Ragionamento automatico

La logica matematica per l'intelligenza artificiale e la verifica del software

Maurizio Proietti, IASI-CNR, Roma

Sommario

- Il programma di Leibniz: *calculemus*!
- Le “Leggi del pensiero”: la logica proposizionale
  - Semantica: Tabelle di verità
  - Rompicapo logici
  - SAT solvers
  - Limitazioni di complessità
- Logica del prim’ordine
  - Sintassi, semantica, dimostrazioni
  - Automated Theorem Provers
  - Limitazioni di decidibilità
- AI planning: Problemi di raggiungibilità
- Verifica del software

Il programma di Leibniz: *calculemus*!

Calculemus!

Gottfried Wilhelm von Leibniz (1646-1716)

Leibniz propone di ideare “una caratteristica della ragione, mediante la quale le verità, in qualsiasi dominio, si presenterebbero alla ragione in virtù di un metodo di calcolo come nell’aritmetica e nell’algebra, purchè essa si sottoponga al corso della deduzione.

Di conseguenza, quando sorgeranno controversie fra due filosofi, non sarà più necessaria una discussione, come [non lo è] fra due calcolatori. Sarà sufficiente, infatti, che essi prendano in mano le penne, si siedano di fronte agli abachi e […] si dicano l’un l’altro: Calculemus!”

Il programma di Leibniz

- Ideare un linguaggio simbolico (*caratteristica*) della ragione
- Formalizzare il ragionamento come un calcolo (*calculus ratiocinator*)
- Il calculus ratiocinator si applica in tutti i domini (ragionatore universale)
- Sviluppare macchine calcolatrici per il ragionamento

Alla realizzazione del programma di Leibniz si è lavorato nei successivi 300 anni!

Le “leggi del pensiero”: la logica proposizionale

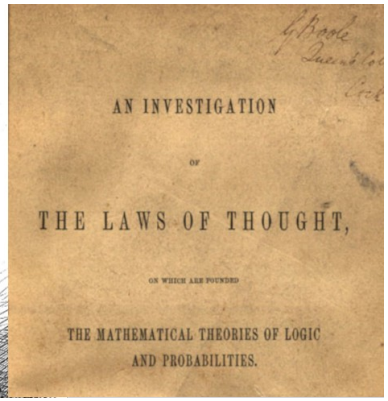
In principio era il Logos

- La logica è la scienza che ha per oggetto la formalizzazione delle leggi del pensiero

- Logos = ragione, discorso, pensiero, calcolo
  (traduzione latina: ratio, verbum)

- C'è un legame intimo tra logica, linguaggio e pensiero.
  "Pensare è parlare con se stessi" (Platone, Teetèto)

Le leggi del pensiero



George Boole (1854)

Formalizzazione matematica del ragionamento simbolico per mezzo di equazioni (Algebra Booleana)

Logica proposizionale

Sintassi: le frasi ben formate (la caratteristica)

- Variabili proposizionali: A, B, C, ...
- Congiunzione (AND): $A \wedge B$ A e B
- Disgiunzione (OR): $A \vee B$ A o B (inclusivo)
- Negazione (NOT): $\neg A$ non A
- Implicazione (IF-THEN): $A \rightarrow B$ se A allora B
- Equivalenza (IFF): $A \leftrightarrow B$ A se e solo se B

Quid est veritas?

(Ponzio Pilato)

Semantica: la verità o falsità di una formula viene definita da tabelle di verità:

A	¬ A
1	0
0	1

A	B	A ∧ B	A ∨ B	A → B	A ↔ B
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1

Modello: assegnazione di valori di verità (0 o 1) alle variabili proposizionali che rende vera la formula.

Formula soddisfacibile se ha (almeno) un modello.

Tautologie e Contraddizioni

- Tautologia: formula vera per ogni assegnazione di valori di verità
  (cioè tutte le assegnazioni di valori di verità sono modelli)

A	$A \vee \neg A$
1	1
0	1

Tertium non datur

A	B	$(A \wedge \neg A) \rightarrow B$
1	1	1
1	0	1
0	1	1
0	0	1

Ex falso quodlibet

- Contraddizione: formula falsa per ogni assegnazione di valori di verità
  (cioè non ha modelli)

A	$A \wedge \neg A$
1	0
0	0

Spot the tautology!

1. $A \rightarrow (B \rightarrow A)$
2. $A \wedge (A \rightarrow \neg A)$
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$
4. $A \rightarrow (\neg A \rightarrow B)$
5. $A \rightarrow \neg A$

Spot the tautology!

1. $A \rightarrow (B \rightarrow A)$ tautologia
2. $A \wedge (A \rightarrow \neg A)$
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$
4. $A \rightarrow (\neg A \rightarrow B)$
5. $A \rightarrow \neg A$

Spot the tautology!

1. $A \rightarrow (B \rightarrow A)$ tautologia
2. $A \wedge (A \rightarrow \neg A)$ contraddizione
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$
4. $A \rightarrow (\neg A \rightarrow B)$
5. $A \rightarrow \neg A$

Spot the tautology!

1. $A \rightarrow (B \rightarrow A)$ tautologia
2. $A \wedge (A \rightarrow \neg A)$ contraddizione
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$ tautologia
4. $A \rightarrow (\neg A \rightarrow B)$
5. $A \rightarrow \neg A$

Spot the tautology!

1. $A \rightarrow (B \rightarrow A)$ tautologia
2. $A \wedge (A \rightarrow \neg A)$ contraddizione
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$ tautologia
4. $A \rightarrow (\neg A \rightarrow B)$ tautologia
5. $A \rightarrow \neg A$

Spot the tautology!

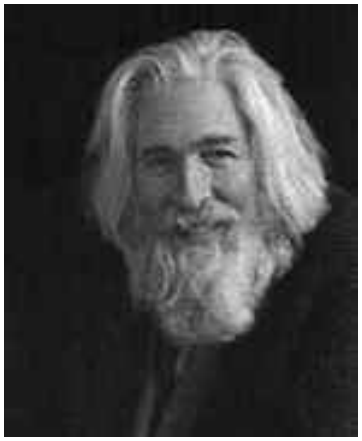
1. $A \rightarrow (B \rightarrow A)$ — tautologia
2. $A \wedge (A \rightarrow \neg A)$ — contraddizione
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$ — tautologia
4. $A \rightarrow (\neg A \rightarrow B)$ — tautologia
5. $A \rightarrow \neg A$ — né tautologia né contraddizione

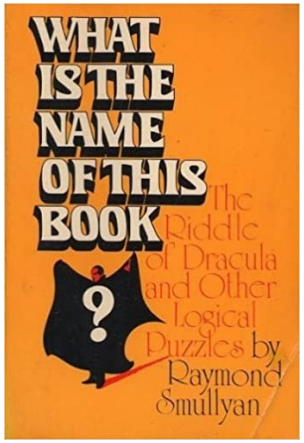
Calculemus, con la logica proposizionale

1. Rappresentare la conoscenza: descrivere una situazione mediante una formula logica (base di conoscenza) K
2. Interrogare la base di conoscenza: la formula F è una conseguenza logica di K?

$K \models F$ se ogni modello di K è un modello di F

L’Isola dei Cavalieri e dei Furfanti





Raymond Smullyan, 1978
What Is the Name of This Book?

Rompicapo logici basati sul paradosso del mentitore.
Epimenide, 600 A.C.

Su un'isola di fantasia ogni abitante è di uno dei due tipi:

- Cavaliere: dice sempre la verità, oppure
- Furfante: mente sempre.

Cavalieri e furfanti

Un visitatore chiede ad A di che tipo sia, ma non riesce a sentire la sua risposta.

B: “A ha detto di essere un furfante”

C: “Non credere a B: sta mentendo!”

Chi è cavaliere? Chi è furfante?

Cavalieri e furfanti: una controversia tra filosofi

“A ha detto di essere un furfante”

Filosofo 1:

Se A è un cavaliere dice la verità
e se ha detto di essere un furfante sta mentendo,
quindi è un furfante!

Filosofo 2:

Se A è un furfante
e ha detto di essere un furfante sta dicendo la verità,
quindi è un cavaliere!

???

Cavalieri e furfanti: rappresentazione della conoscenza

B: “A ha detto di essere un furfante”

C: “Non credere a B: sta mentendo!”

“**X è un cavaliere**” si traduce in: **X**

“**X è un furfante**” si traduce in: ¬ **X**

“**X dice Frase**” si traduce in: **X** ↔ **Frase**

“X dice Frase”: X è un cavaliere (X è vero) se e soltanto se Frase è vera

p.es. “A dice di essere un furfante”, cioè,

“A dice che A è un furfante” si traduce A ↔ ¬ A

Base di conoscenza **K**: (B ↔ (A ↔ ¬ A)) ∧ (C ↔ ¬ B)

Cavalieri e furfanti: tabella di verità

			K
A	B	C	(B ↔ (A ↔ ¬ A)) ∧ (C ↔ ¬ B)
1	1	1	0
1	1	0	0
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	0

Cavalieri e furfanti: soluzione

K

A	B	C	(B ↔ (A ↔ ¬ A)) ∧ (C ↔ ¬ B)
1	1	1	0
1	1	0	0
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	0

K ⊨ ¬ B “B è un furfante”

K ⊨ C “C è un cavaliere”

Nulla si può concludere su A

Cavalieri e furfanti: spiegazione informale

B: “A ha detto di essere un furfante”

C: “Non credere a B: sta mentendo!”

La frase “A ha detto di essere un furfante” è falsa

($A \leftrightarrow \neg A$ è una contraddizione)

Quindi B dice il falso (cioè B è un furfante)

e C dice il vero (cioè C è un cavaliere)

Ragionamento automatico in logica proposizionale con CLP(B) (1)

- Prolog (programmazione logica): un sistema di programmazione basato sulla logica.

- Interfaccia web:

  https://swish.swi-prolog.org/

- Implementa un modulo per la logica proposizionale:

:- use_module(library(clpb)).

Ragionamento automatico in logica proposizionale con CLP(B) (2)

		Sintassi CLP(B)
• Variabili proposizionali:	A, B, C, …	A, B, C, ...
• Congiunzione (AND):	A ∧ B	A * B
• Disgiunzione (OR):	A ∨ B	A + B
• Negazione (NOT):	¬ A	~A
• Implicazione (IF-THEN):	A → B	=<
• Equivalenza (IFF):	A ↔ B	=:=
• Vero:		1
• Falso:		0

Ragionamento automatico in logica proposizionale con CLP(B) (3)

CLP(B) è un sistema che consente interrogazioni (query):

?- sat(F).

Per quali valori delle variabili la formula F è vera?

Risponde:

- calcolando i modelli di F, se F è soddisfacibile
- **false**, altrimenti.

Ragionamento automatico in logica proposizionale con CLP(B) (4)

1. $A \rightarrow (B \rightarrow A)$	A =< (B =< A)
2. $A \wedge (A \rightarrow \neg A)$	A * (A =< ~A)
3. $(A \rightarrow \neg A) \vee (\neg A \rightarrow A)$	(A =< ~A) + (~A =< A)
4. $A \rightarrow (\neg A \rightarrow B)$	A =< (~ A =< B)
5. $A \rightarrow \neg A$	A =< ~A
6. $A \vee \neg A$	A + ~A
7. $(A \wedge \neg A) \rightarrow B$	(A * ~A) =< B
8. $A \wedge \neg A$	A * ~A

Running CLP(B)

Query: ?- sat(A =< (B =< A)).

Answer: sat(A=:=A),
sat(B=:=B)

TAUTOLOGIA: Qualunque valore di A e B rende vera la formula.

Query: ?- sat(A * (A =< ~A)).

Answer: false

CONTRADDIZIONE: nessun valore di A rende vera la formula

Query: ?- sat(A =< ~A).

Answer: A=0

SODDISFACIBILE (vera se A è falsa) ma non TAUTOLOGIA

Risolvere rompicapo logici con CLP(B)

B: “A ha detto di essere un furfante”
C: “Non credere a B: sta mentendo!”

$(B \leftrightarrow (A \leftrightarrow \neg A)) \wedge (C \leftrightarrow \neg B)$

Query:
```
?- sat((B =:= (A =:= ~A)) * (C =:= ~B)).
```

Answer:
```
B = 0,                  % B è un furfante
C = 1,                  % C è un cavaliere
sat(A=:=A)              % A indeterminato
```

Decidibilità

- La logica proposizionale è decidibile: la risposta a tutte le interrogazioni
  - F è soddisfacibile?
  - $K \models F$?

  si può calcolare in tempo finito

- Un algoritmo di decisione "esaustivo": costruire le tabelle di verità di K ed F

- Complessità di calcolo: la tabella di verità di una formula con N variabili ha 2^N righe

Quanto tempo ci vuole?

N	Righe	Tempo di calcolo
10	~ 10^{3}	~ 10^{-8} sec
20	~ 10^{6}	~ 10^{-5} sec
30	~ 10^{9}	~ 10^{-2} sec
40	~ 10^{12}	~ 10 sec
50	~ 10^{15}	~ 3 h
60	~ 10^{18}	~ 133 gg
70	~ 10^{21}	~ 374 anni
80	~ 10^{24}	~ 383.000 anni
90	~ 10^{27}	~ 392 milioni di anni
96	~ $8x10^{28}$	25 miliardi di anni (maggiore del tempo trascorso dal Big Bang)

Tempi approssimati per un computer che calcola 100 miliardi di righe/sec (= 100 GHz)

Una formula con 96 variabili

sat(~A1=:=(~A2 * ~A3 * ~A4) + ~(~A5 + ~A6 * ~A7) * ~(~A8 + ~(~A5 + ~A9 * ~A51) * ~(~A5 + ~A10 * A11=<(~A12=:=(~A2 * ~A13 * ~A14) + ~(~A15 + ~A16 * ~A7) * ~(~A18 + ~(~A5 + ~A9 * ~A17) * ~(~A5 + ~A20 * ~(~A5 + ~A19 * A21=<(~A22=:=(~A2 * ~A23 * ~A24) + ~(~A25 + ~A26 * ~A52) * ~(~A28 + ~(~A5 + ~A29 * ~A27) * ~(~A5 + ~A20 * ~(~A31=:=(~A2 * ~A30 * ~A34) + ~(~A5 + ~A6 * ~A7) * ~(~A8 + ~(~A5 + ~A9 * ~A7) * ~(~A5 + ~A10 * A11=<(~A32=:=(~A2 * ~A33 * ~A14) + ~(~A15 + ~A16 * ~A7) * ~(~A18 + ~(~A5 + ~A9 * ~A35) * ~(~A5 + ~A20 * ~(~A5 + ~A19 * A21=<(~A22=:=(~A2 * ~A23 * (~A1=:=(~A40 * ~A55 * ~A41) + ~(~A42 + ~A6 * ~A53) * ~(~A43 + ~(~A5 + ~A9 * ~A7) * ~(~A5 + ~A44 * A11=<(~A12=:=(~A2 * ~A45 * ~A14) + ~(~A15 + ~A46 * ~A54) * ~(~A18 + ~(~A47 + ~A9 * ~A48) * ~(~A5 + ~A20 * ~A49))))))) + ~(~A25 + ~A56 * ~A57) * ~(~A58 + ~(~A5 + ~A59 * ~A27) * ~(~A39 + ~A20 * ~(~A60 + ~A44 * A61=<(~A12=:=(~A62 * ~A63 * ~A64) + ~(~A65 + ~A66 * ~A54) * ~(~A18 + ~(~A67 + ~A9 * ~A68) * ~(~A69 + ~A70 * ~(~A58 + ~(~A71 + ~A59 * ~A72) * ~(~A39 + ~A73 * ~(~A60 + ~A74 * A61=<(~A12=:=(~A75 * ~A63 * ~A76) + ~(~A77 + ~A66 * ~A54) * ~(~A78 + ~(~A67 + ~A9 * ~A68) * ~(~A79 + ~A20 * ~A80)))))))))))))))))))))))))))) * (~A81=:=(~A82 * ~A83 * ~A84) + ~(~A85 + ~A86 * ~A87) * ~(~A88 + ~(~A5 + ~A89 * ~A7) * ~(~A5 + ~A90 * A91=<(~A12=:=(~A92 * ~A93 * ~A94) + ~(~A15 + ~A16 * ~A7) * ~(~A18 + ~(~A95 + ~A9 * ~A17) * ~(~A5 + ~A20 * ~A96))))))).

Si può fare meglio?

- Si può fare meglio dell'algoritmo "esaustivo"?
- Si possono usare euristiche per evitare di generare, in molti casi, l'intera tabella di verità
- Molti SAT solver, sistemi di risoluzione della soddisfacibilità, si basano su varianti dell'algoritmo di Davis-Putnam-Logemann-Loveland (generazione non-deterministica di un'assegnazione di valori di verità e backtracking). Lineare nel numero delle variabili, nel caso "fortunato", esponenziale nel caso peggiore

Una formula con 96 variabili

```
sat(~A1=:=(~A2 * ~A3 * ~A4) + ~(~A5 + ~A6 * ~A7) * ~(~A8 + ~(~A5 + ~A9 * ~A51) *
 ~(~A5 + ~A10 * A11=<(~A12=:=(~A2 * ~A13 * ~A14) + ~(~A15 + ~A16 * ~A7) *
 ~(~A18 + ~(~A5 + ~A9 * ~A17) * ~(~A5 + ~A20 * ~(~A5 + ~A19 *
 A21=<(~A22=:=(~A2 * ~A23 * ~A24) + ~(~A25 + ~A26 * ~A52) * ~(~A28 + ~(~A5 +
 ~A29 * ~A27) * ~(~A5 + ~A20 * ~(~A31=:=(~A2 * ~A30 * ~A34) + ~(~A5 + ~A6 * ~A7)
 * ~(~A8 + ~(~A5 + ~A9 * ~A7) * ~(~A5 + ~A10 * A11=<(~A32=:=(~A2 * ~A33 * ~A14)
 + ~(~A15 + ~A16 * ~A7) * ~(~A18 + ~(~A5 + ~A9 * ~A35) * ~(~A5 + ~A20 * ~(~A5 +
 ~A19 * A21=<(~A22=:=(~A2 * ~A23 * (~A1=:=(~A40 * ~A55 * ~A41) + ~(~A42 + ~A6 *
 ~A53) * ~(~A43 + ~(~A5 + ~A9 * ~A7) * ~(~A5 + ~A44 * A11=<(~A12=:=(~A2 * ~A45 *
 ~A14) + ~(~A15 + ~A46 * ~A54) * ~(~A18 + ~(~A47 + ~A9 * ~A48) * ~(~A5 + ~A20 *
 ~A49)))))) + ~(~A25 + ~A56 * ~A57) * ~(~A58 + ~(~A5 + ~A59 * ~A27) * ~(~A39 +
 ~A20 * ~(~A60 + ~A44 * A61=<(~A12=:=(~A62 * ~A63 * ~A64) + ~(~A65 + ~A66 *
 ~A54) * ~(~A18 + ~(~A67 + ~A9 * ~A68) * ~(~A69 + ~A70 * ~(~A58 + ~(~A71 + ~A59
 * ~A72) * ~(~A39 + ~A73 * ~(~A60 + ~A74 * A61=<(~A12=:=(~A75 * ~A63 * ~A76) +
 ~(~A77 + ~A66 * ~A54) * ~(~A78 + ~(~A67 + ~A9 * ~A68) * ~(~A79 + ~A20 *
 ~A80))))))))))))))))))))))))))) * (~A81=:=(~A82 * ~A83 * ~A84) + ~(~A85 + ~A86 * ~A87) *
 ~(~A88 + ~(~A5 + ~A89 * ~A7) * ~(~A5 + ~A90 * A91=<(~A12=:=(~A92 * ~A93 *
 ~A94) + ~(~A15 + ~A16 * ~A7) * ~(~A18 + ~(~A95 + ~A9 * ~A17) * ~(~A5 + ~A20 *
 ~A96))))))).
```

CLP(B) risponde in pochi secondi!

Time is Money!

- Non è stato trovato un algoritmo sostanzialmente migliore del DPLL
- L'esistenza di un algoritmo generale "efficiente" (cioè non esponenziale) per verificare la validità di formule proposizionali è un problema aperto (P=NP?)
- E disponibile un premio di $ 1,000,000 per chi lo risolve!

  Millennium Prize
  http://www.claymath.org/millennium-problems

Altri rompicapo logici

1. A: uno tra me e B e' un furfante

2. A: io sono un furfante o B e' un cavaliere

3. A: Io sono un furfante, ma B non lo è

4. A: B e' un furfante
   B: A e C sono dello stesso tipo

5. A: Io e B siamo entrambi furfanti

6. A: parla ma non si capisce
   B: c'e' esattamente un cavaliere tra noi
   C: B e' un furfante

Soluzioni

1. A: “uno tra me e B e' un furfante”
   Query: ?- sat(A=:=(~A + ~B)).
   Answer: A = 1, B = 0

2. A: “io sono un furfante o B e' un cavaliere”
   Query: sat(A=:=(~A + B)).
   Answer: A = B, B = 1

3. A: “io sono un furfante, ma B non lo è”
   Query: ?- sat((A =:= (~A * B))).
   Answer: A=B, B=0

Soluzioni

4. A: “B e' un furfante”
   B: “A e C sono dello stesso tipo”
   Query: ?- sat((A =:= ~B) * (B=:=(A=:=C))).
   Answer: C = 0, sat(A=\=B)

5. A: “Io e B siamo entrambi furfanti”
   Query: ?-sat(A =:= (~A * ~B)).
   Answer: A = 0, B = 1

6. A: parla ma non si capisce
   B: “A dice che c'e' esattamente un cavaliere tra noi”
   C: “B e' un furfante”
   Query: ?- sat((B =:= (A =:= (A * ~(B+C)) +
   (B * ~(A+C)) + (C * ~(A+B)))) * (C =:= ~B)).
   Answer: B = 0, C = 1, sat(A=:=A)

Logica dei predicati del prim’ordine

Individui, predicati e quantificatori

“Socrate è un essere umano”

“Ogni essere umano è mortale”

“Ogni essere umano ha una madre”

“La madre di un essere umano è un essere umano”

“Fenarete è la madre di Socrate”

Logica dei predicati del prim'ordine

Sintassi: le frasi ben formate (la caratteristica)

- Formule atomiche:

  p(X) l'individuo X ha la proprietà p

- Congiunzione (∧), disgiunzione (∨), negazione (¬), implicazione (→), equivalenza (↔)

- Quantificazione universale: ∀X F per ogni X, F

- Quantificazione esistenziale: ∃X F esiste X, F

Formalizzazione in logica dei predicati del prim'ordine

- "Socrate è un essere umano":

  umano(socrate)
- "ogni essere umano è mortale":

  $\forall X$ (umano(X) $\rightarrow$ mortale(X))
- "Ogni essere umano ha una madre":

  $\forall X$ (umano(X) $\rightarrow \exists Y$ madre-di(X,Y))
- "La madre di un essere umano è un essere umano":

  $\forall X \forall Y$ (madre-di(X,Y) $\wedge$ umano(X) $\rightarrow$ umano(Y))
- "Fenarete è la madre di Socrate":

  madre-di(socrate,fenarete)

Dimostrazioni

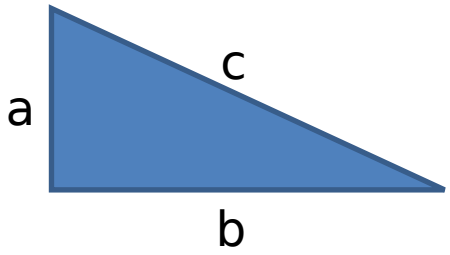
- Le tabelle di verità non sono adeguate in pratica per determinare la verità di formule con quantificatori: il valore di verità di ∀X F e ∃X F dipende dall'insieme (possibilmente infinito) di valori che può assumere X

- Un metodo alternativo è basato sulla nozione di dimostrazione [Grecia, VI sec. A.C.]

- Dimostrazione**:** *Argomentazione attraverso la quale si stabilisce che una certa nozione o tesi o teoria è vera* [Enc. Treccani]

Verso l'infinito e oltre!

- Come dimostrare proprietà universali su domini infiniti?
- Teorema di Pitagora [500 a.C. circa]:
  in ogni triangolo rettangolo il quadrato costruito sull'ipotenusa è equivalente alla somma dei quadrati costruiti sui cateti

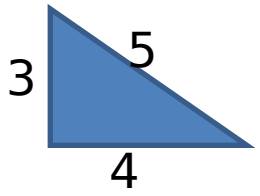


$a^2 + b^2 = c^2$

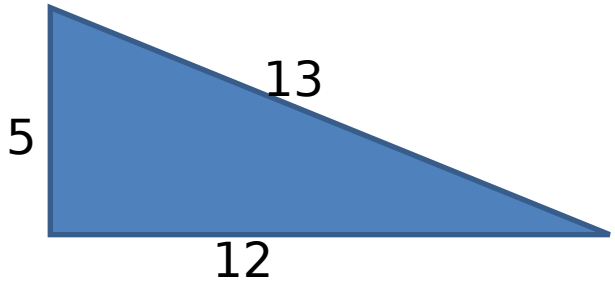
- $\forall T$ (triangoloRettangolo(T) $\rightarrow$
  (cateto1(T)^2 + cateto2(T)^2) = ipotenusa(T)^2)

Il metodo “babilonese”

- $5^2 = 3^2 + 4^2$



- $13^2 = 5^2 + 12^2$



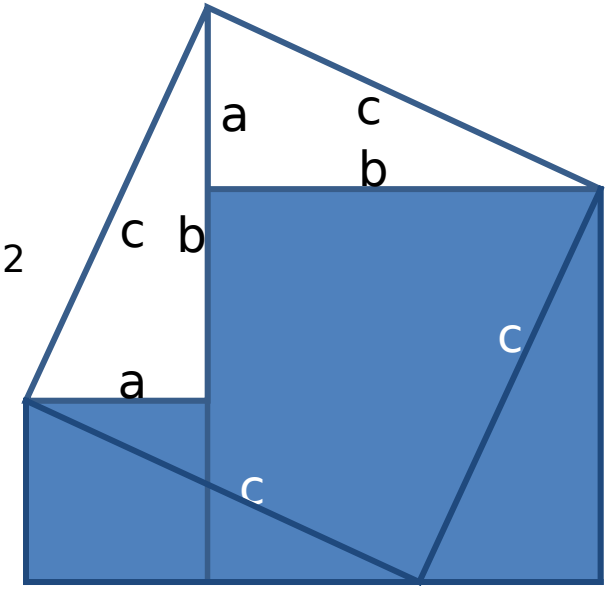
. . .

- Ci sono infiniti triangoli rettangoli: una enumerazione esaustiva è impossibile!

Il metodo “greco”: dimostrazione costruttiva

Dimostrazione del Teorema di Pitagora: una procedura per costruire, in un triangolo rettangolo generico, a^2 e b^2 da c^2

$$a^2 + b^2 = c^2$$



Il metodo assiomatico

- La dimostrazione del teorema di Pitagora è un esempio di applicazione del metodo assiomatico [Euclide,Hilbert]
- Una teoria (p.es., la geometria euclidea) si formalizza attraverso:
  - Definizioni (p.es., triangolo rettangolo)
  - Assiomi propri della teoria (p.es., i postulati)
  - Assiomi logici: identità, non-contraddizione, terzo escluso, …
  - Regole di deduzione, p.es., *modus ponens*: da p(a) e $\forall X\ (p(X) \rightarrow q(X))$ si deduce q(a)
  - Teoremi: enunciati che si dimostrano a partire dagli assiomi usando le regole di deduzione

Dimostrazione

Sequenza finita di formule ottenute dagli assiomi (Ax) della teoria applicando le regole di deduzione

Ax

Formula1

Formula2

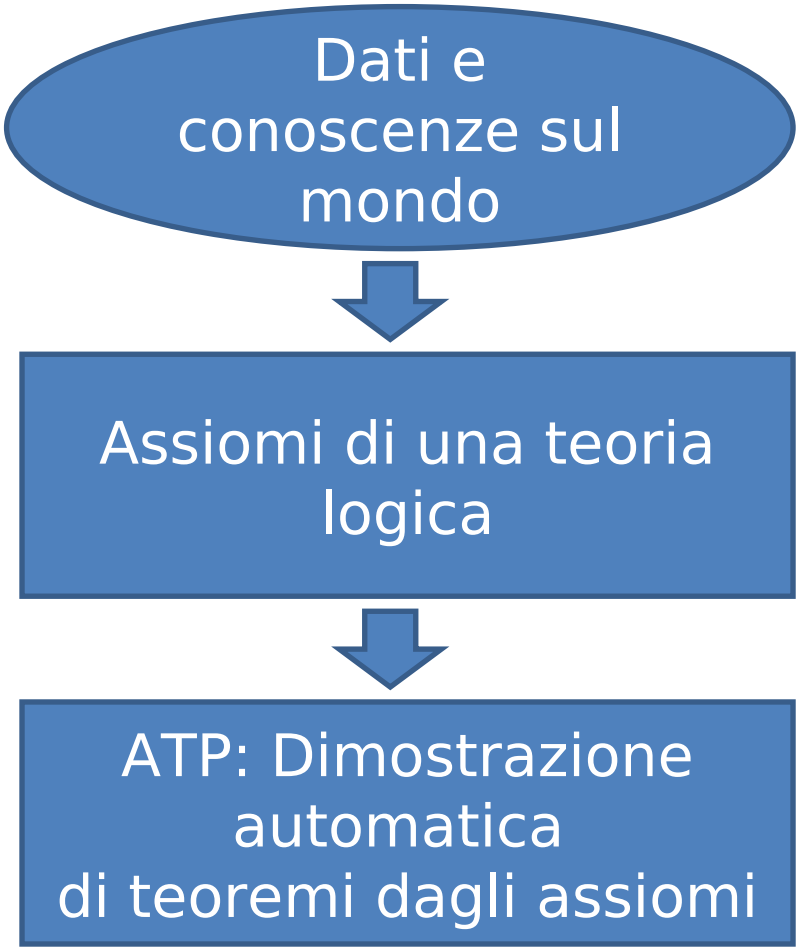
...

E

E è un teorema

Il Ragionatore Automatico Universale

Conoscenza



Rappresentazione formale della conoscenza (base di conoscenza)

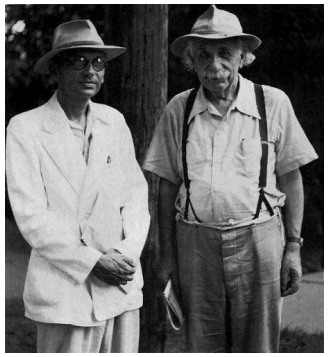
Interrogazione della base di conoscenza

Dimostrazione: Fenarete è mortale

1. Assiomi della teoria di Socrate	(Ax)
2. umano(socrate)	(Ax)
3. madre-di(socrate,fenarete)	(Ax)
4. madre-di(socrate,fenarete) ∧ umano(socrate)	(2∧3)
5. ∀X∀Y (madre-di(X,Y) ∧ umano(X) → umano(Y))	(Ax)
6. umano(fenarete)	(MP 4,5)
7. ∀X (umano(X) → mortale(X))	(Ax)
8. mortale(fenarete)	(MP 6,7)

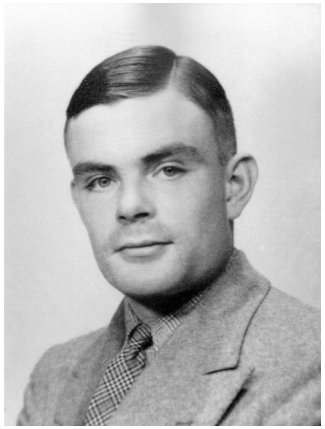
mortale(fenarete) è un teorema

Dedurre = Calcolare



K.Gödel con A.Einstein
(Princeton, 1954)

Kurt Gödel [1931]**:** la logica si può codificare nell'aritmetica e viceversa.



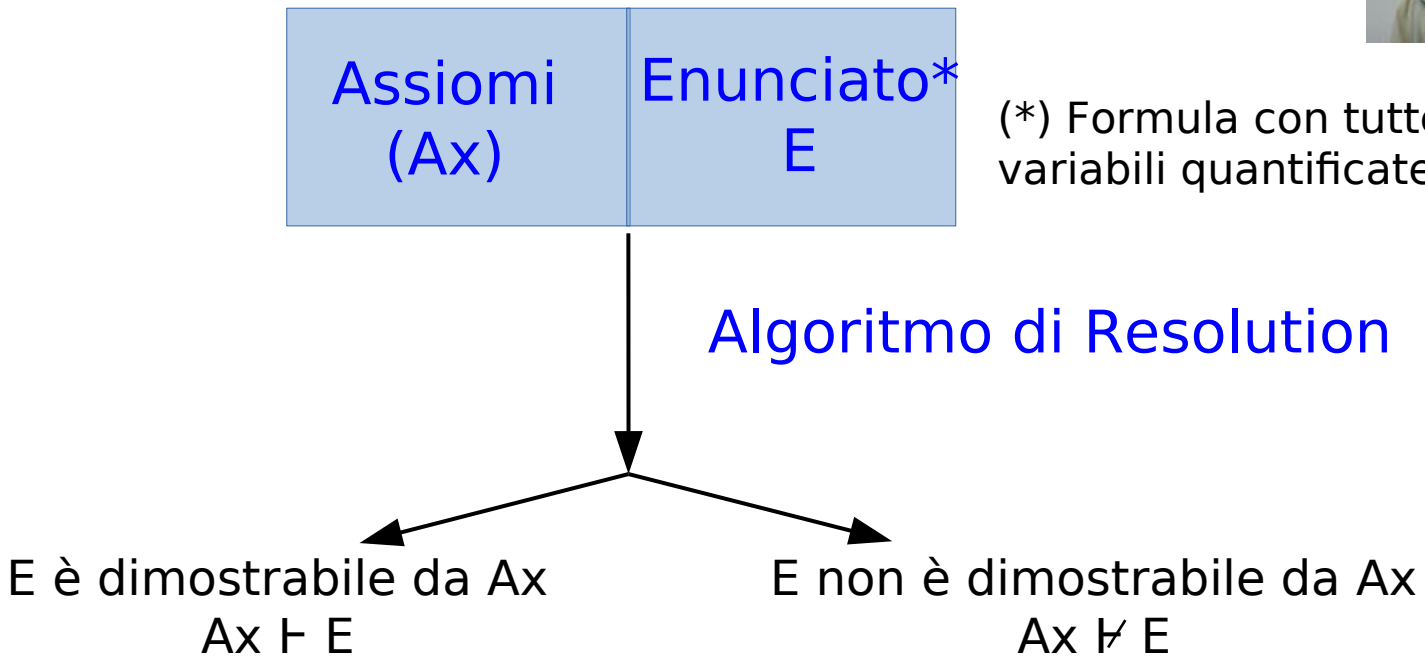
Alan Turing [1936-7] presenta un modello teorico di calcolatore e di algoritmo: la Macchina di Turing.

(Una formalizzazione equivalente, basata sul λ-calcolo è dovuta a Alonzo Church [1936])

Il problema di decidere se una formula è un teorema o no è equivalente alla terminazione di un calcolo della Macchina di Turing.

Dimostrazione Automatica dei Teoremi

Alan J. Robinson (Resolution, 1965)



Dimostrazione Automatica dei Teoremi

Alan J. Robinson (Resolution, 1965)

- Assiomi

  $\forall\Psi$ somma(0, Y, Y)

  $\forall X \forall Y \forall Z$ (somma(X,Y,Z) $\rightarrow$ somma(s(X), Y, s(Z)))

  [s(x) = x+1 “successore”]

- Enunciato da dimostrare o confutare

  $\exists Z$ somma(s(s(0)), s(s(0)), Z) [esiste *z* tale che 2+2=Z]

- Algoritmo di Resolution:

  true

Dimostrazione Automatica in Prolog

- Sintassi (Clausole di Horn):
- Atomi: $p(t_1,\ldots,t_m)$. p predicato (es. umano, madre-di),
  $t_1,\ldots,t_m$ termini (es. socrate, 0, X+1)
- Fatti: A. A atomo (es. umano(socrate)
- Regole: A :- $A_1,\ldots,A_n$. $A_1,\ldots,A_n$ atomi
- Interrogazioni: ?- $A_1,\ldots,A_n$.

- A :- $A_1,\ldots,A_n$. è un modo di scrivere
  $\forall X_1 \ldots \forall X_k\ (A_1 \wedge \ldots \wedge A_n \rightarrow A)$

- I fatti e le regole sono gli assiomi della teoria

Dimostrazione Automatica in Prolog: Socrate

- Fatti:

  umano(socrate).

  madre-di(socrate,fenarete).

- Regole:

  mortale(X) :- umano(X).

  umano(Y) :- madre-di(X,Y), umano(X).

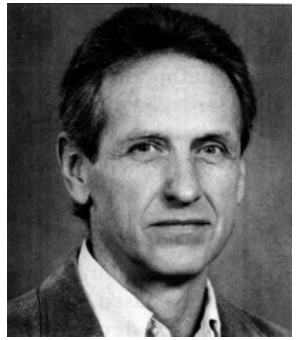
  Interrogazione:

  ?- mortale(fenarete).

  Risposta: true

Programmare in logica: Prolog

Robert A. Kowalski (1970's)



- Istruzioni del programma (Assiomi: fatti e regole)

  somma(0, Y, Y).

  somma(s(X), Y, s(Z)) :- somma(X, Y, Z).

  NB: Regola ricorsiva)

- Interrogazione (Enunciato da dimostrare o confutare)

?- somma(s(s(0)), s(s(0)), Z). [∃Z 2+2=Z]

- Risposta

  true

  Z=s(s(s(s(0)))) ["testimone" Z=4]

Dedurre = Calcolare!

Completezza e Indecidibilità

- Gödel, Turing, Church, Robinson (e altri) hanno stabilito la potenza e i limiti della dimostrazione automatica dei teoremi.

- Completezza. Esiste un algoritmo R che dato comunque un enunciato E:
  - se E è un teorema allora R termina il calcolo e fornisce la risposta "sì" (true). (In particolare: R è l'algoritmo di Resolution)

- Indecidibilità. Non esiste alcun algoritmo di decisione D che data comunque un enunciato E:
  - D termina il calcolo in tempo finito e
  - D fornisce la risposta "sì" se e soltanto se E è un teorema.

Si dice anche che Resolution è un semi-algoritmo (può non terminare).

Conseguenze dell'indecidibilità

- Il Ragionatore Universale Automatico con garanzia di risposta in tempo finito per ogni domanda non esiste.
- Molti task specifici di ragionamento possono comunque essere risolti in modo automatico (e.g., teorie decidibili).
- Il sistema di dimostrazione automatica di teoremi Vampire, vincitore della competizione 2012 tra ATP, ha risolto 429 su 450 problemi proposti.
- Prolog è utilizzato in molte applicazioni del Ragionamento Automatico.

FAQs sull'indecidibilità

- L'indecidibilità dipende dalle risorse (memoria, velocità di calcolo, etc.) del computer?

  No, si assume una nozione di calcolatore (Macchina di Turing) con memoria infinita e tempo a disposizione illimitato.

- L'indecidibilità implica che il computer è meno potente della mente umana?

  No, il limite dell'indecidibilità non è superabile neppure dalla mente umana (il concetto di algoritmo è astratto).

Chi trova una dimostrazione trova un tesoro

- L'ultimo teorema di Fermat (1637):

  $\neg\exists x\ \exists y\ \exists z\ \exists n\ (x>0 \wedge y>0 \wedge z>0 \wedge n>2 \wedge x^n + y^n = z^n)$

  «Dispongo di una meravigliosa dimostrazione di questo teorema che non può essere contenuta nel margine troppo stretto della pagina»

- Il teorema è stato dimostrato nel 1994 da Andrew Wiles (ricevendo un premio di $ 50,000!)
  La dimostrazione è complicata e lunga 130 pagine!

- Questione aperta: generare una dimostrazione del teorema di Fermat usando un Dimostratore (semi-)Automatico di Teoremi (ATP)

A che serve il ragionamento automatico?

Applicazioni

- Intelligenza Artificiale: Sistemi intelligenti (in medicina, economia, legge, etc.)
- AI planning, Robotica
- Verifica automatica dell'hardware e del software
- Semantic Web
- Comprensione del linguaggio naturale
- Apprendimento automatico
- Sicurezza informatica
- E molte altre!

Salvare capra e cavoli

Problema (Alcuino di York: *Propositiones ad acuendo juvenes*, IX secolo):

Un contadino deve trasportare un lupo, una capra e un cavolo sulla riva opposta di un fiume usando una barca.

- la barca può trasportare (oltre al contadino) soltanto uno tra il lupo, la capra e il cavolo;
- senza la presenza del contadino, il lupo mangia la capra e la capra mangia il cavolo.

AI Planning

- Rappresentazione della conoscenza in termini di stati e azioni.
- Initial state:

  initial(state(man(left),goat(left),wolf(left),cabbage(left))).
- Safe states (goat cannot eat cabbage, wolf cannot eat goat):

  safe(state(man(W),goat(X),wolf(_Y),cabbage(_Z))) :- W=X.

  safe(state(man(W),goat(X),wolf(Y),cabbage(Z))) :-
  dif(W,X), dif(X,Y), dif(X,Z).
- Goal state (everything on the right side):

  ?- reach(state(man(right),goat(right),wolf(right),cabbage(right)), Plan).

The answer is a Plan to reach the goal.

AI Planning

- Actions (the man can carry 1 thing per crossing):

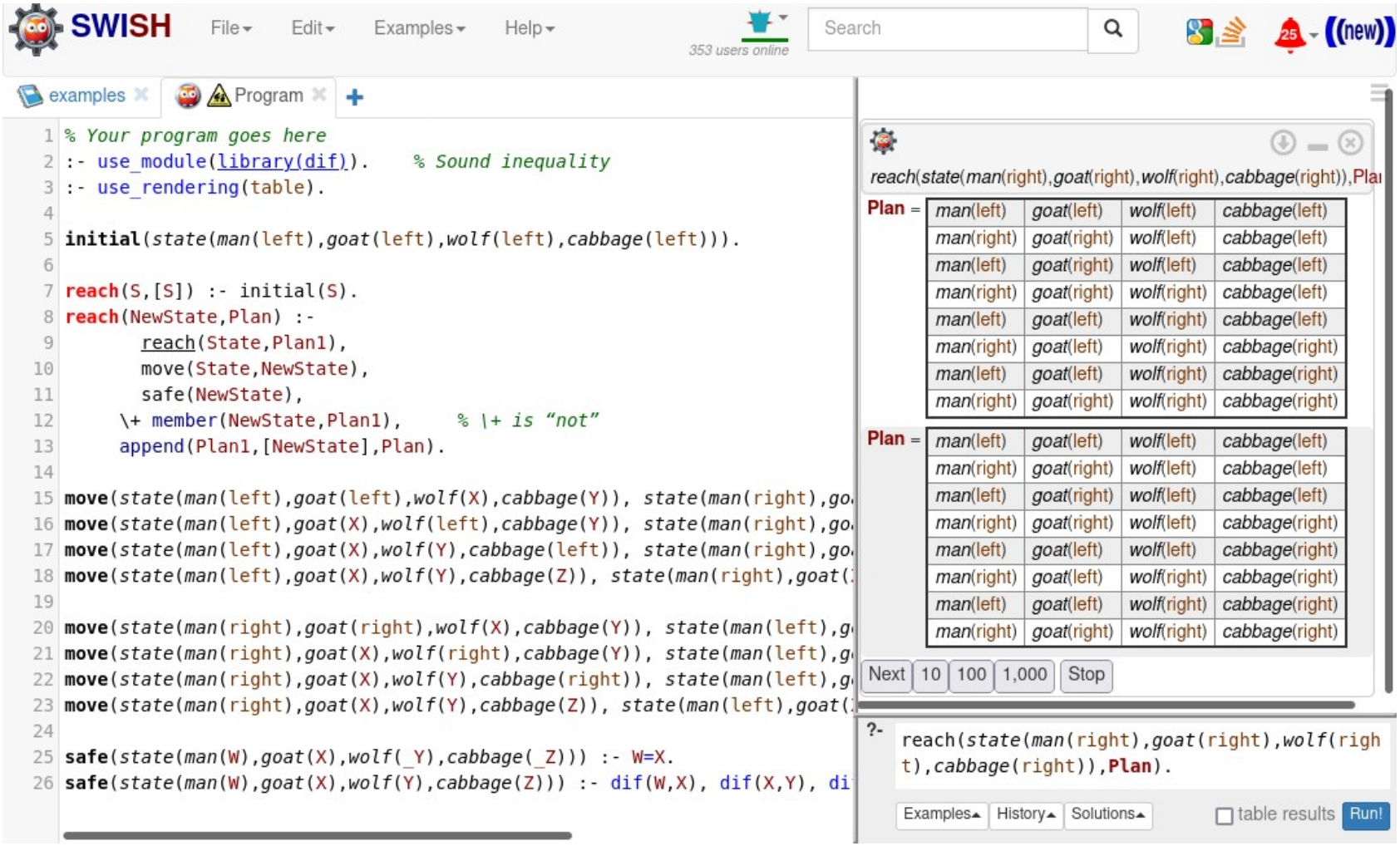
```
move(state(man(left),  goat(left),  wolf(X),cabbage(Y)),
        state(man(right),goat(right),wolf(X),cabbage(Y))).
move(state(man(right),goat(right),wolf(X),cabbage(Y)),
        state(man(left),  goat(left),   wolf(X),cabbage(Y))).
...
```

- State reachability (Recursive):

```
reach(S,[S]) :- initial(S).
reach(NewState,Plan) :-
    reach(State,Plan1),
    move(State,NewState),
    safe(NewState),
    \+ member(NewState,Plan1),      % "\+" is negation
    append(Plan1,[NewState],Plan).
```

Running the Planner

https://swish.swi-prolog.org/



I piani di attraversamento

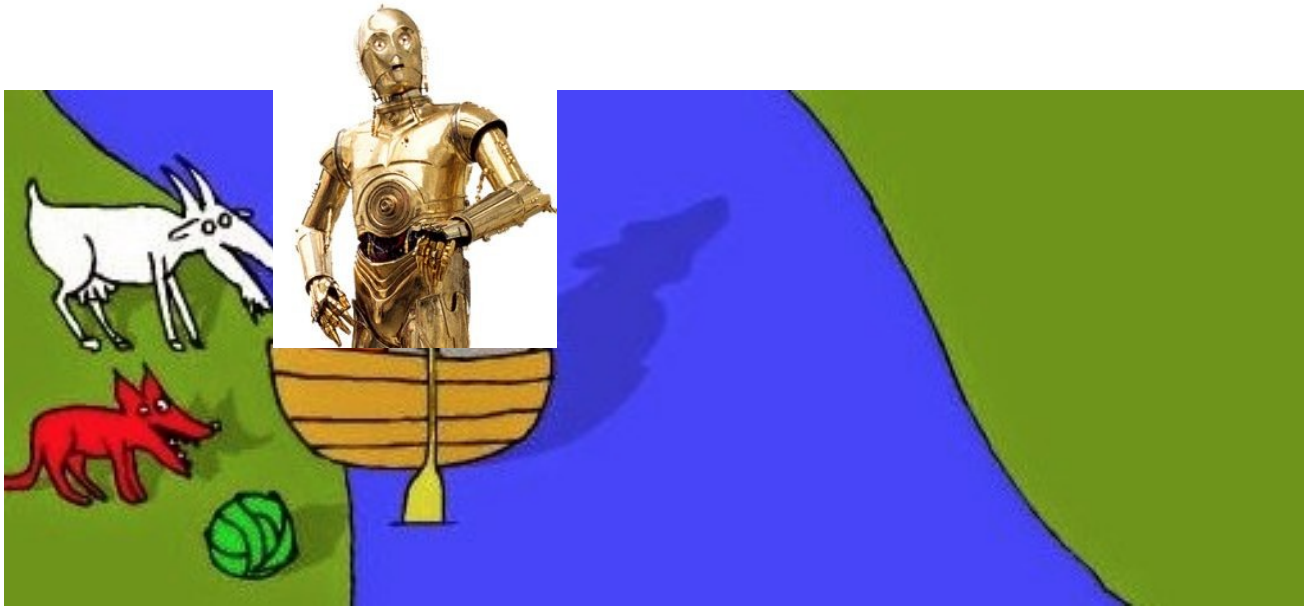
Plan =

man(left)	*goat*(left)	*wolf*(left)	*cabbage*(left)
man(right)	*goat*(right)	*wolf*(left)	*cabbage*(left)
man(left)	*goat*(right)	*wolf*(left)	*cabbage*(left)
man(right)	*goat*(right)	*wolf*(right)	*cabbage*(left)
man(left)	*goat*(left)	*wolf*(right)	*cabbage*(left)
man(right)	*goat*(left)	*wolf*(right)	*cabbage*(right)
man(left)	*goat*(left)	*wolf*(right)	*cabbage*(right)
man(right)	*goat*(right)	*wolf*(right)	*cabbage*(right)

Plan =

man(left)	*goat*(left)	*wolf*(left)	*cabbage*(left)
man(right)	*goat*(right)	*wolf*(left)	*cabbage*(left)
man(left)	*goat*(right)	*wolf*(left)	*cabbage*(left)
man(right)	*goat*(right)	*wolf*(left)	*cabbage*(right)
man(left)	*goat*(left)	*wolf*(left)	*cabbage*(right)
man(right)	*goat*(left)	*wolf*(right)	*cabbage*(right)
man(left)	*goat*(left)	*wolf*(right)	*cabbage*(right)
man(right)	*goat*(right)	*wolf*(right)	*cabbage*(right)

Automated AI Planning



I piani di attraversamento

Confrontate le soluzioni con Wikipedia:

https://it.wikipedia.org/wiki/Problema_del_lupo,_della_capra_e_dei_cavoli

Altri problemi di Planning

- Missionari e cannibali (missionaries and cannibles)
- Mariti gelosi (jealous husbands)
- Trasporto di grano (jeep/desert crossing)
- Riempimento/svuotamento di botti (water pouring puzzle)
- Il gioco Mu [Douglas Hofstadter. *Gödel, Escher, Bach - Un'eterna ghirlanda brillante*, 1979]

Verifica del software

Un fallimento del software: Ariane 5

Il primo volo dell'Ariane 5 (4 giugno 1996) fallì e il razzo si autodistrusse dopo 40 secondi dal lancio per via di un malfunzionamento del software di controllo causato da un “integer overflow”.

Testing del software

```
int x, z;
int sum_upto(int n) {
  int r = 0;
  while (n > 0) {
    r = r + n;  n = n - 1; }
  return r;
}
void main() {
  x = 100;
  z = sum_upto(x);
  assert(z>0);
  ...
}
```

Condizione di errore: z<=0.

In fase di testing, il programma termina se la condizione è vera

Verifica del software

```
int x, z;
int sum_upto(int n) {
  int r = 0;
  while (n > 0) {
    r = r + n;  n = n - 1; }
  return r;
}
void main() {
  x = N;       N>0
  z = sum_upto(x);
  assert(z>0);
  ...
}
```

Il programma è corretto se la condizione di errore z<=0 sarà falsa per ogni esecuzione con valore di input x>0

Correttezza dei programmi

```
bugged :-
    reach(St),
    error(St).
reach(St) :-
    init(St).
reach(St1) :-
    reach(St),
    exec(St,St1).
```

- stato: <comando_da_eseguire,ambiente>
- exec(St0,St1): lo stato St0 viene modificato nello stato St1 dall'esecuzione di un comando
- init(St): St è lo stato iniziale (x>0),
- error(St): St è uno stato di errore (z=<0).
- reach(St): St è raggiungibile dallo stato iniziale

Il programma è corretto se lo stato di errore non è raggiungibile, cioè la risposta all'interrogazione

```
?- bugged.
```

è 'false'

I limiti della verifica del software

Conseguenze dei risultati teorici sulla dimostrazione automatica dei teoremi:

- Completezza: Se il programma non è corretto allora la risposta ‘true’ all’interrogazione ‘?- bugged’ viene calcolata in tempo finito (ma possibilmente molto alto – si veda il caso della logica proposizionale).
- Indecidibilità. Se il programma è corretto allora la risposta ‘false’ all’interrogazione ‘?- bugged’ potrebbe non essere calcolata in tempo finito (qualunque sia il semi-algoritmo di decisione).

Ricerca attuale: sviluppare semi-algoritmi che diano una risposta in tempo finito *più spesso possibile*, considerando classi ristrette di programmi e di proprietà da verificare.

Verification via abstraction: the MU puzzle

[D. Hofstadter. *Gödel, Escher, Bach*, 1979]

From *Wikipedia*: Suppose there are the symbols **M**, **I**, and **U** which can be combined to produce strings of symbols. Start with the string **MI** and transform it into the string **MU** using in each step one of the following rewriting rules:

1. xI → xIU Add a U to the end of any string ending in I
   example: MI → MIU
2. Mx → Mxx Double the string after the M
   example: MIU → MIUIU
3. xIIIy → xUy Replace any III with a U
   example: MUIIIU → MUUU
4. xUUy → xy Remove any UU
   example: MUUU → MU

MU puzzle – Prolog encoding

NB: variabili maiuscole, M,I,U minuscole.

1. % xI → xIU
   rule(L -> R) :- append(X,[i],L), append(X,[i,u],R).
2. % Mx → Mxx
   rule(L -> R) :- L=[m|X], append([m|X],X,R).
3. % xIIIy → xUy
   rule(L -> R) :- append(X,[i,i,i|Y],L), append(X,[u|Y],R).
4. % xUUy → xy
   rule(L -> R) :- append(X,[u,u|Y],L), append(X,Y,R).

- % Rewrite X into Z by N applications of rules 1 – 4

  rewrite(X,Z,N) :- N=0, X=Z.
  rewrite(X,Z,M) :- M>=0, N=M-1, rule(X -> Y), rewrite(Y,Z,N).
- % Query: ?- rewrite([m,i],[mu],N).

MU puzzle – Querying Prolog

?- rewrite([m,i],[m,u],N). for N=1,...,9,...

false

?- rewrite([m,i],[m,u],N).

Does not terminate (out of stack)

The puzzle cannot be solved!

How to prove that the answer is “false” for all N?

MU puzzle – Counting Abstraction

n(Z): Z is the number of i's

- % xI → xIU counting abstraction: n(Z) -> n(Z)
  n(Z) :- n(Z).
- % Mx → Mxx counting abstraction: n(Z) -> n(2*Z)
  n(Y) :- Y=2*Z, n(Z).
- % xIIIy → xUy counting abstraction: n(Z) -> n(Z-3)
  n(Y) :- Y=Z-3, n(Z).
- % xUUy → xy counting abstraction: n(Z) -> n(Z)
  n(Z) :- n(Z).
- % initial configuration: MI
  n(1).
- % Necessary condition for getting mu
  may_get_mu :- Z=0, n(Z).
- % A more general necessary condition:
  may_get_mu :- Z=3*Y, n(Z).

MU puzzle – End of the story

Abs(MU):

```
n(Z) :- n(Z).
n(Y) :- Y=2*Z, n(Z).
n(Y) :- Y=Z-3, n(Z).
n(Z) :- n(Z).
n(1).
may_get_mu :- Z=3*Y, n(Z).
```

The Theorem Prover Z3 is able to prove that

Abs(MU) $\nvdash$ may_get_mu

because the following invariant holds: $\forall Z\ (n(Z) \rightarrow \exists Y\ Z=3*Y)$

Two key points: (1) Use of counting abstraction,

(2) discovery of the invariant.

Conclusioni

- Il sogno di Leibniz di un Ragionatore Universale Automatico non è pienamente realizzabile.
- Perseguirne la realizzazione ha stimolato uno sviluppo enorme di tecniche e strumenti specifici nel campo dell'informatica e, in particolare, dell'Intelligenza Artificiale.
- La ricerca continua!