



Computability

Silvio Peroni

✉ silvio.peroni@unibo.it  0000-0003-0530-4305  @essepuntato

Computational Thinking and Programming (A.Y. 2017/2018)

Second Cycle Degree in Digital Humanities and Digital Knowledge

Alma Mater Studiorum - Università di Bologna



Creative Commons Attribution 4.0 International License

Communication 1

As of yesterday, only 13 people have subscribed to the mailing list

Please subscribe to the mailing list as soon as possible

1. Open a browser and go to <https://www.dsa.unibo.it/>
2. Log in with your University email
3. Look for the mailing list ("*Liste docenti-studenti*") - the name of the list is "*compthink1718*"
4. Subscribe to it (password needed, please ask it to me)

Then, you can send directly emails to

silvio.peroni.compthink1718@studio.unibo.it

Communication 2

Weekly consultations: every Friday, from 14:30 to 16:00 in "studio 12" at the Department of Classical Philology and Italian Studies, in via Zamboni 32

Students must agree to an appointment by sending an email to silvio.peroni@unibo.it

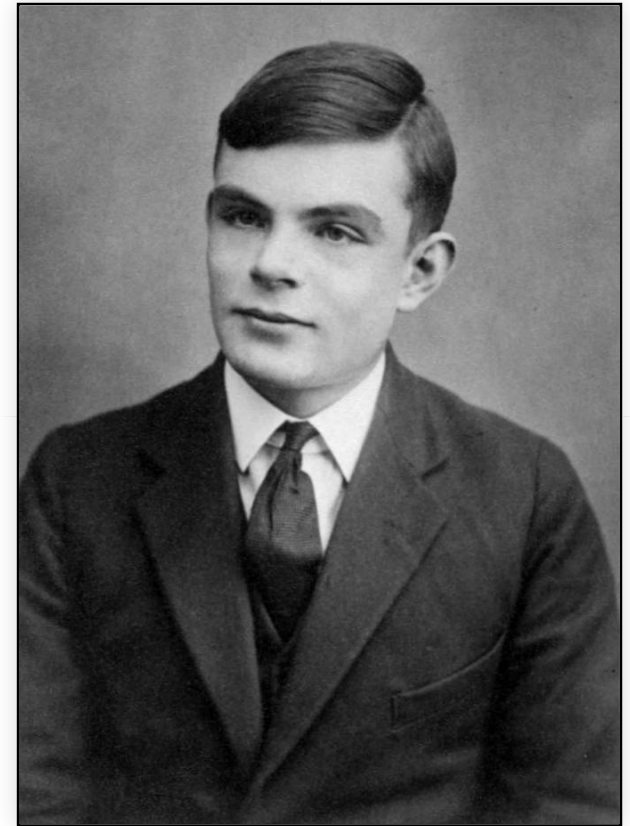
Any question about the previous
lecture?

Historic hero: Alan Turing

Computer scientist (+ works in mathematics, logic, philosophy, and biology)

Father of Theoretical Computer Science (see Turing machine) and of Artificial Intelligence (see Turing test)

Crucial contribution in deciphering the German Nazi Enigma machine during the Second World War - he was appointed an Officer of the Order of the British Empire (OBE)

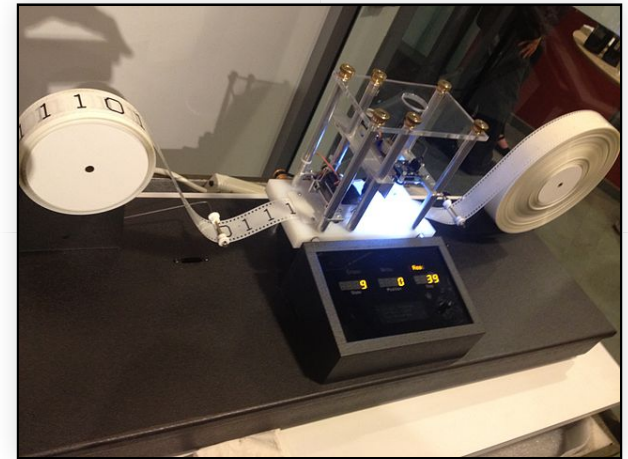


Turing machine

It is a theoretical machine

Turing machine can be used to simulate any algorithm

If you can define an algorithm for solving a particular computational problem, then there exists a Turing machine that can solve the same problem as well, and vice versa



It is composed by:

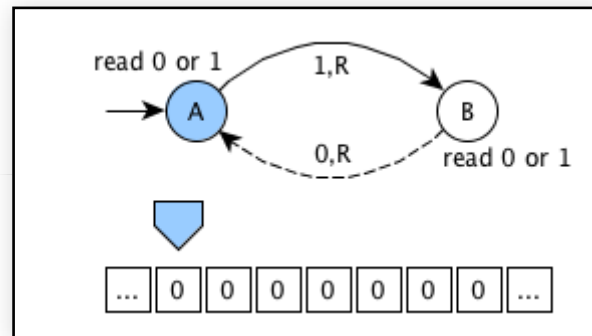
- an infinite memory tape containing cells - each cell can contain 0 (blank symbol) or 1
- an head for reading/writing/moving on the tape
- a storage for recording the current state
- a finite table of instructions

Table of instructions

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A

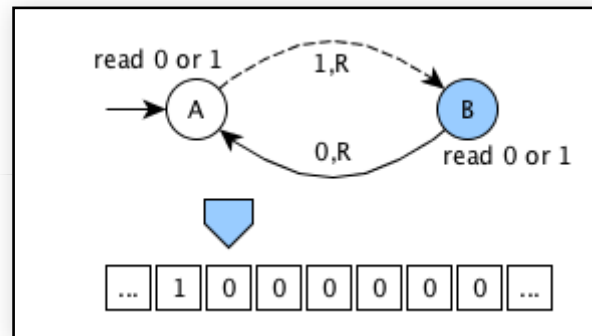
Graphical example

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A



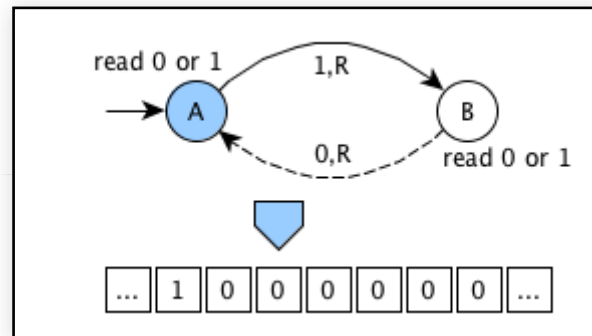
Graphical example

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A



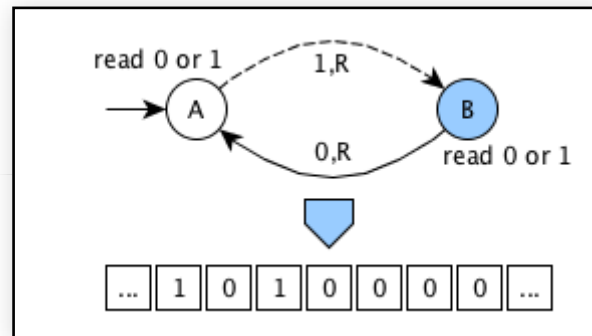
Graphical example

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A



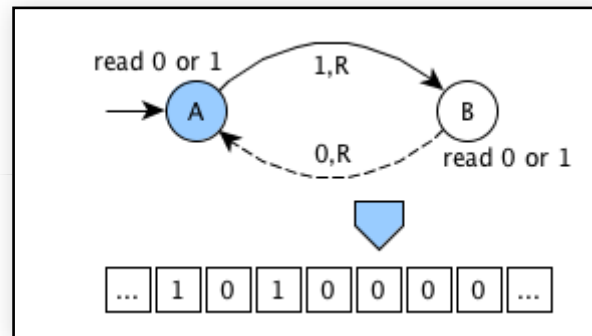
Graphical example

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A



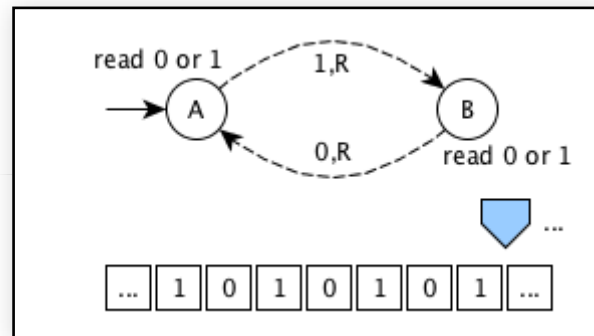
Graphical example

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A



Graphical example

Current state	Tape symbol	Write symbol	Move tape	Next state
A	0 or 1	1	right	B
B	0 or 1	0	right	A



Cost of computation

The previous simple machine can compute indefinitely (there are no ending states)

Other machines could compute a result in a reasonable finite time

There can be also algorithms that must spend an **exaggerated** time (even if still finite) to return a result

Execution cost of an algorithm: how much time, indicatively, an algorithm needs for returning a result

How to do it

Computational complexity theory: classifying **computational problems**

Computational problem: a problem that be solved algorithmically by a computer

Classification to an algorithm is provided according to a specific hierarchy of classes that express the difficulty in solving such problems

Analysis of algorithms: understanding the amount of time, storage and other resources that are needed to execute such algorithm, defined in terms of a mathematical function

Typical question: how many instructions an algorithm has to execute to return the final result from that input?

Lessons to learn

The **efficiency** of an algorithm is directly derived and guided by the way such algorithm has been developed

It is possible to develop **two** different algorithms addressing the **same** computational problem that take two drastic different times for returning the result

Important questions

Can we use algorithms for computing whatever we want?

Alternatively, there exists a limitation on what we can compute?

Alternatively, is it possible to define a computational problem that cannot be solved by any algorithm?

Turing created his machine for answering this question

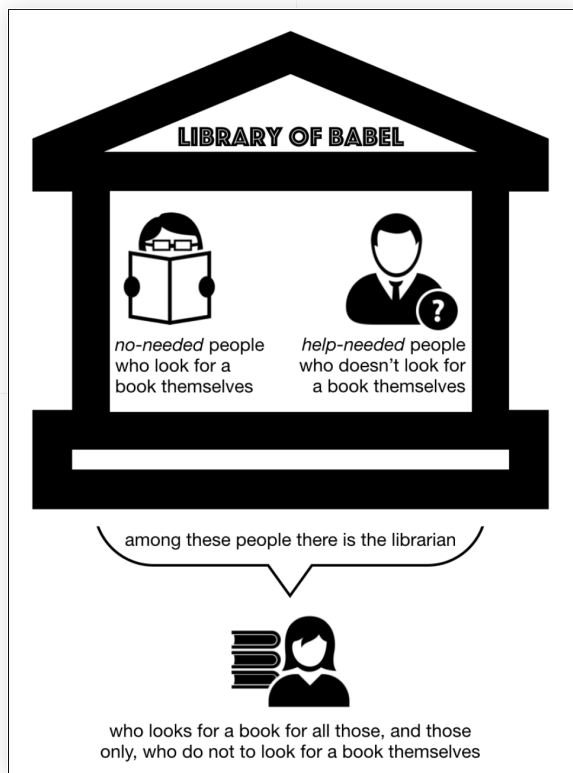
Reductio ad absurdum

One of the most used approaches to demonstrate that something cannot exist

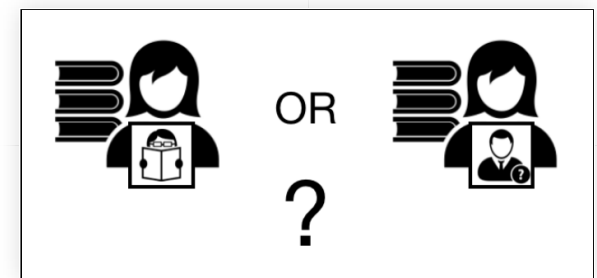
Idea: to come to a paradoxical and self-contradictory situation in which, for instance, the existence of an algorithm contradicts its existence itself

Library paradox

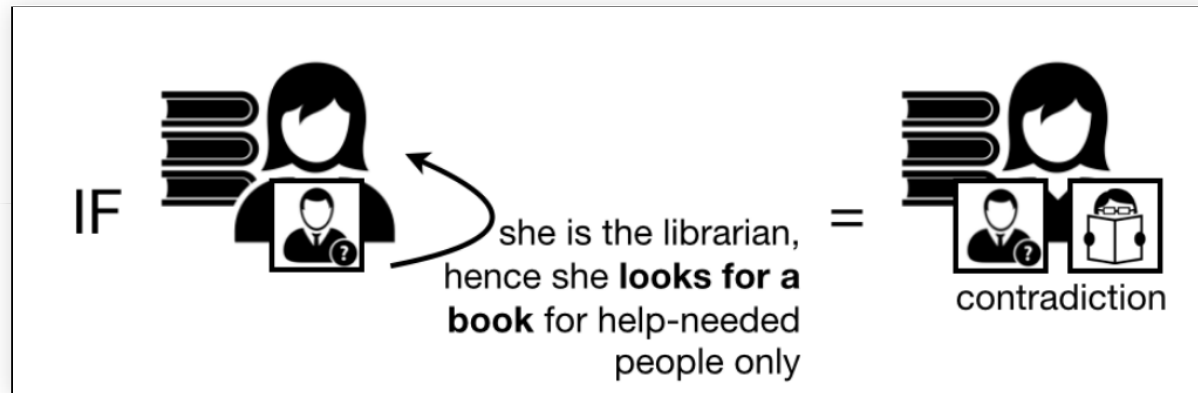
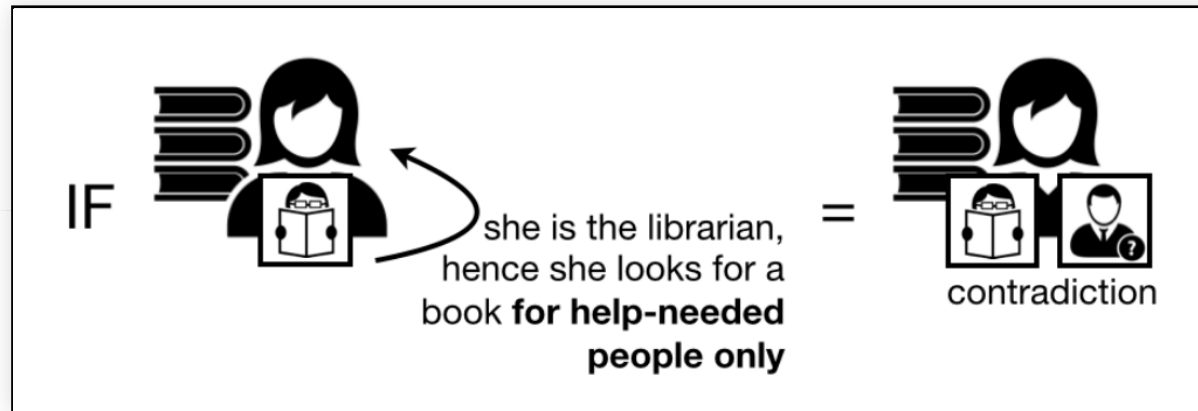
Setting



Problem



Resolution



The halting problem

Proposed by David Hilbert in 1900

Description: is it possible to develop an algorithm that takes another algorithm as input and determines if such input algorithm will terminate its execution at some point or it will run forever?

From Turing point of view: can we develop a Turing machine which is able to certainly decide if another machine will terminate its execution or will not?

Resolution (part 1)

Suppose we have a `def does_it_halt(an_algorithm)` algorithm, which returns *True* if the execution of `an_algorithm()` terminates, *False* otherwise

We reuse the `does_it_halt` algorithm for developing a new algorithm:

```
def a_simple_algorithm(another_algorithm):  
    if does_it_halt(another_algorithm):  
        run_forever()  
    else:  
        return
```

The `run_forever` algorithm exists since we can develop a Turing machine that does not terminate

Resolution (part 2)

What happens if we try to execute
`a_simple_algorithm(a_simple_algorithm)`?

Two situations:

1. if `does_it_halt` says that `a_simple_algorithm` stops (i.e. `does_it_halt(a_simple_algorithm)` returns *True*), then `a_simple_algorithm` run forever
2. if `does_it_halt` says that `a_simple_algorithm` runs forever (i.e. `does_it_halt(a_simple_algorithm)` returns *False*), then `a_simple_algorithm` stops

Contradiction

Whatever is the behaviour of `a_simple_algorithm`, it always generates a contradiction



While `a_simple_algorithm` can be developed (we provided the algorithm!), then it is `does_it_halt` that cannot be developed

Turing's machines and their analyses posed clear limits to what we can compute, since there are specific computational problems that cannot be solved

END

Computability

Silvio Peroni

✉ silvio.peroni@unibo.it  0000-0003-0530-4305  @essepuntato

Computational Thinking and Programming (A.Y. 2017/2018)

Second Cycle Degree in Digital Humanities and Digital Knowledge

Alma Mater Studiorum - Università di Bologna