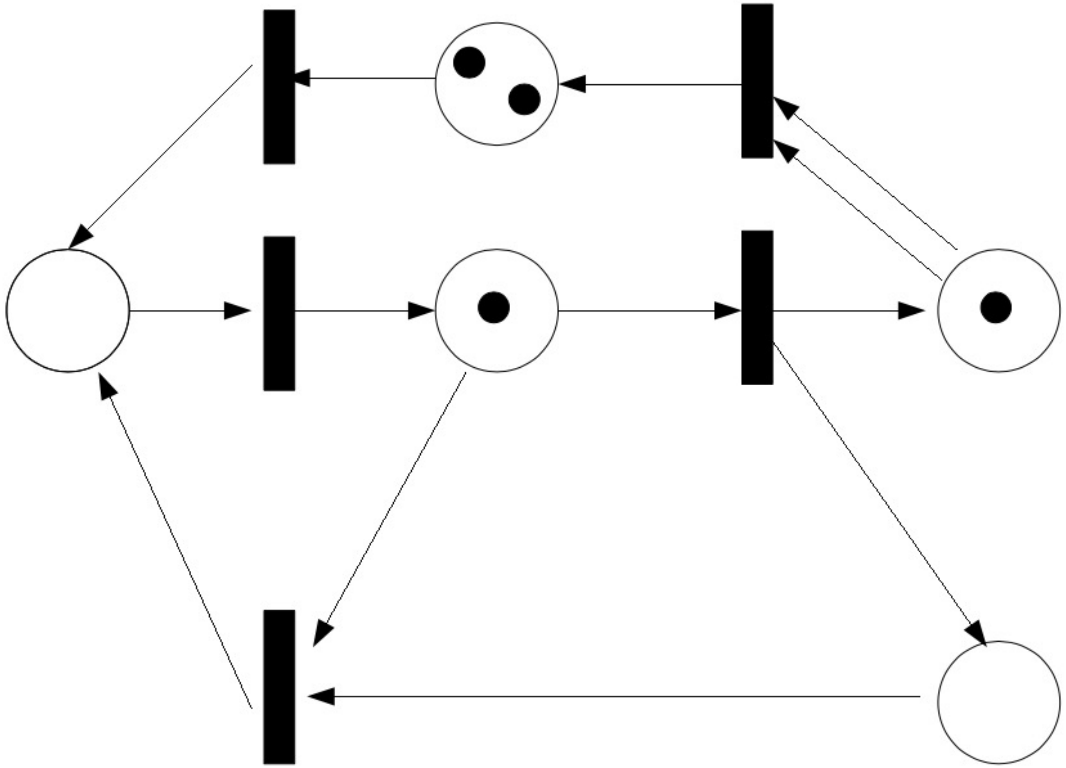
- Similar definitions apply to input and output sets of a place p, denoted by •p and p•, respectively.

Marking

- The state of a Petri net is determined by the distribution of tokens over places and is referred to as its **marking**.
- It is a mapping P→{0, 1, 2, ...} that describes the number of tokens present in each place.
- The marking of the net at the beginning of an analysis is called **initial marking**.

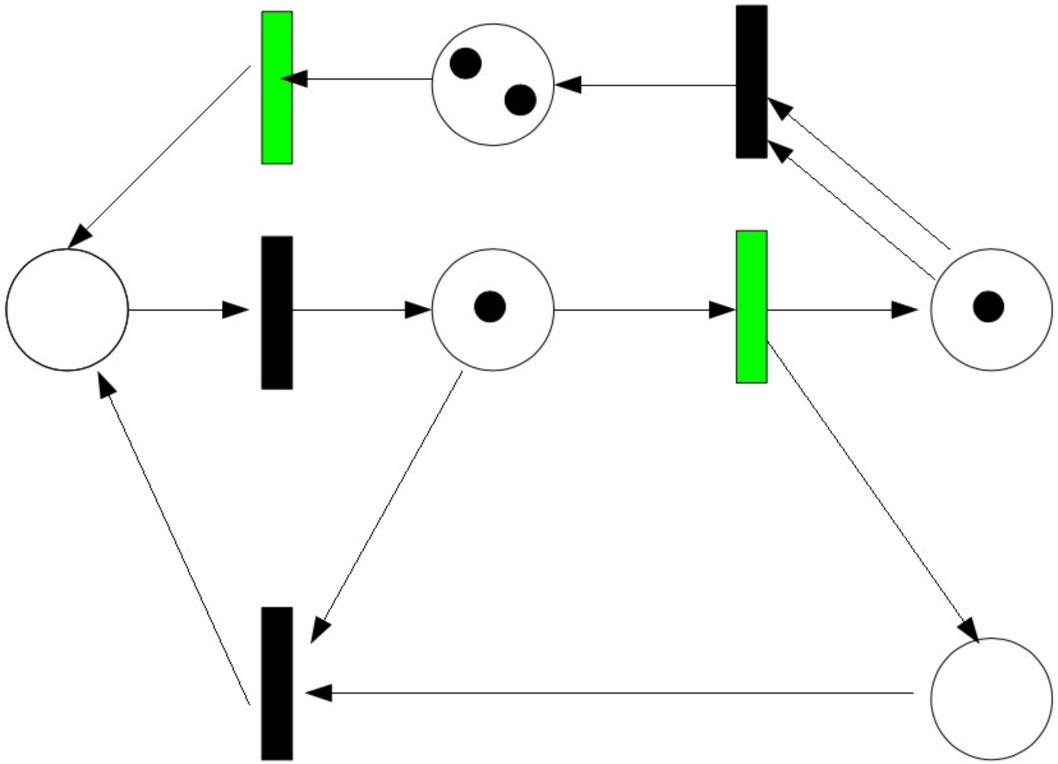


Sample Marking



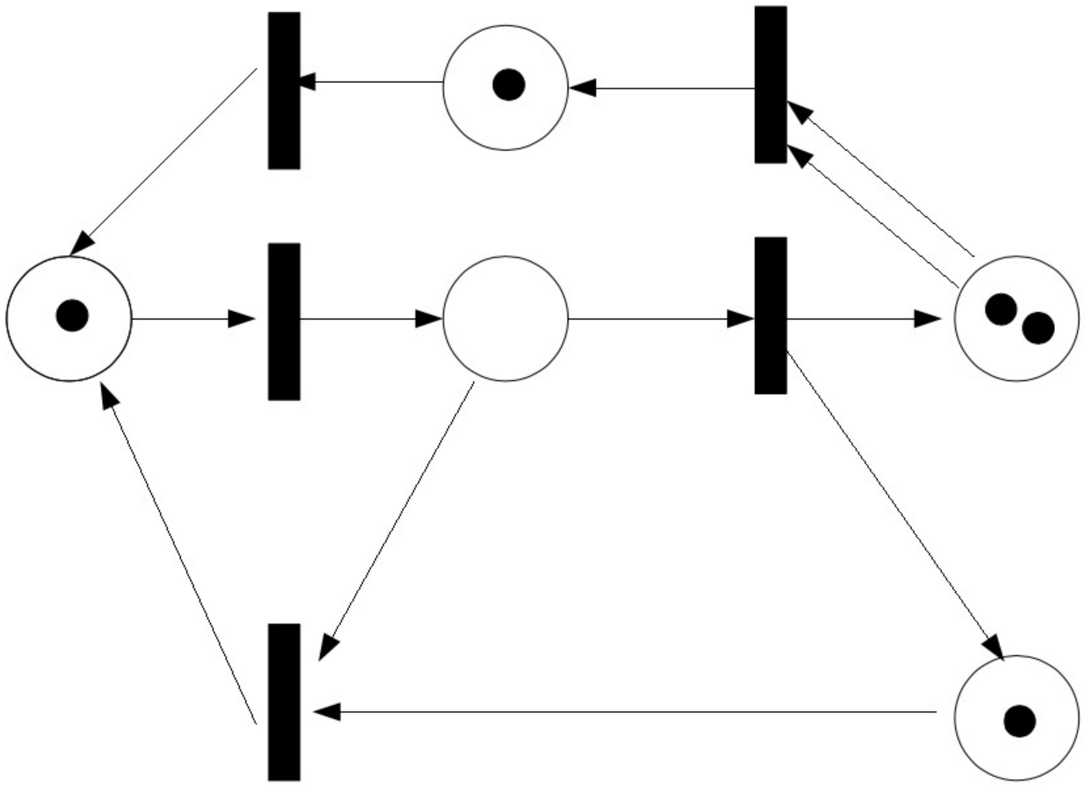
Question: which of the transition(s) are **enabled**?

Firing



Question: what will be the resultant state of the Petri net if we **fire** all the enabled transitions one by one.

Firing



Petri Net

Definition 3.2 (Petri net) A *Petri net* is a triplet N = (P,T,F) where P is a finite set of *places*, T is a finite set of *transitions* such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the *flow relation*.

A *marked* Petri net is a pair (N,M), where N = (P,T,F) is a Petri net and where $M \in \mathbb{B}(P)$ is a *multi-set* over P denoting the *marking* of the net. The set of all marked Petri nets is denoted $\mathcal{N}$.

Petri Net Example

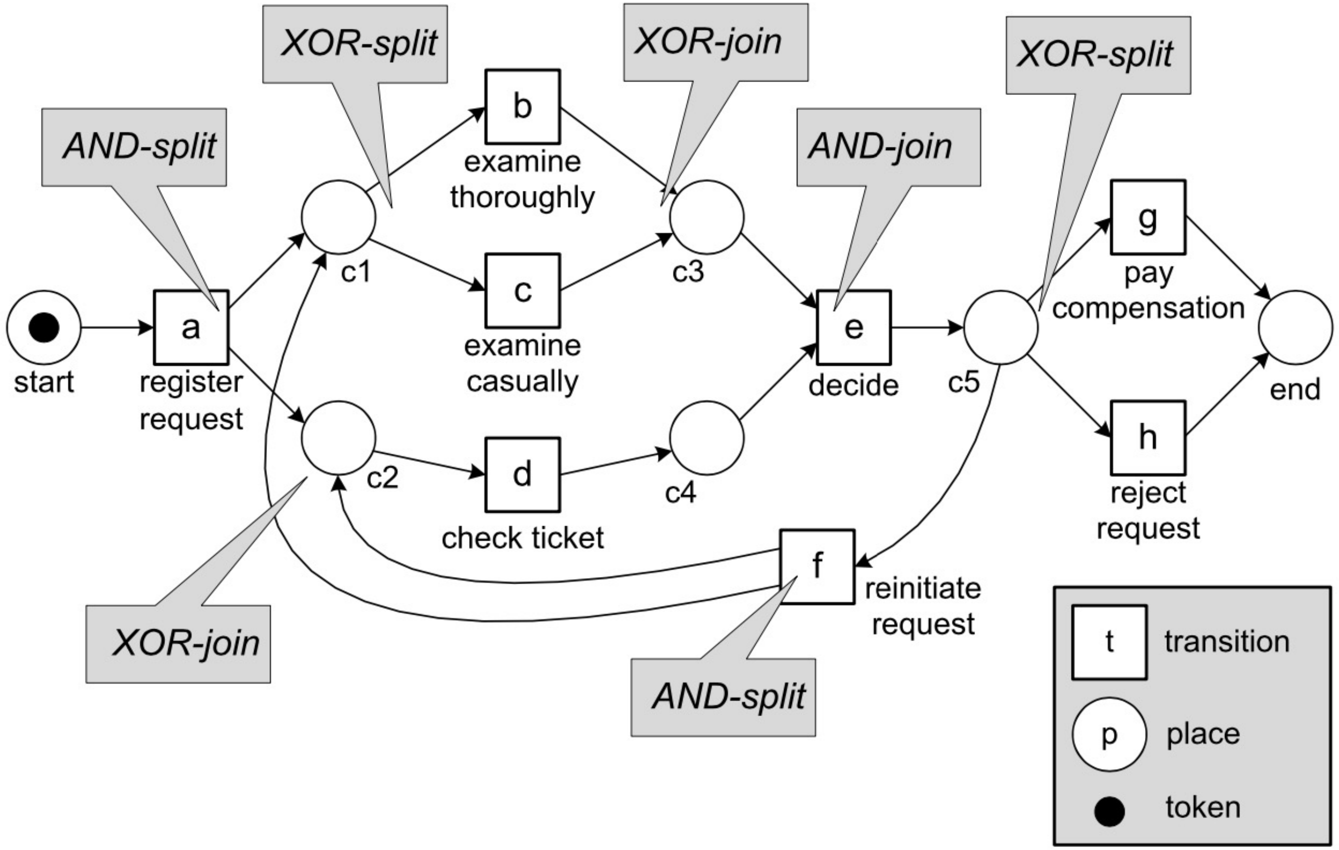


Fig. 3.2 A marked Petri net

P = {*start*, c1, c2, c3, c4,c5,*end*}
T ={a, b, c, d, e, f, g, h}
F ={(*start*,a), (a,c1), (a,c2), (c1,b), (c1,c), (c2,d), (b,c3), (c,c3), (d,c4), (c3,e), (c4,e), (e,c5), (c5,f), (f,c1), (f, c2), (c5, g), (c5, h), (g, *end*), (h, *end*)}

•c1 = {a, f } and c1• = {b, c}

Usefulness of Petri nets

- Petri nets can be used to model complex processes.
- Petri nets can be simulated (executed) in order to illustrate and test system behaviour, benchmark its speed etc.
- It is possible to perform a formal analysis of Petri net to find possible problems of the systems (for example deadlocks).
- For different applications the places and transitions may have different interpretations.

Interpretations of places and transitions

Input places
required resources
input data
input signals
buffers/registers

Transitions
task
computations
signal processing
processor

Output places
freed resources
output data
output signals
buffers/registers

Reading Material

- Chapter 3: Aalst