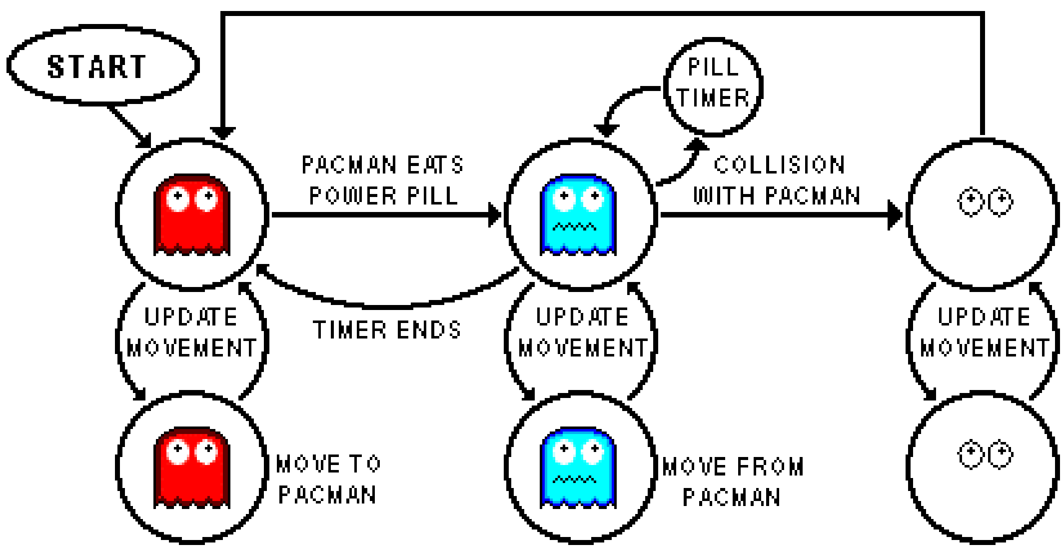
Theory of Computation

Finite state automata

Lecture 2b - Manuel Mazzara

Pac-Man Ghost



States

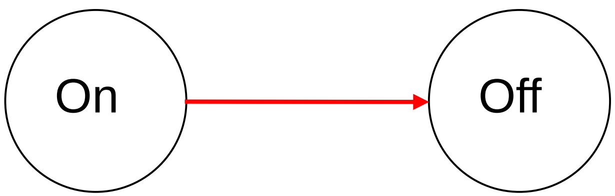
- An FSA has a **finite** set of **states**
  – A system has a limited number of configurations
- Examples
  – {On, Off},
  – {1,2,3,4, …,k}
  – {TV channels}
  – …
- States can be **graphically** represented as follows:

Input

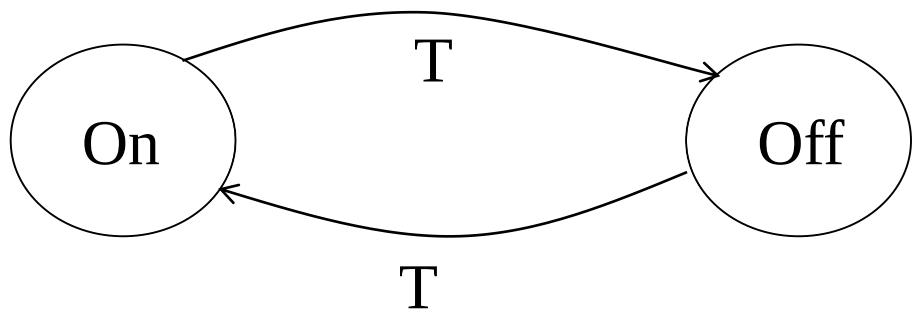
- An FSA is defined over an **alphabet**
- The symbols of the alphabet represent the **input** of the system
- Examples
  - {switch_on, switch_off}
  - {incoming==0, 0<incoming<=10, incoming>10}

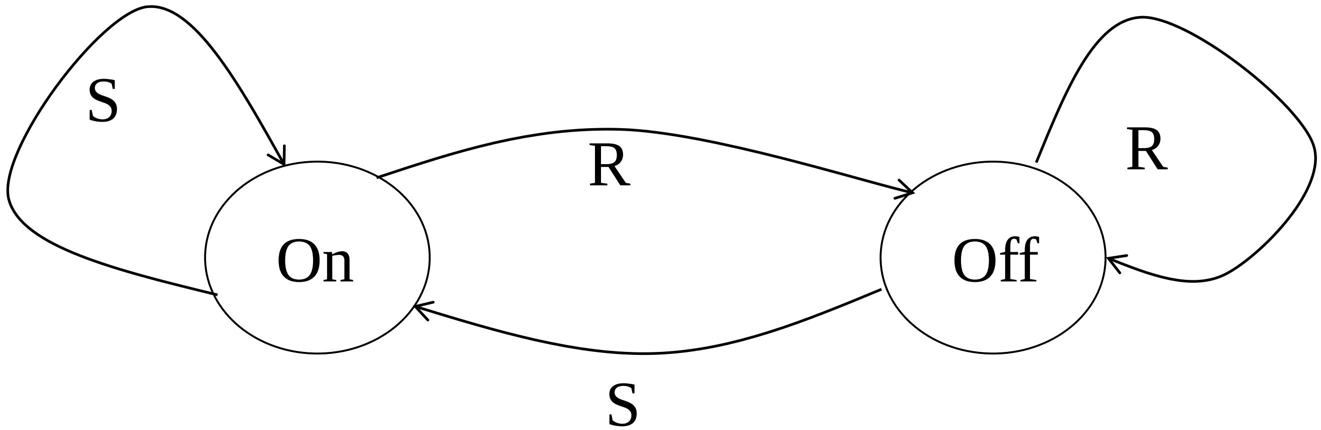
Transitions among states

- 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





FSA

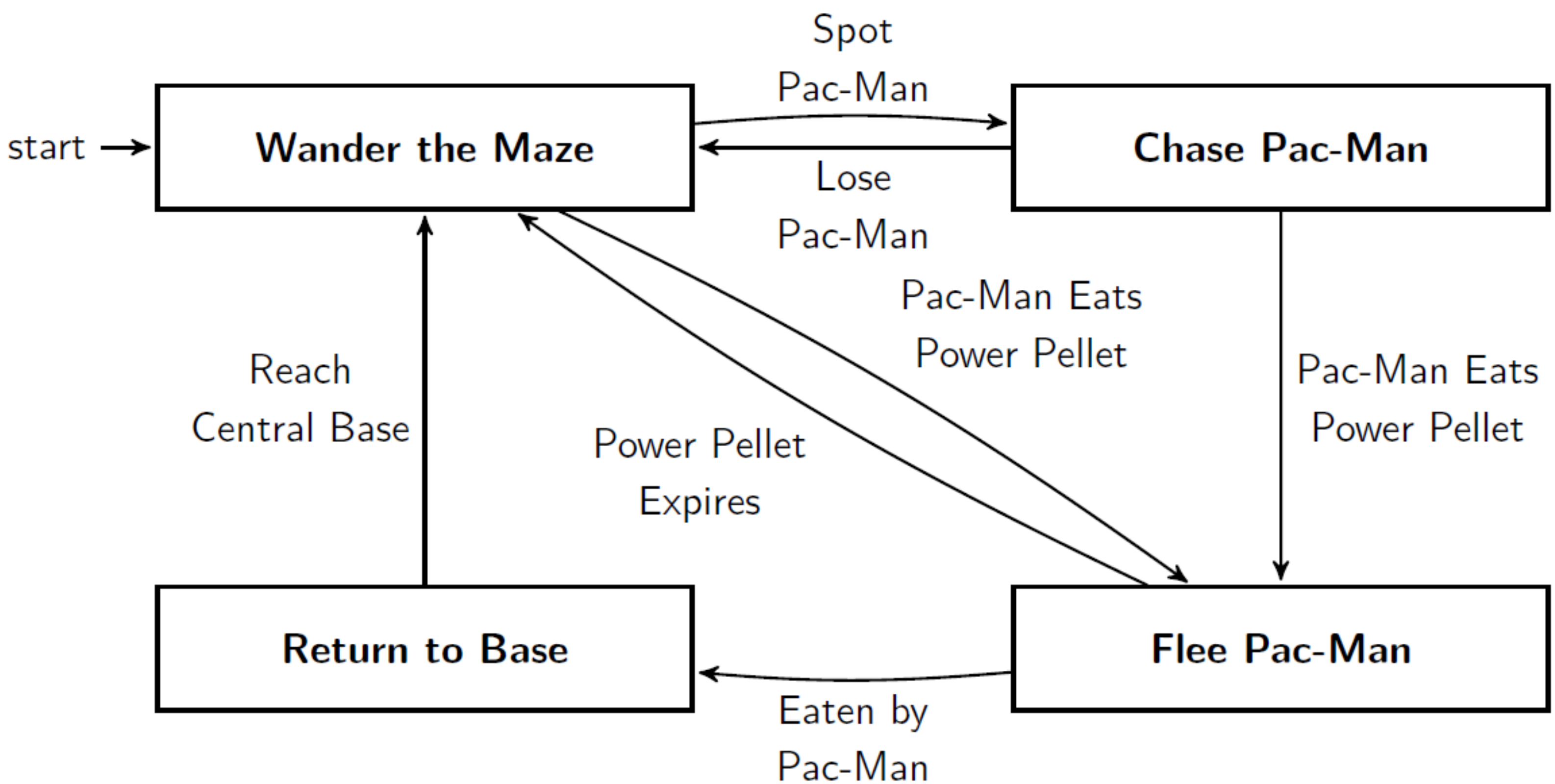
- FSAs are the <u>**simplest**</u> model of computation
- Many useful devices can be modeled using FSAs

… 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



Now, formally

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

Formally

- An FSA is a triple **$<Q, A, \delta>$,** where

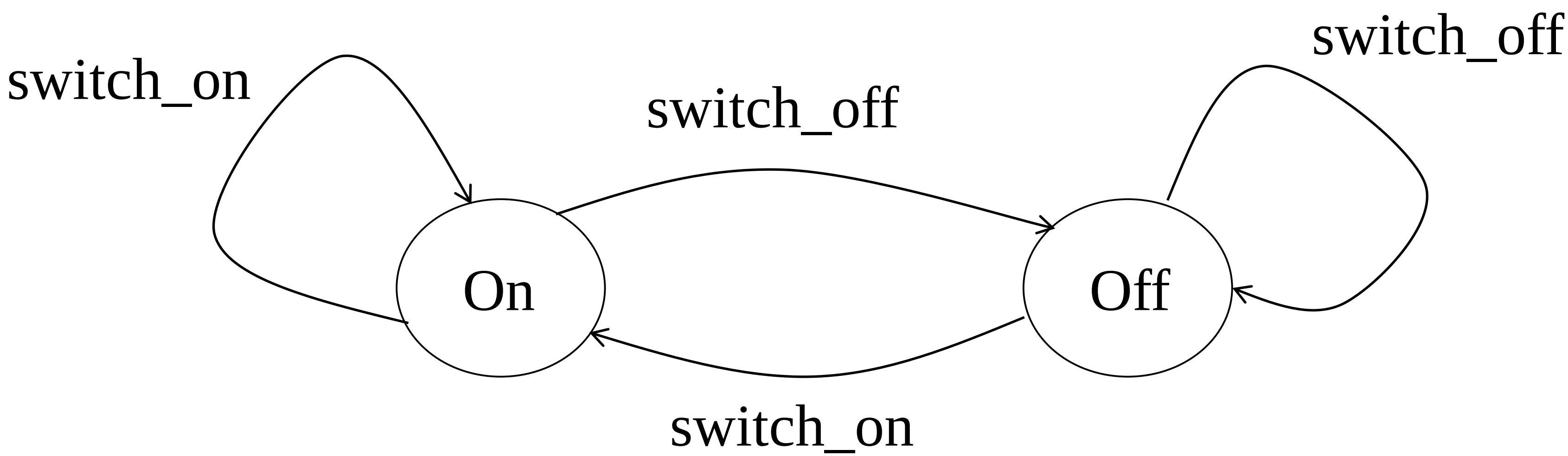
Delta (lowercase)

  - Q is a finite set of **states**
  - A is the input **alphabet**
  - δ is a **transition function** (that can be partial), given by $\delta: Q \times A \rightarrow Q$
- 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

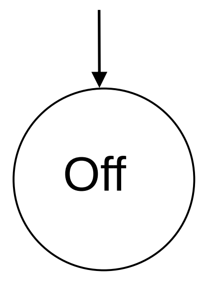
Elements

- The elements of the model are
  - **States**
  - **Transitions**
  - **Input**

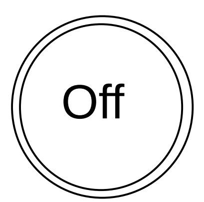
  and also
  - **Initial state(s)**
  - **Final state(s)**

Graphical representation

- Initial state



- Final state

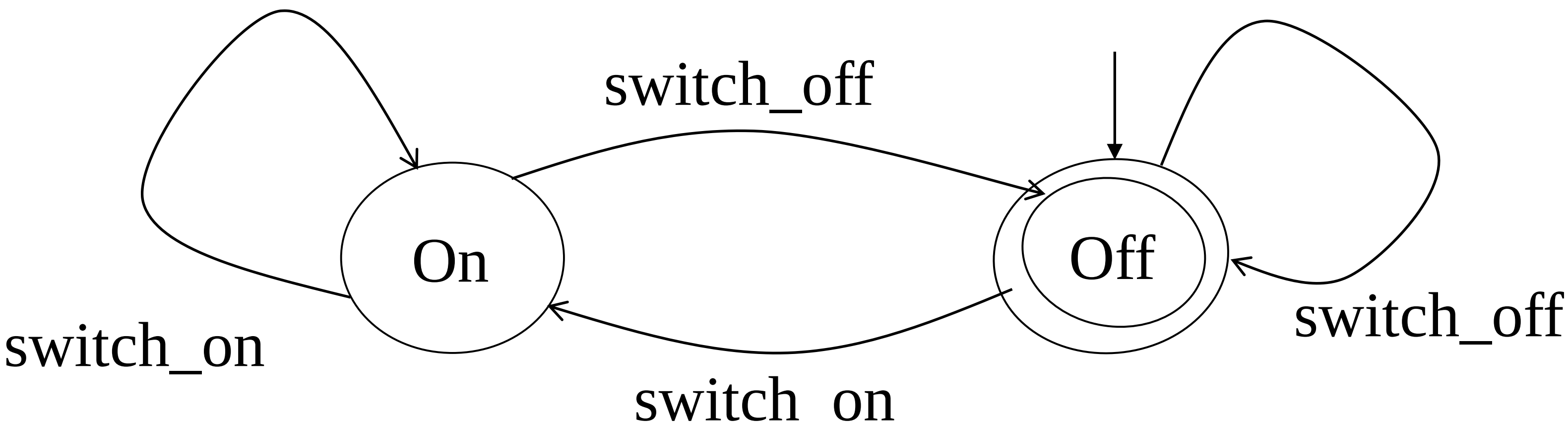


Formally

- An FSA is a **tuple** **<Q, A, δ , q_0, F>**, 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 <u>**move sequence**</u> starts from an initial state and is ***accepting*** if it reaches one of the final states

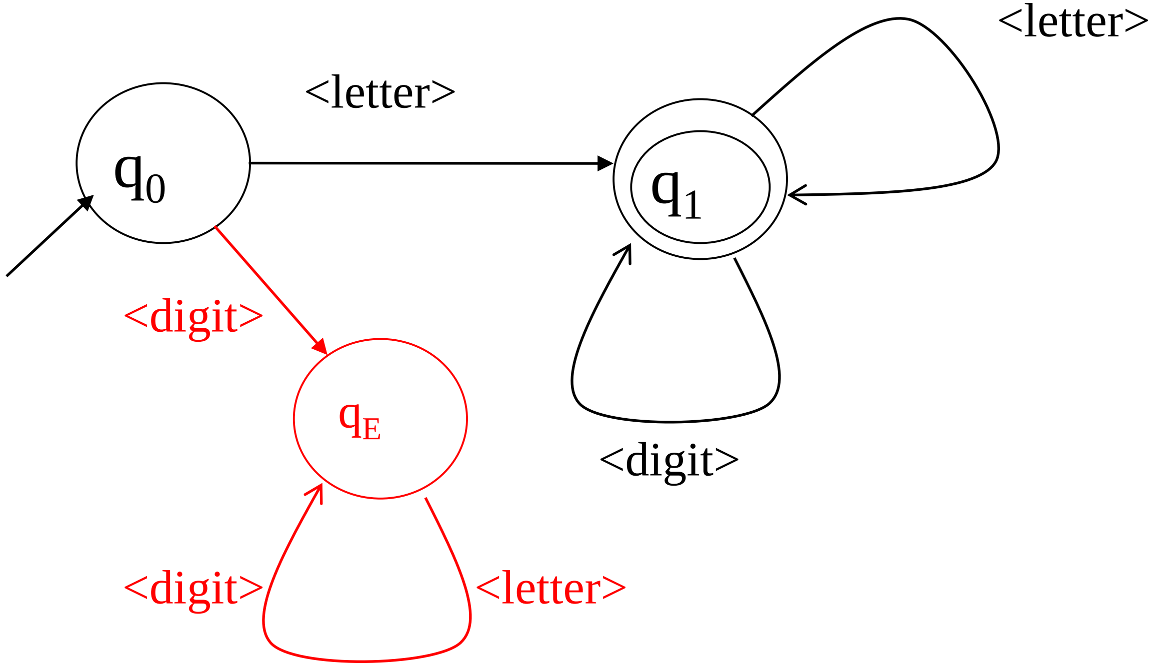


Formally

- Move sequence:
  - $\delta^*: Q \times A^* \to Q$
- 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: $F \subseteq Q$
- $\forall x\ (x \in L \leftrightarrow \delta^*(q_0,x) \in F)$

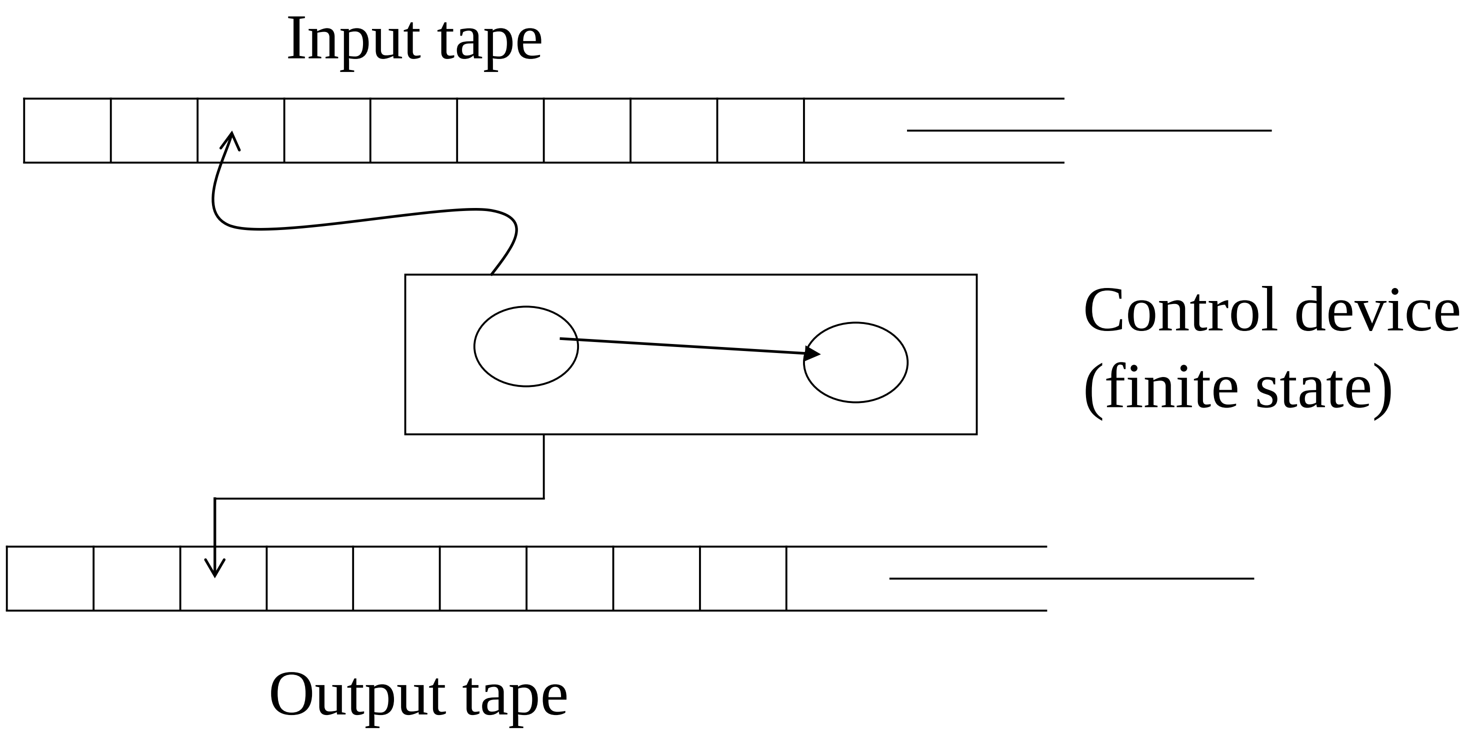
A practical example

- Recognizing Pascal **identifiers**



Finite state transducers

Automata as language translators



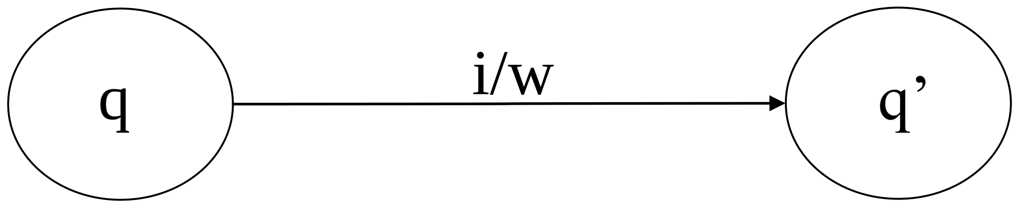
A finite state transducer is an <u>**FSA that works on two tapes**</u>.
→ it is a kind of ``translating machine''.

The idea

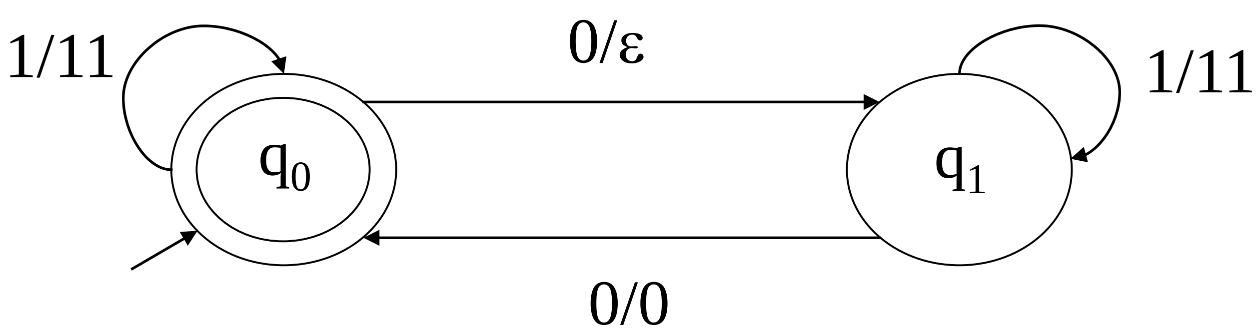
- $y = \tau(x)$
  Tau (lowercase)
  - x: input string
  - y: output
  - τ: function from L_1 to L_2
- Examples:
  - τ_1 the occurrences of "1" are doubled (1 --> 11)
  - τ_2 'a' is swapped with 'b' (a <---> b):
- but also
  - **Compression** of files
  - **Compiling** from high level languages into object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

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
  $T = <Q, I, \delta, q_0, F, O, \eta>$
  - $<Q, I, \delta, q_0, F>$: just like acceptors
  - O: <u>**output alphabet**</u>
  - $\eta : Q \times I \to O^*$

Eta (lowercase)

- Remark: the condition for acceptance remains the same as in acceptors
  - **The translation is performed only on accepted strings**

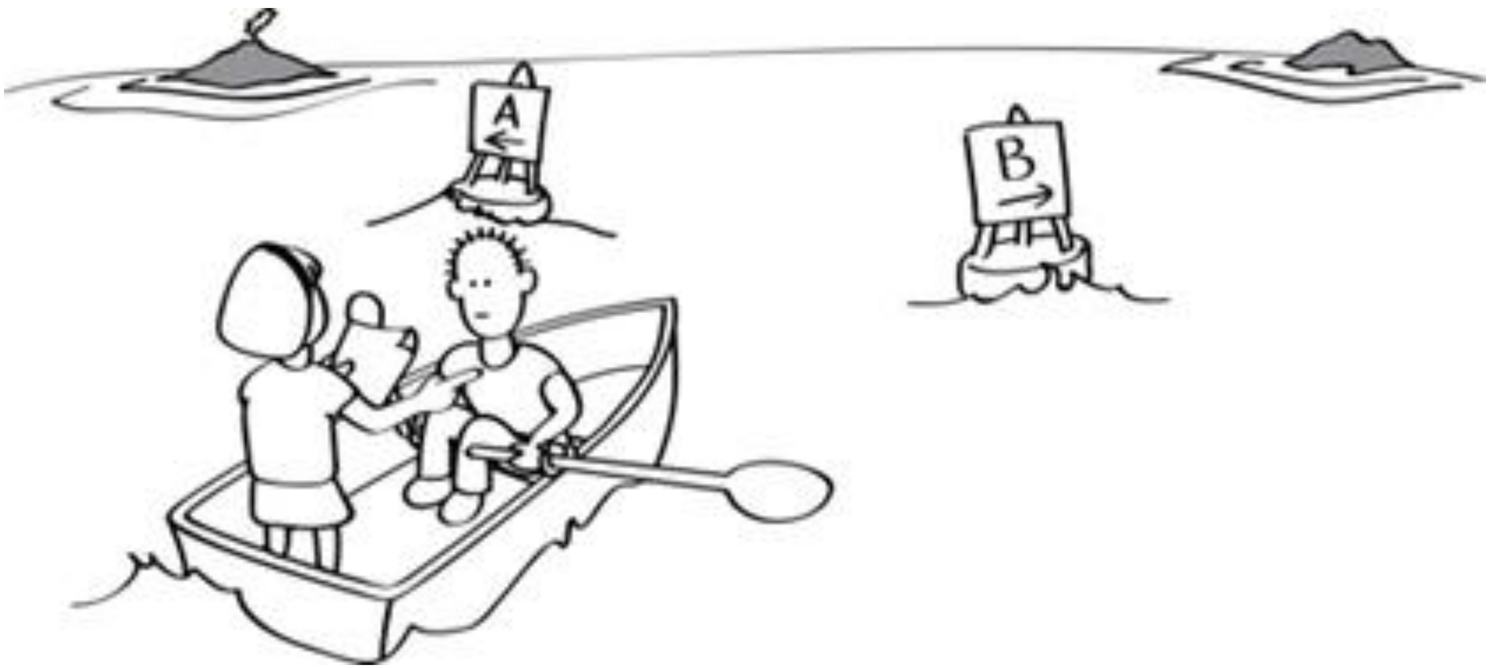
Translating a string

- 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 $\eta^*: Q \times I^* \rightarrow O^*$

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

The translation is performed only on accepted strings

FSA



Operations on FSA

Closure in math

- A **set** is **<u>closed</u>** w.r.t. an **operation** if the operation is applied to elements of the set and the result is **still an element of the set**
- From math we know:
  - Natural numbers are closed w.r.t. sum (but not subtraction)
  - Integers are closed w.r.t. sum, subtraction, multiplication (but not division)
  - Rationals: are they closed by division? Consider zero!
  - Reals…
  - …

Rationals

- A rational number is a number that can be represented as a fraction **m/n**, where **m** and **n** are integers and **n≠0**

- Rational numbers are closed under **addition**, **subtraction**, **multiplication**, as well as **division by a nonzero rational.**

$$\frac{a}{b} \times \frac{c}{d} = \boxed{\frac{ac}{bd},} \frac{a}{b} + \frac{c}{d} = \boxed{\frac{ad+bc}{bd}} \text{ and } \frac{a}{b} \div \frac{c}{d} = \boxed{\frac{ad}{bc}}$$

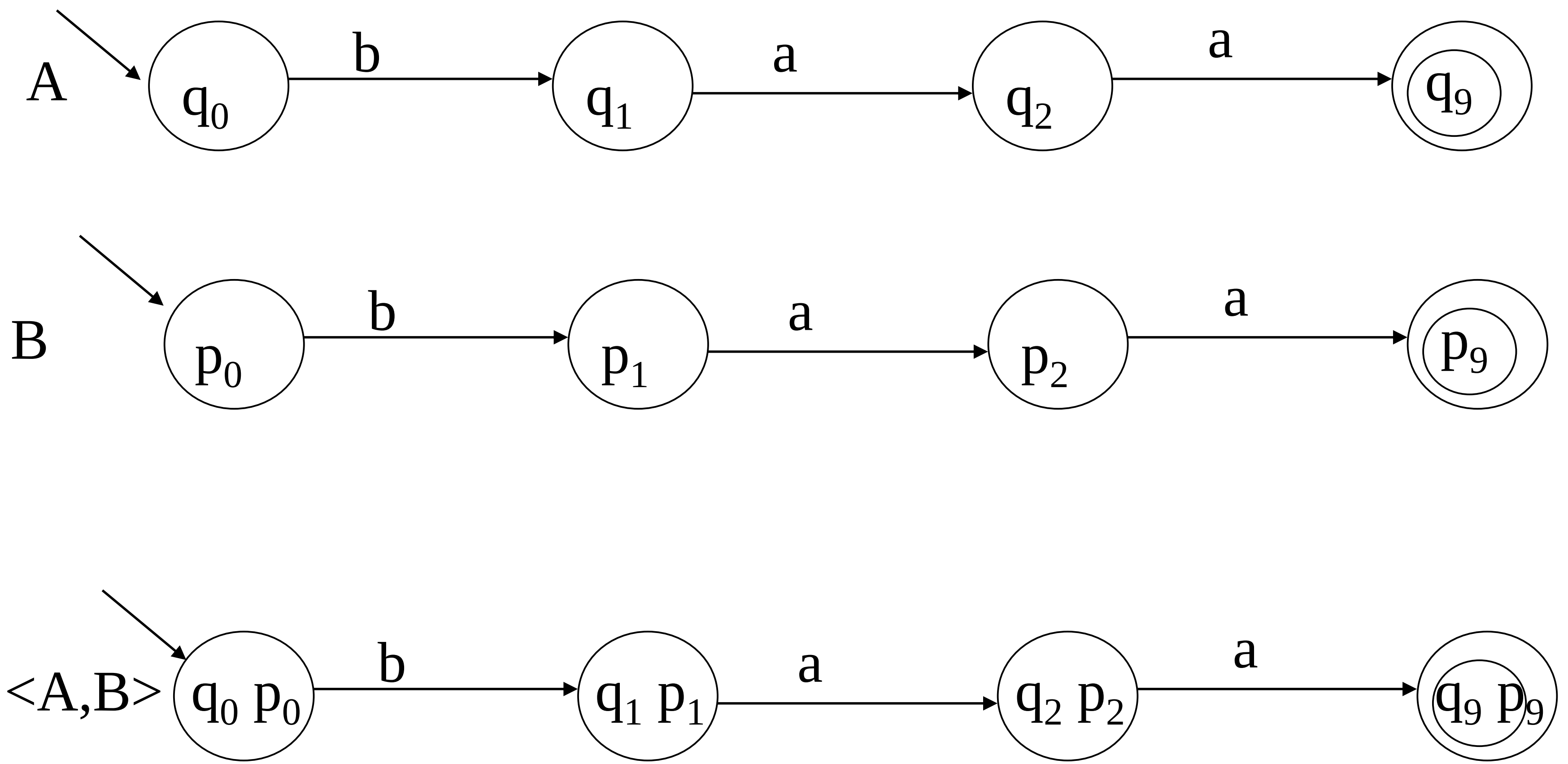
integers are closed under addition and multiplication

Closure for languages

- $\mathcal{L} = \{L_i\}$: **family** of languages
- $\mathcal{L}$ is closed w.r.t. operation OP if and only if, for every $L_1, L_2 \in \mathcal{L}$, L_1 OP $L_2 \in \mathcal{L}$.
- $\mathcal{R}$: **regular languages** (recognized by FSAs)
- $\mathcal{R}$ is closed w.r.t. set-theoretic operations, concatenation, "*", …

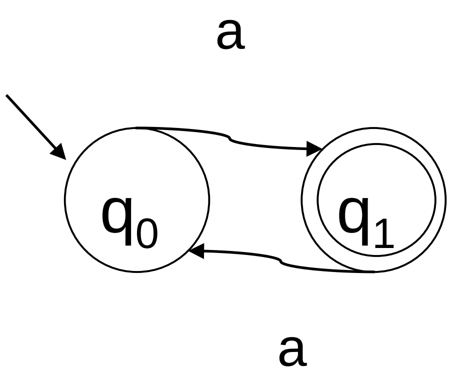
Intersection

The “**parallel run**” of A and B can be simulated by “coupling them”



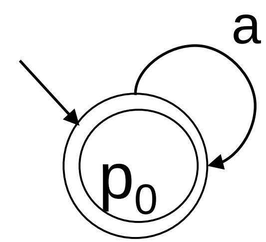
Example

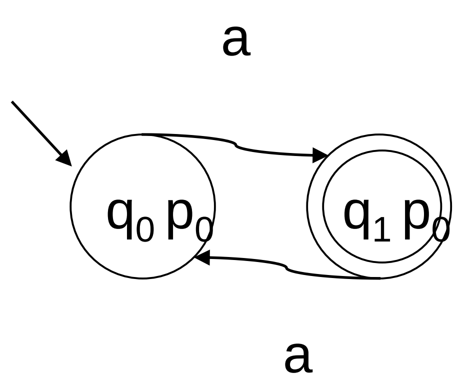
A^1:



$\cap$

A^2:





Formally

- Given
  - $A^1 = <Q^1, I, \delta^1, q_0^1, F^1>$
  - $A^2 = <Q^2, I, \delta^2, q_0^2, F^2>$

  $< A^1, A^2 > = <Q^1 \times Q^2, I, \delta, <q_0^1, q_0^2>, F^1 \times F^2 >$
  - $\boldsymbol{\delta(<q^1,q^2 >, i) = <\delta^1(q^1, i), \delta^2(q^2,i)>}$
- One can show (by simple induction) that

  $L(< A^1,A^2>) = L(A^1) \cap L(A^2)$
- Can we do the same for union?

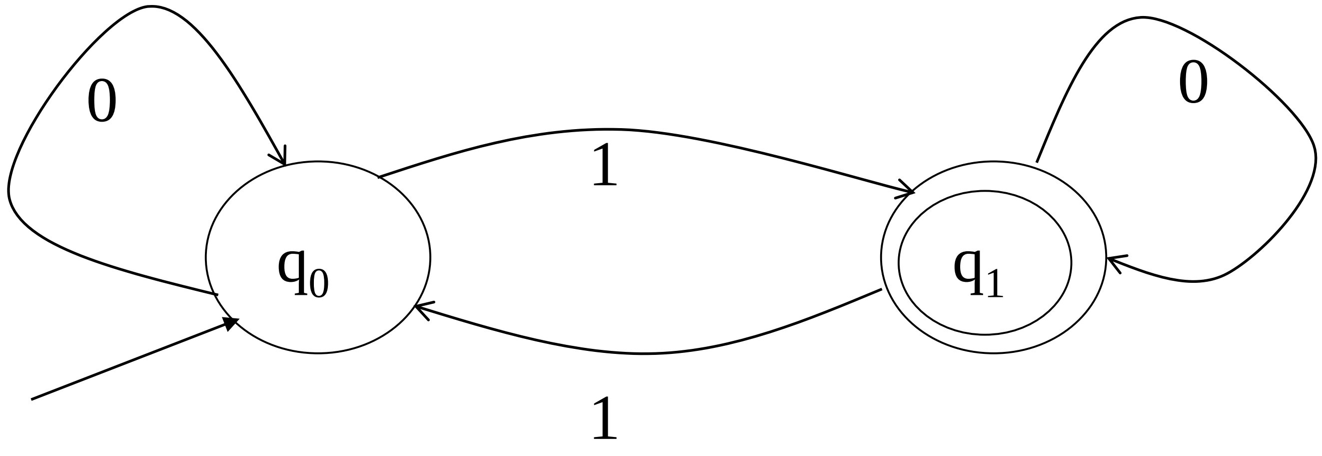
Union

- The union is built analogously
- Given
  - $A^1 = <Q^1, I, \delta^1, q_0{}^1, F^1>$
  - $A^2 = <Q^2, I, \delta^2, q_0{}^2, F^2>$

  $<A1, A2> = <Q^1 x Q^2, I, \delta, <q_0{}^1, q_0{}^2>, F^1 x Q^2 \cup Q^1 x F^2>$
  - $\delta(<q^1, q^2>, i) = <\delta^1(q^1, i), \delta^2(q^2, i)>$

Complement (1)

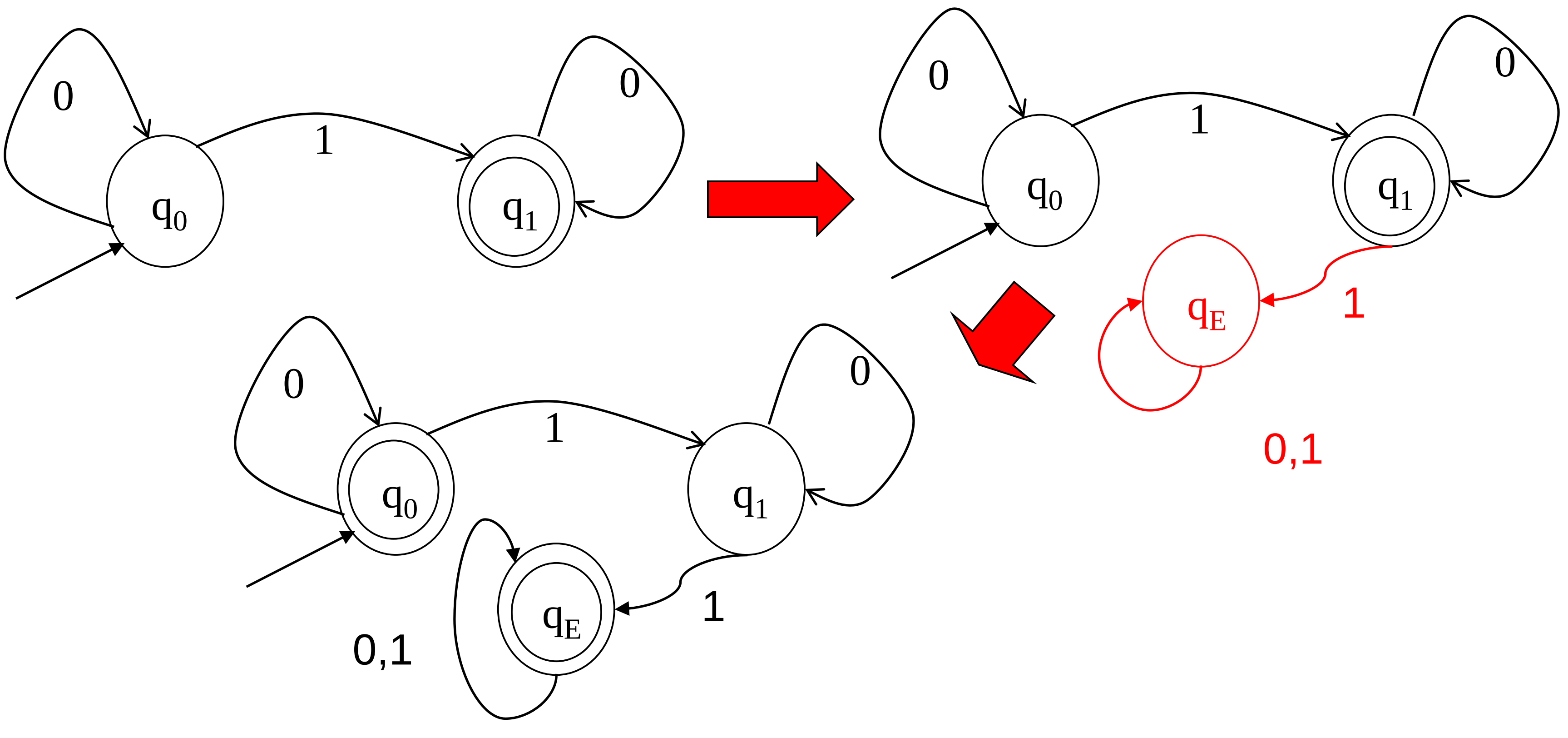
- Basic idea $F^c = Q\text{-}F$



Since the transition function may be partial this is not enough!

Complement (2)

- Before swapping final and non final states it is necessary to **<u>complete the FSA</u>**



Union again

- Another possibility is to use complement and **De Morgan's laws**:

  $A \cup B = \neg(\neg A \cap \neg B)$

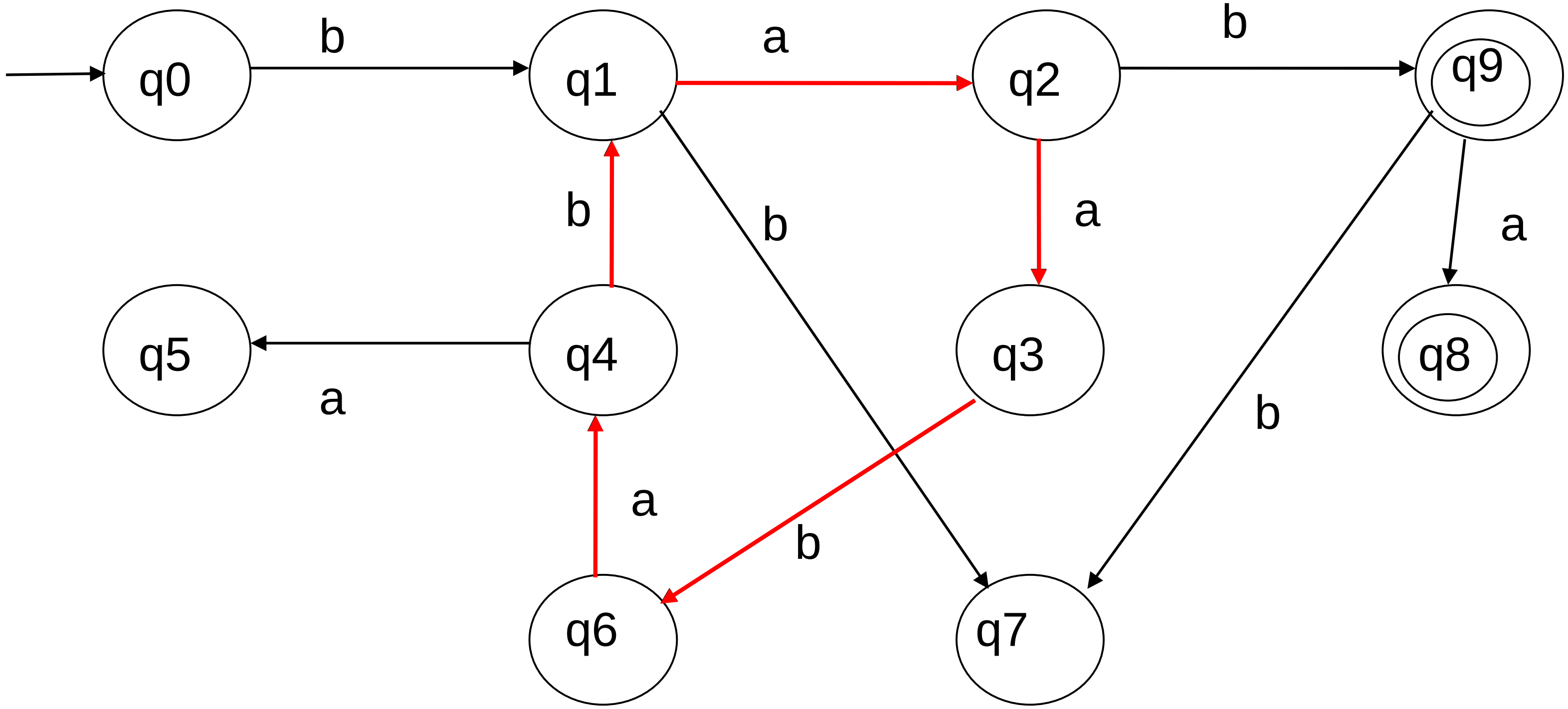
Philosophy of complement

- If I scan the entire input string, then it suffices to "swap yes and no" (F with Q-F)
- If I cannot reach the end of the string, then swapping F with Q-F does not work
- In the case of FSAs there is an easy workaround (completing the FSA)
- In general **we cannot consider the negative answer to a question as equivalent to the positive answer to the opposite question!**

PUMPING LEMMA

Cycles

There is a cycle: q1 ----aabab---> q1



If one goes trough the cycle once, then one can also go through it 2,3, …, n times

More formally

- If $x \in L$ and $|x| \geq |Q|$, then there exists a $q \in Q$ and a $w \in I^+$ such that:
  - $x = ywz$
  - $\delta^* (q,w) = q$

- Therefore the following also holds:
  - $yw^nz \in L, \forall n \geq 0$

This is the *Pumping Lemma* (one can "pump" w)

Consequences of pumping lemma

- $L = \emptyset$? $\exists\, x \in L \leftrightarrow \exists\, y \in L, |y| < |Q|$: Just "remove all cycles" from the FSA accepting x
- $|L| = \infty$? Check by a similar argument whether $\exists\, x \in L, |Q| <= |x| < 2|Q|$
- Note that *in general* knowing how to answer the question "$x \in L$?" for a generic x, does *not* entail knowing how to answer the other questions
  - It works for FSAs, but…

Impact in practice

- Are we interested in a programming language consisting of… 0 correct programs?
- Are we interested in a programming language in which one can only write a finite number of programs?
- …

A negative consequence of pumping lemma

- Is the language $L = \{a^n b^n \mid n > 0\}$ recognized by some FSA?
- Let us suppose it is. Then:
- Consider $x = a^m b^m$, $m > |Q|$ and let us apply P.L.
- Possible cases:
  - $x = ywz$, $w = a^k$, $k > 0$ ====> $a^{m-k} a^{r \cdot k} b^m \in L$, $\forall r$: NO
  - $x = ywz$, $w = b^k$, $k > 0$ ====> same
  - $x = ywz$, $w = a^k b^s$, $k, s > 0$ ====> $a^{m-k} a^{rk} b^{rs} b^{m-s} \in L$ $\forall r$: NO

Intuitively

- In order to "count" an arbitrary *n* we need an infinite memory!
- Rigorously speaking, every computer is an FSA, but… it is the wrong abstraction: intractable number of states!

  (same thing as studying every single molecule in the flight of an airplane)
- Importance of an **abstract notion of infinity**
- From the toy example $\{a^n b^n\}$ to more concrete cases:
  - Checking well-balancing of brackets (typically used in programming languages) cannot be done with finite memory
- We therefore need more powerful models (**PDA**)