Unité d'Enseignement :

Concours Programmation

Gilles Menez

Université de Nice – Sophia-Antipolis
Département d'Informatique
email : menez@unice.fr
www : www.i3s.unice.fr/˜menez

12 novembre 2018: V 1.1

Finite State Machines

State Machines

Most of the time, State Machines are introduced when students learn **theoretical fundations of software**.

As a model of computations (i.e which describes how a set of outputs are computed given a set of inputs) their graphical representations (of course we do not forget the underneath mathematical model) is often used to illustrate for example :

- the evolution of the Turing machine,
- or if an input is accepted or rejected by the grammar of a formal language.
- ...

This close connexion to the theoretical could explain why, as these courses finish, the "state machine" model will be ASAP buried.

Such a useful framework !

State machines are such a general formalism that a HUGE class of discrete-time system can be described as states machines.

- They are **useful frameworks for modeling/implementing systems**.

Every day life systems will use them :

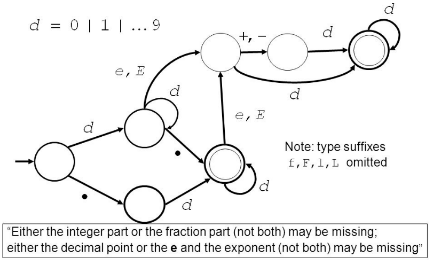
- Embedded systems (washing, espresso, *.* ... machines),
- GUI software,
- Processes guidances (included protocols or OS),
- Complex systems operations (lifts, spacecrafts, ...),
- ...

I do love the citation of Knuth (LaTeXfather) (about operating algorithm of campus lift) in the conclusion (page 165) of :

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011/unit-1-software-engineering/state-machines/MIT6_01SCS11_chap04.pdf

Finite Automata

Finite Automata (FA) is the simplest machine to **recognize patterns**.



State Diagram of the FSA :
"Floats in C language"

A FA consists of the following :

- $\mathcal{Q}$: finite set of states.
- Σ : set of input symbols.
- q : initial state (q is a member of $\mathcal{Q}$) .
- F : set of final states (F is a subset of $\mathcal{Q}$).
- δ : transition function.

$$\delta : Qx\Sigma \longrightarrow Q$$

Formal specification of this machine $\mathcal{M}$ is $\boxed{\mathcal{Q}, \Sigma, q, F, \delta}$

Input word : An automata reads a finite string of symbols $a1, a2, \ldots, an$, where $a_i \in \Sigma$, which is called an input word.

- ✔ The set of all words is denoted by Σ^*.

"Accept or Reject" : This is "the aim" of a Finite State Automata described machine.

Accepting word : A word $w \in \Sigma^*$ is accepted by the automaton if $qn \in F$ (equiv. The final state deduced from transitions conducted by inputs/symbols of the word w is in F)

We stop there because probably theses slides should be a remake and you should know from L1/L2/L3 :

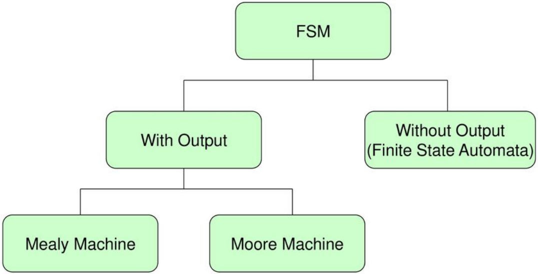
① Deterministic Finite Automata,

② Non Deterministic Finite Automata,
Several possible transitions from a state for a same input.

③ Language recognized by $\mathcal{M}$, ...

FSM with Outputs

In a theoretical context, FSA are "just" accepting or rejecting.



But some machines, described by FSA, have outputs :

- They are FSM : **Moore Machines** or **Mealy Machines**

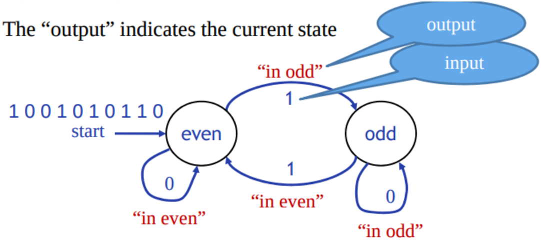
They are **intensively used to control (all kind) of processes**, . . . washing machines, espresso machines, . . .

Mealy Machine

A Mealy machine is a **deterministic finite-state transducer** :

- "deterministic" : for each state and input, at most one transition is possible.
- "transducer" : inputs will imply outputs.

Example : The following Mealy machine "determines whether the number of 1s is even or odd, for a given binary number."



As you see, in mealy machines **outputs are on transitions** !

A Mealy Machine consists of the following :

- $\mathcal{Q}$: finite set of states.
- Σ : set of input symbols (e.g. input alphabet).
- q : initial state (q is a member of $\mathcal{Q}$) .
- δ : transition Function.

$$\delta : Qx\Sigma \longrightarrow Q$$

- Γ : **set of output symbols** (e.g output alphabet)
- ω : **output function**

$$\omega : Qx\Sigma \longrightarrow \Gamma$$

Very similar to a Finite Automaton (FA), **with a few key differences** :

- It has **no final states**.
- Its transitions **produce output** :

  It does not accept or reject input, **instead, it generates output from input**.
- Lastly, Mealy machines **cannot have** nondeterministic states.

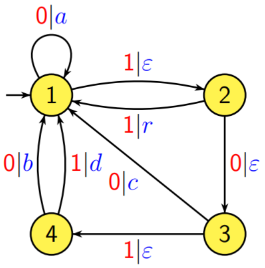
UnCiphering/Decoding machine

Mealy machines provide a rudimentary mathematical model for "toy" cipher machines :

Considering the "input Latin alphabet" and "output $\{0, 1\}$ alphabet", a Mealy machine can be designed that given a string of letters (a sequence of inputs) **it can process it into a ciphered bit string** (a sequence of outputs) :

010101101000101101010110

Propose this input sequence to this "unciphering" machine and tell the input !



➢ Which string (in the Latin alphabet) is this ?

- ✖ Inputs are in red color.
- ✖ Outputs are in blue color (ϵ is empty/no output).

Prefix code

Can you deduce the ciphering code (/prefix code) ?

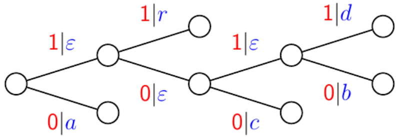
"Prefix code" : a set C of nonempty words is a prefix code if no word of C is a proper prefix of another word in C (i.e there is no whole code word in C that is a prefix (initial segment) of any other code word in C.).

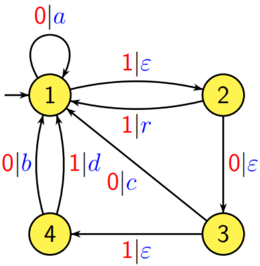
- For example, a code with code words 9, 55 has the prefix property ; a code consisting of 9, 5, 59, 55 does not, because "5" is a prefix of "59" and also of "55"
- aka, Huffman, Shannon-Fano, . . .

Consider the coding

$a \rightarrow 0 \quad b \rightarrow 1010 \quad c \rightarrow 100 \quad d \rightarrow 1011 \quad r \rightarrow 11$

Decoding function





Construction of a FSM decoder for a code γ

The decoder for the encoding γ is built as follows :

- ✔ Take **a state for each proper prefix** of some codeword : **1**[1], **10**[0], **101**[0,1] for state "2","3","4".
- ✔ The state corresponding to the empty word ϵ is the **initial and the terminal state** : state "1".
- ✔ There is an edge $p \xrightarrow{a|\epsilon} pa$ for each prefix p and letter a, such that pa is (again/still) a prefix
- ✔ There is an edge $p \xrightarrow{a|b} \epsilon$ for each p and letter a with pa $= \gamma(b)$.

Properties :

- ➢ When the code is prefix, the decoder is deterministic.
- ➢ In the general case, unique decipherability is reflected by the fact that the transducer is unambiguous.

Deterministic / Unambiguous

An automaton is deterministic

- if it has a unique initial state and
- if, for each state p and each letter a, there is at most one edge starting in p and labeled with a.

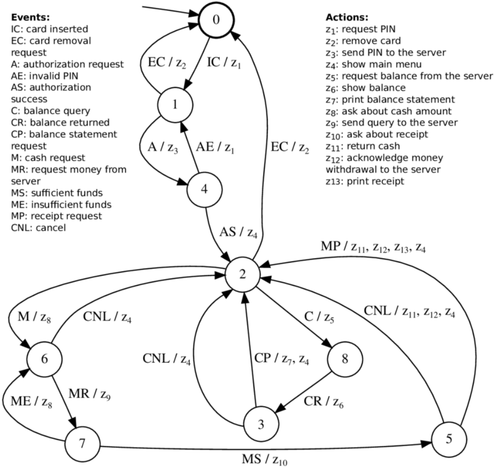
This implies that, for each state p and each word w, there exists **at most one path starting in p and labeled with w**.

An automaton is unambiguous

- if, for all states p, q and all words w, there is at most one path from p to q labeled with w.

Clearly, a deterministic automaton is unambiguous.

Automatic Teller Machine (ATM) instance



from : Ulyantsev, Vladimir & Buzhinsky, Igor & Shalyto, Anatoly. (2016). Exact Finite-State Machine Identification from Scenarios and Temporal Properties. International Journal on Software Tools for Technology Transfer. 10.1007/s10009-016-0442-1.

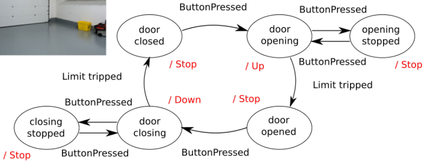
Moore Machine

Moore machines are**different than Mealy ones in the output function**, ω.

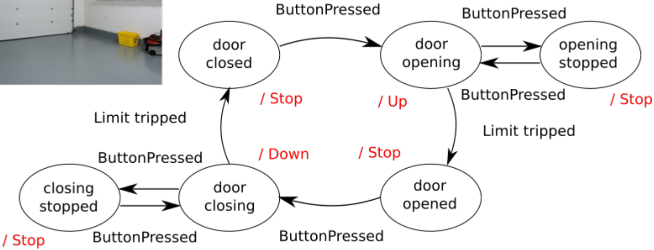
➡ In a Mealy machine output values are determined both by its current state and the current inputs.

So output is produced by its transitions.

➡ In a Moore machine, **output** (in red on figure) is **produced by its states**.



Moore example : Gate operator



Moore vs Mealy

Moore Machine

- ✔ Output is placed on states.
- ✔ So, output depends only upon present state.
- ✔ If input changes, output does not change (before clock triggering).
- ✔ More number of states are required.
- ✔ So there is more hardware requirement.
- ✔ They react slower to inputs (One clock cycle later)
- ✔ Synchronous output and state generation.
- ✔ Easy to design.

Mealy Machine

- ✔ Output is placed on transitions.
- ✔ Output depends on present state as well as present input.
- ✔ If input changes, output also changes.
- ✔ Less number of states are required.
- ✔ There is less hardware requirement.
- ✔ They react faster to inputs.
- ✔ Asynchronous output generation.
- ✔ It is difficult to design.

Mealy Machine in/from Moore Machine

Moore <— transformation/conversion is possible –> Mealy

https://fr.wikipedia.org/wiki/Machine_de_Mealy

Synthetize a FSM

The "big problem" is often to "move" your implementation in the state machine framework.

You have to synthetize the FSM from a specification :

- Traffic lights,
- Sequence detector,
- Lift control,
- ...

Implementation techniques

At least three solutions :

① Goto

② State variable

③ Transition matrix

The good one ? . . .

It depends . . .

- Size of code,
- Size of data,
- Changeability, scalability
- Coding context (challenge)

Goto implementation

```
int main(void){
  char c;

  goto E1; // Etat initial

 E1:
  scanf("%c%*c",&c);   /* Entree */
  if(c=='0'){
    printf("\t0 \n"); /* Sortie */
      goto E1;
    }
  if(c=='1')
    goto E2;
  goto END;

  ...
```

Goto implementation

- It is unusual to promote goto and label ? In fact, **NO** !
  https://www.reddit.com/r/linuxmasterrace/comments/8fmn20/uses_of_goto_in_linux_kernel_source_over_versions/
- In the context of FSM, the "spaghetti" effect is under control since the specification is given by the state diagram.
- "State" is in the code : No need for a variable, neither memory !

  But as a consequence, it is **"hard coded"** !
- Fast to code and efficient to run for a specific automata.

  Good for embedded applications !

"State Variable" implementation

```c
#include <stdio.h>
#define A 0
#define B 1
#define C 2
int main(void){
  int entree;
  int etat = A; /* Etat initial */
  for(;;) {
    switch(etat){
    case A: /** Etat : A */
      /* Lecture de l'entree */
      scanf("%d",&entree);
      /* Calcul de l'etat suivant */
      switch(entree){
      case 0:
        etat = A;
        printf("\t%d\n",0); /* Sortie */
        break;
      case 1:
        etat = B;
        break;
      }
      break;
    case B: /** Etat : B */
    ...
```

"State Variable" implementation

- ➡ The value of the variable is the state.
- ➡ Infinite for-loop
- ➡ The structure of the FSM is in the code.
- ➡ Low memory cost

Transition matrix implementation I

```
#include <stdio.h>
// Author : http://web.archive.org/web/20160808120758/http://www.gedan.net/2009/03/18/finite-state-machine-matrix-style-c-implementation-function-pointers-addon/

typedef enum {
  STATE1,
  STATE2,
  STATE3
} state;

typedef enum {
  NILEVENT,
  EVENT1,
  EVENT2
} event;

typedef void (*action)();
typedef struct {
  state nextState;     // Enumerator for the next state
  action actionToDo;   // function-pointer to the action that shall be released in current state
}  stateElement;

//Actions
void action1_1(void);
void action1_2(void);
void action1_3(void);
void action2_1(void);
void action2_2(void);
void action2_3(void);
void action3_1(void);
void action3_2(void);
void action3_3(void);

int main(void){
```

Transition matrix implementation II

```c
stateElement stateMatrix[3][3] = { // next state,action on transition
  { {STATE1, action1_1}, {STATE2, action1_2}, {STATE3, action1_3} },
  { {STATE2, action2_1}, {STATE2, action2_2}, {STATE3, action2_3} },
  { {STATE3, action3_1}, {STATE3, action3_2}, {STATE3, action3_3} }
};

//Initializations
state  currentState = STATE1;
event  eventOccured = NILEVENT;
action actionToDo   = stateMatrix[currentState][eventOccured].actionToDo;

while(1) {
  // event input, NIL-event for non-changing input-alphabet of FSM
  // in real implementation this should be triggered by event
  // registers e.g. evaluation of complex binary expressions could
  // be implemented to release the events

  int e = 0;

  printf("----------------\n");
  printf("Event to occure (uint) : ");
  scanf("%u",&e);

  //determine the State-Matrix-Element in dependany of current state and triggered event
  stateElement stateEvaluation = stateMatrix[currentState][(event)e];

  // do the transition to the next state
  currentState = stateEvaluation.nextState;

  //... and fire the proper action
  (*stateEvaluation.actionToDo)();

  printf("-----------------\n");

```

Transition matrix implementation III

```
  };
  return (0);
}

/*** action functions ******************************/

void action1_1() {
  printf("action1.1␣\n");
}
void action1_2() {
  printf("action1.2␣\n");
}
void action1_3() {
  printf("action1.3␣\n");
}
void action2_1() {
  printf("action2.1␣\n");
}
void action2_2() {
  printf("action2.2␣\n");
}
void action2_3() {
  printf("action2.3␣\n");
}
void action3_1() {
  printf("action3.1␣\n");
}
void action3_2() {
  printf("action3.2␣\n");
}
void action3_3() {
  printf("action3.3␣\n");
}
```

Transition matrix implementation

- ➡ The automata is in the data structure,
- ➡ The code is independant of the automata topology.

Challenges

2 challenges :

① Rebound filtering

② Vehicule identifiers classification

Rewriting : Rebound filtering

The following example uses a Mealy machine to filter a bit sequence **to remove isolated value occurence**.

Input	->	Output
0	->	0
101	->	111
00101101	->	00001111
01100101001110	->	0110000000111

Two remarks :

① The initial state (resuming the unknown past) is given by the first value of the sequence.

② On the last caracter of the sequence, if you cannot conclued if it is a 0 or a 1, you remove it :

001 will give 00

Can you propose a FSM ?

Remark : Why a FSM ? You could try without ?

- FSM is a formalism . . . to replace the natural language.
- FSM approach will really get this exercise easier because **implementation will be straigth forward** !

  All the "understanding" of the solution is in the FSM !

Vehicule identifiers classification

From a speed limit control system, you get a string which has been proposed by an image recognition tool.

From this string, you would like to define and output if it is a car or a motorcycle (small or big) or a "bad recognition".

If the picture is good, the recognition tool will produce :

① For a car/big motorcycle/. . . , a sequence of 2 upper case letters, 3 digits and 2 upper case letters.

output will be : 'V'

② For a small motorcycle a sequence of 2 upper case letters, 2 digits and 1 upper case letters.

output will be : 'M'

If the picture was not sharp enought, you could get a "nonsense" sequence from the recognition which is different from standards.

output will be : '?'

Index

Index :