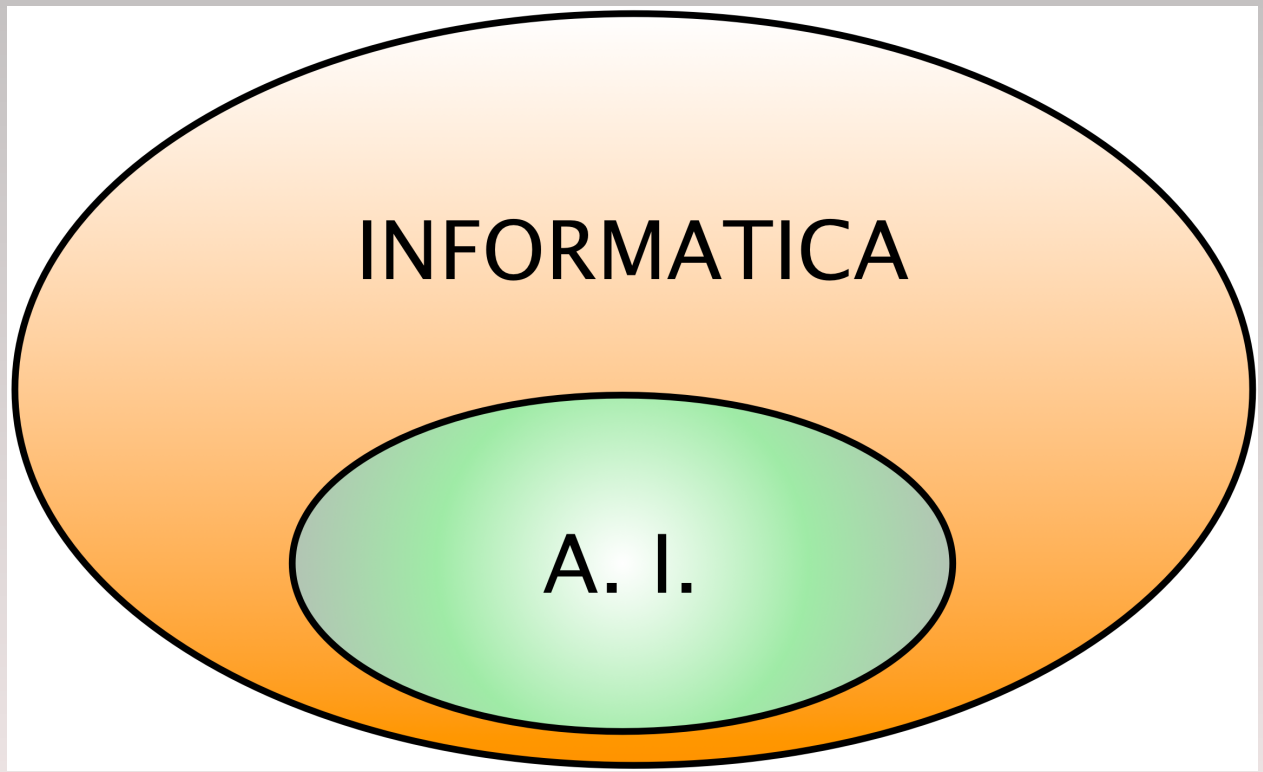
La programmazione logica come metodologia per la codifica e la risoluzione di rompicapi

Agostino Dovier

Dipartimento di Matematica e Informatica
Università di Udine

5 Novembre 2015

Artificial Intelligence



L'Intelligenza Artificiale (AI) è una delle principali e più radicate aree dell'Informatica

Informatica

L'informatica (contrazione di informazione automatica) studia i fondamenti teorici, le tecniche e gli strumenti per la gestione ed elaborazione automatica dell'informazione.

Il nome anglosassone è Computer Science — scienza del calcolatore.

Computer Science is no more about computers than astronomy is about telescopes. [Edsger Wybe Dijkstra]

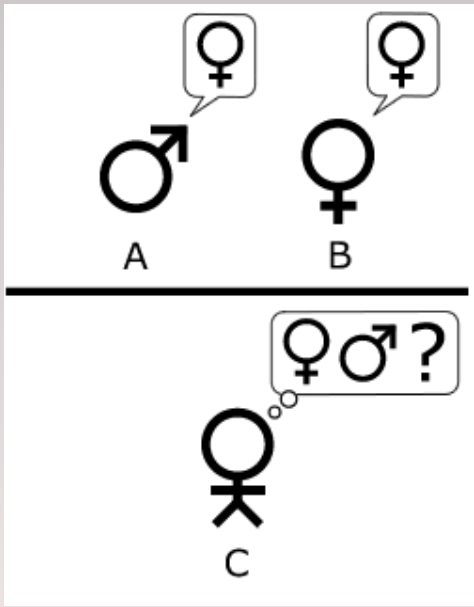
Il padre dell'informatica

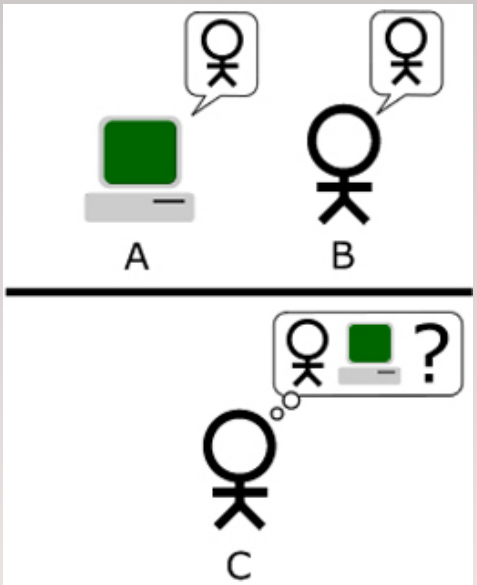
Alan Turing (London 1912 - Cheshire 1954)

Il modello matematico del calcolatore universale:
La **macchina di Turing**.

Turing e l'Intelligenza Artificiale

Il Test di Turing è un criterio per determinare se una macchina sia in grado di pensare [Mind, 1950]:
I propose to consider the question, "Can machines think?" . . .





A (maschio cerca di ingannare), *B* (femmina, cerca di aiutare).
Una macchina è intelligente se si comporta in modo indistinguibile da *A* nell'imitation game.

La nascita dell'Intelligenza Artificiale

L'espressione "Artificial Intelligence" fu coniata nel 1955 dal matematico americano John McCarthy (a sinistra): "the science and engineering of making intelligent machines"

Minsky (a destra) dice che lo scopo di questa nuova disciplina sarebbe stato quello di "far fare alle macchine delle cose che richiederebbero l'intelligenza se fossero fatte dagli uomini"

AI per risolvere problemi: KR (and reasoning)

Knowledge representation is one of the most important subareas of artificial intelligence. If we want to design an entity (a machine or a program) capable of behaving intelligently in some environment, then we need to supply this entity with sufficient knowledge about this environment. To do that, we need an unambiguous language capable of expressing this knowledge, together with some precise and well understood way of manipulating sets of sentences of the language which will allow us to draw inferences, answer queries, and to update both the knowledge base and the desired program behavior.

Chitta Baral and Michael Gelfond. JLP 19/20:73–148, 1994.

Expressing information in declarative sentences is far more modular than expressing it in segments of computer programs or in tables. Sentences can be true in a much wider context than specific programs can be used. The supplier of a fact does not have to understand much about how the receiver functions or how or whether the receiver will use it. The same fact can be used for many purposes, because the logical consequences of collections of facts can be available. [McC59]

J. McCarthy. Programs with Common Sense, 1959.

AI per risolvere problemi: Linguaggi

Vengono introdotti linguaggi di programmazione adatti alla rappresentazione della conoscenza e al ragionamento su essa.

Mc Carthy propone il LISP (LISt Processor) nel 1958.

Kowalski propone il PROLOG (PROgramming with LOGic) nel 1974.

Le comunità di

- Programmazione Funzionale (`www.icfpconference.org/`) e
- Programmazione Logica (`www.logicprogramming.org`)

sono molto attive ancora oggi.
Da LISP e PROLOG nascono un numero enorme di nuovi loro dialetti e altri linguaggi (dichiarativi) a loro ispirati.

AI per risolvere problemi: Search

L'idea di questi linguaggi è che il programmatore deve concentrarsi sulla rappresentazione del problema (rappresentazione della conoscenza–KR).

Il linguaggio possiede tecniche di ragionamento che, data la codifica del problema cerca la soluzione (search).

Nasce allora il problema per l'informatico di studiare tecniche efficienti e generali da inserire come cuore "intelligente" di questi linguaggi.

Ma come si fa a capire se le tecniche sviluppate sono buone o meno?

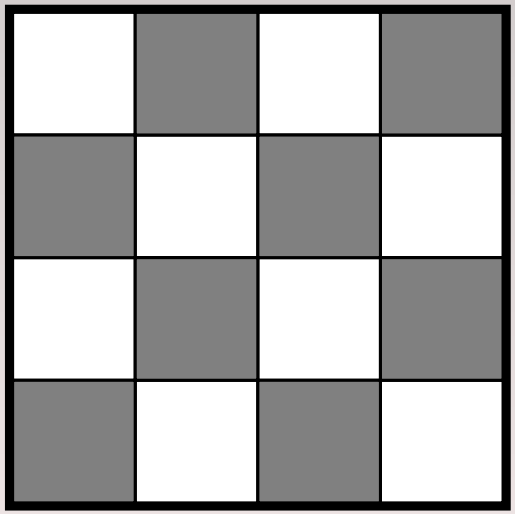
Qui entrano in campo le sfide dei sistemi contro i rompicapi!

Sono sfide in espressività (il linguaggio che permette la codifica più bella) e in velocità (tempo necessario a trovare le soluzioni su esempi).

Esempio: le *N* regine

Cerchiamo di intuire alcune delle tecniche concentrandosi su un problemino.

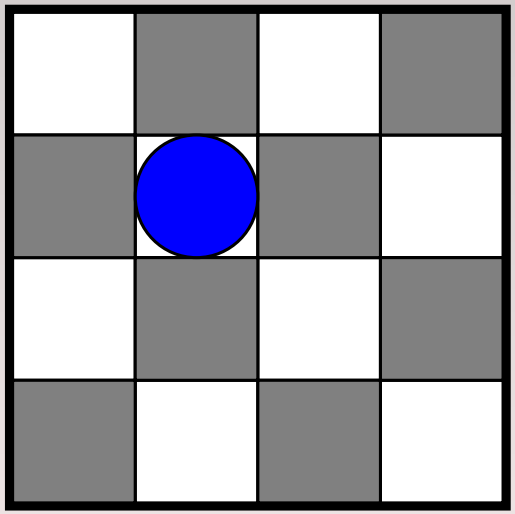
4-Regine: sistemare 4 regine su una scacchiera 4×4 in modo tale che le regine non si attacchino.



Esempio: le *N* regine

Cerchiamo di intuire alcune delle tecniche concentrandosi su un problemino.

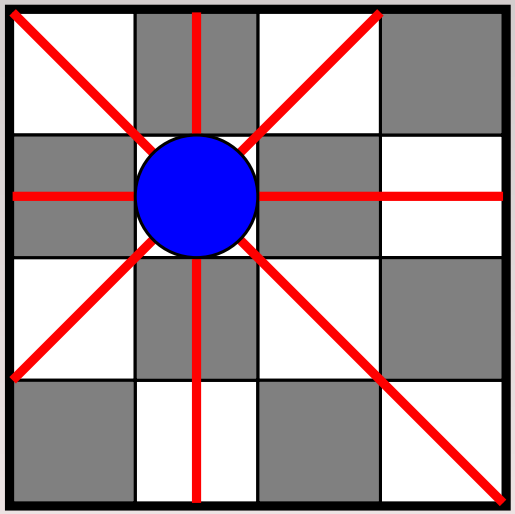
4-Regine: sistemare 4 regine su una scacchiera 4×4 in modo tale che le regine non si attacchino.



Esempio: le *N* regine

Cerchiamo di intuire alcune delle tecniche concentrandosi su un problemino.

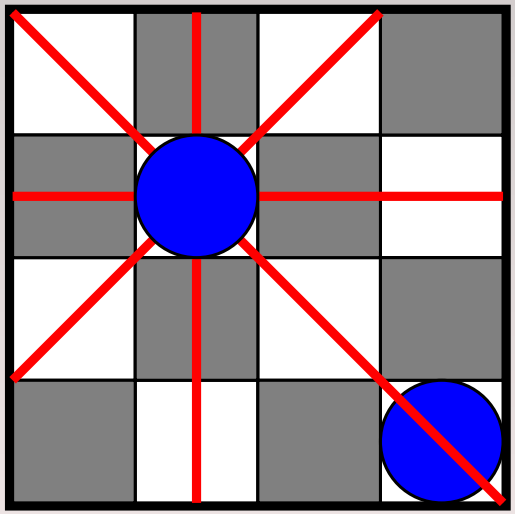
4-Regine: sistemare 4 regine su una scacchiera 4×4 in modo tale che le regine non si attacchino.



Esempio: le *N* regine

Cerchiamo di intuire alcune delle tecniche concentrandosi su un problemino.

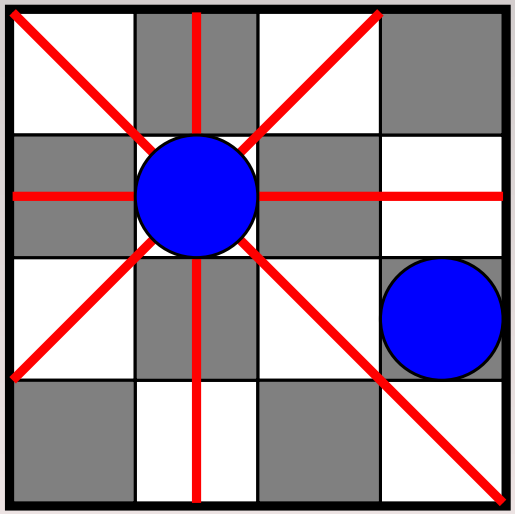
4-Regine: sistemare 4 regine su una scacchiera 4×4 in modo tale che le regine non si attacchino.



Esempio: le *N* regine

Cerchiamo di intuire alcune delle tecniche concentrandosi su un problemino.

4-Regine: sistemare 4 regine su una scacchiera 4×4 in modo tale che le regine non si attacchino.



Codifica (in linguaggi “a vincoli”)

- Variabili: $X_1, \ldots, X_4$
- Domini (valori possibili):

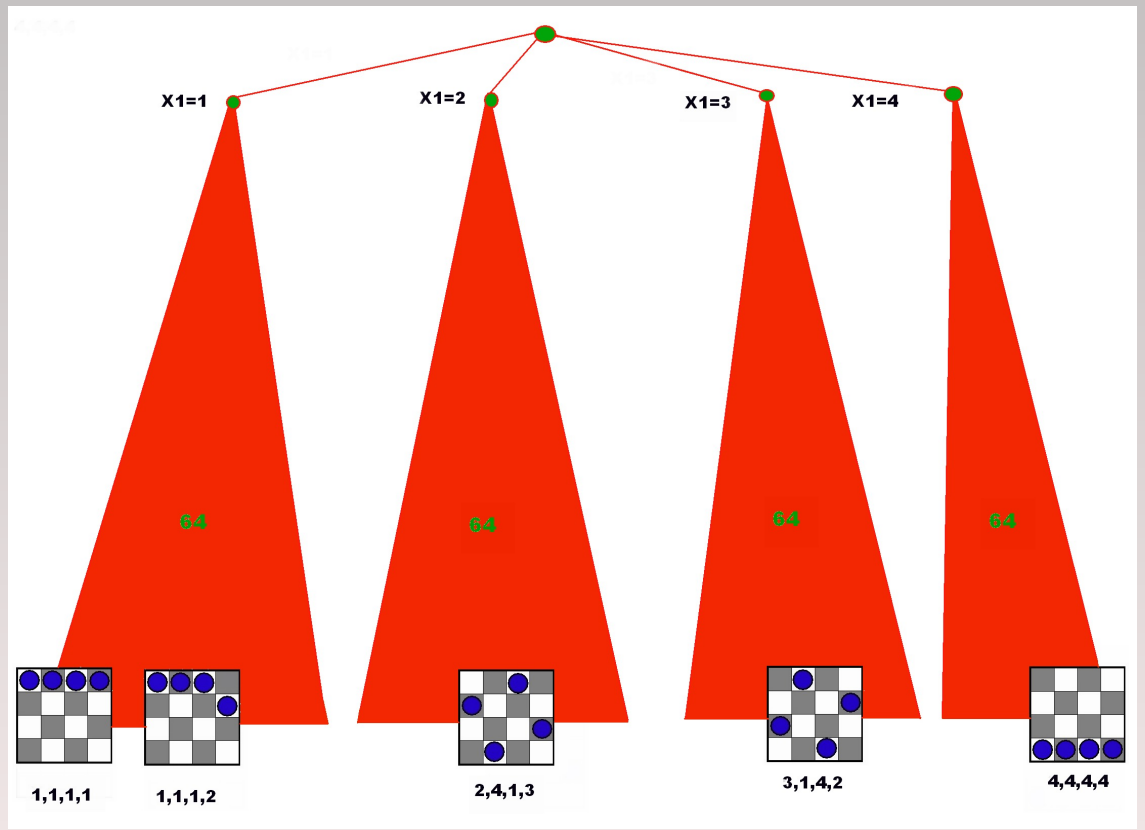
$$Dom(X_1) = Dom(X_2) = Dom(X_3) = Dom(X_4) = \{1, \ldots, 4\}$$

  $X_i = j$ significa “una regina sta in colonna i, riga j”
- Vincoli (per $i, j = 1..4$ e $i < j$)

$$\begin{aligned} X_i &\neq X_j && \text{(attacco orizzontale)} \\ |X_j - X_i| &\neq j - i && \text{(attacco in diagonale)} \end{aligned}$$

Ricerca delle soluzioni

Forza Bruta: Generate and Test



Si visita tutto l'albero di ricerca:

Si rischia di fare $4^4 = 256$ tentativi (N^N in generale)

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli.

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli.
  Assumiamo sia $\frac{1}{10} mm$.

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli. Assumiamo sia $\frac{1}{10}mm$.

1. Pieghiamo un foglio in 2. Diventa spesso $\frac{2}{10}mm = \frac{2^1}{10}mm$

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli. Assumiamo sia $\frac{1}{10}mm$.

1. Pieghiamo un foglio in 2. Diventa spesso $\frac{2}{10}mm = \frac{2^1}{10}mm$
2. Pieghiamo ancora in 2. Diventa ora (attenzione!) $\frac{4}{10}mm = \frac{2^2}{10}mm$

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli.
  Assumiamo sia $\frac{1}{10}mm$.

1. Pieghiamo un foglio in 2. Diventa spesso $\frac{2}{10}mm = \frac{2^1}{10}mm$
2. Pieghiamo ancora in 2. Diventa ora (attenzione!) $\frac{4}{10}mm = \frac{2^2}{10}mm$
3. Pieghiamo ancora in 2. Diventa ora $\frac{8}{10}mm = \frac{2^3}{10}mm$

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli.
  Assumiamo sia $\frac{1}{10}mm$.

1. Pieghiamo un foglio in 2. Diventa spesso $\frac{2}{10}mm = \frac{2^1}{10}mm$
2. Pieghiamo ancora in 2. Diventa ora (attenzione!) $\frac{4}{10}mm = \frac{2^2}{10}mm$
3. Pieghiamo ancora in 2. Diventa ora $\frac{8}{10}mm = \frac{2^3}{10}mm$
4. Pieghiamo ancora in 2. Diventa ora $\frac{16}{10}mm = \frac{2^4}{10}mm = 1.6mm$

Ricerca delle soluzioni

La crescita esponenziale

- Quant'è spesso un foglio di carta? Provate a misurare lo spessore di un quaderno e a dividere per il numero di fogli.
  Assumiamo sia $\frac{1}{10}mm$.

1. Pieghiamo un foglio in 2. Diventa spesso $\frac{2}{10}mm = \frac{2^1}{10}mm$
2. Pieghiamo ancora in 2. Diventa ora (attenzione!) $\frac{4}{10}mm = \frac{2^2}{10}mm$
3. Pieghiamo ancora in 2. Diventa ora $\frac{8}{10}mm = \frac{2^3}{10}mm$
4. Pieghiamo ancora in 2. Diventa ora $\frac{16}{10}mm = \frac{2^4}{10}mm = 1.6mm$
5. Pieghiamo ancora in 2. Diventa ora $\frac{32}{10}mm = \frac{2^5}{10}mm = 3.2mm$

La crescita esponenziale

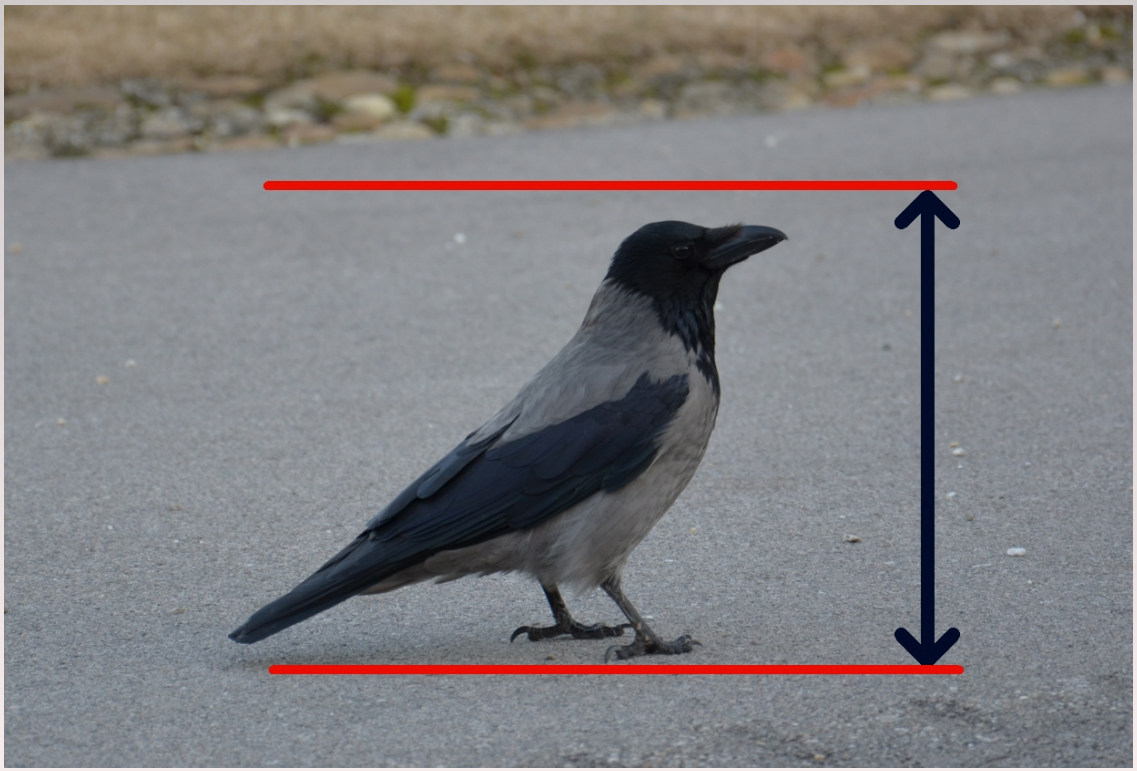
11 piegature

$$\frac{2^{11}}{10} mm = \frac{2048}{10} mm \approx 20 cm$$

La crescita esponenziale

11 piegature

$$\frac{2^{11}}{10} mm = \frac{2048}{10} mm \approx 20 cm$$



La crescita esponenziale

14 piegature

$$\frac{2^{14}}{10} mm = \frac{16384}{10} mm \approx 1.64 m$$

La crescita esponenziale

14 piegature

$$\frac{2^{14}}{10} mm = \frac{16384}{10} mm \approx 1.64 m$$

La crescita esponenziale

15 piegature

$$\frac{2^{15}}{10} mm = \frac{32768}{10} mm \approx 3.27 m$$

La crescita esponenziale

15 piegature

$$\frac{2^{15}}{10} mm = \frac{32768}{10} mm \approx 3.27 m$$

La crescita esponenziale

20 piegature

$$\frac{2^{20}}{10} mm = \frac{1048576}{10} mm \approx 104m$$

La crescita esponenziale

20 piegature

$$\frac{2^{20}}{10} mm = \frac{1048576}{10} mm \approx 104m$$

La crescita esponenziale

42 piegature

$$\frac{2^{42}}{10} mm \approx 439804 Km$$

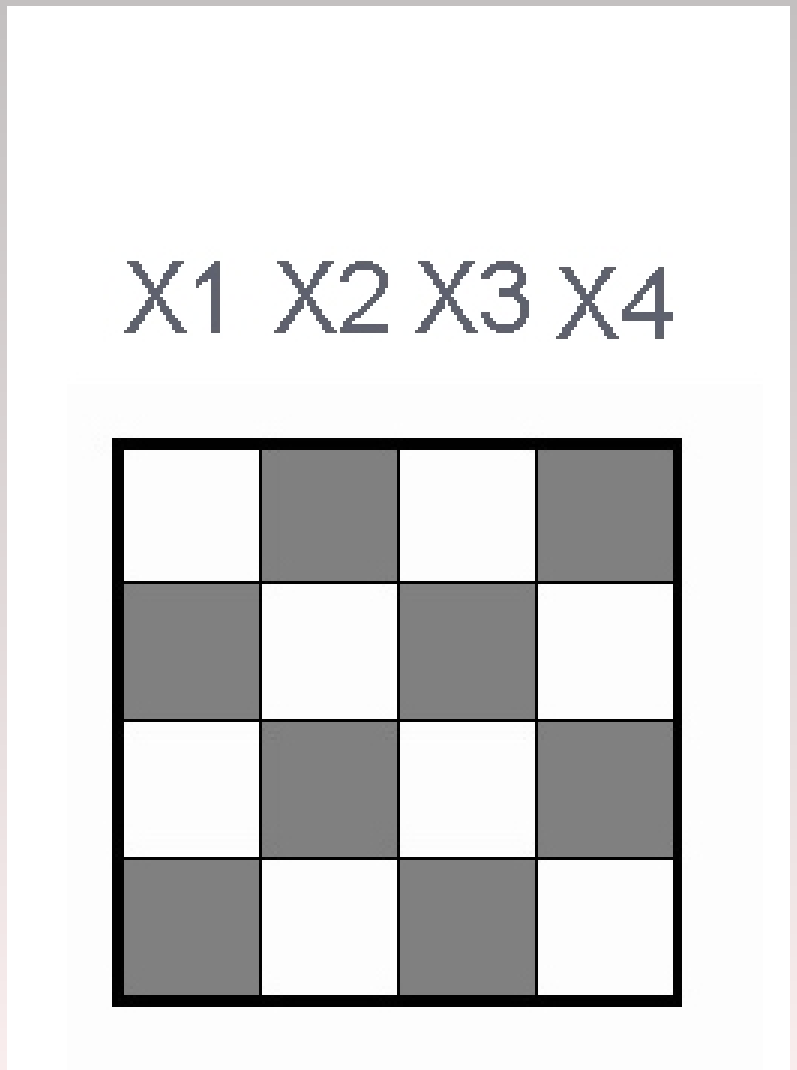
La crescita esponenziale

42 piegature

$$\frac{2^{42}}{10} mm \approx 439804 Km$$

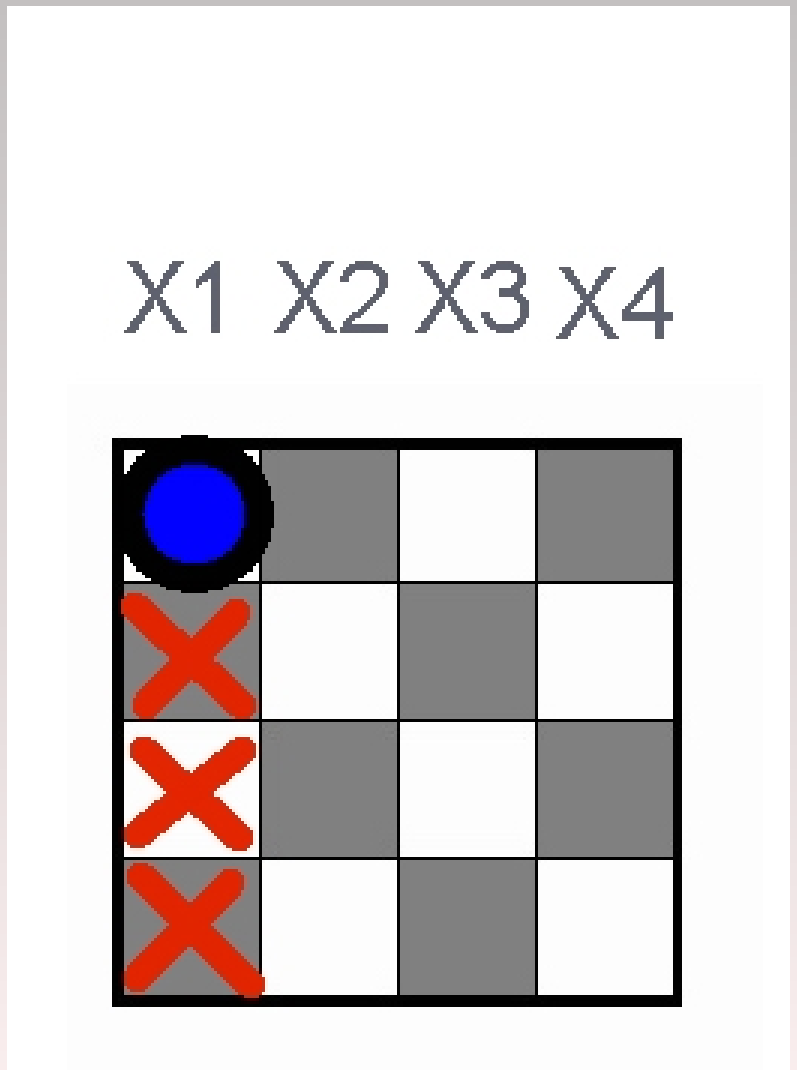
Constraint Propagation

Assegnamo $X_1 = 1$



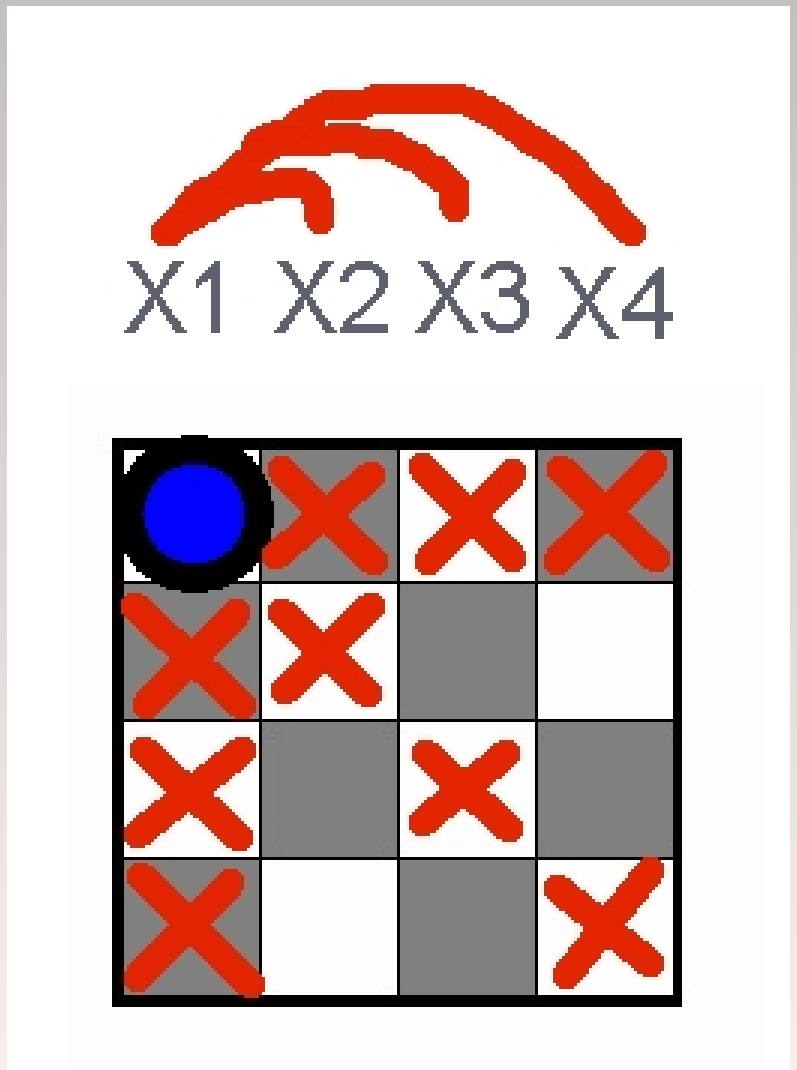
Constraint Propagation

Assegnamo $X_1 = 1$



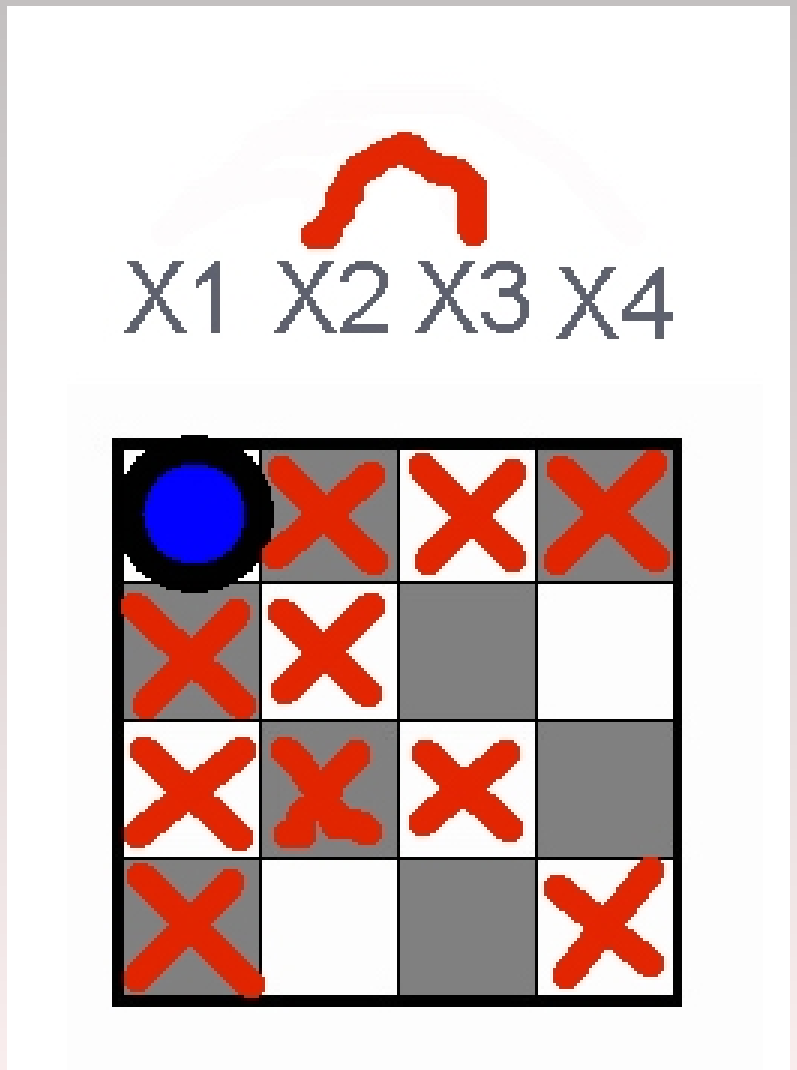
Constraint Propagation

Assegnamo $X_1 = 1$



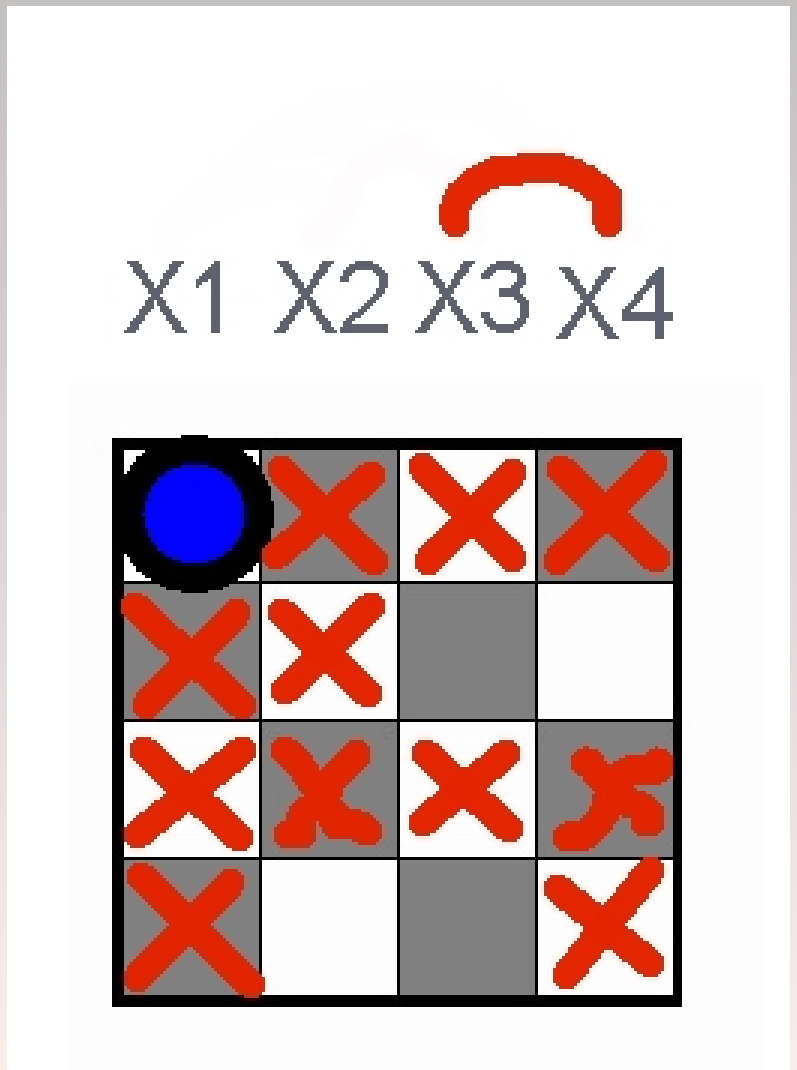
Constraint Propagation

Assegnamo $X_1 = 1$



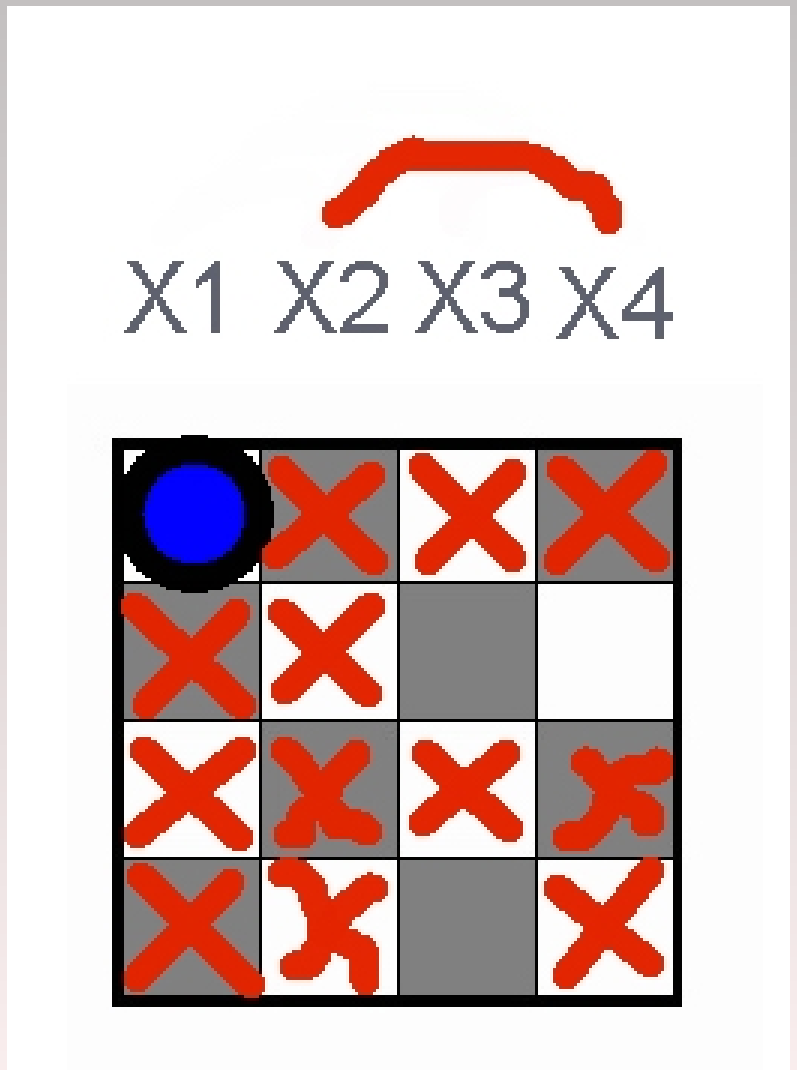
Constraint Propagation

Assegnamo $X_1 = 1$



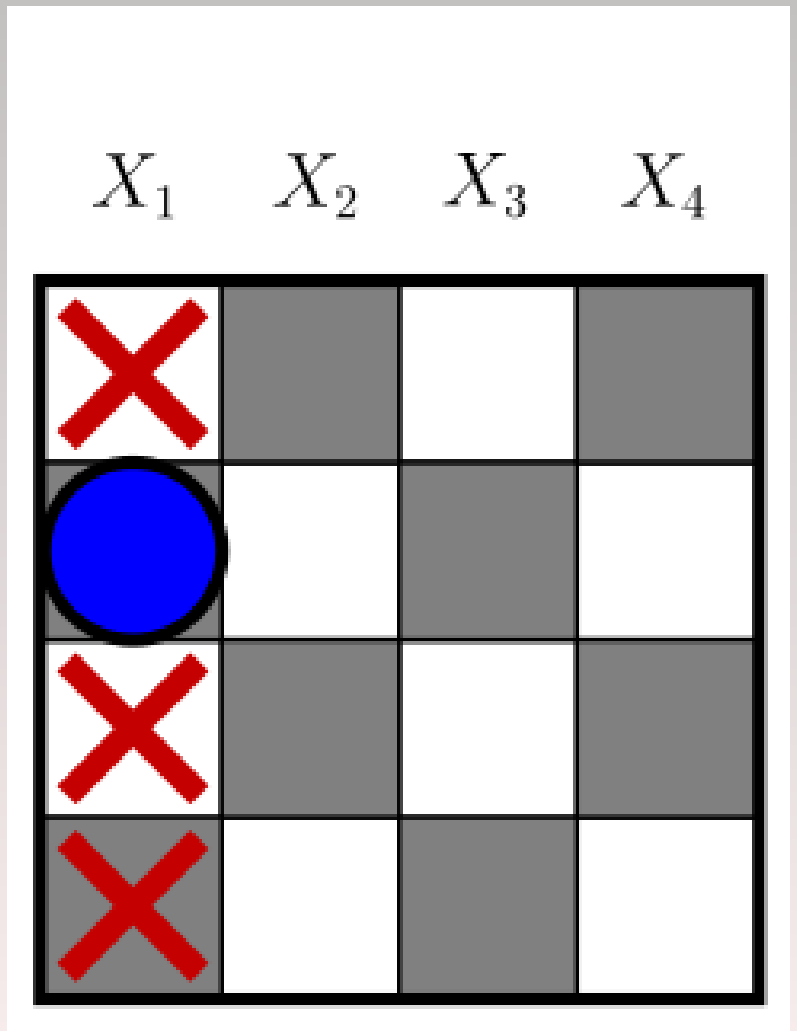
Constraint Propagation

Assegnamo $X_1 = 1$



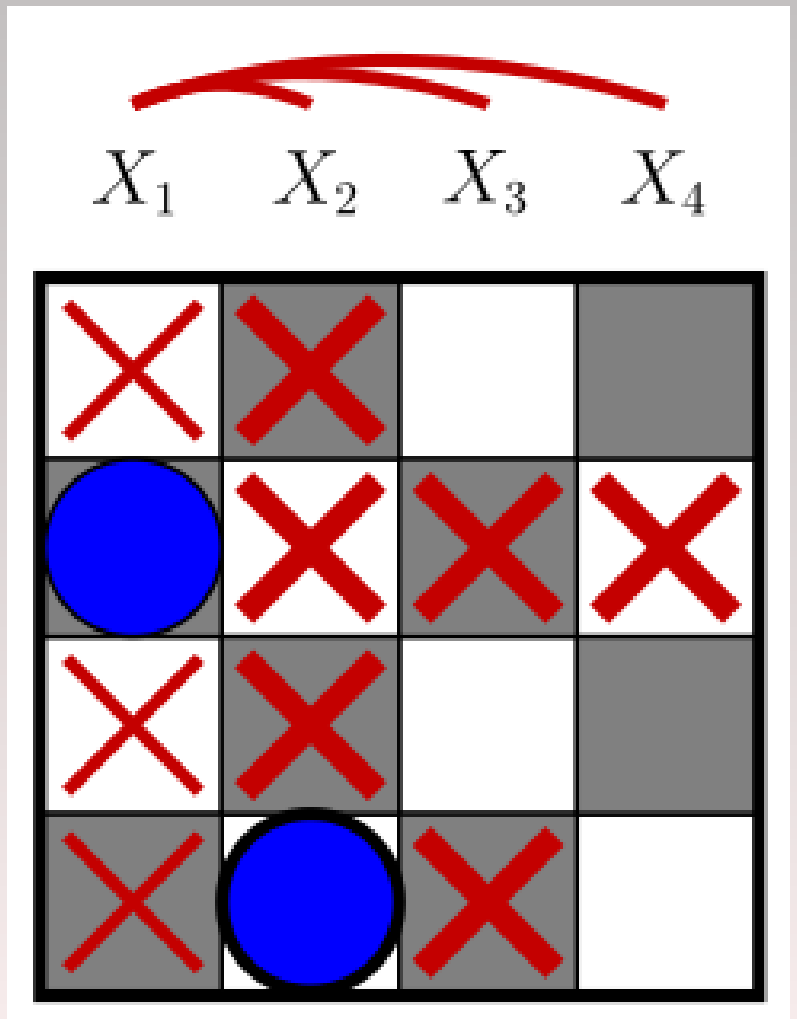
Constraint Propagation

Assegnamo: $X_1 = 2$



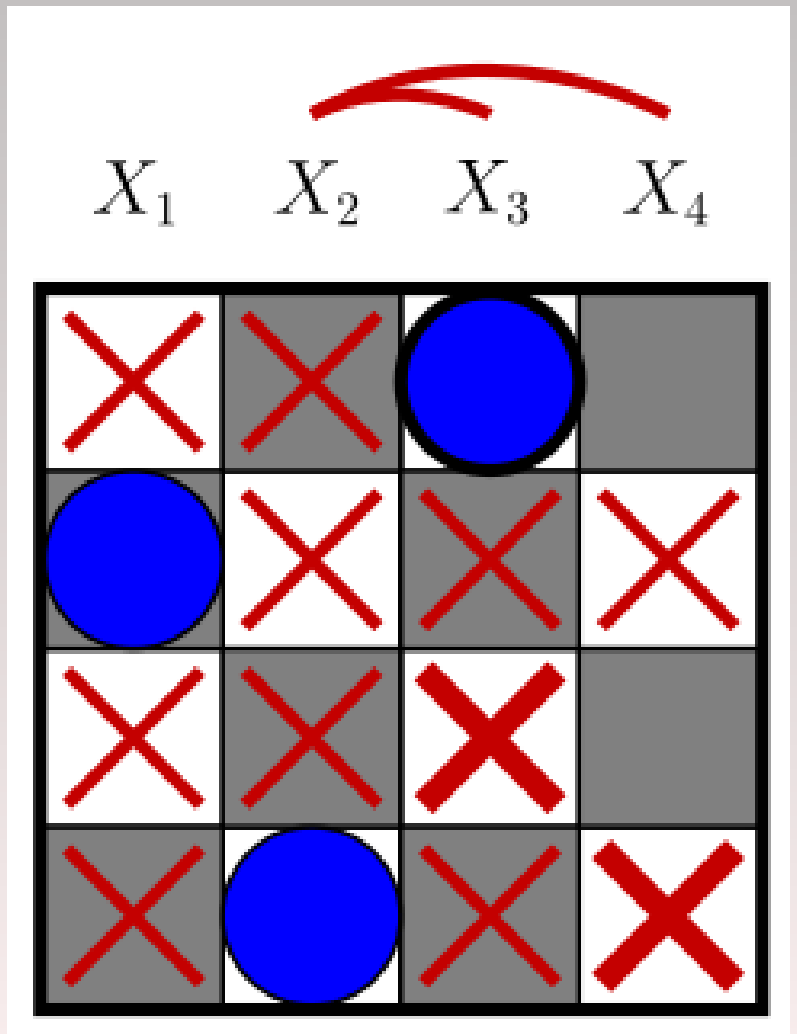
Constraint Propagation

Assegnamo: $X_1 = 2$



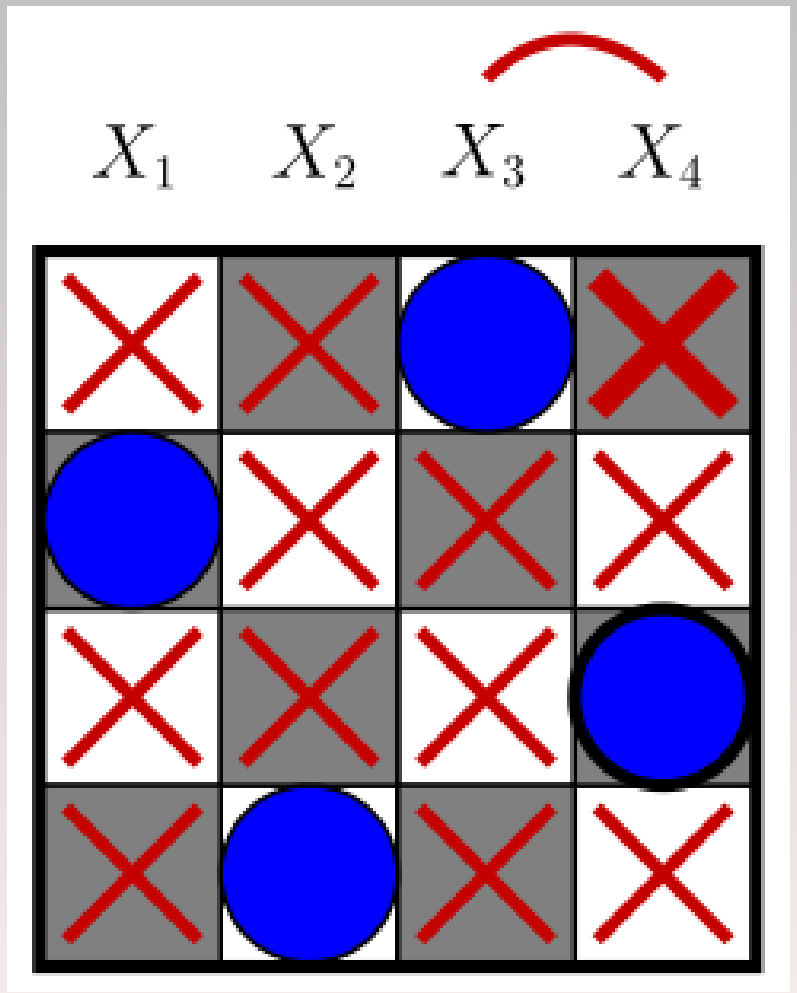
Constraint Propagation

Assegnamo: $X_1 = 2$



Constraint Propagation

Assegnamo: $X_1 = 2$



Search & Constraint Propagation

Con la sola idea di alternare assegnamenti e propagazioni sono passato da 256 tentativi a 2.

Il tentativo è a caso, la propagazione simula (con buoni risultati) un ragionamento

I linguaggi per l'AI contengono molte tecniche come queste (spesso invisibili al programmatore) per trovare le soluzioni ai problemi codificati.

Quali problemi?

Esempi di problemi AI

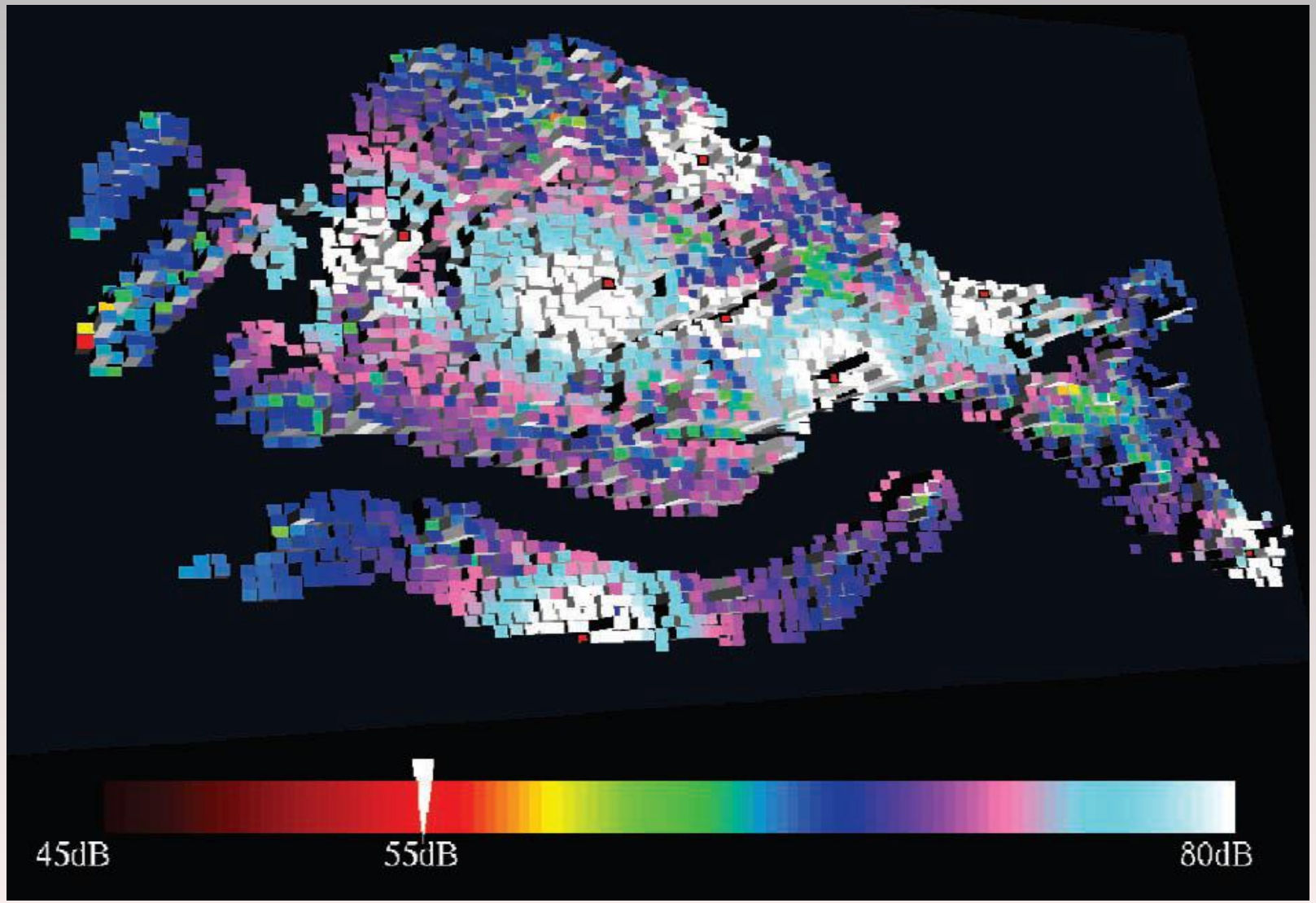
Esempi di problemi AI

Esempi di problemi AI

Esempi di problemi AI

Esempi di problemi AI

Esempi di problemi AI



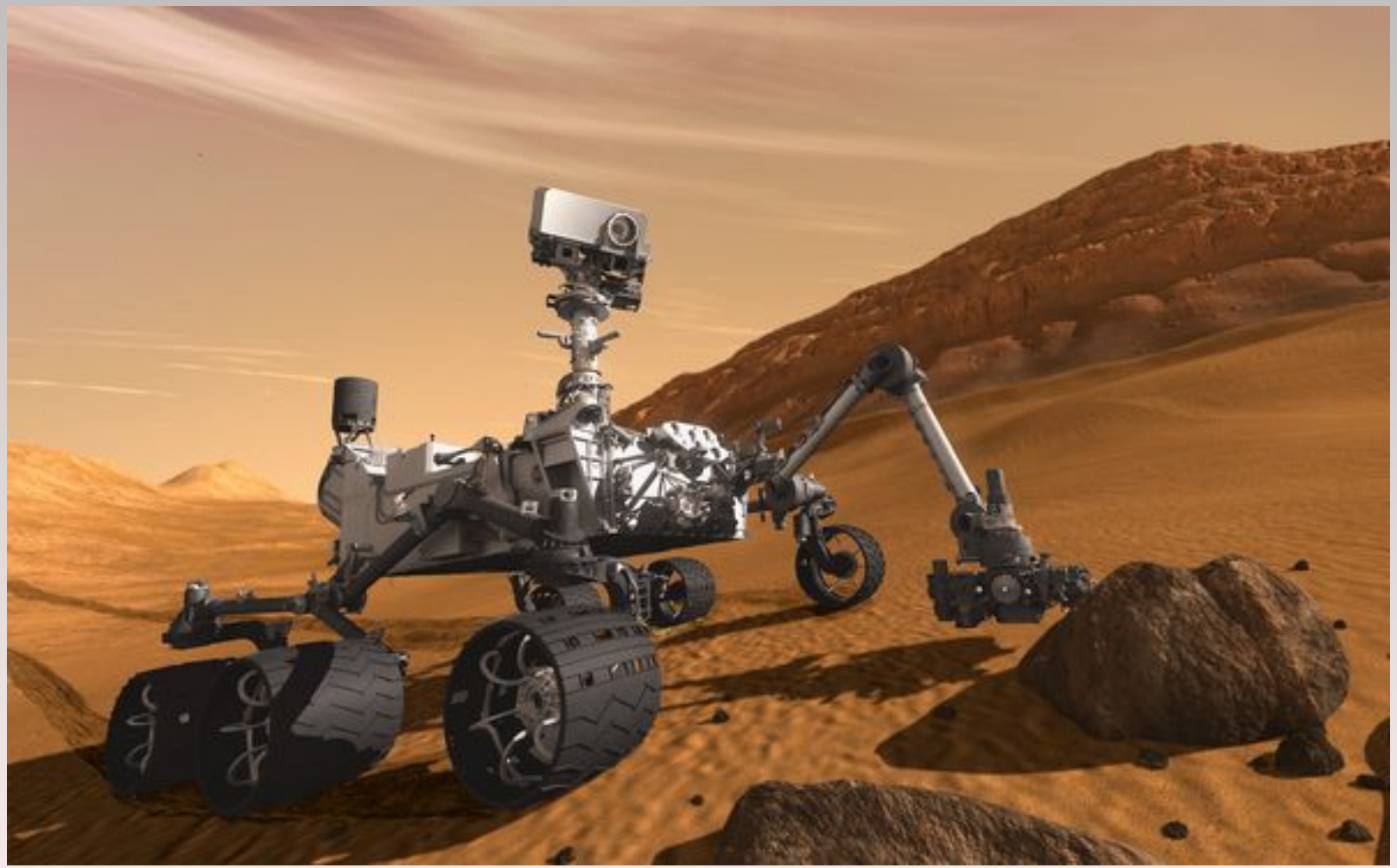
Esempi di problemi AI

Esempi di problemi AI

Esempi di problemi AI

Esempi di problemi AI

Esempi di problemi AI



Rompicapi come benchmarks

Come dicevamo, per capire se le tecniche sviluppate sono buone o meno si sfidano i linguaggi (o i solver) su una serie di problemi detti benchmarks.

Nelle varie competizioni, gran parte dei benchmarks sono dei giochi (dei rompicapi)

Vediamone alcuni.

La capra e il cavolo

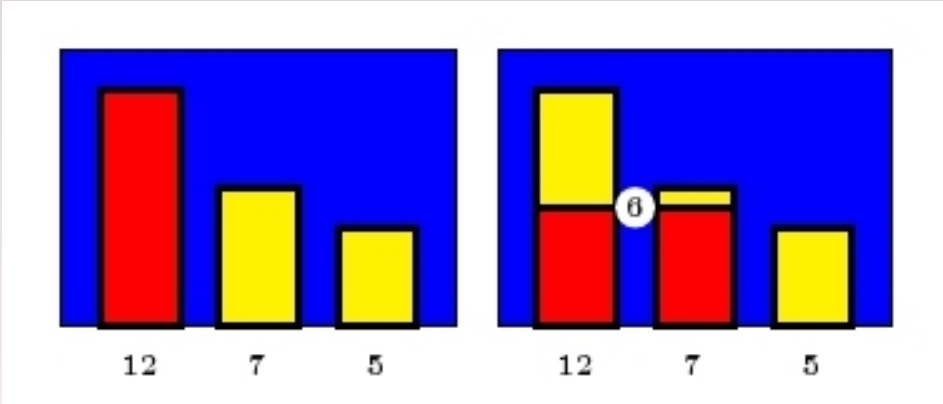
Un pastore deve attraversare un fiume portando sull'altra riva un lupo e una capra affamati e un cavolo gigante. Ha a disposizione una barca a remi con la quale può traghettare un solo oggetto o animale alla volta. Ma, attenzione, non può lasciare da soli:

- il lupo e la capra perché il lupo si mangia la capra;
- la capra ed il cavolo perchè la capra si mangia il cavolo.

Quanti viaggi deve fare per portare sull'altra riva il lupo, la capra ed il cavolo?

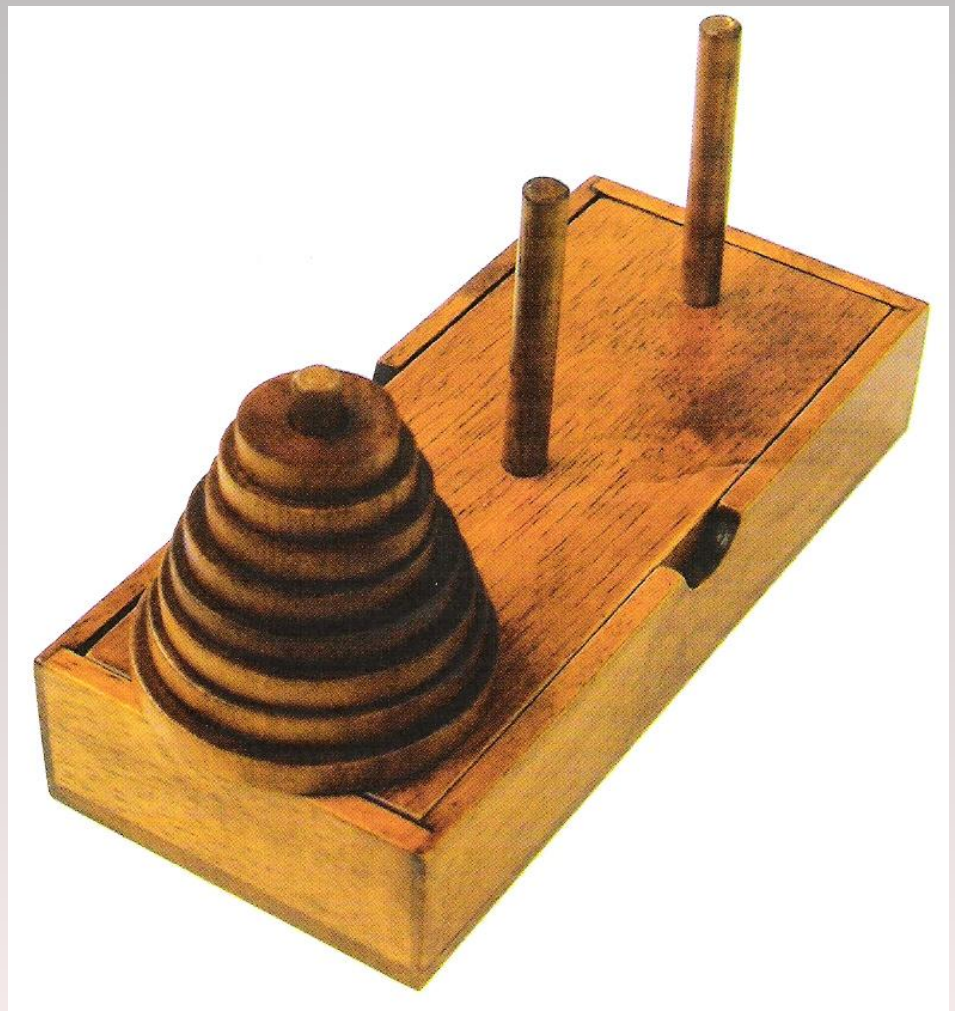
The three barrels

"Ci sono tre botti di capacità N (pari), $N/2+1$, e $N/2-1$ (esempio: 12,7,5). All'inizio la botte più capace è piena di vino e le altre sono vuote. Si vuol raggiungere una conformazione in cui la più piccola è vuota e le altre due contengono esattamente la stessa quantità. Le sole azioni permesse sono di svuotare una botte in un'altra (non è permesso di misurare i flussi o l'altezza del vino o pesare le botti)."



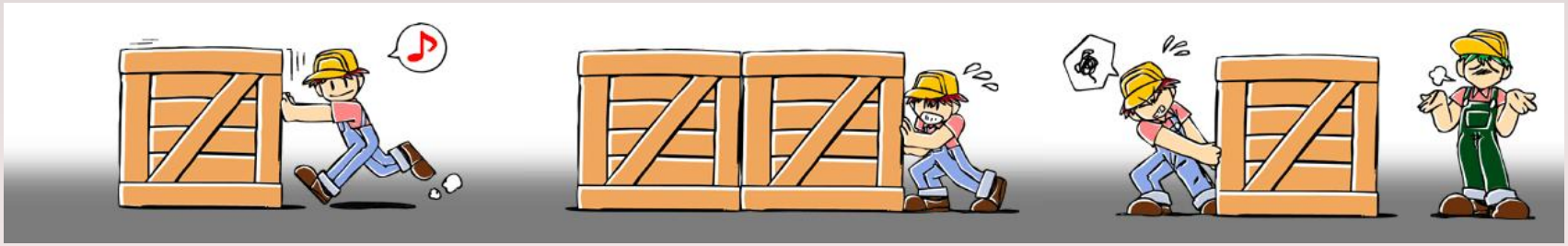
The Hanoi Tower

The Hanoi Tower

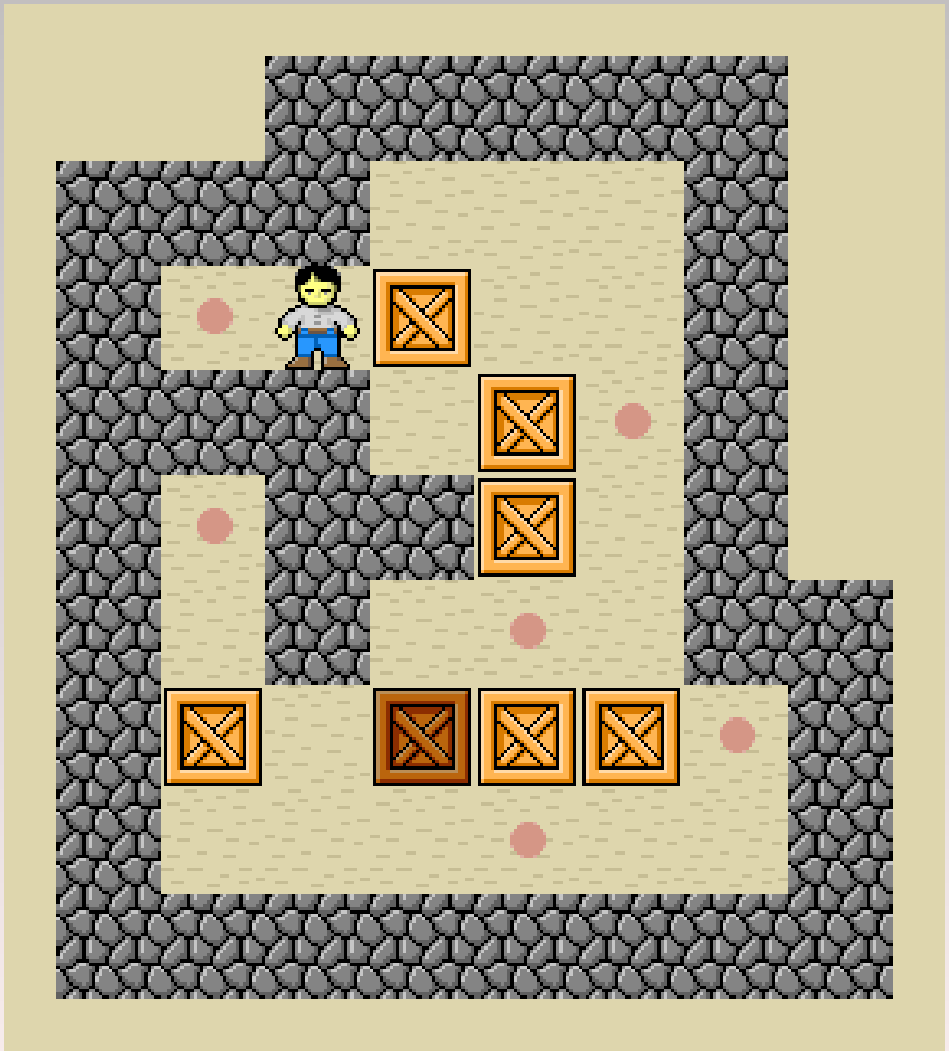


Sokoban

- Sokoban è un rompicapo che coinvolge problemi di trasporto, magazzini, robotica inventato da *Hiroyuki Imabayashi* nel 1980
- Sokoban significa magazziniere in Giapponese
- E' uno dei primi videogiochi. Ha solo tre regole:

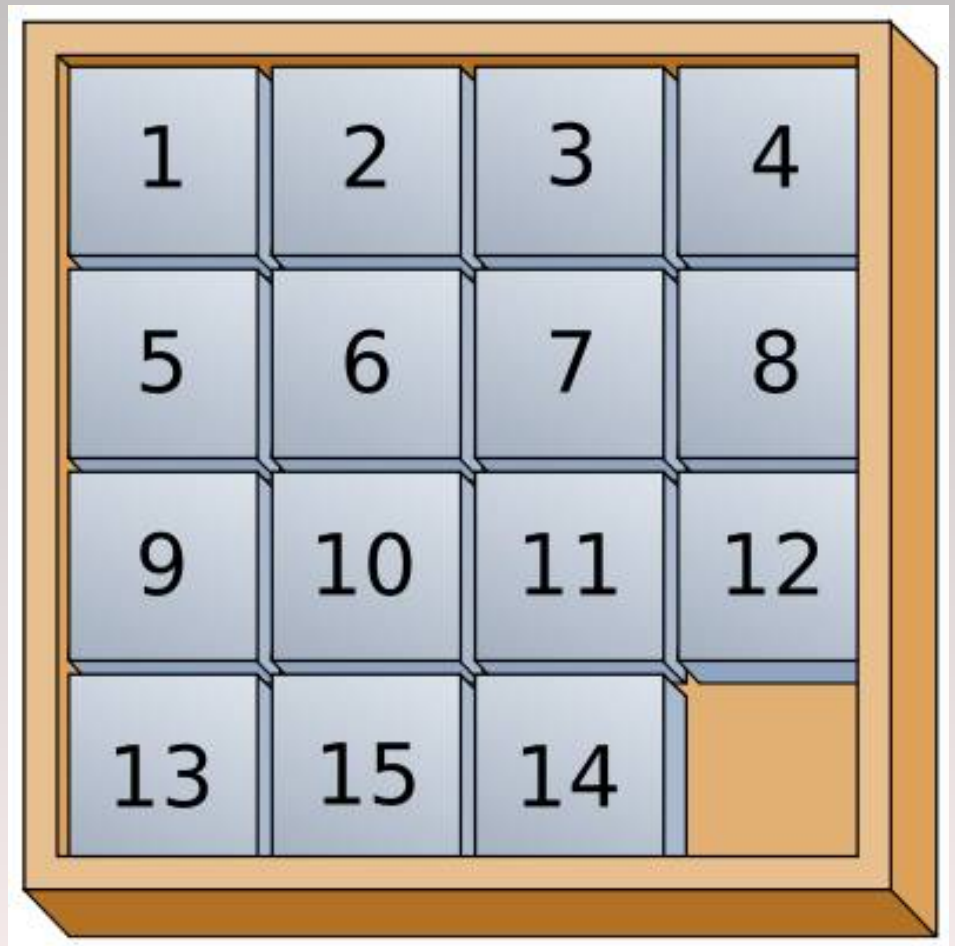


Sokoban @ work

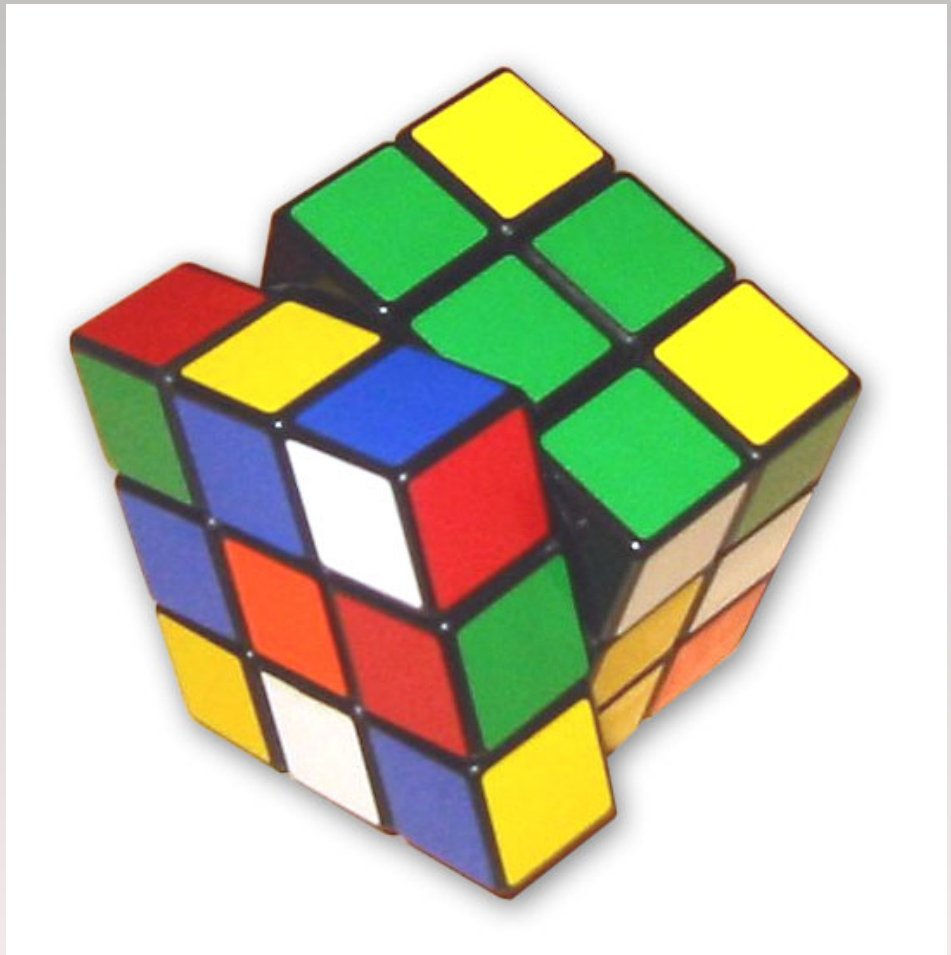


Peg solitaire

Sam Lloyd's Puzzle



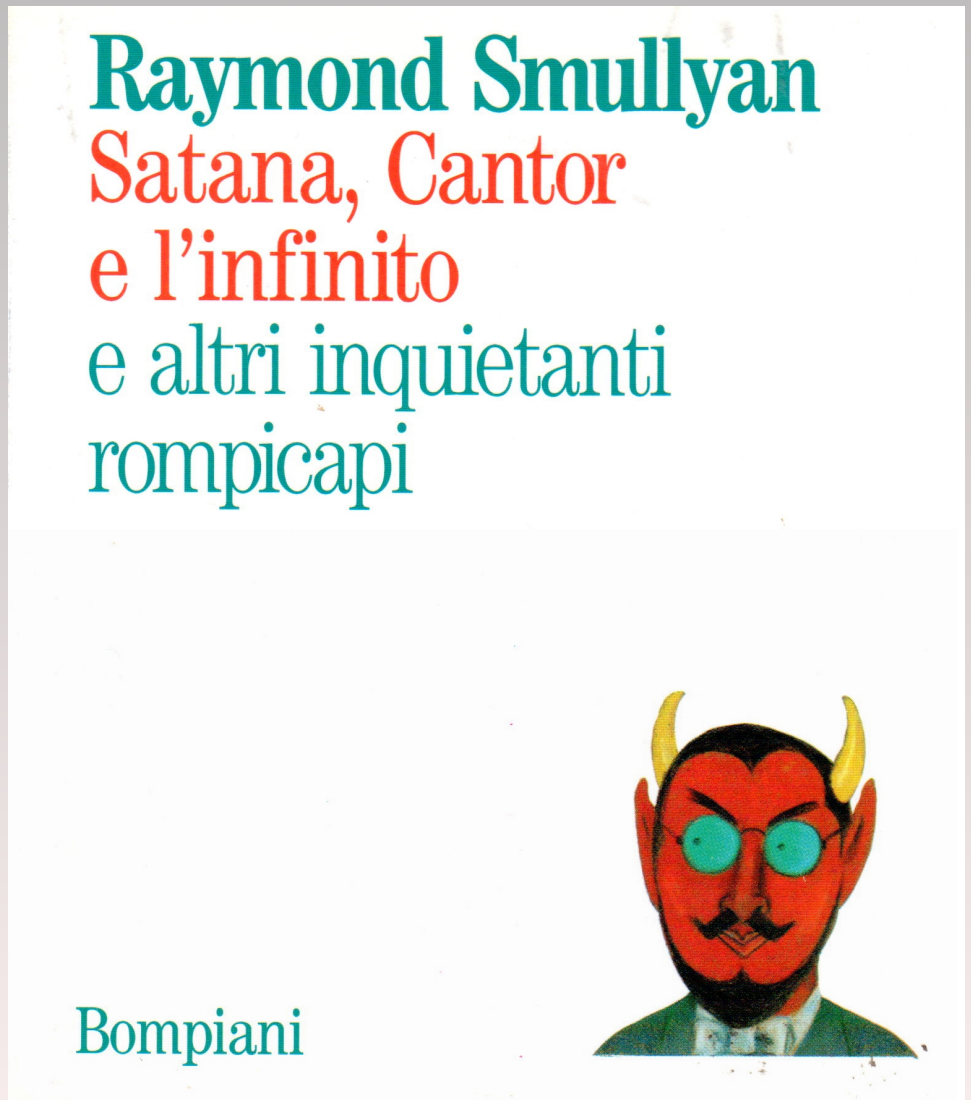
Il cubo di Rubik



Sudoku

		1						
		2		3				4
			5			6		7
5			1	4				
	7						2	
				7	8			9
8		7			9			
4				6		3		
						5		

Smullyan's puzzles



Smullyan's puzzles

Con una fitta di apprensione come non aveva mai provato prima, un antropologo di nome Abercrombie mise piede sull'isola dei Cavalieri e dei Fanti. Sapeva che quest'isola era popolata da gente quanto mai bizzarra: i cavalieri, che facevano solo affermazioni vere, e i fanti, che facevano solo affermazioni false. "Come farò a sapere qualcosa di quest'isola se non riesco a distinguere chi mente e chi dice la verità?" si chiedeva Abercrombie.

Abercrombie si rendeva conto che, prima di poter scoprire alcunché, avrebbe dovuto farsi un amico, qualcuno di cui potesse essere certo che gli avrebbe sempre detto la verità. Così, quando si imbatté nel primo gruppo di indigeni — tre persone, i cui nomi erano presumibilmente Arturo, Bernardo e Carlo —, Abercrombie pensò tra sé: "Ecco una buona occasione per trovarmi un cavaliere." Anzitutto Abercrombie chiese ad Arturo: "Bernardo e Carlo sono entrambi cavalieri?" Arturo rispose: "Sì." Allora Abercrombie chiese: "Bernardo è un cavaliere?" Con sua grande sorpresa, Arturo rispose: "No."

Carlo è un cavaliere o un fante?

Smullyan's puzzles

Abercrombie fu informato da quello dei tre che sapeva ormai essere un cavaliere che l'isola aveva uno stregone.

"Oh, ottimo!" esclamò Abercrombie. "Noi antropologi abbiamo un interesse particolare per gli stregoni, i maghi, i guaritori, gli sciamani e altra gente del genere. Dove posso trovarlo?"

"Deve chiedere al re" fu la risposta.

Bene, l'antropologo riuscì a ottenere un'udienza dal re, e gli disse che desiderava incontrare lo stregone.

"Oh, non può incontrarlo," disse il re, "se non incontra prima il suo apprendista. Se l'apprendista stregone darà di lei un giudizio favorevole, allora le consentirà di incontrare il suo maestro; altrimenti non glielo permetterà."

"Ha un apprendista, lo stregone?" chiese l'antropologo.

"Certamente!" rispose il re. "C'è una famosa composizione musicale su di lui: credo che il compositore sia Dukas. Comunque, se lei desidera incontrare l'apprendista stregone, adesso è a casa sua, che è la terza casa di via delle Palme. Al momento, ha due ospiti. Se, arrivando, lei riesce a dedurre quale delle tre persone presenti è l'apprendista stregone, credo che farà abbastanza colpo su di lui perché acconsenta a farle incontrare lo stregone. Buona fortuna!"

Smullyan's puzzles

Una breve camminata condusse l'antropologo alla casa. Quando entrò, erano presenti tre persone.

"Chi di voi è l'apprendista stregone?" chiese Abercrombie.

"Io," rispose uno.

"Sono io l'apprendista stregone!" gridò un secondo.

Il terzo rimase in silenzio.

"Può dirmi qualcosa anche lei?" chiese Abercrombie.

"È buffo," rispose la terza persona, con un sorriso scaltro. "Al massimo, uno soltanto di noi tre dice la verità!"

È possibile dedurre quale dei tre è l'apprendista stregone?

The zebra puzzle

Vi sono cinque case di cinque colori diversi: rosso, verde, avorio, blu, giallo.

The zebra puzzle

Vi sono cinque case di cinque colori diversi: rosso, verde, avorio, blu, giallo.
Gli inquilini delle case hanno nazionalità diversa. Essi provengono da Giappone, Inghilterra, Norvegia, Ucraina, e Spagna.

The zebra puzzle

Vi sono cinque case di cinque colori diversi: rosso, verde, avorio, blu, giallo.
Gli inquilini delle case hanno nazionalità diversa. Essi provengono da Giappone, Inghilterra, Norvegia, Ucraina, e Spagna.
Ogni inquilino possiede un animale. Gli animali sono: cavallo, chiocciola, zebra, volpe, e cane.

The zebra puzzle

Vi sono cinque case di cinque colori diversi: rosso, verde, avorio, blu, giallo.
Gli inquilini delle case hanno nazionalità diversa. Essi provengono da Giappone, Inghilterra, Norvegia, Ucraina, e Spagna.
Ogni inquilino possiede un animale. Gli animali sono: cavallo, chiocciola, zebra, volpe, e cane.
Inoltre sappiamo che ogni inquilino beve usualmente una sola delle seguenti bevande: acqua, caffè, tea, latte, aranciata

The zebra puzzle

Vi sono cinque case di cinque colori diversi: rosso, verde, avorio, blu, giallo.
Gli inquilini delle case hanno nazionalità diversa. Essi provengono da Giappone, Inghilterra, Norvegia, Ucraina, e Spagna.
Ogni inquilino possiede un animale. Gli animali sono: cavallo, chiocciola, zebra, volpe, e cane.
Inoltre sappiamo che ogni inquilino beve usualmente una sola delle seguenti bevande: acqua, caffè, tea, latte, aranciata
e guida una (sola) auto di una delle seguenti marche: Fiat, Lancia, BMW, Skoda, Audi.

The zebra puzzle

Sono note inoltre le seguenti informazioni:

- l'inglese vive nella casa rossa;
- lo spagnolo ha il cane;
- il norvegese vive nella prima casa a sinistra;
- nel garage della casa gialla c'è una Skoda;
- chi guida la BMW vive nella casa vicina a chi possiede la volpe;
- il norvegese vive in una casa vicino alla casa blu;
- chi possiede la Lancia possiede anche la chiocciola;
- chi guida la Fiat beve aranciata;
- l'ucraino beve tea;
- il giapponese guida l'Audi;
- la Skoda è parcheggiata nel garage di una casa vicina alla casa dove c'è il cavallo;
- nella casa verde si beve caffè;
- la casa verde è immediatamente a destra della casa avorio;
- il latte si beve nella casa di mezzo (la terza).

The zebra puzzle

La domanda e': E' stata ritrovata una zebra in mezzo alla strada. Chi la ha lasciata scappare?

Le 12 monete

- Ci sono 12 monete
- Sappiamo che una di esse è falsa
- Quella falsa ha un peso diverso dalle altre 11 (non sappiamo se più pesante o più leggera)

Le 12 monete

- Ci sono 12 monete
- Sappiamo che una di esse è falsa
- Quella falsa ha un peso diverso dalle altre 11 (non sappiamo se più pesante o più leggera)
- Per scoprirlo disponiamo di una bilancia come questa:

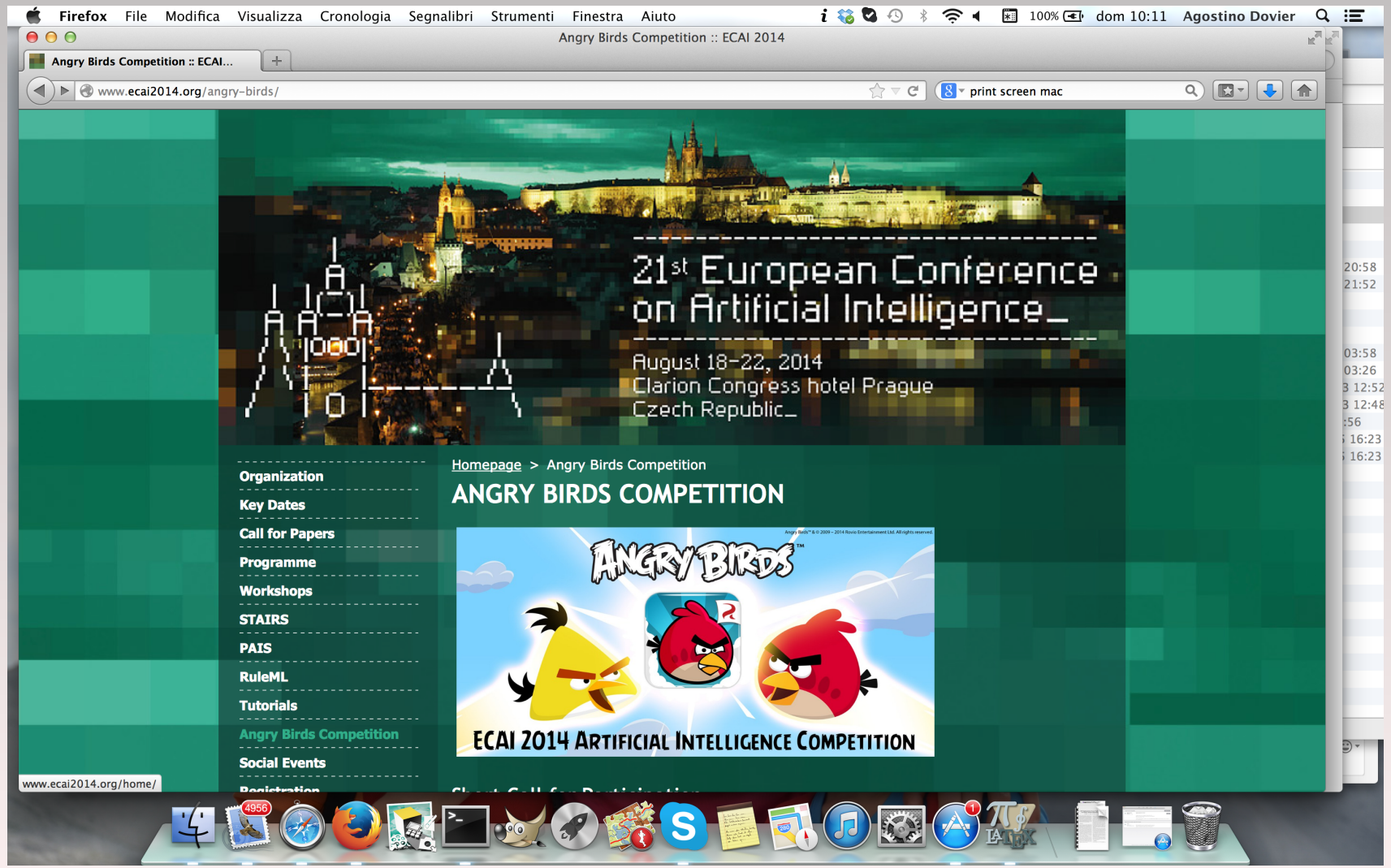
Le 12 monete

- Ci sono 12 monete
- Sappiamo che una di esse è falsa
- Quella falsa ha un peso diverso dalle altre 11 (non sappiamo se più pesante o più leggera)
- Per scoprirlo disponiamo di una bilancia come questa:

- Come possiamo farlo in sole 3 pesate?

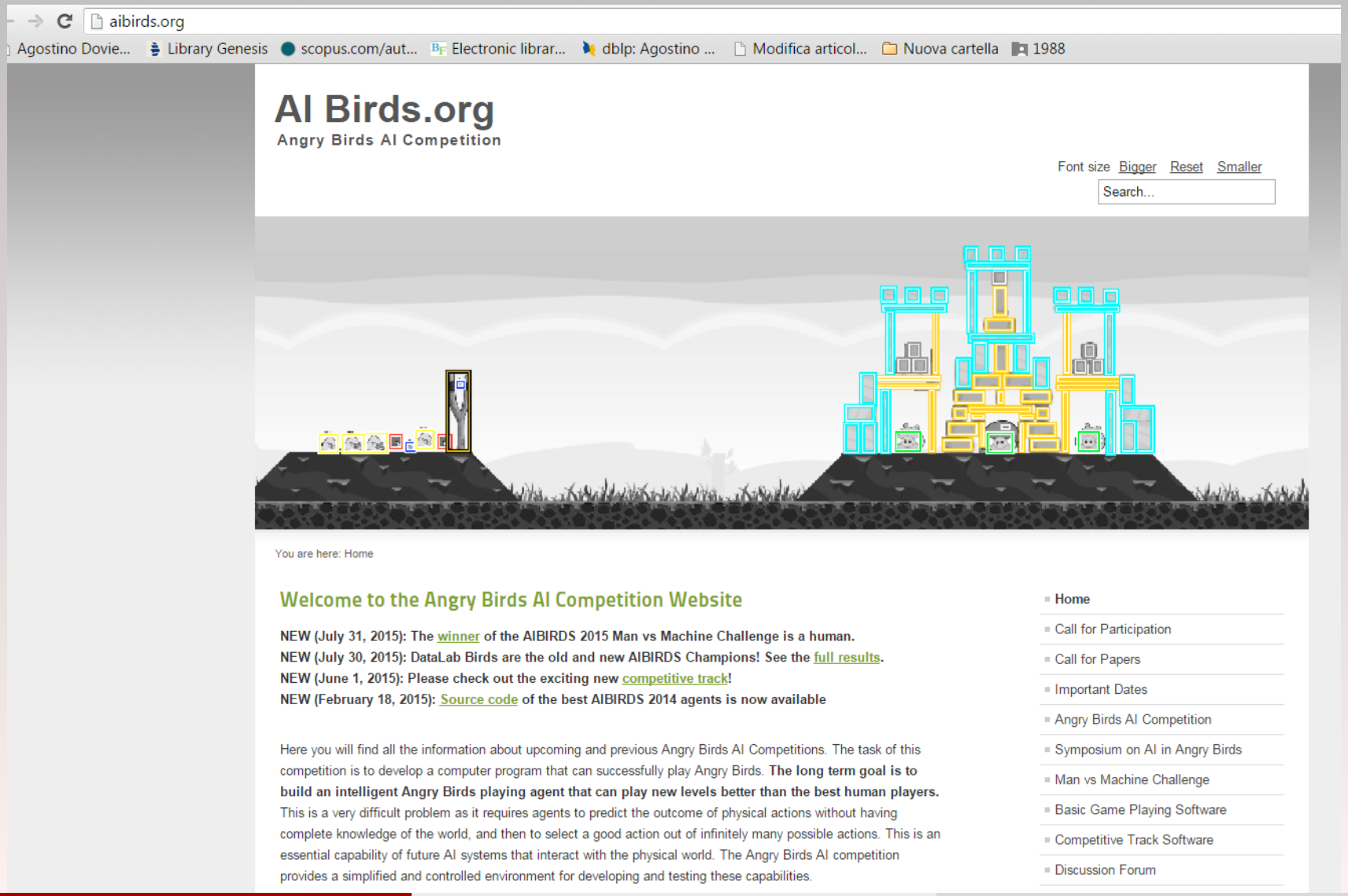
Angry Birds

ECAI 2014



Angry Birds

IJCAI 2015



Knowledge Representation

Knowledge representation is one of the most important subareas of artificial intelligence. If we want to design an entity (a machine or a program) capable of behaving intelligently in some environment, then we need to supply this entity with sufficient knowledge about this environment. To do that, we need an unambiguous language capable of expressing this knowledge, together with some precise and well understood way of manipulating sets of sentences of the language which will allow us to draw inferences, answer queries, and to update both the knowledge base and the desired program behavior.

Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation Journal of Logic Programming 19/20, 73–148, 1994. (disponibile on-line il RR di UTEP dalle pagine degli autori)

Knowledge Representation

Expressing information in declarative sentences is far more modular than expressing it in segments of computer programs or in tables. Sentences can be true in a much wider context than specific programs can be used. The supplier of a fact does not have to understand much about how the receiver functions or how or whether the receiver will use it. The same fact can be used for many purposes, because the logical consequences of collections of facts can be available. [McC59]

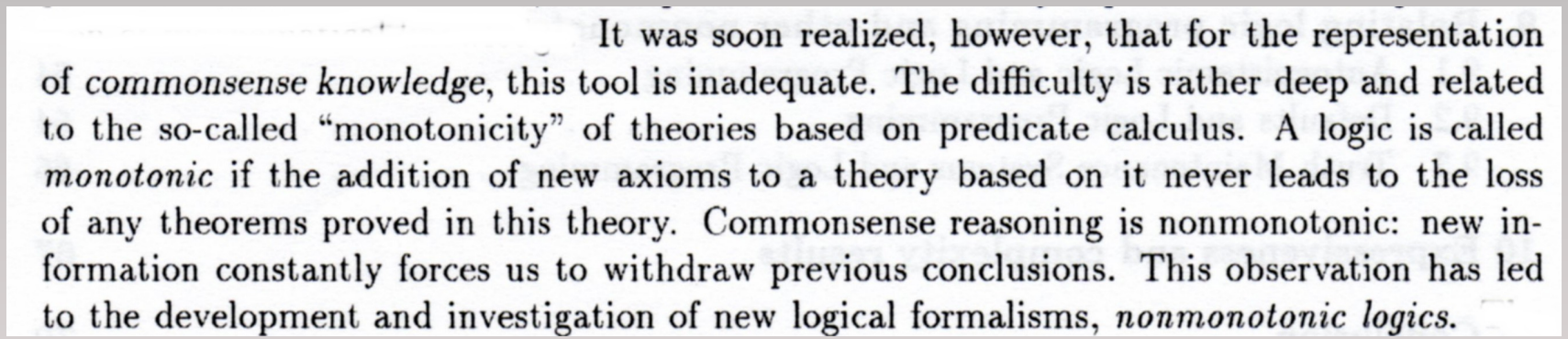
Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation Journal of Logic Programming 19/20, 73–148, 1994. (disponibile on-line il RR di UTEP dalle pagine degli autori)

Knowledge Representation

This idea has been further developed by many researchers with various backgrounds and interests. First, the classical logic of predicate calculus served as the main technical tool for the representation of knowledge. It has a well-defined semantics and a well-understood and powerful inference mechanism, and it proved to be sufficiently expressive for the representation of mathematical knowledge.

Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation Journal of Logic Programming 19/20, 73–148, 1994. (disponibile on-line il RR di UTEP dalle pagine degli autori)

Knowledge Representation



It was soon realized, however, that for the representation of *commonsense knowledge*, this tool is inadequate. The difficulty is rather deep and related to the so-called "monotonicity" of theories based on predicate calculus. A logic is called *monotonic* if the addition of new axioms to a theory based on it never leads to the loss of any theorems proved in this theory. Commonsense reasoning is nonmonotonic: new information constantly forces us to withdraw previous conclusions. This observation has led to the development and investigation of new logical formalisms, *nonmonotonic logics*.

Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation Journal of Logic Programming 19/20, 73–148, 1994. (disponibile on-line il RR di UTEP dalle pagine degli autori)

Knowledge Representation

Il sillogismo

Socrate è un uomo

Knowledge Representation

Il sillogismo

Socrate è un uomo
Tutti gli uomini sono mortali

Knowledge Representation

Il sillogismo

Socrate è un uomo
Tutti gli uomini sono mortali
Pertanto Socrate è mortale

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo(socrate)} \tag{1}$$

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo(socrate)} \tag{1}$$

Tutti gli uomini sono mortali:

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo}(\texttt{socrate}) \tag{1}$$

Tutti gli uomini sono mortali:

$$\forall X\,(\texttt{uomo}(X) \rightarrow \texttt{mortale}(X)) \tag{2}$$

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo}(\texttt{socrate}) \tag{1}$$

Tutti gli uomini sono mortali:

$$\forall X\,(\texttt{uomo}(X) \rightarrow \texttt{mortale}(X)) \tag{2}$$

Riflettiamo sul verso dell'implicazione.

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo}(\texttt{socrate}) \tag{1}$$

Tutti gli uomini sono mortali:

$$\forall X\,(\texttt{uomo}(X) \rightarrow \texttt{mortale}(X)) \tag{2}$$

Riflettiamo sul verso dell'implicazione. Ad esempio, se `rex` è un cane, avremo che `mortale(rex)` ma non `uomo(rex)`

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo}(\texttt{socrate}) \tag{1}$$

Tutti gli uomini sono mortali:

$$\forall X\,(\texttt{uomo}(X) \rightarrow \texttt{mortale}(X)) \tag{2}$$

Riflettiamo sul verso dell'implicazione. Ad esempio, se `rex` è un cane, avremo che `mortale(rex)` ma non `uomo(rex)` Si dovrebbe sapere che chi è cane non è uomo, a dire il vero (CWA).

Knowledge Representation

Il sillogismo

Modelliamo la cosa con la logica del prim'ordine.
Socrate è un uomo:

$$\texttt{uomo}(\texttt{socrate}) \tag{1}$$

Tutti gli uomini sono mortali:

$$\forall X\,(\texttt{uomo}(X) \rightarrow \texttt{mortale}(X)) \tag{2}$$

Riflettiamo sul verso dell'implicazione. Ad esempio, se `rex` è un cane, avremo che `mortale(rex)` ma non `uomo(rex)` Si dovrebbe sapere che chi è cane non è uomo, a dire il vero (CWA).
Inoltre, vale che

$$(1), (2) \models \texttt{mortale}(\texttt{socrate})$$

Knowledge Representation

Il sillogismo

Inoltre, se ora scopriamo che

$$\texttt{uomo}(\texttt{agostino}) \tag{3}$$

allora vale che

$$(1), (2), (3) \models \texttt{mortale}(\texttt{socrate})$$

e

$$(1), (2), (3) \models \texttt{mortale}(\texttt{agostino})$$

Knowledge Representation

Il sillogismo

Inoltre, se ora scopriamo che

$$\texttt{uomo}(\texttt{agostino}) \tag{3}$$

allora vale che

$$(1), (2), (3) \models \texttt{mortale}(\texttt{socrate})$$

e

$$(1), (2), (3) \models \texttt{mortale}(\texttt{agostino})$$

All'aumentare delle premesse, aumentano i teoremi (e quelli vecchi rimangono validi). Siamo in presenza di monotonia.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.
Vogliamo andare dall'altra parte.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.
Vogliamo andare dall'altra parte.
Le sbarre non sono abbassate.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.
Vogliamo andare dall'altra parte.
Le sbarre non sono abbassate.
Alla luce di questa conoscenza sembrerebbe plausibile attraversare

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.
Vogliamo andare dall'altra parte.
Le sbarre non sono abbassate.
Alla luce di questa conoscenza sembrerebbe plausibile attraversare

Se c'è un guasto elettrico, allora le sbarre sono sempre alzate o sempre abbassate.

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.
Vogliamo andare dall'altra parte.
Le sbarre non sono abbassate.
Alla luce di questa conoscenza sembrerebbe plausibile attraversare

Se c'è un guasto elettrico, allora le sbarre sono sempre alzate o sempre abbassate.

E con questa nuova informazione, attraverseste?

Knowledge Representation

La perdita della monotonia

Se si attraversano i binari quando passa il treno non si raggiunge l'altra parte.
Se si attraversano i binari quando non c'è il treno si raggiunge l'altra parte.
Se le sbarre sono abbassate c'è il treno.
Vogliamo andare dall'altra parte.
Le sbarre non sono abbassate.
Alla luce di questa conoscenza sembrerebbe plausibile attraversare

Se c'è un guasto elettrico, allora le sbarre sono sempre alzate o sempre abbassate.

E con questa nuova informazione, attraverseste?

E se sapessimo anche che c'è un guasto elettrico?

Knowledge Representation

La perdita della monotonia

Nel commonsense reasoning si fa uso continuo di asserzioni note come normative statements, ovvero del tipo:

A's are normally *B*'s

che ammettono però delle eccezioni.

Knowledge Representation

La perdita della monotonia: i pinguini

Suppose that a reasoning agent has the following knowledge about birds: birds typically fly and penguins are non flying birds. He also knows that Tweety is a bird. Suppose now that the agent is hired to build a cage for Tweety, and he leaves off the roof on the grounds that he does not whether or nor Tweety can fly. It would be reasonable for us to view this argument as invalid and to refuse the agent's product. This would be not the case if Tweety could not fly for some reason (unknown to the agent), and we refused to pay for the bird cage because had unnecessarily put a roof on it. [McCarthy, 1959]

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.
Ogni pinguino è anormale.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.
Ogni pinguino è anormale.
Metti il tetto alla gabbietta per un uccello che vola.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.
Ogni pinguino è anormale.
Metti il tetto alla gabbietta per un uccello che vola.
tweety è un uccello.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.
Ogni pinguino è anormale.
Metti il tetto alla gabbietta per un uccello che vola.
tweety è un uccello.
Costruisci la gabbietta per tweety

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.
Ogni pinguino è anormale.
Metti il tetto alla gabbietta per un uccello che vola.
tweety è un uccello.
Costruisci la gabbietta per tweety
tweety è un canarino

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola.
Ogni pinguino è un uccello.
Ogni canarino è un uccello.
Ogni pinguino è anormale.
Metti il tetto alla gabbietta per un uccello che vola.
tweety è un uccello.
Costruisci la gabbietta per tweety
tweety è un canarino
Pensate invece a pingu, che è un pinguino.

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola (per $\leftarrow$, pensate alle api)

$$\forall X\,(\texttt{uccello}(X) \wedge \neg \texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{4}$$

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola (per $\leftarrow$, pensate alle api)

$$\forall X\,(\texttt{uccello}(X) \wedge \neg \texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{4}$$

Ogni pinguino è un uccello. Ogni canarino è un uccello.

$$\forall X\,(\texttt{pinguino}(X) \rightarrow \texttt{uccello}(X)) \tag{5}$$

$$\forall X\,(\texttt{canarino}(X) \rightarrow \texttt{uccello}(X)) \tag{6}$$

Knowledge Representation

La perdita della monotonia: i pinguini

Ogni uccello, che non sia anormale, vola (per $\leftarrow$, pensate alle api)

$$\forall X\,(\texttt{uccello}(X) \wedge \neg\texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{4}$$

Ogni pinguino è un uccello. Ogni canarino è un uccello.

$$\forall X\,(\texttt{pinguino}(X) \rightarrow \texttt{uccello}(X)) \tag{5}$$

$$\forall X\,(\texttt{canarino}(X) \rightarrow \texttt{uccello}(X)) \tag{6}$$

Ogni pinguino è anormale, metti il tetto alla gabbietta per un uccello che vola, tweety è un uccello.

$$\forall X\,(\texttt{pinguino}(X) \rightarrow \texttt{ab}(X)) \tag{7}$$

$$\forall X\,(\texttt{uccello}(X) \wedge \texttt{vola}(X) \rightarrow \texttt{metti\_tetto\_gabbia}(X)) \tag{8}$$

$$\texttt{uccello}(\texttt{tweety}) \tag{9}$$

$$(4), (5), (6), (7), (8), \texttt{CWA} \models \texttt{metti\_tetto\_gabbia}(\texttt{tweety})$$

Knowledge Representation

La perdita della monotonia: i pinguini

Se ora so che tweety è un canarino.

$$\texttt{canarino(tweety)} \tag{10}$$

vale ancora che

$$(4), (5), (6), (7), (8), (9), (10)\texttt{CWA} \models \texttt{metti\_tetto\_gabbia(tweety)}$$

Knowledge Representation

La perdita della monotonia: i pinguini

Se ora so che tweety è un canarino.

$$\texttt{canarino(tweety)} \tag{10}$$

vale ancora che

$$(4), (5), (6), (7), (8), (9), (10)_{\texttt{CWA}} \models \texttt{metti\_tetto\_gabbia(tweety)}$$

Ragioniamo con pingu. Se so che pingu è un uccello:

$$\texttt{uccello(pingu)} \tag{11}$$

vale che

$$(4)\text{–}(11), \texttt{CWA} \models \texttt{metti\_tetto\_gabbia(pingu)}$$

Ma se scoprissi ora che

$$\texttt{pinguino(pingu)} \tag{12}$$

$$(4)\text{–}(11), (12)_{\texttt{CWA}} \not\models \texttt{metti\_tetto\_gabbia(pingu)}$$

Knowledge Representation

La ricerca della perdita della monotonia

Abbiamo perso la monotonia.

Knowledge Representation

La ricerca della perdita della monotonia

Abbiamo perso la monotonia.
Dove?

Knowledge Representation

La ricerca della perdita della monotonia

Abbiamo perso la monotonia.
Dove?
Lo investigheremo nelle prossime lezioni. Per ora vi accenno solo che il punto critico è questo:

$$\forall X\,(\texttt{uccello}(X) \wedge \neg \texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{13}$$

Knowledge Representation

La ricerca della perdita della monotonia

Abbiamo perso la monotonia.
Dove?
Lo investigheremo nelle prossime lezioni. Per ora vi accenno solo che il punto critico è questo:

$$\forall X\,(\texttt{uccello}(X) \wedge \neg\texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{13}$$

Che non è una clausola di Horn

Knowledge Representation

La ricerca della perdita della monotonia

Abbiamo perso la monotonia.
Dove?
Lo investigheremo nelle prossime lezioni. Per ora vi accenno solo che il punto critico è questo:

$$\forall X\,(\texttt{uccello}(X) \wedge \neg\texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{13}$$

Che non è una clausola di Horn
Vedremo anche come lo studio dei frammenti di logica predicativa adatti alla KR e NMR ha portato dei risultati molto eleganti della teoria della complessità computazionale.

Knowledge Representation

La ricerca della perdita della monotonia

Abbiamo perso la monotonia.
Dove?
Lo investigheremo nelle prossime lezioni. Per ora vi accenno solo che il punto critico è questo:

$$\forall X\,(\texttt{uccello}(X) \wedge \neg \texttt{ab}(X) \rightarrow \texttt{vola}(X)) \tag{13}$$

Che non è una clausola di Horn
Vedremo anche come lo studio dei frammenti di logica predicativa adatti alla KR e NMR ha portato dei risultati molto eleganti della teoria della complessità computazionale.
Tali risultati sono anche UTILI, nel senso che avremo dei linguaggi per diverse classi di complessità (tra cui NP) corredati di risolutori con efficienza al massimo delle conoscenze attuali.

Deductive reasoning and formal logic

Deductive reasoning argues from the general to a specific instance. The basic idea is that if something is true of a class of things in general, this truth applies to all legitimate members of that class.

All human beings are mortal. Socrates is human.
Therefore, Socrates is mortal.

(syllogism by Aristotle)

Deductive reasoning and formal logic

Deductive reasoning argues from the general to a specific instance. The basic idea is that if something is true of a class of things in general, this truth applies to all legitimate members of that class.

All human beings are mortal. Socrates is human.
Therefore, Socrates is mortal.

(syllogism by Aristotle)

Formal Logic is a formal version of human deductive logic. It provides a formal language with an unambiguous syntax and a precise meaning, and it provides rules for manipulating expressions in a way that respects this meaning.

$$\frac{(\forall X)(\text{human}(X) \rightarrow \text{mortal}(X)) \qquad \text{human}(\text{Socrates})}{\text{mortal}(\text{Socrates})}$$

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible.

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible. *Computational logic* is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and "completeness" of reasoning. However, it is also concerned with efficiency.

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible. *Computational logic* is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and "completeness" of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible. *Computational logic* is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and "completeness" of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible. *Computational logic* is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and "completeness" of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
?- yes
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible. *Computational logic* is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and "completeness" of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
?- yes

?- mortal(Y).
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make automated reasoning using computers possible. *Computational logic* is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and "completeness" of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
?- yes

?- mortal(Y).
?- Y = socrates
```

Computational Logic and Logic Programming

Logic programming in its present form can be traced back to debates in the late 1960s and early 1970s about declarative versus procedural representations of knowledge in Artificial Intelligence.
Techniques (i.e., resolution) come from Automated Theorem Proving
Advocates of declarative representations were notably working at Stanford, associated with John McCarthy and in Edinburgh, with John Alan Robinson, Pat Hayes, and Robert Kowalski. Advocates of procedural representations were mainly centered at MIT, under the leadership of Marvin Minsky and Seymour Papert.

Logic Programming

www.logicprogramming.org www.programmazionelogica.it

Hayes and Kowalski in Edinburgh tried to reconcile the logic-based declarative approach to knowledge representation with Planner's procedural approach. In 1971 Kowalski developed the SLD resolution and, by collaborating with Colmerauer in Marseille, developed these ideas in the design and implementation of the programming language *Prolog* (1972).

In 1976 Alberto Martelli and Ugo Montanari developed their efficient unification algorithm. In 1983 D.H.Warren designed and implemented the WAM.

The Italian Association for Logic Programming (GULP) was founded to promote Logic Programming in 1985. The (international) Association for Logic Programming (ALP) was founded to promote Logic Programming in 1986.

LP and Teaching

In the last couple of years, there has been some discussion in the logic programming community about the use of Prolog as a teaching language and, in particular, as a first language. A remarkable characteristic of this debate is that, while most logic programmers argue that Prolog should be taught *somewhere* in the undergraduate curriculum, almost nobody is arguing that it should be the *first* teaching language! Usually, the reason given for avoiding Prolog as the first language is that its non-logical features make it too complicated for beginning programmers. I think this is true, but I would also add as equally serious flaws its lack of a type system and a module system. This situation is surely a serious indictment of the field of logic programming. In spite of its claims of declarativeness, until recently at least, there hasn't been a single logic programming language sufficiently declarative (and hence simple) to be used successfully as a first teaching language.

[J.W. Lloyd. Practical Advantages of Declarative Programming. GULP-PRODE'94, Peñiscola]

Aggiungerei pure le limitate capacità espressive di Prolog (pur Turing completo) di esprimere la negazione.

LP and Teaching

In the last couple of years, there has been some discussion in the logic programming community about the use of Prolog as a teaching language and, in particular, as a first language. A remarkable characteristic of this debate is that, while most logic programmers argue that Prolog should be taught *somewhere* in the undergraduate curriculum, almost nobody is arguing that it should be the *first* teaching language! Usually, the reason given for avoiding Prolog as the first language is that its non-logical features make it too complicated for beginning programmers. I think this is true, but I would also add as equally serious flaws its lack of a type system and a module system. This situation is surely a serious indictment of the field of logic programming. In spite of its claims of declarativeness, until recently at least, there hasn't been a single logic programming language sufficiently declarative (and hence simple) to be used successfully as a first teaching language.

[J.W. Lloyd. Practical Advantages of Declarative Programming. GULP-PRODE'94, Peñiscola]

Aggiungerei pure le limitate capacità espressive di Prolog (pur Turing completo) di esprimere la negazione.

Superare il problema

Ci sono alcuni momenti chiave che portano al linguaggio che useremo nel corso.

- La nozione di *stable model* di Gelfond e Lifschitz (1988)
- La conseguente nozione di *Answer Set Programming* di Marek e Truszczynski (1998) privo di predicati extra-logici e indipendente dall'ordine.
- L'implementazione del primo ASP solver *Smodels* (Simons-Niemelä, 1997)
- L'implementazione e l'evoluzione del conflict-driven ASP solver *clasp/clingo* (dal 2009) e le competizioni per gli ASP solvers.

E dunque?

Useremo ASP anzichè Prolog!

Impareremo a codificare problemi in Answer Set Programming.

Come Software vi basta un editor di testi (qualunque: ne avete già almeno uno nel vostro PC) per creare un file `nomefile.lp`

E il tool libero multipiattaforma CLINGO che si scarica da `http://potassco.sourceforge.net/`

Si esegue da linea di comando

`clingo <A> nomefile.lp <B>`

Ove al posto di `<A>` ci possono essere degli assegnamenti di valori (es `-c n=5`) e al posto di `<B>` il numero di soluzioni che cerchiamo (0 vuol dire tutte).