Escola Politécnica da USP - Depto. de Enga. Mecatrônica

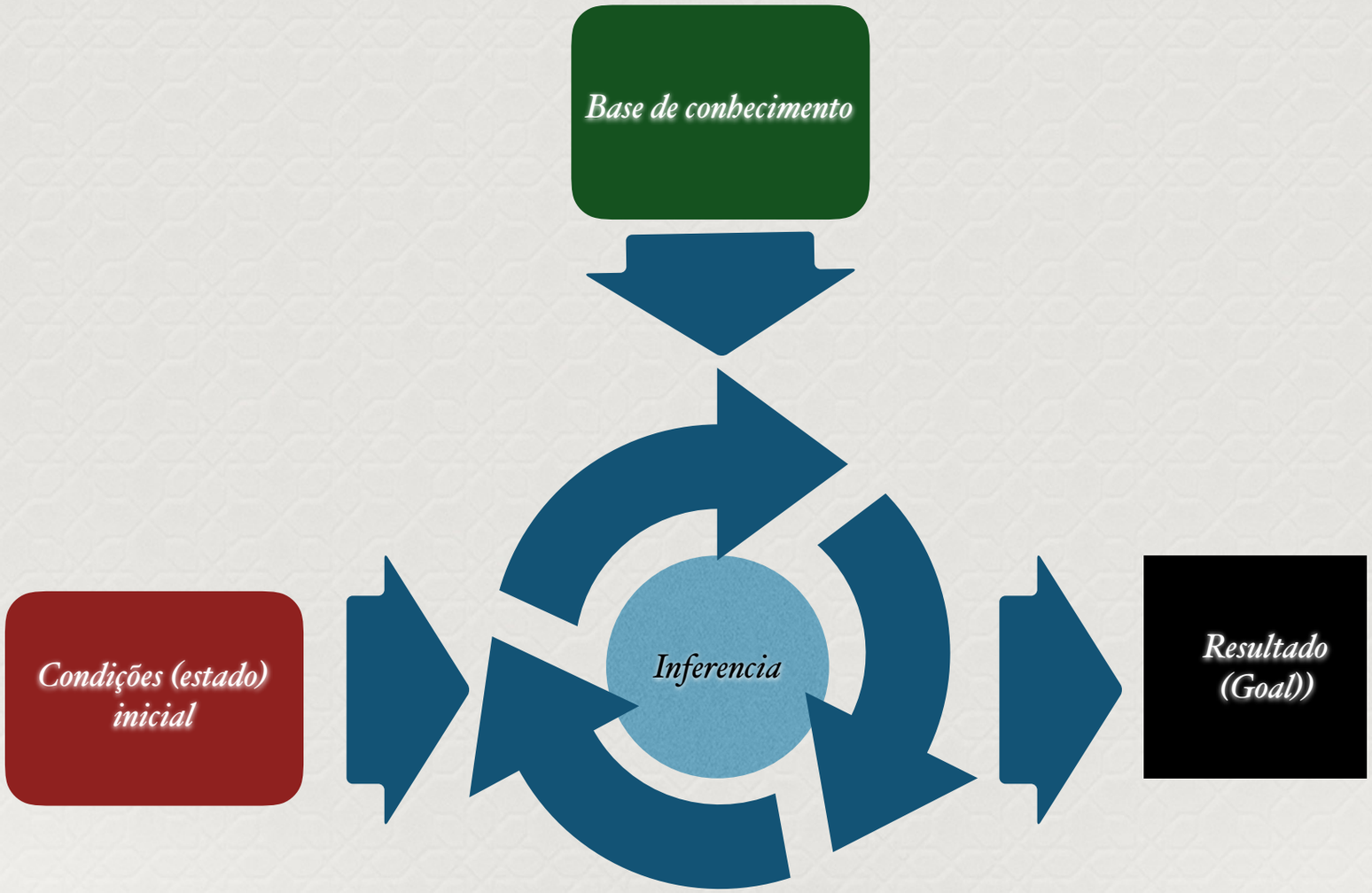
PMR-3510 Inteligência Artificial

Aula 4- Resolução de problemas por máquinas e por humanos

Prof. José Reinaldo Silva

reinaldo@usp.br

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

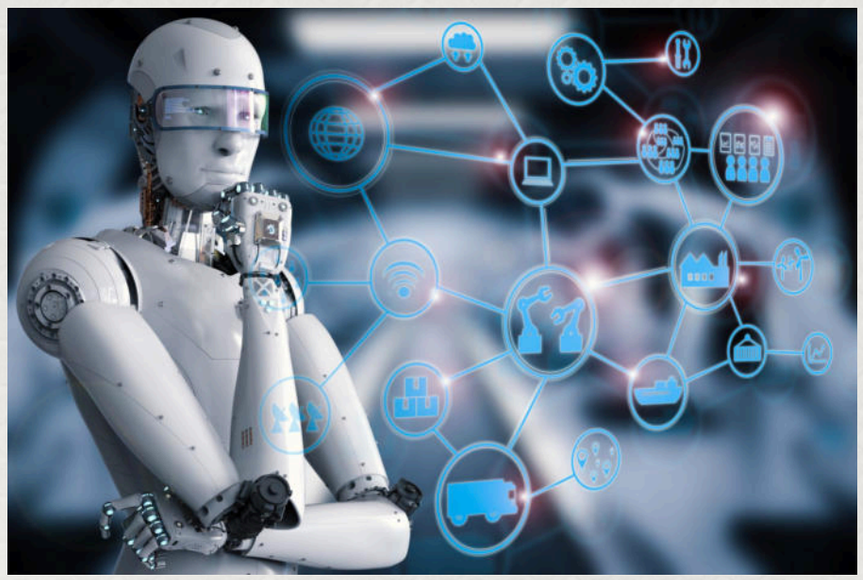
1. achar uma abordagem geral que leva do estado inicial ao estado final;
2. testar esta abordagem em alguns casos (sem levar em conta o tempo para chegar à solução);
3. checar se a abordagem é completa, isto é, resolve todos os casos ou há casos especiais onde o problema "não converge";
4. preparar a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

Solução

Achar um método geral de resolução de problemas

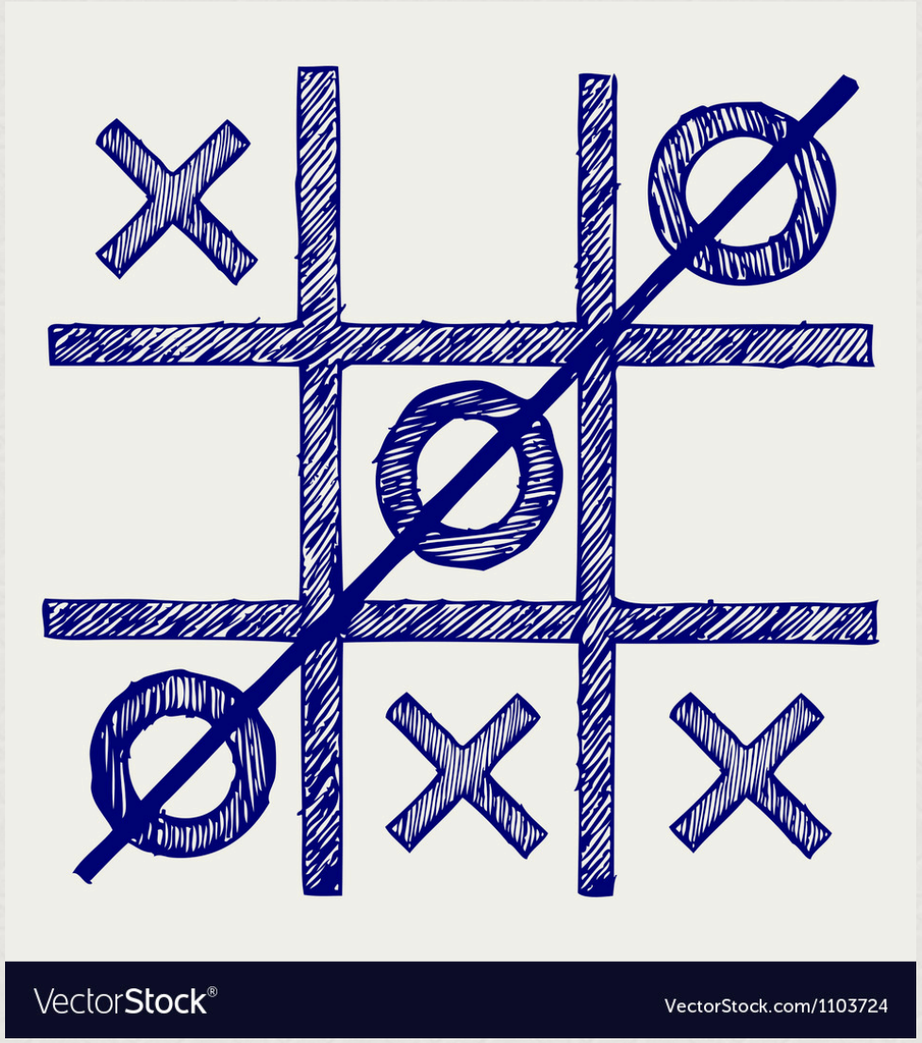
Estrutura de um "resolvedor automático de problemas"

Para dotar uma máquina da capacidade de resolver problemas (ou uma classe de problemas) é preciso ter uma estrutura com os seguintes atributos:



1. uma descrição clara do "estado inicial" ou seja das condições iniciais do problema a ser resolvido;
2. uma descrição clara do objetivo ou "estado final", de modo que seja possível saber quando (e se) o problema foi resolvido;
3. em cada estágio do processo de solução saber quais os próximos estados que podem ser atingidos;
4. poder escolher um (ou o melhor) caminho entre os estados acima;
5. saber que operadores (ou passos) aplicar para fazer a "transição" para um próximo estado;
6. discernir se estamos convergindo para a solução.

Tic-tac-toe (o jogo da velha)



swish.swi-prolog.org/p/Tic-Tac-Toe.swinb

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

SWISH File Edit Examples Help

77 users online

Search

Tic-Tac-Toe

```
58 % The predicate orespond generates the computer's (playing o) reponse
59 % from the current Board.
60
61 orespond(Board,Newboard) :-
62   move(Board, o, Newboard),
63   win(Newboard, o),
64   !.
65 orespond(Board,Newboard) :-
66   move(Board, o, Newboard),
67   not(x_can_win_in_one(Newboard)).
68 orespond(Board,Newboard) :-
69   move(Board, o, Newboard).
70 orespond(Board,Newboard) :-
71   not(member(b,Board)),
72   !,
73   write('Cats game!'), nl,
74   Newboard = Board.
75
76 % The following translates from an integer description
77 % of x's move to a board transformation.
```

?- playo

?- Your query goes here ...

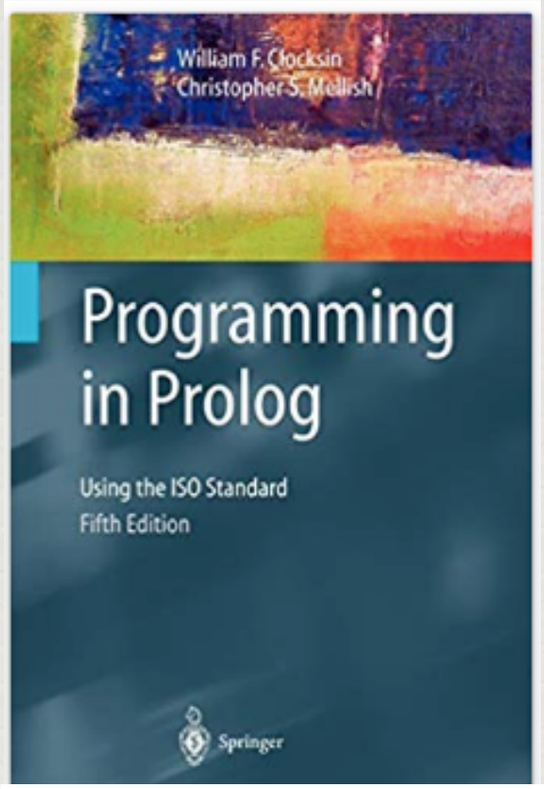
Examples History Solutions table results Run!

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



Programming in PROLOG: Using the ISO Standard
25 jul 2003, 5th. Edition
William F. Clocksin and Christopher S. Hellish

Conceitualmente a linguagem é composta por:

Fatos

Regras

Queries

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).
```

swish.swi-prolog.org/p/Tic-Tac-Toe.swinb

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

SWISH File Edit Examples Help

75 users online

Search

Tic-Tac-Toe | Program

```
1  mother_child(trude, sally).
2
3  father_child(tom, sally).
4  father_child(tom, erica).
5  father_child(mike, tom).
6
7  sibling(X, Y):- parent_child(Z, X), parent_child(Z, Y).
8
9  parent_child(X, Y) :- father_child(X, Y).
10 parent_child(X, Y) :- mother_child(X, Y).
```


parent_child(trude,Y).

Y = sally

?-

```
parent_child(trude,Y).
```

Examples | History | Solutions | table results | Run!

```
?- father_child(X, Y),
   father_child(Y, Z).
```

?

```
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).
```

swish.swi-prolog.org/p/Tic-Tac-Toe.swinb

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

SWISH File Edit Examples Help

69 users online

Search

Tic-Tac-Toe | Program

```
1  mother_child(trude, sally).
2
3  father_child(tom, sally).
4  father_child(tom, erica).
5  father_child(mike, tom).
6
7  sibling(X, Y):- parent_child(Z, X), parent_child(Z, Y).
8
9  parent_child(X, Y) :- father_child(X, Y).
10 parent_child(X, Y) :- mother_child(X, Y).
```


father_child(X,Y),father_child(Y,Z).

X = mike,
Y = tom,
Z = sally

Next | 10 | 100 | 1,000 | Stop

?-

```
father_child(X,Y),father_child(Y,z).
```

Examples | History | Solutions | table results | Run!

Relacionando objetos e fatos: regras

O que usamos de fato para modelar conhecimento (no modelo clássico da IA) são as regras. Regras relacionam objetos, fatos e outras regras - de forma recursiva.

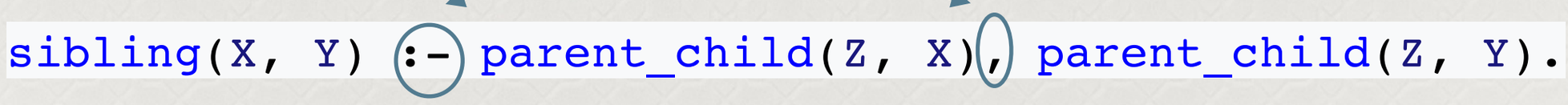
```
mother_child(trude, sally).

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).
```

Mas o que é de fato uma "regra"?

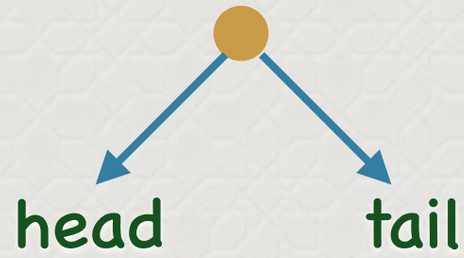
(if) parent(Z,X) (and) parent(Z,Y) (then) sibling(X,Y)

parent(Z,X) (and) parent(Z,Y) (implies) sibling(X,Y)

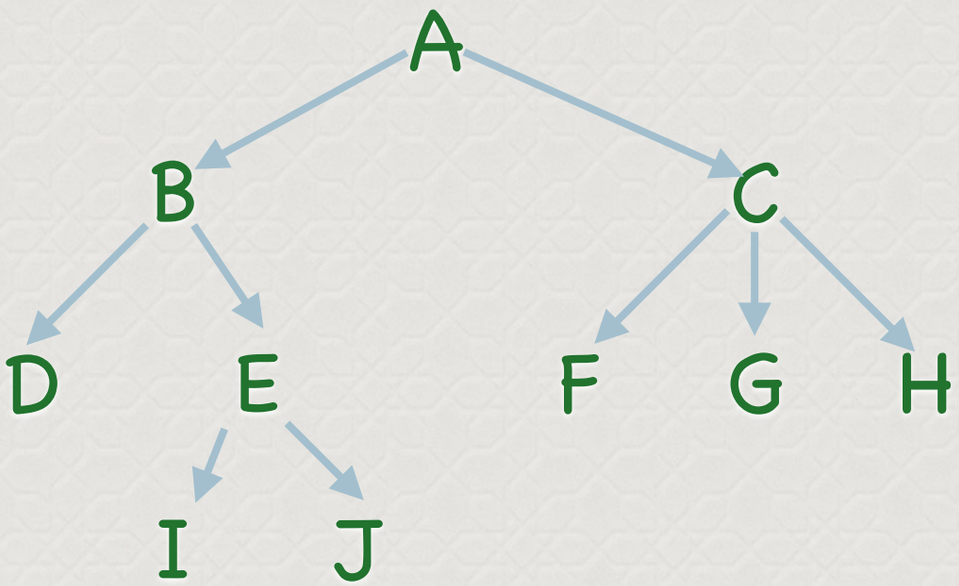
parent(Z,X) ∧ parent(Z,Y) → sibling(X,Y)



Estrutura de lista: [head | tail]



List	Head	Tail
[a, b, c]	a	[b, c]
[]	(none)	(none)
[[the, cat], sat]	[the, cat]	[sat]
[the, [cat, sat]]	the	[[cat, sat]]
[the, [cat, sat], down]	the	[[cat, sat], down]
[X+Y, x+y]	X+Y	[x+y]



[a | [b | [d, [e | [i, j]]]], [c | [f, g, h]]]

Operadores de comparação

X =:=Y	X and Y stand for the same number
X =\=Y	X and Y stand for different numbers
X <Y	X is less than Y
X >Y	X is greater than Y
X =<Y	X is less than or equal to Y
X >=Y	X is greater than or equal to Y

Comandos de controle: cut

Um dos poucos comandos de controle em Prolog é o cut, representado pelo símbolo "!". Vocês já se depararam com este comando analisando o programa to "jogo da velha" nas aulas anteriores e no segundo exercício pra casa.

O comando cut visa melhorar a eficiência dos programas Prolog evitando buscas inúteis.

Usando o comando cut

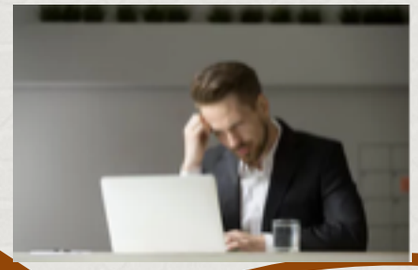
Vamos por exemplo calcular a soma dos N primeiros números naturais.

```
sum_to(1, 1) :- !.
sum_to(N, Res) :-N1 is N - 1, sum_to(N1, Res1),Res is Res1 + N.
```

Generalizando problemas e sua resolução...

Segundo Newell e Simon, no seu artigo "Human Problem Solving: The State of the Theory in 1970", se alguém se depara com uma "magica" da retirada de um coelho da cartola, pode reagir de duas maneiras distintas:

Aceitação

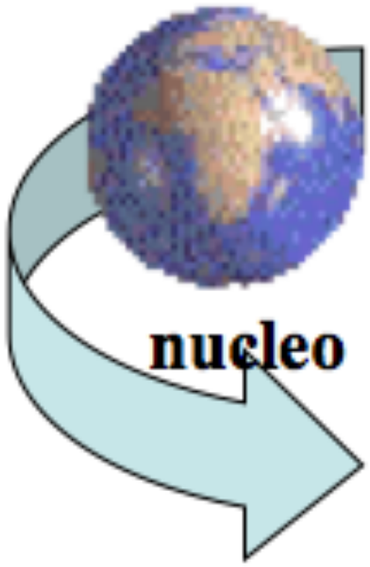


Curiosidade

Como se resolve um problema cientificamente ?

Elementos básicos de uma teoria

Imre Lakatos



heuristicas

Cinturão protetor

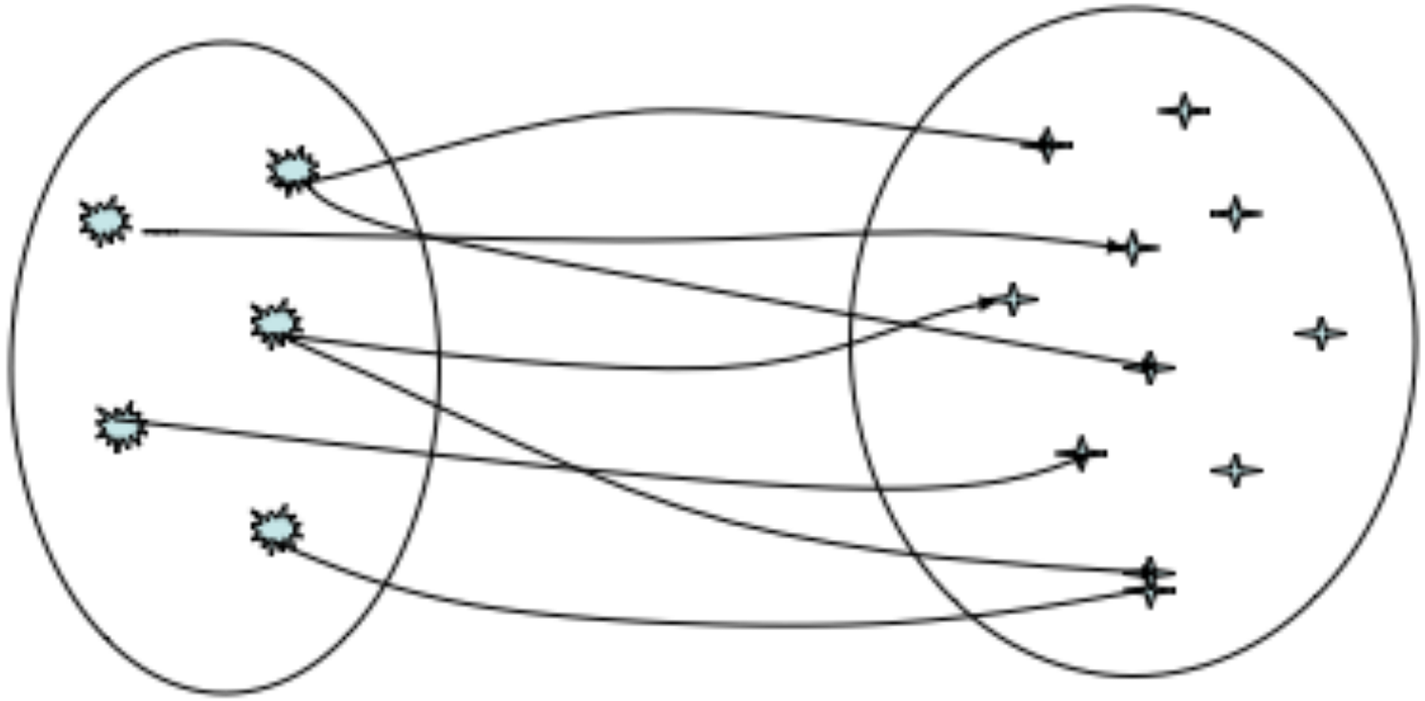
Racionalidade é a resposta

Allen Newel

Herbert Simon

Por volta de 1965 Herbert Simon e Allen Newell desenvolveram em Carnegie Mellon a teoria geral da racionalidade e propuseram que uma parte limitada dela fosse capturada em um sistema computacional capaz de "resolver problemas".

Porque o problema parece tão complicado?



Requisito de omnisciência

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).
```

A verdade na teoria pragmatista

A primeira teoria pragmatista foi publicada por Charles Sanders Peirce no artigo "How to Make Our Ideas Clear" ("Como tornar claras nossas idéias"), no número de janeiro de 1878, da revista Popular Science Monthly. Peirce desenvolveu a máxima pragmatista: "Para averiguar o significado de um conceito intelectual, é preciso considerar que suas conseqüências práticas podem ser inferidas como resultantes, necessariamente, da verdade desse conceito. A soma dessas conseqüências constituiria o significado do conceito."

Em outras palavras, postula que, para ter significado, um conceito ou idéia deve apresentar, em primeiro lugar, um correlato prático suscetível de comprovação experimental; segundo, que suas "conseqüências" se diferenciem claramente das de outro conceito. Dessa forma, a verdade de um conceito seria seu processo de verificação.

	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).
```

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)).
```

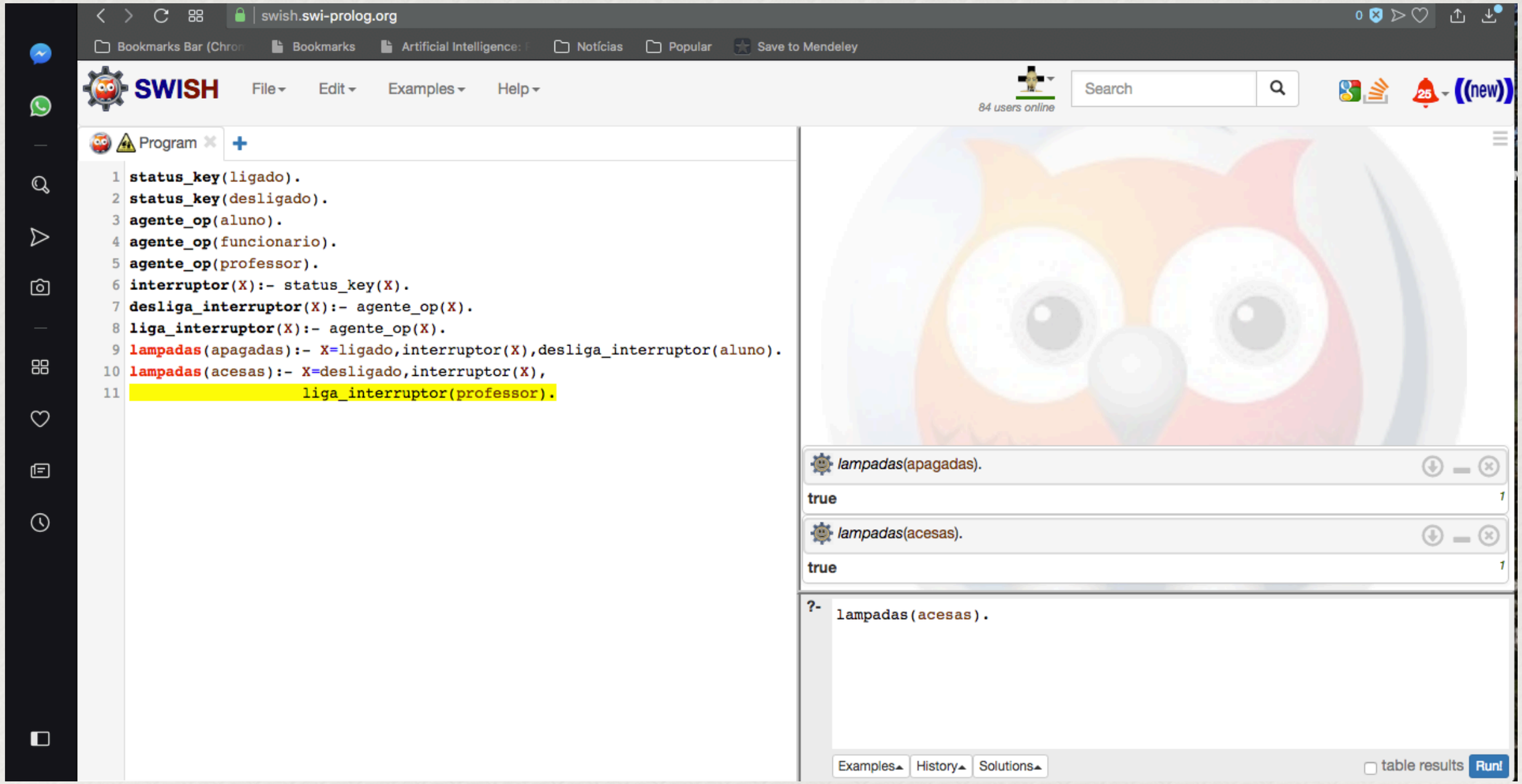
	P	Q	$P \rightarrow Q$
1	V	V	V
2	V	F	F
3	F	V	V
4	F	F	V

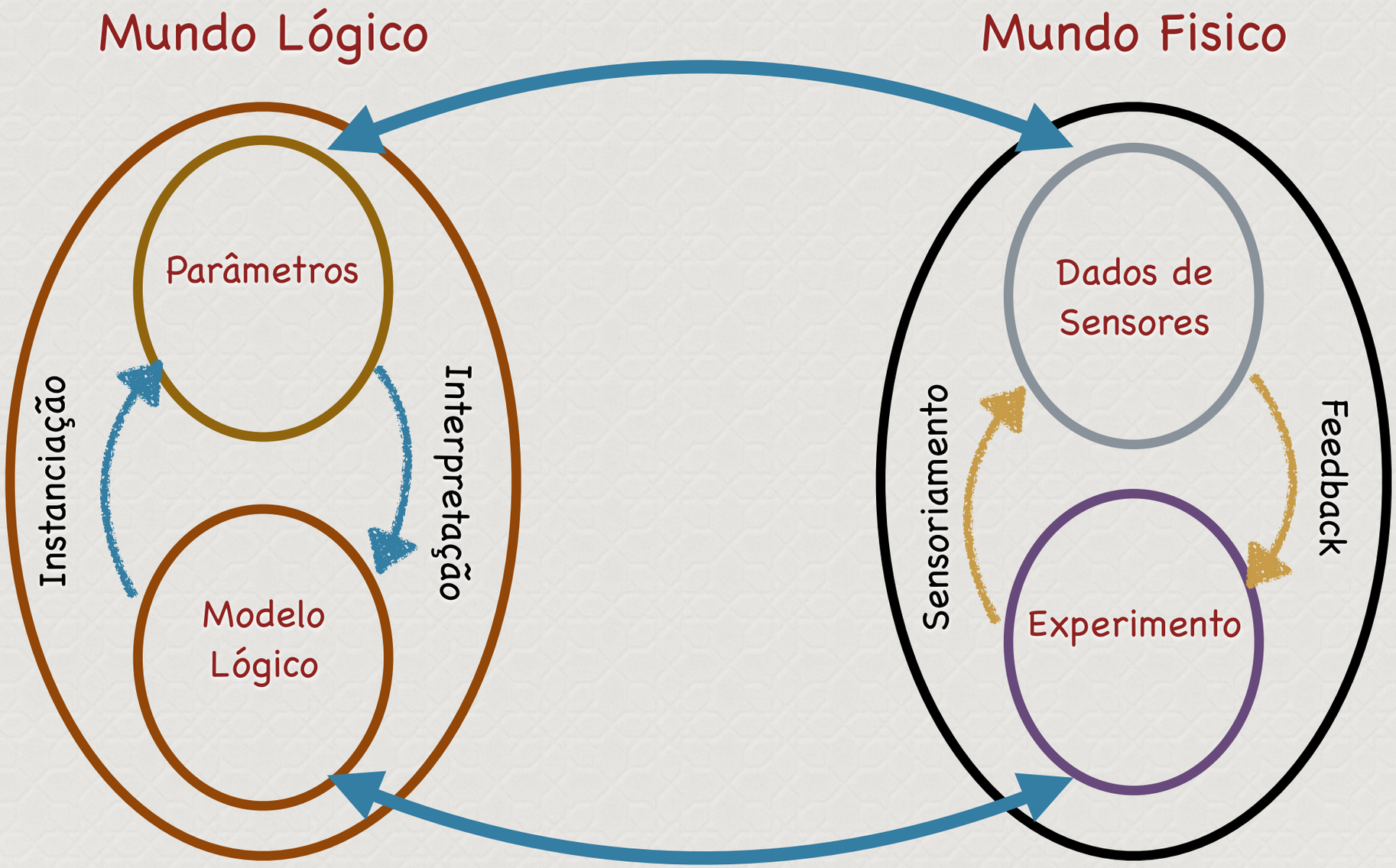
((A → B) AND not(B)) → not(A)

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

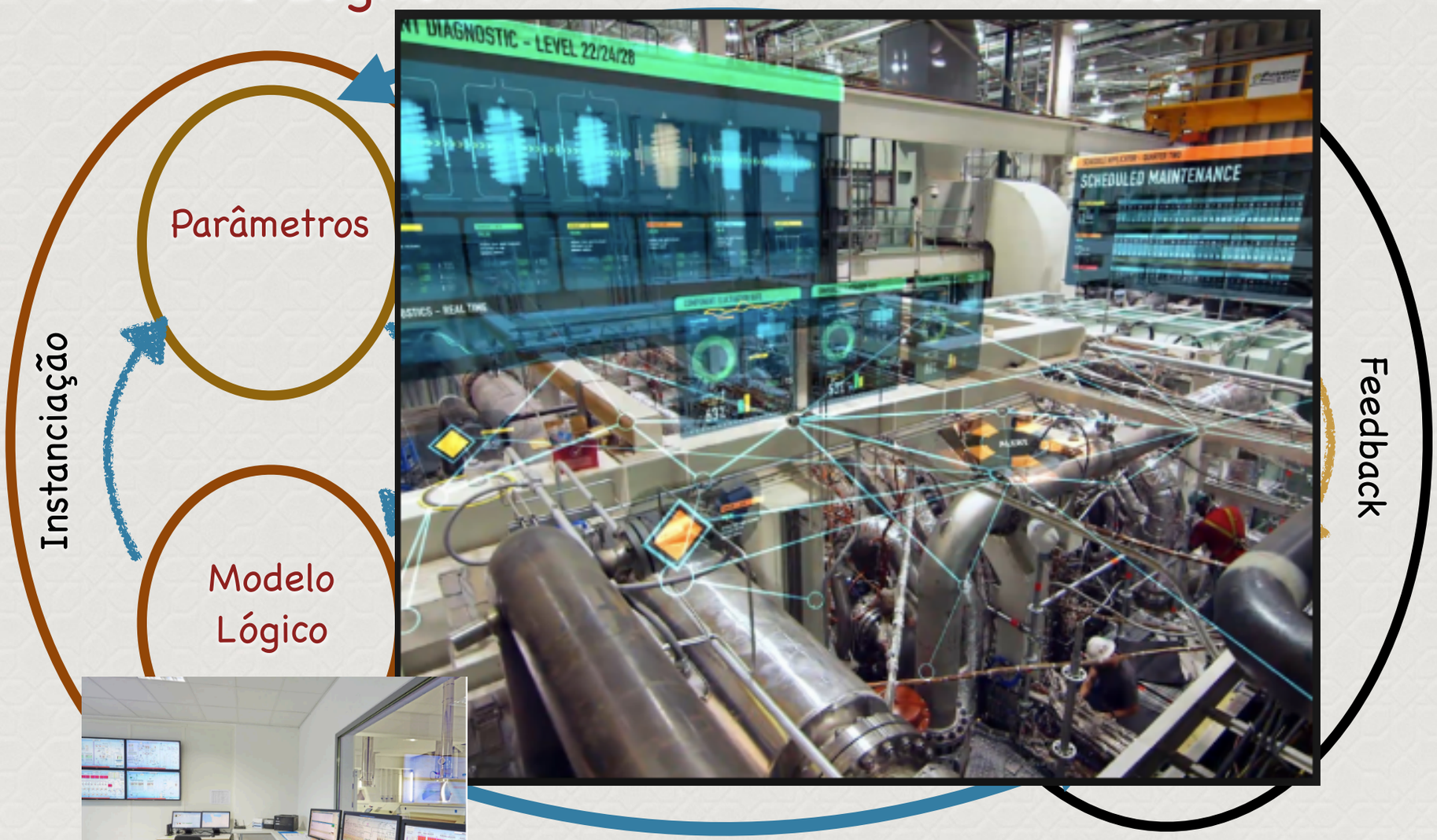
Será que este programa está correto... e completo?

	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
lampadas(acesas).
true
?-
lampadas(acesas).
Examples
History
Solutions
table results
Run!

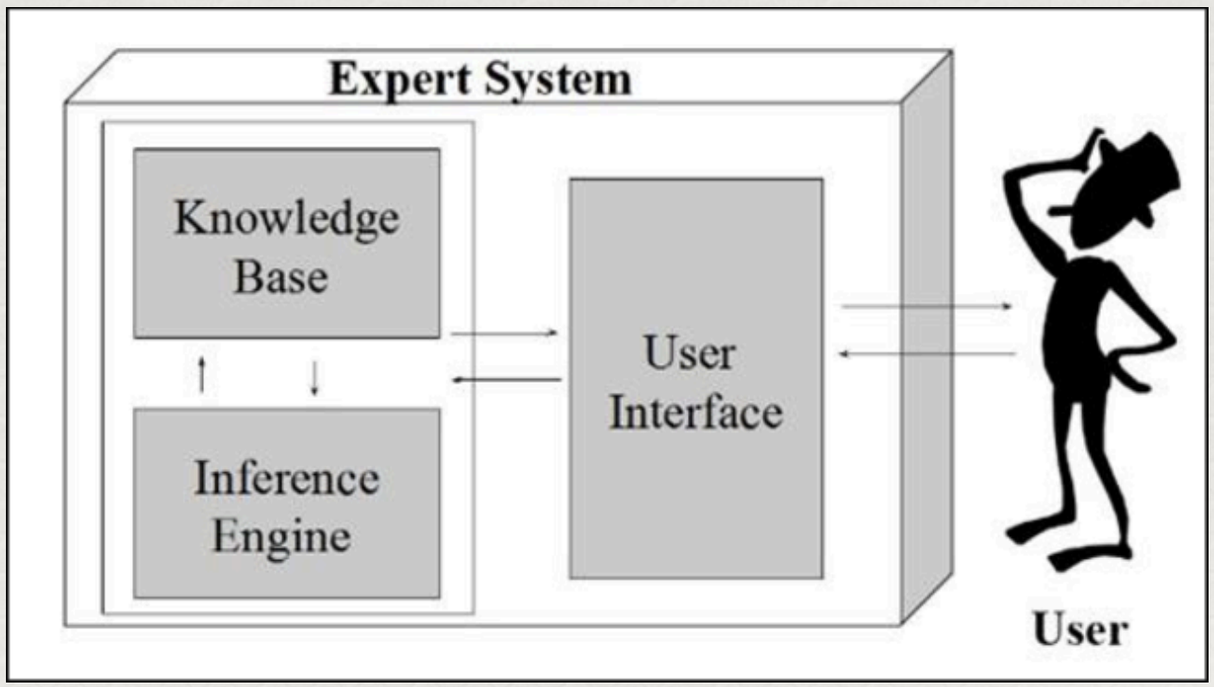


Virtualização em IA

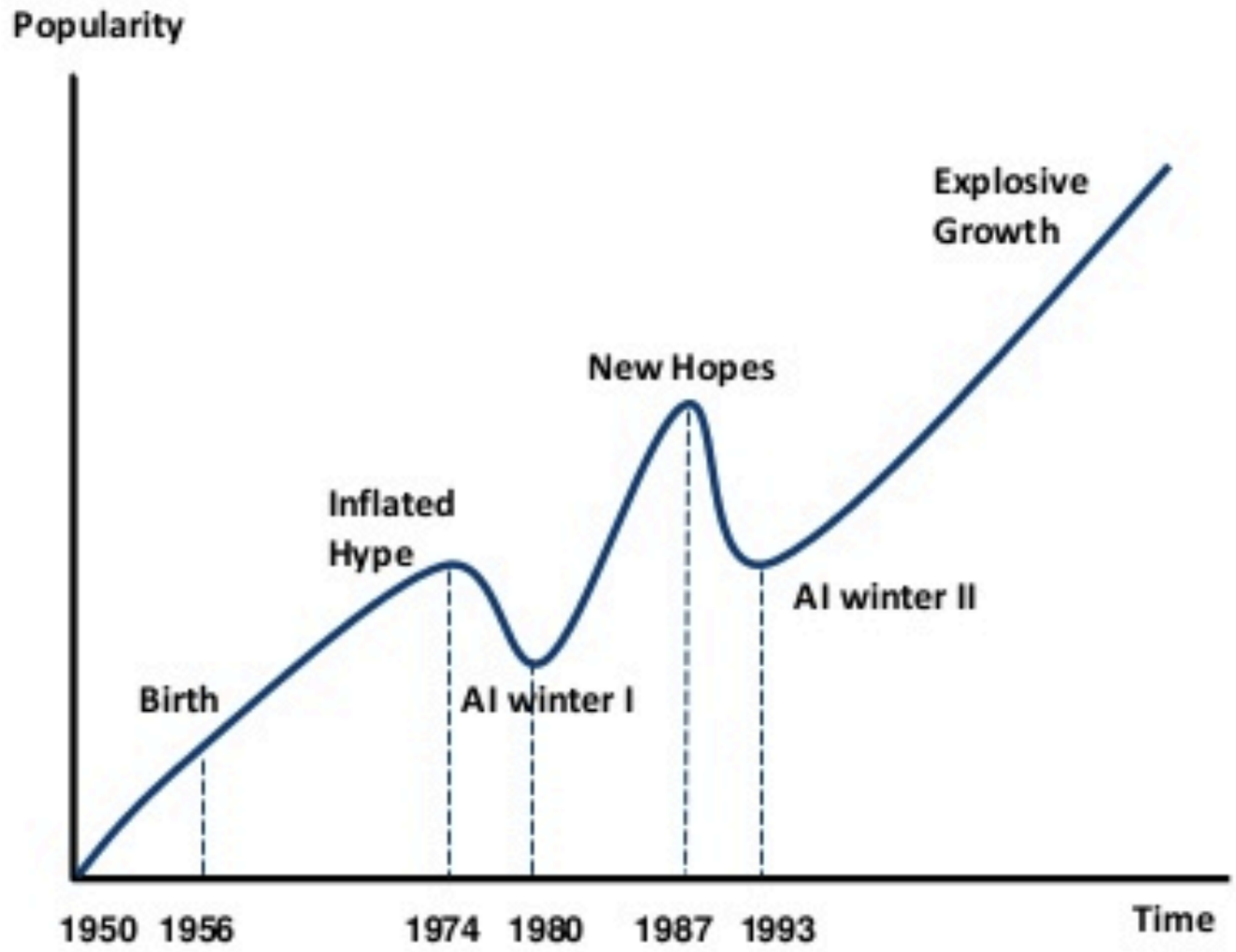


Virtualização em IA

Edward Feigenbaum

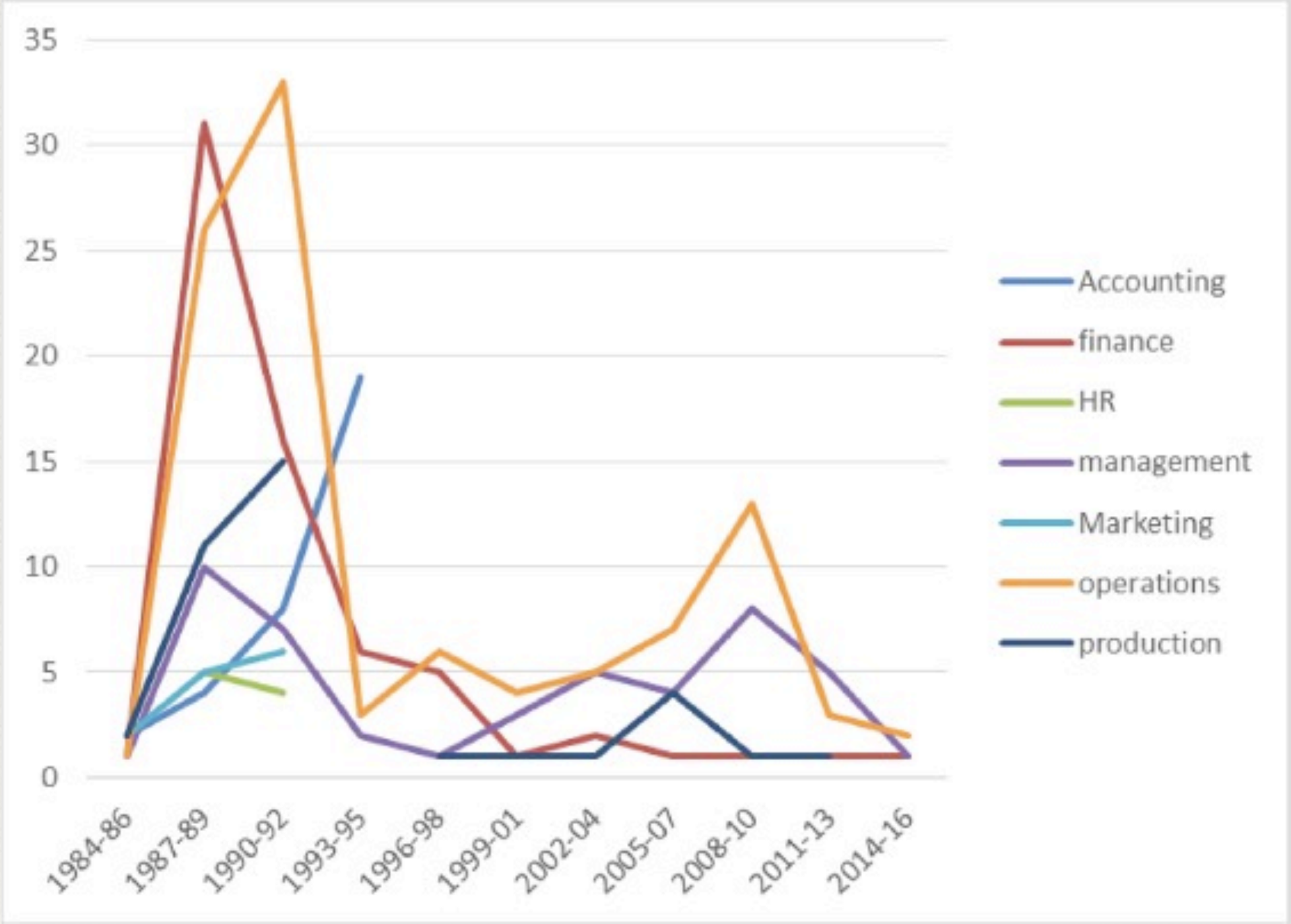


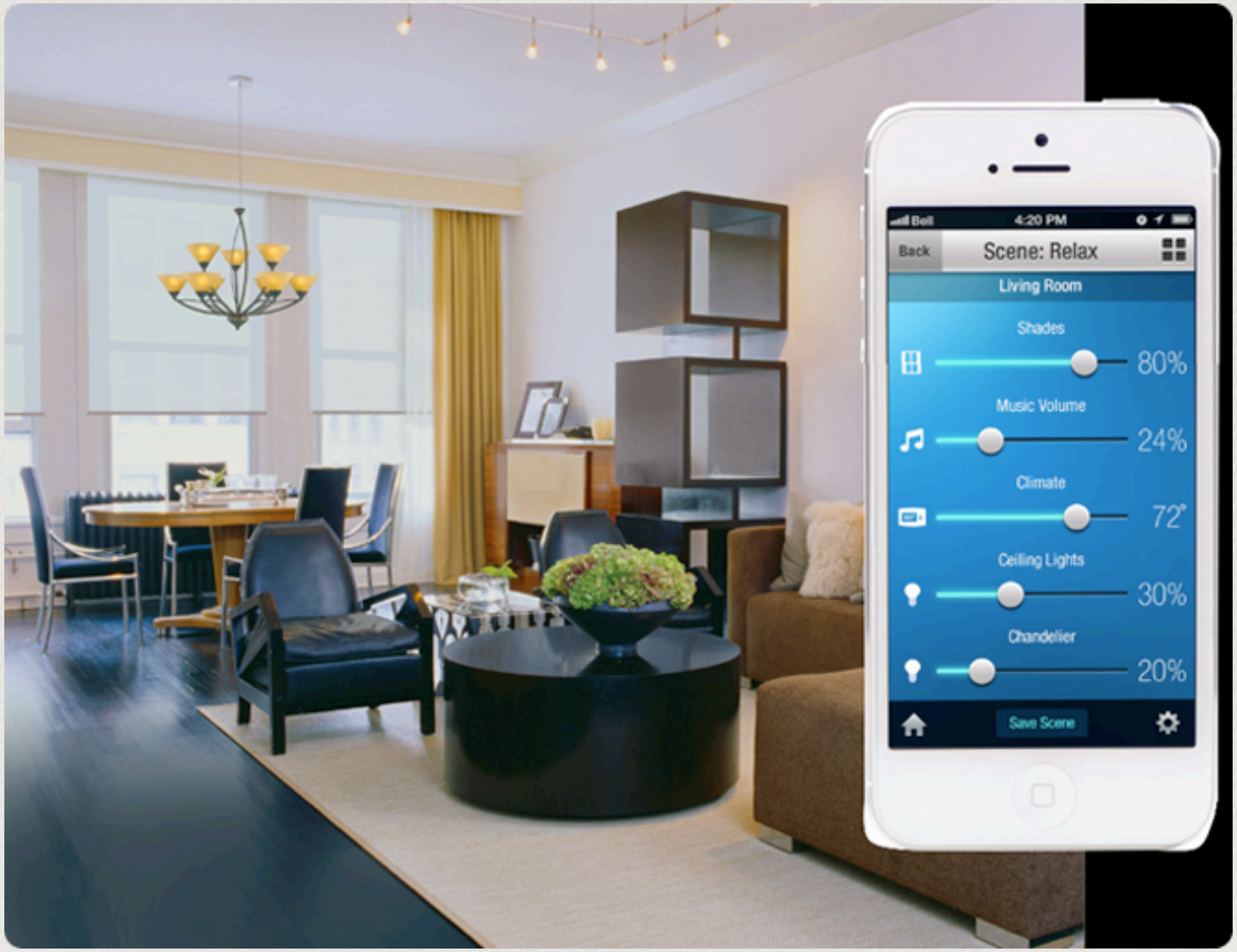
AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"…

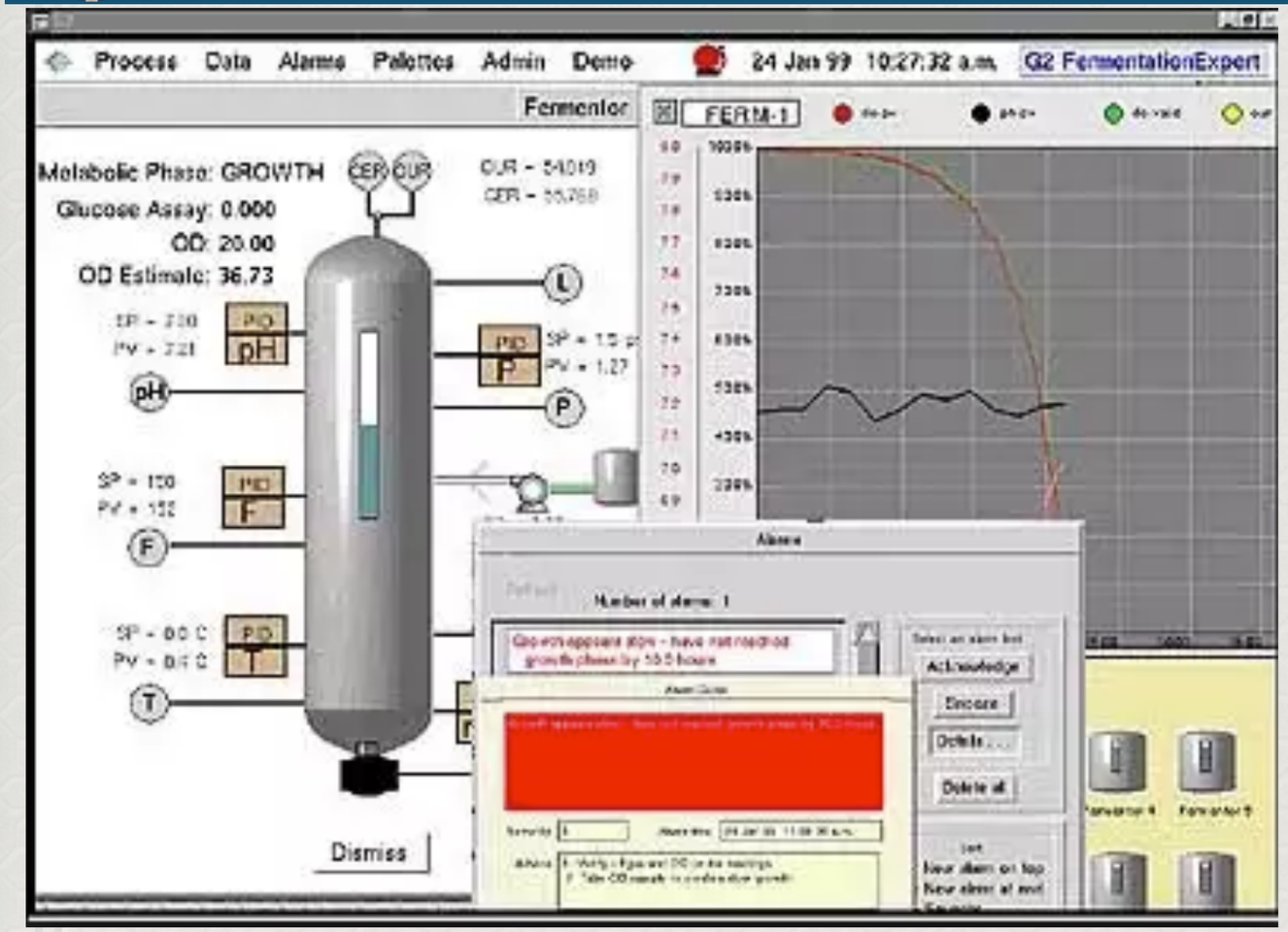


Timeline of AI Development
▪ **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
▪ **1970s**: AI winter I
▪ **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
▪ **1990s**: AI winter 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


35
30
25
20
15
10
5
0
1984-86
1987-89
1990-92
1993-95
1996-98
1999-01
2002-04
2005-07
2008-10
2011-13
2014-16
Accounting
finance
HR
management
Marketing
operations
production


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

Falamos na aula passada que vocês deveriam se organizar para fazer o trabalho final do curso. Mas é preciso também se organizar para o primeiro trabalho que será a construção de um programa Prolog ligado a busca e resolução de problemas. O problema está ligado ao "mundo de blocos" trata-se de prover uma "inteligência" que poderá ser conectada a um braço robótico e usar o método clássico STRIPS.

Até a próxima aula!