Computabilità (capitolo 2)

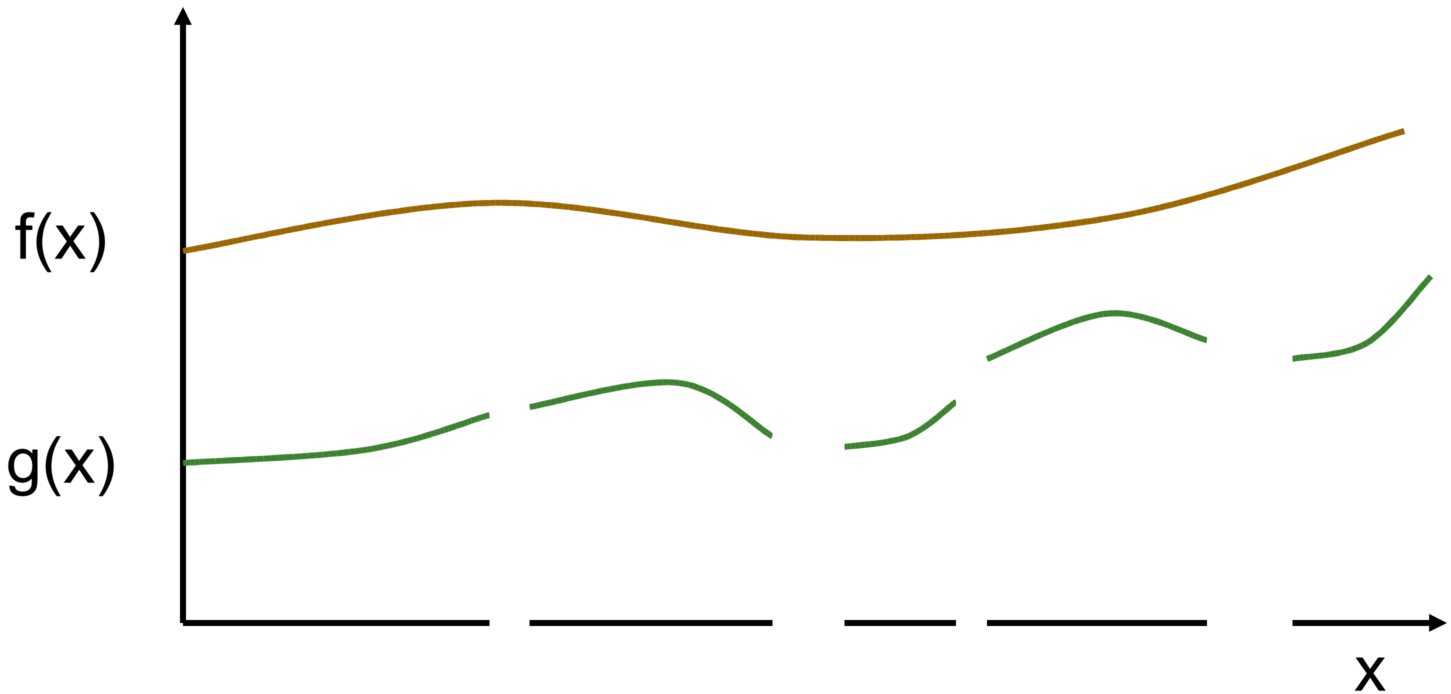
Angelo Gargantini

lezioni per informatica 3 ,
Università di Bergamo

Foundations: Partial,Total Functions

- Value of an expression may be undefined
  - Undefined operation, e.g., division by zero
    - 3/0 has no value
    - implementation may halt with error condition
  - Nontermination
    - f(x) = if x=0 then 1 else f(x-2)
    - this is a partial function: not defined on all arguments
    - cannot be detected at compile-time; this is halting problem
  - These two cases are
    - “Mathematically” equivalent, but Operationally different

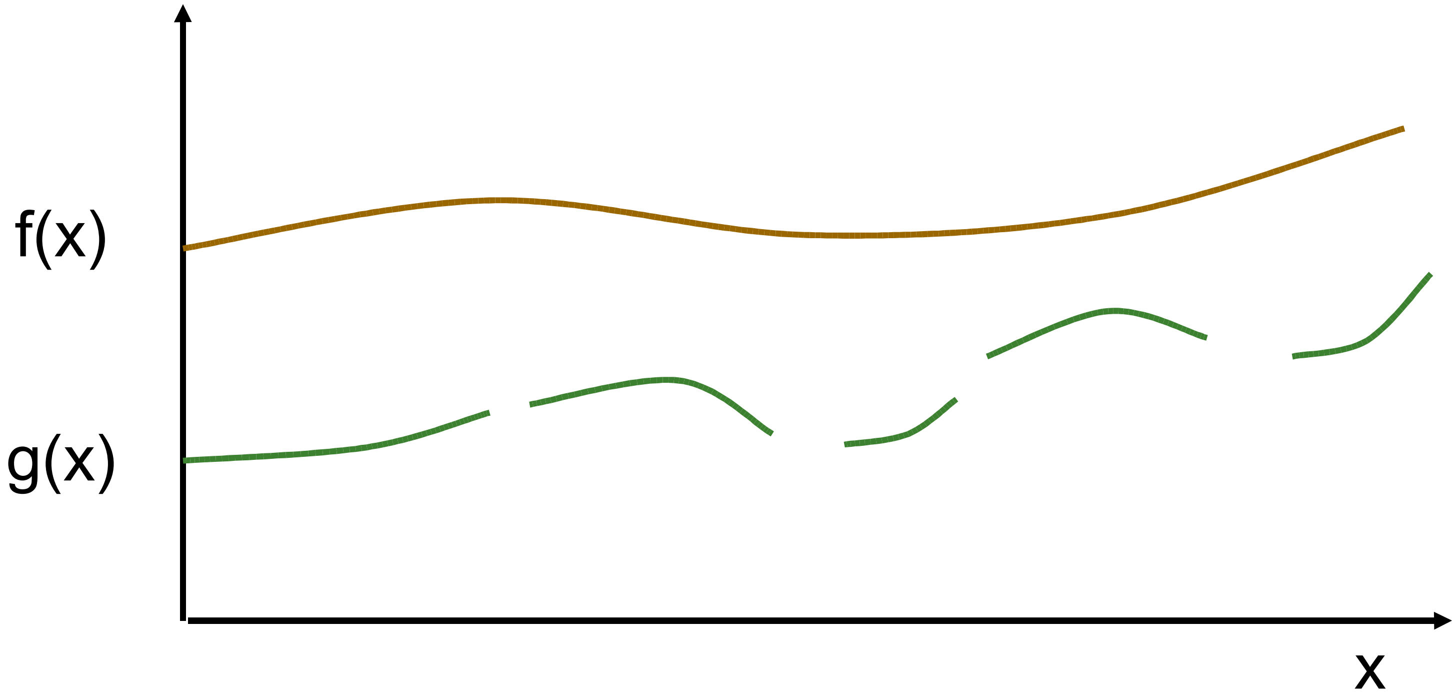
Partial and Total Functions



Total function: f(x) has a value for every x

Partial function: g(x) does not have a value for every x

Functions and Graphs



Graph of f = { ⟨x,y⟩ | y = f(x) }

Graph of g = { ⟨x,y⟩ | y = g(x) }

Mathematics: a function is a set of ordered pairs (graph of function)

Partial and Total Functions

Total function $f:A \rightarrow B$ is a subset $f \subseteq A \times B$ with

- For every $x \in A$, there is some $y \in B$ with $\langle x,y \rangle \in f$ (total)
- If $\langle x,y \rangle \in f$ and $\langle x,z \rangle \in f$ then $y=z$ (single-valued)

Partial function $f:A \rightarrow B$ is a subset $f \subseteq A \times B$ with

- If $\langle x,y \rangle \in f$ and $\langle x,z \rangle \in f$ then $y=z$ (single-valued)

Programs define partial functions for two reasons

- partial operations (like division)
- nontermination
  - f(x) = if x=0 then 1 else f(x-2)

Halting Problem

Entore Buggati: "I build cars to go, not to stop."

Self-Portrait in the Green Buggati (1925)
Tamara DeLempicka

Computability

Definition

Function f is computable if some program P computes it:

For any input x, the computation P(x) halts with output f(x)

Terminology

Partial recursive functions

= partial functions (int to int) that are computable

P written in which programming language

- Some functions may be computable by programs written in a programming language (e.g. C) , but not in another
- let’s use a simple “programming language”: Turing Machines

Turing Machines

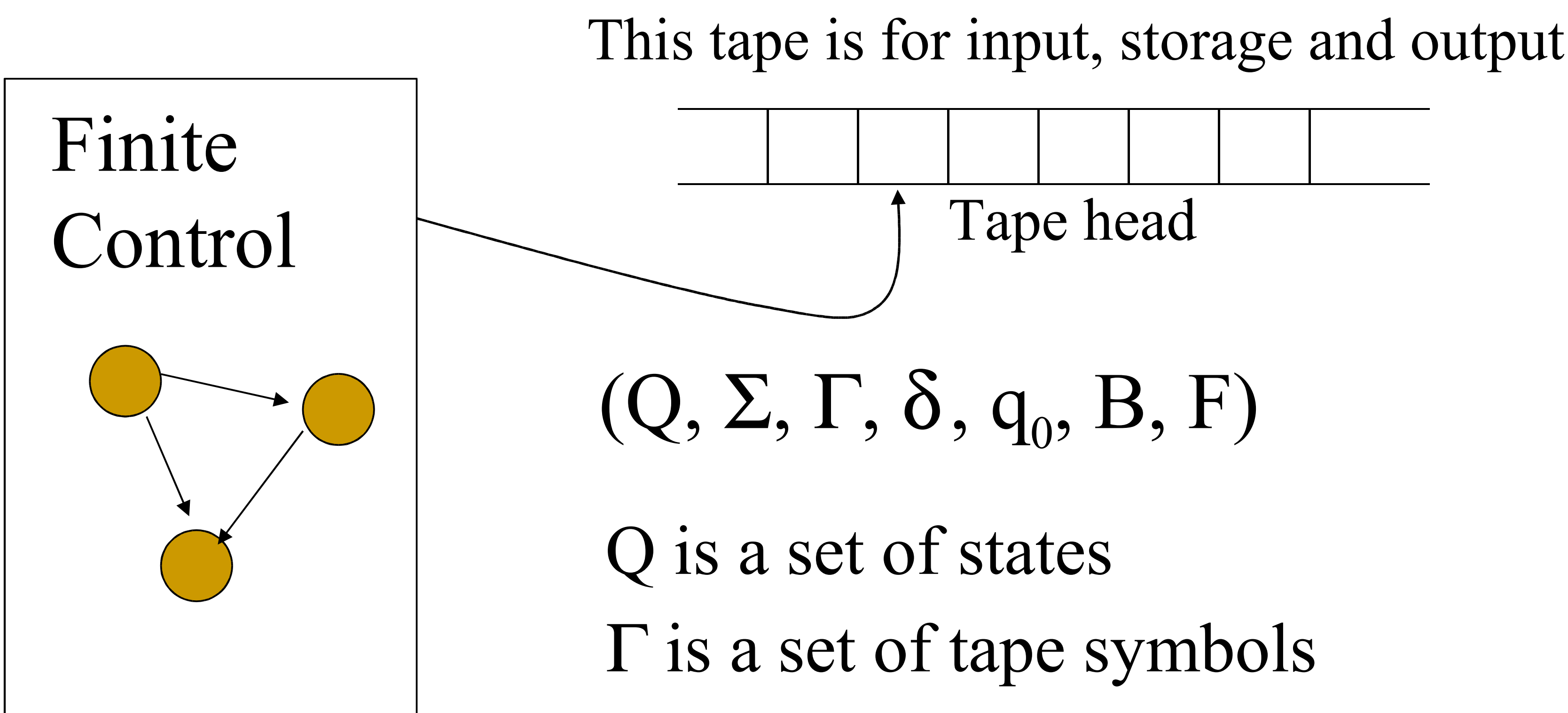
- In 1936, A.M. Turing proposed the Turing machine as a model of *any possible computation*.
- This model was *built* using electro-mechanical devices after several years.
- Turing machine long has been recognized as an accurate model for what any physical computing device is capable of doing.

Turing Machines

Turing formalized the idea of “mechanical procedure”:

- Can be described by a finite number of instructions.
- Instructions are simple and mechanical.
- Have a finite number of internal states.
- Can deal with input not restricted in size.
- Have an unlimited storage space for calculations.
- Can produce output of unlimited size.

Turing Machines



$(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Q is a set of states

Γ is a set of tape symbols

$B \in \Gamma$ is a blank symbol

$\Sigma \subseteq \Gamma \setminus \{B\}$ is a set of input symbols

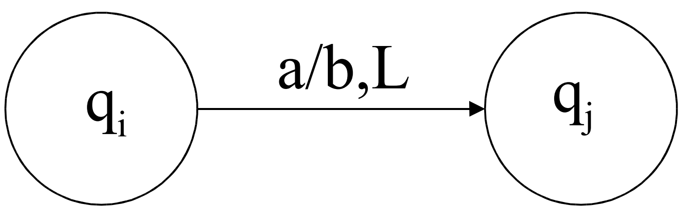
q_0 is the start state

$F \subseteq Q$ is a set of final states

Turing Machines

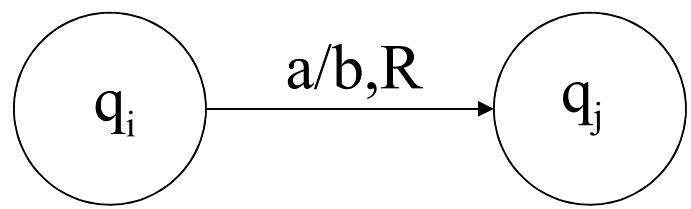
δ is a move function mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L,R\}$

Replace the current symbol "a" by "b", and move to the left.



$\delta(q_i, a) = (q_j, b, L)$

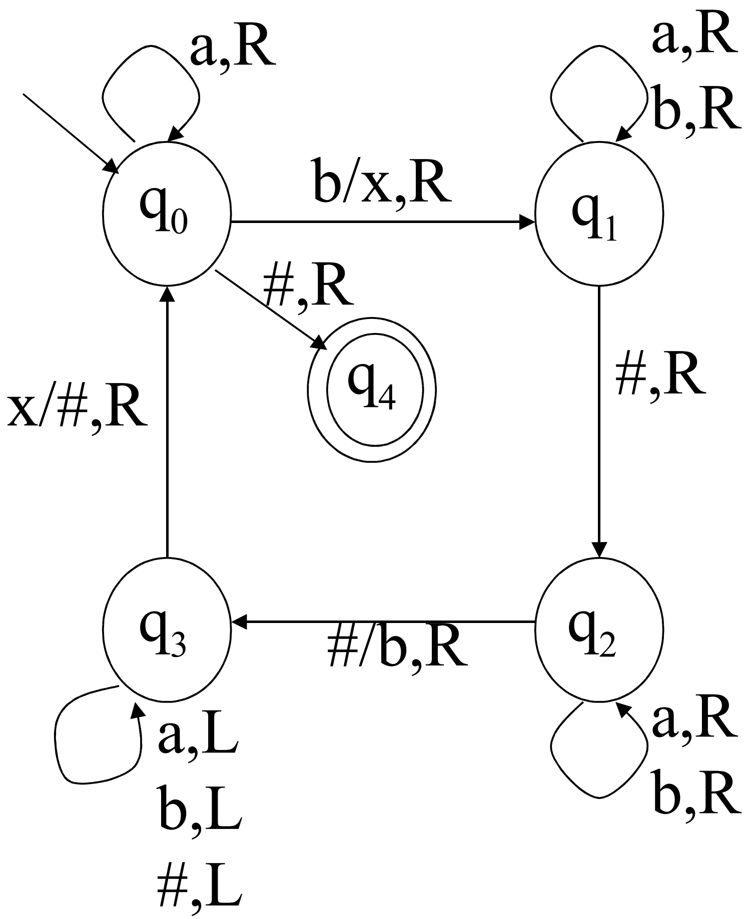
Replace the current symbol "a" by "b", and move to the right.



$\delta(q_i, a) = (q_j, b, R)$

Note that if a = b, we write "a,L" instead of "a/a,L" along the edge.

An Example



$Q = \{q_0, q_1, q_2, q_3, q_4\}$

$\Gamma = \{a, b, x, \#\}$

$B = \#$

$\Sigma = \{a, b\}$

q_0 is the start state

$F = \{q_4\}$

Consider the input:

baaba

What is the output?

Turing Machines

By using this powerful but simple model, we can start looking at the question of what languages can be defined (equivalently, what problems can be solved) by a computational device. Is there any problem that a computer cannot solve, and what are they?

We will see in some later classes that there are a lot of them, and they are called “undecidable” problems.

Other Examples

TM can do all sorts of things. Try the followings:

- Add 1 to a unary number. (Easy)
- Add 1 to a binary number.
- Convert a unary number to binary, and vice versa.
- Compare two unary numbers.
- Add two unary numbers.
- Compare two binary numbers x and y. If x > y, output 1. Otherwise, output 0.
- ……

TM Simulator

http://www.igs.net/~tril/tm/tm.html

http://www.cheransoft.com/vturing/download.h

chi scrive un simulatore salta la parte teorica sulle TM

Church thesis

Anni30!!

Non esiste meccanismo di calcolo automatico superiore alla MT o ai formalismi ad essa equivalenti.

Fin qui potrebbe essere un *Teorema* di Church (da aggiornare ogni volta che qualcuno si svegli al mattino con un nuovo modello di calcolo)

Nessun *algoritmo*,indipendentemente dallo strumento utilizzato per implementarlo, può risolvere problemi che non siano risolvibili dallaMT: la MT è il calcolatore più potente che abbiamo e che potremo mai avere!

Quali sono I problemi risolvibili algoritmicamente o automaticamente?

Gli stessi risolvibili dalla semplicissima MT!

Esistono funzioni non computabili?

Halting function

- Decide whether program halts on input
  - Given program P and input x to P,

$$
Halt\ (P,x) = \begin{cases} \text{yes} & \text{if } P(x) \text{ halts} \\ \text{no} & \text{otherwise} \end{cases}
$$

- Clarifications
  - Assume program P requires one string input x
  - Write P(x) for output of P when run in input x
  - Program P is string input to *Halt*
- *Halt* does not go in infinite loops
- Fact: There is no program for *Halt*

Proof

Suppose Q(P, x) is a program that:

returns "*halts*" if P(x) halts

returns "*does not halt*" if P(x) does not halt

Construct program D

D(P) = if Q(P,P) = "halts" then run forever
else halt

D(P) will halt if P(P) runs forever

D(P) will run forever of P(P) halts

Proof (2)

Applying as input D itself

What does D(D) do?

- If D(D) halts, then D(D) will run forever.
- If D(D) runs forever, then D(D) halts.

This is a contradiction!

Therefore, our assumption that Q solves the halting problem is not valid.

Implications of Halting Problem

There are useful program properties we cannot determine:

- will a program run forever or not?
- will a program eventually cause an error? (compilers do conservative checking- more on this later)
- will a program touch a specific piece of memory again?
- will be a statement covered by any input?

Main points about computability

- Some functions are computable, some are not
  - Halting problem
- Programming language implementation
  - *Can* report error if program result is undefined due to division by zero, other undefined basic operation
  - *Cannot* report error if program will not terminate

Other undecidable problems

- Program equivalence
- Do two programs always produce the same output?
  - where's my bug?
  - useful for debugging

In general

At the beginning of the 20th century, D. Hilbert asked whether it was possible to find an algorithm for determining the truth or falsehood of any mathematical proposition.

In 1931, K. Godel proved that for every consistent logic, there are some questions which cannot be answered “yes” or “no” - **Incompleteness Theorem.**

Halting Problem Proof in Java

Assume the existence of halt(f,x):

```
public boolean halt(String f, String x) {
if (???) return true;
else return false;
}
```

encode f and x as strings

Halting Problem Proof

Assume the existence of halt(f,x):

Construct function strange(f) as follows:

If halt(f,f) returns true, then strange(f) goes into an infinite loop

If halt(f,f) returns false, then strange(f) halts.

f is a string so legal to use for either input

```
public void strange(String f) {
  if (halt(f, f)) {
    while (true);
  }
}
```

Halting Problem Proof

Call strange()with ITSELF as input.

If strange(strange) halts then strange (strange) does not halt.

If strange(strange) does not halt then strange(strange) halts.

Either way, a contradiction. Hence halt (f,x) cannot exist.

More Undecidable Problems

Hilbert's 10th problem: " Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root."

Examples.

f(x, y, z) = 6x^3 y z^2 + 3xy^2 – x^3 – 10. yes: f(5, 3, 0) = 0

f(x, y) = x2 + y2 – 3. no

f(x, y, z) = xn + yn – zn yes if n = 2, x = 3, y = 4, z = 5

no if n>= 3 and x, y, z > 0.

(Fermat's Last Theorem)

.