COMP201 – Software Engineering I
Lecture 9 – Finite State Machines and Petri Nets

Lecturer: T. Carroll

Email: Thomas.Carroll2@Liverpool.ac.uk

Office: G.14

See Vital for All Notes

Today

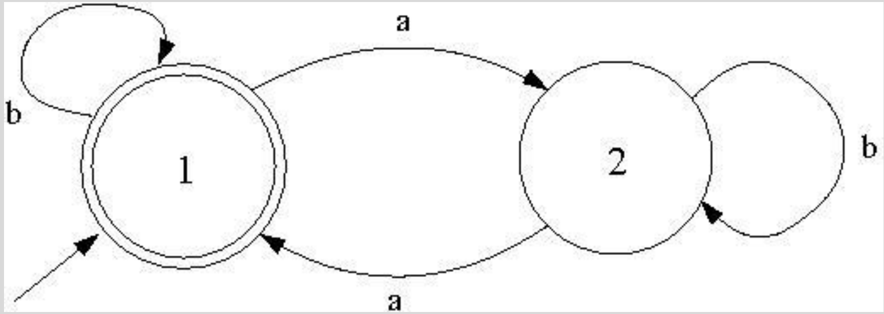
Overview

- Finite State Machines
- Mealy Machines
- Moore Machines
- Intro to Petri Nets

Finite State Machines

Finite State Machines (Finite State Automata)

- Model behaviour of a system or a complex object
- Limited number of defined states, changing with circumstance
- States change with input
- Machine either accepts a string, or does not accept a string
- *You may recall finite state machines (or automata) from COMP209.*



Finite State Machines - Definition

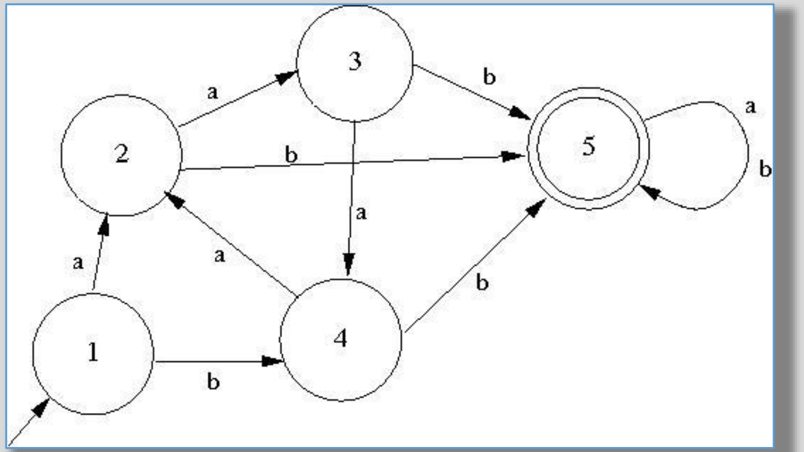
- A *model of computation* consisting of
  - a set of *states*
  - a *start (initial) state*,
  - Accepting state,
  - an input *alphabet*, and
  - a *transition function* that maps input symbols and current states to a *next state*

Finite State Machines - Definition

- Computation begins in the start state with an input string over the input alphabet.
- Automaton transitions to new states depending on the transition function.
  - ***states*** *define behaviour and may produce actions*
  - ***state transitions*** *are movement from one state to another*
  - ***input events*** *are either externally or internally generated, which may possibly trigger rules and lead to state transitions.*

Excercise

- What strings would be accepted by this Automaton?



Variants of FSMs

- There are many variants, for instance,
  - machines having actions (outputs) associated with transitions (***Mealy machine***) or states (***Moore machine***),
  - multiple start states,
  - transitions conditioned on no input symbol (a null) or more than one transition for a given symbol and state (***nondeterministic finite state machine***),
  - one or more states designated as *accepting states* (***recognizer***), etc.

FSM with Output (Mealy and Moore Machines)

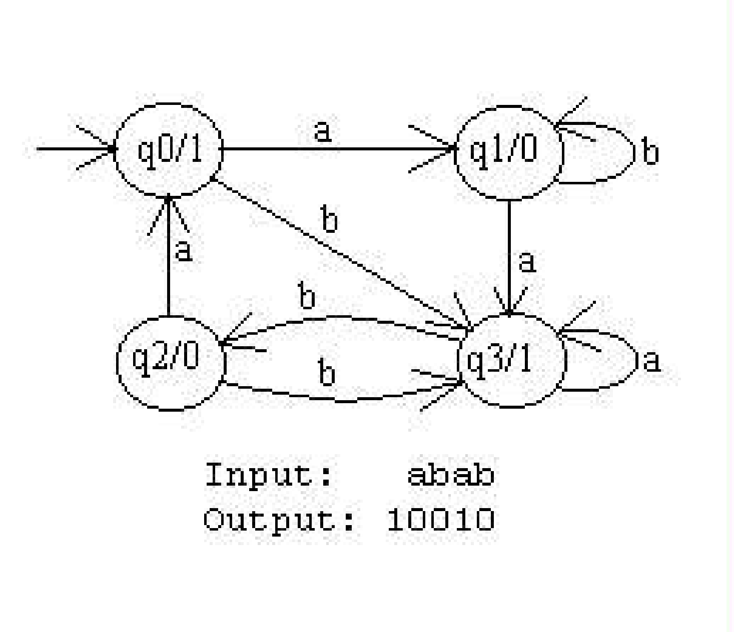
- Finite state automata are like computers:
  - they receive input
    they process the input by changing states.
- The only output that we have seen finite automata produce so far is a yes/no at the end of processing.
- We will now look at two models of finite automata that produce more output than a yes/no.

Moore Machines

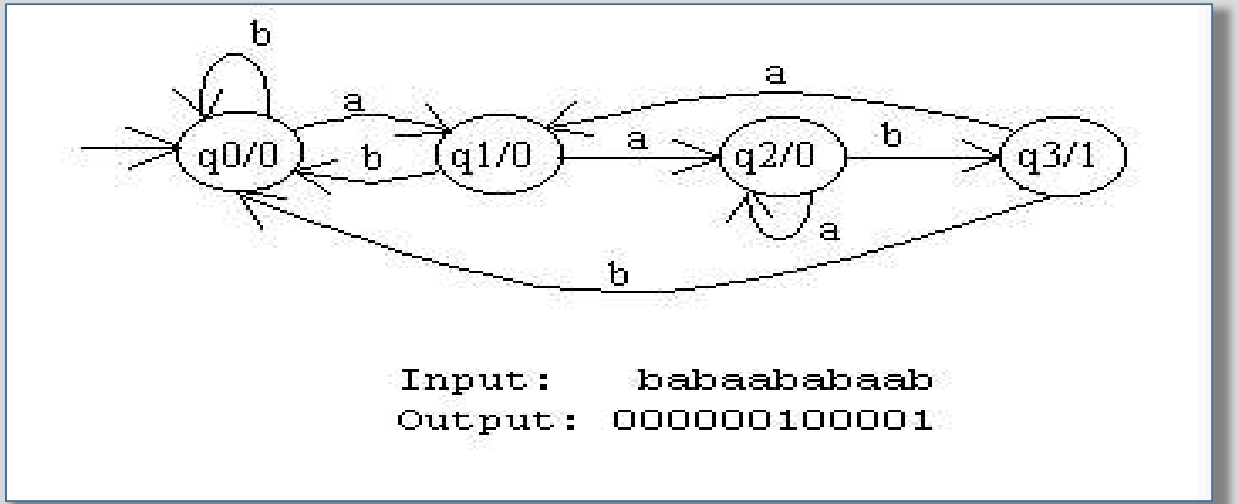
Moore Machine

- A **Moore machine** is a FSA with two extra attributes.
  1. It has TWO alphabets, an input and output alphabet.
  2. It has an output letter associated with each state.

  The machine writes the appropriate output letter as it enters each state.



Example - Moore Machine



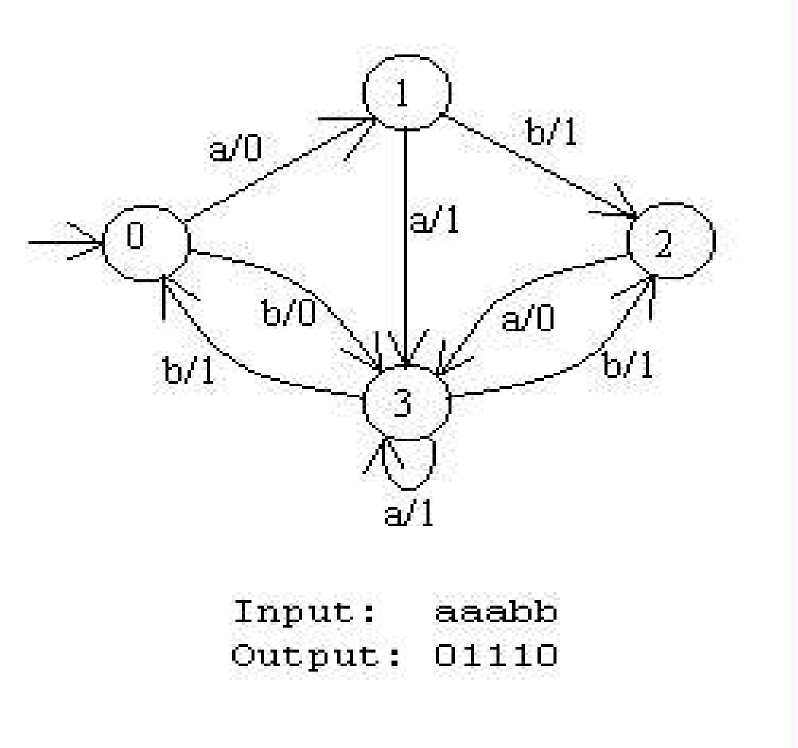
This machine might be considered as a "counting" machine.

The output produced by the machine contains a 1 for each occurrence of the substring **aab** found in the input string.

Mealy Machines

Mealy Machine

- **Mealy machines** are computationally equivalent to Moore machines
- However, Mealy machines move the output function from the state to the transition.
- This turns out to be easier to deal with in practice, making Mealy machines more practical.

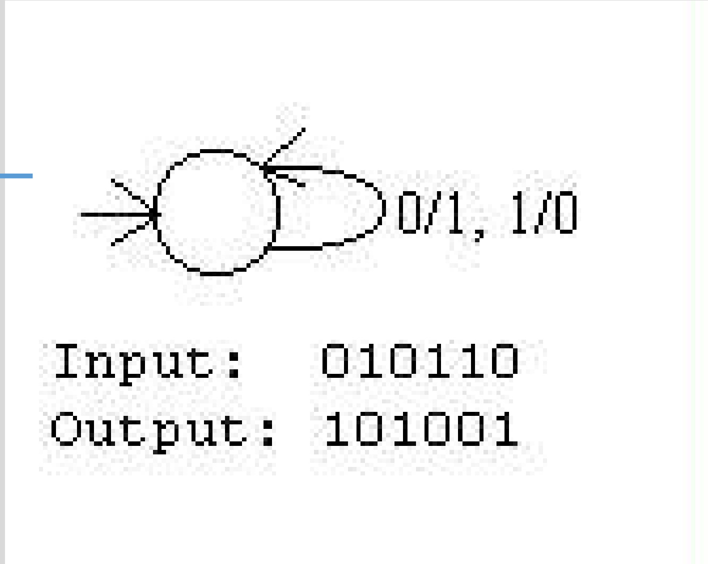


A Mealy Machine Produces Output on a Transition

Transitions are labelled *i/o* where

- ***i*** is a character in the input alphabet and
- ***o*** is a character in the output alphabet.

The following Mealy machine takes the one's complement of its binary input. In other words, it flips each digit from a 0 to a 1 or from a 1 to a 0.



A Mealy Machine Produces Output on a Transition

- Mealy machines are complete in the sense that there is a transition for each character in the input alphabet leaving every state.
- There are no accept states in a Mealy machine because it is not a language recogniser, it is an *output producer*.
- Its output will be the same length as its input.

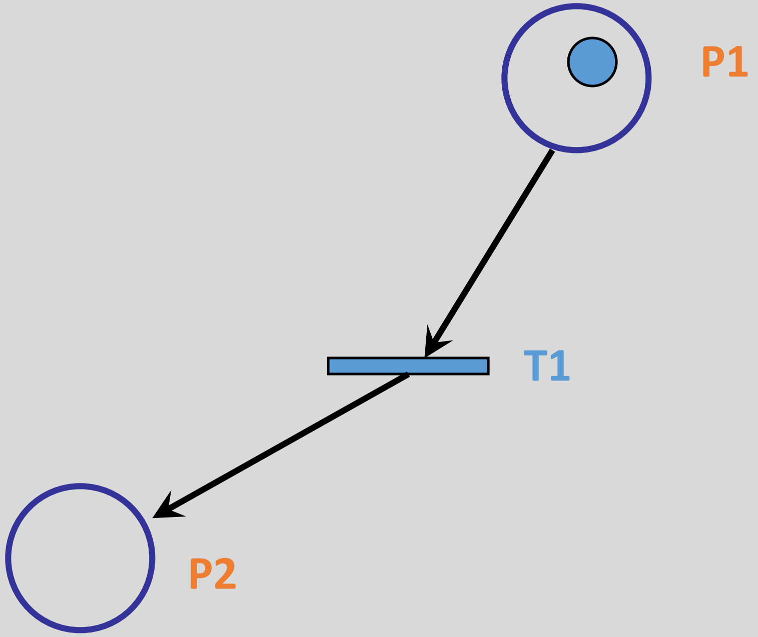
Petri Nets

Petri Net Models

- **Petri Nets** were developed originally by *Carl Adam Petri (when he was 13)*, and were the subject of his doctoral dissertation in 1962, at Technischen Hochschule Darmstadt.
- Originally created to model chemical reactions
- Extended, developed, and applied in a variety of areas.
- While the mathematical properties of Petri Nets are interesting and useful, the beginner will find that a good approach is to learn to model systems by constructing them graphically.

The Basics

- A **Petri Net** is a collection of <u>directed arcs</u> connecting <u>places</u> and <u>transitions</u>.
- Places may hold <u>tokens</u>.
- The <u>state or marking</u> of a net is its assignment of tokens to places.

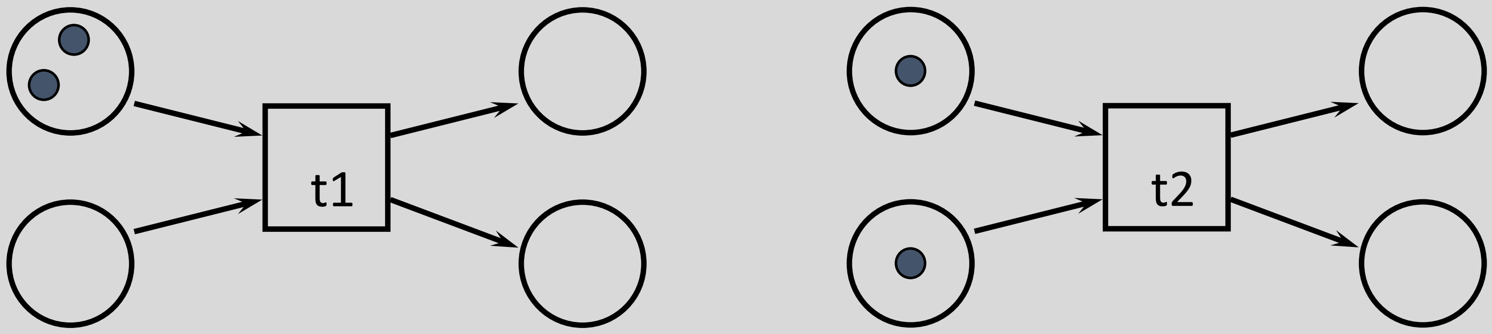


Capacity and Connections

- **Arcs have capacity 1 by default;** if otherwise, the capacity is marked on the arc, or use multiple arrows
- **Places have infinite capacity by default**.
- **Transitions have no capacity**, and cannot store tokens at all.
- **Arcs can only connect places and transitions**

Enabling Condition

- Transitions are the **active** components and places and tokens are **passive** components.
- A transition is enabled if each of the input places contains tokens, and the tokens are at least equal to the arc weight.

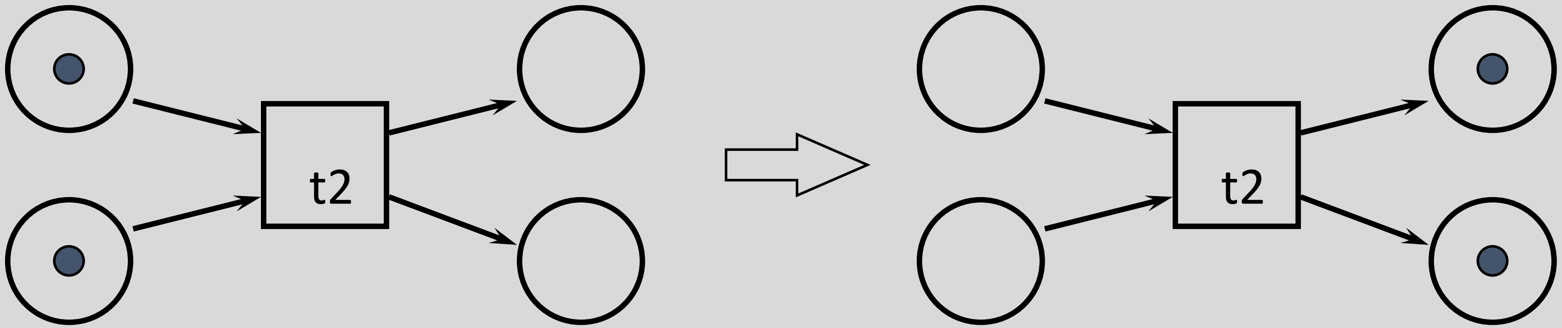


Transition t1 is not enabled, transition t2 is enabled.

Firing

An enabled transition may **fire**.

Firing corresponds to **consuming** tokens from the input places and **producing** tokens for the output places.

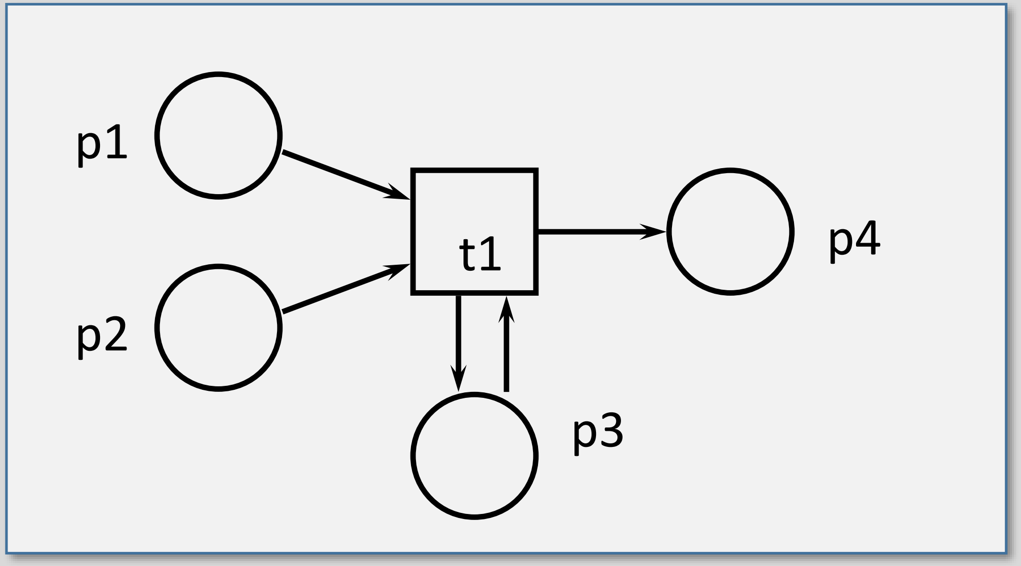


Firing is **atomic** (only one transition fires at a time, even if more than one is enabled)

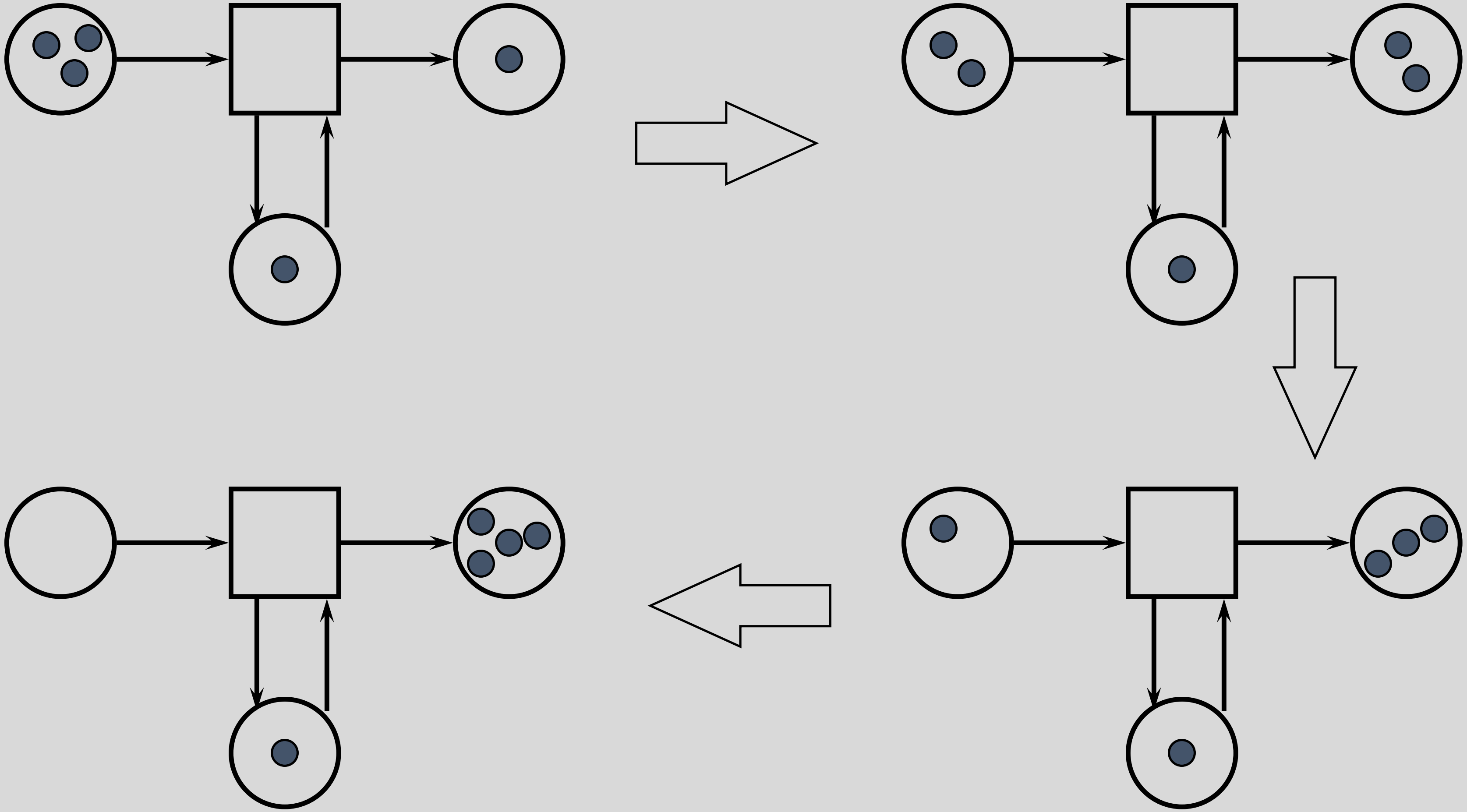
Transitions with Multiple Inputs and Outputs

Transition t1 has three **input places** (p1, p2 and p3) and two **output places** (p3 and p4).

Place p3 is both an input and an output place of t1.

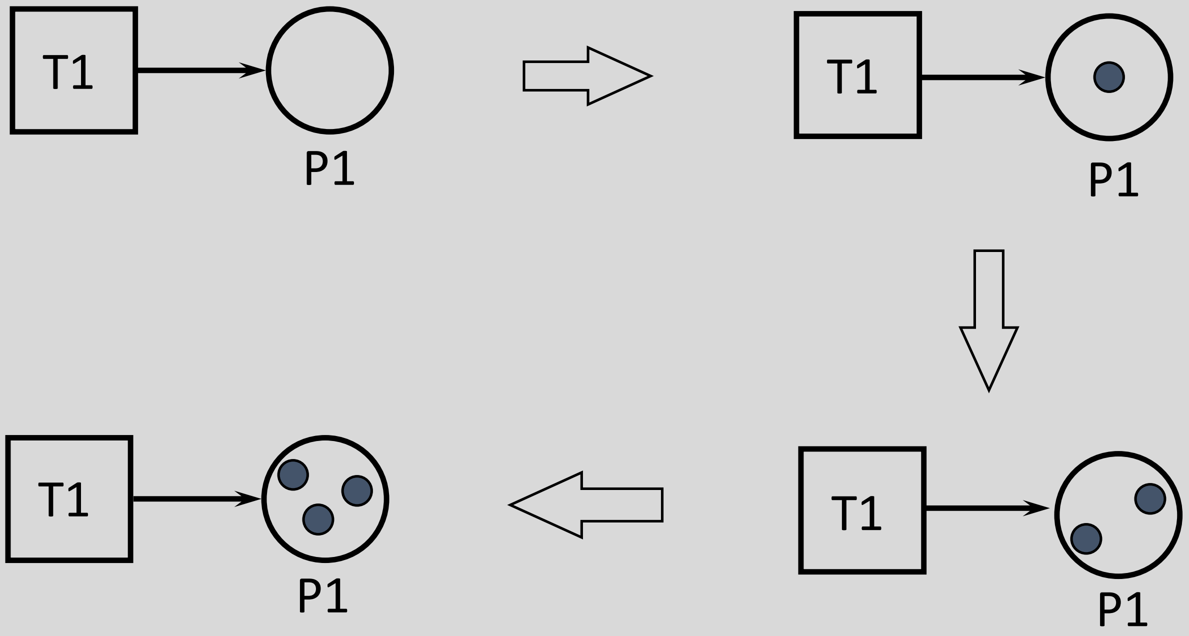


An Example Petri Net



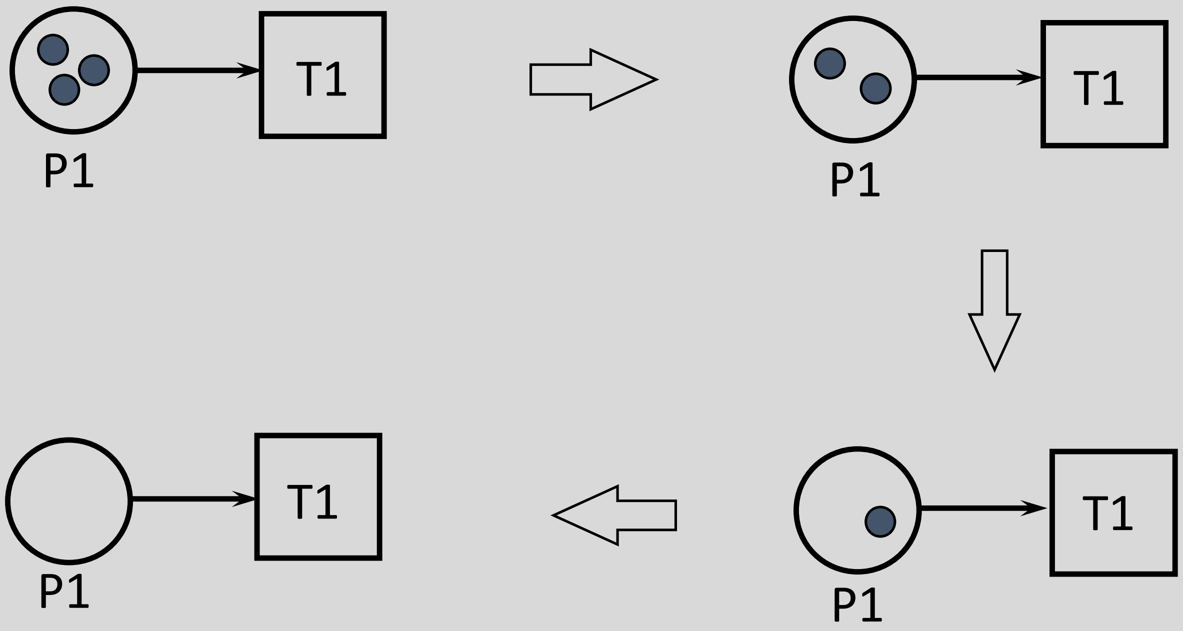
Creating/Consuming Tokens

A transition without any input can fire at any time and produces tokens in the connected places:



Creating/Consuming Tokens

A transition without any output must be enabled to fire and deletes (or consumes) the incoming token(s):

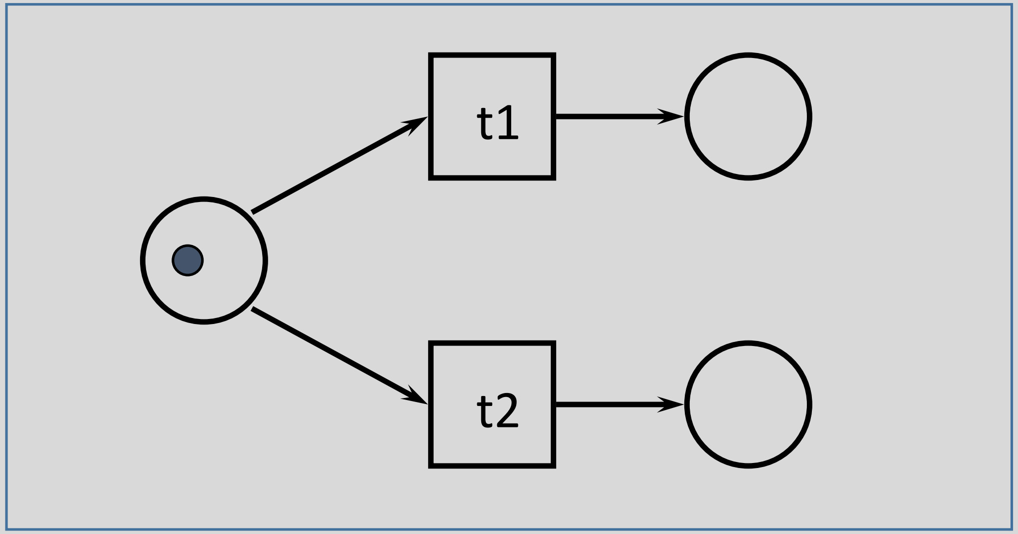


Non-Determinism in Petri Nets

Two transitions fight for the same token: **conflict**.

Even if there are two tokens, there is still a conflict.

The next transition to fire (t1 or t2) is arbitrary (non-deterministic).

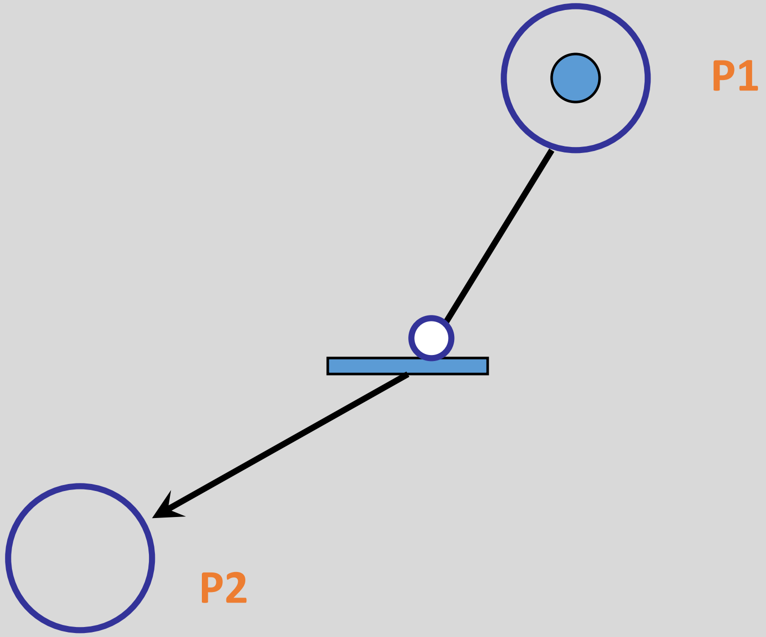


When Arcs have Different Weights…

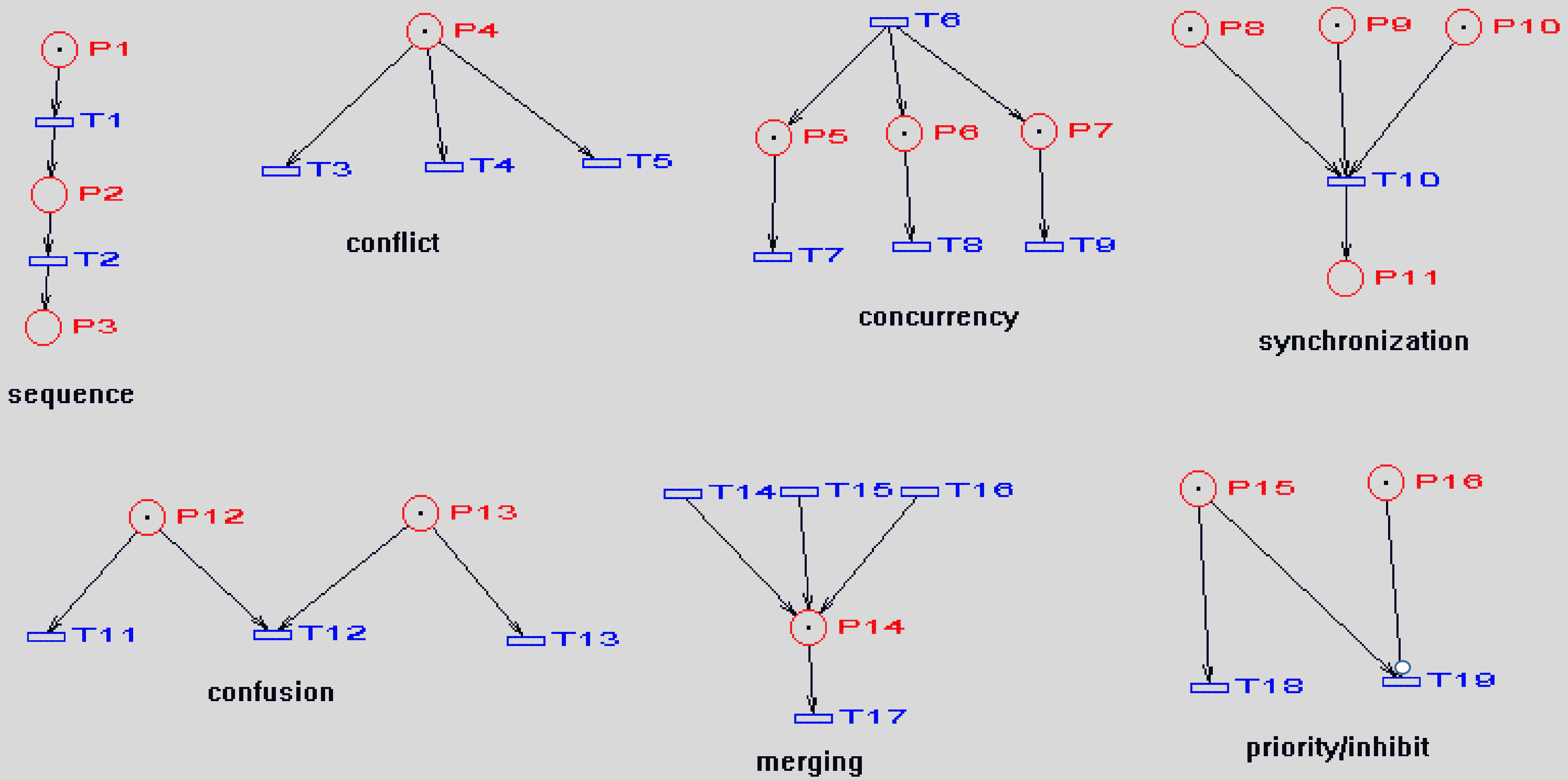
- When fired, the tokens in the input places are consumed, according to arc weights and place capacities
- New tokens are created in output places, according to arc weights and place capacities.
- This results in a new marking of the net

Inhibition

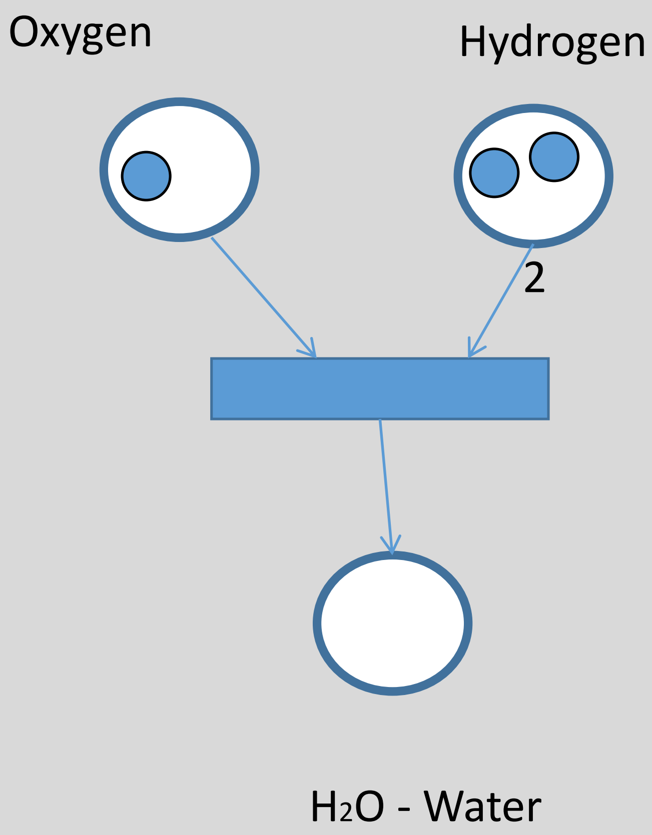
- Inhibition prevents a transition from firing



A Collection of Primitive Structures that Occur in Real Systems



How are they used in Chemistry?



Why use Petri Nets?

- Petri nets are **non-deterministic** and thus may be used to model discrete distributed systems.
  - There may be several arcs which can fire and we do not know in which order this will happen..
- There have been many variations and extensions of Petri nets to more effectively model a wider variety of systems.
- Since Petri nets have a rigorous mathematical notation, many questions about a system may be verified by studying properties of the Petri net.

Example Extension: High-Level Petri Nets

- The classical Petri net was invented by Carl Adam Petri in 1962.
- **High-level Petri nets** extend the classical Petri nets:
  - colour (for the modelling of attributes)
  - time (for performance analysis)
  - hierarchy (for the structuring of models, DFD's)

Recap

Recap

- Finite State Machines have states, transition functions, alphabets, and accepting state – Language acceptors
- Mealy and Moore machines add output to FSM – process input, rather than accept languages
- Mealy machines are more practical
- Petri nets have Arcs, Places and Transitions.
- Petri nets are **non-deterministic** and thus may be used to model discrete distributed systems.
- They have a well defined semantics and many variations and extensions of Petri nets exist.
- The state or marking of a net is an assignment of tokens to places.