Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 13- Revisão geral do curso

Prof. José Reinaldo Silva

reinaldo@usp.br

PMR 3510 - Inteligencia Artificial

Módulo 1

1.1 Introdução, Agentes Inteligentes, A sociedade da Mente (Minsky), Aplicações da IA em automação (Aula 1);

1.2 Resolução de Problemas, Métodos clássicos de busca (Aula 2 e 3);

1.3 Busca Informada (Aulas 4 e 5)

Módulo 2

2.1 Sistemas baseados em conhecimento, representação de conhecimento (Aula 6);

2.2 Introdução à Lógica de 1a. ordem (Aula 7 e 8);

1.3 Métodos de inferência, exercícios (Aulas 9 e 10);

PMR 3510 - Inteligencia Artificial

Módulo 3

3.1 Planejamento inteligente (Aula 11 e 12);

3.2 Problemas e casos práticos de planejamento, STRIPS (Aula 14 e 15);

3.3 Ações e planejamento (Aulas 16)

Módulo 4: Aplicações

Avaliação:

Trabalho em grupo (2)(25%); prova no final do curso (50%)

Acompanhamento do curso e aprendizado paralelo de programação Prolog via e-disciplinas e SWI-online

Livro texto: Artificial Intelligence: a new synthesis, Nils Nilsson (2009); Artificial Intelligence: a modern approach, Stuart Russel, Peter Norvig (3rd. ed.) (2009) - site AIMA, UC Berkeley.

Apoio: Programming in Prolog, W. F. Clocksin, C. S. Mellish (5a. ed.) (2003).

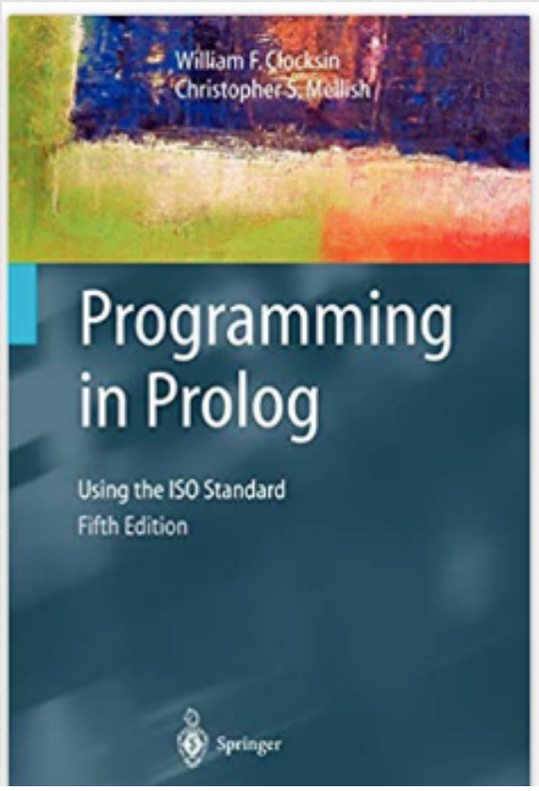
PMR 3510 - Inteligencia Artificial

Livro texto: Artificial Inteligence: a new synthesis, Nils Nilsson (2009);
Artificial Intelligence: a modern approach, Stuart Russel, Peter Norvig (3rd. ed.) (2009) - site AIMA, UC Berkeley.
Apoio: Programming in Prolog, W. F. Clocksin, C. S. Mellish (5a. ed.) (2003).

Apoio Web:
Site e-disciplinas (PMR 3510): o acompanhamento da disciplina, material de apoio, exercícios, trabalhos, tópicos de programação, serão repassados e recebidos somente através deste site.

Programação em Prolog: www.learnprolognow.org

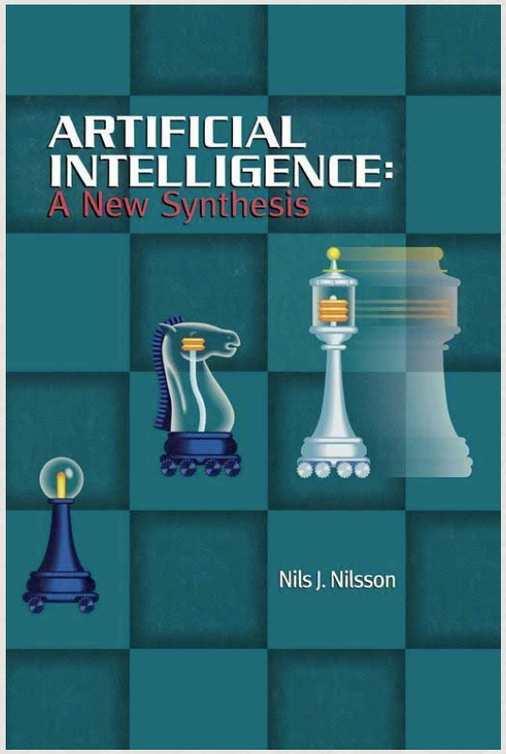
Sistema para programação: swish.swi-prolog.org (SWI Prolog)

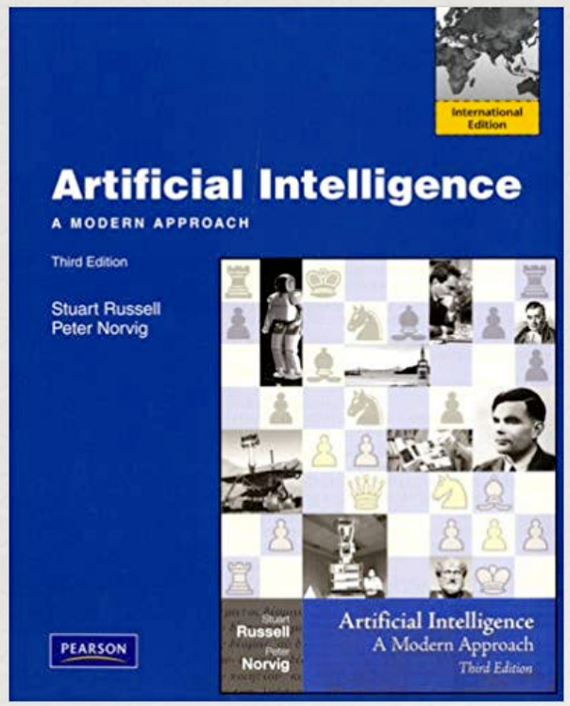


Programming in PROLOG: Using the ISO Standard

25 jul 2003, 5th. Edition

William F. Clocksin and Christopher S. Hellish





Introduction

- a máquina capaz de emular um jogador de xadrez

A "máquina inteligente" (automato) conhecida como "o Turco", foi criada na Austria-Hungria em 1770 por Wolfgan Von Kepelen, para agradar a duques de Habsburgo, Maria Theresa. O Turco ficou famoso por vencer vários jogador habilidosos no período 1770-1789.

Introduction

O tema do processamento computacional de processos racionais continuou na pauta de vários pesquisadores e engenheiros até que em 1956, John MacCarthy propôs um evento com os principais interessados no Darthmouth College, financiado pelo Rockfeller Foundation. Organizadores foram, além de MacCarthy, Marvin Minsky (Harvard), Nathaniel Rochester (IBM) e Claude Shannon (Bell Labs). O termo "Inteligência Artificial" usado pela primeira vez com o significado atual neste evento.

Objetos e relacionamentos em Prolog

Cloksin, W. S., Mellish, C. S.; Programming in Prolog, Springer, 2003.

Portanto a primeira forma de representação de "conhecimento" é através de fatos e admitimos que os fatos inseridos em um programa lógico são verdadeiros. A omissão de um fato significa portanto a sua negação. Podemos ter uma atribuição direta a um objeto sem relacioná-lo com outro objeto: por exemplo, podemos dizer que "Maria é corintiana", o que simplesmente atribui uma propriedade ao objeto "Maria".

obs: Prolog não é uma linguagem orientada a objetos e o termo aqui tem somente parte do significado hoje dado a "objetos".

Fatos

```
sobre(livro, mesa)
pertence(livro, José)
```

```
corintiana(Maria)
```

Objetos e relacionamentos em Prolog

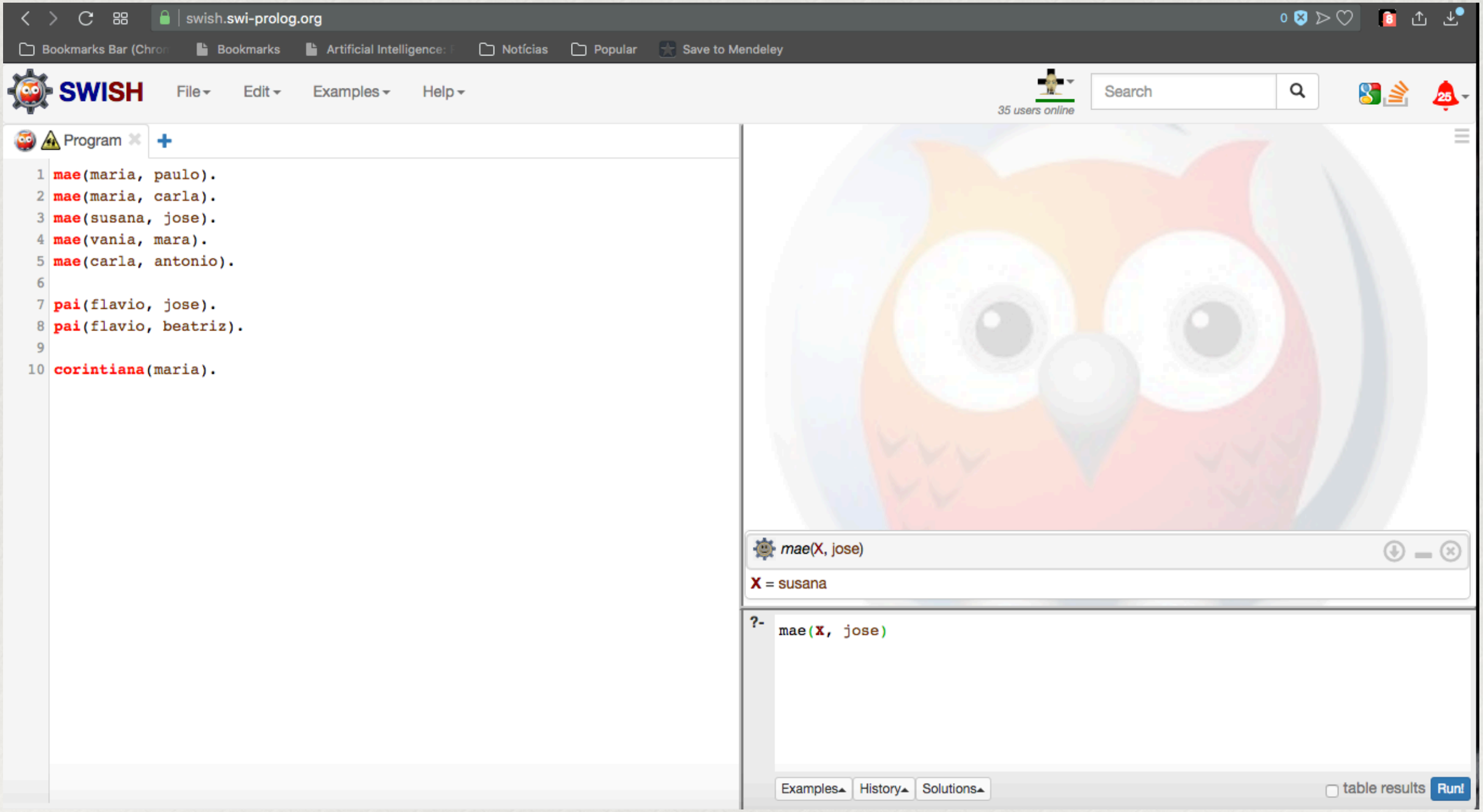
Cloksin, W. S., Mellish, C. S.; Programming in Prolog, Springer, 2003.

Podemos agora escrever nossa primeiras linhas de um "programa inteligente" usando como tema a relação entre elementos de uma familia hipotética.

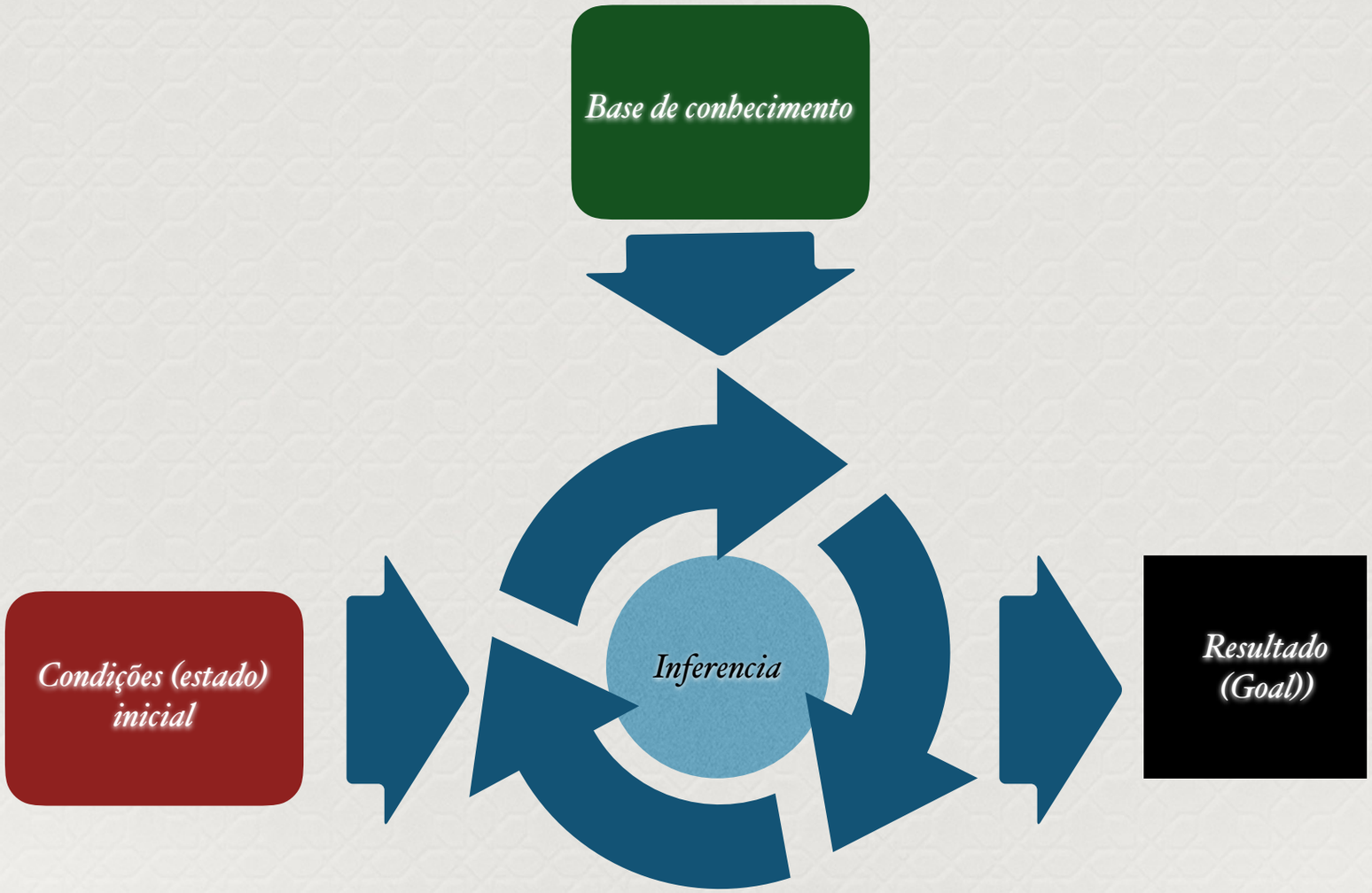
Base de conhecimento

```
mae(maria, paulo)
mae(maria, carla)
mae(susana, jose)
mae(vania, mara)
mae(carla, antonio)

pai(flavio, jose)
pai(flavio, beatriz)

corintiana(maria)
```

swish.swi-prolog.org



Em uma primeira abordagem, gostaríamos de ter "agentes inteligentes" capazes de "resolver problemas". O que significa isso?



Como "resolver problemas automaticamente"

Podemos utilizar duas grande abordagens para resolver problemas automaticamente:

Problema

1. achar uma abordagem geral que leva do estado inicial ao estado final;
2. testar esta abordagem em alguns casos (sem levar em conta o tempo para chegar à solução);
3. checar se a abordagem é completa, isto é, resolve todos os casos ou há casos especiais onde o problema "não converge";
4. preparar a implementação do revolvedor (estrutura de dados e base de conhecimento, regras de dedução);

Solução

Achar um método geral de resolução de problemas

Nesta aula vamos discutir a primeira abordagem...

... antes porém vamos deixar claro a hierarquia de "métodos" para resolução de problemas que vamos abordar nesta e nas próximas aulas:

Paradigma de resolução: estado/transição

método geral de resolução (STRIPS)

Solução específica para os problemas

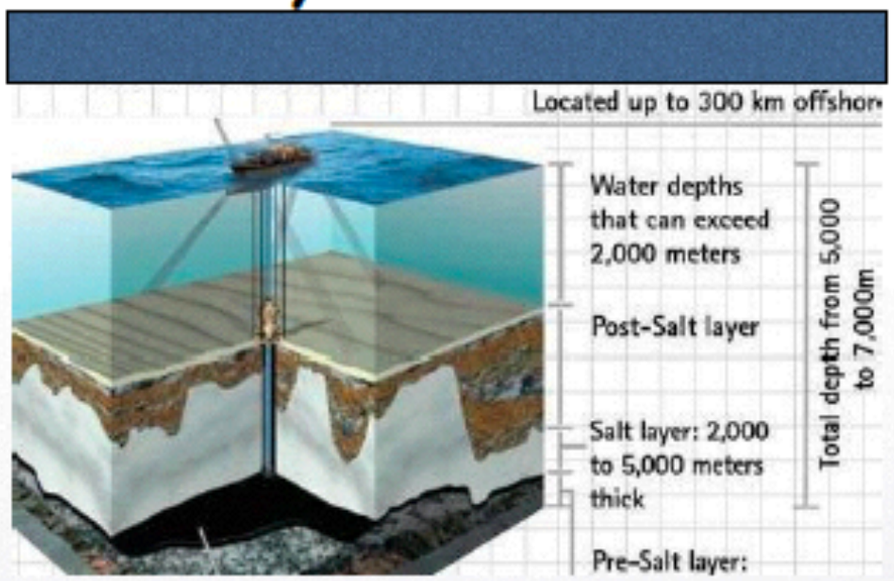
abstração

Exemplos: ii) ROADEF 2005

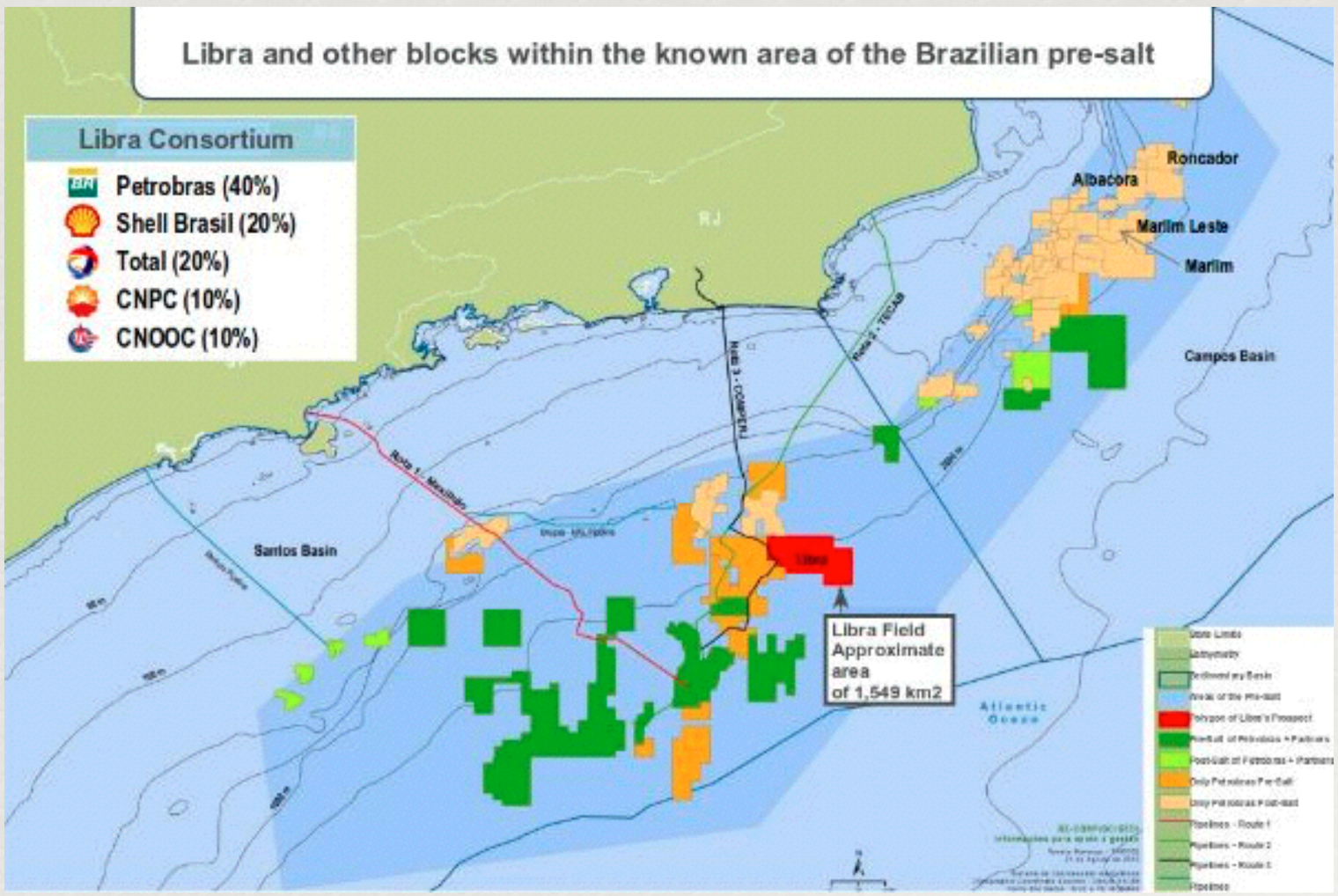
RENAULT

Exemplos: ii) SIPROV

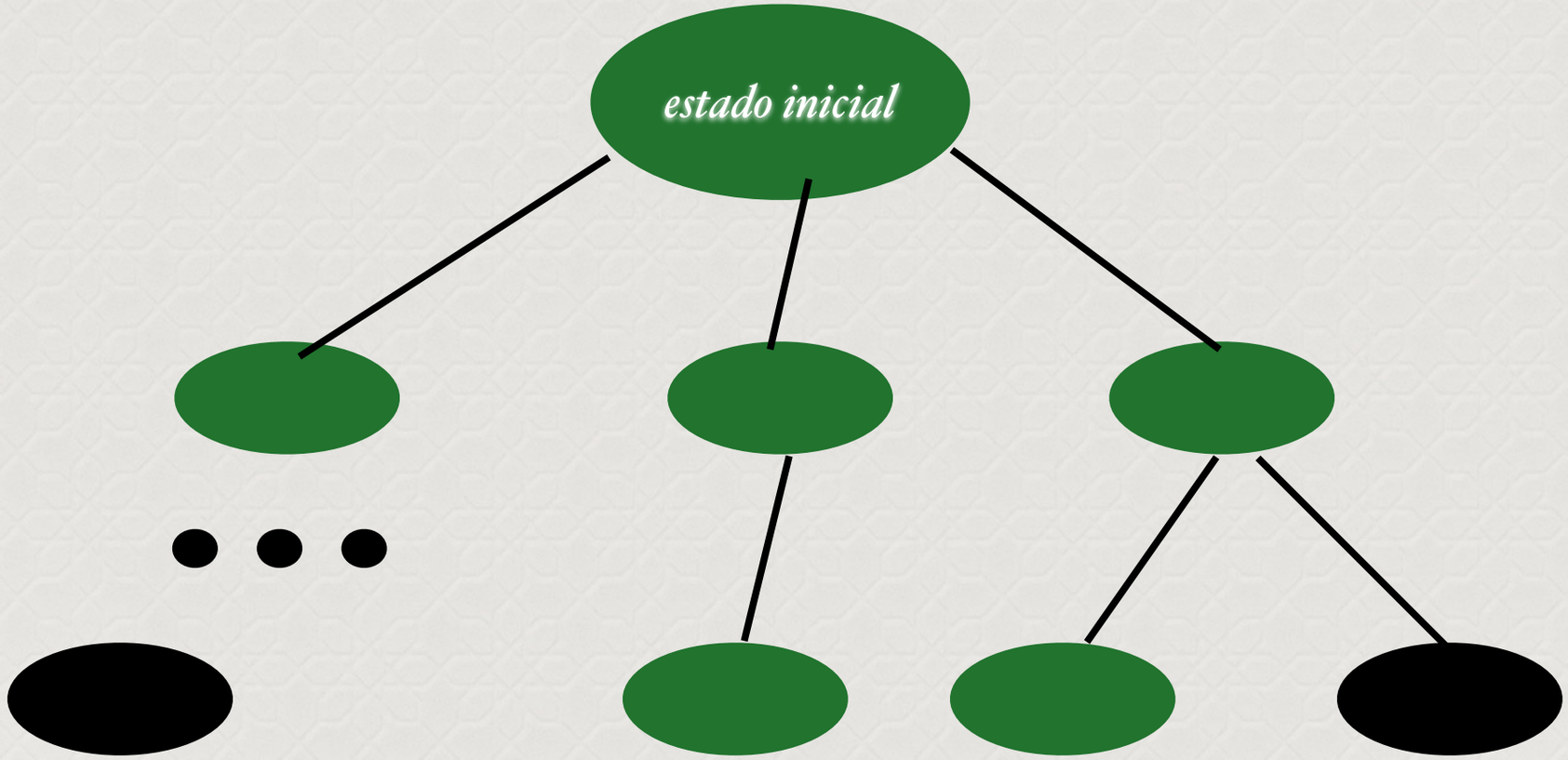
Petroleum exploitation in the pre-salt layer



Exemplos: ii) SIPROV



A estrutura do revolvedor de problemas é uma árvore, onde a raiz é o estado inicial.



Programming Logic - PROLOG

O prolog nasceu de um projeto conjunto entre Alain Colmerauer da Univ. de Aix-Marseille e Robert Kowalski da Univ. of Edinburgh. O lançamento oficial é de 1972, e desde então participou de projetos importantes como o Esprit da Europa (https://ec.europa.eu/inea/en/horizon-2020/projects/h2020-transport/green-vehicles/esprit) e o projeto japonês ICOT Fifth Gen. Computer (http://museum.ipsj.or.jp/en/computer/other/0002.html) dos anos 80.

Alain Colmerauer
Aix-Marseille

Robert Kowalski
Univ. of Edinburgh

Os fatos são na verdade relacionamentos entre objetos, e, convencionalmente está vinculado ao primeiro elemento, como em

mae(maria, pedro) - Maria é mãe de Pedro

Fato

Interpretação

valuable(gold). Gold is valuable.

female(jane). Jane is female.

owns(jane, gold). Jane owns gold.

father(john, mary). John is the father of Mary.

gives(john, book, mary). John gives the book to Mary.

Predicado

Argumentos

Aridade (arity)= no. de argumentos

valuable/1, female/1, owns/2, father/2, gives/3

Instanciação e busca

?- parent_child(trude, X).

parent_child(trude, X)

father_child(trude, X) NO

parent_child(trude, X)

mother_child(trude, X)

X=sally

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y)       :-
parent_child(Z, X),
parent_child(Z, Y).

parent_child(X, Y) :-
father_child(X, Y).
parent_child(X, Y) :-
mother_child(X, Y).
```

Portanto, relacionar problemas e soluções não é trivial

Uma maneira de buscar soluções para o problema, usando a hipótese do "mundo fechado" é trabalhar com uma base de conhecimento baseado em fatos e regras e tentar inferir, dada uma pergunta, qual a "resposta correta", como vimos fazendo nos exemplos práticos anteriores.

Modus Ponens

Modus Tollens

Modus Ponens

A base do Modus Ponens é a expressão condicional A implica B ou A → B. Relações causa-efeito podem então ser representadas, tais como:

i) se o interruptor da sala for desligado as lâmpadas se apagarão.
ii) um aluno desligou o interruptor;
iii) as lâmpadas estão agora apagadas

```
status_key(ligado).
status_key(desligado).
agente_op(aluno).
agente_op(funcionario).
agente_op(professor).
interruptor(X):- status_key(X).
desliga_interruptor(X):- agente_op(X).
lampadas(apagadas):- X=ligado, interruptor(X), desliga_interruptor(aluno).
```

	P	Q	$P \rightarrow Q$
1	V	V	V
2	V	F	F
3	F	V	V
4	F	F	V

A → B também pode ser escrito como if A then B onde A é chamado antecedente e B o consequente

((A → B) AND B) → A

lampadas(apagadas) :- ligado(interruptor), desliga_interruptor(X).

if :

swish.swi-prolog.org

Bookmarks Bar (Chron | Bookmarks | Artificial Intelligence: F | Notícias | Popular | Save to Mendeley

SWISH File Edit Examples Help

83 users online

Search

Program

```
1  status_key(ligado).
2  status_key(desligado).
3
4  agente_op(aluno).
5  agente_op(funcionario).
6  agente_op(professor).
7
8  interruptor(X):- status_key(X).
9
10 desliga_interruptor(X):- agente_op(X).
11
12 lampadas(apagadas):- X=ligado, interruptor(X), desliga_interruptor(aluno).
```

lampadas(apagadas).

true 1

?-

```
lampadas(apagadas).
```

Examples History Solutions table results Run!

Modus Tollens

A base do Modus Tollens também é a expressão condicional A implica B ou A → B.

Relações causa-efeito podem então ser representadas, tais como:

i) se o interruptor da sala for desligado as lâmpadas se apagarão.

ii) ninguém desligou o interruptor;

iii) as lâmpadas estão acesas

```
status_key(ligado).
status_key(desligado).
agente_op(aluno).
agente_op(funcionario).
agente_op(professor).
interruptor(X):- status_key(X).
desliga_interruptor(X):- agente_op(X).
not(lampadas(apagadas)):- X=ligado, interruptor(X), not(desliga_interruptor(aluno)).
```

	P	Q	$P \rightarrow Q$
1	V	V	V
2	V	F	F
3	F	V	V
4	F	F	V

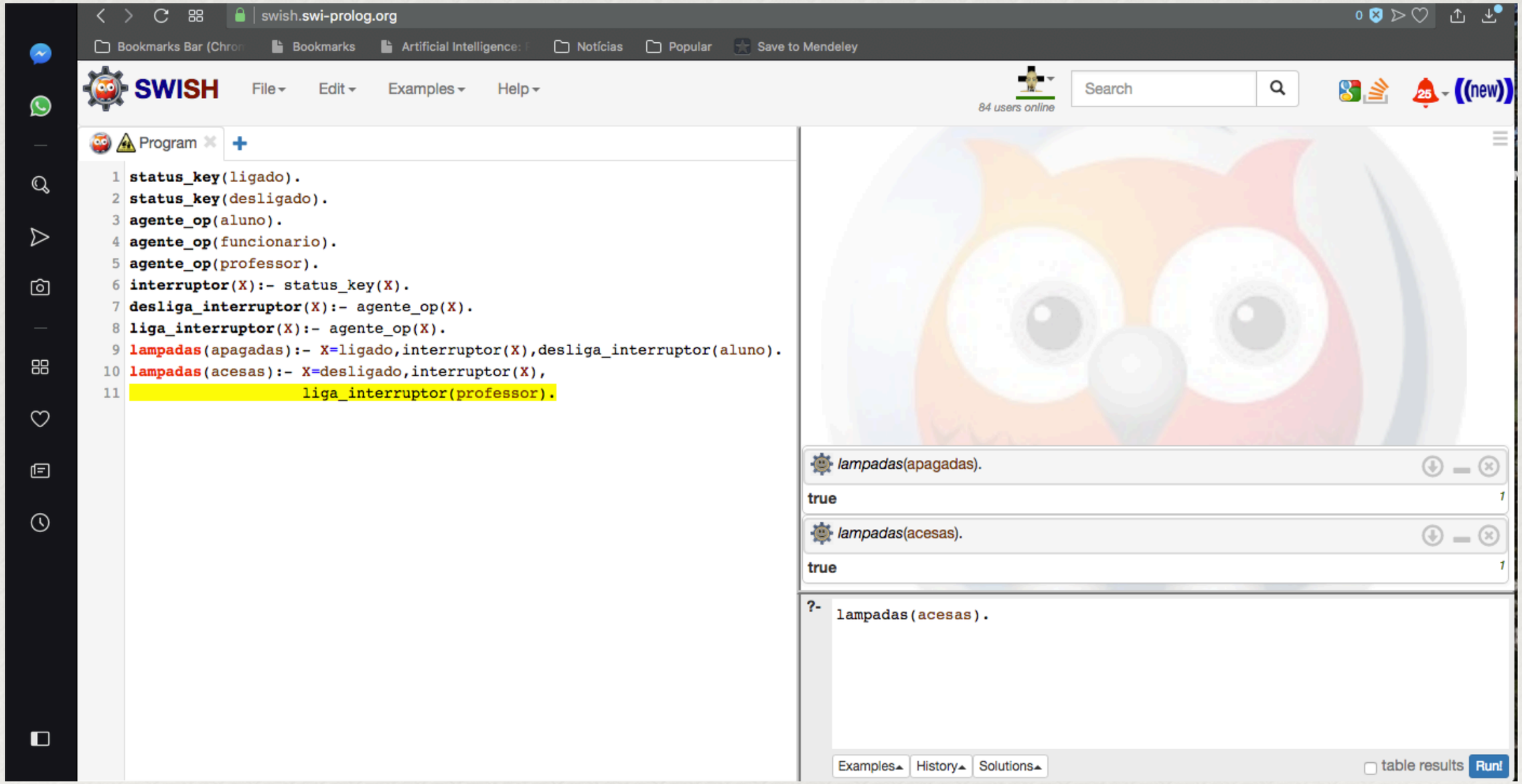
((A → B) AND not(B)) → not(A)

not(lampadas(apagadas)) :- ligado(interruptor),not(desliga_interruptor(X)).

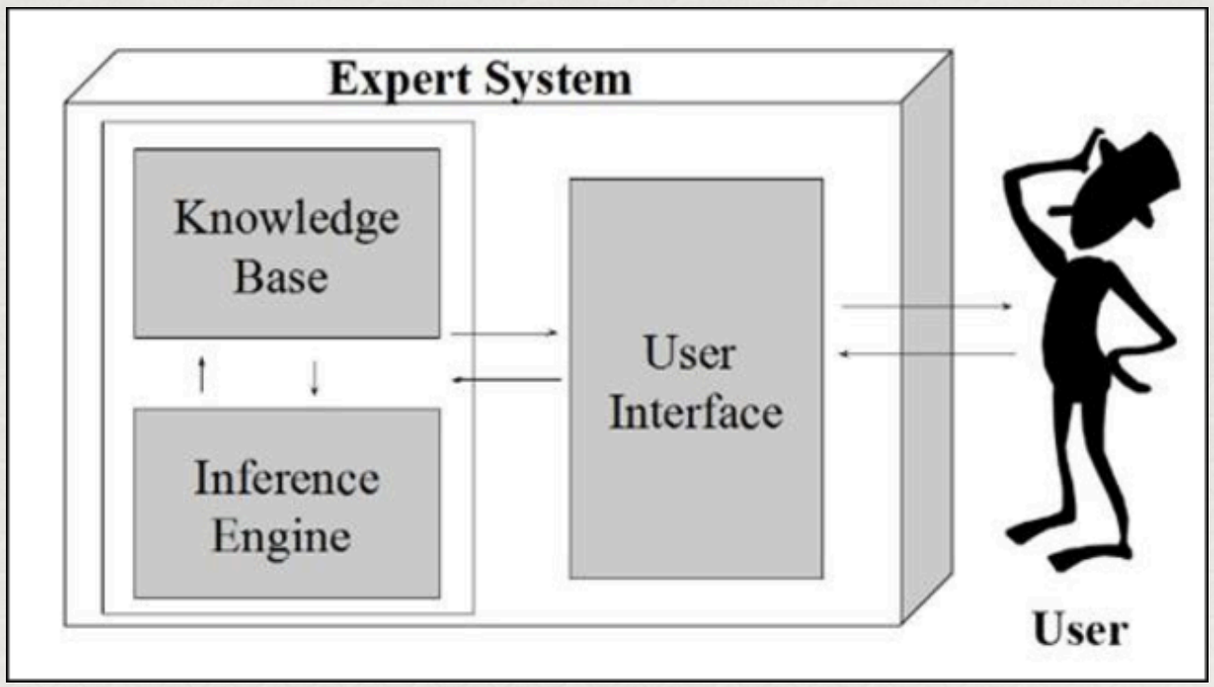
Isso significa que precisaremos de uma representação de conhecimento que vá além de fatos e regras diretas. Precisaremos também definir predicados, funções e operadores, sua "aridade", e conjunto-verdade. Voltaremos a este assunto na aula que vem.

Será que este programa está correto... e completo?

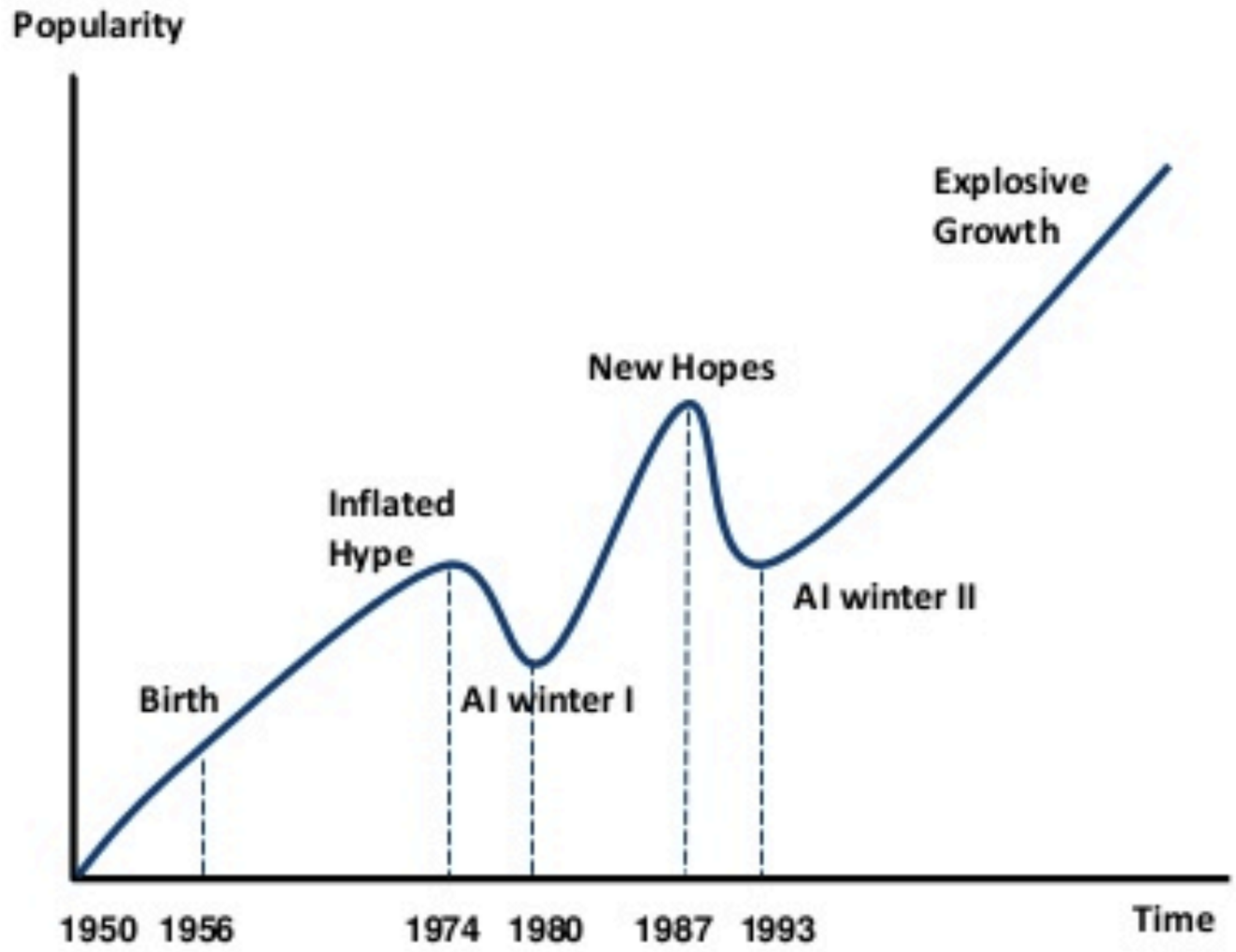
	P	Q	$P \rightarrow Q$
1	V	V	V
2	V	F	F
3	F	V	V
4	F	F	V


swish.swi-prolog.org
Bookmarks Bar (Chron
Bookmarks
Artificial Intelligence:
Notícias
Popular
Save to Mendeley
SWISH
File
Edit
Examples
Help
84 users online
Search
Program
1 status_key(ligado).
2 status_key(desligado).
3 agente_op(aluno).
4 agente_op(funcionario).
5 agente_op(professor).
6 interruptor(X):- status_key(X).
7 desliga_interruptor(X):- agente_op(X).
8 liga_interruptor(X):- agente_op(X).
9 lampadas(apagadas):- X=ligado,interruptor(X),desliga_interruptor(aluno).
10 lampadas(acesas):- X=desligado,interruptor(X),
11 liga_interruptor(professor).
lampadas(apagadas).
true
1
lampadas(acesas).
true
1
?-
lampadas(acesas).
Examples
History
Solutions
table results
Run!

Edward Feigenbaum

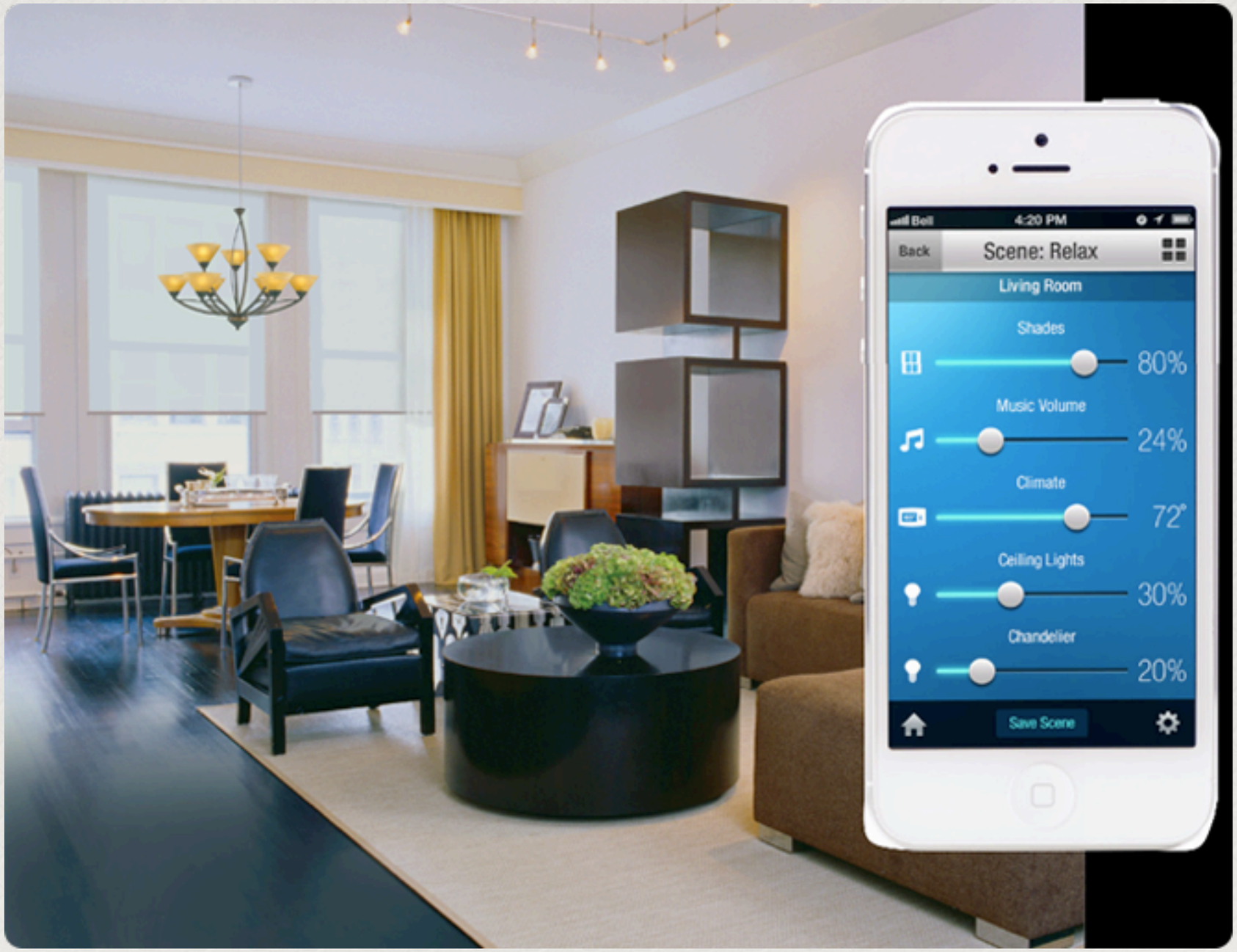


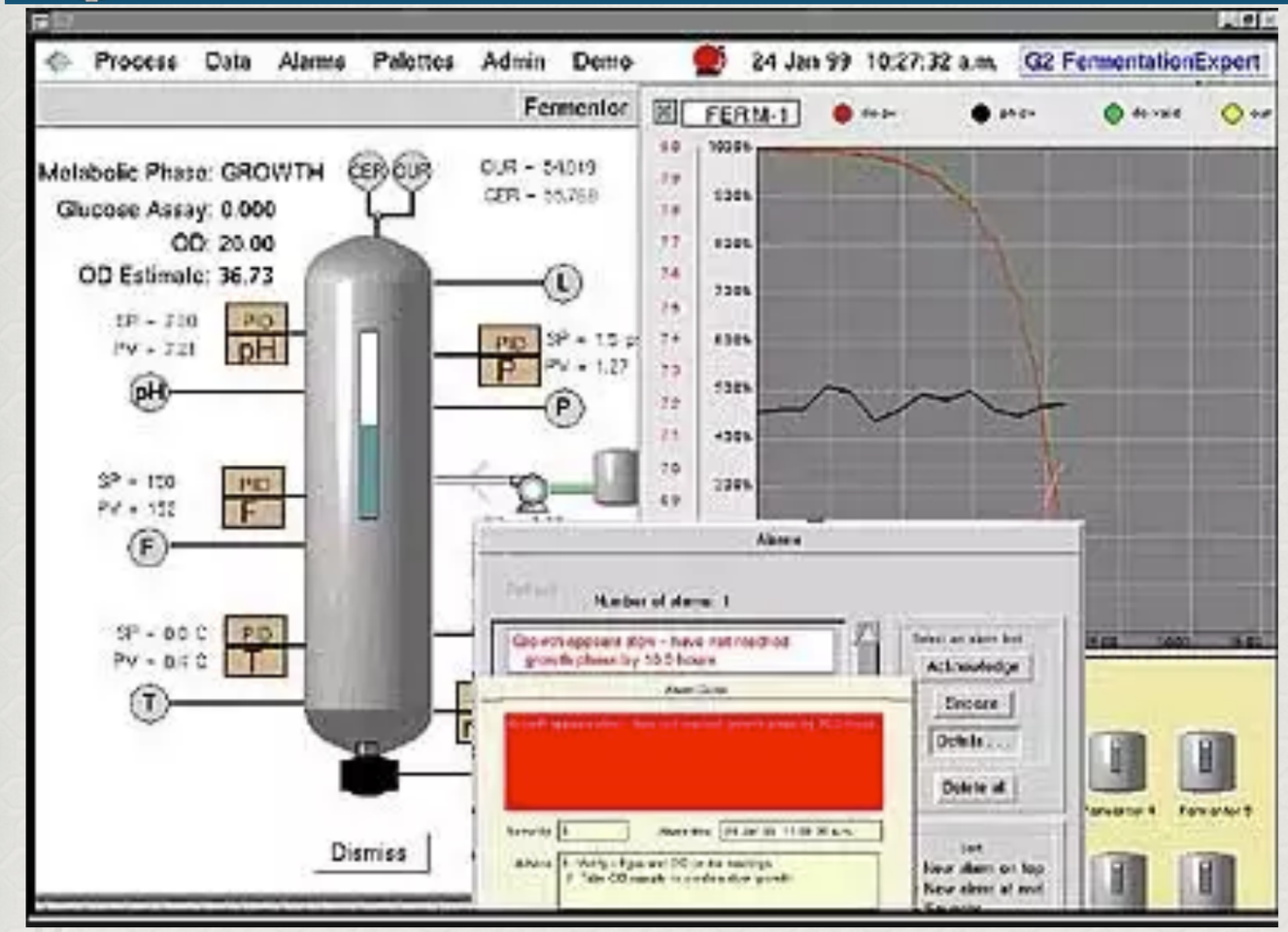
AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"…



Timeline of AI Development
▪ **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
▪ **1970s**: AI winter I
▪ **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
▪ **1990s**: AI winter II
▪ **1997**: Deep Blue beats Gary Kasparov
▪ **2006**: University of Toronto develops Deep Learning
▪ **2011**: IBM's Watson won Jeopardy
▪ **2016**: Go software based on Deep Learning beats world's champions

Source: Literature review, Gao Feng analysis


Bell
4:20 PM
Back
Scene: Relax
Living Room
Shades
80%
Music Volume
24%
Climate
72°
Ceiling Lights
30%
Chandelier
20%
Save Scene


Process
Data
Alarms
Palettes
Admin
Demo
24 Jan 99 10:27:32 a.m.
G2 FermentationExpert
Fermentor
FERM-1
Metabolic Phase: GROWTH
Glucose Assay: 0.000
OD: 20.00
OD Estimate: 36.73
Alarms
Acknowledge
Encore
Delete...
Delete all
Dismiss

gensym

STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving[1]

Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

Presented at the 2nd IJCAI, Imperial College, London, England, September 1–3, 1971.

ABSTRACT

We describe a new problem solver called STRIPS that attempts to find a sequence of operators in a space of world models to transform a given initial world model into a model in which a given goal formula can be proven to be true. STRIPS represents a world model as an arbitrary collection of first-order predicate calculus formulas and is designed to work with models consisting of large numbers of formulas. It employs a resolution theorem prover to answer questions of particular models and uses means-ends analysis to guide it to the desired goal-satisfying model.

DESCRIPTIVE TERMS

Problem solving, theorem proving, robot planning, heuristic search.

1. Introduction

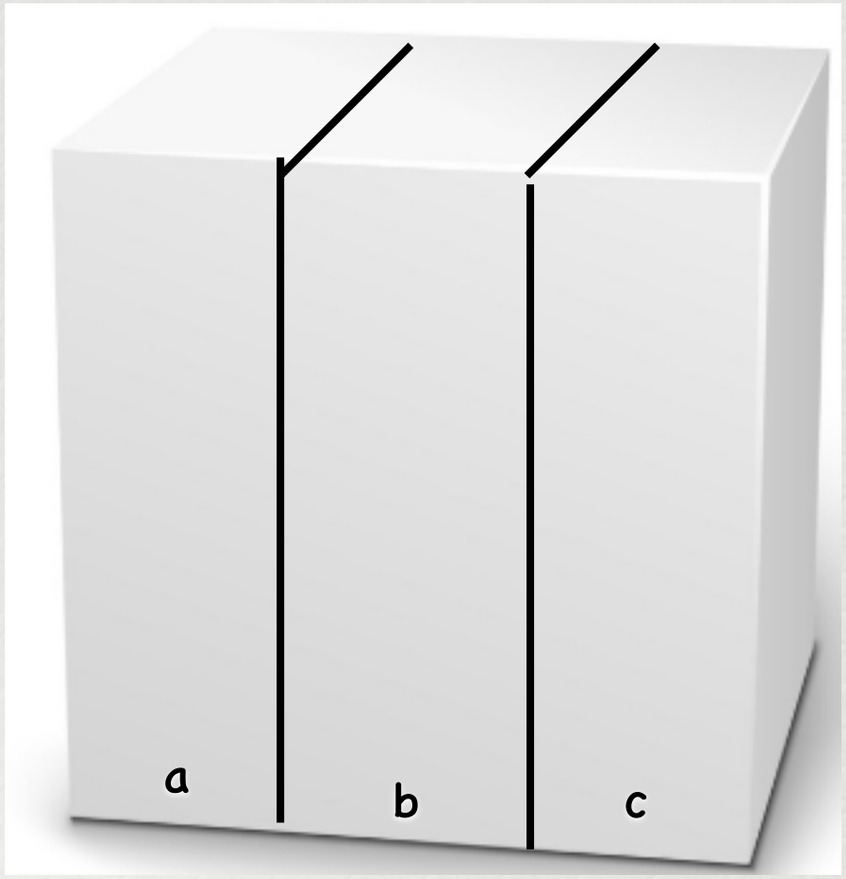
This paper describes a new problem-solving program called STRIPS (*ST*anford *R*esearch *I*nstitute *P*roblem *S*olver). An initial version of the program has been implemented in LISP on a PDP-10 and is being used in conjunction with robot research at SRI. STRIPS is a member of the class of problem solvers that search a space of "world models" to find one in which a given goal is achieved. For any world model, we assume that there exists a set

[1] The research reported herein was sponsored by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS12-2221.

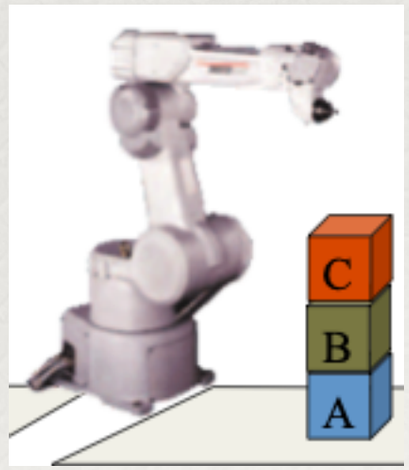
Artificial Intelligence 2 (1971), 189–208

STRIPS Basic Concepts

O domínio é uma descrição dos elementos que devem ser manipulados e suas restrições: vamos tomar como exemplo o "mundo de blocos"...



O mundo é composto de três colunas onde é possível acumular até 3 "blocos". Existem no total 3 blocos plenamente identificados. Existe também um robô que pode manipular um bloco de cada vez e só pode pegar o bloco no topo da pilha.



STRIPS Basic Concepts

O problema de planejamento (planning problem) é dado pela definição de estado do problema e pelas ações e operadores que mudam este estado ...

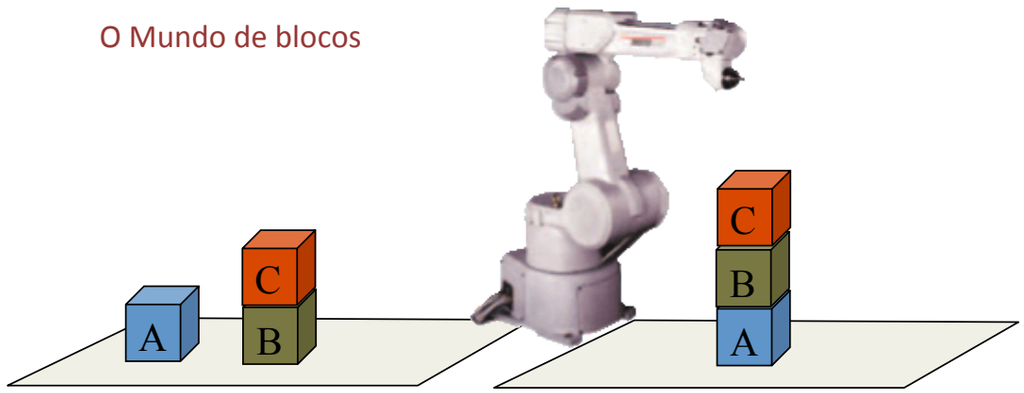
livre(A)
sobre(A,mesa)
sobre(C, B)
sobre (B, mesa)
livre(C)

Operadores:
pick(a), drop(b), move(a, table)...

IA Planning: o STRIPS

O sistema STRIPS é a estratégia de resolução de problemas mais usada em planning. Note-se que é uma estratégia baseada no método estado-transição e por isso é passível de ser analisada em Redes de Petri. O problema modelo mais conhecido resolvido com o sistema STRIPS é o problema do mundo de blocos.

O Mundo de blocos



O método do STRIPS é baseado em estado-transição e consiste em:

- Modelar o domínio onde o plano será aplicado: o que implica em definir claramente estados e restrições que devem nortear a aplicação de ações e operadores.

- Definir o problema de planejamento, isto é, as ações admissíveis, pre e pós-condições para a aplicação destas ações;

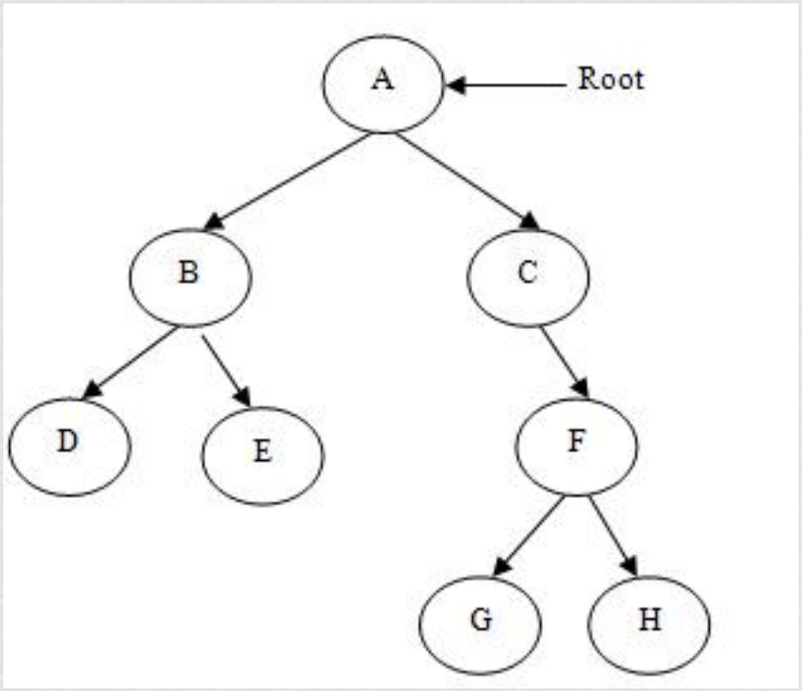
- Definir um processo de busca, direta, heurística ou definir um modelo de problema para ser resolvido (este último caso está fora do escopo desta disciplina).

Portanto o método, que pode ser aplicado em vários problemas reais, depende do processo de busca e pode ser resolvido por um algoritmo diretamente ligado ao domínio e ao problema de planejamento ou por sistemas mais genéricos (que resolveriam qualquer problema) chamado de **planejadores**.

No caso do mundo de blocos usaremos a primeira abordagem.

Usando árvores como base para a solução

Uma opção muito interessante é modelar o espaço de estados na forma de uma árvore

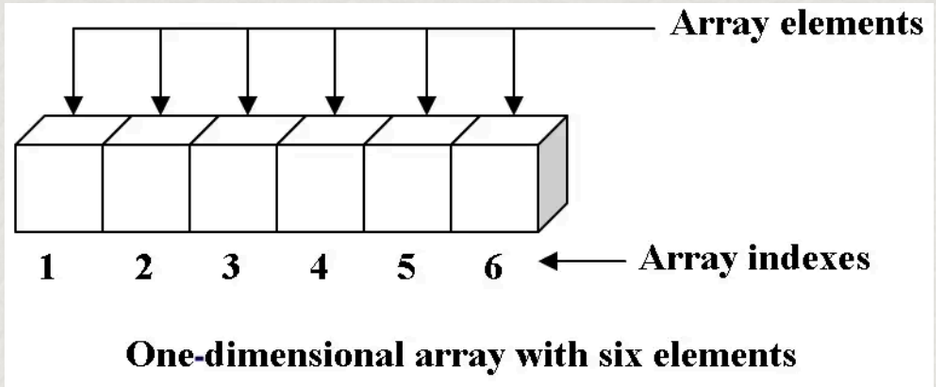


Estruturas e Listas

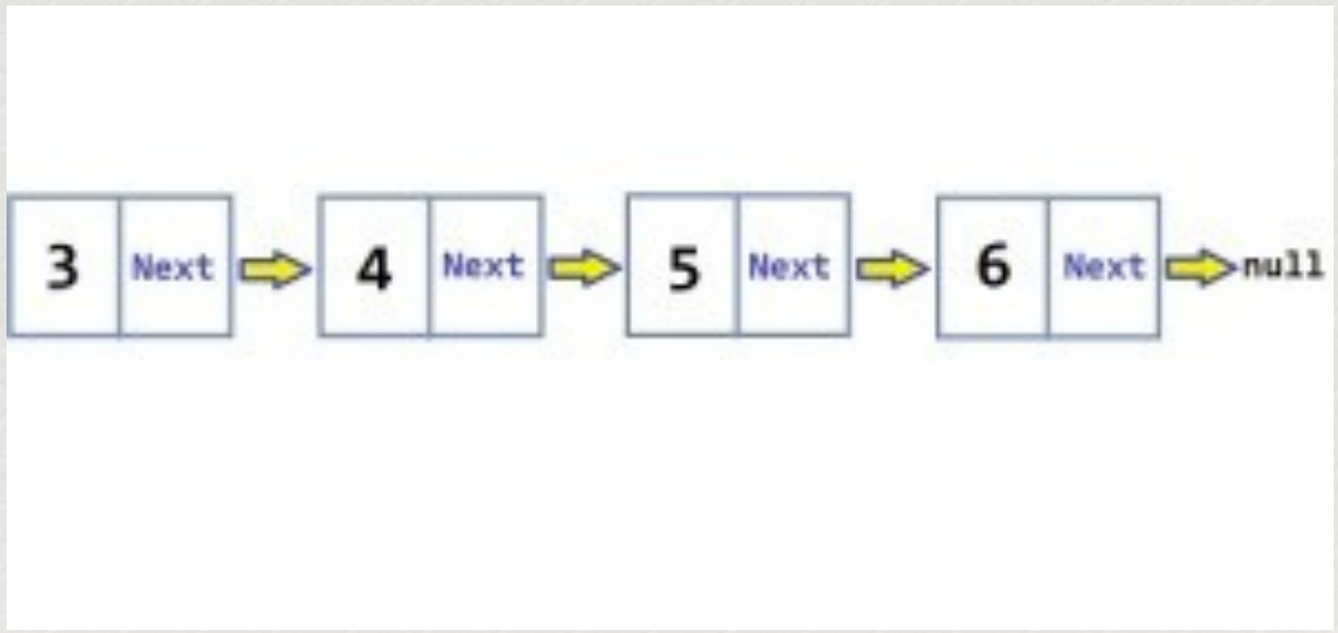
Uma estrutura básica em programação (lógica) é a lista. Conceitualmente uma lista é uma sequencia de registros homogênea. Normalmente estas listas são indexadas (arrays) ou direcionadas por ponteiros (listas ligadas).

No caso da programação em Prolog a lista segue seu conceito mais básico, isto é, composta de uma cabeça (head) que é o primeiro elemento da lista, e de um corpo (body) que é a sub-lista restante. Estes dois elementos básicos são suficientes para dar suporte a todas os algoritmos de manipulação de estruturas como árvores, vetores, etc.

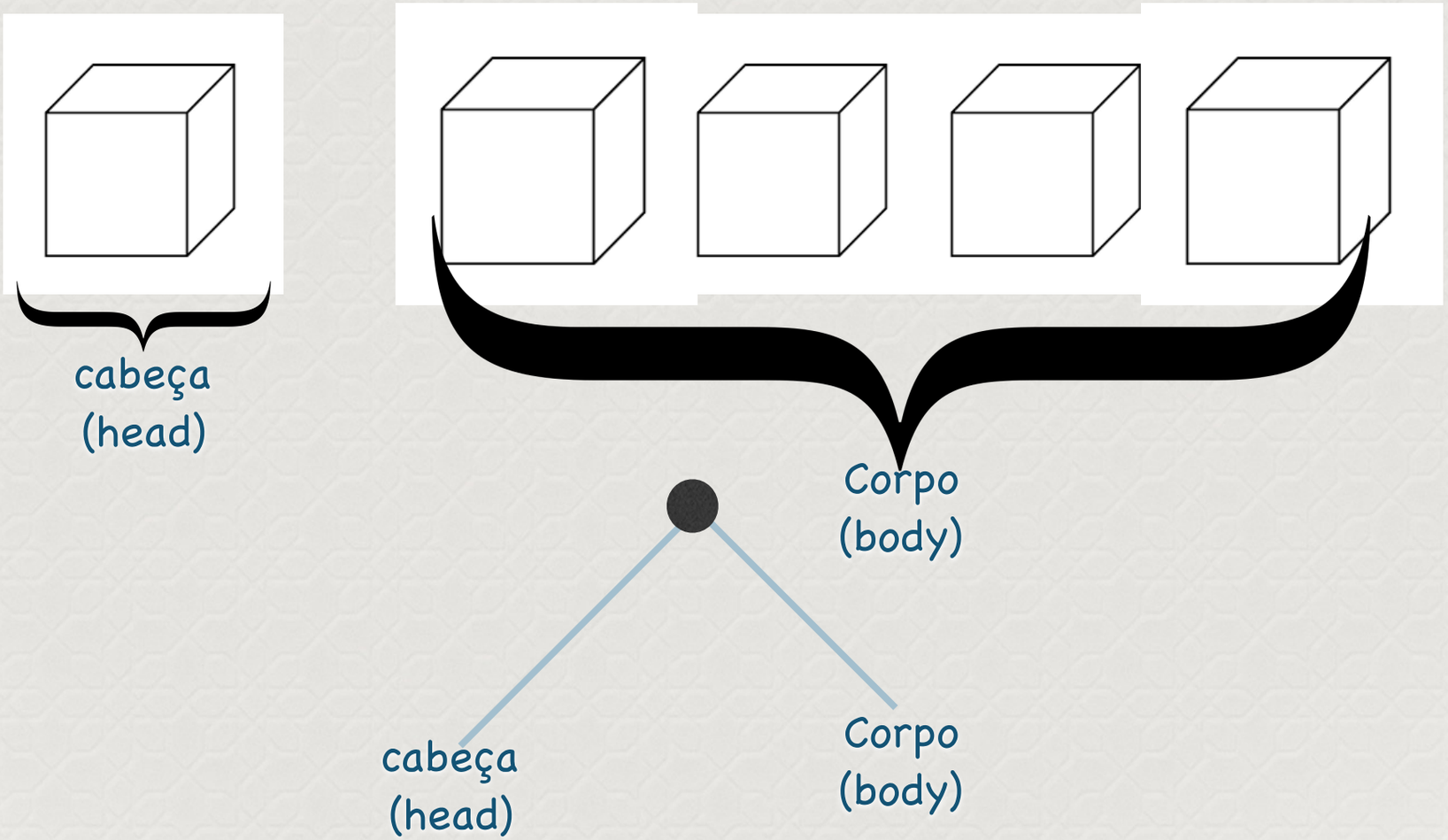
Instancias da estrutura abstrata lista



One-dimensional array with six elements



Estrutura abstrata de lista



```
append([], L, L).
append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
```

False

append([guilherme_russo, natan, natalia],[fernando, marcos, sverker], Y).

X=guilherme_russo
L1=[natan, natalia]
L2=[fernando, marcos, sverker]
Y= [guilherme_russo | L3]

append([natan, natalia],[fernando, marcos, sverker], L3).

X=natan
L1=[natalia]
L2=[fernando, marcos, sverker]
Y= [natan | L3]

O predicado (interno) "member" checa se um dado elemento pertence à lista...

```
member(X,[X|T]).
member(X,[H|T]) :- member(X,T).
```

Algoritmos de busca

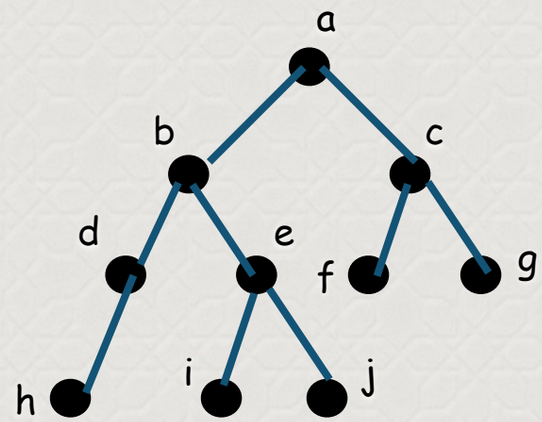
Busca não informada - quando todos os nós gerados são igualmente promissores, ou não se tem informação sobre o seu potencial: busca em profundidade, busca em largura, busca de custo uniforme

Busca informada - quando conhecimento heurístico pode ser levantado que distingue entre os nós gerados em um mesmo nível da árvore.

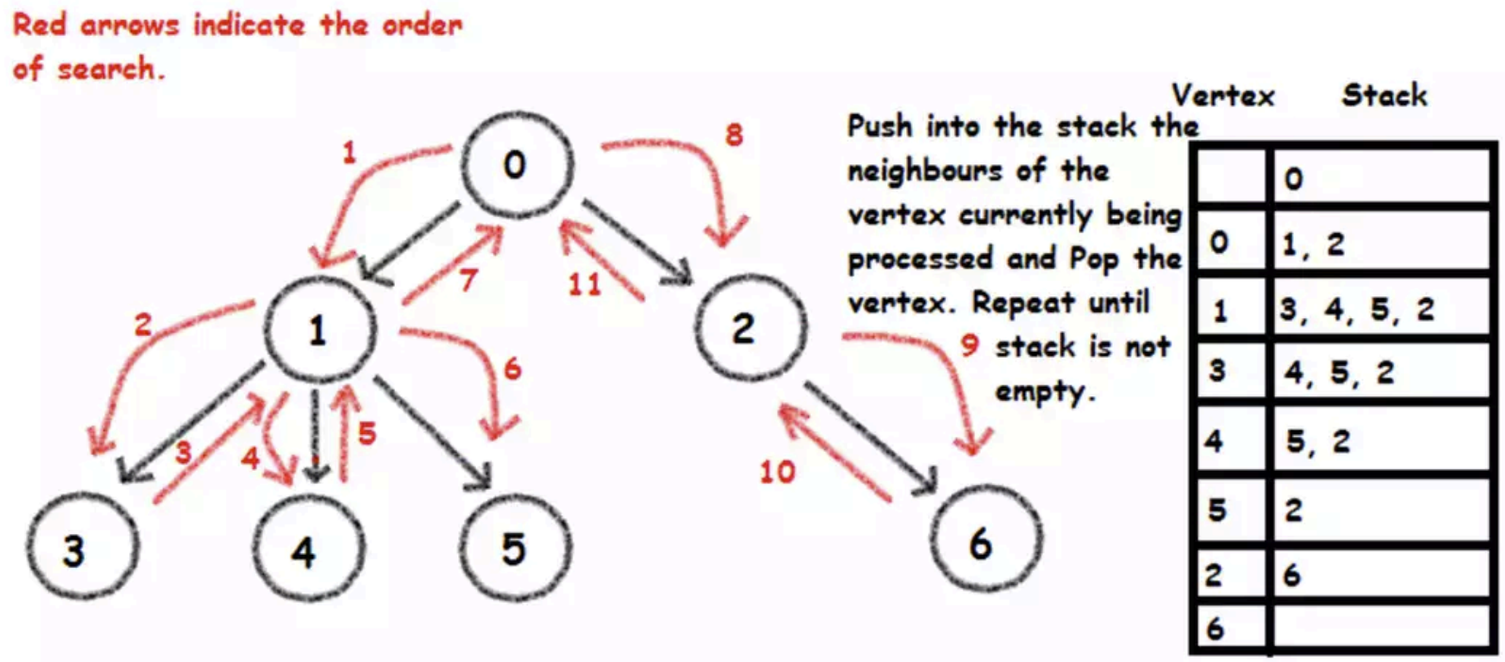
A busca informada é um algoritmo interessante quando sabemos que a solução é uma das folhas. No exemplo abaixo temos uma árvore e estamos buscando nós que estão nas folhas.

Ainda na hipótese que podemos gerar todo o espaço de estados (o que não será possível na maioria dos problemas práticos) podemos representar o grafo ao lado pelas arestas:

s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).

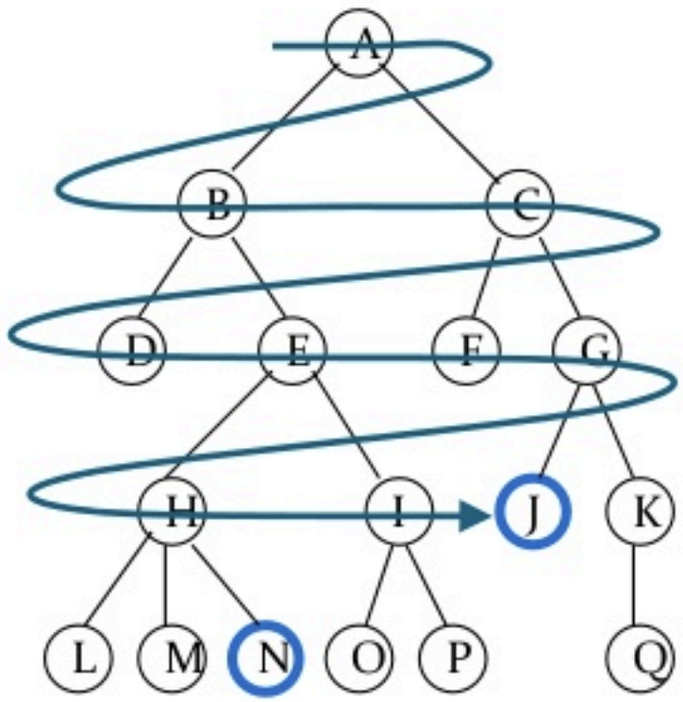


Busca em profundidade



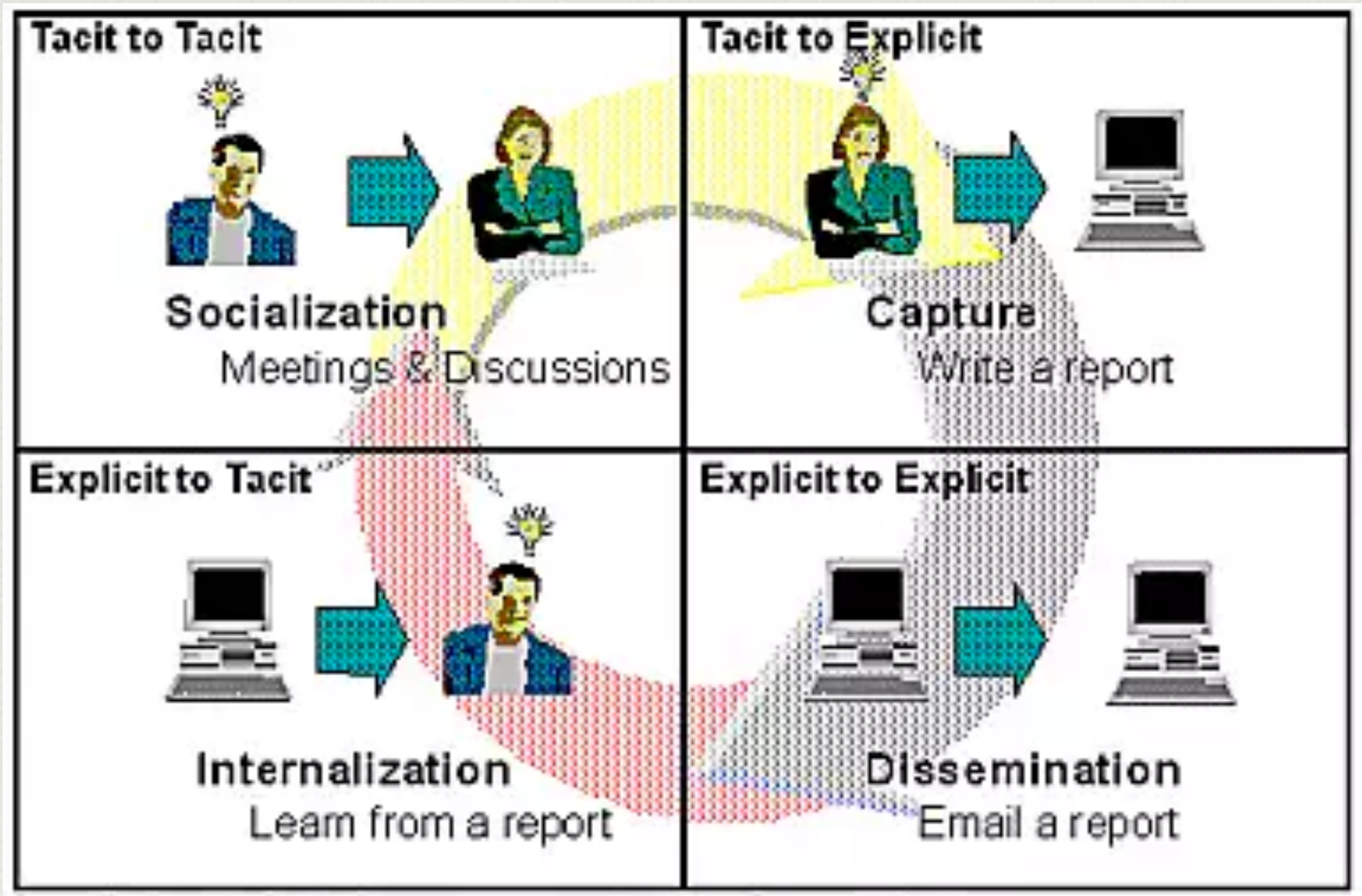
Busca em largura

Breadth-first searching[1]



- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Node are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

Introduzindo heurísticas



Heuristic search

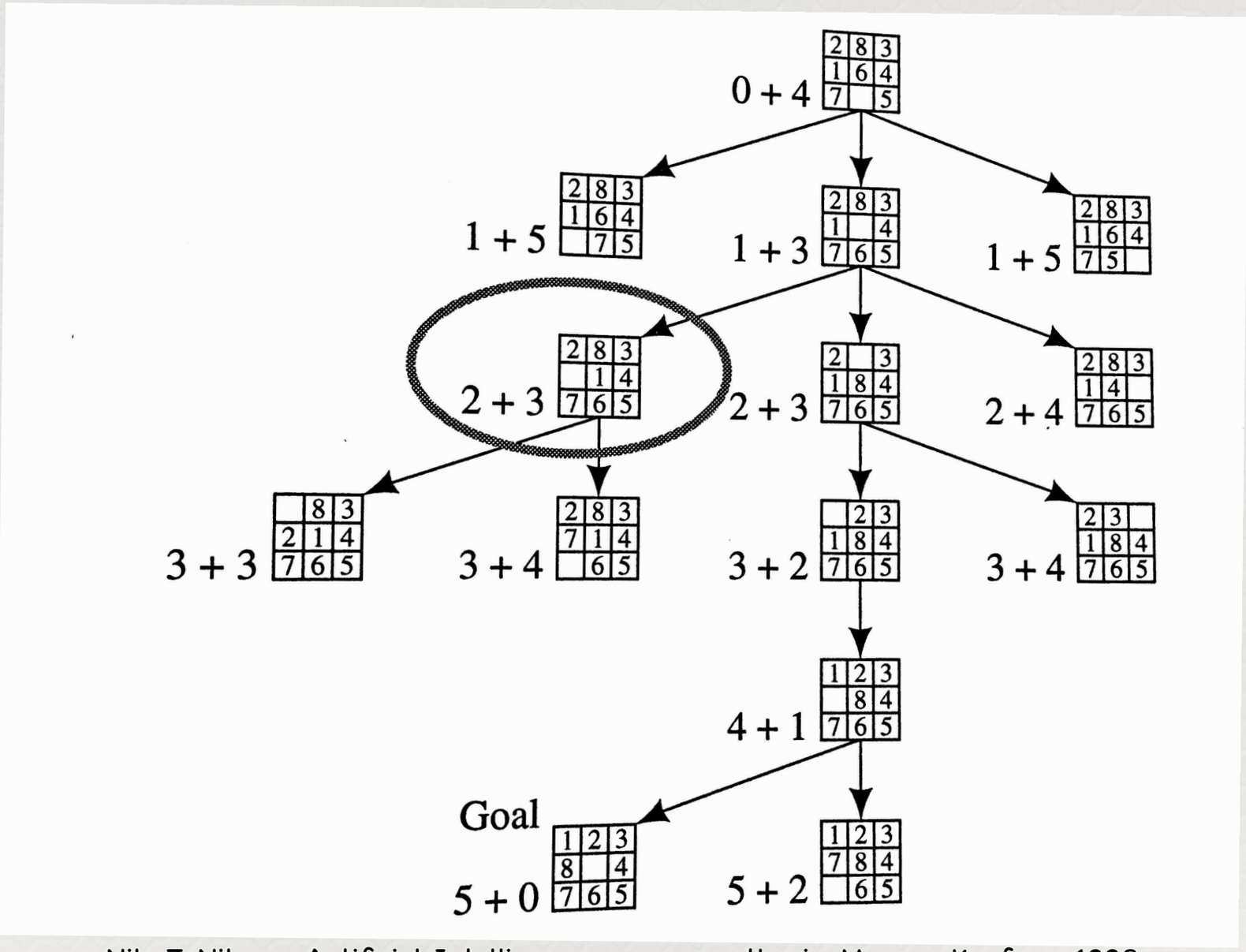
In computer science, artificial intelligence, and mathematical optimization, a **heuristic** (from Greek εὑρίσκω "I find, discover") is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

find a function

O algoritmo "best-first"

Portanto, uma possibilidade de algoritmo com melhor performance que qualquer um daqueles da "busca não-informada seria aplicar o breadth-first, mas, escolhendo como primeira escolha não o primeiro nó à esquerda mas o melhor segundo a função de avaliação.

Árvore de busca e a busca informada



Nils J. Nilsson, Artificial Intelligence: a new synthesis, Morgan Kaufann, 1998

The A* Algorithm

The central idea in the so-called *A* algorithm* is to guide best-first search both by

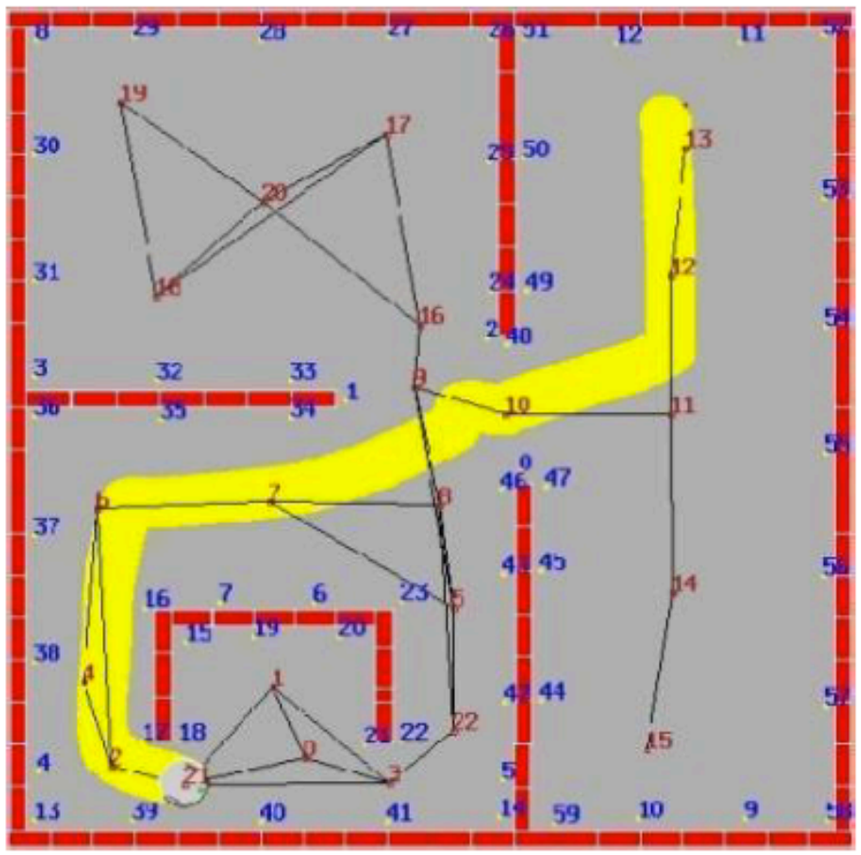
- the estimate to the goal as given by the heuristic function h and
- the cost of the path developed so far.

Let n be a node, $g(n)$ the cost of moving from the initial node to n along the current path, and $h(n)$ the estimated cost of reaching a goal node from n. Define $f(n)$ as follows:

$$f(n) \quad = \quad g(n) + h(n)$$

This is the *estimated cost of the cheapest path through* n leading from the initial node to a goal node. *A* is the best-first search algorithm that always expands a node* n *such that* $f(n)$ *is minimal.*

Robot Navigation



Source: http://www.ics.forth.gr/cvrl/

Estratégias de Busca Informada

Análise de custo

$f(x) = g(x)$

Heurística

$f(x) = h(x)$

Algorítmo A*

$f(x) = g(x) + h(x)$

Critérios para a escolha da heurística

O objetivo do A* é guiar a busca para a solução de modo a atingir o estado alvo mais rapidamente e com menor custo. Portanto a heurística escolhida deve ser admissível.

Uma heurística h(x) é dita admissível se para cada nó n da árvore de busca $h(n) \leq h^*(n)$, onde onde h(n) é o custo estimado para atingir a solução e $h^*(n)$ é o custo real. Portanto, o algoritmo não deve superestimar o custo para atingir o objetivo.

formalmente...

Teorema: Se h(n) é admissível, o algoritmo A* baseado em árvore de busca é "otimizante", isto é, pode chegar a um **caminho ótimo de solução**.

Comparando diferentes propostas de heurísticas

Sejam duas heurísticas admissíveis $h_1(x)$ e $h_2(x)$. Ambas devem ter como limite o valor real $h^*(x)$ e tendem para este limite. Portanto, se para qualquer nó x $h_2(x) \geq h_1(x)$, significa que $h_1(x)$ está mais próxima do limite e dizemos que h_1 "domina" h_2.

A heurística dominante deve gerar uma busca melhor, e expandir um número menor de nós alternativos para a busca.

Teorema: Se uma heurística h é consistente, então usando A* em uma busca em grafo orientada - onde se checa se o sucessor já foi visitado antes para evitar loops - é ótima.

A função de seleção

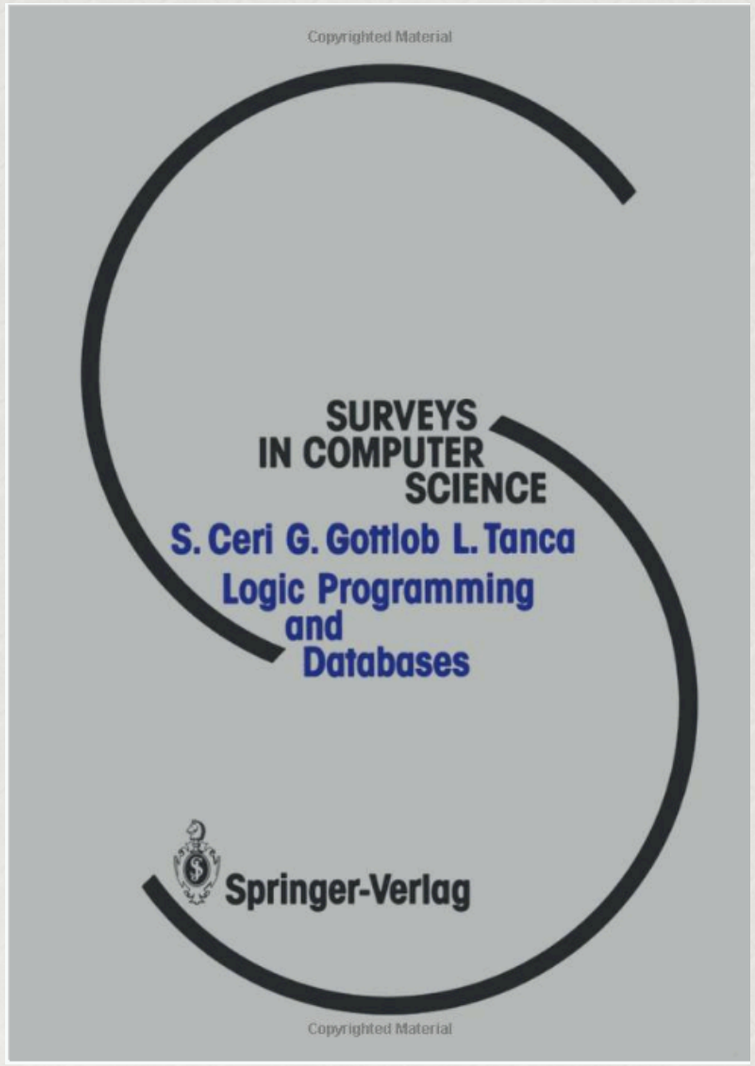
$$f(x) = g(x) + h(x)$$

custo para chegar ao nó n da árvore de busca

custo estimado para chegar ao nó objetivo

Teorema: Se uma heurística h é consistente, então usando A* em uma busca em grafo, isto é, uma busca onde se checa o sucessor já foi visitado antes para evitar loops, é ótima.

Programação Lógica e Bancos de Dados



A conexão entre programação lógica e bancos de dados poderia associar o poder de dedução do Prolog com o modelo relacional dos BDs.


```
init_state([on(b,a), on(a,table), on(c,table), free(b), free(c)]).

stack_plan([]).

move(X,table,Z,S):- subset([on(X, R)],Z),
    remove([on(X,R)],Z,W),
    add([on(X,table),free(R)], W, S).
move(X,Y,Z,S):- subset([free(X)],Z), subset([free(Y)],Z),
    subset([on(X,R)],Z),
    remove([free(Y),on(X,R)],Z,W),
    add([on(X,Y), free(R)], W, S).

make_plan(X,X).
make_plan(P1,P2):-  next_state(P1,Z), ..., make_plan(Z,P2).

remove(M,N,K):- subtract(N,M,K).
add(M,N,K) :- append(M,N,K).
```

swish.swi-prolog.org/example/examples.swinb

AliExpress | Booking.com | Dafiti | Americanas | Facebook | Getting Started

SWISH File Edit Examples Help

134 users online

Search

examples | Program

```
1  init_state([on(b,a), on(a,table), on(c,table), free(b), free(c)]).
2
3  stack_plan([]).
4
5  move(X,table,Z,S):- subset([on(X, R)],Z),
6      remove([on(X,R)],Z,W),
7      add([on(X,table),free(R)], W, S).
8  move(X,Y,Z,S):- subset([free(X)],Z), subset([free(Y)],Z),
9      subset([on(X,R)],Z),
10     remove([free(Y),on(X,R)],Z,W),
11     add([on(X,Y), free(R)], W, S).
12
13 make_plan(X,X).
14 make_plan(P1,P2):-  next_state(P1,Z), ..., make_plan(Z,P2).
15
16 remove(M,N,K):- subtract(N,M,K).
17 add(M,N,K) :- append(M,N,K).
18
19
```

init_state(X), *move*(b,c,X,S).

S = [*on*(b, c), *free*(a), *on*(a, (table)), *on*(c, (table)), *free*(b)],
X = [*on*(b, a), *on*(a, (table)), *on*(c, (table)), *free*(b), *free*(c)]

?- init_state(X), move(b,c,X,S).

Examples | History | Solutions | table results | Run!

Relating AI Planning and Problem Solving

Domain-depend planning

One develops predictive models for the type of actions to be planned for and for the states of the system in which they take place. Computational tools for running these models, in order to predict and assess the effects of alternate actions and plans in various situations, exploit the specifics of the domain.

Domain-independent planning

Domain-independent planning relies on abstract, general models of actions. These models range from very simple ones that allow only for limited forms of reasoning to models with richer prediction capabilities. There are in particular the following forms of models and planning capabilities.

Type of modelling in planning

Project Planning

in which models of actions are reduced mainly to temporal and precedence constraints, e.g., the earliest and latest start times of an action or its latency with respect to another action.

Scheduling and resource allocation

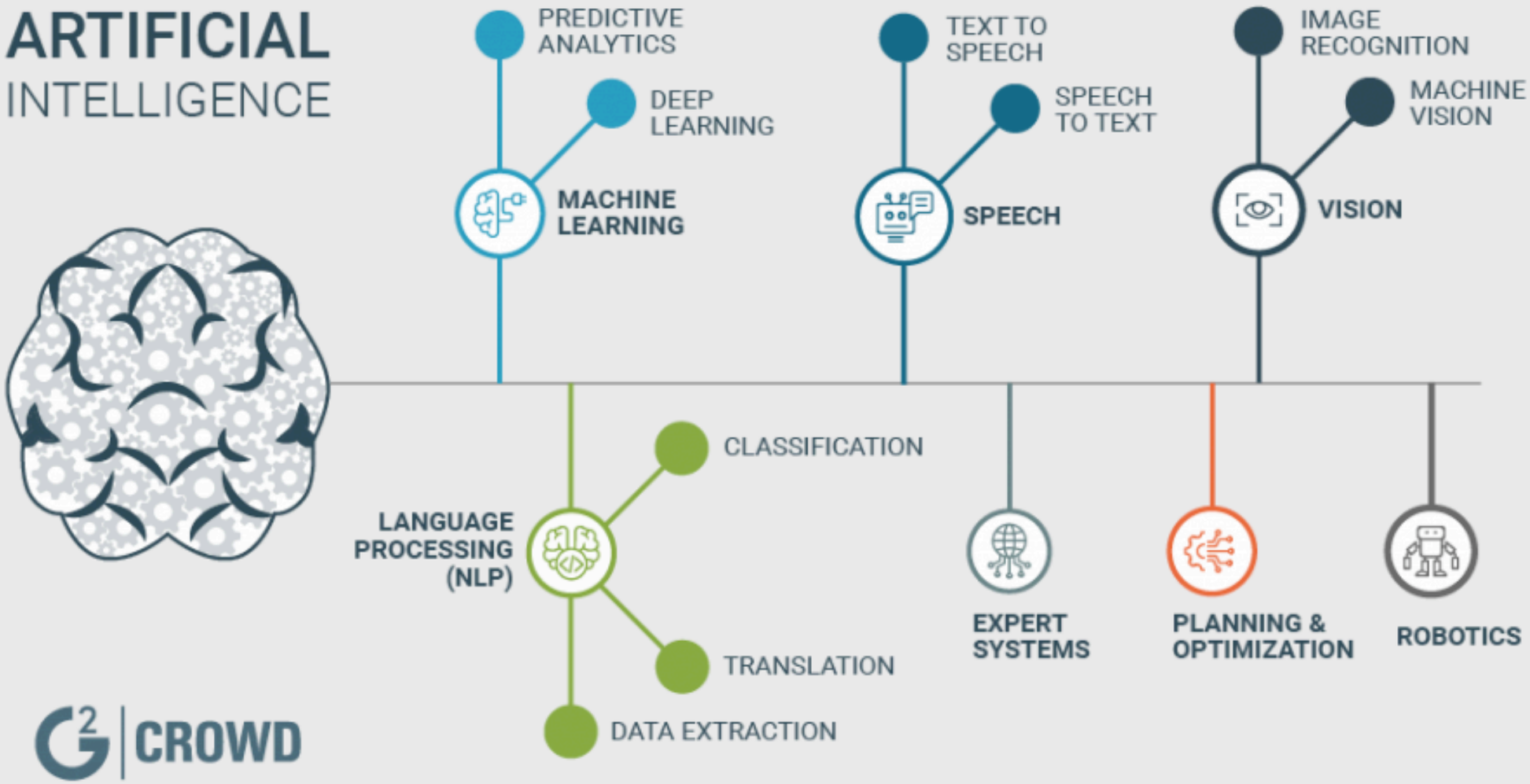
in which the action models include the above types of constraints plus constraints on the resources to be used by each action.or its latency with respect to another action.

Plan Synthesis

in which the action models enrich the precedent models with the conditions needed for the applicability of an action and the effects of the action on the state of the world.

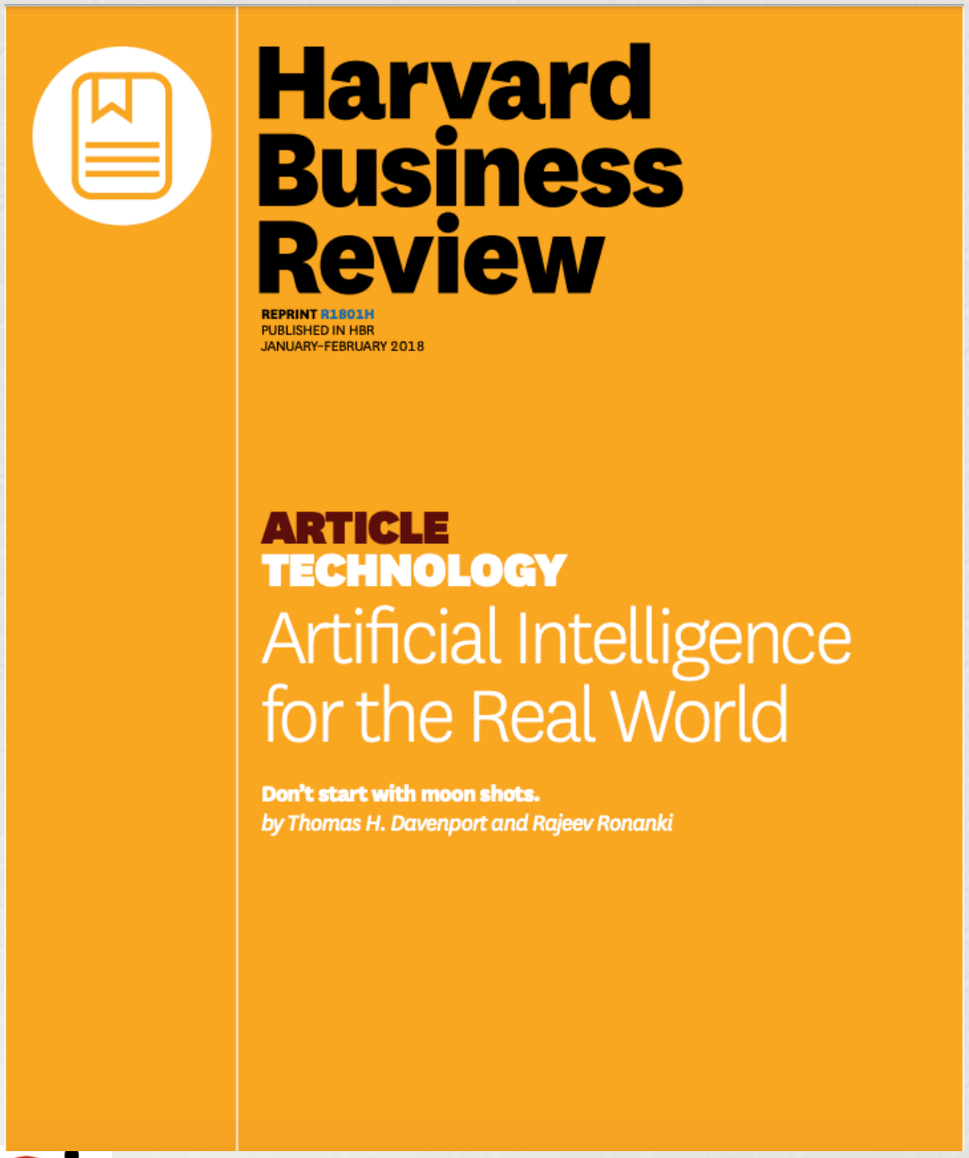
The logistic of containers in ports

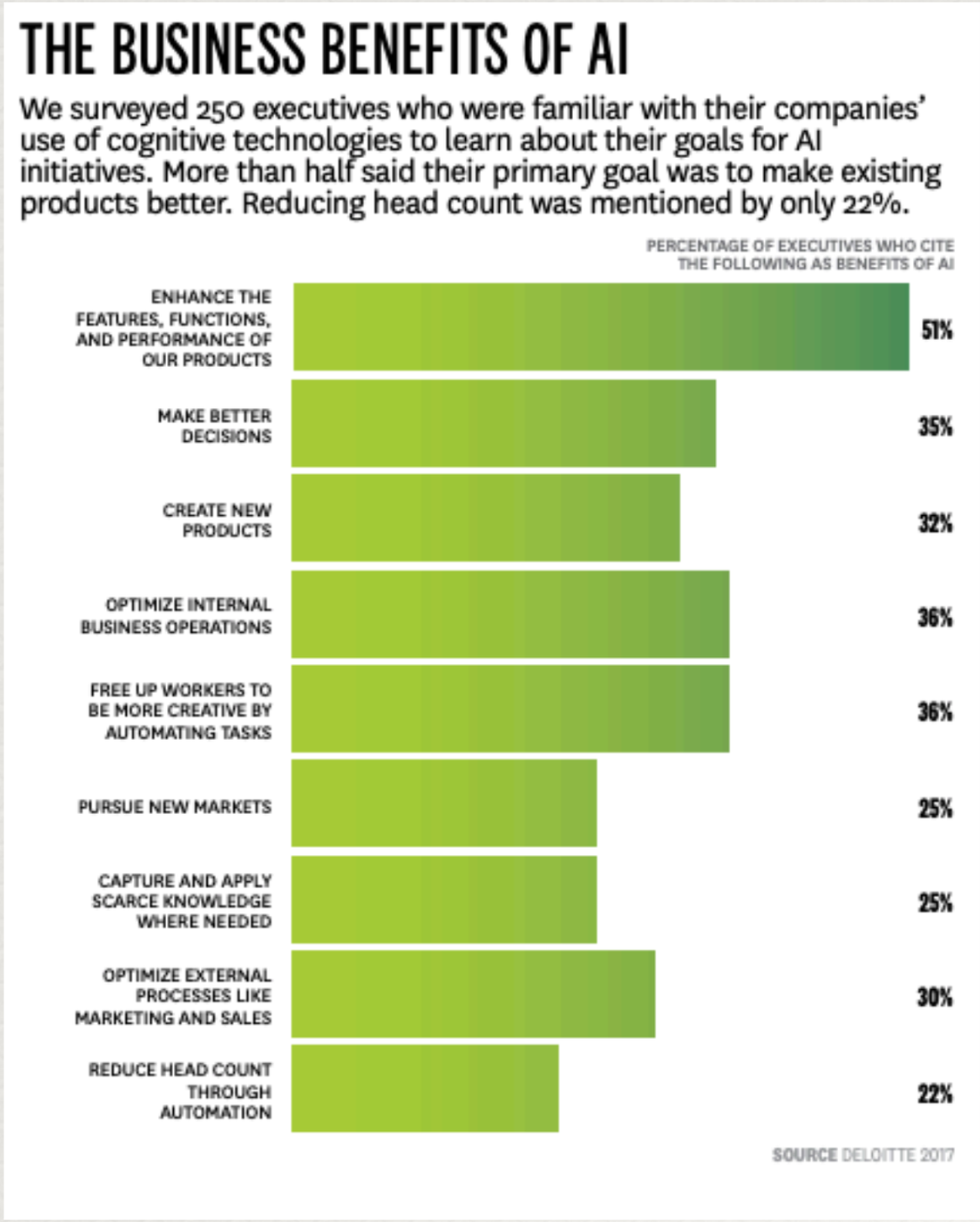
Space is valuable in port operation as well as the operation of storing and retrieving the proper container.


ARTIFICIAL
INTELLIGENCE
MACHINE LEARNING
PREDICTIVE ANALYTICS
DEEP LEARNING
SPEECH
TEXT TO SPEECH
SPEECH TO TEXT
VISION
IMAGE RECOGNITION
MACHINE VISION
LANGUAGE PROCESSING (NLP)
CLASSIFICATION
TRANSLATION
DATA EXTRACTION
EXPERT SYSTEMS
PLANNING & OPTIMIZATION
ROBOTICS
G2 CROWD

Stakeholder expectations

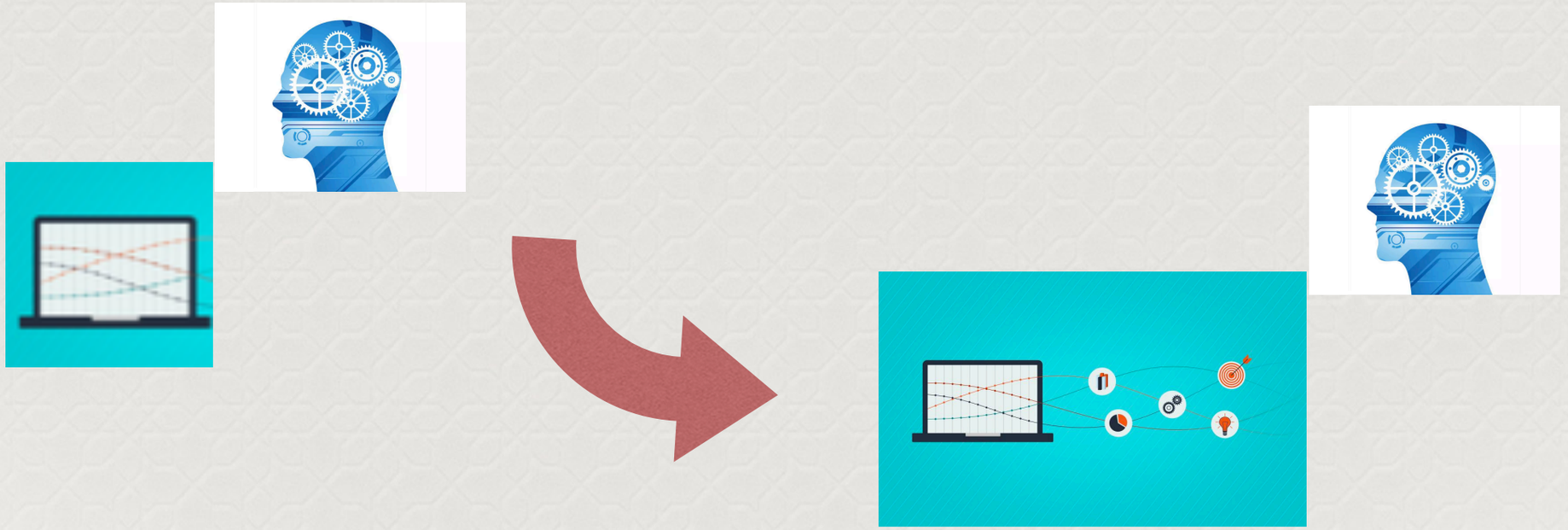
https://www.kungfu.ai/wp-content/uploads/2019/01/R1801H-PDF-ENG.pdf

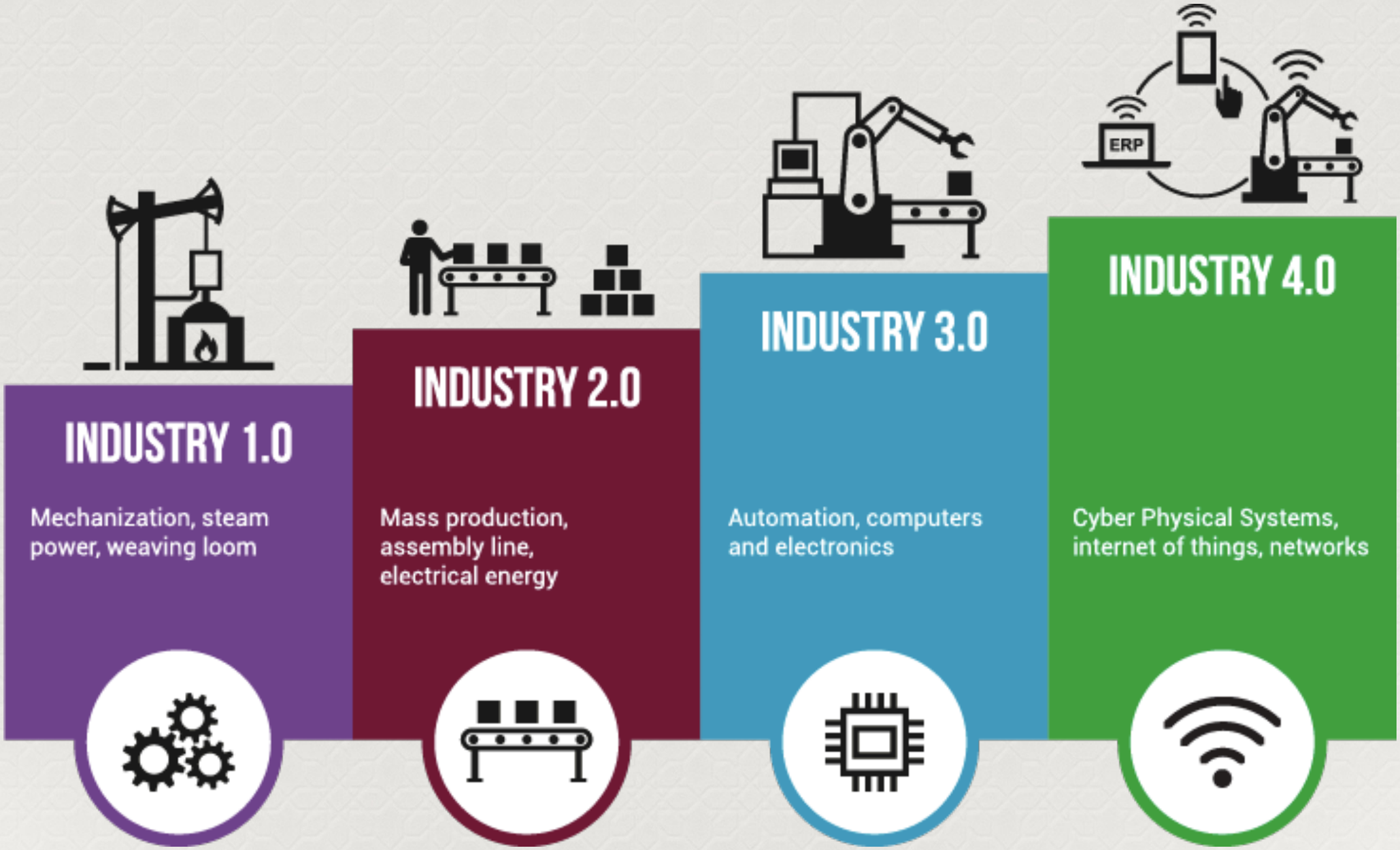


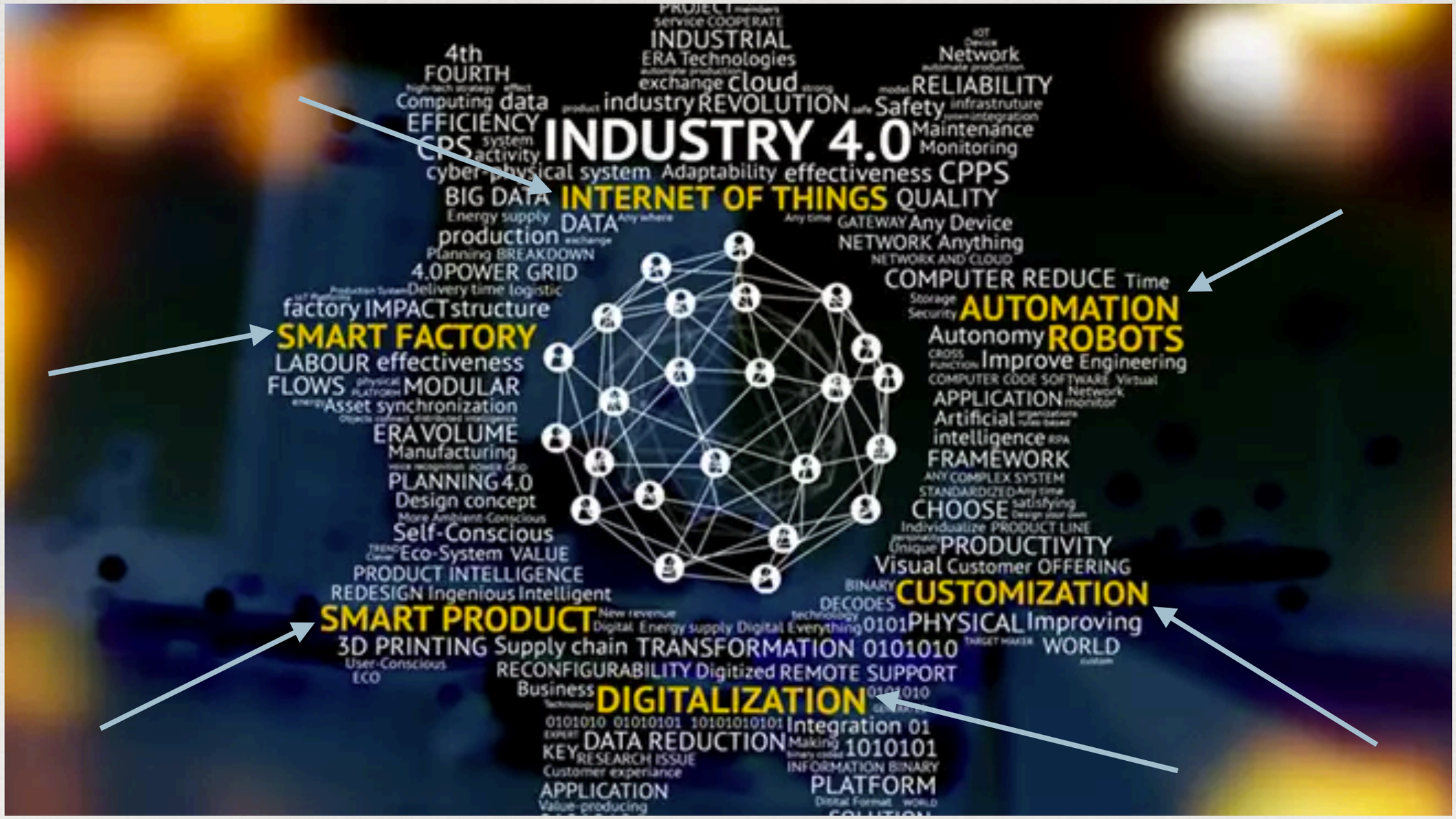


Resolução de problemas com IA

Existem duas maneiras de resolver problemas usando IA: uma é algorítmica, usando busca (clássica ou informada); a outra é usando inferência lógica.

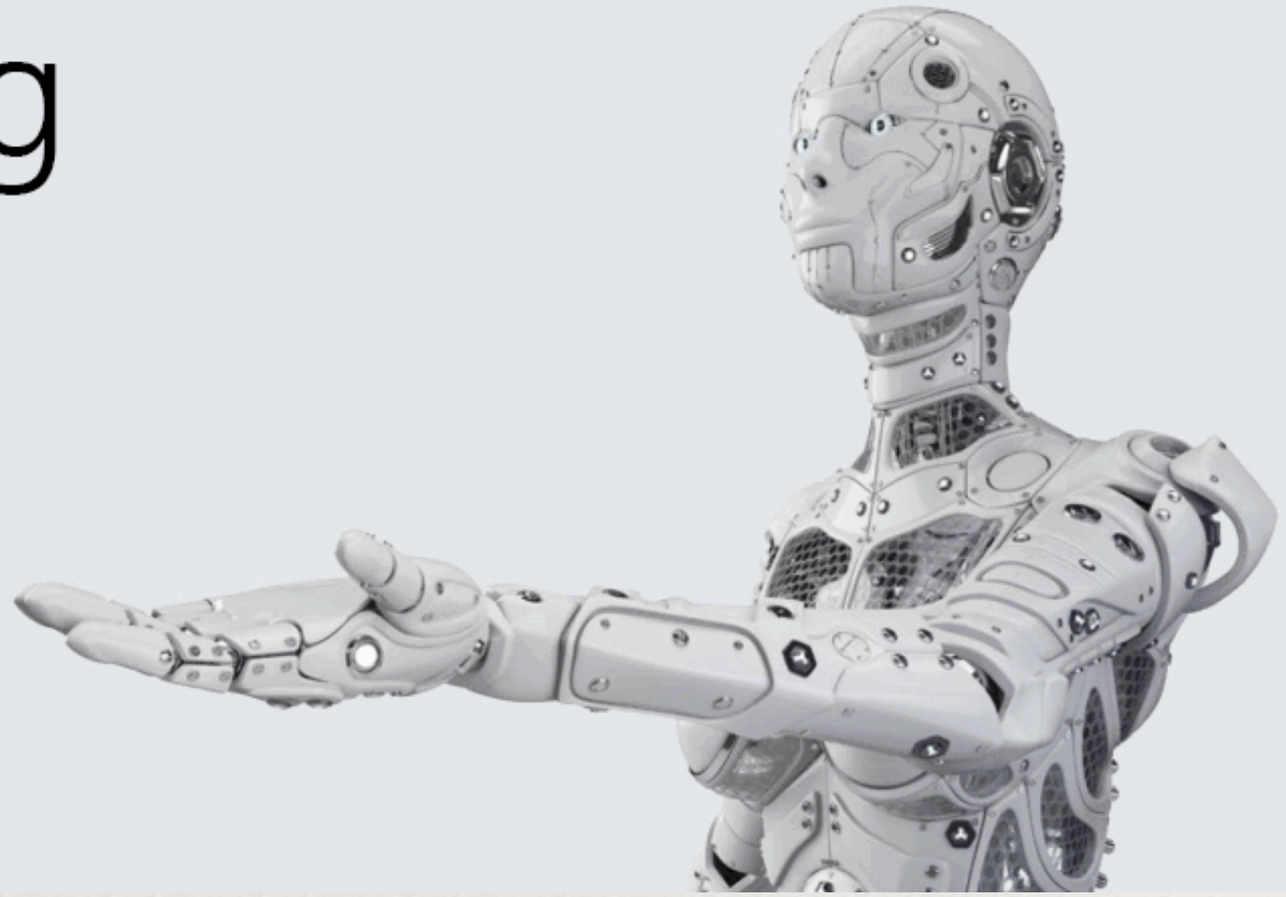



ERP
INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks


INDUSTRY 4.0
INTERNET OF THINGS
SMART FACTORY
SMART PRODUCT
DIGITALIZATION
AUTOMATION
ROBOTS
CUSTOMIZATION
INDUSTRIAL
ERA Technologies
cloud
REVOLUTION
RELIABILITY
Safety
Maintenance
Monitoring
Adaptability
effectiveness
CPPS
QUALITY
BIG DATA
DATA
production
GATEWAY
Any Device
NETWORK Anything
COMPUTER REDUCE Time
Improve
Engineering
APPLICATION
Artificial
intelligence
FRAMEWORK
CHOOSE
PRODUCTIVITY
Visual Customer OFFERING
PHYSICAL
WORLD
3D PRINTING
Supply chain
TRANSFORMATION 0101010
RECONFIGURABILITY
Digitized
REMOTE SUPPORT
Business
DATA REDUCTION
Integration 01
1010101
PLATFORM
APPLICATION
LABOUR effectiveness
FLOWS
MODULAR
Asset synchronization
ERA VOLUME
Manufacturing
PLANNING 4.0
Design concept
Self-Conscious
Eco-System VALUE
PRODUCT INTELLIGENCE
REDESIGN Ingenious Intelligent
factory IMPACT structure
4.0POWER GRID
Delivery time logistic
Planning BREAKDOWN
Energy supply
4th
FOURTH
Computing data
EFFICIENCY
CPS
cyber-physical system
exchange
industry
Network
Safety
Maintenance
Monitoring
COMPUTER CODE SOFTWARE
Storage
Security
Autonomy
CHOOSE
Individualize PRODUCT LINE
Unique
Visual Customer OFFERING
CUSTOMIZATION
PHYSICAL Improving
WORLD
Digital Energy supply Digital Everything
0101
DECODES
BINARY
Business
DATA REDUCTION
KEY
RESEARCH ISSUE
Customer experiance
APPLICATION
Value-producing
INFORMATION BINARY
PLATFORM
Digital Format
0101010 01010101 1010101010
Integration 01
1010101
Making
New revenue
Digital
User-Conscious
ECO

AI's coming of age

The progress into the AGI phase and the beginning of true autonomy.



Chegamos ao final!