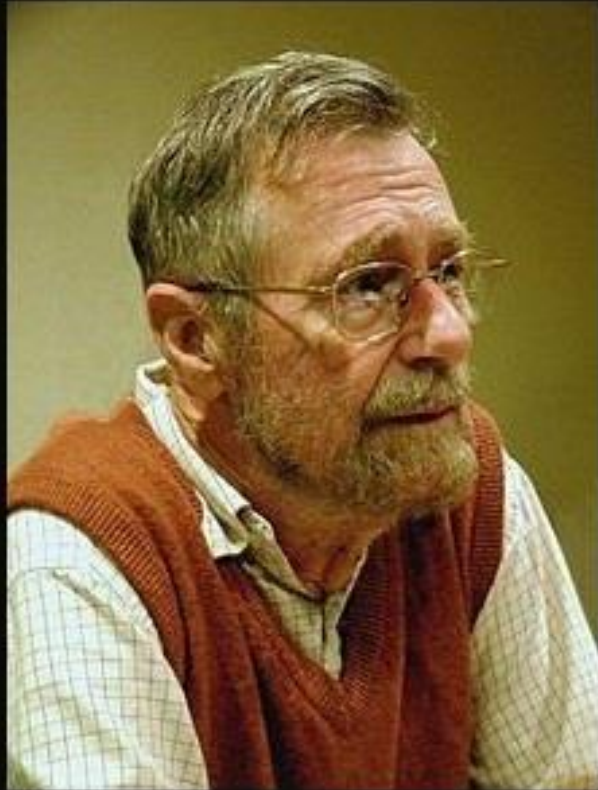


Theoretical Computer Science

So far so good!

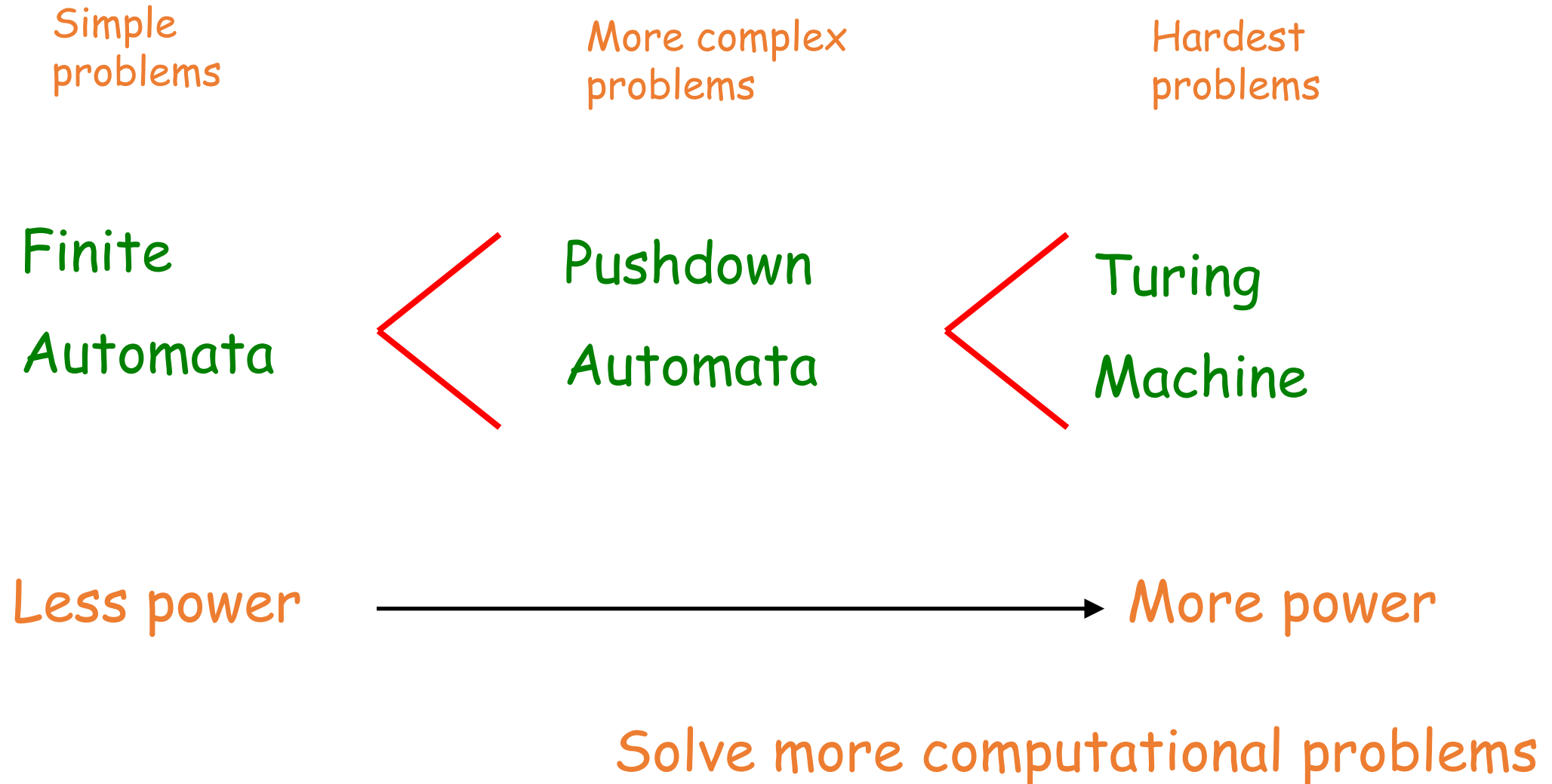
Lecture 3 - Manuel Mazzara



Computer science is no more about computers
than astronomy is about telescopes.

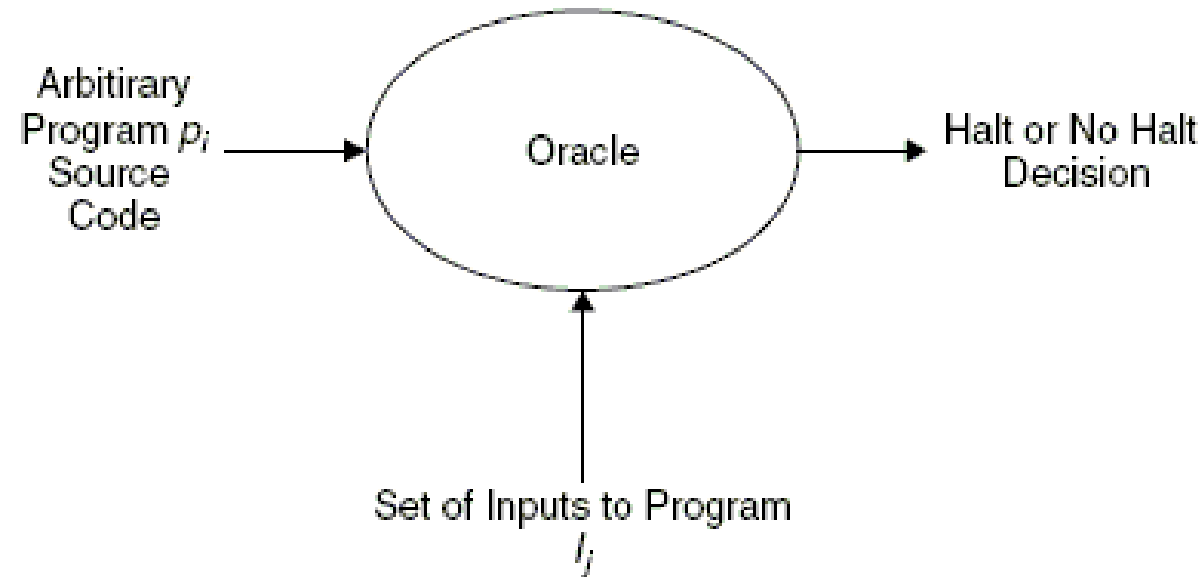
(Edsger Dijkstra)

Power of Automata



Our big question

- Is every function computable?
 - Can I write an algorithm for any function $N \rightarrow N$?
 - **Halting Problem**





From now on I will consider a language to be a set (finite or infinite) of sentences, **each finite in length** and constructed out of a **finite set of elements**. All natural languages in their spoken or written form are languages in this sense.

— Noam Chomsky —

An Alphabet!

And so are
Programming
Languages!

Theoretical Computer Science

Finite State Automata (FSA)

Lecture 3 - Manuel Mazzara

**A finite-state
automaton (FSA) is
a simple form of
imaginary machine**

AKA

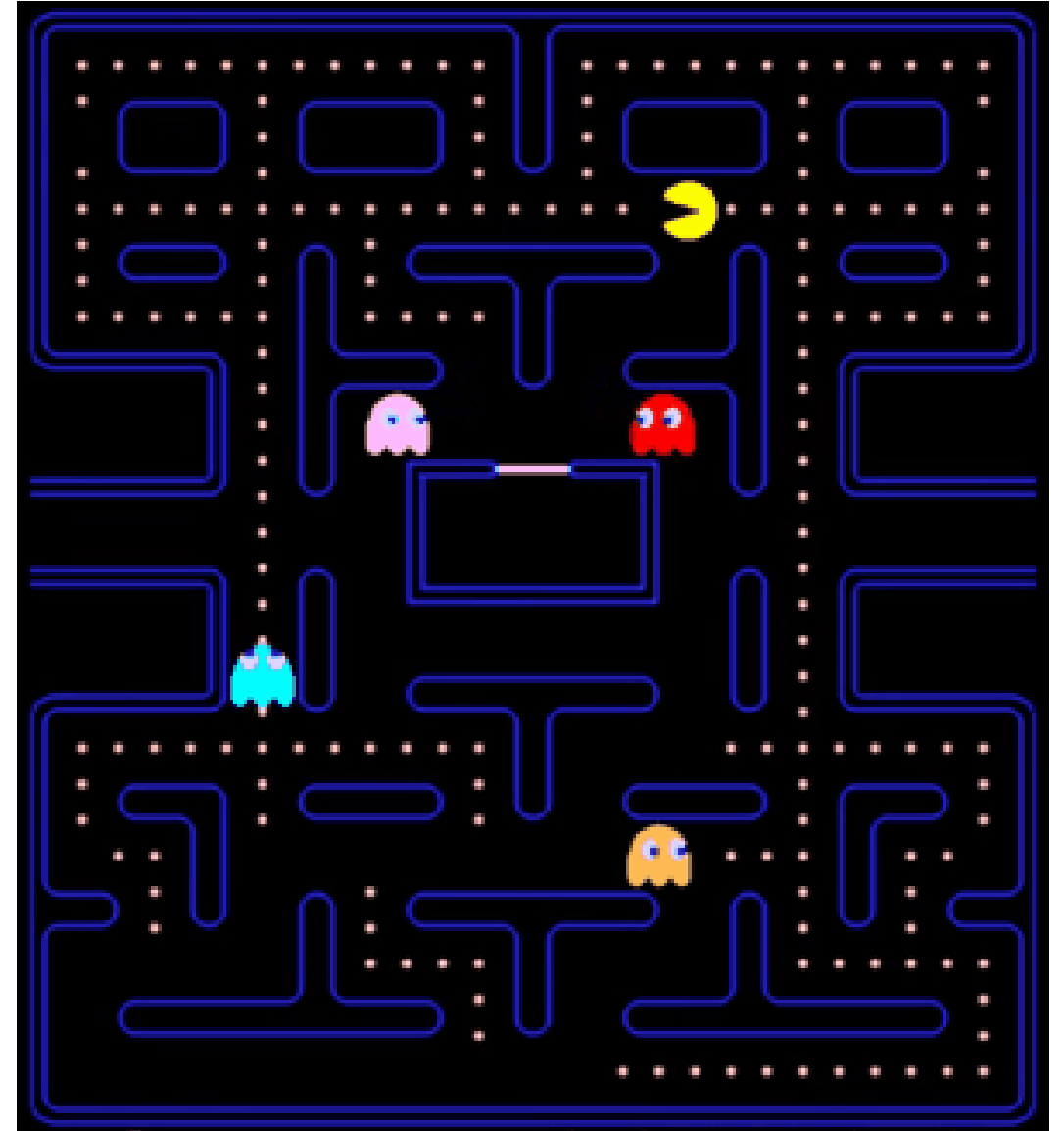
- Finite-state machine
- Finite automaton
- Finite-state transducer
 - It is a variant that we will see
- Representations
 - transition table
 - **directed graph where vertices are states and edges are transitions**

FSA can have several purposes, we will mention

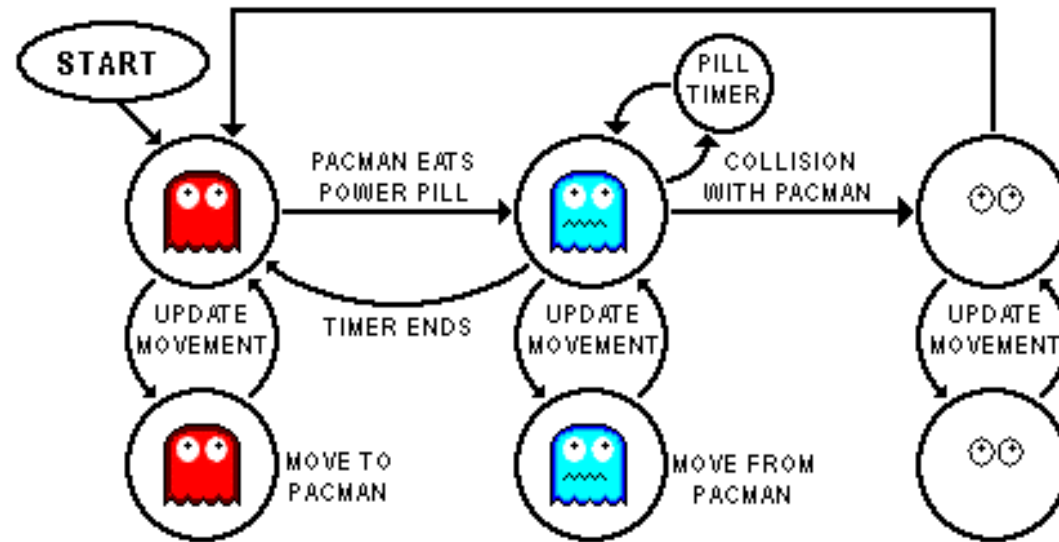
- ▼ some, but our main interest is **language recognition**

Pac-Man

Released in Japan in 1980



Pac-Man Ghost



States

- An FSA has a **finite** set of **states**
 - A system has a **limited number of configurations**
- Examples
 - {On, Off},
 - {1,2,3,4, ...,k}
 - {TV channels}
 - ...
- States can be **graphically** represented as follows:



Input

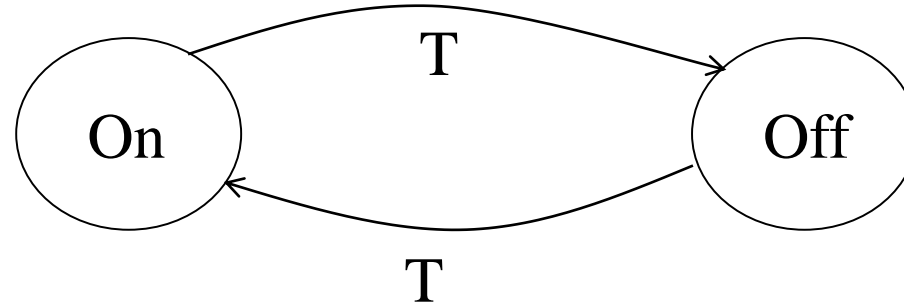
- An FSA is defined over an **alphabet**
- The symbols of the alphabet represent the **input** of the system
- Examples
 - {switch_on, switch_off}
 - {incoming==0, 0<incoming<=10, incoming>10}

Transitions among states

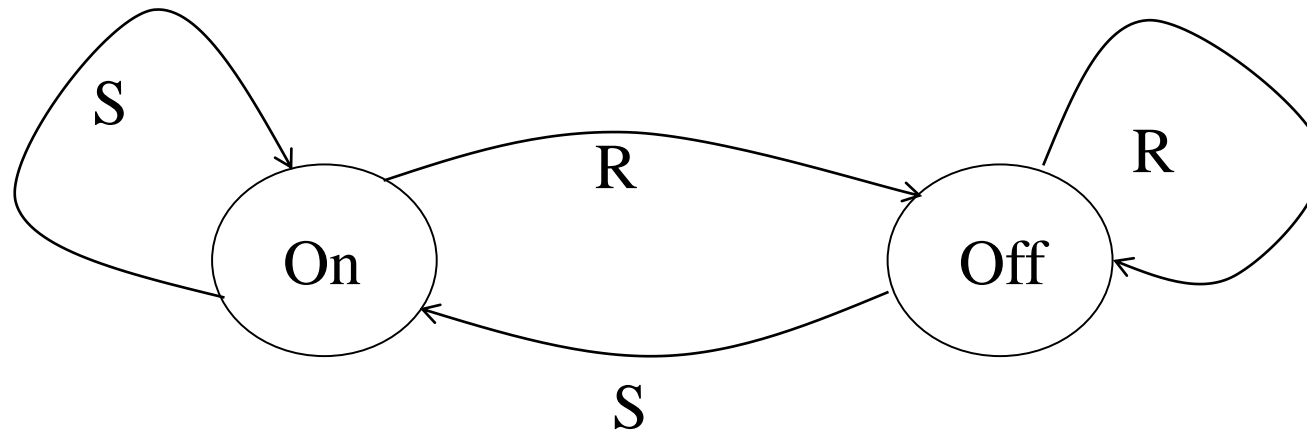
- When an input is received, the system **changes its state**
- The passage between states is performed through **transitions**
- A transition is graphically represented by arrows:



Simple examples



What are these two FSAs modelling?





FSA

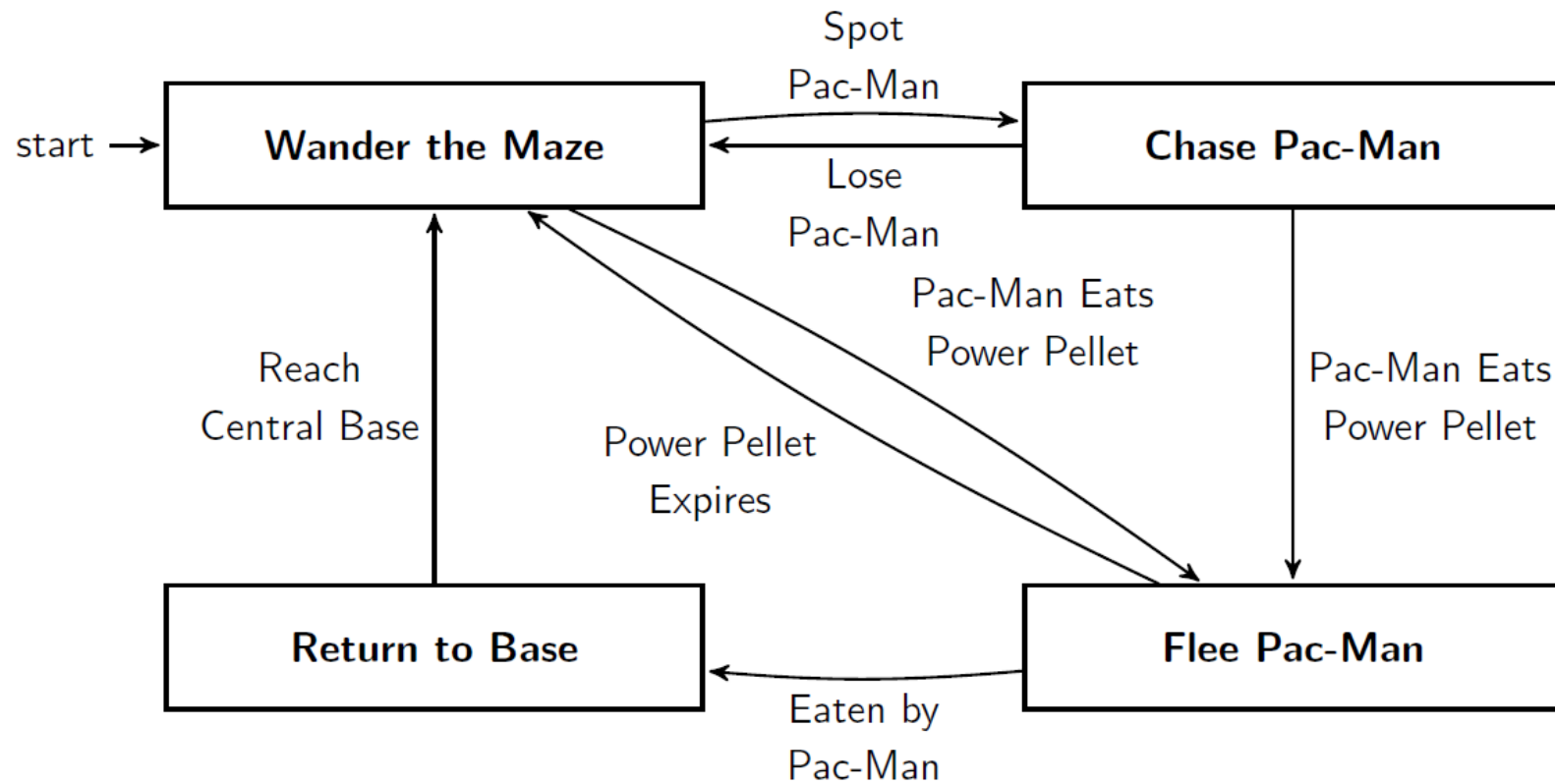
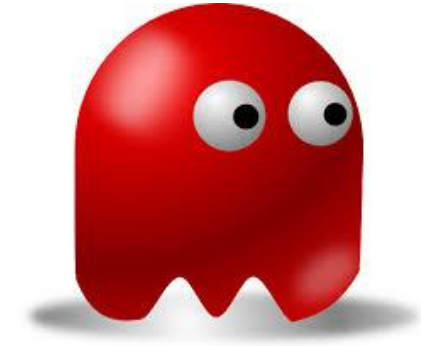
- FSAs are the **simplest** model of computation
- Many useful devices can be modeled using FSAs

... but they have some limitations

Applications of FSA

- Vending Machines
- Traffic Lights
- Video Games
- Text Parsing
- CPU Controllers
- Protocol Analysis
- Natural Language Processing
- Speech Recognition

Pac-Man Ghost again



History

- The world automata come from very far – we will see
 - Systems which could transition between a finite number of internal states have been known of for a long time
- Only relatively late in the early history of computer science they were formally studied
 - McCulloch&Pitts : A logical calculus of the ideas immanent in nervous activity (1943)
 - Kleene : equivalence of FSA and regexps (1956)
 - Mealy, Moore : generalized (1955-56)
 - Scott-Rabin : NDFSA (1959)
 - Ginsburg : Finite Transducers (1962)

“Turing machines are widely considered to be the abstract prototype of digital computers; workers in the field, however, have felt more and more that the notion of a Turing machine is too general to serve as an accurate model of actual computers. It is well known that even for simple calculations it is impossible to give an a priori upper bound on the amount of tape a Turing machine will need for any given computation. It is precisely this feature that renders Turing's concept unrealistic.”

Rabin and Scott - "Finite automata and their decision problems," IBM Journal of Research and Development (April, 1959)

“In the last few years the idea of a finite automaton has appeared in the literature. These are machines having only a finite number of internal states that can be used for memory and computation. The restriction of finiteness appears to give a better approximation to the idea of a physical machine. Of course, such machines cannot do as much as Turing machines, but the advantage of being able to compute an arbitrary general recursive function is questionable, since very few of these functions come up in practical applications”


Rabin and Scott - "Finite automata and their decision problems," IBM Journal of Research and Development (April, 1959)

Theoretical Computer Science

Finite State Automata, formally

Lecture 3 - Manuel Mazzara

Math is easy,
notation may
overwhelm you!



As a computer scientist you must cope with notation – for example, programming

Informal vs. Formal

- Always three stages:
 - **Intuition**/idea/informal
 - **Examples**/instances
 - **Formal** definition
 - Human vs. machine understanding

Formally

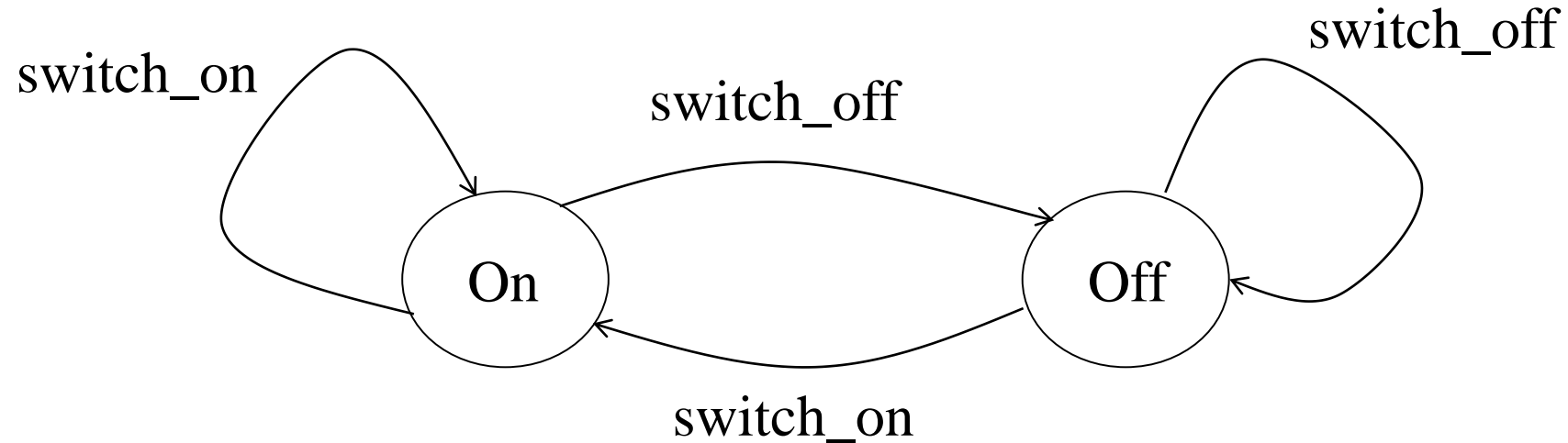
- An FSA is (**for now**) a triple $\langle Q, A, \delta \rangle$, where
 - Q is a finite set of states
 - A is the input alphabet
 - δ is a transition function (that can be **partial**), given by $\delta: Q \times A \rightarrow Q$

Delta (lowercase)

- Remark

if the function is **partial**, then not all the transitions from all the possible states for all the possible elements of the alphabet are defined (for example pressing sugar+ in a vending machine during coffee release)

Partial vs Total Transition Function



An FSA with a total transition function is called **complete**

Recognizing languages

- In order to be able to use FSAs for recognizing languages, it is important to identify:
 - the initial conditions of the system
 - the final admissible states
- Example:
 - The light should be off at the beginning and at the end

Elements

- The elements of the model are

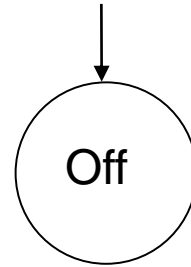
- **States**
- **Transitions**
- **Input**

and also

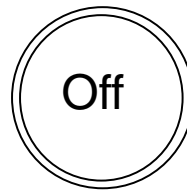
- **Initial state(s)**
- **Final state(s)**

Graphical representation

- Initial state



- Final state

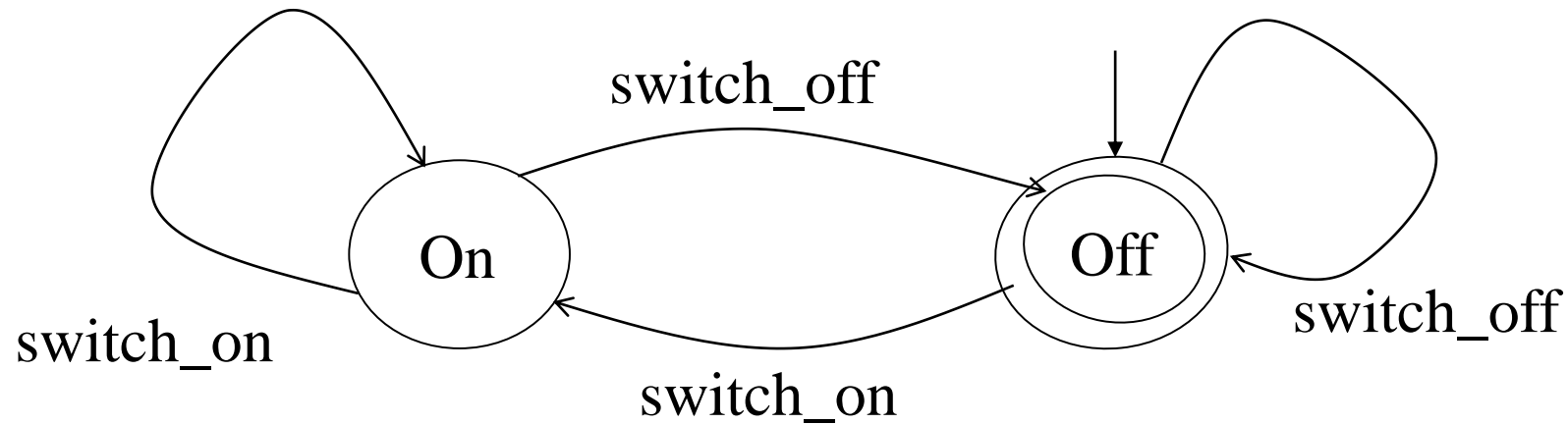


Formally

- An FSA is a tuple $\langle Q, A, \delta, q_0, F \rangle$ where
 - Q is a finite set of states
 - A is the input alphabet
 - δ is a (partial) transition function, given by
$$\delta: Q \times A \rightarrow Q$$
 - $q_0 \in Q$ is called initial state
 - $F \subseteq Q$ is the set of final states

Move sequence

- A **move sequence** starts from an initial state and is ***accepting*** if it reaches one of the final states

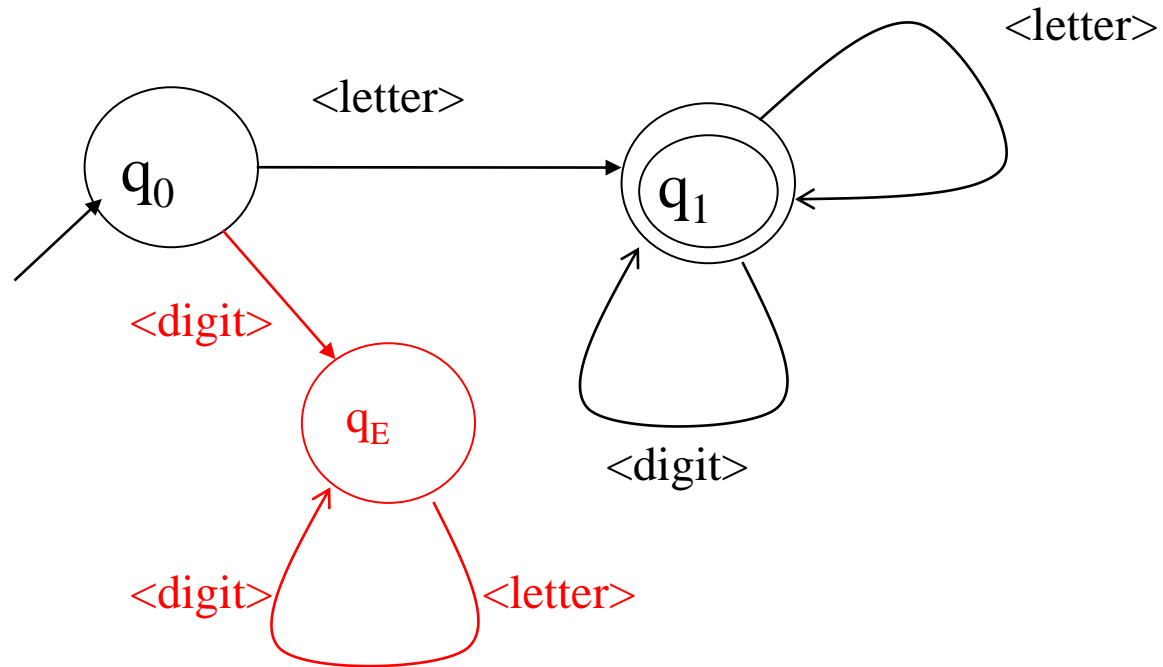


Formally

- Move sequence:
 - $\delta^*: Q \times A^* \rightarrow Q$
- δ^* is inductively defined from δ
 - $\delta^*(q, \varepsilon) = q$
 - $\delta^*(q, y.i) = \delta(\delta^*(q, y), i)$
- Initial state: $q_0 \in Q$
- Final (or accepting) states: $F \subseteq Q$
- $\forall x (x \in L \leftrightarrow \delta^*(q_0, x) \in F)$

A practical example

- Recognizing Pascal identifiers



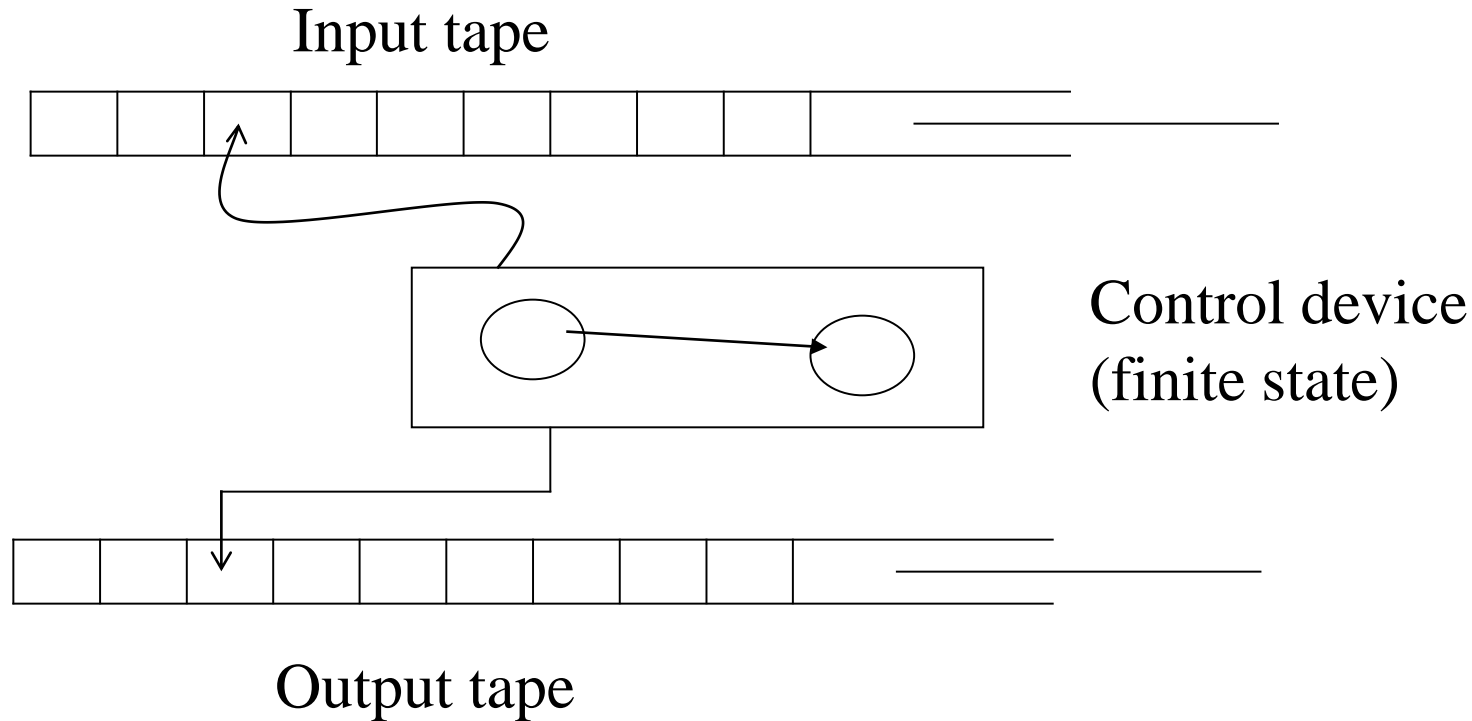
You can argue that every physical instance of a computer is an FSA, but...? And BTW.. why? Remember the words of Rabin and Scott!

Theoretical Computer Science

Finite state transducers

Lecture 3 - Manuel Mazzara

Automata as language translators



A finite state transducer is an **FSA that works on two tapes**
→ it is a **translating machine**

Finite-state transducer (FST)

- Ordinary finite-state automata (also called finite-state *acceptor* for contrast with FST) have a **single tape**
- **Finite-state transducer (FST)**
 - Finite-state machine with **two tapes**
 - **input** tape and an **output** tape
- FST is a type of FSA which **maps** between two sets of symbols
- FST is more general than FSA
 - FSA defines a formal language by defining **a set of accepted strings**
 - FST **defines relations** between sets of strings

The idea

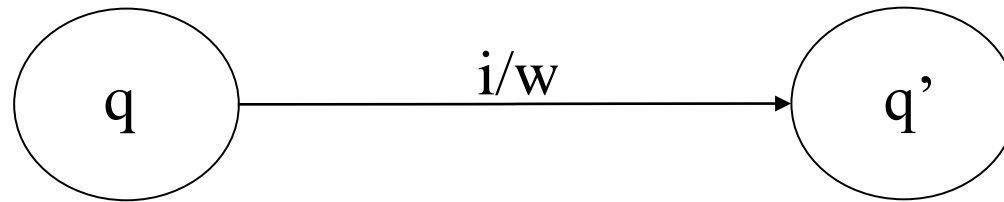
- $y = \tau(x)$
 - x : input string
 - y : output
 - τ : function from L_1 to L_2
- Examples:
 - τ_1 the occurrences of “1” are doubled ($1 \rightarrow 11$)
 - τ_2 ‘a’ is swapped with ‘b’ ($a \leftrightarrow b$):
- but also
 - **Compression** of files
 - **Compiling** from high level languages into object languages
 - **Translation** from English into Russian



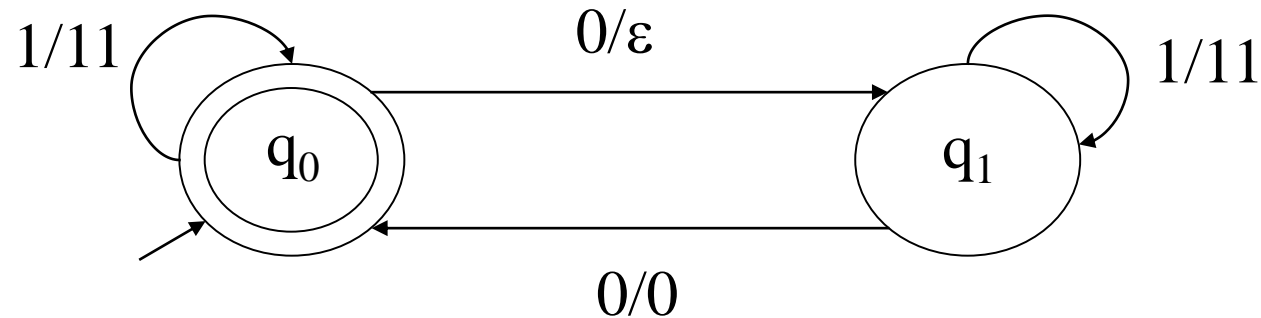
Tau (lowercase)

Informally

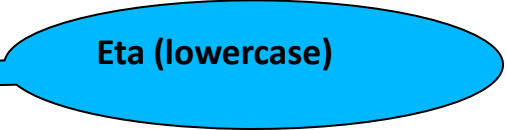
- Transitions with output



- Example: τ halves the number of “0”s and doubles the number of “1”s



Formally

- A finite state transducer (**FST**) is a tuple $T = \langle Q, I, \delta, q_0, F, O, \eta \rangle$
 - $\langle Q, I, \delta, q_0, F \rangle$: just like acceptors
 - O : output alphabet
 - $\eta : Q \times I \rightarrow O^*$ 
- Remark: the condition for acceptance remains the same as in acceptors
 - **The translation is performed only on accepted strings**

Translating a string

- As we did for δ , we define η^* **inductively**
 - $\eta^*(q, \varepsilon) = \varepsilon$
 - $\eta^*(q, y.i) = \eta^*(q, y). \eta(\delta^*(q, y), i)$
- Remark $\eta^*: Q \times I^* \rightarrow O^*$

$$\forall x (\tau(x) = \eta^*(q_0, x) \text{ iff } \delta^*(q_0, x) \in F)$$

The translation is performed only on accepted strings