Theoretical Computer Science

Administrative Information

Lecture 5 - Manuel Mazzara

Mid-term Exam

When : Thursday, 4 March 2021 09:30-~11:30

Where : 108 (here) and possibly another room

+ online for abroad students only

What:

Formal Languages, FSA, Pumping Lemma, PDA (only in part)

Assignment 1

FSA Design

Design and drawing exercise where you have to demonstrate an understanding of the basic principles and functioning of FSAs

In Moodle – please respect the deadline!

There will be Assignment 2 around Week 9 (release) and 10 (submission)

Theoretical Computer Science

Regular Languages and closure

Lecture 5 - Manuel Mazzara

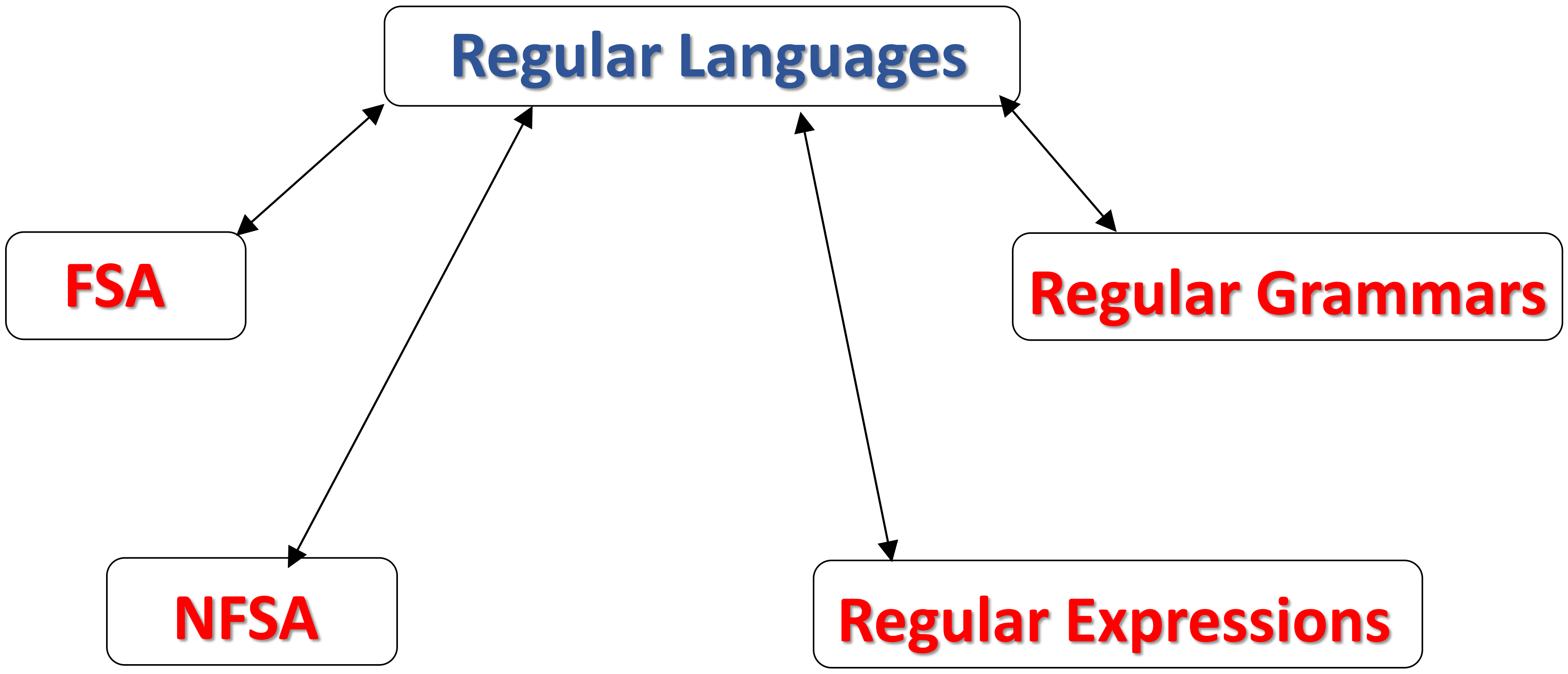
Regular Languages

- A **regular language** is a language recognized by a FSA

- Regular languages are very useful in **input parsing and programming language design**

- We will see models that are equivalent to languages recognized by FSA
  - **Regular expressions**
  - Specific type of **generative grammars**

Representations of Regular Languages



Closure for languages

- $\mathcal{L}$ = $\{L_i\}$: **<u>family</u>** 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)
- **<u>$\mathcal{R}$ is closed w.r.t. set-theoretic operations, concatenation and "*"</u>**

Theoretical Computer Science

Pumping Lemma for Regular Languages

Lecture 5 - Manuel Mazzara

Representations of Regular Languages

- We are given a **Regular Language** L
- It means that Language L comes in one of the standard representations:
  - **Deterministic Finite State Automata**
  - Nondeterministic Finite State Automata
  - Regular expressions
  - Regular grammars

All finite languages are regular

You can easily build an FSA to recognize a finite language!

Some infinite languages are regular – not all

There are languages that are not regular

What does it mean?

These languages cannot be represented in any of these standard representations

Examples of non regular languages

$$\{a^n b^n : \quad n \geq 0\}$$

$$\{ww^R : \quad w \in \{a,b\}^*\}$$

- **How can we prove that a language is not regular?**

- Can we prove that there is no FSA that accepts it?

- This is not easy to prove! How would you prove it?

Proving that a language is not regular

- To prove that a language is regular, we can just find an FSA (or another standard representation) for it
- To prove that a language is **not** regular, we need to prove that there is no possible FSA (or another standard representation) for it
- This is possible in principle. For each specific language, a technical proof with ad-hoc argument could be built
- Here, however, we will show something more powerful: a **generalization**

Such generalization is known with the name of…

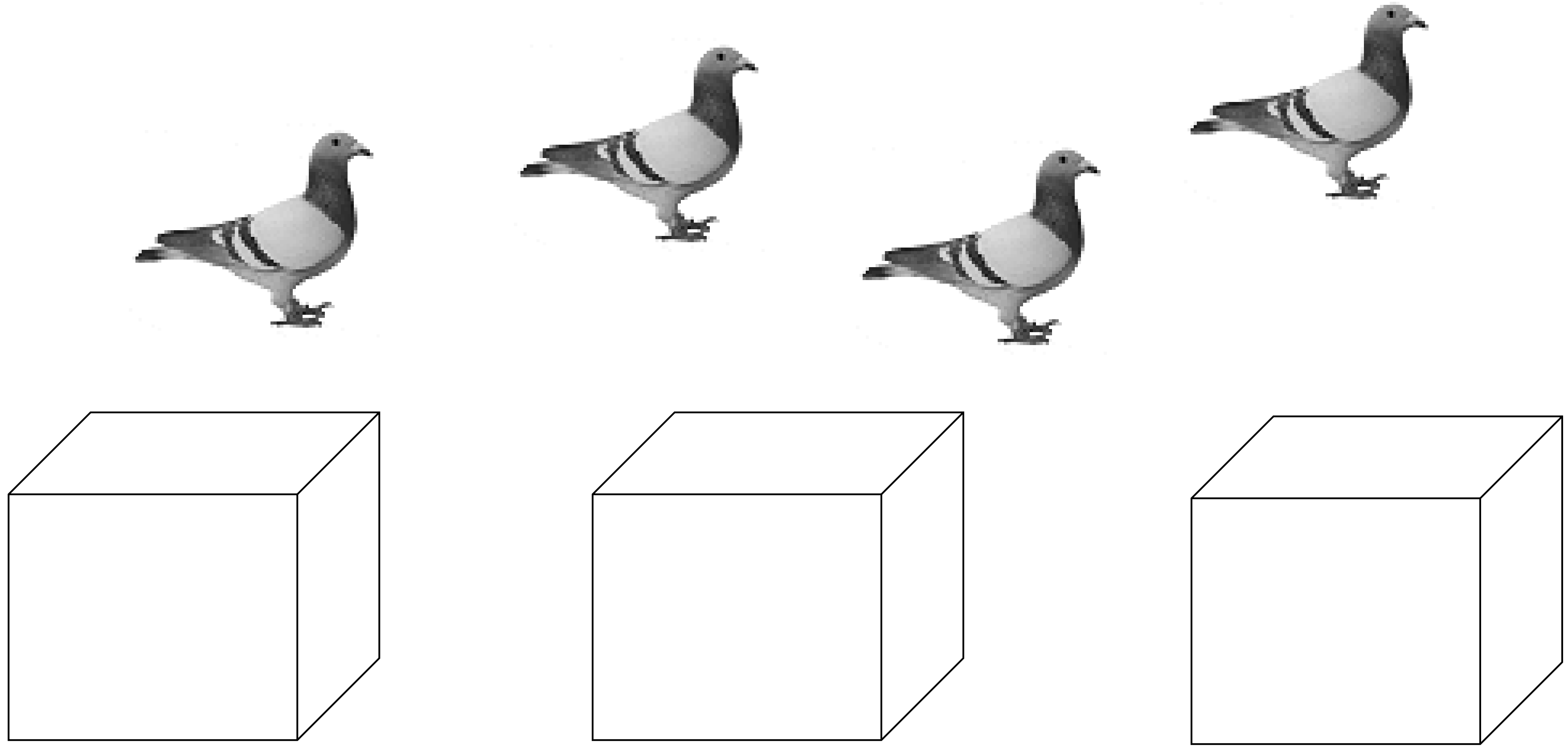
Pumping Lemma


ONE DOES NOT SIMPLY
UNDERSTAND PUMPING LEMMA
ator.net


KEEP
IT
SIMPLE

The Pigeonhole Principle (1)

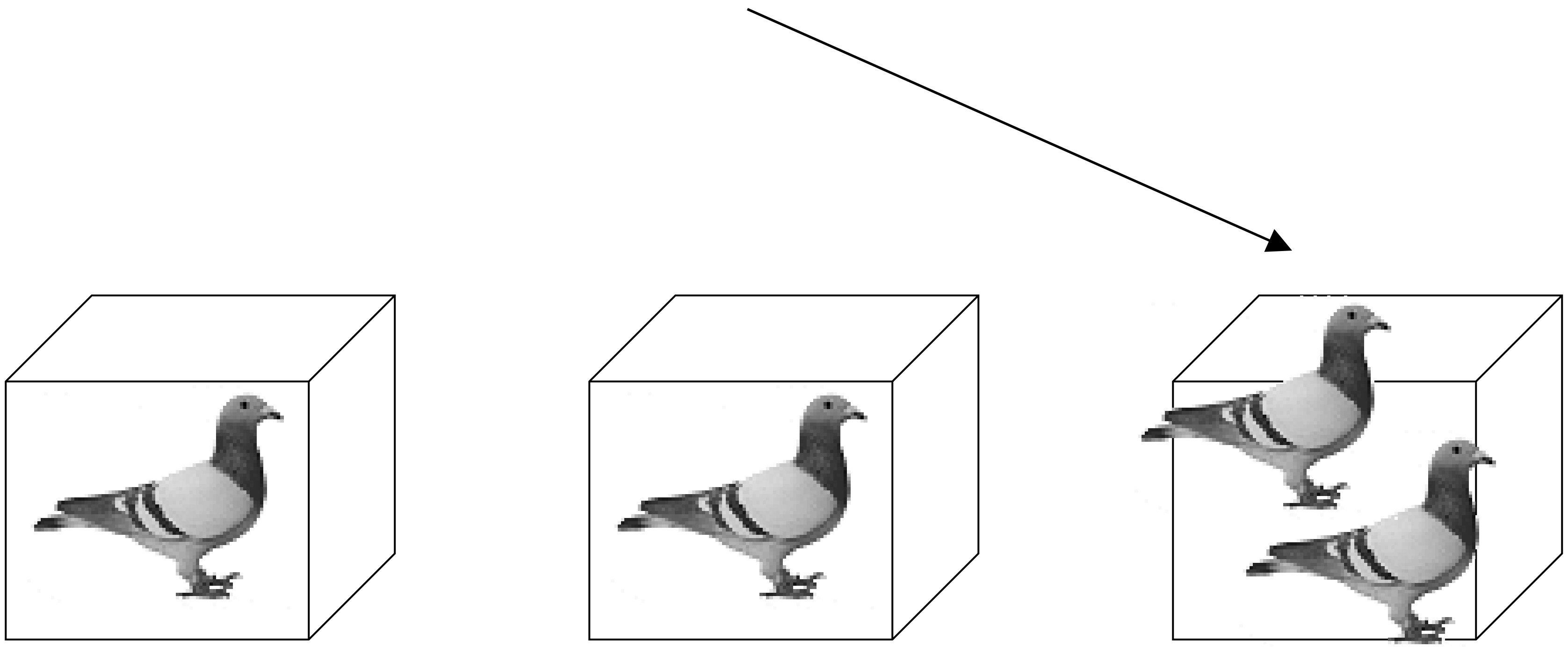
4 pigeons



3 pigeonholes

The Pigeonhole Principle (2)

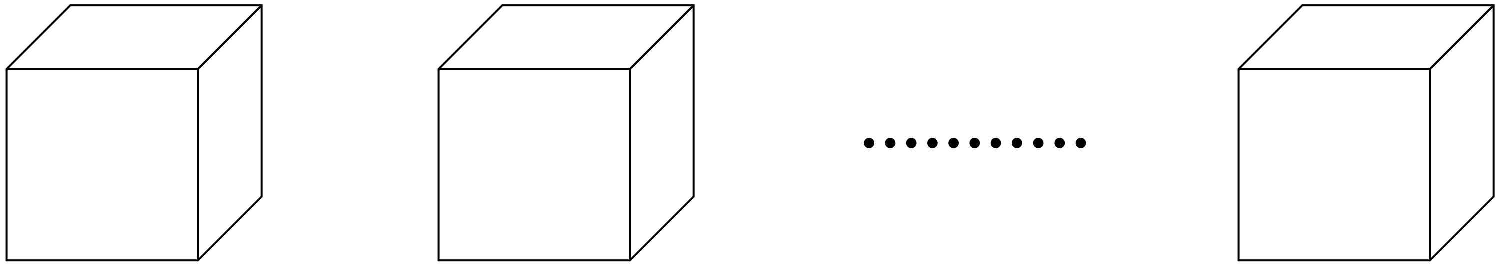
A pigeonhole must
contain at least two pigeons



Generalization

n pigeons

m pigeonholes $n > m$



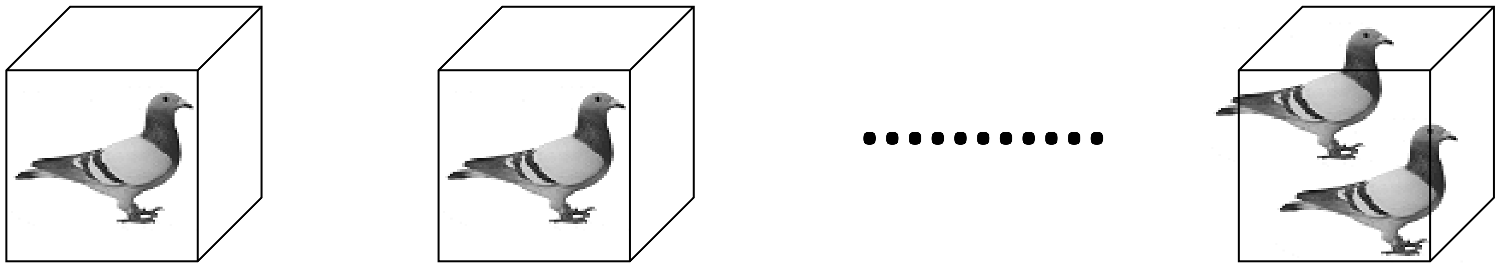
Straightforward results

n pigeons

m pigeonholes

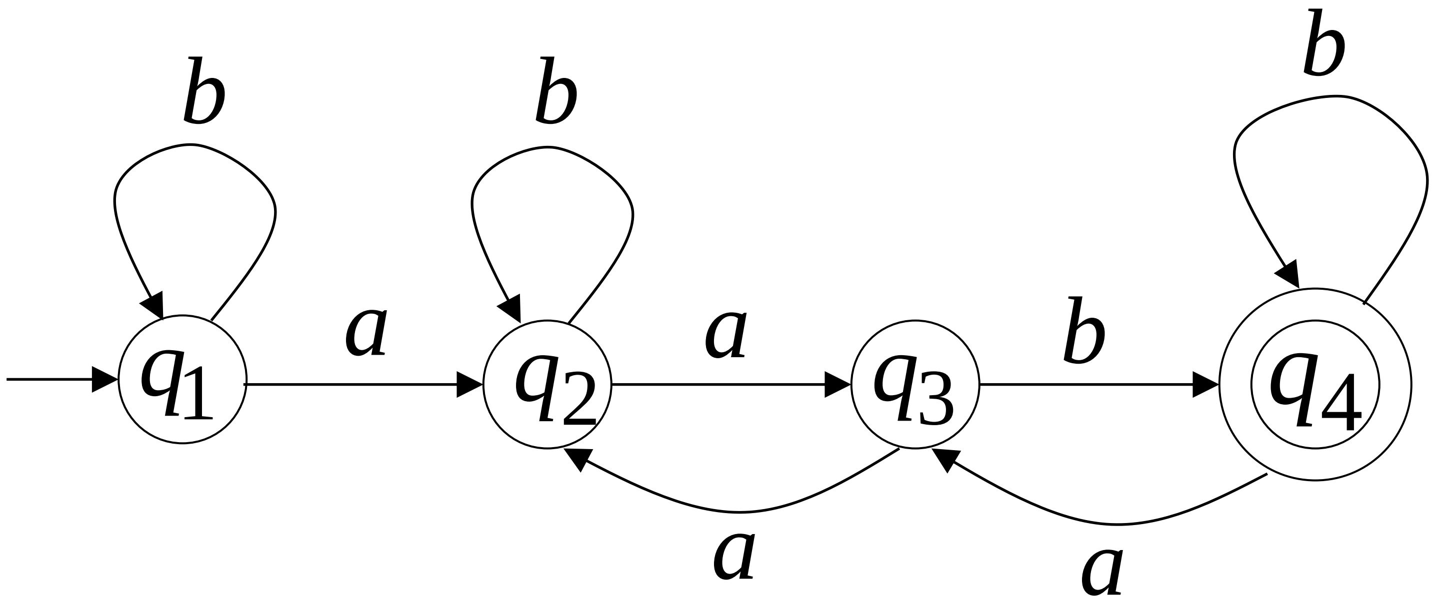
$n > m$

There is a pigeonhole with at least 2 pigeons



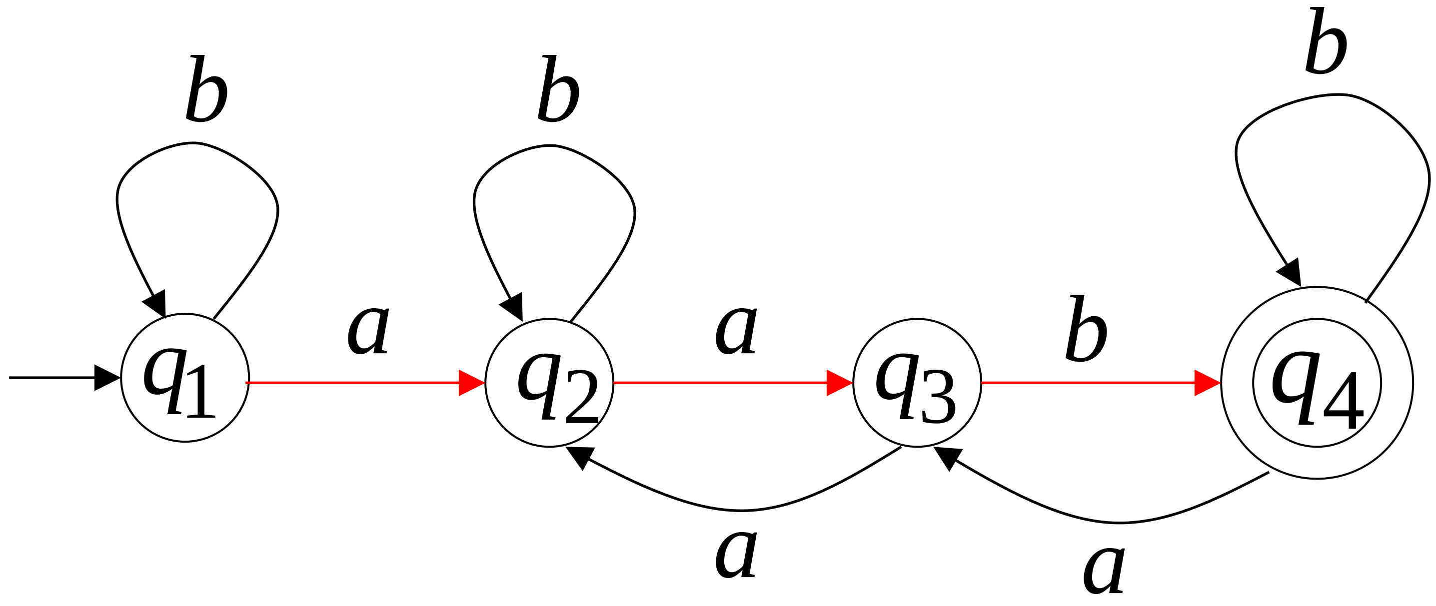
Example (1)

FSA with 4 states



Example (2)

The path *aab* has no repetition of state



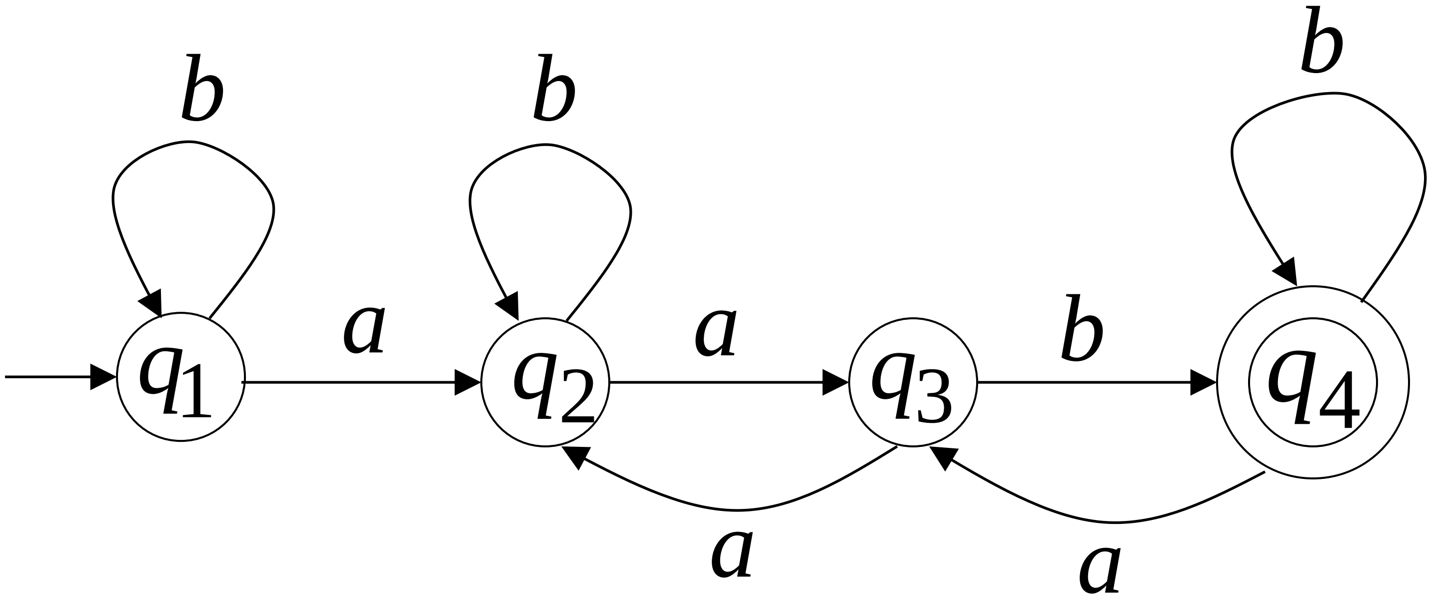
States can be repeated

$aabb$

$bbaa$

$abbabb$

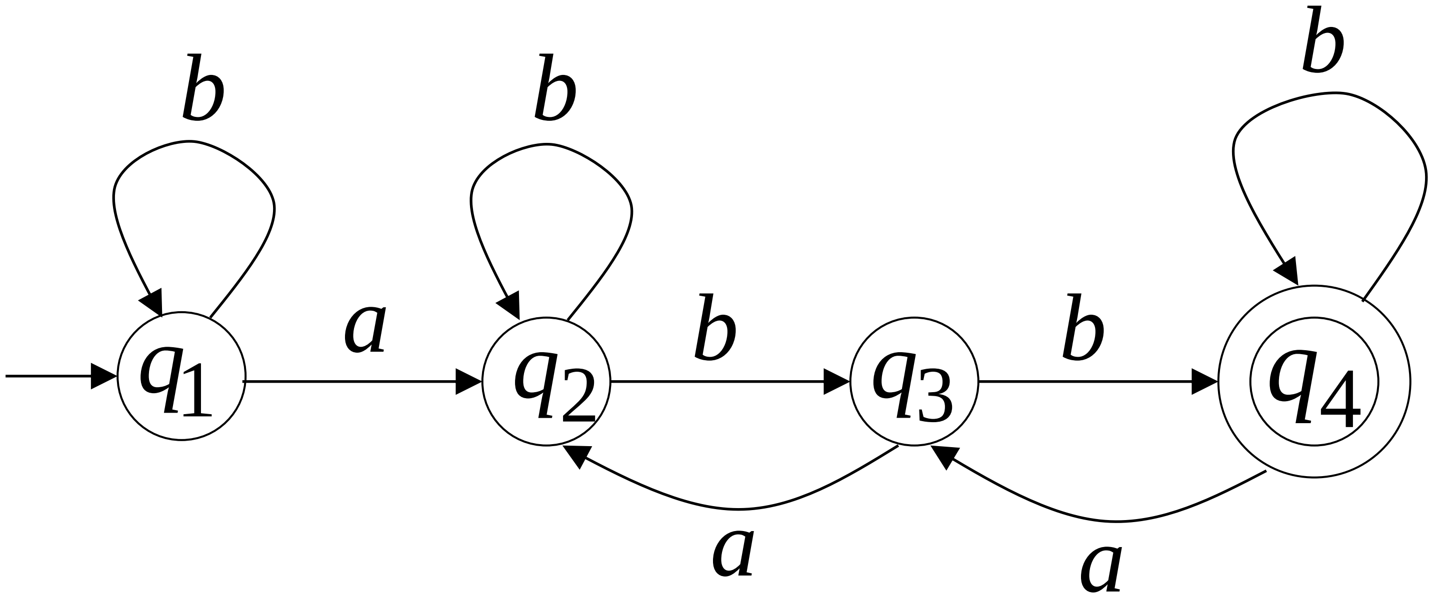
$abbbabbabb...$



Simple fact

If the path for string x is greater than 4

then a state must be repeated



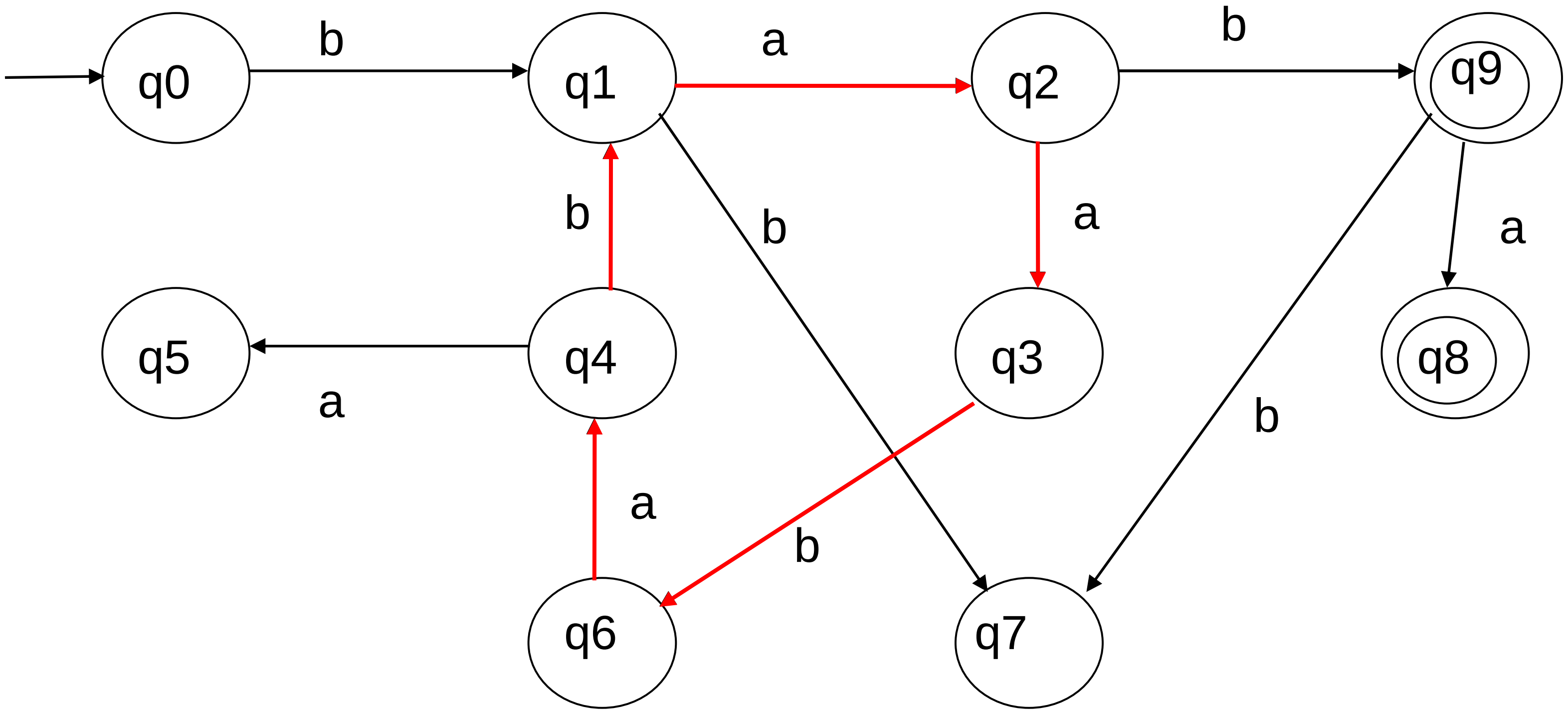
Anyone called me?

FSA and Pigeonhole Principle

- If in a path $\boxed{\text{transitions} \geq \text{states}}$ then a state is **repeated**
- Transitions are pigeons
- States are pigeonholes
- **If a string has length greater than the number of states, there has to be a repeated state in the visit**

Cycles

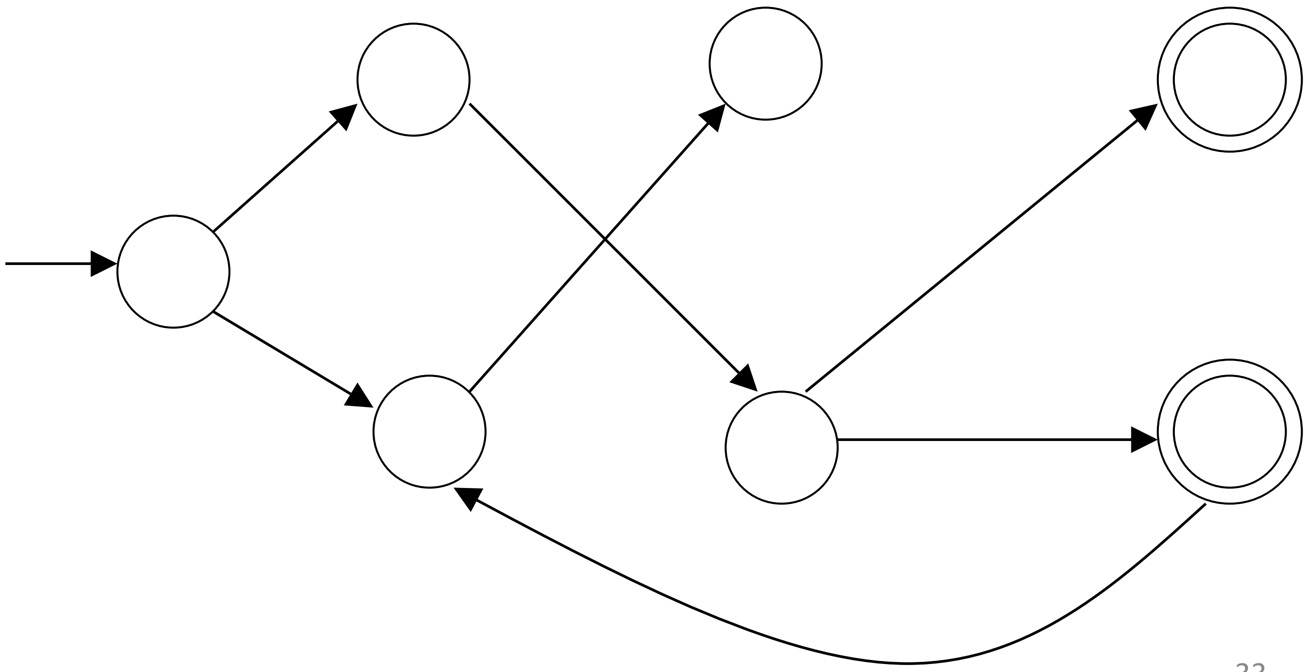
There is a cycle: q1 ----aabab---> q1



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

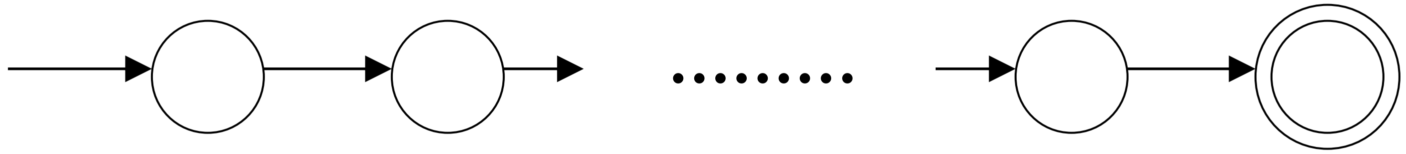
Languages and Cycles (1)

- Consider an **infinite** regular language
- The FSA recognizing it has *m* states

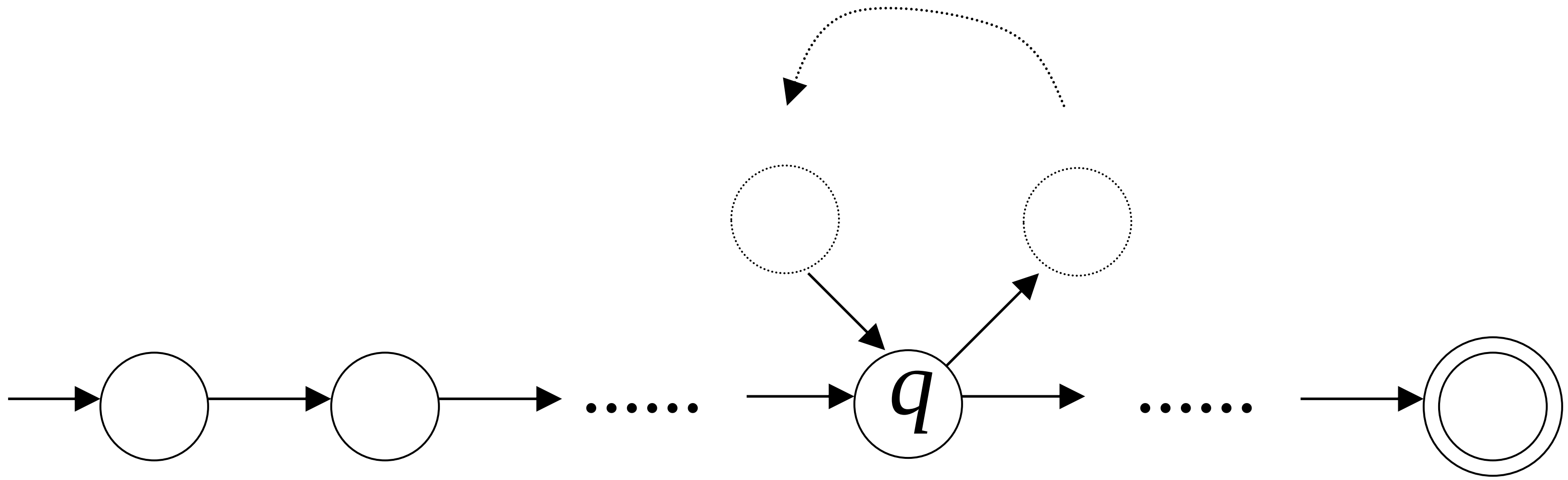


Languages and Cycles (2)

Take string x belonging to L and with length greater than m



A state has to be repeated in the walk



Pumping Lemma for Regular Languages

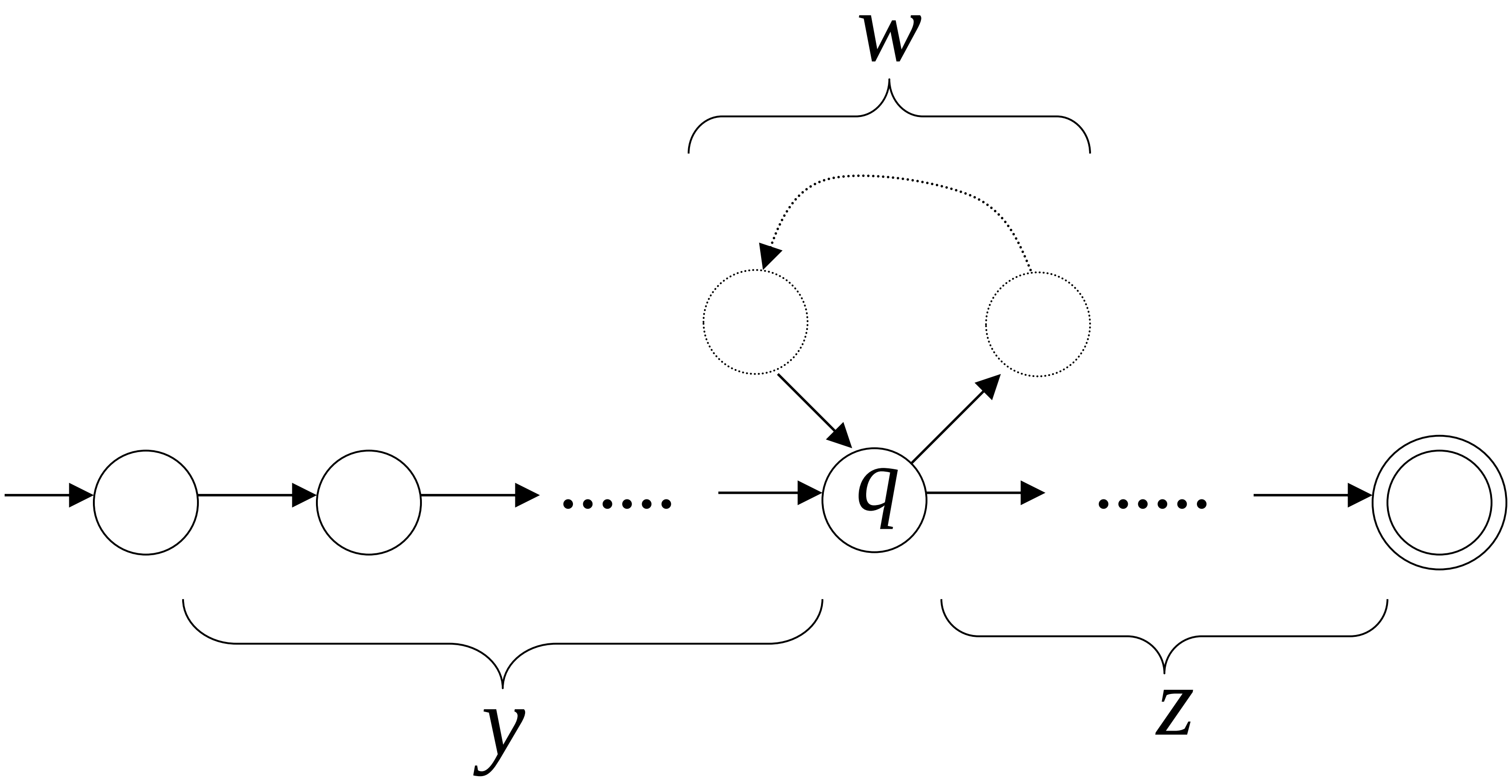
- It is an essential property of **all regular languages**
- All **sufficiently long words** in a regular language may be *pumped*, i.e. have a middle section of the word repeated an arbitrary number of times, **to produce a new word that also belong to the same language**
- We will not give the proof here, just the intuition
- **The pumping lemma is used to prove that a particular language is non-regular**
    - You will see a few examples in the tutorial and lab sessions

Formally

- If $\mathbf{x} \in \mathbf{L}$ and $|\mathbf{x}| \geq |\mathbf{Q}|$, then there exists a $q \in Q$ and a $\mathbf{w} \in I^+$ such that:
  - $x = ywz$
  - $\delta^* (q_0,y) = q$
  - $\delta^* (q,z) = q' \in F$
  - $\delta^* (q,w) = q$
  - $yw^nz \in L, \forall n \geq 0$

- This is the *Pumping Lemma* (one can “pump” **w**)

Pumping Lemma, graphically



The “Pigeonhole” principle

- If *p* pigeons are placed into fewer than *p* holes, some holes must hold more than one pigeon

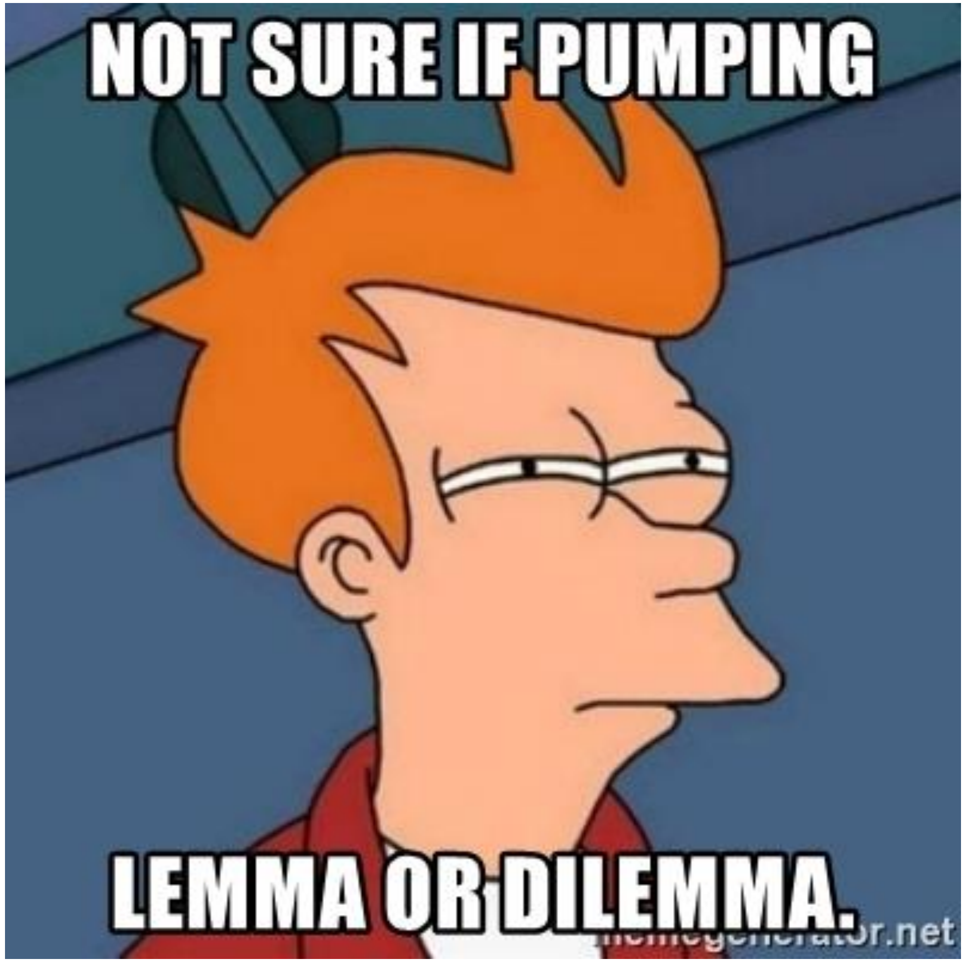
- The sequence of states traversed during the recognition of a string **must have a repeated state** (in our formal definition we called it **q**)

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 the Pumping Lemma
- Possible cases:
  - $x = ywz$, $w = a^k$, $k > 0$ ====> $a^{m-k} a^{(k*r)} b^m \in L, \forall r$: NO
  - $x = ywz$, $w = b^k$, $k > 0$ ====> $a^m b^{(k*r)} b^{m-k} \in L, \forall r$: NO
  - $x = ywz$, $w = a^k b^s$, $k, s > 0$ ====> $a^{m-k} (a^k b^s)^r b^{m-s} \in L \; \forall r$: NO

Levels of expressiveness

- In order to “count” an arbitrary *n* we need an infinite memory!
- **Fixed vs finite**
  - **FSA and regular languages are about fixed memory**
- 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 fixed memory
- We therefore need more powerful models (**PDA**)

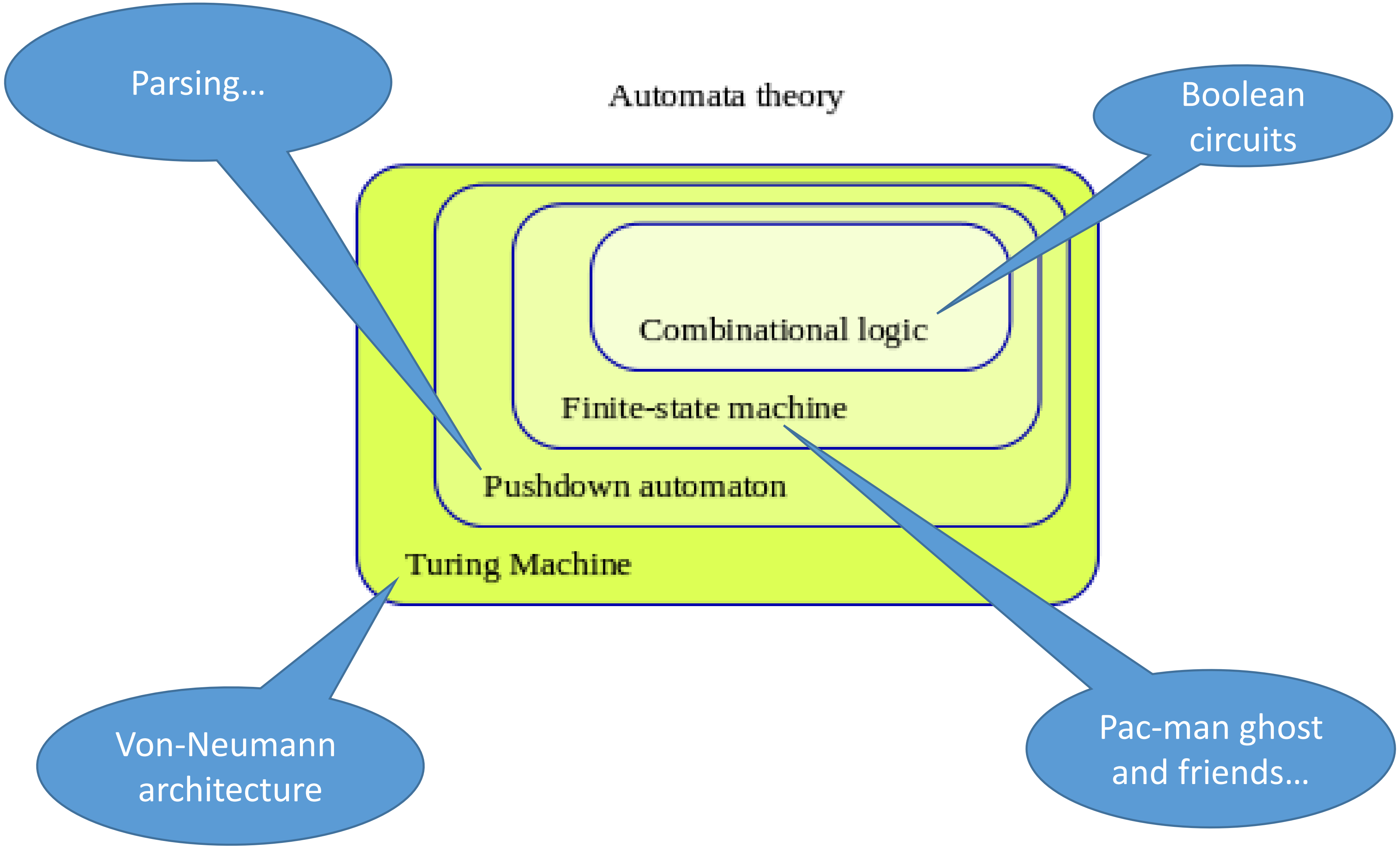

NOT SURE IF PUMPING
LEMMA OR DILEMMA.

Theoretical Computer Science

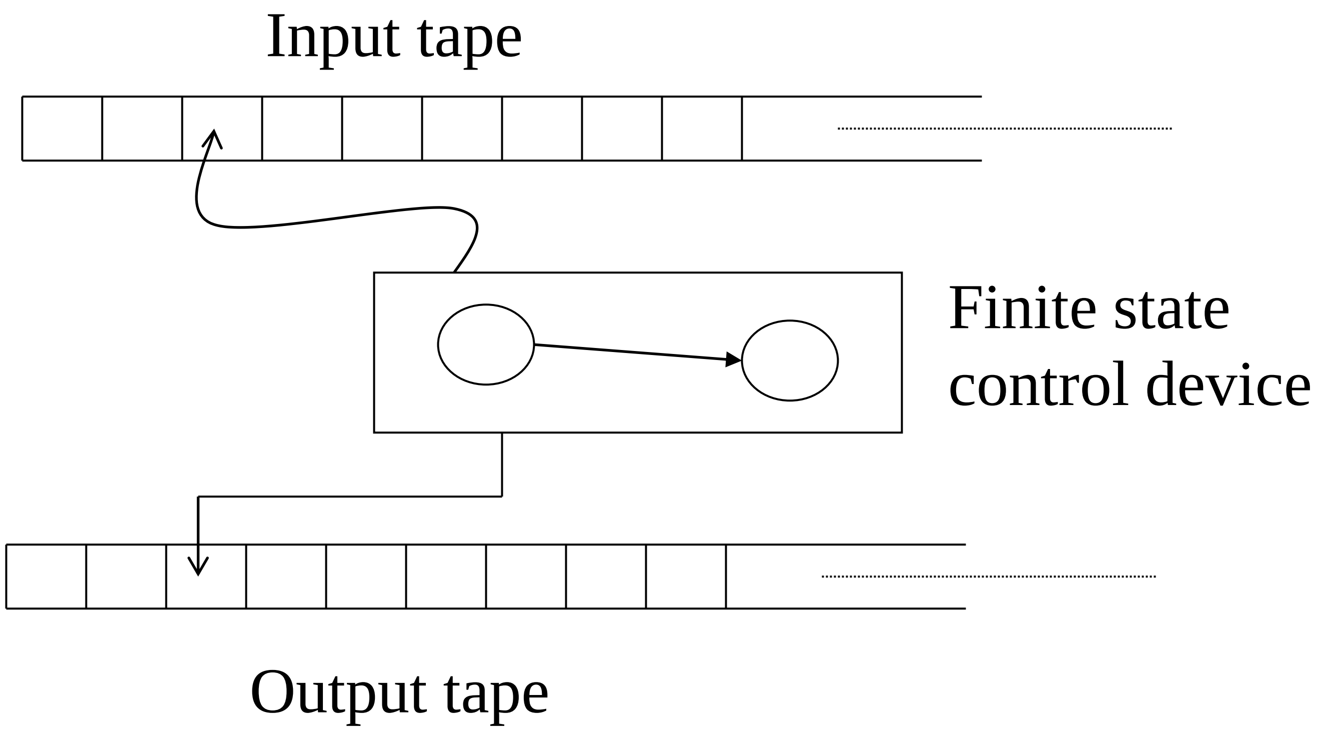
Introduction to Pushdown Automata

Lecture 5 - Manuel Mazzara

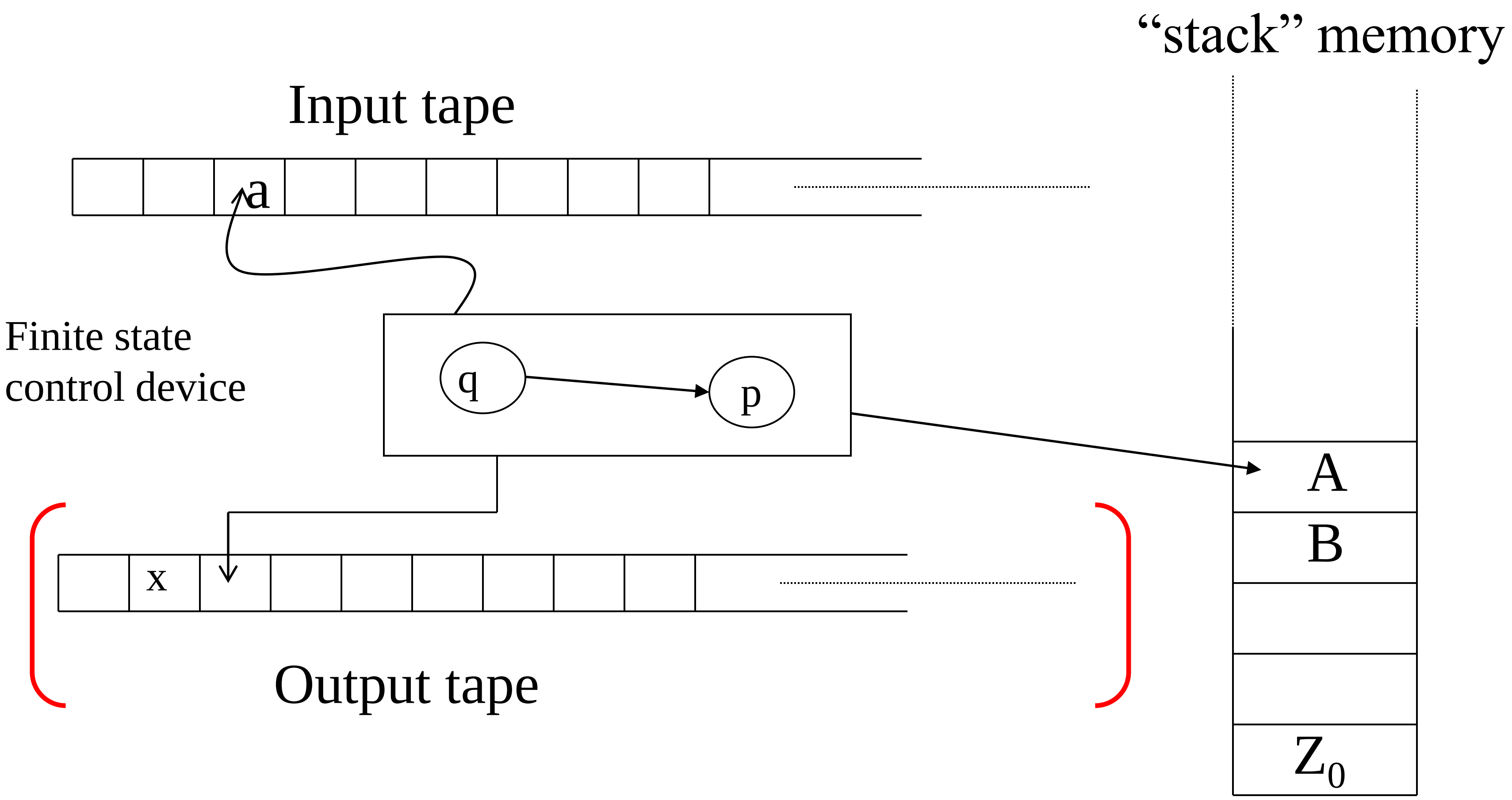
A bit of context



The mechanical view of FSAs

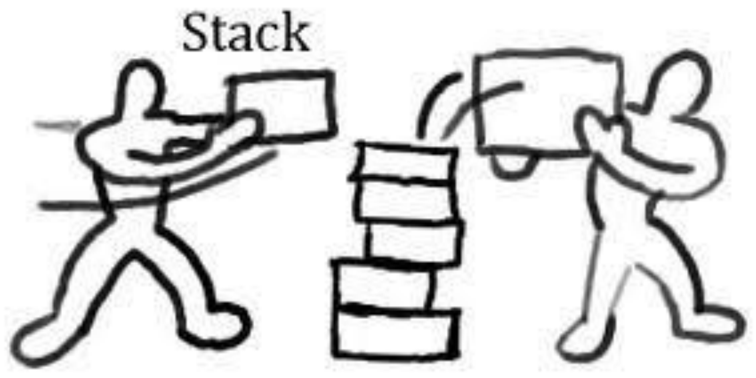


Adding a (destructive) external memory



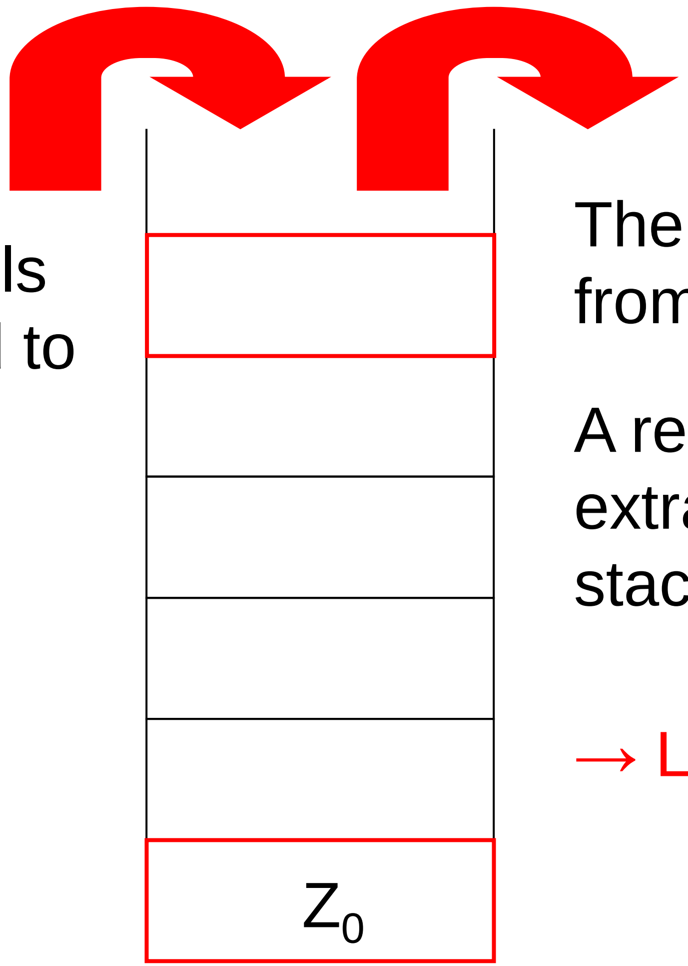
What is a stack?

- Everyone should know at this point!
- If not “Learn stack the fun way (data structure)”:
    - https://www.youtube.com/watch?v=PRHZoY2YE2U



What is a stack (boring way)

New symbols are inserted to the top



The stack is read from the top

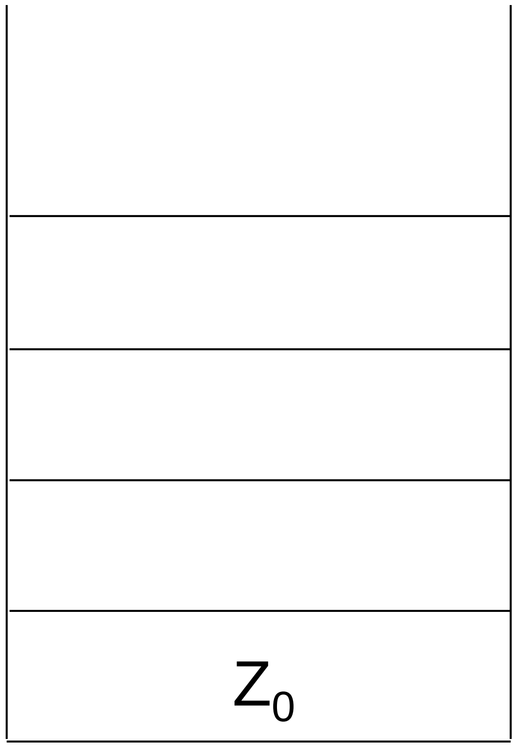
A read symbol is extracted from the stack

→ LIFO policy

Example

Insert in the following order the symbols

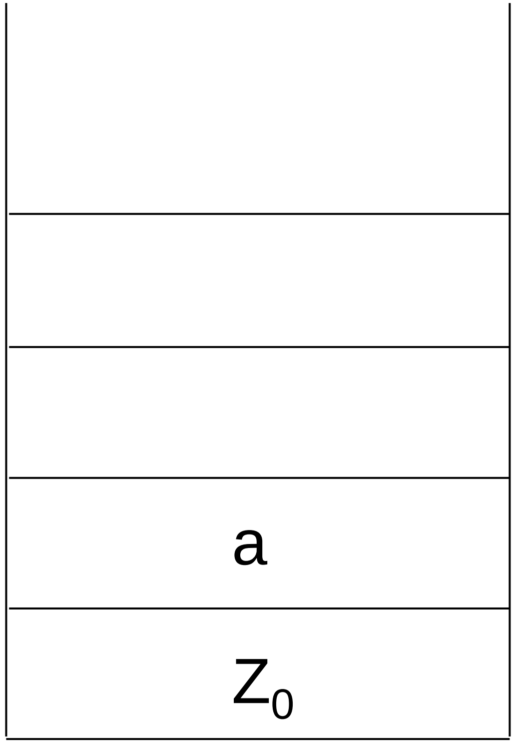
- a
- b
- c



Example

Insert in the following order the symbols

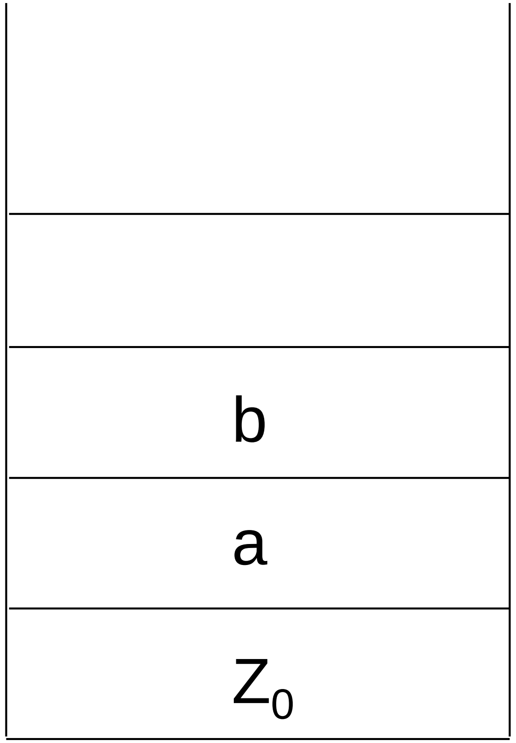
- a
- b
- c



Example

Insert in the following order the symbols

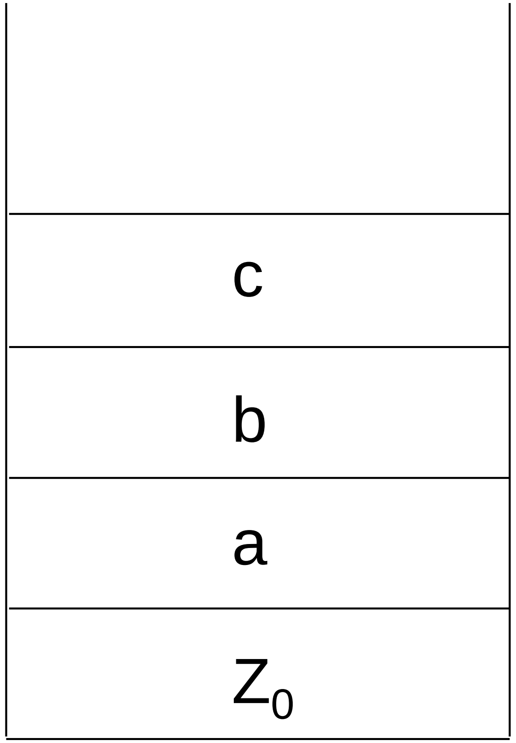
- a
- b
- c



Example

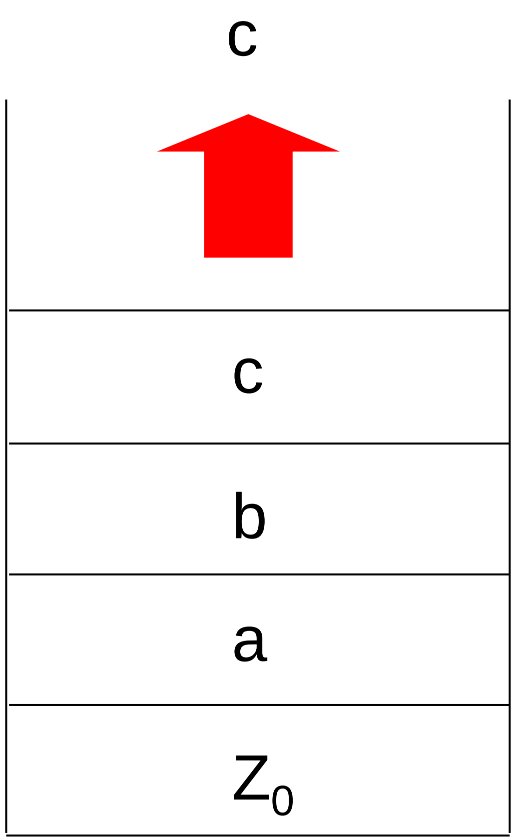
Insert in the following order the symbols

- a
- b
- c



Example

Insert in the following order the symbols
- a
- b
- c



Read from the stack

Trying to tell something new…

- Who invented the stack idea?
- **Alan Turing**'s 1946 paper on the Automatic Computing Engine (ACE), the first electronic computer developed in UK
- “*To arrange for the splitting up of operations into subsidiary operations*“
- Theory of **subroutines**
- Two instructions: BURY and UNBURY

Pushdown automata

- Finite state automata can be enriched with a stack

  → Pushdown Automata (PDA)
- PDAs differ from finite state machines in two ways:
  - They can **use the top of the stack to decide which transition has to be made**
  - They can **manipulate the stack** as part of performing a transition

Moves of a PDA

Depending on

- the symbol read from the input (but it could also read nothing)
- the symbol read from the top of the stack
- the state of the control device

the PDA

- *changes its state*
- *moves ahead the scanning head*
- *changes the symbol read from the stack with a string α (possibly empty)*

Of course, we will formally dissect this kind of automata and understand where they are useful