VII. Specifications (I)
Objectives

- To discuss the importance of models for the definition of specifications
- To show formalisms that can be used to define system models
- To discuss properties and compare such formalisms
- To provide examples
Topics

- To distinguish among formal and informal specifications
- Operational vs. Descriptive Models
- Introduce some operational model:
  - Data Flow Diagrams (DFD)
  - Finite State Machines (FSM)
  - Petri Nets (PN)

The material that will be discussed today and tomorrow is covered by [GJM] Chapter 5 and/or [Som] Chapters 8,10
How to increase Specification Qualities

- Spec should be: **Clear, Unambiguous, Understandable**
- Should not be: Contradictory or **Inconsistent, Incomplete**

- Models are description of the system abstracting away unimportant details, so to make the system “tractable”

- **Formal** specifications vs. **Informal** specifications
  - Use of formalisms make spec precise and augment automatic verification possibilities
  - Informal spec more flexible, leave more decision space to the implementer

- **Semiformal** often we use notation which semantics has not been defined so precisely (e.g. UML)
Formal Spec and Verification

- Formal specs are a powerful tool for **making easier many development phases** in particular for analysis and verification purpose.

- Just some keyword:
  - Model Checking
  - Model Based Testing
  - Simulation and prototyping
Operational vs. Descriptive Specification

**Operational specification** describe the system in term of the expected behaviour generally providing a model.

**Descriptive specifications** describe the system in term of desired properties for the system.

An example from mathematics:
- Take a string of length “r” and fix at one of its extreme a pencil. Then fix the other extreme to a sheet using a pin. Now tightening the string move the pencil over the sheet describing a circle.
- $x^2 + y^2 = r^2$
- **which one is a descriptive or an operational specification?**
Operational Specifications – DFD

Data Flow Diagrams:
- Used to specify functions of a system and how data flow from functions to functions

Systems are seen as collections of functions that manipulate data
- Data can be stored in repositories
- Data can flow
- Data can be transferred from and to the external environment
DFD – graphical notation

**DFD legenda**

- **O** function symbol
- **□** input device symbol
- **→** data flow symbol
- **□** output device symbol
- **____** data store symbol

**Example – a simple arithmetical function:**

\[(a+b) \times (a+c \times d)\]
DFD – a supermarket example (I)

- **Prodotti Acquistati**
  - Codice a barre prodotto
  - Gestione acquisti clienti
  - Dati prodotto

- **Carta Fidelizzazione**
  - Codice cliente
  - Gestione acquisti clienti
  - Totale

- **Data Base informazioni prodotti**
  - Dettagli scontrino

- **Data Base informazioni clienti**
  - Punti

- **Gestione vendite ed incassi Giomalieri**
  - Incassi
  - Dati contabili
  - Libri contabili

- **Stampa totale su supporto cartaceo**
  - Data Base incassi
The functionality “gestione acquisti cliente” is certainly a macro activity that could be split.

In case of supermarket wants to apply discounts for some products, in a certain period, to some clients, how these information could be introduced in the system?
DFD Characteristics

- DFD is certainly a **semi-formal** notation

- **Lack of precise semantics:**
  - Function definition can be defined more precisely using more formal description
  - No concept of **control** in such kind of diagrams
    - Diagrams do not specify how data are used and output are produced

- **Synchronization** between functions is not specified

![Diagram](image)

When the function D should start?
How to overcome DFD weakness

- Use of **complementary notation** to express those aspects that are not adequately described by DFD
- Augment the DFD introducing **mechanisms that allow to express the missing aspects**
- **Revise the traditional definition** of DFD to make it fully formal
A Finite State Machine (FSM) is an abstract automaton that permits to describe the **control flow** of a system.

Mathematically FSM are defined by:
- A finite set of states $Q$
- A finite set of input $I$
- A transition function $\delta: Q \times I \rightarrow Q$

A simple switch can be described by the following FSM:
- $Q = \{\text{On}, \text{Off}\}$
- $I = \{\text{Switch}\}$
- $\delta = \{ (\text{On},\text{Switch}) \rightarrow \text{Off},
             (\text{Off},\text{Switch}) \rightarrow \text{On} \}$
A simple example of an electrical equipment including a safety mechanism:

\[ Q = \{q_0, q_1, q_2\} \]
\[ I = \{On, Off, Unlock\} \]
\[ \delta = [(q_0, On) \rightarrow q_1, (q_1, Unlock) \rightarrow q_2, (q_2, Off) \rightarrow q_0] \]
A Finite State Machine (FSM) extension permits to distinguish among input and output data.

Mathematically I/O FSM are defined by:
- A finite set of states $Q$
- A finite set of input $I$
- A finite set of output $O$
- A transition function $\delta: Q \times I \rightarrow Q \times O$

How FSM can be composed originating a system FSM?
- Resulting machine have a number of state given by the product of the number of states for each composing machine.
- We take a simple (and a bit simplistic) approach.
FSM – Thinking Philosophers (I)

Filosofo A

\[ q_0 \rightarrow q_1 \rightarrow q_2 \]

\[ \text{PrendiBastoncino1dx} \]

\[ \text{PrendiBastoncino2sx} \]

Rilascia

Filosofo B

\[ q_0 \rightarrow q_1 \rightarrow q_2 \]

\[ \text{PrendiBastoncino2dx} \]

\[ \text{PrendiBastoncino1sx} \]

Rilascia

Risorse

\[ q_0 \rightarrow q_1 \]

\[ \text{PrendiBastoncino1dx} \]

\[ \text{PrendiBastoncino2dx} \]

Rilascia

\[ \text{PrendiBastoncino1sx} \]

\[ \text{PrendiBastoncino2sx} \]

Rilascia
FSM and continuous systems

- FSM can store a finite quantity of information (**Finite-memory devices**)
- FSM can be cumbersome to describe even “finite” systems requiring to formally express details that sometimes are easier to understand using natural language
- In practice computer always have a finite memory but number of states is unmanageably large.

How to manage these situations:
- Ignore details
- Complement diagrams with natural language comments
- Change model (LTS, PN, ...)
- Enrich FSM (i.e. Introduce a language to annotate transitions)
FSM for a Producer/Consumer System

PRODUCER
produce

p1
write
p2

CONSUMER
read

c1
consume
c2

BUFFER
write
write

0
read
1
read
2
The Buffered Producer/Consumer System
FSM limitations

- Composing machines the **number of states raise drastically**
- We could leave the FSM as they are defined for the subsystems and use composition rules.
- Still some problems persist
  - The system must be **always in a unique state and can perform only one action at any instant of time**
- FSM permit to express only **synchronous interactions**
  - Not really adequate to describe general concurrent systems
FSM Exercises

- FSM for a two switch lighting systems

- Two lamps and one button
  - Different alternatives to model the same system
Key points

On today lesson we introduced and discussed:

- Models to describe software systems have been introduced
- Properties that permit to classify different modelling language
  - Operational vs. Descriptive
  - Formal vs. Semiformal vs. Informal
- Data Flow Diagram (DFD)
- Finite State Machine (FSM)