Finite state automata

Prof. Daniele Gorla

A formalism for sequential nets

FFs are the simplest sequential nets, but provide the basic issues of such nets:
- storage of boolean values (*state*)
- change of the stored values according to the input signals (*state transitions*)

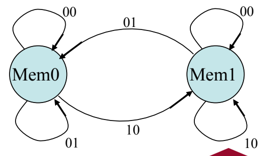
Like TTs are the formalisms for representing combinatorial nets, we look for an analogous formalism for sequential nets.

To describe the behaviour of FFs, we used "extended" TTs, where the time aspect play a crucial role (y vs Y); such a formalism can be made more intuitive by representing it in a graphical way.

s	*r*	*y*	*Y*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

A state for every storable value

For every row of the table, we have an arrow from a state with the value of y to the state with the value of Y. The arrow is labeled with the associated input sequence



Labelled Transition System

A Labelled Transition System (LTS) is a 4-tuple (Q,Σ,q_0,δ) where:
- Q is a (finite) set of *states*;
- Σ is the input *alphabeth*;
- $q_0 \in Q$ is the *initial state*;
- $\delta: Q\times\Sigma \to Q$ is the *transition function*.

REMARK: δ is a function; so, for every pair $(q,a) \in (Q\times\Sigma)$ there exists one and only one state reached by the automaton!!

In the previous example, we have that:
- Q = {Mem0, Mem1};
- Σ = {00,01,10};
- δ: (Mem0,00) → Mem0 (Mem1,00) → Mem1
  (Mem0,01) → Mem0 (Mem1,01) → Mem0
  (Mem0,10) → Mem1 (Mem1,10) → Mem1
- What about q_0? It depends to the initial value stored into the FF (typically, we assume 0)

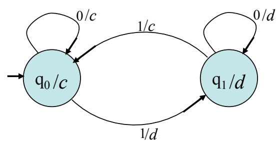
Automata with output

To the definition of LTS, we can add an *output alphabeth* Δ and an *output function* λ, to obtain *automaton with output*, that is a 6-tuple $M = (Q,\Sigma,\Delta,q_0,\delta,\lambda)$.

To define the output function, we can naturally associate the output to states or to transitions; this yields two different models:
- *Moore Automaton*, in which $\lambda : Q \to \Delta$
- *Mealy Automaton*, in which $\lambda : Q\times\Sigma \to \Delta$

Graphically, outputs are denoted by writing "/ b" (for $b \in \Delta$) after the name of the state (Moore model) or after the input character (Mealy model).

Ex.:



Mealy model

Moore model

Example: Drink-delivery machine

We'd like to model a concrete system that delivers drink cans:

- Cans costs 30c;
- The machine only accepts coins of 10c and 20c;
- The machine delivers a can if the user has inserted at least 30c and gives no change (but holds the change for the next can).

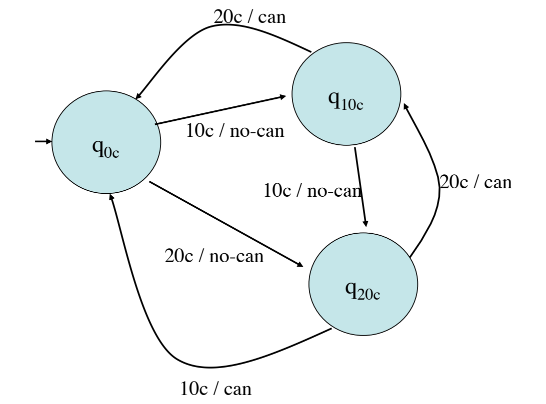
The fundamental step to design an automaton is to understand what must be stored during the computation, that is defining the states and their meaning.

In this example, we have to remember the credit received after the last can delivery (or from the outset). How many different states?

0c, 10c, 20c → 3 states

REMARK: we don't need states for 30c or 40c because in this case the machine delivers the can and comes back to state 0c or 10c, respectively.

Drink-delivery machine (graphical representation of the automaton)



Drink-delivery machine (mathematical representation of the automaton)

$Q = \{q_{0c}, q_{10c}, q_{20c}\}$
$\Sigma = \{10c, 20c\}$
$\Delta = \{\text{no-can}, \text{can}\}$
Initial state: q_{0c}

δ: $(q_{0c},10c) \to q_{10c}$
$(q_{0c},20c) \to q_{20c}$
$(q_{10c},10c) \to q_{20c}$
$(q_{10c},20c) \to q_{0c}$
$(q_{20c},10c) \to q_{0c}$
$(q_{20c},20c) \to q_{10c}$

λ: $(q_{0c},10c) \to$ no-can
$(q_{0c},20c) \to$ no-can
$(q_{10c},10c) \to$ no-can
$(q_{10c},20c) \to$ can
$(q_{20c},10c) \to$ can
$(q_{20c},20c) \to$ can

For simplicity, all these info's can be written down in a more compact way in the so-called *tabular representation*:

	10c	20c
→ q_{0c}	q_{10c} / no-can	q_{20c} / no-can
q_{10c}	q_{20c} / no-can	q_{0c} / can
q_{20c}	q_{0c} / can	q_{10c} / can

An evolved Drink-delivery machine

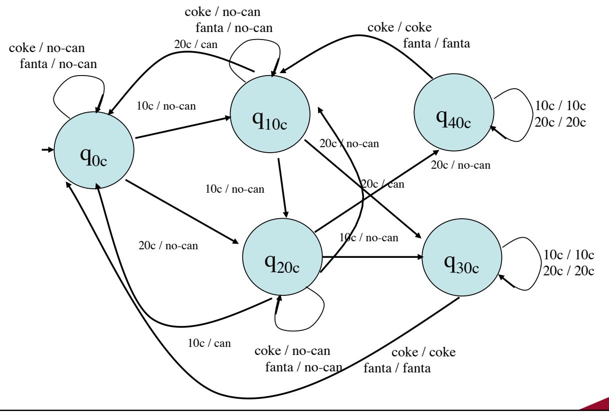
Previous example can be made more realistic by giving the user the possibility of choosing the desired can (e.g., coke or fanta)

This new specification, substantially modifies the automaton. Indeed, 3 states are no more enough, but we need 5 of them:

→ previously, the can was delivered upon reaching 30c
→ now we have also to wait the choice of the can (this requires for the states 30c and 40c)
→ moreover, in such new states, if the user still adds more coins, the machine has to give them back, without changing state
→ by contrast, in old states any can selection must be ignored, since the total of 30c has not yet been reached

REMARK: we have to consider all states and inputs, since δ and λ are functions!

Evoled Drink-delivery machine



Example: automaton for adding naturals

IN: two bit sequences, A = $a_0\, a_1\, a_2 \dots$ and B = $b_0\, b_1\, b_2 \dots$

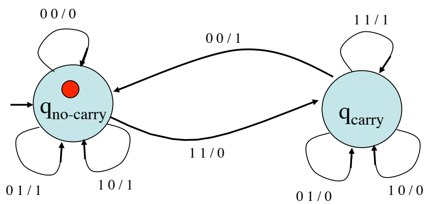
OUT: a bit sequence $s_0\, s_1\, s_2 \dots$ s.t., for every i, $s_i \dots s_0 = a_i \dots a_0 + b_i \dots b_0$, by ignoring the final carry.

Example: A: 0
B: 1
output: 1

SOLUTION:

- The automaton receives input bits from the less to the more signifying ones; it produces the outputs in the same way
- This is similar to the way in which the sum is manually performed (also at the core of the combinatorial adder)
- The only info that we need for passing from bit i-th to bit $(i+1)$-th is knowing whether at step i there was a carry or not
  → this will be the meaning of the states, that hence will be just 2

Automaton for the adder



Let's check that the automaton behaves is the desired way:

A: 0 1 1 0 0 1 0
B: 1 0 1 0 1 1 1
output: 1 1 0 1 1 0 0

Example: automaton for calculating the remainder modulo 4

IN: a bit sequence, B = $b_0\, b_1\, b_2 \dots$

OUT: a sequence of bit pairs $r_0\, r'_0\, r_1\, r'_1 \dots$ s.t., for every i,
$r_i\, r'_i = b_0 \dots b_i$ MOD 4.

Example: B: 1
output: 0
1

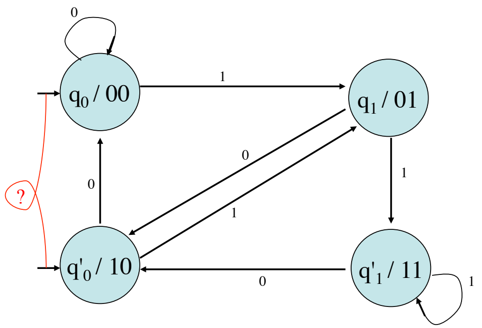
SOLUTION:

- In this example, the automaton receives bits from the most signifying it (new bits are enqueued)
- According to the reminder theorem, according to which the reminder in a division of a natural (in base 2) by 2^k is given by the k less signifying bits of the given number.
- The only info we need when the i-th bit arrives is the values of the $(i\text{–}1)$-th bit, apart from the first one (for which no info is required)
  → we only have 2 states
  → the initial state is that associated to bit 0, since the natural represented by bit b has the same reminder (modulo 4) as $0b$

Automaton for reminders modulo 4

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

This time we use Moore model:



The initial state can be q_0 or q'_0 , since in the Moore model the first output is usually ignored (it is a "default" output that is always produced, without considering any input)