Theoretical Computer Science

Abstractions in context

Lecture 4 - Manuel Mazzara

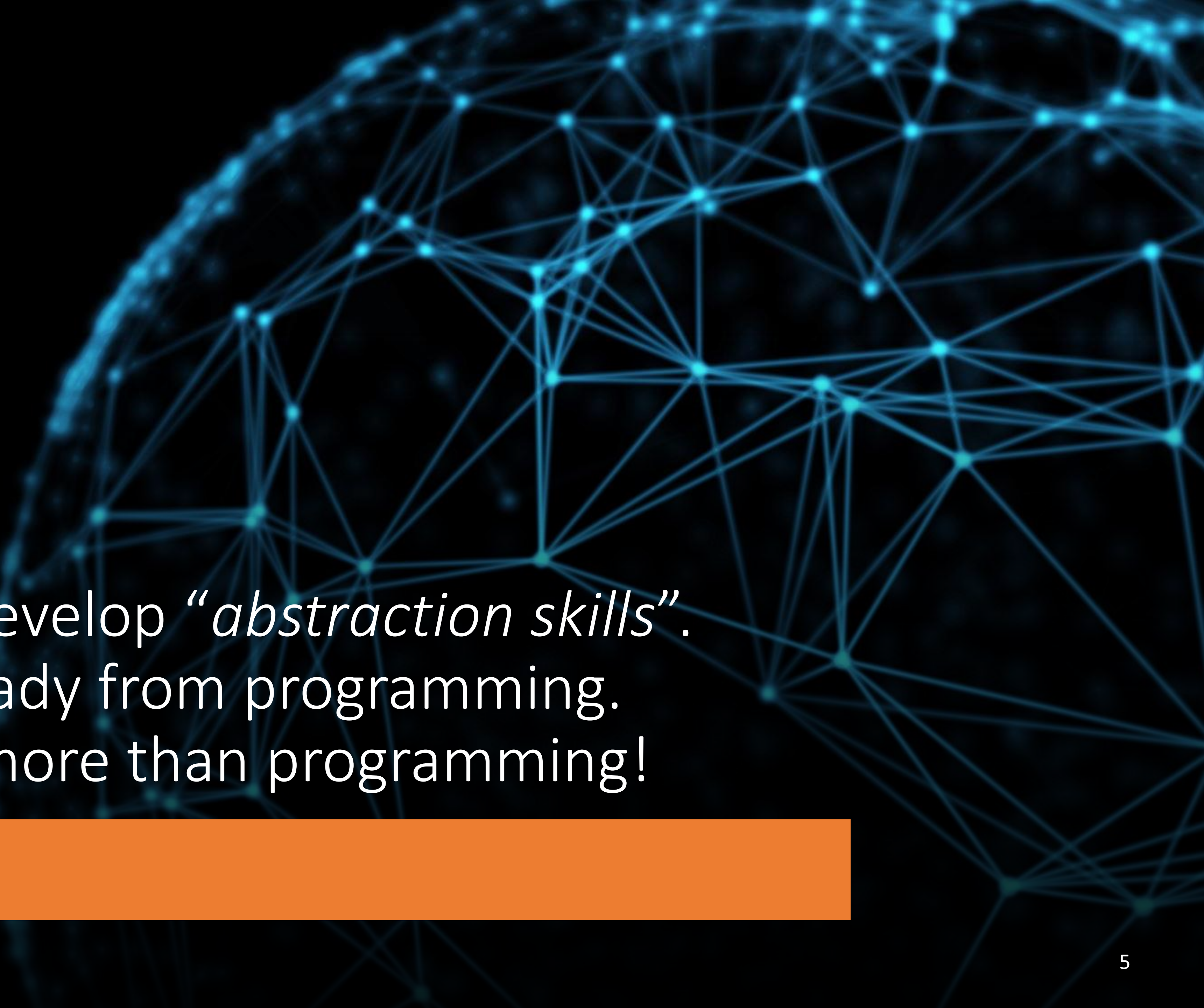
Let us go deeper for some key points!

- Hierarchy of expressiveness of automata
- Alphabet, language
- Pacman and Finite State Automata
- **Informal vs. Formal**
- **FSA, formally**
- **Recognizing Pascal identifiers**
- Finite State Transducers

Informal vs. Formal

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

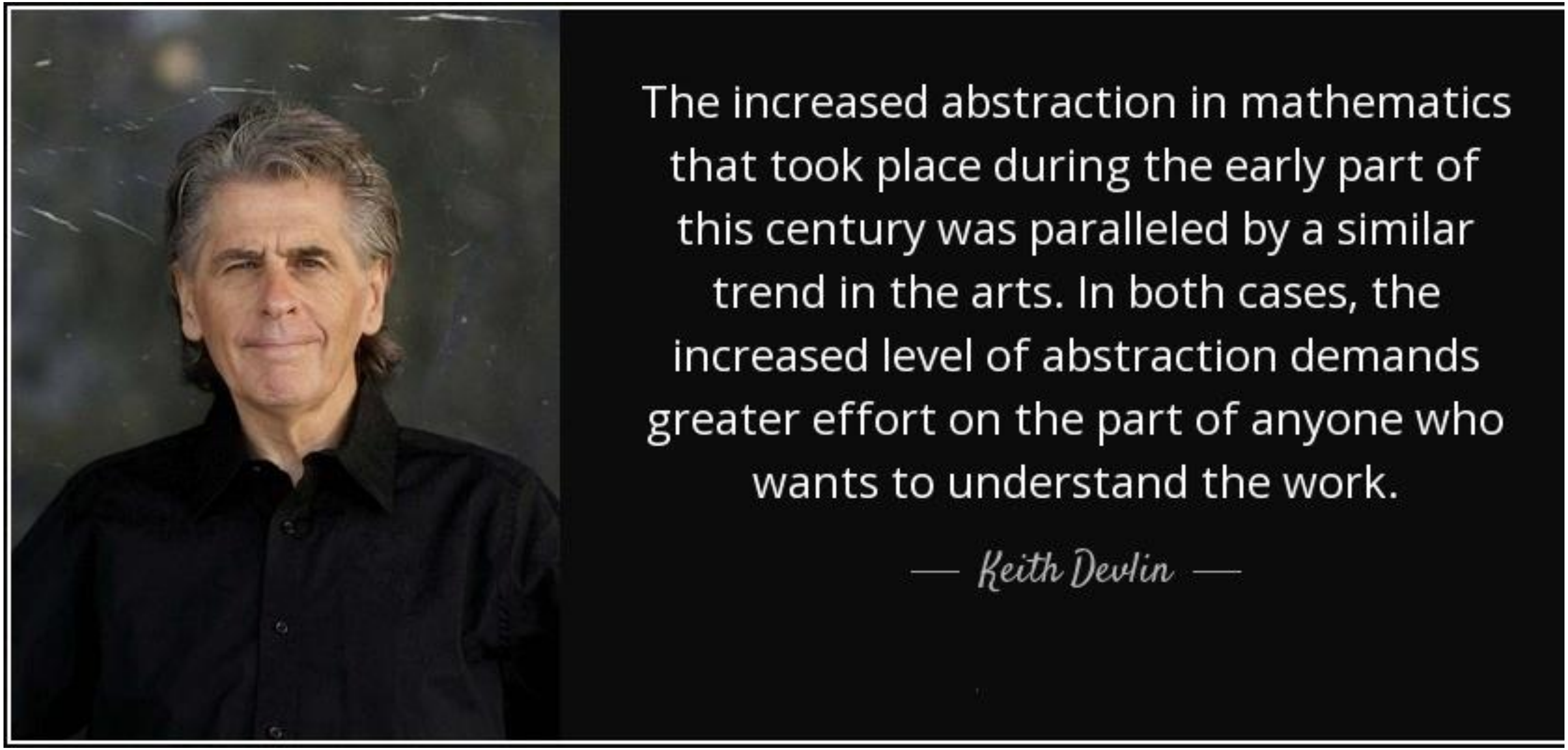
The ability to “dance” between formal and informal is a key skill of Computer Scientists and Software Engineers



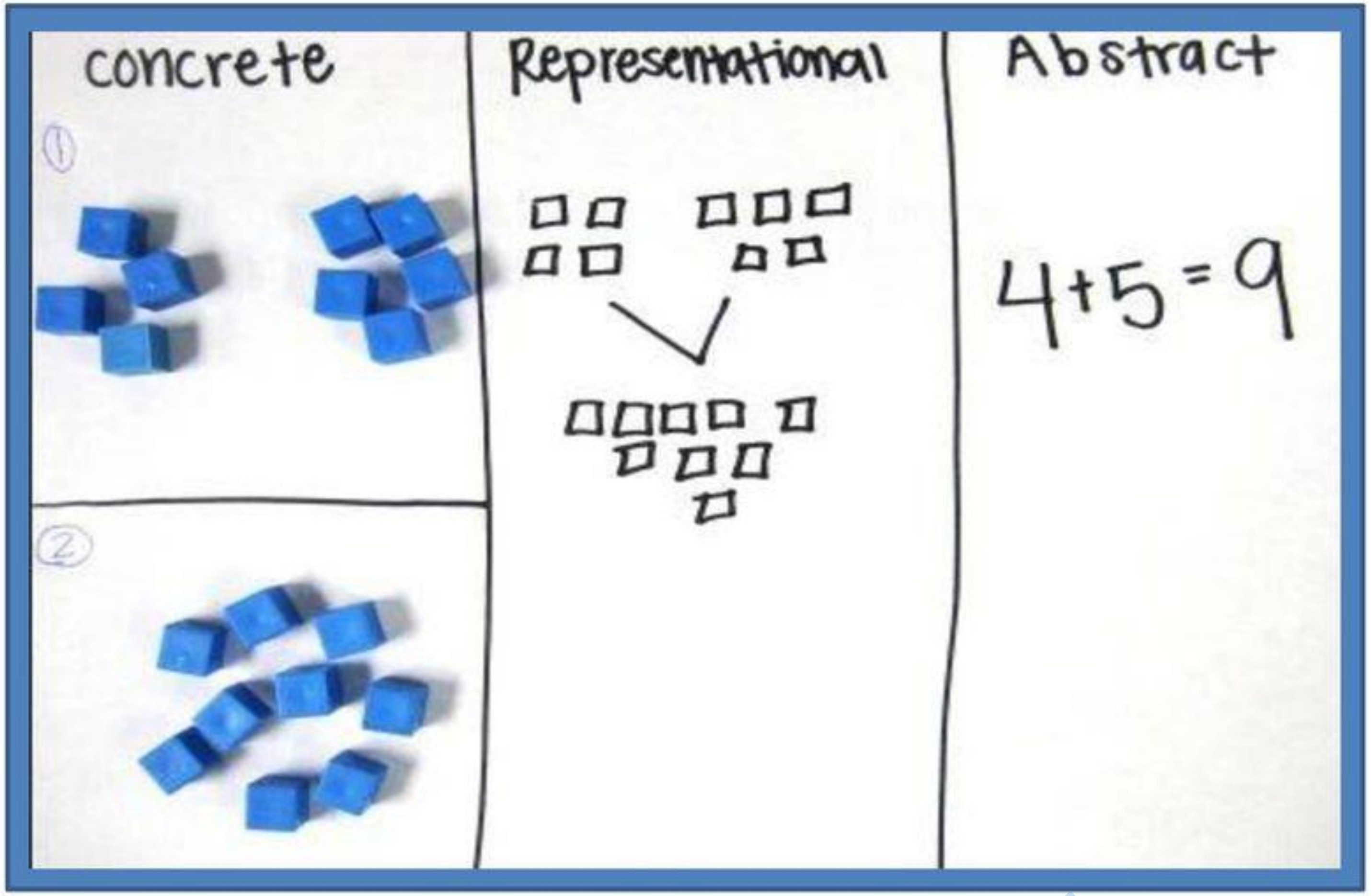
It is important to develop “*abstraction skills*”.
This you know already from programming.
However, there is more than programming!

Do you remember our discussion on abstractions?

From this course you should leave *at least* with an enhanced ability to **understand and build abstractions**!

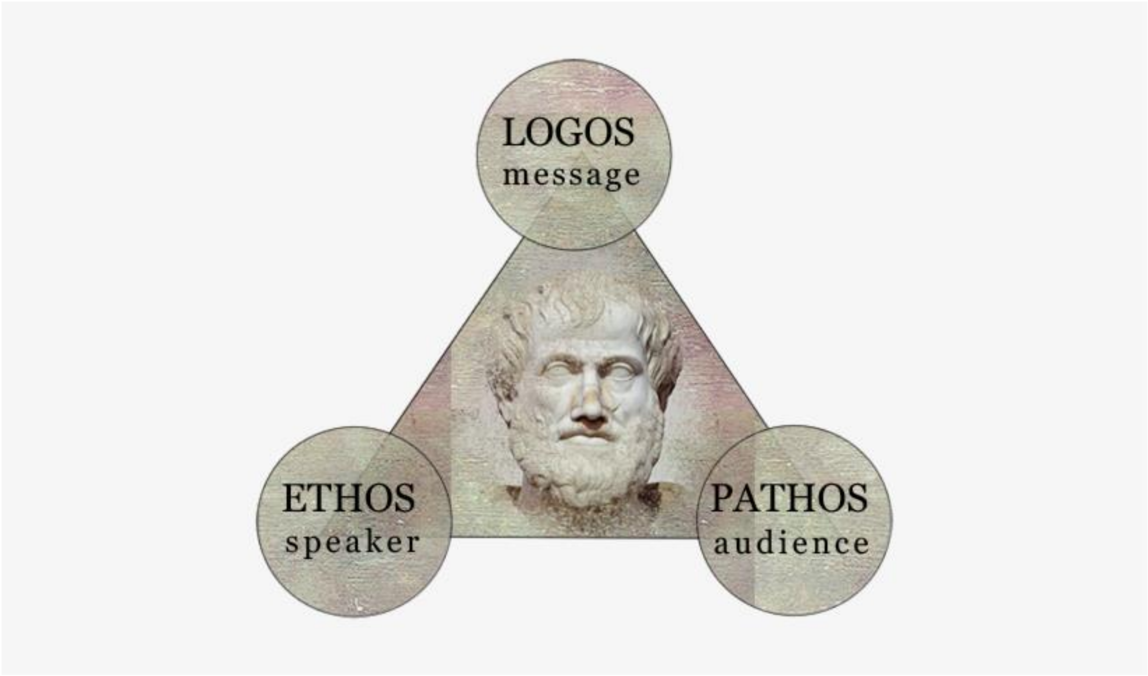

The increased abstraction in mathematics that took place during the early part of this century was paralleled by a similar trend in the arts. In both cases, the increased level of abstraction demands greater effort on the part of anyone who wants to understand the work.
— Keith Devlin —

The C-R-A Learning Progression



Fluency in the use of **abstractions** will also increase your **communication abilities**

The Rhetorical Triangle (Aristotle)



Let us recall a couple of examples of mathematical abstractions…

An FSA, formally

- An FSA is a **tuple** **$<Q, A, \delta, q_0, F>$** 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**

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

Theoretical Computer Science

Mathematical Induction and Peano Axioms

Lecture 4 - Manuel Mazzara

Induction

- It is as old as history of mathematics (Plato, Pascal, De Morgan…)

- In modern time **Giuseppe Peano (**1858-1932) defined the so-called axioms for the natural numbers (now called **Peano axioms)**

- The axioms define **arithmetical properties of *natural numbers***

Peano axioms (1)

- **0 is a natural number**

- About **equality relation**
  - For every natural number x, $x = x$. **Equality is reflexive**
  - For all natural numbers x and y, if $x = y$, then $y = x$. **Equality is symmetric**
  - For all natural numbers x, y and z, if $x = y$ and $y = z$, then $x = z$. **Equality is transitive**
  - For all a and b, if b is a natural number and $a = b$, then a is also a natural number. **Natural numbers are closed under equality**
    - *We will see soon the notion of closure in details*

Peano axioms (2)

- **Successor** function S
  - For every natural number n, $S(n)$ is a natural number. **<u>Natural numbers are closed under *S*</u>**
  - For all natural numbers m and n, $m = n$ if and only if $S(m) = S(n)$. That is, ***S* is an injection** (i.e. a function that maps distinct elements of its domain to distinct elements of its codomain)
  - For every natural number n, $S(n) = 0$ is false. **<u>There is no natural number whose successor is 0</u>**

Peano axioms (3)

- **Axiom of induction**

- If φ is a **unary predicate** (boolean-valued function P: $X \rightarrow$ {true, false}) such that:

    - $\varphi(0)$ is true, and
    - for every natural number n, $\varphi(n)$ being true implies that $\varphi(S(n))$ is true,

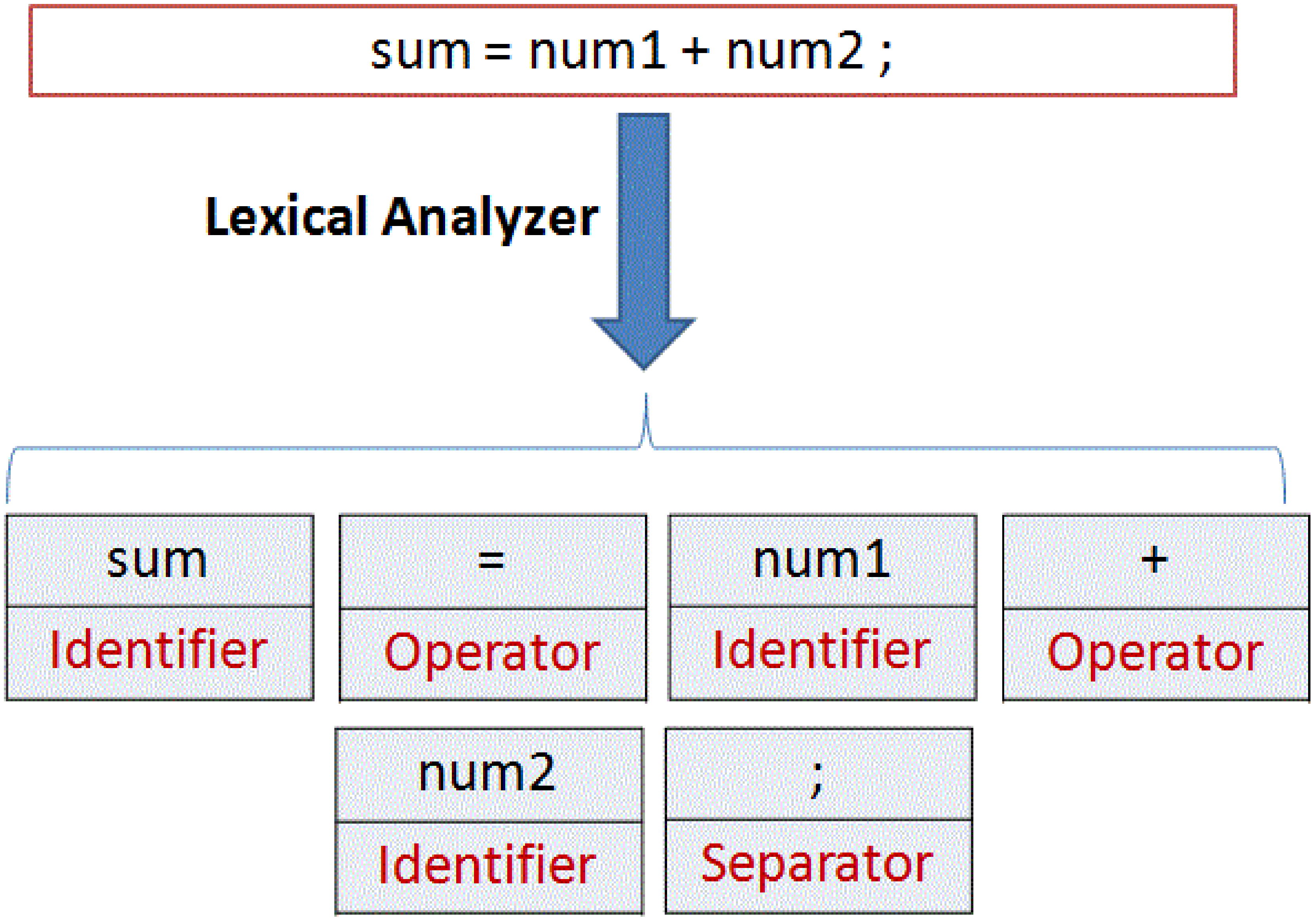
then $\varphi(n)$ is true for every natural number n

Theoretical Computer Science

Lexical analysis and FSA, a few hints

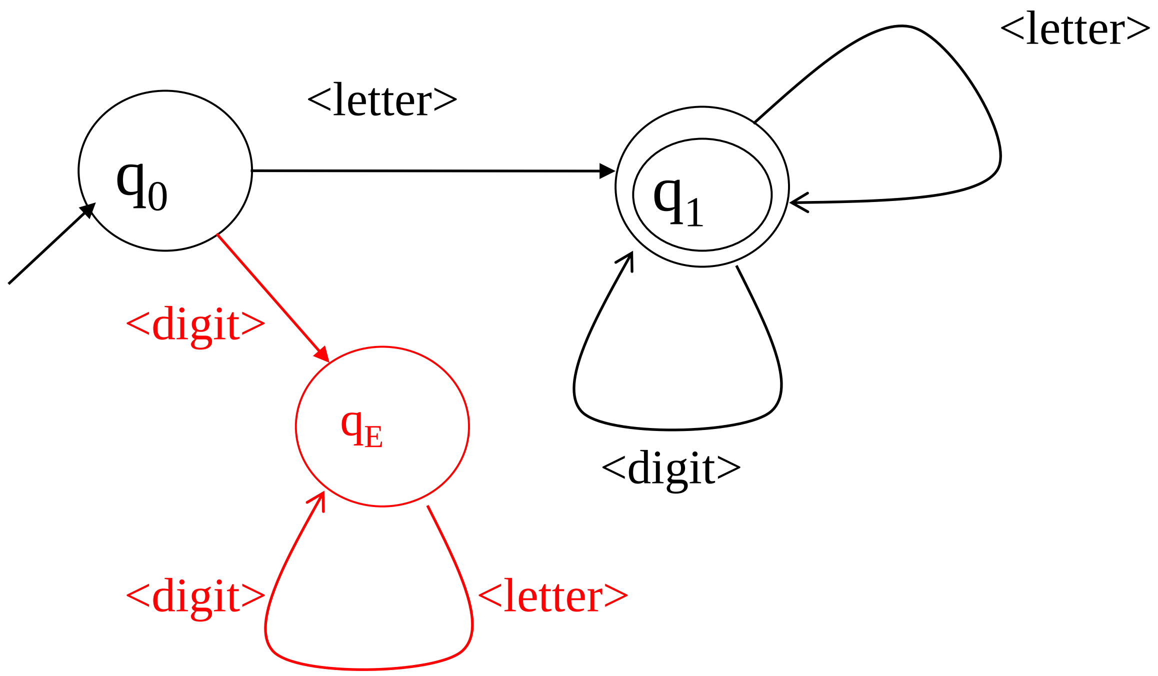
Lecture 4 - Manuel Mazzara

<u>Lexical analysis</u> is the first phase of a compiler. The lexical analyzer breaks the program syntax into a series of **<u>tokens</u>**.


sum = num1 + num2 ;
Lexical Analyzer
sum
Identifier
=
Operator
num1
Identifier
+
Operator
num2
Identifier
;
Separator

FSA, a practical example

- Recognizing Pascal **identifiers**



Theoretical Computer Science

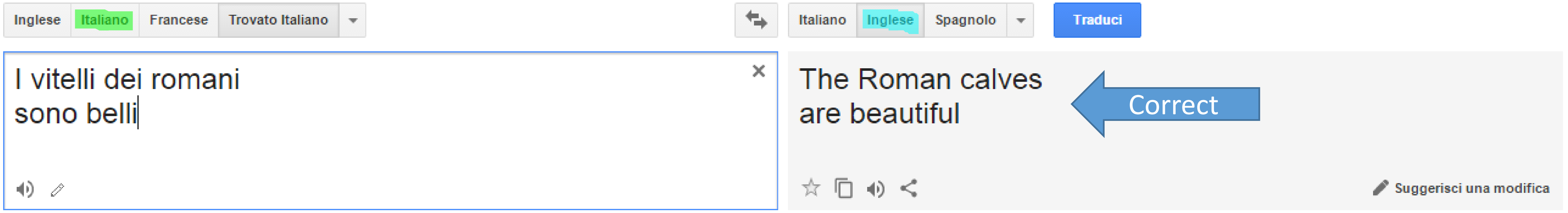
Syntax vs. Semantics

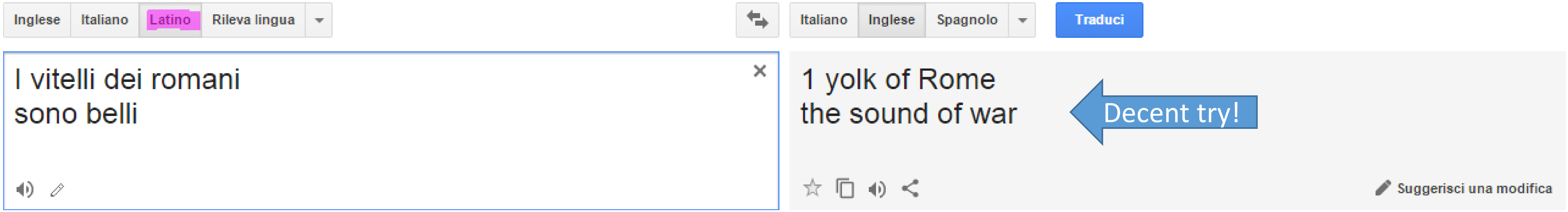
Lecture 4 - Manuel Mazzara

What is the meaning of this sentence?

«I vitelli dei romani sono belli»

Is the question well-posed?





We do not have all the information needed to answer the question!

Syntax vs. Semantics (linguistics)

- **Syntax** is the set of rules, principles, and processes that **govern the structure of sentences** in a given language, specifically word order. The term syntax is also used to refer to the study of such principles and processes.
  - The word *syntax* comes from Ancient Greek "coordination“

- **Semantics** is primarily the linguistic, and also philosophical study of **meaning** in language, programming languages, formal logics, and semiotics. It focuses on the **relationship between signifiers** (words, phrases, signs) and **symbols**, and **what they stand for**, their ”**denotation**” (translation of a sign to its meaning).
  - The word *semantics* comes from Ancient Greek "significant"

Solution of the quiz

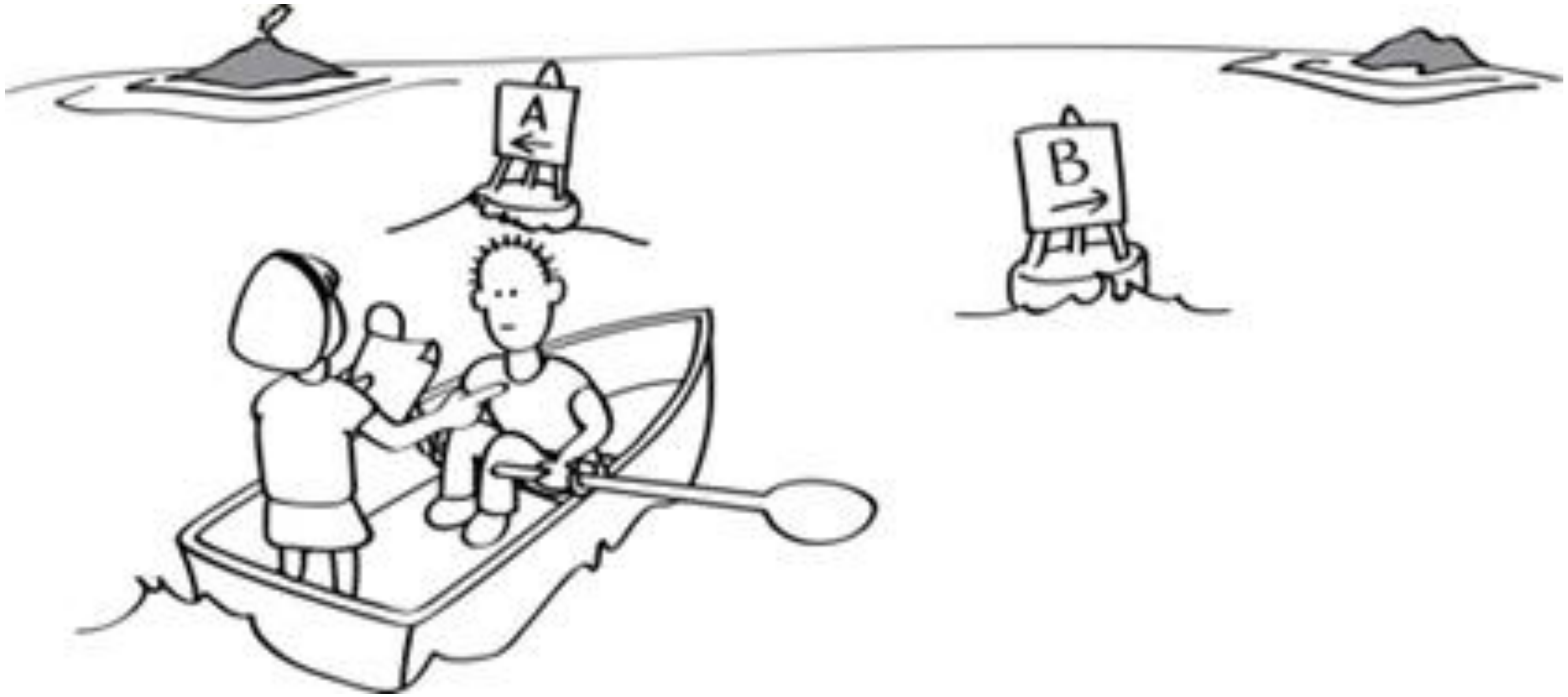
- ***I vitelli dei romani sono belli*** (intended as a sentence in Italian)
  - *"The Roman calves are beautiful"* (Google does it fine!)

- ***I vitelli dei romani sono belli*** (intended as a sentence in Latin)
  - *"Go, oh Vitellius, at the sound of the Roman god of war"* (Google cannot make it!)

Theoretical Computer Science

Regular languages

Lecture 4 - Manuel Mazzara

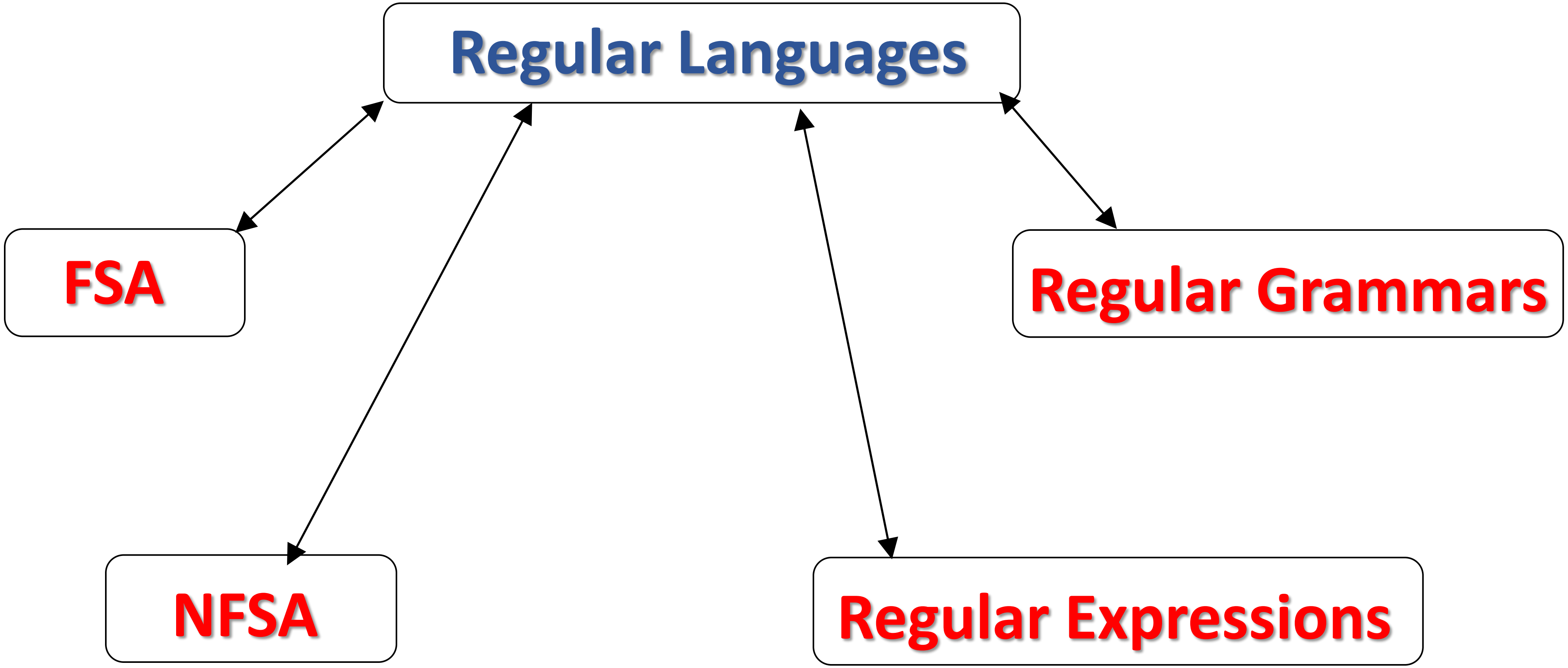


FSA

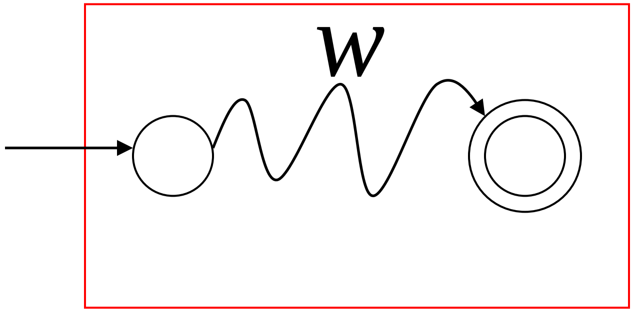
Regular Languages

- A **regular language** is a language recognized by a FSA
- Regular languages are very useful in **input parsing and programming language design**
  - See the previous part of this lecture
- We will see models that are equivalent to languages recognized by FSA
  - **Regular expressions**
  - Specific type of **generative grammars**

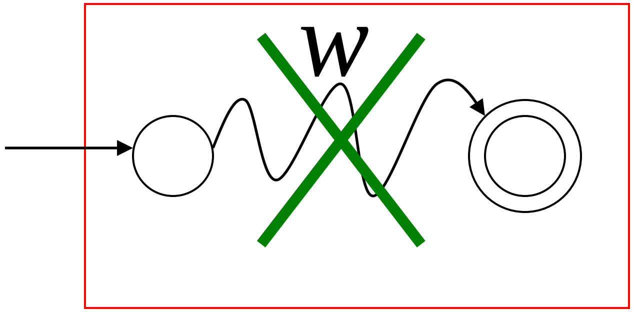
Representations of Regular Languages



Belonging of a string w to the language L



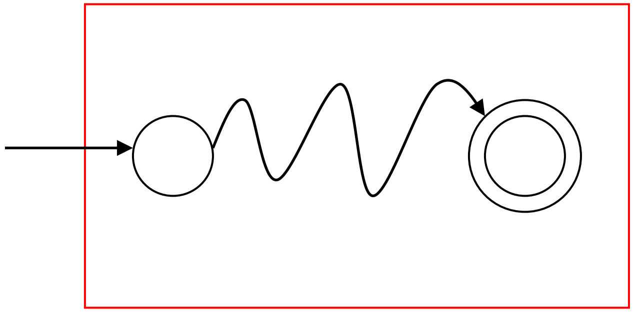
$w \in L$



$w \notin L$

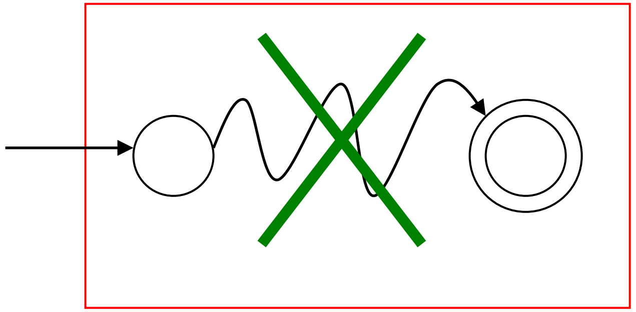
We can compute this property

Is L empty?



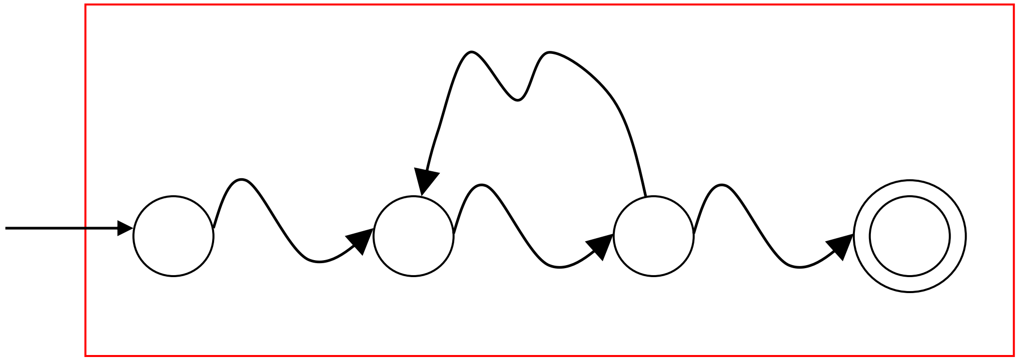
$$L \neq \varnothing$$

We can compute this property



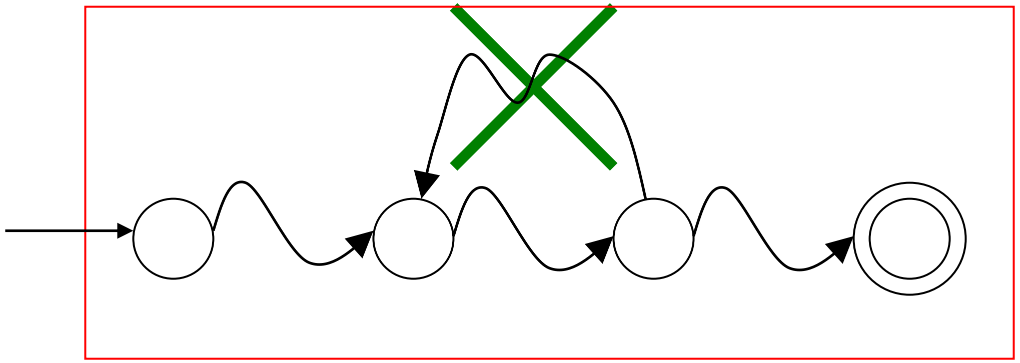
$$L = \varnothing$$

Is L finite?



L is infinite

We can compute this property



L is finite

Theoretical Computer Science

Closure and languages

Lecture 4 - Manuel Mazzara

Closure in math (recap)

- 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? <u>Consider zero!</u>**
  - Reals…
  - …

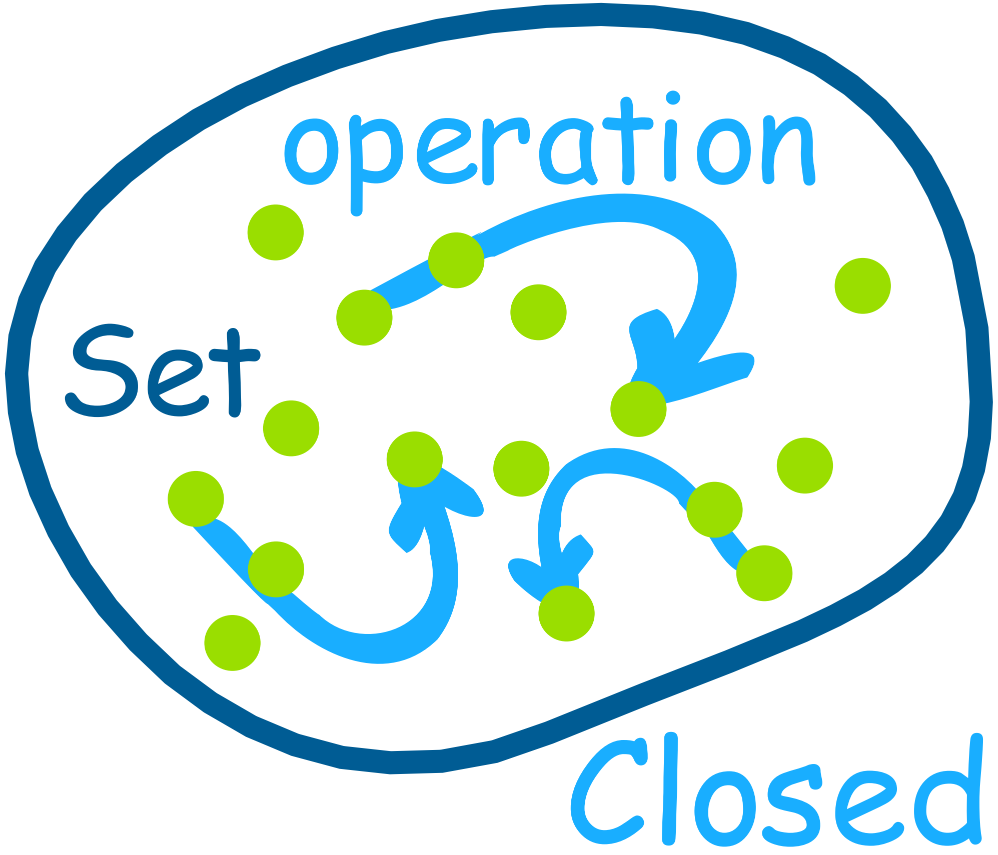
Rationals

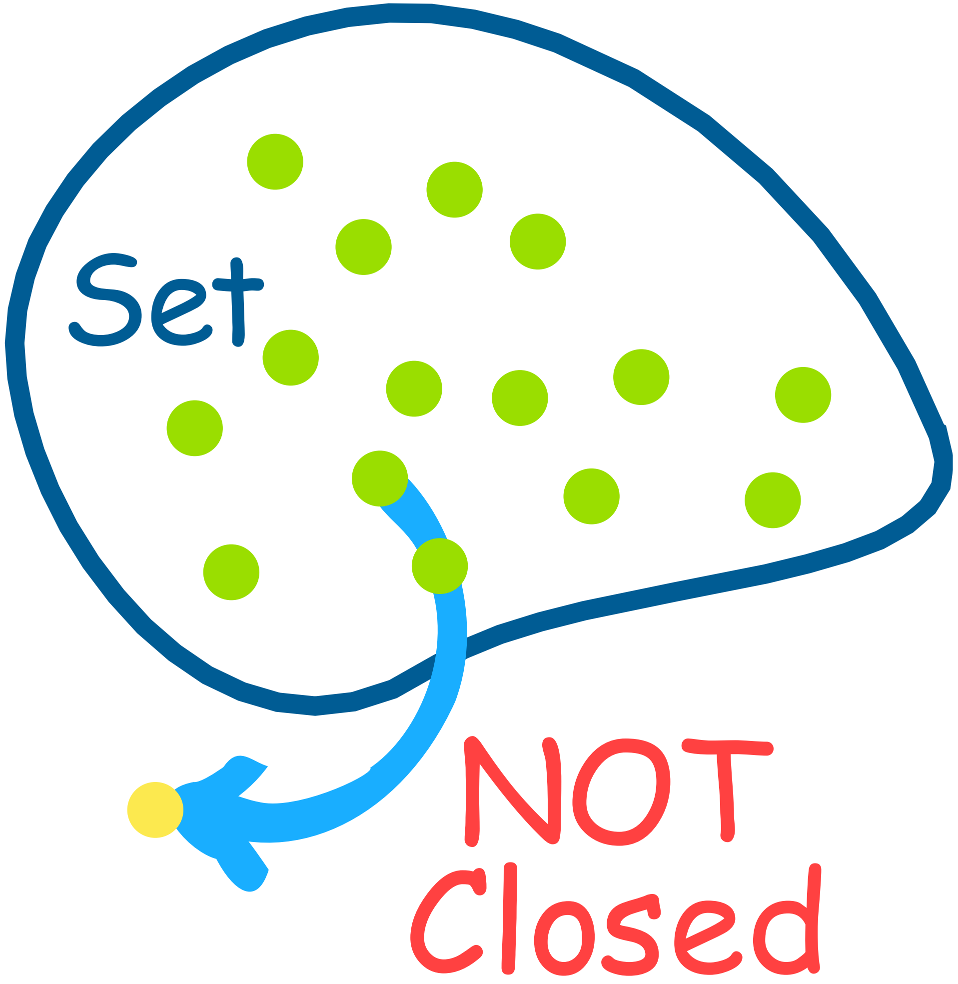
- A rational number is a number that can be represented as a fraction **m/n**, where **m** and **n** are integers and **n≠0**

- 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


operation
Set
Closed


Set
NOT
Closed

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 and "*"

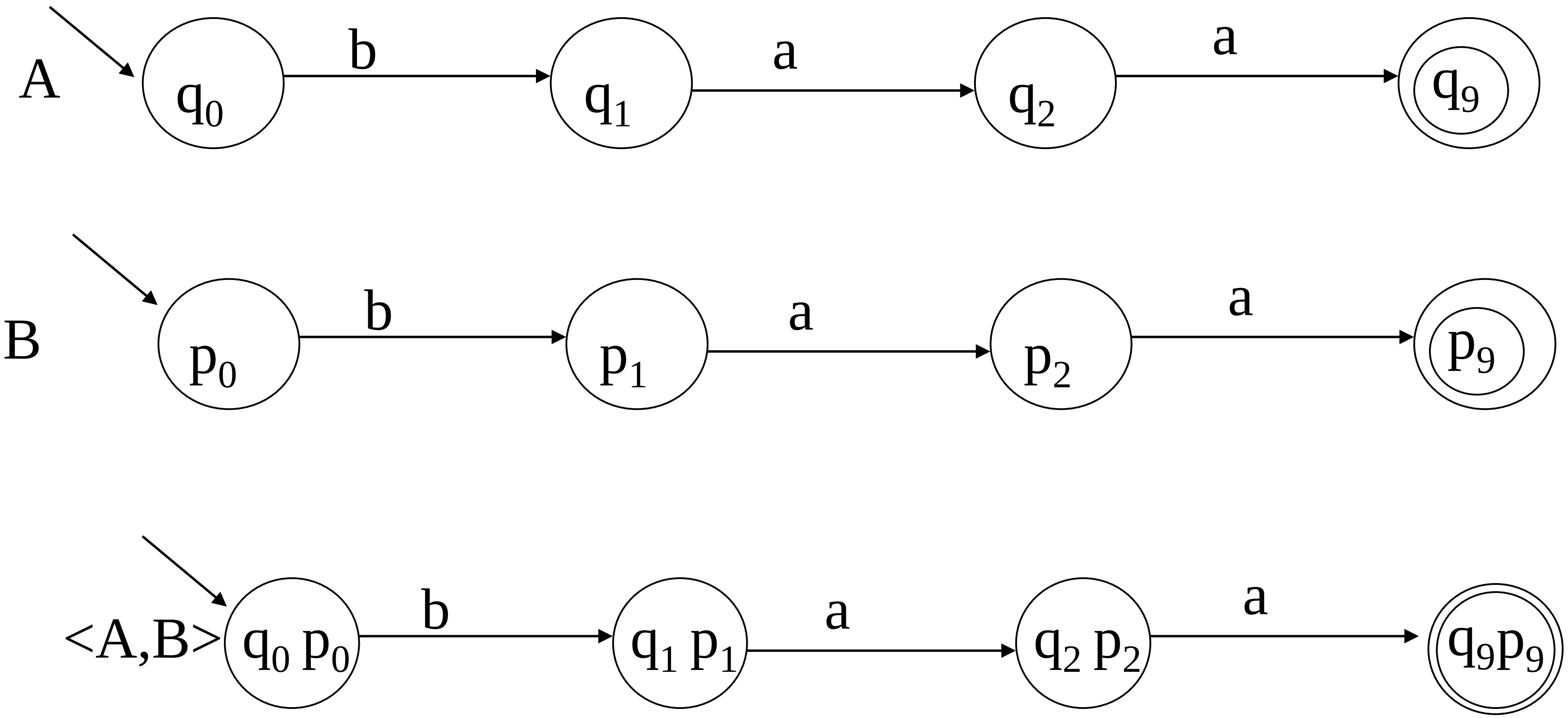
Theoretical Computer Science

Operations on FSA

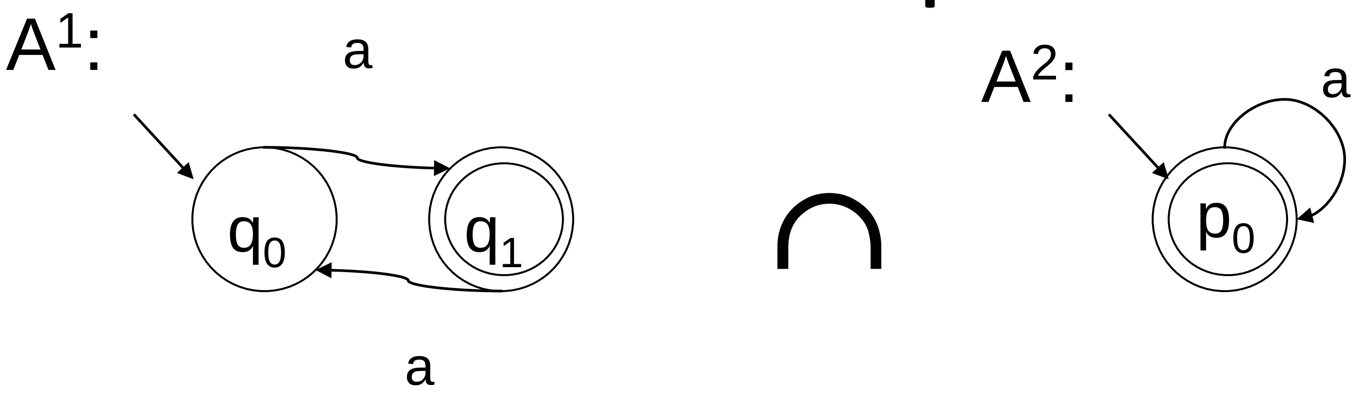
Lecture 4 - Manuel Mazzara

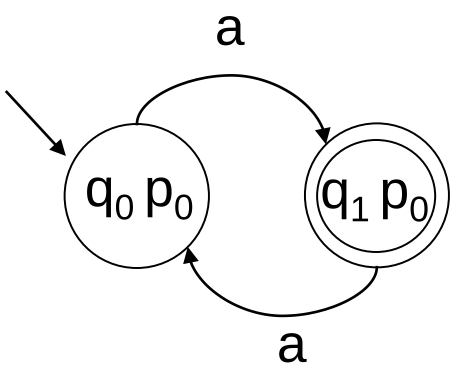
Intersection

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



Example





43

Formally

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

  $< A^1, A^2 > = <Q^1 \times Q^2, I, \delta, <q_0{}^1, q_0{}^2>, \boxed{F^1 \times F^2} >$

  - $\boldsymbol{\delta(<q^1,q^2>, i) = <\delta^1(q^1, i), \delta^2(q^2,i)>}$
- One can show (by induction) that

  $\mathbf{L(<A^1,A^2>) = L(A^1) \cap L(A^2)}$
- **<u>Can we do the same for union?</u>**

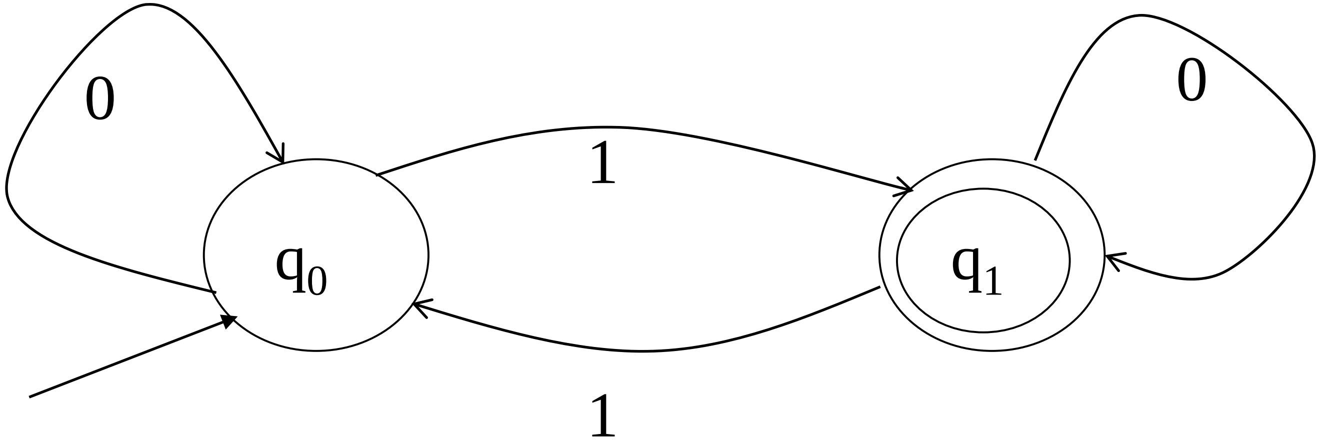
Union

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

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

Complement (1)

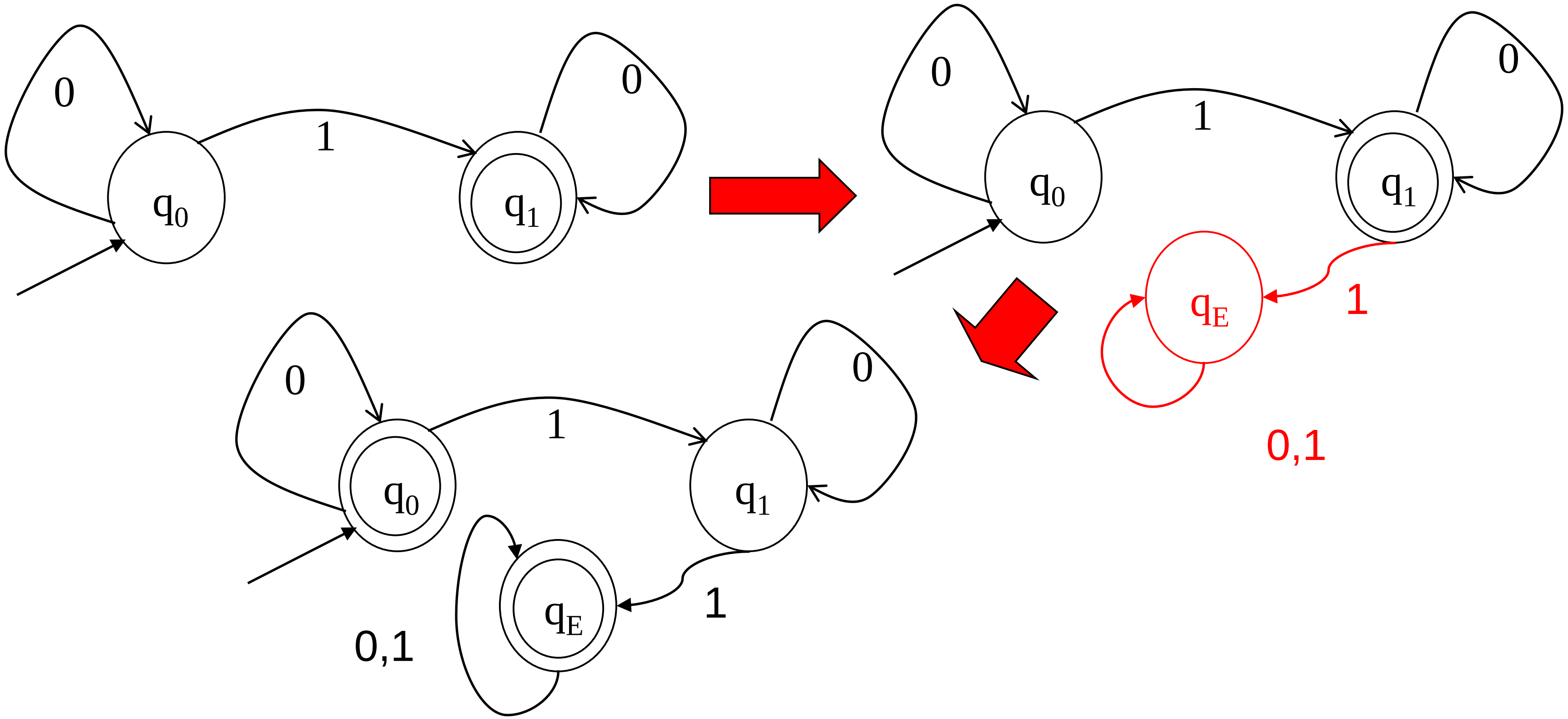
- Basic idea $F^c = Q - F$



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

Complement (2)

- Before swapping final and non final states it is necessary to **complete the FSA**



Union again

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

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

Complement and FSA

- Strings in FSA are **accepted only if the scan reaches a final state**
  - If a final state is not reached the string is not accepted

- If the input string is always scanned (**complete FSA**), then it suffices to “swap yes and no” (F with Q-F)

- If the end of the string cannot be reached (**not complete FSA**), then swapping F with Q-F does not work

- In the case of FSAs there is an easy workaround
  - **Completing the FSA**

General Observation

- Swapping final states means **asking the opposite question** (having an automaton for complement language) **and looking for positive answers** (accepted strings)

- In general, **we cannot consider the negative answer to a question as equivalent to the positive answer to the opposite question!**
  - **We will see what this means for Turing machines**

- In fact, **closure over complement is fundamental when it comes to computability issues**!

Complement and TM (spoiler ahead!)

- TMs are more expressive than FSA
  - More “programs” can be expressed

- TMs and Turing-complete programming languages allows **nonterminating programs** (it is important to express important algorithms)

- A TM accepts a string if it will eventually halt and say "Yes“
- **<u>If it does not halt you cannot know whether it will ever do</u>**
  - Non closure wrt complement