Una lezione (quasi) universitaria di Informatica.

Liceo Scientifico Statale "Galileo Ferraris".

Torino, 27 aprile 2007

Parte V

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

Informatica teorica e matematica: un problema da un milione di dollari.

Problema algoritmico: Data una formula booleana composta di n variabili booleane, ad esempio

A and (B or (not B and C) xor …

stabilire se esiste oppure no una assegnazione di valori di verità alle variabili che renda vera la formula.

- L'unico algoritmo risolvente conosciuto è di fatto quello banale che prova tutte le possibili combinazioni di valori di verità; il suo tempo di calcolo cresce esponenzialmente rispetto ad n, quindi tale algoritmo è usabile solo per formule "piccole".
- Si ha la fondata convinzione che non possa esistere un algoritmo più veloce che esponenziale.
- Ma nessuno è ancora riuscito a dimostrare che un tale algoritmo non può esistere !

Un problema da un milione di dollari.

Il problema è un caso particolare di un problema più generale di informatica teorica classificato come uno dei "Problemi matematici del Millennio", per ognuno dei quali è in palio un premio di un milione di dollari (Istituto Clay).

La soluzione (che vincerà il premio) consisterà quindi in una delle due possibili alternative seguenti:

- o si riuscirà a trovare un algoritmo con tempo di calcolo meno che esponenziale (ad esempio quadratico, o cubico, ecc.): ciò è ritenuto altamente improbabile;
- oppure si riuscirà a dimostrare che un tale algoritmo non può esistere (è la dimostrazione che viene cercata).

Buona fortuna !

Problemi "intrinsecamente" difficili
cioè problemi algoritmici per i quali si sa (= si è dimostrato) che non possono esistere soluzioni efficienti.

Esempi

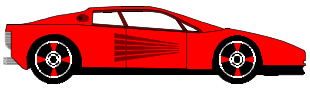
Il problema delle Torri di Hanoi.

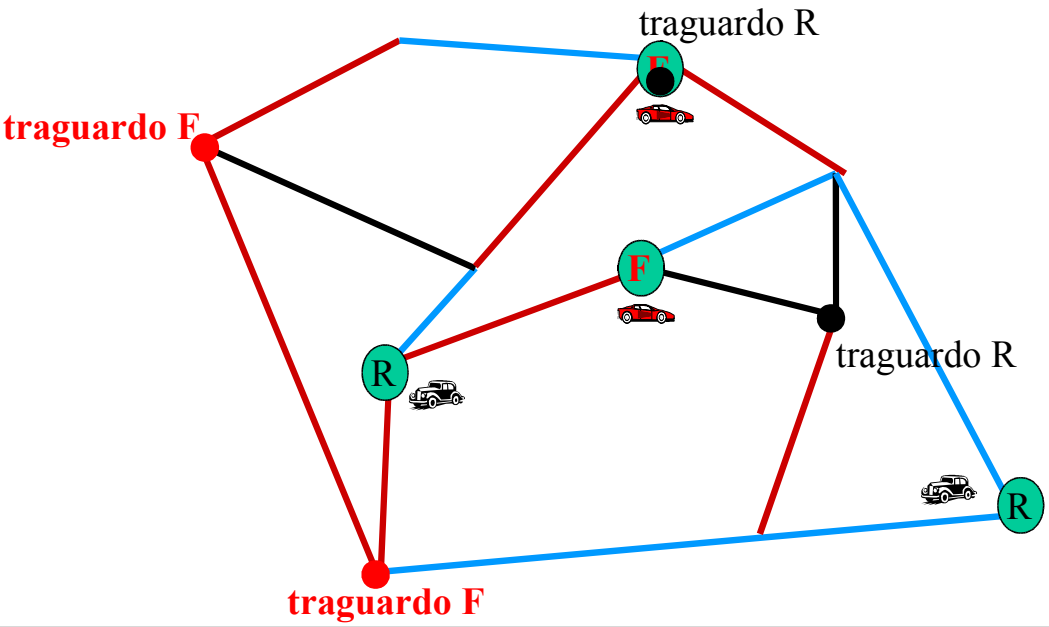
Il problema *dei blocchi stradali (roadblocks).*

Il problema dei blocchi stradali: il gioco.

- È un gioco con due giocatori (Ferrari e Renault) su una rete di strade che si intersecano; ogni tratto di strada fra due intersezioni è colorato con uno fra tre colori possibili.
- Alcune intersezioni sono marcate “traguardo Ferrari”, altre sono marcate “traguardo Renault”, le rimanenti non sono marcate. Ognuno dei due giocatori possiede un certo numero di auto, ciascuna delle quali inizialmente occupa un’intersezione.
- A turno ogni giocatore sposta una delle sue auto da un’inter-sezione ad un’altra, lungo un percorso che deve essere tutto dello stesso colore, e su cui non si devono trovare altre auto.
- Vince il primo giocatore che raggiunge con un’auto un’intersezione marcata come proprio traguardo.

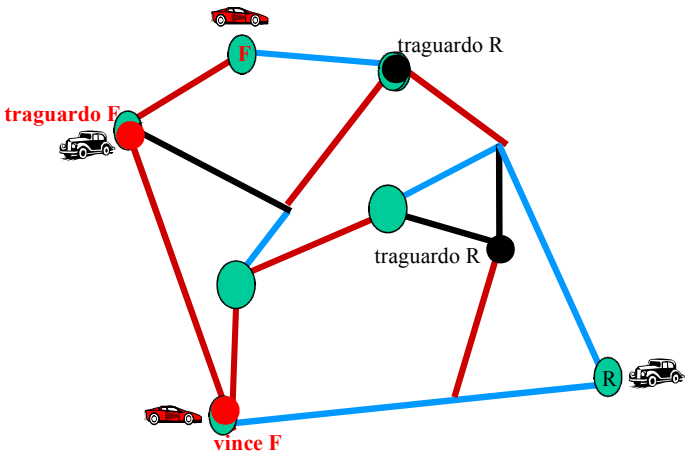
Blocchi stradali





Blocchi stradali

Se F muove per primo, può vincere quali che siano le mosse di R.



Il problema

- Determinare se, data un'arbitraria configurazione iniziale del gioco, esiste una strategia vincente per il giocatore che gioca per primo.
- L'algoritmo che risolve il problema impiega un tempo esponenziale nel numero delle intersezioni.
- Nota: occorrerebbe un tempo esponenziale anche solo per dimostrare che una data risposta è corretta.
- Si dimostra che non può esistere un algoritmo migliore.

Telecomunicazioni

- Problema di trovare un percorso non congestionato in una rete:
  - La congestione può essere definita in modo preciso per mezzo del ritardo nella trasmissione e della diminuzione della larghezza di banda.
  - La congestione è variabile nel tempo.
- La soluzione esatta del problema, formulato in modo preciso, è un algoritmo anch'esso esponenziale, e si dimostra che non può esistere un algoritmo migliore.

Correttezza e terminazione.

- Naturalmente, un algoritmo che risolva correttamente un problema deve terminare con la risposta corretta per ogni input soddisfacente alla precondizione.
- Un algoritmo che per alcuni valori dell'input termina con la risposta corretta mentre per altri non termina è un algoritmo solo *parzialmente corretto*, che di solito non ha molta utilità.
- Consideriamo ora un algoritmo dato, o più concretamente un programma Pascal dato, indipendentemente dal problema che esso dovrebbe risolvere, o anche senza aver specificato alcun problema da risolvere; chiediamoci se, esaminando il programma senza eseguirlo, possiamo ricavare alcune informazioni sul suo comportamento: ad esempio scoprire se esso termina per ogni input oppure no, oppure se è equivalente ad un altro programma, ecc.

Terminazione

Il più corto programma TurboPascal che non termina:

```
begin
  while true do
end.
```

Un programma TurboPascal che termina se l'input è un numero intero positivo dispari, altrimenti non termina:

```
var n: integer;
begin
  write('immetti un intero: ');
  readln(n);
  while n <> 1 do begin
    n:= n-2;
    write(n, ', ');
  end;
  writeln;
  writeln('ho finito');
  readln
end.
```


Un problema aperto: la sequenza di Collatz

```
var n: integer;
begin
  write('immetti un intero positivo: ');
  readln(n);
  while n > 1 do begin
    if n mod 2 = 0 then n:= n div 2
    else n:= 3*n + 1;
  end;
  writeln(n);
end.
```

Il programma precedente termina (scrivendo 1) per ogni valore (intero positivo) dell'input per cui è stato provato; tuttavia non si è ancora riusciti a dimostrare che esso termina per tutti i possibili valori (interi positivi) dell'input.

Sequenza di Collatz e matematica "sperimentale".

In altre parole, sperimentalmente la sequenza di Collatz termina sempre (con 1), ma non si è riusciti a dimostrare che non può esistere un qualche numero per il quale non termina.

Cioè si è convinti, sulla base dei dati sperimentali (la cosiddetta, con inglesismo, *evidenza empirica*), che la sequenza di Collatz termini sempre, ma non si è ancora riusciti a capire perché.

Matematica, logica, e metodo assiomatico.

- la lingua naturale in cui si esprime la matematica viene sostituita da un linguaggio artificiale con sintassi rigida, come quella dei linguaggi di programmazione;
- tutte le regole puramente logiche del ragionamento corretto sono tradotte in un insieme di regole meccaniche (regole per la logica predicativa);
- ogni teoria matematica è tradotta in un insieme di assiomi e/o regole da aggiungersi alle regole logiche;
- ogni dimostrazione di un teorema può, in linea di principio, essere trascritta in una *derivazione* formale, cioè in un "calcolo" puramente meccanico in cui si applicano solo le regole logiche e le regole e assiomi della teoria;
- se esteso a tutti i campi del sapere e della vita, il metodo realizzerebbe l'ideale del "Calculemus !" di Leibnitz ...

David Hilbert, Königsberg, 8 settembre 1930

Für uns gibt es kein *Ignorabimus*, und meiner Meinung nach auch für die Naturwissenschaft überhaupt nicht. Statt des törichten *Ignorabimus* heiße im Gegenteil unsere Losung:

wir müssen wissen,
wir werden wissen.

traduzione:
Per noi non ci sono *ignorabimus*, e a mio parere anche nella scienza non ce n'è assolutamente nessuno. Invece dello sciocco *ignorabimus*, il nostro motto è, al contrario:

dobbiamo sapere, e sapremo.

Soluzione finale ?

Una volta individuato un insieme sufficientemente potente di assiomi per una teoria, tutti i problemi della teoria avrebbero potuto essere risolti in linea di principio meccanicamente: una "soluzione finale" ai problemi della matematica ...

Henry Poincaré, già nel 1905, ironizzava:

Ainsi c'est bien entendu, pour démontrer un théorème, il n'est pas nécessaire ni même utile de savoir ce qu'il veut dire. On pourrait remplacer le géomètre par le *piano à raisonner* imaginé par Stanley Jevons ; ou, si l'on aime mieux, on pourrait imaginer une machine où l'on introduirait les axiomes par un bout pendant qu'on recueillerait les théorèmes à l'autre bout, comme cette machine légendaire de Chicago où les porcs entrent vivants et d'où ils sortent transformés en jambons et en saucisses. Pas plus que ces machines, le mathématicien n'a besoin de comprendre ce qu'il fait.

Henri Poincaré, « Les mathématiques et la logique » in Revue de Métaphysique et de Morale, 1905, p. 815-835

Kurt Gödel, 1930

Scrive il famoso articolo

"*Über formal unentscheinbare Sätze der* Principia Mathematica *und verwandter Systeme I"*

(Sulle proposizioni formalmente indecidibili dei Principia Mathematica *e sistemi affini I")*

Problemi algoritmici non risolubili

Non tutti i problemi algoritmici sono risolubili: per alcuni problemi algoritmici si può dimostrare che non può esistere una soluzione, cioè che non può esistere un algoritmo che risolve il problema.

Esempi

Dati due programmi arbitrari, determinare se essi sono equivalenti, cioè se per qualunque input producono lo stesso output.

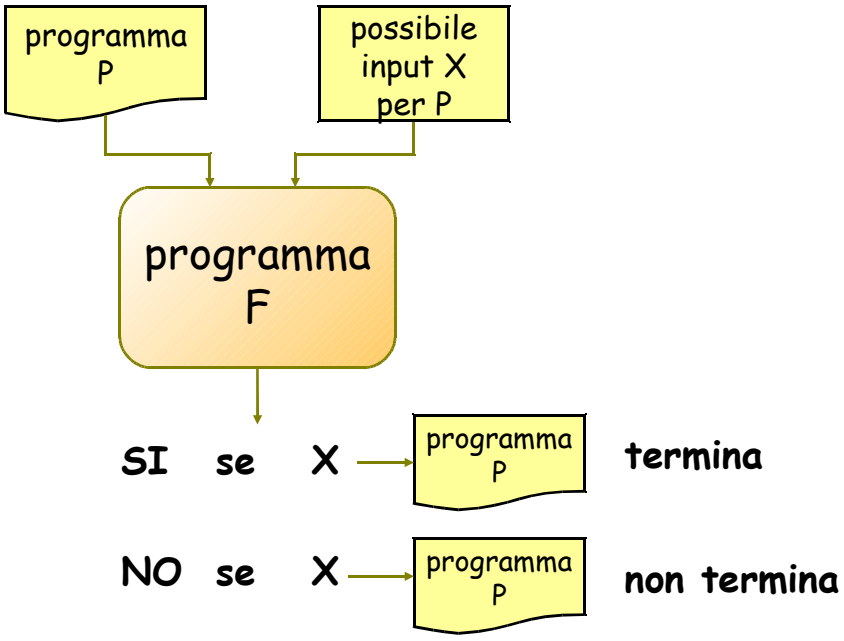
Dato un programma arbitrario, ed un input legale arbitrario per tale programma, determinare se il programma, per quell'input, termina.

Turing, 1936
Indecidibilità del problema della fermata.

Teorema:

Non può esistere un programma (o una macchina) il quale, dato un qualunque programma P e un input X per P, sia sempre in grado di determinare se il programma P eseguito con input X termina oppure no.

Cioè non può esistere un programma F tale che:



Nota Bene

Il programma F non può consistere semplicemente nel lanciare l'esecuzione del programma P con input X, poiché, se P non termina, il programma F (esattamente come noi) non lo scoprirà mai.

Il programma F deve evidentemente stabilire se P termina (con input X) senza eseguire P ma solo analizzandone il testo.

Ad esempio, se il testo di P è:

```
while X = 3 do X:= X;
```

una semplice analisi, che può essere incorporata in F, mostra che se X è 3 il programma P non termina, mentre per X diverso da 3 il programma P termina.

Nota Bene 2

Come si vede dal banale esempio precedente, un programma che sia in grado di riconoscere alcuni programmi che non terminano non è impossibile da realizzare.

Ciò che che è impossibile è un programma F il quale sia in grado, per ognuno di tutti i possibili programmi P e input X, di stabilire se esso termina o no con quell'input.

Nota Bene 3

Un programma è un testo o, in forma binaria, una sequenza di bit, e quindi può sempre essere l'input di un altro programma, eventualmente anche di se stesso.

Ad esempio si può scrivere in Pascal un compilatore Pascal, e poi farlo compilare a se stesso ...

È la tecnica cosiddetta del bootstrap, cioè del sollevarsi tirandosi per le stringhe delle proprie scarpe, come il Barone di Münchausen.

Dimostrazione

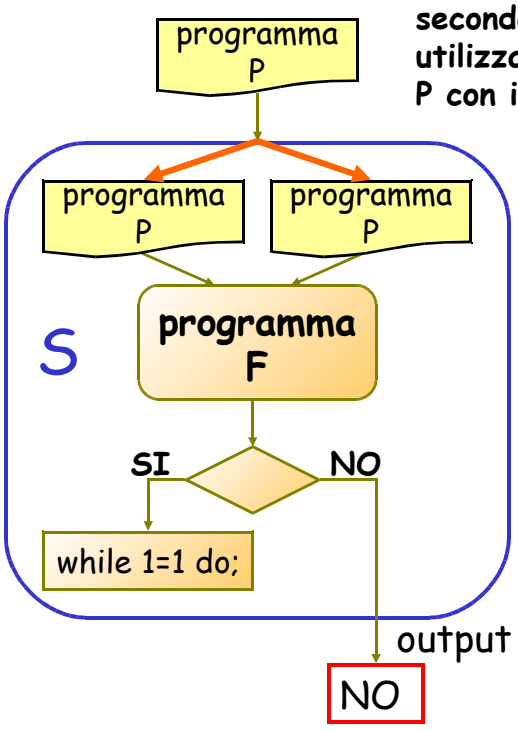
Assumiamo che un siffatto F esista, e mostriamo che ciò conduce ad una contraddizione.

Sia dunque F un programma che, per ogni programma P e ogni dato X, produce la risposta SI se il programma P con input X termina, produce la risposta NO se il programma P con input X non termina.

Allora si può facilmente realizzare il programma S rappresentato schematicamente nella slide seguente.

Il programma S:

Il programma S crea una seconda copia di P, poi utilizza S per stabilire se P con input P termina.



Come si comporta il programma S ?

Il programma S, eseguito con input un programma P, dopo aver utilizzato il programma F:
- non termina se P eseguito con input P termina;
- termina se P eseguito con input P non termina.

Ora proviamo a dare in input al programma S il programma S stesso, cioè eseguiamo S dandogli un input P coincidente con S stesso.

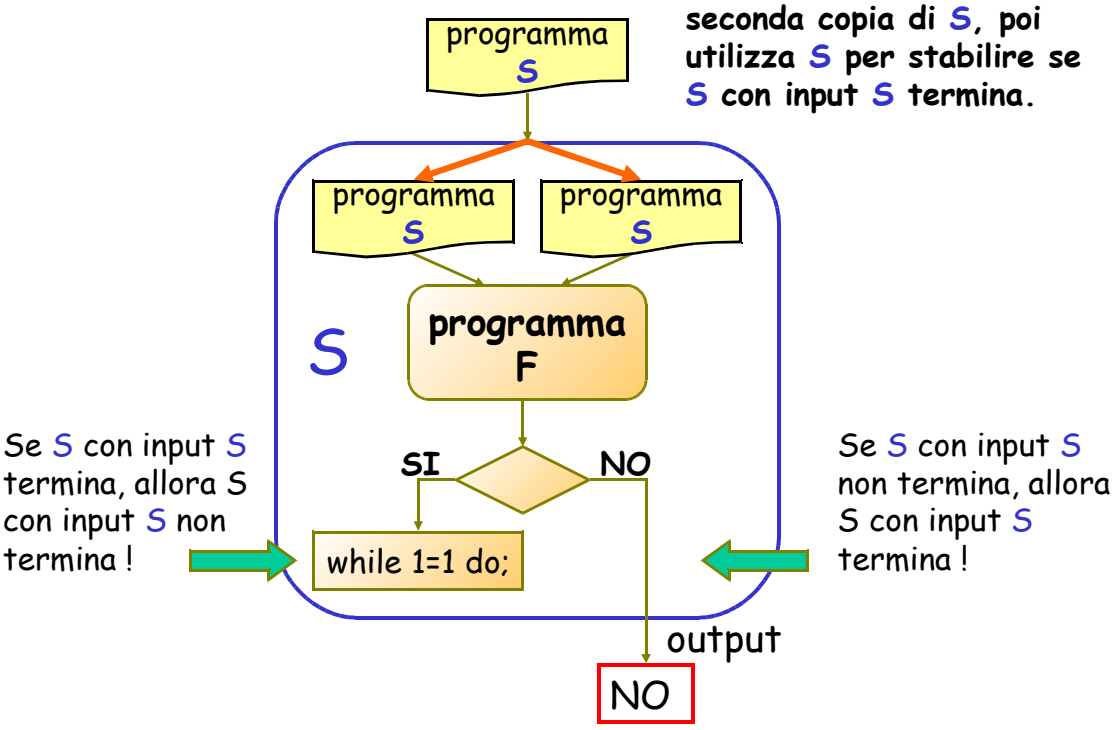
Come si comporta S con input S ? Basta evidentemente sostituire P con S nella descrizione precedente in blu.
Il programma S eseguito con input S :
- non termina se S eseguito con input S termina;
- termina se S eseguito con input S non termina.

Cioè: S con input S non termina se termina, e termina se non termina ! CONTRADDIZIONE !

Il programma F non può esistere !

graficamente:

Il programma S crea una seconda copia di S, poi utilizza S per stabilire se S con input S termina.



Se S con input S termina, allora S con input S non termina !

Se S con input S non termina, allora S con input S termina !

Dimostrazione con notazione pseudo-Pascal

Assumiamo che si sia riusciti a definire una funzione

```
function stops(prog: File, input: File)
```

la quale, preso in input un file costituente un qualunque programma o sottoprogramma Pascal e un file contenente un input legale per tale procedura, esamini il programma e restituisca true o false a seconda che il programma eseguito con quell'input termini oppure no.

Facciamo vedere che ciò conduce ad una contraddizione.

Grazie ad essa possiamo definire la funzione:

```
function selfStops(prog: File): boolean;
begin
  result:= stops(prog, prog)
end;
```

la quale, preso in input un programma, restituisce *true* se il programma eseguito con input se stesso termina (magari con un errore), *false* altrimenti.

Utilizzando *selfstops*, si può allora definire la procedura:

```
procedure strange(prog: File);
begin
  if selfStops(prog) then while true do
  else writeln('ho finito');
end;
```

la quale, preso in input un programma *prog*, non termina se *prog* applicato a *prog* termina, e invece termina se *prog* applicato a *prog* non termina.

Dunque ripetiamo:

La procedura strange, preso in input un programma *prog*, non termina se *prog* applicato a *prog* termina, e viceversa termina se *prog* applicato a *prog* non termina.

Allora, se applichiamo il programma *strange* ad un file che contiene se stesso, sostituendo *prog* con *strange* nella frase precedente, otteniamo:

La procedura strange, presa in input la procedura *strange*, non termina se *strange* applicata a *strange* termina, e termina se *strange* applicata a *strange* non termina.

Cioè non termina se termina, e termina se non termina !

Contraddizione ! La procedura stops non può esistere !

Incompletezza.

Assumiamo che si possa stabilire un insieme *computabile* di assiomi di una teoria dei programmi che sia *completo*, cioè che permetta di dedurre, per qualunque proposizione della teoria, o la proposizione stessa o la sua negata, e che sia *consistente* (o *coerente*), cioè non permetta di dedurre una contraddizione (come una proposizione e la sua negata).

Allora si può scrivere un programma F il quale, dato un programma P e un input X, generi uno dopo l'altro tutti gl'infiniti teoremi della teoria, ad esempio in ordine crescente rispetto alla lunghezza della dimostrazione.

Poiché il sistema è completo, prima o poi F genererà o il teorema "P con input X termina", o il suo negato "P con input X non termina": ma così il programma F risolve il problema della fermata, il che abbiamo dimostrato essere impossibile !

Gödel come conseguenza di Turing.

Un insieme computabile (i logici dicono *effettivo*) di assiomi consistente e completo per una teoria generale dei programmi non può esistere !

Ogni insieme di assiomi di una teoria dei programmi che sia effettivo e consistente non può essere completo: vi sarà sempre qualche proposizione tale che né essa né la sua negata sono derivabili dagli assiomi: cioè vi sarà sempre qualche proposizione *indecidibile*.

È una forma del celebre primo teorema di Gödel, che in realtà riguarda qualunque insieme (effettivo) di assiomi (per un linguaggio aritmetico) da cui si possano derivare le usuali proprietà dell'addizione e della moltiplicazione.

Per questi argomenti vedi testi di Logica, ad esempio:
Gabriele Lolli, Incompletezza, ed. Il Mulino

Correttezza di programmi

- "Il debugging può mostrare la presenza di errori in un programma, mai la loro assenza" (E.W. Dijkstra).
- Esigenza di programmi "dimostrati" corretti, soprattutto in applicazioni critiche (centrali nucleari, sistemi di volo, ecc. ecc.)
- È possibile definire una "teoria dei programmi" e fare in essa dimostrazioni di correttezza.
- Ma la dimostrazione formale di correttezza anche di un programma molto semplice è incredibilmente complicata: impensabile dimostrare manualmente la correttezza di un programma reale !
- Si può però realizzare un programma per la dimostrazione automatica, e la dimostrazione farla con l'aiuto di esso. Chi controlla che il programma dimostratore sia corretto ?

Dimostrazione automatica.

- La macchina di Chicago per i teoremi non è stata ancora realizzata: alcuni pensano che lo sarà, altri che non sia possibile.
- Esistono diversi e molto sofisticati "proof assistant" che permettono di costruire in modo interattivo dimostrazioni formali, e di estrarre programmi da dimostrazioni costruttive.
- Alcuni importanti risultati recenti di matematica pura sono stati ottenuti con l'ausilio essenziale del calcolatore: il più celebre è la dimostrazione del teorema dei quattro colori.
- Spesso, una dimostrazione condotta con l'ausilio del calcolatore non può essere controllata a mano, proprio come un calcolo complicato effettuato al calcolatore: tempo irragionevole, probabilità di errori umani altissima. Appunto per ciò si usa il computer !

Correttezza di programmi e logica.

- Semplificando molto, si può dire che la specifica di un problema algoritmico è una proposizione della forma:

  $$\forall X.\ \exists Y.\ R(X, Y)$$

  che si può leggere come:

  per ogni (input) X esiste un (output) Y che è in relazione R con X, dove X e Y possono essere di tipo non numerico. Esempio:

  Per ogni sequenza X di interi esiste una permutazione Y di X che è ordinata in ordine crescente.
- Dimostrare il teorema *costruttivamente* (cioè nella logica *intuizionista*) equivale grosso modo a trovare una funzione calcolabile f(X) tale che valga R(X, f(X)).
- Dalla dimostrazione della realizzabilità della specifica si può estrarre un algoritmo che soddisfa alla specifica: nell'esempio precedente, un algoritmo di ordinamento.

Informatica teorica, logica, teorie di tipi

- Nei linguaggi tipati le espressioni che compaiono in un programma hanno ciascuna un tipo, e il compilatore o interprete controlla che in un programma non ci siano *errori di tipo* : simile al controllo dimensionale in fisica, ciò può far scoprire un discreto numero di errori. Esempio:

```
var condiz: boolean; m, n: integer;
... condiz:= ((m + n) > 0) and  (m <= 10);
```

  m+n è un'espressione di tipo intero perché *m* ed *n* lo sono, allora *(m+n) > 0* è un'espressione di tipo booleano, ecc.
- Si stanno studiando teorie di tipi più sofisticate, che permettano di esprimere proprietà di correttezza non banali.
- Vi è una ben precisa analogia (detta di *Curry-Howard*) fra proposizioni in senso logico e tipi.

Conclusioni

- L'informatica è una scienza e una tecnica che comprende aree molto diverse fra diloro.
- Principi di base di natura logico-matematica, attività di ricerca in informatica teorica simile a quella in matematica ma con concetti, metodi e problemi suoi propri.
- Intelligenza artificiale, comprensione e trattamento del linguaggio naturale, multimedialità, giochi elettronici, sistemi informativi e basi di dati, servizi web, sicurezza informatica, biometria, telecomunicazioni e reti, mobile computing, reti neurali, bioinformatica, ...
- Domani: DNA-computing, quantum computing, computer indossabili, androidi ... ?