
# Da un problema pratico insolubile a un problema teorico irrisolto (da un milione di dollari)

## Parte 1

Liceo Alfieri, Torino
20 gennaio 2015

*Elio Giovannetti*
*Dipartimento di Informatica*
*Università degli Studi di Torino*

# Corsi di Laurea in Informatica – Orientamento

Algoritmica per i licei

http://orientamento.educ.di.unito.it/course/view.php?id=49

http://orientamento.educ.di.unito.it/course/view.php?id=49

Per poter seguire con profitto i corsi è necessario iscriversi: potete farlo tramite il vostro account su un social network (facebook, google, ecc.).

Ambiente unico per tutte le scuole del progetto:

- copie delle slides, materiale multimediale, esercizi e compiti, forum, blog, ecc.

# Introduzione

# "Algoritmo": una parola chiave.

http://ricerca.repubblica.it/repubblica/archivio/repubblica/2014/02/12/fuggite-e-la-frana-cosi-un-algoritmo.html?ref=search

http://www.unita.it/politica/italicum-legge-elettorale-renzi-berlusconi-cuperlo-minoranza-pd-emendamenti-voto-camera-1.550641

http://www.repubblica.it/salute/ricerca/2013/11/04/news/genetica_un_algoritmo_identifica_varianti_dna_tumori-70212409/?ref=search

http://www.repubblica.it/scienze/2013/02/05/news/argot_algoritmo_per_investigare_dna-52009185/?ref=search

http://ricerca.repubblica.it/repubblica/archivio/repubblica/2014/01/20/un-algoritmo-puo-salvare-gli-animali-dallestinzione.html?ref=search

http://www.lastampa.it/2014/02/18/societa/lalgoritmo-della-rivoluzione-JN26HjPlKC6ovLLraDoqGO/pagina.html

http://www.lastampa.it/2013/09/27/tecnologia/veloce-come-un-colibr-google-rinnova-lalgoritmo-di-ricerca-EbY3w5Md0sWMURILdFIrkN/pagina.html

http://it.wikipedia.org/wiki/Algoritmo_di_Dijkstra

# Anche qualche titolo esagerato …

http://ricerca.repubblica.it/repubblica/archivio/repubblica/2013/03/13/lalgoritmo-della-felicita.html?ref=search

"a dificuldade de realizar o simples sobrepassa em complexidade todos os oficios e técnicas, ou, por outras palavras, é menos dificultoso conceber, criar, construir e manipular um cérebro electrónico do que encontrar no nosso próprio a simples maneira de ser feliz"

(J. Saramago, História do Cerco de Lisboa)

# Algoritmi e … logaritmi

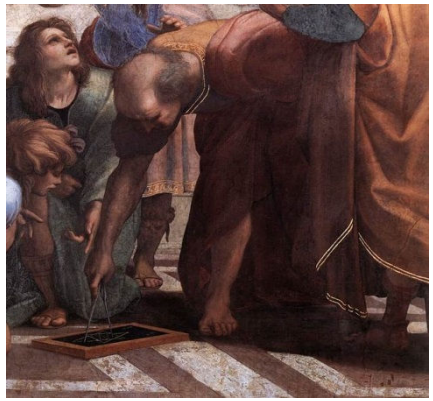
- la parola *algoritmo* deriva dal nome del matematico uzbeko-persiano al-Khwarismi الخوارزمي (vissuto intorno all'anno 800), il cui libro "Calcoli con i numerali indiani" descriveva i procedimenti di calcolo per le operazioni aritmetiche con il sistema di numerazione indiano, cioè quelli che ancora oggi studiamo nella scuola elementare;
- è un anagramma di *logaritmo*; la funzione logaritmo ha un ruolo abbastanza importante nell'analisi degli algoritmi !

# Tashkent, monumento ad al-Khwarismi (foto di Simona Castello)

# Un complesso di edifici a Sophia Antipolis (Antibes)

# Quando sono nati gli algoritmi ?

- algoritmi sono stati inventati ben prima della nascita dei calcolatori:
  - uno dei più antichi è l'algoritmo di Euclide per il MCD;



- un altro è il crivello di Eratostene per la generazione dei numeri primi;

# Algoritmi per risolvere problemi

## Esempi di problemi algoritmici

- Problema dell'ordinamento di una sequenza: Data una sequenza di elementi, trasformarla in una sequenza ordinata secondo un certo criterio (ad es., ordinare alfabeticamente una lista di nomi, ordinare per prezzo crescente una lista di articoli di un supermercato, ecc.)
- Problema del cammino minimo: Data una mappa stradale che riporti località, strade e lunghezze dei percorsi, e data una coppia di località (raggiungibili l'una dall'altra), trovare il percorso di minor lunghezza fra di esse.
- Problema del commesso viaggiatore: Dato un insieme di località su una mappa stradale, trovare il percorso di minor lunghezza che visita tutte le città una volta sola.
- Problema dell'arresto dei programmi: Dato un qualunque programma per computer, con un certo input, stabilire in anticipo se esso prima o poi terminerà, o se "si pianta".

# Problemi e algoritmi.

- Un problema algoritmico è una descrizione precisa di cosa si vuole ottenere, cioè della relazione che deve valere fra i dati di partenza (input) e il risultato (output), ma non del modo in cui ottenerlo.
- Un algoritmo è un procedimento di calcolo ben definito: è quindi una descrizione di come risolvere un problema.
- Un algoritmo, quando viene espresso in un linguaggio di programmazione, diventa un programma, eseguibile da un computer.
- Uno stesso problema può essere risolto con algoritmi diversi, di diversa efficienza.
- Ad esempio, algoritmi di ordinamento diversi operano in modi diversi per raggiungere lo stesso risultato, con ben diverse "efficienze" (cioè tempo necessario).

# Algoritmi che risolvono problemi non precisamente definiti

Oggi in molti casi non vi è una specifica precisa di ciò che si vuole ottenere, ed è la costruzione dell'algoritmo stesso che definisce che cosa si ottiene.

Esempio:

- L'algoritmo di ranking di un motore di ricerca: dato un insieme di parole o di frasi fornite da un utente, produrre un elenco di siti web contenenti tali parole o frasi, in ordine di rilevanza per l'utente.
- La definizione del problema è vaga: ma l'algoritmo di ranking di Google ha risposto, come è ben noto, ai desideri degli utenti in modo molto migliore dei precedenti algoritmi, e ha decretato il successo planetario di Google.

# Algoritmi che risolvono problemi non precisamente definiti

- Data la sequenza degli acquisti effettuati in passato da un utente presso un venditore on-line (ad es. Amazon), dare come risultato un insieme di nomi di prodotti (ad es. titoli di libri, cd, dvd) al cui acquisto quell'utente potrebbe essere interessato.
  - Anche qui il problema non ha una definizione matematica precisa; ma l'algoritmo usato da Amazon sembra essere responsabile di una notevole percentuale delle vendite.
- L'algoritmo EdgeRank di Facebook, che determina quali post degli amici e followers ciascun utente vede.
- Gli algoritmi che cercano di ottimizzare il guadagno in borsa di un grande investitore ("algorithmic trading").
- ecc.

Parte 1: prima metà del '900.

Calcolabilità:

che cosa i calcolatori possono (e non possono) in linea di principio fare.

# Lo Entscheidungsproblem.

Nel 1928, al Congresso matematico mondiale di Bologna, il sommo matematico Hilbert, in un famoso intervento, aveva posto il seguente problema (sperando ottimisticamente in una risposta positiva):

esiste un procedimento meccanico il quale, data una qualunque teoria matematica e dato un qualunque enunciato nel linguaggio di quella teoria, permetta di stabilire se tale enunciato è o non è un teorema della teoria, cioè se è deducibile dagli assiomi?

*Procedimento meccanico* è ciò che oggi chiamiamo *algoritmo* !

Il grande logico contemporaneo francese Jean-Yves Girard ha, a posteriori, qualificato ironicamente questo genere di speranze come "tipiche dello scientismo tedesco".

# David Hilbert (1862 -1943)

Forse il più grande dei matematici a cavallo fra '800 e '900.

# L'importanza dei risultati negativi

Negli anni '30 un certo numero di matematici e logici vollero dimostrare che un tale procedimento non può esistere. Però:

- Finché si tratta di mostrare che un dato problema può essere risolto da un certo procedimento di calcolo, basta descrivere in modo preciso tale procedimento, in modo che chiunque possa eseguirlo, e dare una dimostrazione del fatto che esso risolve il problema. Oggi diremmo: basta dare l'algoritmo, insieme con una dimostrazione della sua correttezza.
- Ma per dimostrare che non può esistere alcun procedimento di calcolo che risolve un dato problema, bisognava preventivamente stabilire che cosa è un procedimento di calcolo, cosa vuol dire "calcolare". Occorreva cioè dare una definizione precisa e rigorosa della nozione intuitiva di "procedimento di calcolo".

# Che cosa vuol dire "calcolare"?

- I matematici e logici Church, Gödel, Kleene (e altri) diedero ciascuno una diversa definizione matematica (usando formalismi diversi, inventati allo scopo o inventati per altri motivi) di cosa vuol dire "calcolare".
- Ciascuno di essi dimostrò che, in base alla propria definizione di procedimento di calcolo, il procedimento richiesto dallo Entscheidungsproblem non poteva esistere.
- Dimostrarono che tali diverse definizioni erano tutte fra loro equivalenti, e che quindi esse verosimilmente catturavano la nozione intuitiva.
- Rimaneva però il dubbio (soprattutto in Gödel, il più grande logico del tempo) se in questo modo si catturasse davvero interamente la nozione intuitiva di calcolabilità.

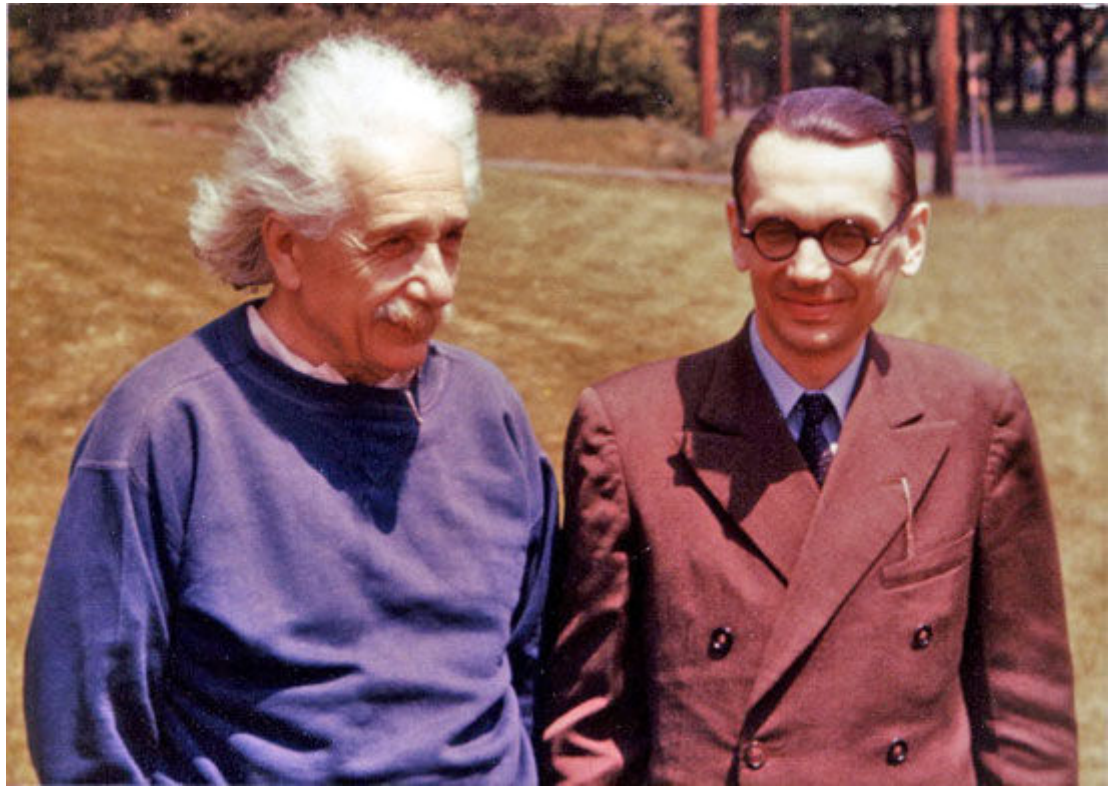
# Alonzo Church (Washington 1903 – Princeton 1995)

Matematico americano, nel 1932 pubblica un articolo in cui si propone di fondare la logica (e quindi la matematica) sul concetto di funzione invece che su quello di insieme:

A Set of Postulates for the Foundation of Logic

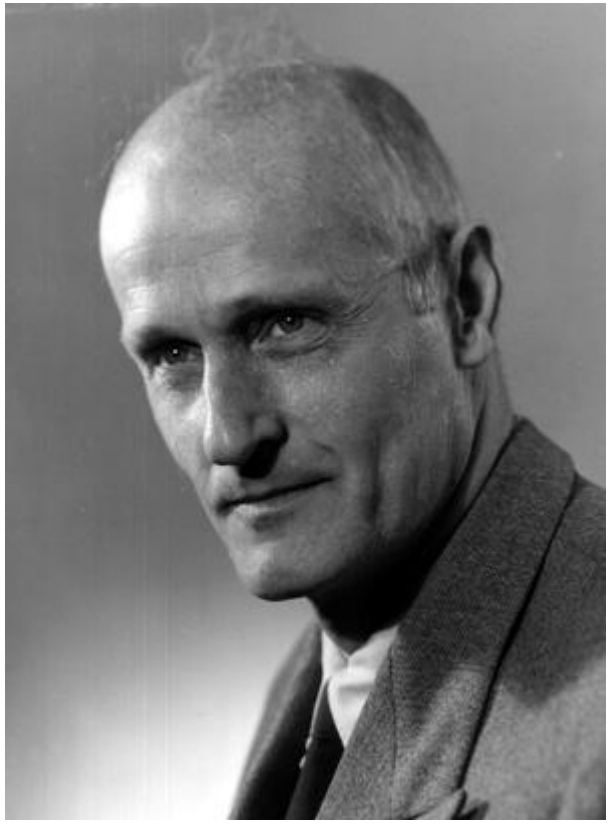
In esso faceva la sua comparsa un particolare formalismo con associato un calcolo: il $\lambda$-calcolo.

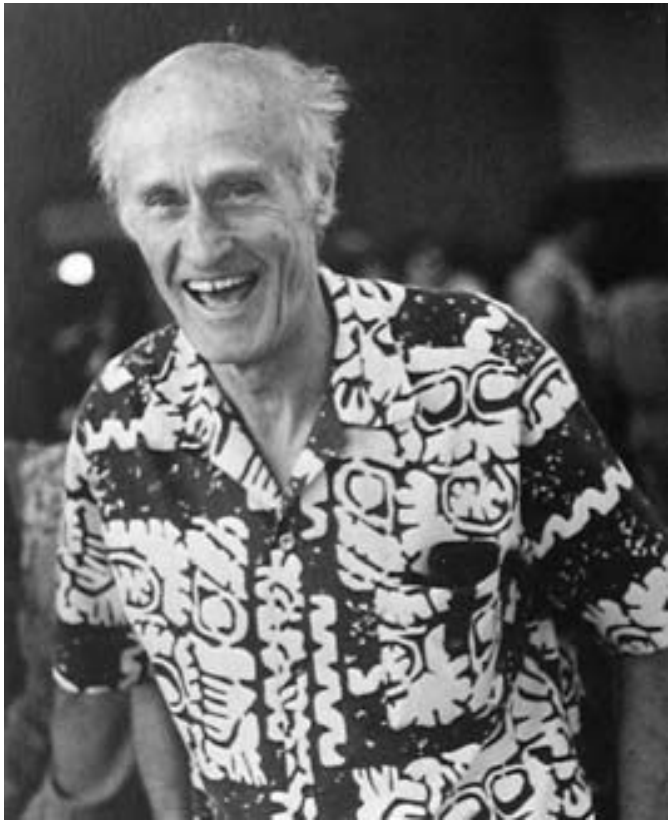
# Kurt Gödel (Brno/Brünn 1906 – Princeton 1978) in compagnia di un suo amico …



# Stephen Kleene

## Connecticut, 1909 – Wisconsin, 1994

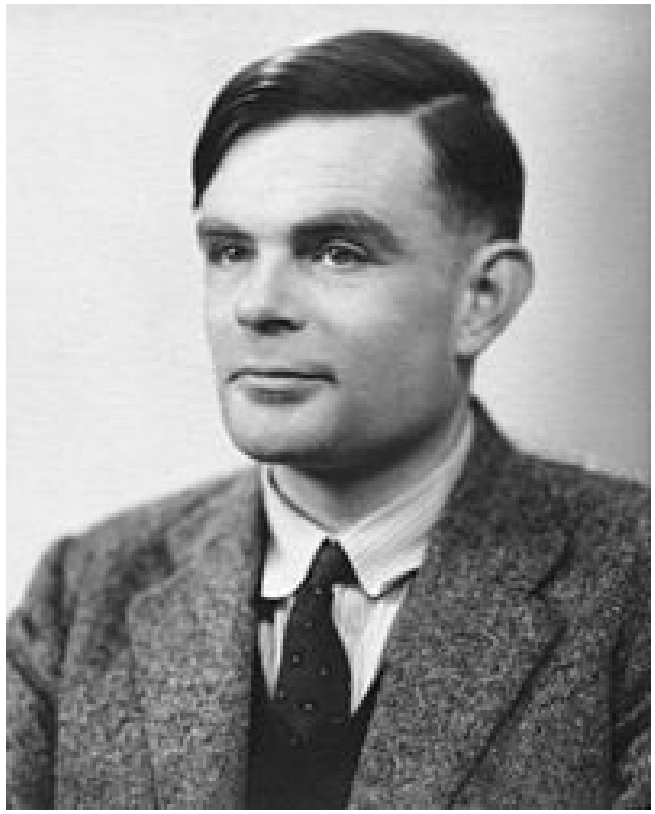




# Alan Turing

Nel 1936, un brillante matematico inglese di 24 anni, del King's College di Cambridge, Alan Turing, pubblica un articolo in cui dà della nozione di calcolabilità una definizione completamente diversa da quelle di Gödel, di Kleene e di Church, partendo da una accurata analisi delle azioni elementari di un uomo nell'atto di calcolare, "a man in the process of computing".

# Alan Turing, 1912-1954



On Computable Numbers, with an Application to the Entscheidungsproblem.
1936

# L'analisi di Turing dell'uomo che calcola.

Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book. In elementary arithmetic the two-dimensional character of the paper is sometimes used.

But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimen-sional paper, i.e. on a tape divided into squares.

I shall also suppose that the number of symbols which may be printed is finite. […]

# L'analisi di Turing (continuazione)

Nel 1936, quando Turing scriveva il suo articolo, i calcolatori elettronici non esistevano ancora, e la parola "computer" indicava un essere umano (di solito donna) addetto ai calcoli (ad es. delle tavole di tiro dei cannoni per l'esercito).

The behaviour of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment.

We may suppose that there is a bound to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations.

We will also suppose that the number of states of mind which need be taken into account is finite. [...] If we admitted an infinity of states of mind, some of them will be "arbitrarily close" and will be confused.

# L'analisi di Turing (continuazione).

L'azione del calcolatore (umano) è dunque determinata dalla coppia ⟨stato della mente, simbolo letto sul nastro⟩**,** e poiché sia l'insieme dei simboli che l'insieme degli stati sono finiti, anche l'insieme delle possibili azioni specificate è finito.

La quantità di carta a disposizione del calcolatore umano è invece illimitata, cioè il nastro quadrettato è potenzialmente infinito, benché ad ogni dato istante solo un numero finito di quadretti sia non bianco.

Turing conclude la sua analisi osservando che un'azione elementare può essere la scrittura di un simbolo nel quadratino esaminato (se questo è bianco), oppure la cancellazione del simbolo contenuto nel quadratino, oppure lo spostamento dello "sguardo" su un quadratino adiacente; ognuna di tali azioni può essere accompagnata da un passaggio della mente da uno stato a un altro.

# Macchine di Turing

Il calcolatore umano (computer) può pertanto essere simulato da una macchina:

"we may now construct a machine
to do the work of this computer"

La macchina di Turing non è il progetto di una macchina reale, bensì una macchina astratta, costituita da:

- una testina di lettura-scrittura
- un nastro quadrettato potenzialmente infinito;
- uno stato interno (lo "stato della mente" del calcolatore umano), appartenente ad un insieme finito di stati;

A ciascun istante la macchina si trova in un ben determinato stato interno e la testina è posizionata su uno dei quadretti.

La macchina "calcola" compiendo una successione di passi elementari.

# La macchina di Turing

Ad ogni passo, la macchina:

- legge il simbolo contenuto nel quadretto (l'alfabeto dei simboli è fissato per ciascuna macchina, ed è finito; e contiene un simbolo indicante il "quadretto bianco")
- a seconda del simbolo letto e del proprio stato interno, esegue una ben determinata azione, consistente in:
  - scrivere un simbolo nel quadretto, cancellando quello appena letto;
  - spostare la testina a destra o a sinistra di un quadretto, oppure lasciarla ferma.

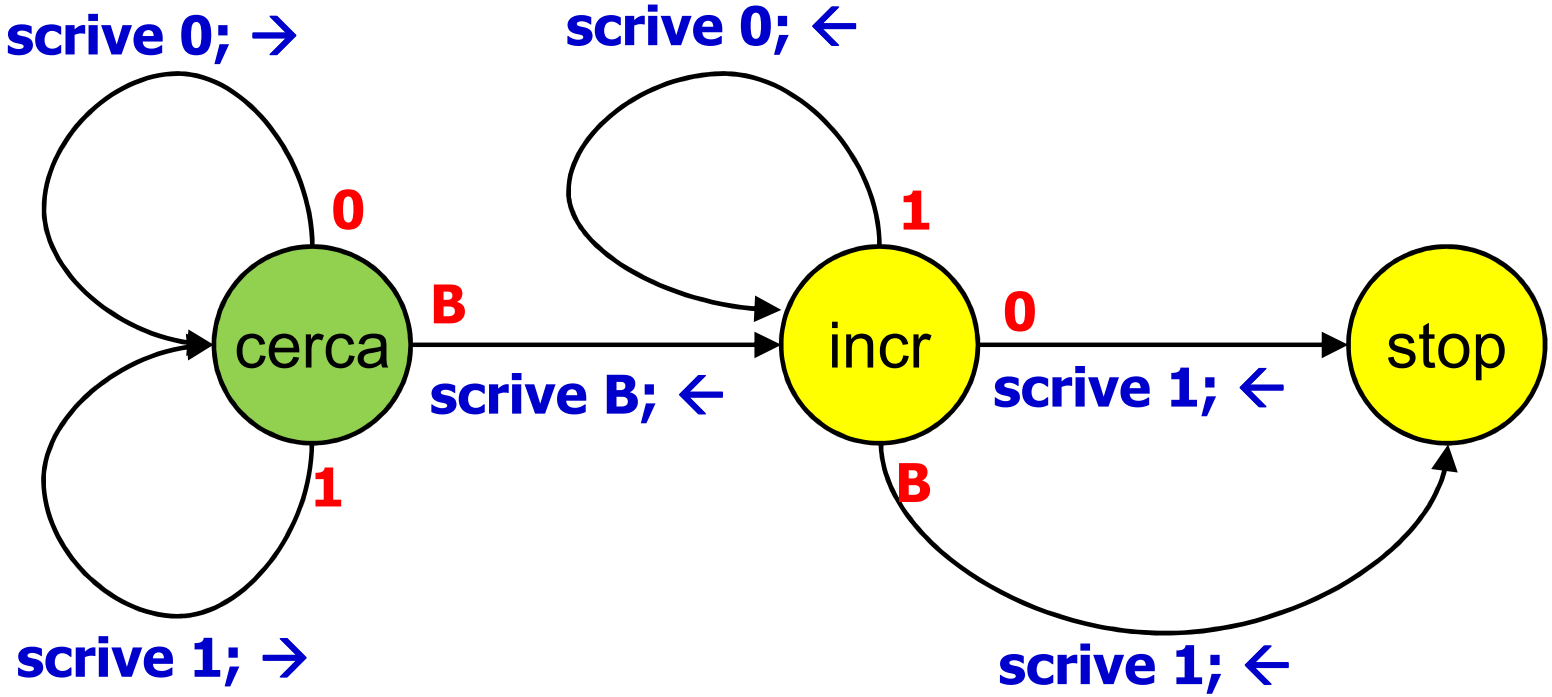
Una macchina di Turing è dunque descrivibile da una tabella in cui per ogni coppia ⟨simbolo letto, stato interno⟩ si specifica l'azione compiuta dalla macchina e il nuovo stato in cui entra.

# Esempio: una macchina che aggiunge 1 a un numero in notazione binaria.

La macchina lavora con un alfabeto di tre simboli: 0, 1, bianco, e ha tre stati interni: cercaBitDestro, incrementa, e stop. La sua tabella è la seguente:

| stato | simbolo letto | scrive | si sposta | nuovo stato |
|---|---|---|---|---|
| **cercaBitDestr** | **bianco** | **bianco** | ← | **incrementa** |
| **cercaBitDestr** | **0** | **0** | → | **cercaBitDestr** |
| **cercaBitDestr** | **1** | **1** | → | **cercaBitDestr** |
| **incrementa** | **bianco** | **1** | ← | **stop** |
| **incrementa** | **0** | **1** | ← | **stop** |
| **incrementa** | **1** | **0** | ← | **incrementa** |

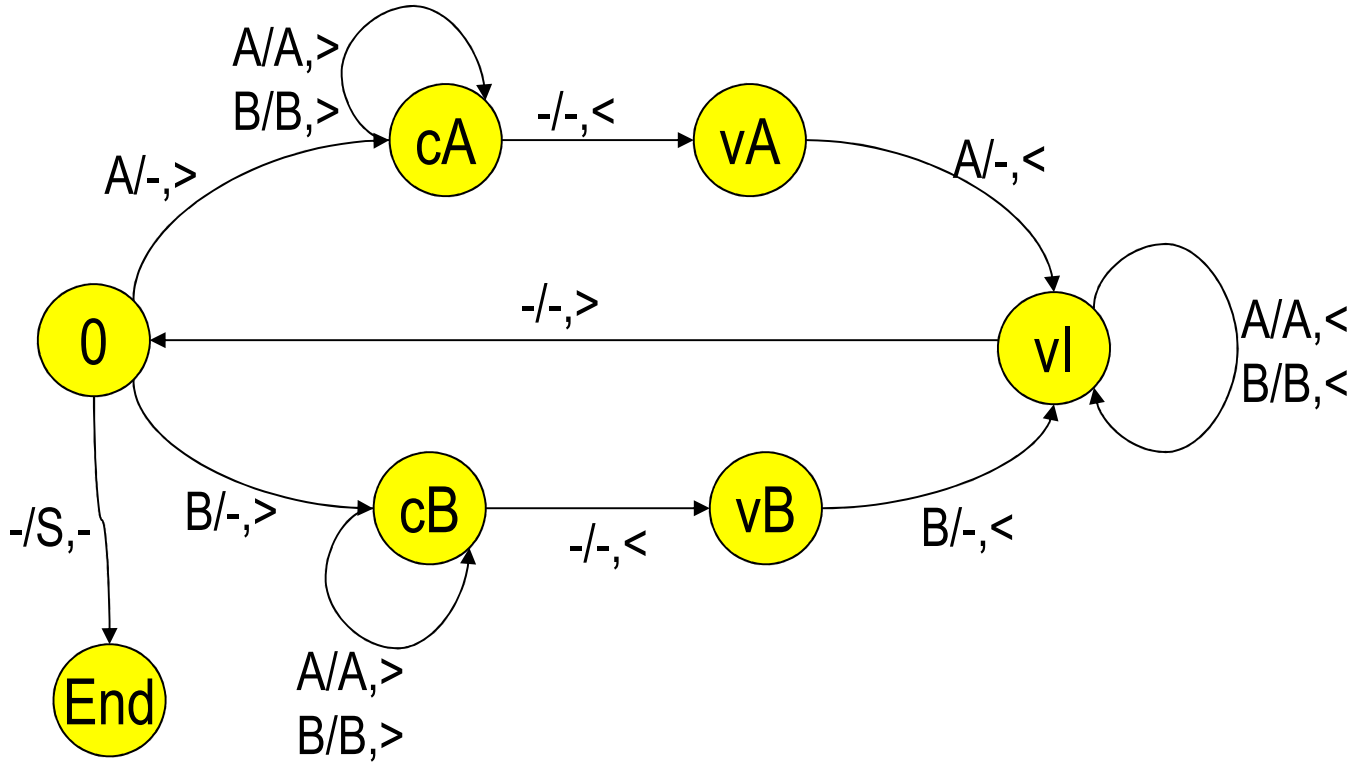
# Una rappresentazione grafica.




dove: lo stato iniziale è "cerca",

B = bianco, ← = "si sposta a sinistra", → = "si sposta a destra"

# Altro esempio: una macchina controlla-palindrome



# Equivalenza fra la definizione di Turing e le altre.

Nel 1937 Turing dimostrò che anche la sua definizione, pur così diversa, era equivalente alle precedenti (Church, Gödel, Kleene, ecc.): ogni procedimento di calcolo effettuabile con le sue macchine poteva essere tradotto in un procedimento di calcolo secondo le precedenti definizioni, e viceversa.

La definizione di Turing era però molto più persuasiva delle altre, e fu quella che convinse Gödel che la nozione rigorosa di calcolabilità individuata da tutte queste definizioni equivalenti corrispondeva davvero alla nozione intuitiva:

"[...]The most satisfactory way, in my opinion, is that of reducing the concept of finite procedure to that of a machine with a finite number of parts, as has been done by the British mathematician Turing."

"I was completely convinced only by Turing's paper".

# Turing come Michelangelo?

Un informatico americano, Robert Soare, in vena di paragoni straordinari, ha paragonato la differenza fra le definizioni di computabilità di Church e di Turing a quella fra i David di Donatello e di Michelangelo:

"Michelangelo and Turing both completely transcended conventional approaches. First ,they both created something completely new from their own visions, something which went far beyond the achievements of their contemporaries.

Second, both emphasized the human form.

Michelangelo brought out the human form in his statues and in the Sistine ceiling with magnificent human figures often shown in contraposto. Turing left behind the formal systems of lambda-definable or recursive functions. Turing searched into how a human being actually computes. ..."

# Varietà delle macchine

Macchine di Turing diverse, cioè con insiemi di regole diversi, effettuano processi di calcolo diversi.

Ciascuna particolare macchina realizza quindi un particolare algoritmo, e viceversa un particolare algoritmo può essere realizzato da (almeno) una macchina.

# Macchina di Turing universale.

Turing dimostrò che si può definire una MdT universale, cioè una MdT la quale, presi come input sul nastro:

- una descrizione, in un opportuno linguaggio di codifica, di una qualunque MdT particolare, cioè di un qualunque algoritmo,
- e un input per tale algoritmo,

è in grado di eseguire l'algoritmo su quell'input.

Una MdT universale è perciò un modello astratto di computer:

- il nastro della MdT universale rappresenta sia i dispositivi di input-output che la memoria (potenzialmente infinita);
- il linguaggio in cui devono essere descritti gli algoritmi è il linguaggio-macchina della MdT universale.

# Problemi insolubili.

La definizione precisa e matematicamente rigorosa della nozione intuitiva di procedimento di calcolo, data in forme diverse ma equivalenti da Church, Turing, ecc. permise di dimostrare che vi sono problemi, come l'Entscheidungsproblem, che non sono risolubili da alcun algoritmo.

Il più noto di tali problemi, in realtà connesso alll'Entscheidungs-problem, è il problema dell'arresto dei programmi, citato nelle prime slides, detto anche "problema della fermata", o in inglese "halting problem".

Turing dimostrò che non può esistere un algoritmo il quale, presi in input il testo di un qualunque programma e un input per quel programma, è sempre in grado, esaminando il programma e il suo input, di stabilire a priori se, eseguendo il programma con quell'input, esso terminerà o no.

# Il problema della fermata del tram (*)

- Se il tram che dobbiamo prendere passerà alla fermata, aspettando abbastanza a lungo lo scopriremo. Se lo aspettiamo e non arriva, non sappiamo se passerà oppure se ha cambiato percorso o c'è un incidente, o c'è sciopero.
- Per stabilire se un programma (con un certo input) termina o no, non si può provare ad eseguirlo: infatti se lo vediamo terminare scopriamo ovviamente che termina, ma se non lo vediamo terminare non sappiamo se terminerà.
- Se il programma termina, aspettando abbastanza a lungo (ma non sappiamo quanto a lungo !) lo scopriremo.
- Se il programma non termina, non lo sapremo mai !
- L'unico modo per stabilire che un programma non termina è dimostrare che non termina, analizzando il testo del programma ma senza eseguirlo !

(*) L'esempio è preso, con adattamento, da presentazioni e articoli di J.-Y. Girard.

# Dimostrazione dell'insolubilità dello halting problem

Alan Turing, 1936

(riformulata, come è d'uso, in termini moderni)

# Premessa

Non è strano che un programma prenda in input un altro programma: ad esempio il "compilatore" di un linguaggio di programmazione (C, C++, Java, ecc.) prende in input un testo e controlla se esso costituisce un programma corretto in quel linguaggio, cioè se obbedisce a tutte le regole sintattiche del linguaggio:

- se no, genera in output delle segnalazioni di errore;
- se si, produce in output una traduzione del programma in linguaggio-macchina oppure in un linguaggio intermedio, che a sua volta verrà poi tradotto in istruzioni della macchina.

Più banalmente, si può scrivere un programma che conta le righe di cui è composto il testo di un altro programma.

Non è strano neppure che un programma possa prendere in input se stesso: ad es. un programma che conta le righe, se gli viene dato in input il testo di se stesso, ne conta le righe!

# Un programma che controlla la terminazione.

Vogliamo scrivere un programma controlloTerminazione che, presi in input:

- il testo di un qualunque programma P,
- e il testo di un qualunque input Inp per quel programma,

compie delle analisi sui due testi e produce sempre come risultato, dopo un tempo finito:

- la scritta "termina" oppure la scritta "non termina".

Non è difficile scrivere un programma controlloTerminazione che scopra alcuni casi evidenti di non-terminazione: ad es., quando il programma esaminato è costituito da un ciclo *while* che per certi valori dell'input si vede subito che non termina.

# Esempio

Il banale programmino Python seguente:

```
n = int(input("digita un intero: "))
while n != 100 :
  print(n)
  n=n+1

print("ho finito")
```

se l'input è un numero n < 100, scrive sullo schermo tutti gli interi da n a 99 e poi termina;

se l'input è 100, termina subito senza scrivere niente;

se l'input è n > 100, continua a scrivere tutti gli interi da n in avanti senza mai arrestarsi.

È ovviamente possibile incorporare tale conoscenza in un programma controlloTerminazione (vedi slide successiva)

# Esempio di programma controlloTerminazione

```
nomeFile = input("digita il nome del programma: ")
prog = open(nomeFile, 'r')
inp = input("digita l'input per quel programma: ")
riga1 = prog.readline()
riga2 = prog.readline()
riga3 = prog.readline()
riga4 = prog.readline()

if riga1 == 'n = int(input("digita un intero: "))\n' \
  and riga2 == 'while n != 100:\n' \
  and riga3 == '  print(n)\n' and riga4 == '  n=n+1\n':
  if int(inp) <= 100: print("termina")
  else: print("non termina")
else:
  print("non so se termina o no")
```

Il problema è che controlloTerminazione dovrebbe scoprire tutti i possibili casi di non-terminazione, non solo qualche caso particolare, e quindi non dovrebbe mai produrre la scritta "non so se termina o no".

Il numero di possibili programmi diversi è infinito (come infinito è il numero di testi finiti che si possono scrivere in una lingua), ma potremmo sperare che un programma sufficientemente complicato riesca ad individuare tutte quelle caratteristiche che rendono non-terminante un programma per un dato input.

Dimostriamo che non è possibile, attraverso una dimostrazione per assurdo.

# Uno strano programma.

Assumiamo dunque che si sia riusciti a scrivere un programma controlloTerminazione il quale, presi due input:

- il testo di un qualunque programma P,
- e il testo di un qualunque input Inp per quel programma,

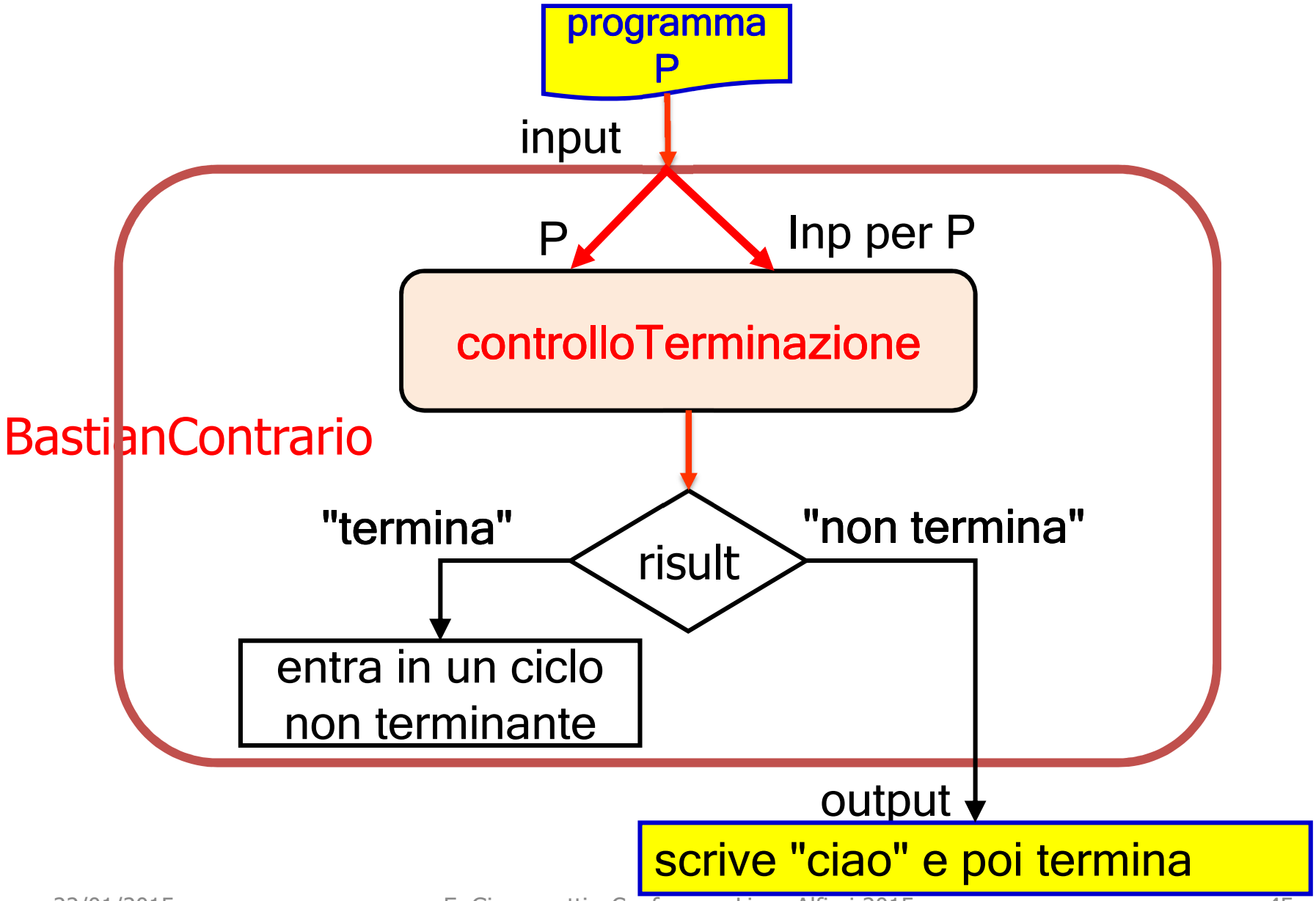
produce sempre come risultato, dopo un tempo finito:

- la scritta "termina" oppure la scritta "non termina".

Allora è possibile scrivere un programma BastianContrario che opera nel modo seguente:

- prende come unico input un qualunque programma P;
- usa il programma controlloTerminazione per farsi dire se il programma P con input se stesso termina oppure no; poi:
  - se la risposta è "termina", entra in un ciclo non-terminante;
  - se la risposta è "non termina", termina la propria esecuzione

# Schematicamente:



# Osservazione

Osserva che se il programma P è tale che non ha senso che esso abbia come input se stesso (ad es. perché si aspetta in input un numero e non un testo), se ciò nonostante gli si dà come input se stesso, il programma usualmente terminerà con un errore.

Quindi in tal caso il programma con input se stesso termina, anche se non fornisce alcun risultato utile.

Oppure, in qualche caso, il programma potrebbe "piantarsi", cioè non terminare.

Ha comunque sempre senso parlare di un programma che prende come input se stesso.

# Nota Bene

- Il programma BastianContrario è strano, ma non ha in sé nulla di contradditorio;
  se il programma controlloTerminazione esiste,
  BastianContrario può essere facilmente realizzato.
- BastianContrario è semplicemente un programma che prende un qualunque programma P come input, e poi "chiede" al programma controlloTerminazione se P con input se stesso termina oppure no;
- ottenuta la risposta (che per ipotesi controlloTerminazione fornisce sempre) fa "personalmente" il contrario di quello che controlloTerminazione "dice" che P con input se stesso fa.

# Esempio

Se si dà in input a BastianContrario un programma contaRighe corretto, il controlloTerminazione "dirà" a BastianContrario che contaRighe con input contaRighe termina, e allora BastianCon… entra in un ciclo non terminante.

Quindi il programma BastianContrario, con input contaRighe, non termina.

Se invece a BastianContrario si dà in input una versione errata del programma contaRighe la quale, supponiamo, per files più lunghi di tre righe "si pianta", controlloTerminazione "dirà" che contaRighe con input contaRighe non termina, e allora Bastian… scrive sullo schermo "ciao" e termina.

Quindi il programma BastianContrario, con input il contaRighe errato, termina.

Dov'è allora il problema?

# Eccolo!

Che cosa succede se al programma BastianContrario si dà come input BastianContrario stesso?
Cioè, come si comporta BastianContrario con input se stesso?

"Chiede" a controlloTerminazione qual è il comportamento di BastianContrario con input BastianContrario, e poi fa esattamente il contrario:

- se controlloTerminazione "dice" che BastianContrario con input se stesso termina, allora BastianContrario con input se stesso non termina;
- se controlloTerminazione "dice" che BastianContrario con input se stesso non termina, allora BastianContrario con input se stesso termina;

Qualunque sia il risultato fornito da controlloTerminazione, BastianContrario si incarica, per così dire, di renderlo sbagliato!

# Conclusione

Il programma controlloTerminazione non può quindi fornire sempre la risposta giusta.

Il programma BastianContrario con input se stesso sfrutta il risultato di controlloTerminazione per fare esattamente il contrario di ciò che controlloTerminazione "gli dice" che lui (BastianContrario) fa !

Quindi il metodo controlloTerminazione, se dà un risultato, lo dà in ogni caso sbagliato, perché BastianContrario lo rende sbagliato.

# Halting problem e Entscheidungsproblem.

Turing dimostrò che se l'Entscheidungsproblem fosse risolubile, allora sarebbe risolubile anche il problema della fermata.
Ma poiché il problema della fermata non è risolubile, allora non è risolubile nemmeno l'Entscheidungsproblem.

La dimostrazione si basa sul seguente ragionamento.

Si può costruire una teoria assiomatica dei programmi, in cui si possono scrivere degli enunciati equivalenti ad affermazioni del tipo "il programma P termina".

Per la definizione, se l'Entscheidungsproblem fosse risolubile, esiterebbe un algoritmo il quale, data una qualunque teoria assiomatica (del prim'ordine) e dato un qualunque enunciato nel linguaggio di quella teoria, stabilisce se tale enunciato è o non è un teorema della teoria.

Ma allora, applicando tale algoritmo alla suddetta teoria dei programmi, si risolverebbe il problema della fermata!

# Perché è importante questo risultato?

Perché si può facilmente dimostrare che numerosi altri problemi generali, quali ad esempio l'equivalenza fra programmi o la correttezza di programmi rispetto a specifiche, sono insolubili, perché se avessero delle soluzioni (cioè se ci fossero degli algoritmi che li risolvono), queste potrebbero essere sfruttate per risolvere lo halting problem.

Il risultato di Turing fa quindi vedere che ci sono "cose che i calcolatori non possono fare".

Ciò non impedisce di sviluppare degli algoritmi che sono in grado di stabilire se un programma termina o no all'interno di una ben definita classe di programmi, che pur non essendo la classe di tutti i programmi possibili, contiene una grande parte di programmi di rilevanza pratica. La MicroSoft, ad es., ha in corso un progetto di ricerca chiamato Terminator che ha proprio tale obiettivo.

# Importanza del risultato di Turing per la matematica.

Se lo halting problem fosse stato risolubile, avrebbe permesso di risolvere difficiissimi problemi matematici.

Consideriamo la famosa congettura di Golbach:
Ogni numero pari maggiore di due può essere espresso (eventualmente in più di un modo) come la somma di due numeri primi (non necessariamente distinti).

Esempi.
4 = 2+2
6 = 3+3
8 = 5+3
10 = 7+3 = 5+5
12 = 7+5
…
100 = 97+3 = 47+53 = …
…

È facile scrivere un programma goldbach per computer che:

considera uno dopo l'altro tutti i successivi numeri pari 2,4,6, … e per ognuno di essi cerca due numeri primi la cui somma dia quel numero: se trova un numero pari che non è esprimibile come somma di due numeri primi, si interrompe scrivendo sullo schermo "la congettura di Goldbach è falsa!".

Se esistesse un programma controlloTerminazione, allora dandogli in input goldbach e leggendo la risposta, sapremmo che:

se controlloTerminazione mi dice che goldbach non termina, allora la congettura è vera (come si suppone);

ovviamente se mi dicesse che è falsa, allora la congettura sarebbe falsa.

# Conclusione

- Nella prima metà del '900 viene trovata una nozione ben precisa, definita rigorosamente, corrispondente alla nozione intuitiva un po' vaga di "procedimento di calcolo".
- Viene tracciata una netta distinzione fra problemi che sono risolubili, almeno in linea di principio, con un procedimento di calcolo, e problemi che non lo sono.
- Il concetto fondamentale è quindi la calcolabilità.
- Vengono progettate e costruite delle macchine in grado di eseguire qualunque possibile procedimento di calcolo, cioè di risolvere, almeno in linea di principio, qualunque problema "calcolabile": sono i nostri computer.