Lecture 8 – System Models
Introduction

- Today we shall consider several system models:
  - Mealy machines
  - Moore machines
  - Petri Nets
  - UML digrams.
- Each of these can be used for specifying parts of the system and analysing properties.
- When you see a new model, ask yourself what type of properties that model could potentially represent.
Finite State Machines - Recap

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

What language is recognised by this FSA?
Finite State Machines - Recap

- Computation begins in the start state with an input string. It changes to new states depending on the transition function.
  - **states** define behaviour and may produce actions
  - **state transitions** are movement from one state to another
  - **rules or conditions** must be met to allow a state transition
  - **input events** are either externally or internally generated, which may possibly trigger rules and lead to state transitions.
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.
Finite State Machines with Output (Mealy and Moore Machines)

- **Finite state automata** are like computers in that they receive input and 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 Machine

- Basically a Moore machine is just 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.

```
Input: abab
Output: 10010
```
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 Machine

- **Mealy machines** are computationally equivalent to Moore machines
  - (we can implement any Mealy machine using a Moore machine, and vice versa).
- 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.

![Diagram of a Mealy Machine with inputs and outputs labeled.
Input: aaabb
Output: 01110]
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.

Input: 010110
Output: 101001
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 Net Models

- **Petri Nets** were developed originally by *Carl Adam Petri (when he was 12)*, and were the subject of his dissertation in 1962.
- Originally to model chemical reactions
- Since then, Petri Nets and their concepts have been 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 directed arcs connecting places and transitions.
- Places may hold tokens.
- The state or marking of a net is its assignment of tokens to places.

Place with token

P1

Arc with capacity 1

T1

Place

P2

Transition
Capacity

- **Arcs** have capacity 1 by default; if other than 1, the capacity is marked on the arc, or else we use multiple arrows to denote the capacity (i.e. 5 arrows for capacity 5).
- **Places** have infinite capacity by default.
- **Transitions** have no capacity, and cannot store tokens at all.

- Arcs can only connect places to transitions and vice versa.
- A few other features and considerations will be added as we need them.
Enabled Transitions and Firing

- A **transition is enabled** when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition.
- An enabled transition may fire at any time.
When Arcs have Different Weights...

- When fired, the tokens in the input places are moved to output places, according to arc weights and place capacities.
- This results in a new marking of the net, a state description of all places.
Inhibition

P1

P2
A Collection of Primitive Structures that Occur in Real Systems

- Sequence
- Conflict
- Concurrency
- Synchronization
- Confusion
- Merging
- Priority/inhibit
Re-cap, Petri-Nets invented for Chemistry

Oxygen

Hydrogen

2
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. We shall study them more next lecture.
Semantic Data Models

- Used to describe the logical structure of data processed by the system
- **Entity-relation-attribute model** sets out the entities in the system, the relationships between these entities and the entity attributes
- Widely used in database design. Can readily be implemented using relational databases
- No specific notation provided in the UML but objects and associations can be used
Software Design Semantic Model

Diagram:

- **Design**
  - name
  - description
  - C-date
  - M-date

- **Node**
  - name
  - type

- **Link**
  - name
  - type

- **Label**
  - name
  - text
  - icon

Relationships:
- has-nodes
- is-a
- has-labels
- has-links

COMP201 - Software Engineering
Data Dictionary Entries

- A **Data dictionaries** is a list of all of the names used in the system models.
- Descriptions of the **entities**, **relationships** and **attributes** are also included
- It may be used for **name-management** so that all names used in a system are consistent and do not clash
- It serves as a store of organisational information so that all data is stored in one location.
- On the next slide we shall see an example of a data dictionary.
## Data Dictionary Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>has-labels</td>
<td>1:N relation between entities of type Node or Link and entities of type Label.</td>
<td>Relation</td>
<td>5.10.1998</td>
</tr>
<tr>
<td>Label</td>
<td>Holds structured or unstructured information about nodes or links. Labels are represented by an icon (which can be a transparent box) and associated text.</td>
<td>Entity</td>
<td>8.12.1998</td>
</tr>
<tr>
<td>Link</td>
<td>A 1:1 relation between design entities represented as nodes. Links are typed and may be named.</td>
<td>Relation</td>
<td>8.12.1998</td>
</tr>
<tr>
<td>name (label)</td>
<td>Each label has a name which identifies the type of label. The name must be unique within the set of label types used in a design.</td>
<td>Attribute</td>
<td>8.12.1998</td>
</tr>
<tr>
<td>name (node)</td>
<td>Each node has a name which must be unique within a design. The name may be up to 64 characters long.</td>
<td>Attribute</td>
<td>15.11.1998</td>
</tr>
</tbody>
</table>
Object Models

- **Object models describe the system in terms of object classes**
- An **object class** is an abstraction over a set of objects with common attributes and the services (operations) provided by each object
- **Various object models may be produced**
  - Inheritance models
  - Aggregation models
  - Interaction models
Object Models

• Natural ways of reflecting the real-world entities manipulated by the system
• More abstract entities are more difficult to model using this approach
• Object class identification is recognised as a difficult process requiring a deep understanding of the application domain
• Object classes reflecting domain entities are reusable across systems
The Unified Modelling Language

- Devised by the developers of widely used object-oriented analysis and design methods
- Has become an effective standard for **object-oriented modelling**
- We will study UML in more detail in later lectures
  - For now, note that UML is an expressive, easy to use, unambiguous and widely used modeling language which is well supported by a variety of suitable tools (both proprietary and open source CASE tools).
The unified modeling language contains many different types of diagram we may use to model systems, for example:

- Use-case diagrams
- Class diagrams
- Sequence diagrams
- Statechart diagrams
- Deployment diagrams
- Etc.
The Unified Modeling Language

- Notation for UML Class Diagrams:
  - Object classes are rectangles with the **name** at the top, **attributes** in the middle section and **operations** in the bottom section
  - Relationships between object classes (known as associations) are shown as lines linking objects
  - Inheritance is referred to as generalisation and is shown ‘upwards’ rather than ‘downwards’ in a hierarchy
Lecture Key Points

- **Semantic data models** describe the logical structure of data which is imported to or exported by the systems.
- **Object models** describe logical system entities, their classification and aggregation.
- **Sequence models** show the interactions between actors and the system objects that they use.
- **Structured methods** provide a framework for developing system models.
- **Next lecture**: an in-depth look at Petri nets.