

# Documentation

## Notations and tools

---

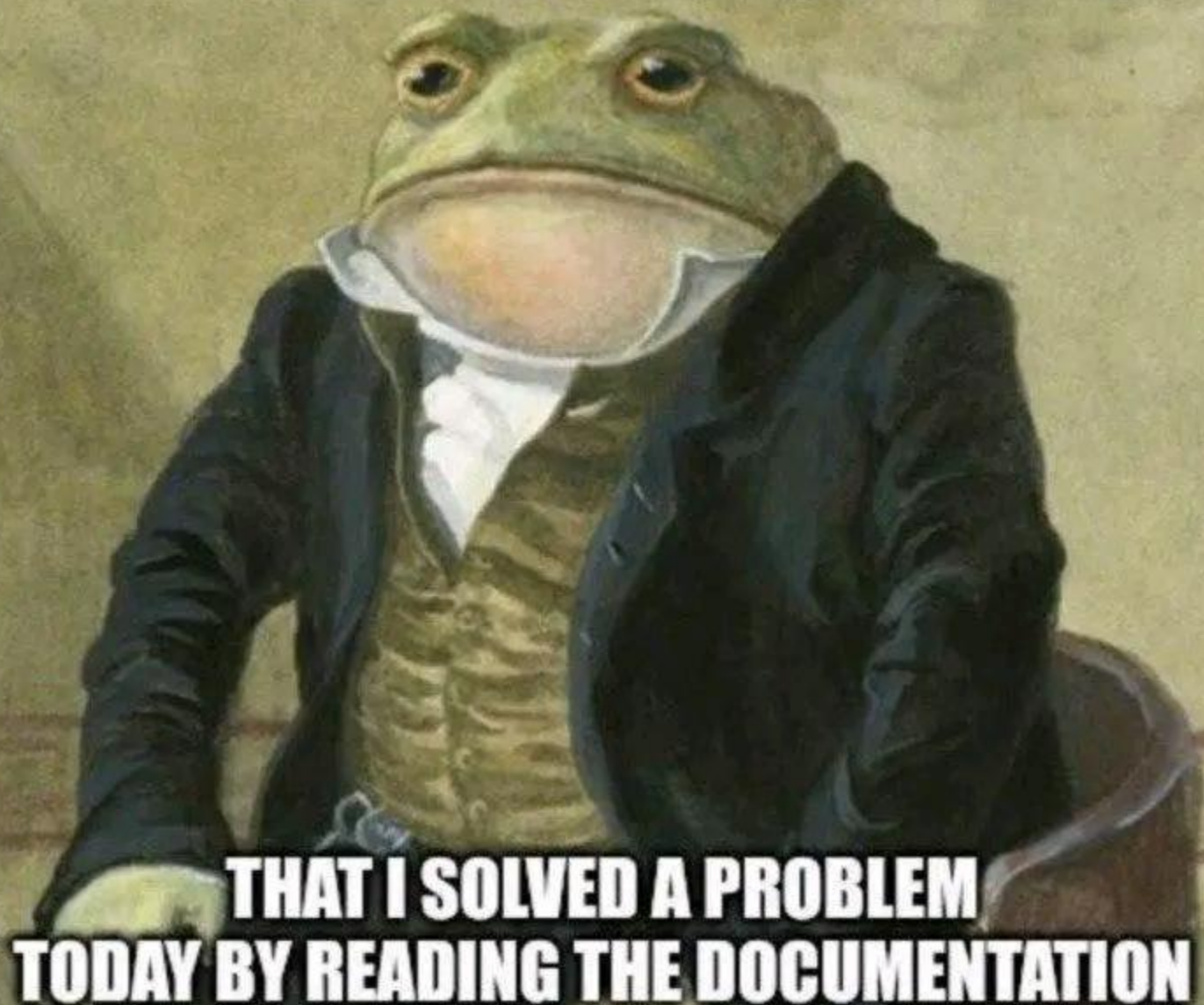
Paolo Burgio  
paolo.burgio@unimore.it



**UNIMORE**  
UNIVERSITÀ DEGLI STUDI DI  
MODENA E REGGIO EMILIA

High Performance  
Real Time **Lab**

**GENTLEMAN, IT IS WITH  
GREAT PLEASURE THAT I INFORM YOU**



**THAT I SOLVED A PROBLEM  
TODAY BY READING THE DOCUMENTATION**



# Tools and diagrams

---

Specifications are a **contract** between us and the customer (cit.)

- › We use well-known tools and models

We typically specify/distinguish among:

- › Operational diagrams
  - Data flow, UML, models such as FSMs, and Petri Nets
- › Descriptive/structural diagrams
  - Entity-Relationship (inspired by DB entities analysis and design)

UML (standard) diagrams

- › Structural diagrams
  - Use-cases/scenarios
  - Notations for classes/objects/packages/components – From OOP
- › Behavioral diagrams
  - Sequence diagrams
  - State diagrams
  - Activity diagrams



Sorry but... I cannot explain them in this order

We start from specifications, then system design, then implementation

UML has dedicated slide decks



We typically specify/distinguish among:

- › Operational diagrams
  - Data flow, UML, models such as FSMs, and Petri Nets
- › Descriptive/structural diagrams
  - Entity-Relationship (inspired by DB entities analysis and design)

UML (standard) diagrams

- › Structural diagrams
  - Use-cases/scenarios
  - Notations for classes/objects/packages/components – From OOP
- › Behavioral diagrams
  - Sequence diagrams
  - State diagrams
  - Activity diagrams



# Data Flow Diagram (DFD)

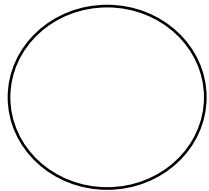


# Data Flow Diagrams (DFDs)

---

Describe functionalities (nodes) and data arcs, both input and output

- › I show them in B/W, but the recommendation is “play” with shapes to be more “communicative”
- › One color per functionality
- › Lines can also be dotted/bold(er) etc



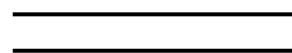
Functionality



Data flow



Input



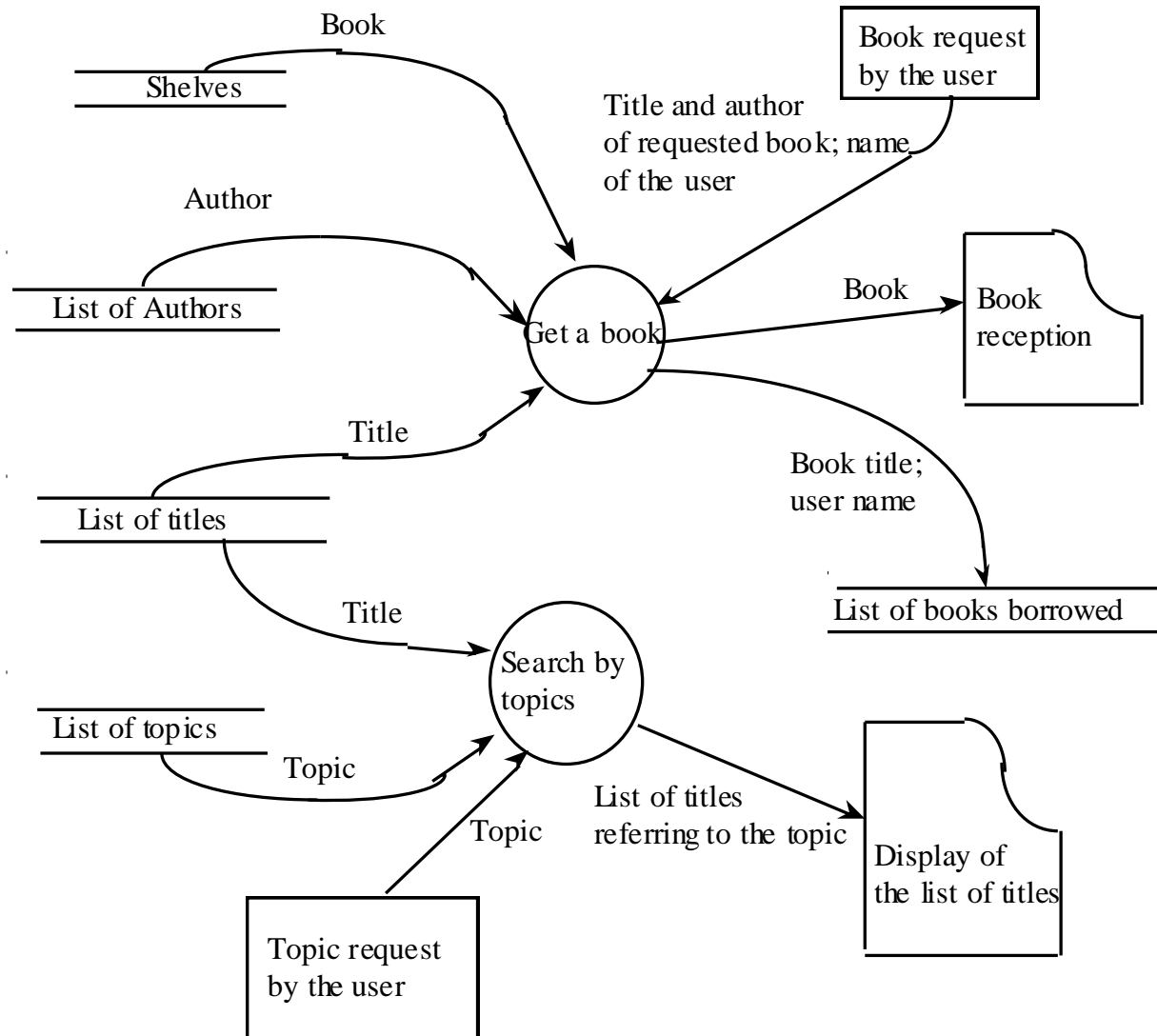
Storage/archive



Output

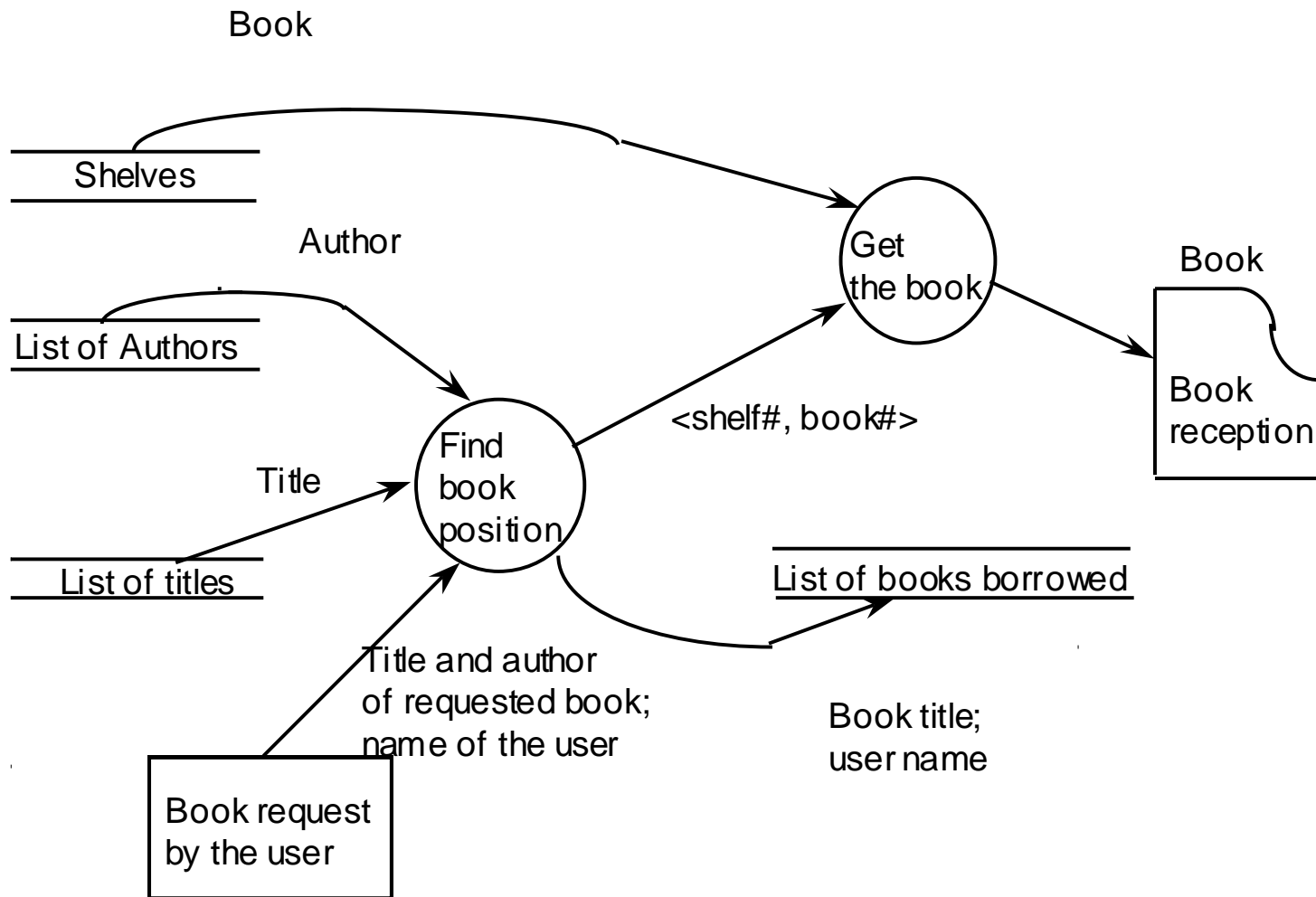


# Example of DFD





# Example of DFD (cont'd)









# DFD are not standardized

---

**Pros:** they are extremely simple, and everyone uses them

**Cons:**

- › Informal, not standardized
- › I typically use a variant with additional symbols
- › They are not operational: they cannot, specify "control flows (if, or, switch,...)



# Unified Modeling Language (UML)



# UML

---

See the dedicated slide decks

- › 05 - Unified Modeling Language.pptx





# Finite state machines



# Modeling stateful systems: an example

---

E.g., an elevator, reacts to multiple events

- › Typically in idle state
- › If you are press the button, the door opens
- › You select the floor, doors close
- › Then, it reaches the floor (feat. velocity control)
- › Then, it opens the door, which subsequently closes after X seconds

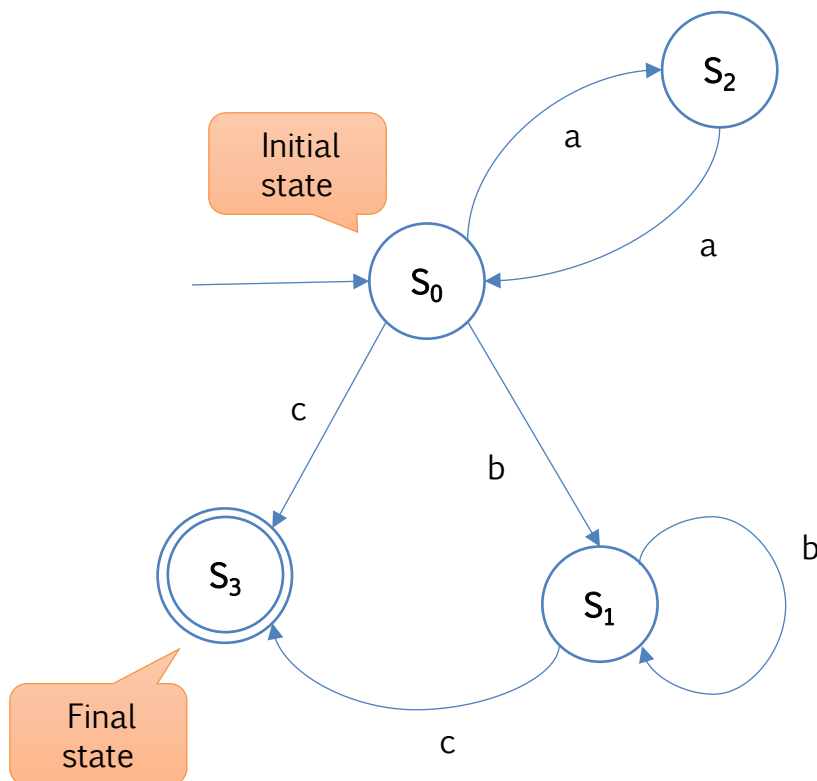
This behavior is controlled by a **finite state automations/machine**



# Finite State Machines/Automata

## Example problem

- › Identify even sequences of  $a$  (even empty), followed by one, or more, or no,  $b$ , ended by  $c$



Given an alphabet that models a set of inputs

And validation rules for producing the sequence (aka: words)

Define FSA as

- ›  $S$ : a non-empty states set
- ›  $s_0 \in S$ : initial state
- ›  $S_f \subseteq S$ : final states set
- ›  $t: S \times V \rightarrow S$ : states transaction func



# Exercise

Let's  
code!

Implement the FSM that understands whether a word has the following form

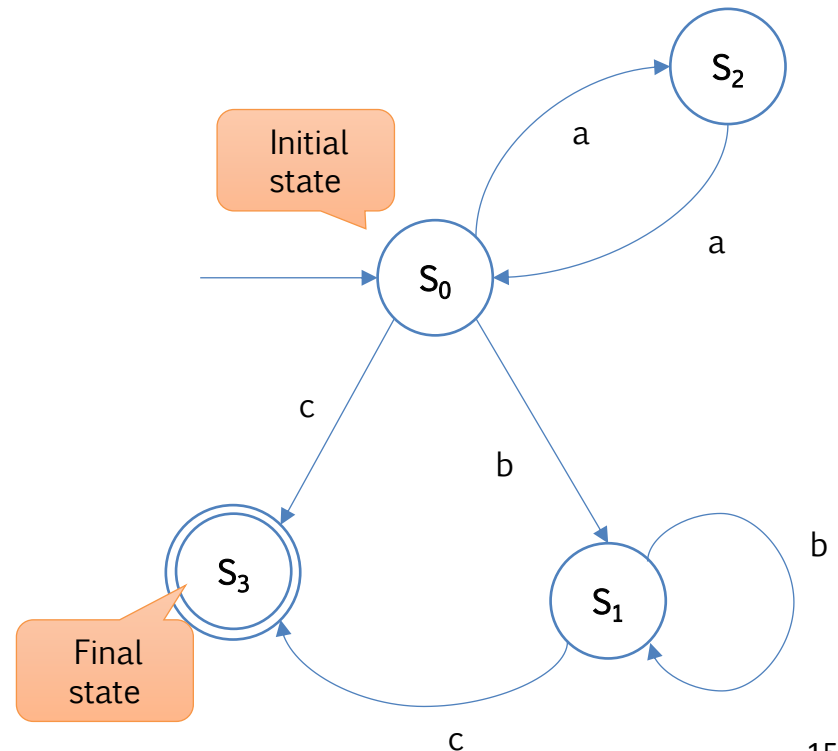
*"Identify even sequences of a (even empty),  
followed by one, or more, or no, b, ended by c"*

Use the language that you want

- › You just need IFs, CASE-SWITCH, recursion, tables
- › Receive the target word from stdin
- › Hint: start simple...

What's missing?

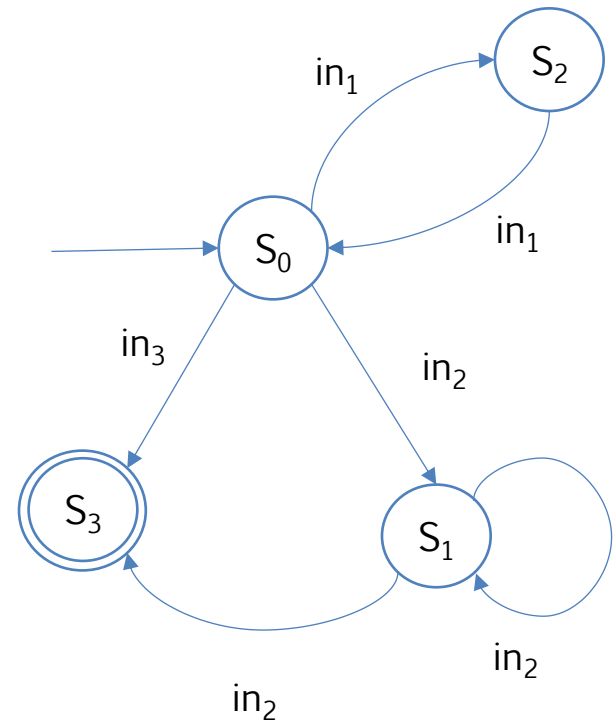
- › In case of error => default error state
- › Typically implicit in state diagrams





# A generic FSM

- › Till now, we only saw machines that can recognize a **word** from a language
  - I say “word”, you might want to understand “sentence”
- › Let's now see how a machine can actually **produce** an output







# The Machine of Mealy

- › When crossing an edge, produce an output

**$\langle I, O, S, mfn, sfn \rangle$**

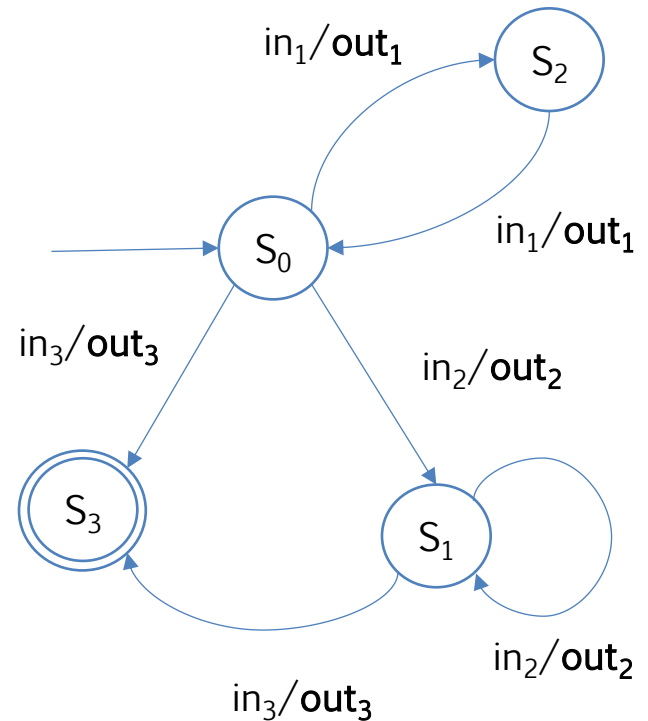
$I$  : (finite) set of Input symbols

$O$  : (finite) set of output symbols

$S$  : (finite) set of states ( $s_0$  initial state)

$mfn: I \times S \rightarrow O$  machine/output function

$sfn: I \times S \rightarrow S$  state transition function





# The Machine of Moore

- › When in a state an edge, produce an output

**$\langle I, O, S, mfn, sfn \rangle$**

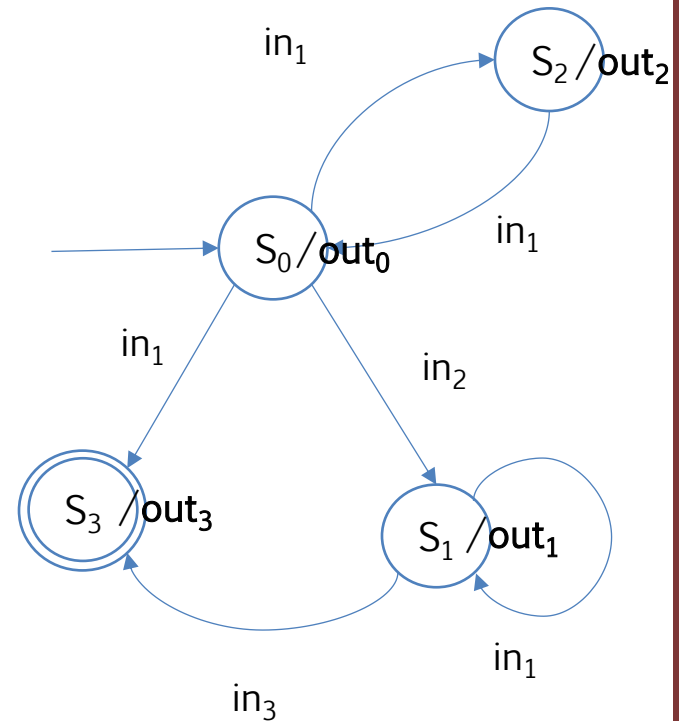
$I$  : (finite) set of Input symbols

$O$  : (finite) set of output symbols

$S$  : (finite) set of states ( $s_0$  initial state)

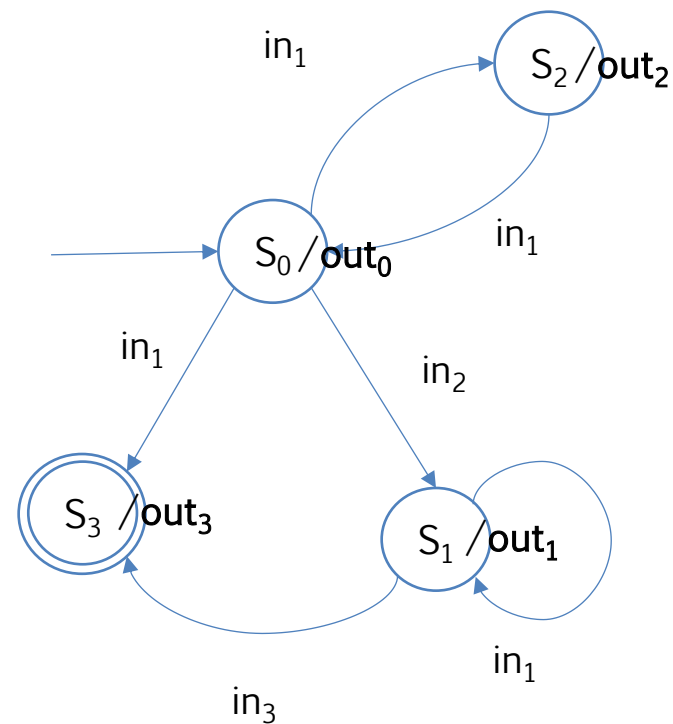
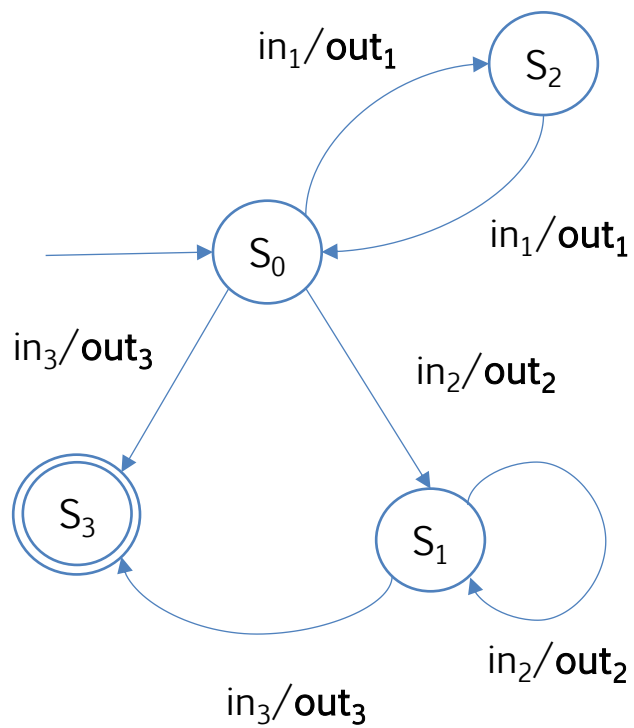
$mfn: S \rightarrow O$  machine/output function

$sfn: I \times S \rightarrow S$  state transition function





# What's the difference?





# What's the difference?

---

Mathematically equivalent

- › One can be transformed in another

..but..

- › Mealy can potentially have different outs, to different inputs/transitions
  - Less states, if output depends on inputs one can add an edge to the machine
- › Moore potentially keeps the output stable for all the state
  - Moore requires more states, in case out depends on input and not only on state



# Exercise

Let's  
code!

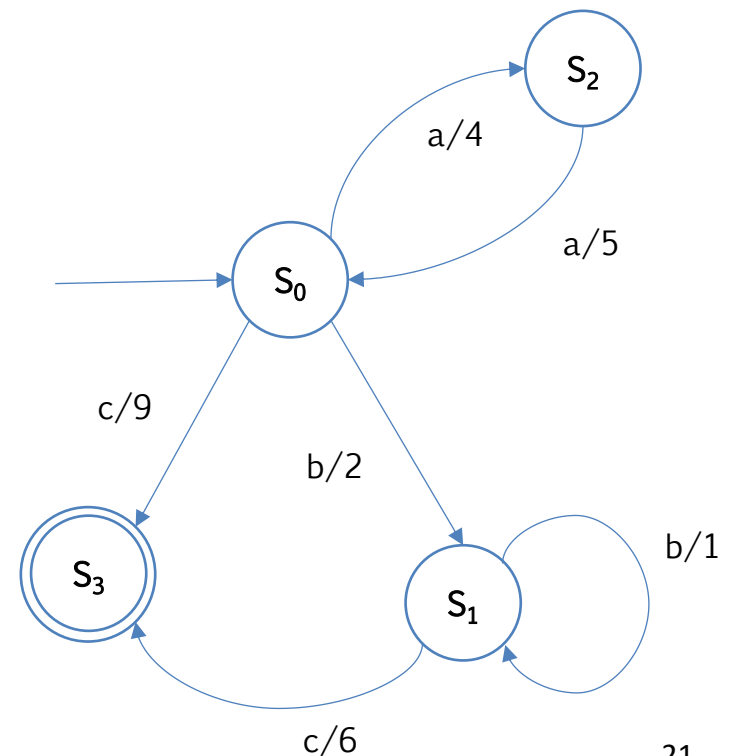
- › Implement the automata that understands whether a words is from L

*"Identify even sequences of a (even empty),  
followed by one, or more, or no, b, ended by c"*

- › ..and writes the corresponding number  
(I choose them randomly)
- › Mealy? Moore? You choose
  - Here, I show Mealy

Hint

- › If not already done, use tables  
for state/output transactions



# More formalism

**$\langle I, O, S, \text{mfn}, \text{sfn} \rangle$**

$I$  : (finite) set of Input symbols

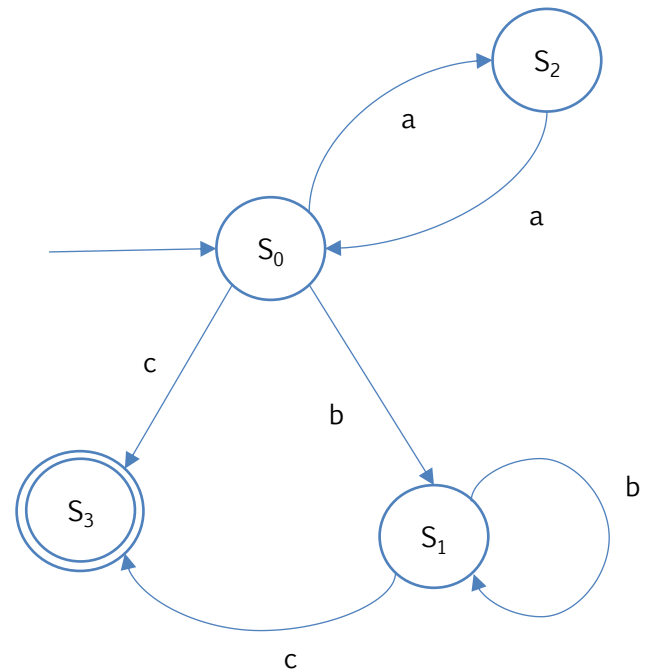
$O$  : (finite) set of output symbols

$S$  : (finite) set of states

$\text{mfn}: I \times S \rightarrow O$  machine function

$\text{sfn}: I \times S \rightarrow S$  state function

- › Partly already seen
- › Has memory
- › Memory is a limitation





# Automate..the automata production process

---

At the end of the day, we just need to model the grammar, and then!



Several tools to support the design

- › Matlab Stateflow, UML

$I$  : (finite) set of Input symbols

$O$  : (finite) set of output symbols

$S$  : (finite) set of states

$mfn: I \times S \rightarrow O$  machine function

$sfn: I \times S \rightarrow S$  state function

Several grammar interpreters to rely the burden of writing FSM code

- › FSF's GNU Bison – Included in GCC
- › YACC – Yet Another Compiler-Compiler



# FSMs/Automata and UML

---

See the dedicated slide decks

- › 05 - Unified Modeling Language.pptx



# References

---



## Course website

- › <http://hipert.unimore.it/people/paolob/pub/ProgSW/index.html>
- › [http://hipert.unimore.it/people/paolob/pub/Industrial\\_Informatics/index.html](http://hipert.unimore.it/people/paolob/pub/Industrial_Informatics/index.html)

## Book

- › I. Sommerville, "Introduzione all ingegneria del software moderna", Pearson
  - Chapter 3
- › Alessandro Fantechi, «Informatica Industriale», Città Studi Edizioni

## My contacts

- › [paolo.burgio@unimore.it](mailto:paolo.burgio@unimore.it)
- › <http://hipert.mat.unimore.it/people/paolob/>
- › <https://github.com/pburgio>