Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 7- O Frame Problem e a modelagem do dominio no STRIPS

Prof. José Reinaldo Silva

reinaldo@usp.br

d-l@b

Trabalho em grupo

Os grupos estão já definidos. A configuração é a seguinte:

Grupo 1:

Fernando Vicente Grando Monteiro 8992919

Marcos Menon José 8989112

Sverker Fabian Hugert 11462480

Vitor Augusto Martin 8993100

Grupo 2:

David Calil Spindola Pedro - 8989384

Diego Augusto Vieira Rodrigues - 8989276

Felipe Cominato Nemr - 9345662

Guilherme Sugahara Faustino - 9348971

Grupo 3:

Lucas Hideki Sakurai 8989193

Lucas Pereira Cotrim 8989092

Matheus Torres Guinezi 9345679

Monize Bessa Arabadgi 7961944

Renan Masashi Yamaguchi 8989151

Grupo 4:

Alexandre Zamora Zerbini Denigres - 8583072

Dylan Kim Heleno - 8586072

Henrique Yda Yamamoto - 9349502

Vinicius Augusto Carnevali Miquelin - 8988410

Vinicius Takiuti Miura - 9345874

Grupo 5:

Guilherme Dello Russo - 9345895

Nathan Géraud PERRIN- 10935360

Natália Thoma Ricardo - 9344806

Grupo 6:

Beatriz Santin de Araujo Pinho - 8533851

Bianca Faria Silva - 8991599

Bruna Sayuri de Souza Suzuki - 7987501

Murillo José Almeida Faria de Oliveira - 9436785

Grupo 7:

Daniel Tsutsumi - 9349005

Gabriel Pinto - 8988017

Juliana Lopes - 8512600

Kaio Takase - 9345690

Matheus Ramalho - 9345710

Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será dividido em "milestones":

Setembro 2019

webcid.com.br

Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

7: Independência do Brasil 22: Início da primavera
06 - Quarto Crescente 14 - Lua Cheia 21 - Quarto Minguante 28 - Lua Nova

Outubro 2019

webcid.com.br

Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

20: Início do horário de verão 12: Nsa. Sra. Aparecida 15: Dia dos Professores
05 - Quarto Crescente 13 - Lua Cheia 21 - Quarto Minguante 28 - Lua Nova

25/09 - início do processo, definição do domínio
02/10 - definição do algoritmo e implementação
09/10 - entrega do programa final
16/10 - competição

Cronograma de entrega do trabalho em grupo

O segundo trabalho consiste em procurar uma aplicação real de mercado que usa Inteligência Artificial clássica (não machine learning que é objeto de outra disciplina). O trabalho do grupo é avaliar esta aplicação tecnicamente, modelos aplicados, algoritmos de busca, classificação como sistema especialista, etc. uso de computação evolutiva (algoritmo genético, fuzzy logic...), problemas de implementação, uso e manutenção, satisfação do usuário com o sistema, e finalmente, desempenho. O cronograma de milestones é o seguinte

06/11 - texto (pdf) apresentando a aplicação

13/11 - resumo da análise completa

27/11 - apresentação do trabalho final

webcid.com.br

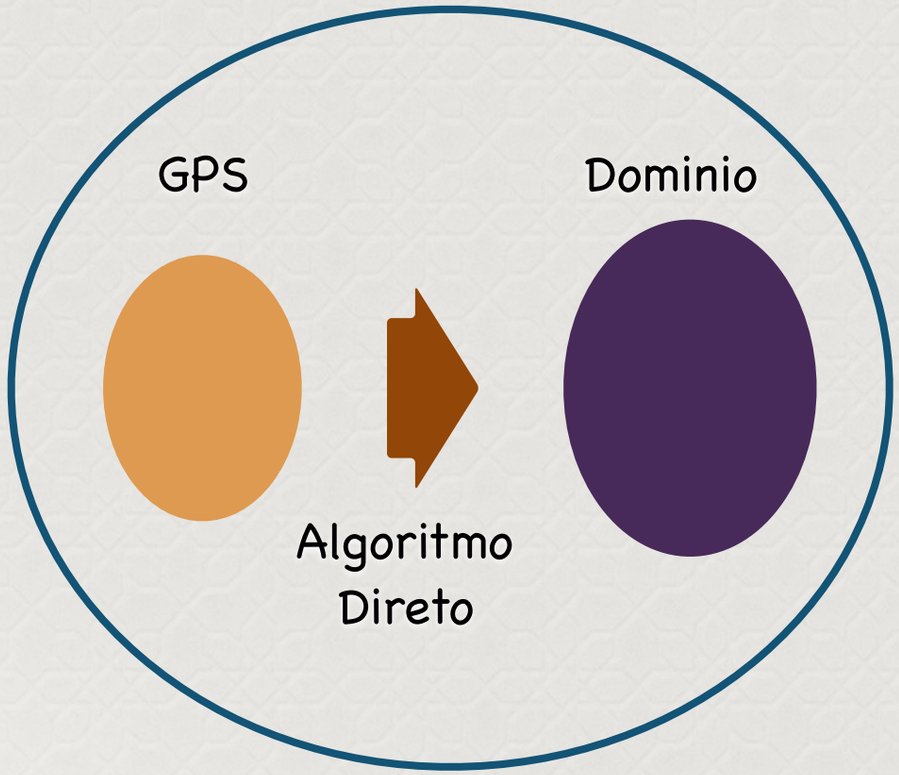
Dezembro 2019

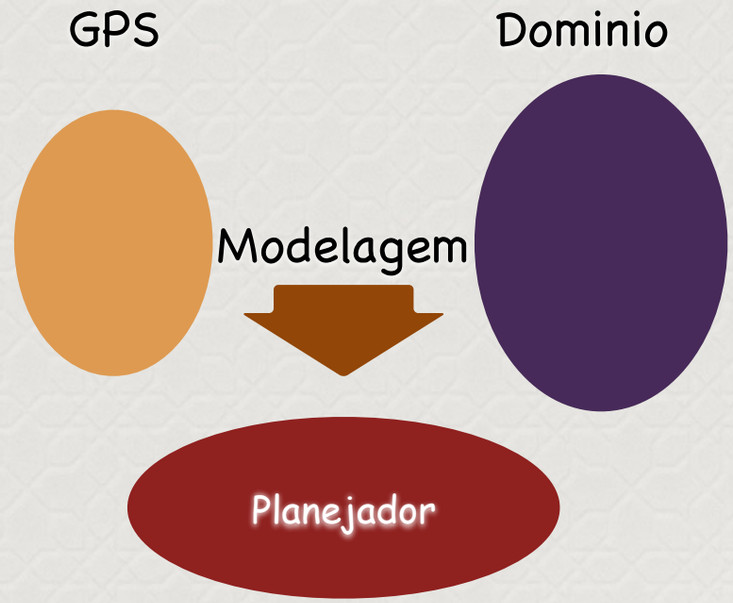
Domingo	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

21: Início do verão 25: Natal

04 - Quarto Crescente 12 - Lua Cheia 19 - Quarto Minguante 26 - Lua Nova

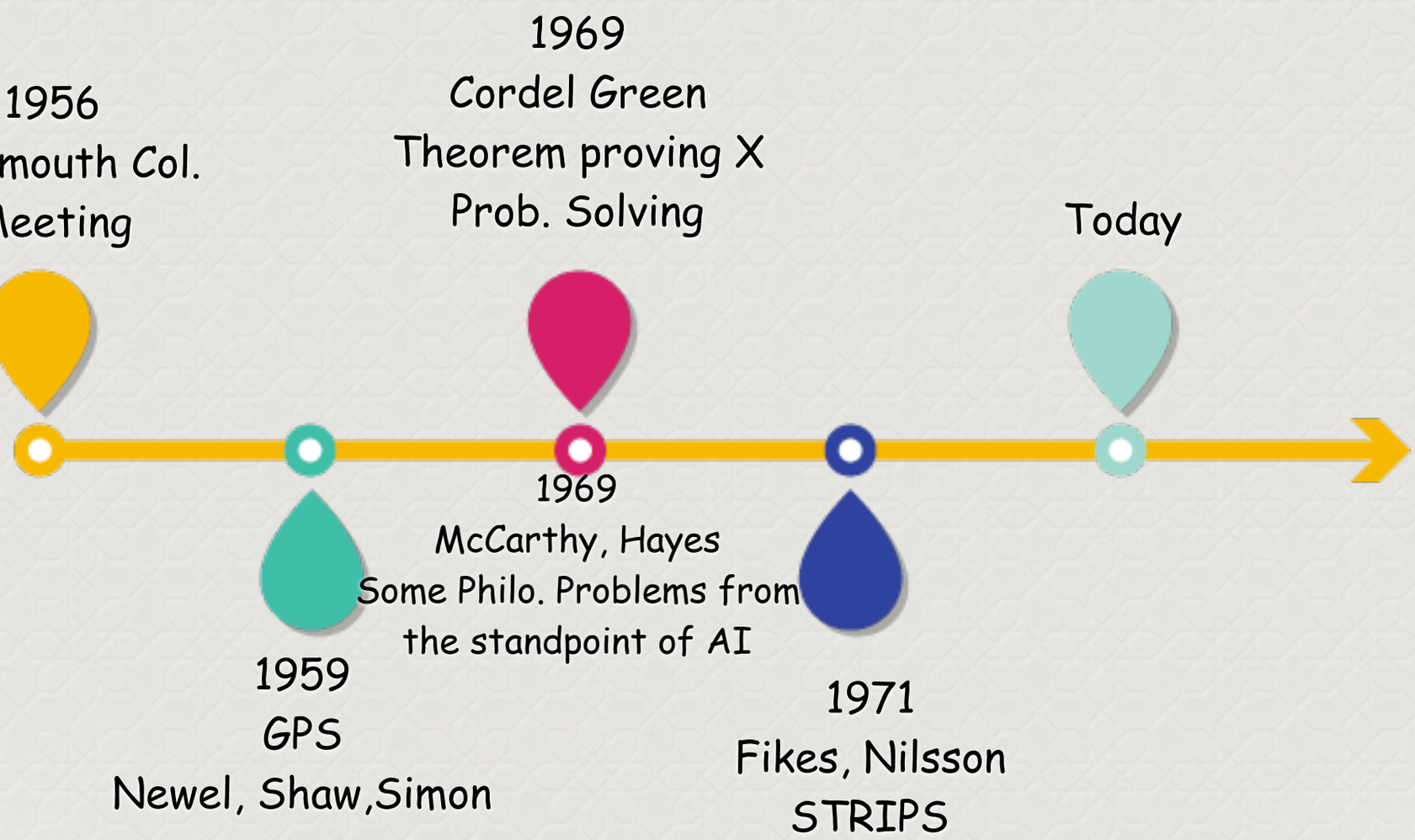
Na aula passada falamos sobre o STRIPS e consequentemente sobre "planning" e concluímos que algoritmos de resolução de problemas baseados no SRIPS podem ser de dois tipos:





Planejador genérico

"Line up" sobre resolução de problemas



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

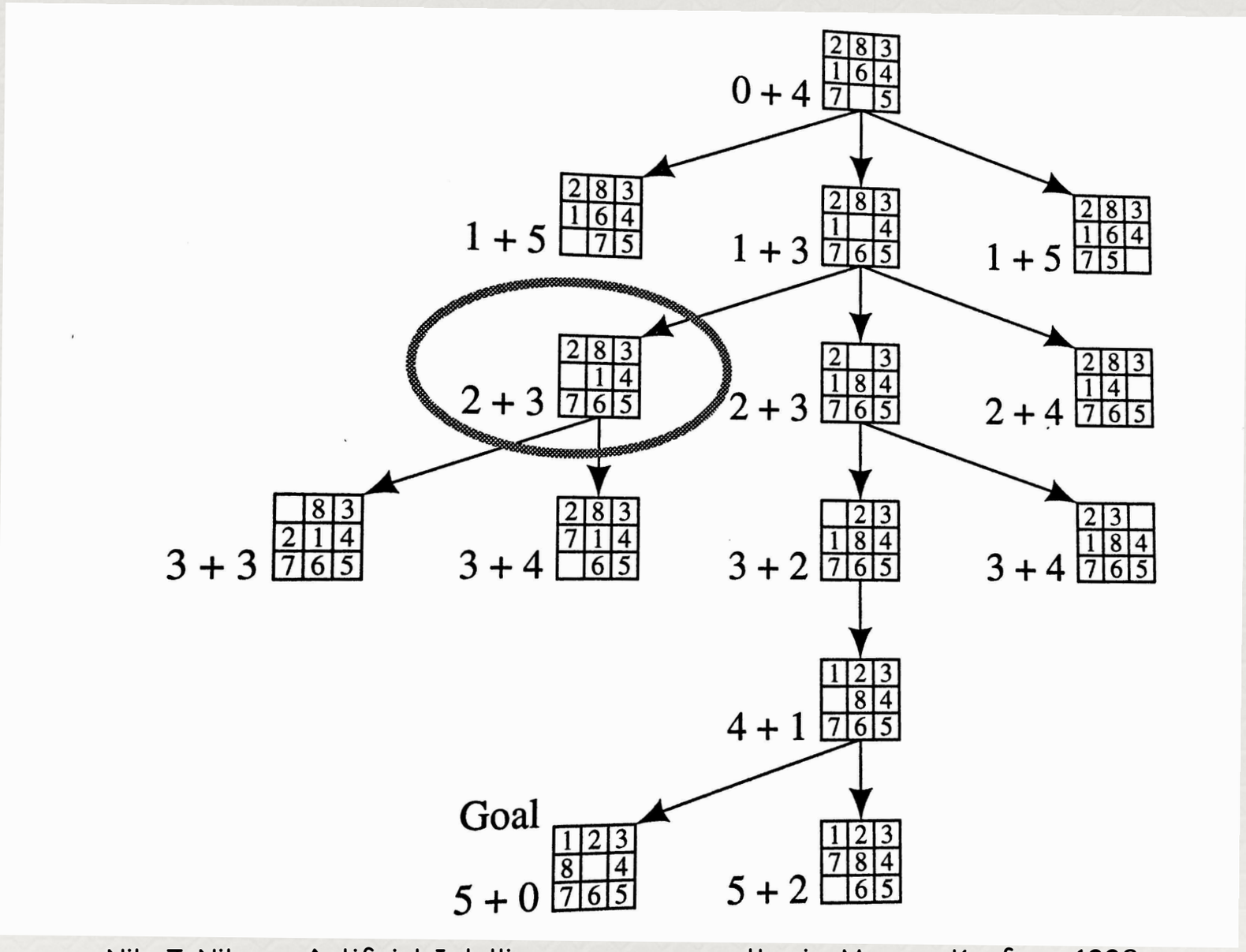
Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

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

This framework for problem solving has been central to much of the research in artificial intelligence [1]. Our primary interest here is in the class of problems faced by a robot in re-arranging objects and in navigating, i.e., problems that require quite complex and general world models compared to those needed in the solution of puzzles and games. In puzzles and games, a simple matrix or list structure is usually adequate to represent a state of the problem. The world model for a robot problem solver, however, must include a large number of facts and relations dealing with the position of the robot and the positions and attributes of various objects, open spaces, and boundaries. In STRIPS, a world model is represented by a set of well-formed formulas (wffs) of the first-order predicate calculus.

Árvore de busca e a busca informada



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

Um programa em lógica de predicados (ou em cláusulas de Horn), não consegue capturar "mudança de estado".

APPLICATION OF THEOREM PROVING TO PROBLEM SOLVING*

Cordell Green
Stanford Research Institute
Menlo Park, California

Abstract

This paper shows how an extension of the resolution proof procedure can be used to construct problem solutions. The extended proof procedure can solve problems involving state transformations. The paper explores several alternate problem representations and provides a discussion of solutions to sample problems including the "Monkey and Bananas" puzzle and the "Tower of Hanoi" puzzle. The paper exhibits solutions to these problems obtained by QA3, a computer program bused on these theorem-proving methods. In addition, the paper shows how QA3 can write simple computer programs and can solve practical problems for a simple robot.

Key Words: Theorem proving, resolution, problem solving, automatic programming, program writing, robots, state transformations, question answering.

Automatic theorem proving by the resolution proof procedure § represents perhaps the most powerful known method for automatically determining the validity of a statement of first-order logic. In an earlier paper Green and Raphael" illustrated how an extended resolution procedure can be used as a question answerer—e.g., if the statement (3x)P(x) can be shown to follow from a set of axioms by the resolution proof procedure, then the extended proof procedure will find or construct an x that satisfies P(x). This earlier paper (1) showed how one can axiomatize simple question-answering subjects, (2) described a question-answering program called QA2 based on this procedure, and (3) presented examples of simple question-answering dialogues with QA2 . In a more recent paper" the author (1) presents the answer construction method in detail and proves its correctness, (2) describes the latest version of the program, QA3, and (3) introduces state-transformation methods into the constructive proof formalism. In addition to the question-answering applications illustrated in these earlier papers, QA3 has been used as an SRI robot[4] problem solver and as an automatic program writer. The purpose of this paper is twofold: (1) to explore the question of predicate calculus representation for state-transformation problems in general, and (2) to elaborate upon robot and program-writing applications of this approach and the mechanisms underlying them.

Exactly how one can use logic and theorem proving for problem solving requires careful thought on the part of the user. .Judging from my experience, and that of others using QA2 and QA3, one of the first difficulties encountered is the representation of problems, especially state-transformation problems, by statements in formal logic. Interest has been shown in seeing several detailed examples that illustrate alternate methods of axiomatizing such problems—i.e., techniques for "programming" in first-order logic. This paper provides detailed examples of various methods of representation. After presenting methods in Sees. I and II, a solution to the classic "Monkey and Bananas" problem is provided in Sec. III. Next, Sec. IV compares several alternate representations for the "Tower of Hanoi puzzle. Two applications, robot problem solving and automatic programming, are discussed in Sees. V and VI, respectively.

1. An Introduction to State-Transformation Methods

The concepts of states and state transformations have of course been in existence for a long time, and the usefulness of these concepts for problem solving is well known. The purpose of this paper is not to discuss states and state transformations as such, but instead to show how these concepts can be used by an automatic resolution theorem prover. In practice, the employment of these methods has greatly extended the problem-solving capacity of QA2 and QA3. McCarthy and Hayes present a relevant discussion of philosophical problems involved in attempting such formalizations .

First we will present a simple example. We begin by considering how a particular universe of discourse might be described in logic.

Facts describing the universe of discourse

4. Green, C. Application of theorem proving to problem solving. *Proc. Int'l. Joint Conf. Artificial Intelligence*, Washington, D.C. (May 1969).

In artificial intelligence, the **frame problem** describes an issue with using first-order logic (FOL) to express facts about a robot in the world. In the logical context, actions are typically specified by what they change, with the implicit assumption that everything else (the **frame**) remains unchanged.

"Line up" sobre resolução de problemas

1956
Dartmouth Col.
Meeting

1959
GPS
Newel, Shaw,Simon

1969
Cordel Green
Theorem proving X
Prob. Solving

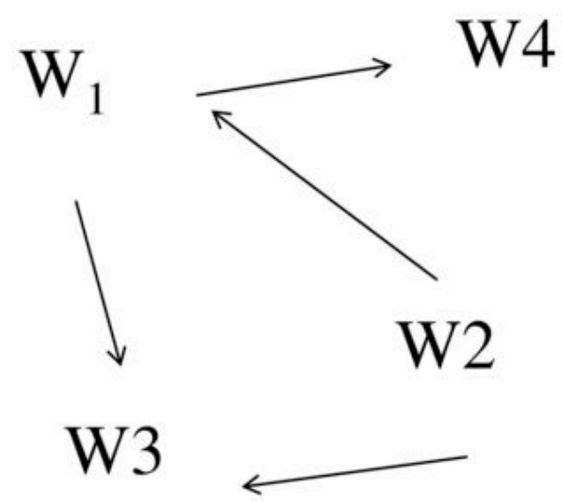
1969
McCarthy, Hayes
Some Philo. Problems from
the standpoint of AI

1971
Fikes, Nilsson
STRIPS

Today

Kripke Semantics of Modal Logic

- The "universe" seen as a collection of worlds.
- Truth defined "in each world".
- Say U is the universe.
- I.e. each w ε U is a prepositional or predicate model.

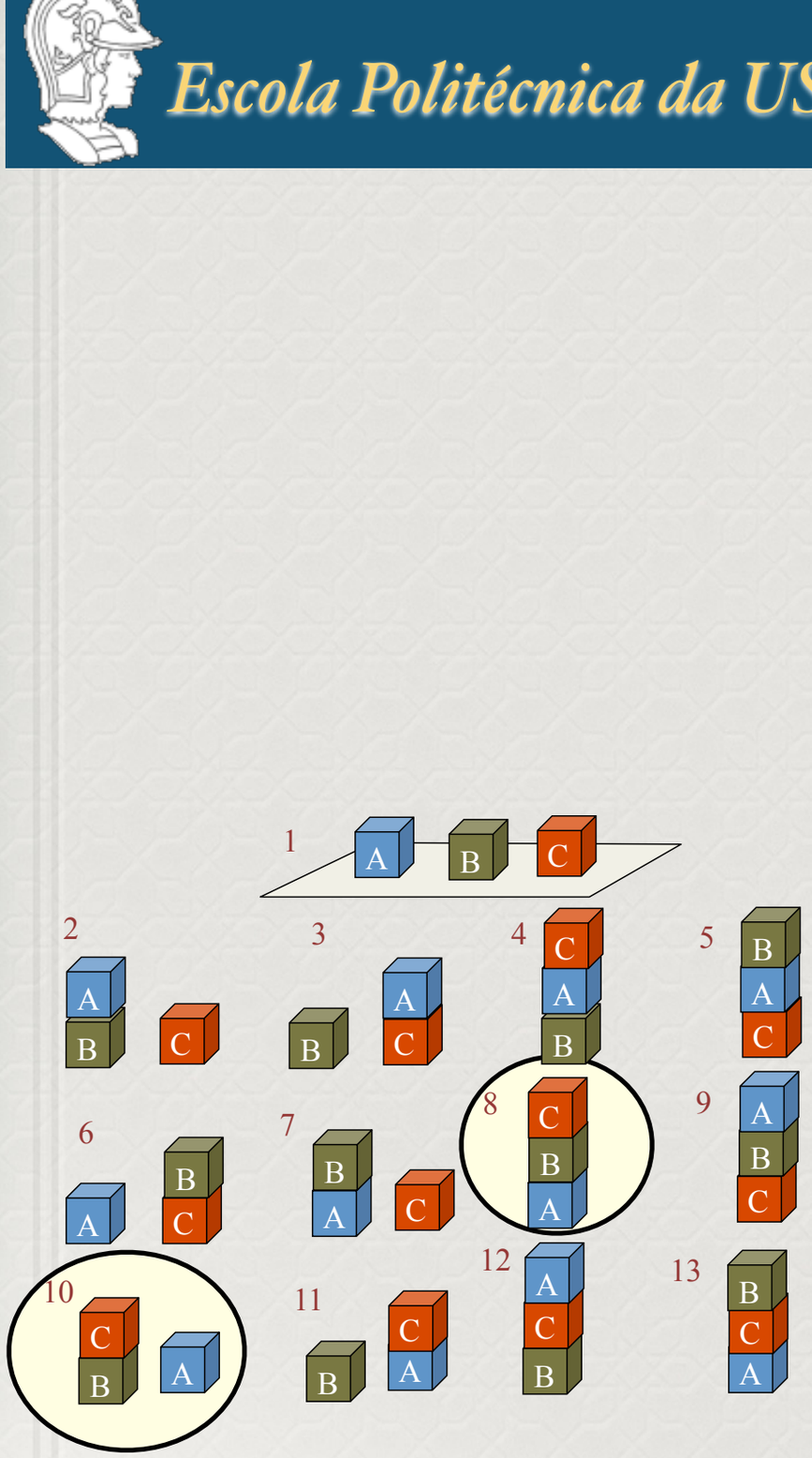


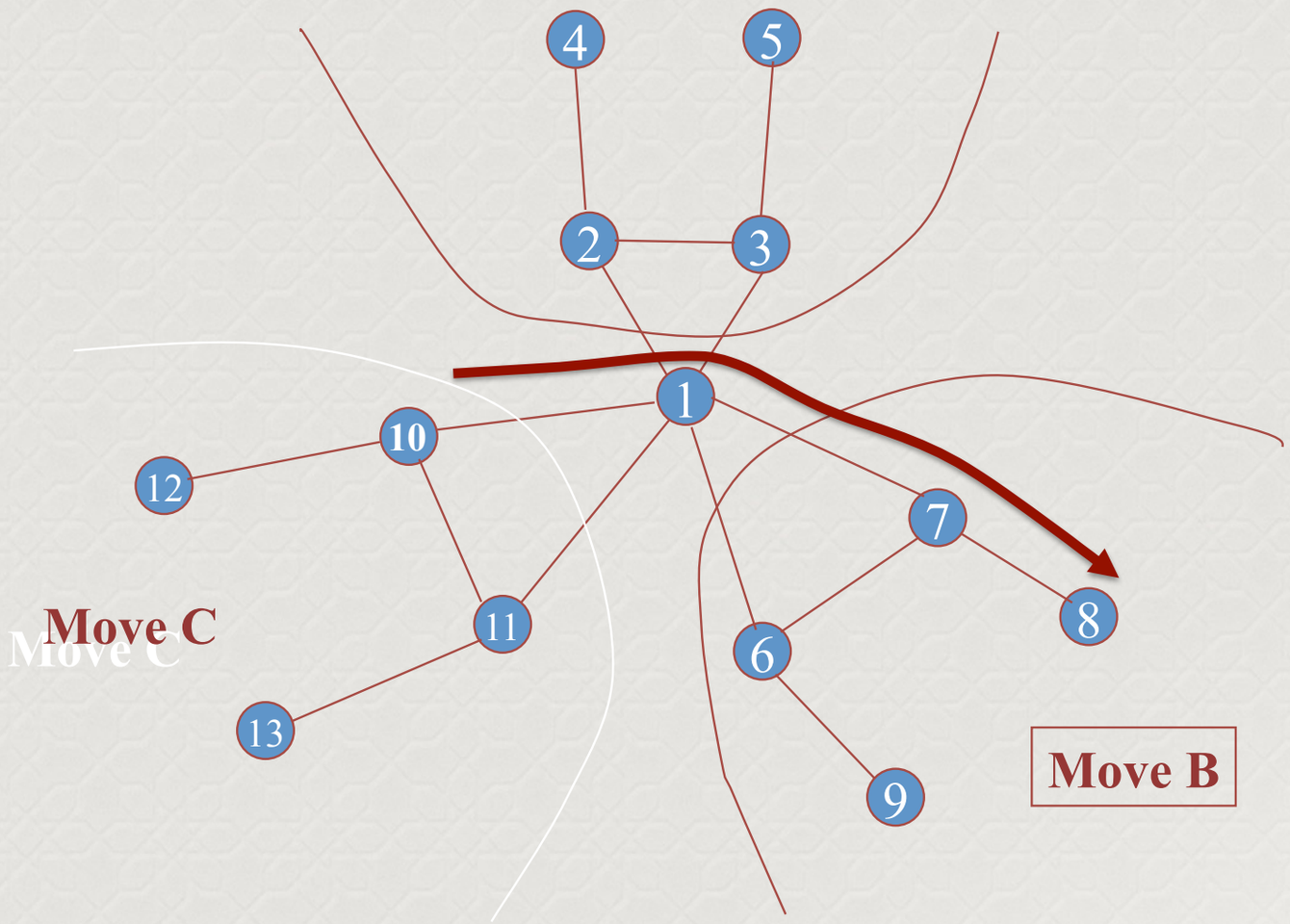
SWE 623 Duminda Wijesekera 3

Saul Kripke

Kripke Models for Propositional Modal Logic

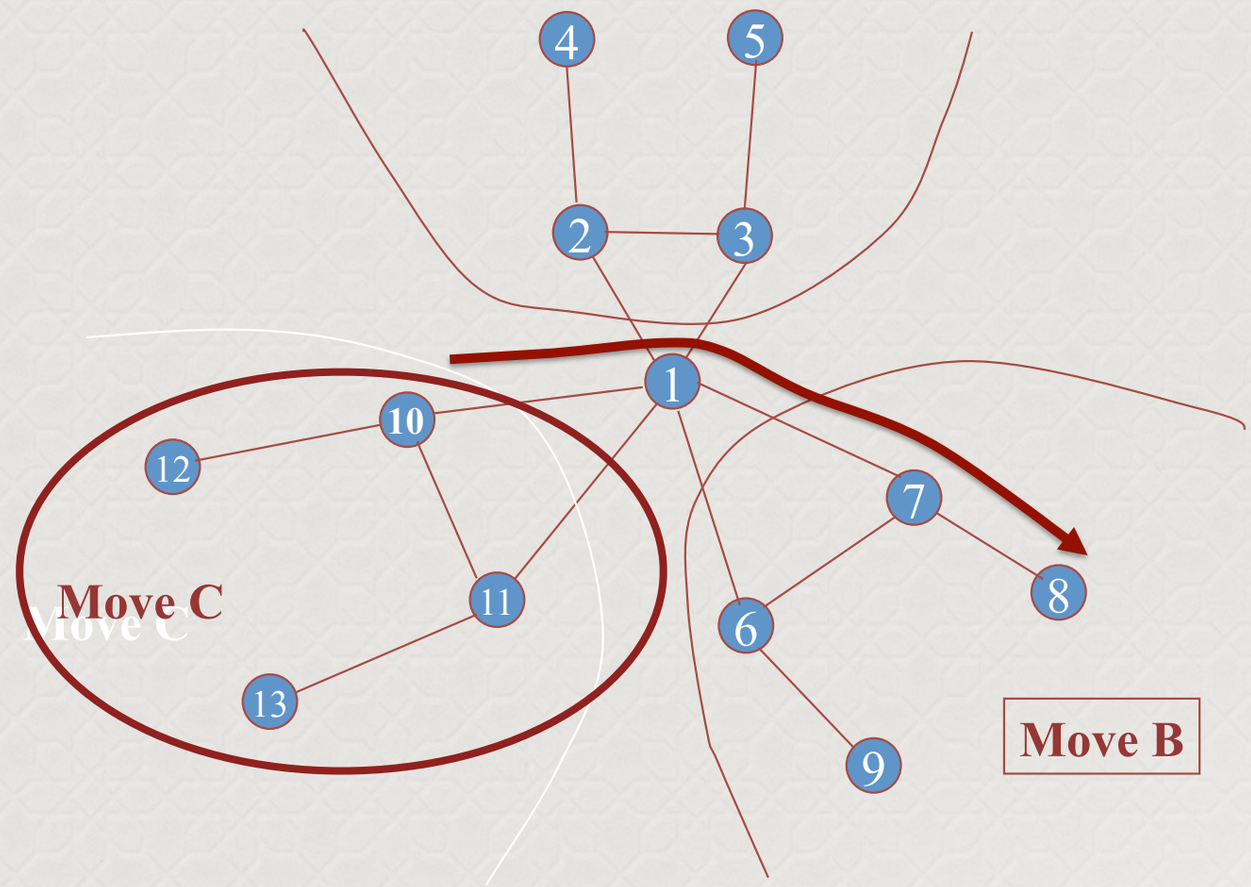
- A model <W,R,h> consists of:
  - W: a set of worlds
  - R $\subseteq$ W $\times$ W: an accessibility relationship
    - W and R together is called a frame
  - h: W $\times$ P $\rightarrow$ {true, false}: a propositional truth-assignment function for each world
    - Thus, in a Kripke model, statements are true relative to whichever world they are being evaluated in.


1
A
B
C
2
3
4
5
6
7
8
9
10
11
12
13


4
5
2
3
1
10
12
7
11
8
6
13
9
Move C
Move C
Move B

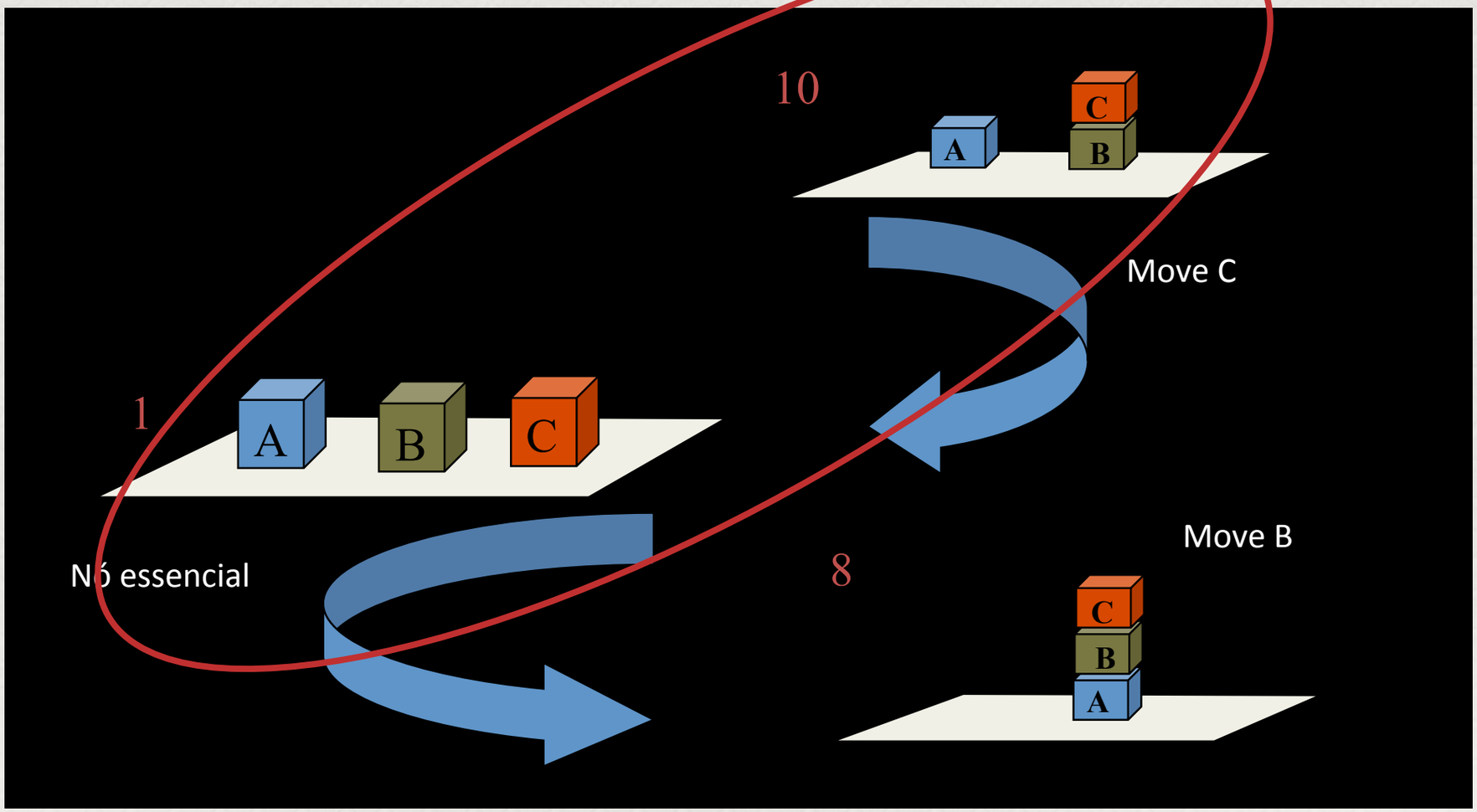
edge(12, 10)
edge(10,12)
edge(10,11)
edge(11,10)
edge(13,11)
edge(11,13)

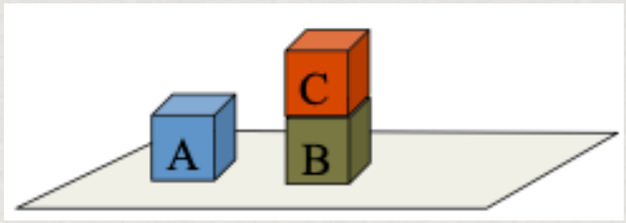
	10	11	12	13
10	0	1	1	0
11	1	0	0	1
12	1	0	0	0
13	0	1	0	0



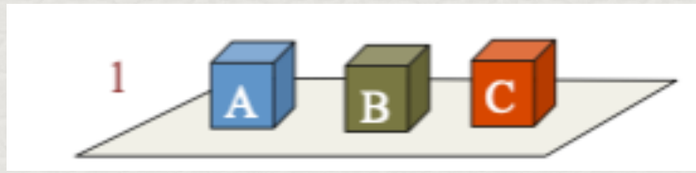
STRIPS Basic Concepts

Vamos usar novamente o problema modelo do mundo de blocos - que inspirou o STRIPS - para mostrar como opera a "máquina de inferência" ...





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



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

Pre-condição

livre(C)
sobre(C,X)

remove

move(C, mesa)

Pós-condição

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

add

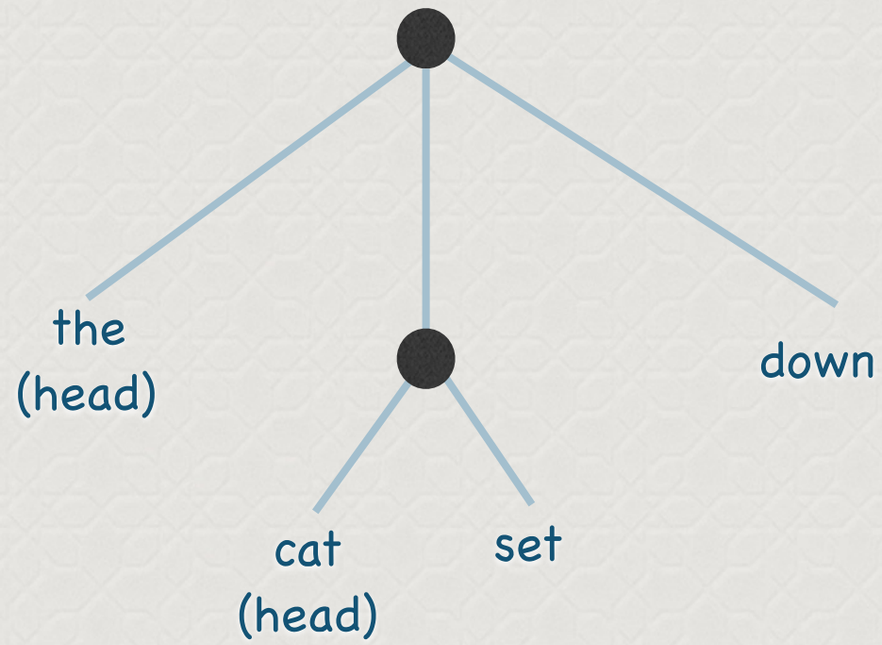
Entretanto... uma busca cega deve ser feita sobre uma estrutura

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.

estrutura

Em prolog...

[the, [cat, set], down]



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

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

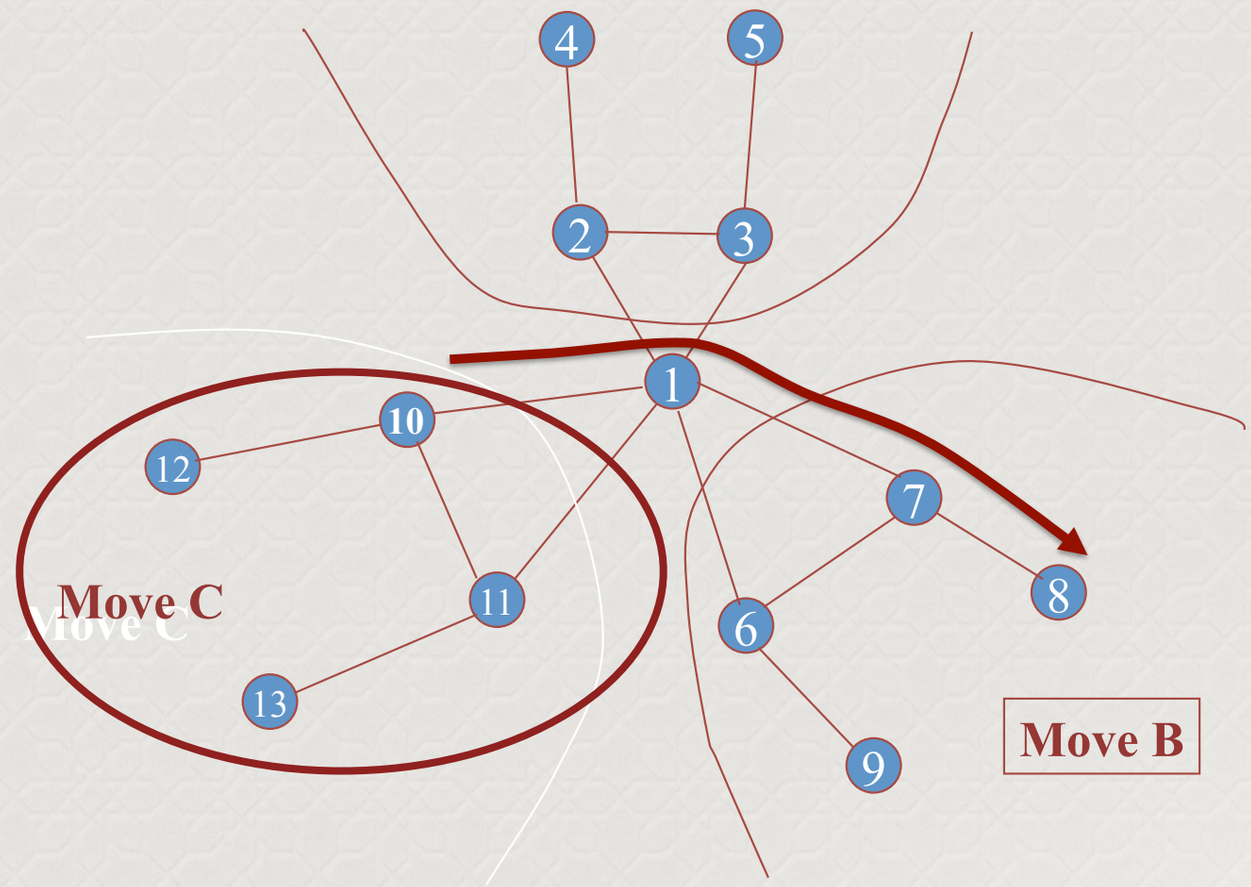
operador interno append/3

Uma operação básica com listas é juntar duas listas em uma terceira lista.
O operador append/3 pode fazer isso facilmente...

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

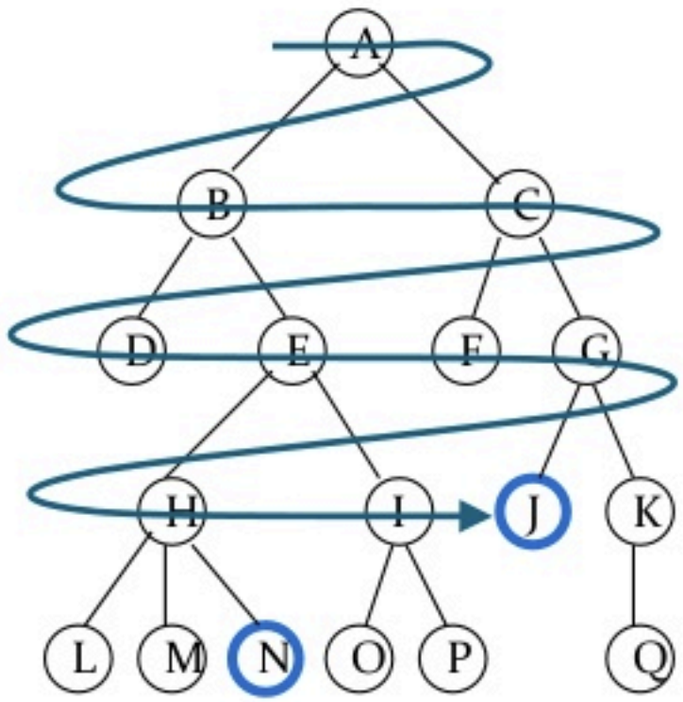
edge(12, 10)
edge(10,12)
edge(10,11)
edge(11,10)
edge(13,11)
edge(11,13)

	10	11	12	13
10	0	1	1	0
11	1	0	0	1
12	1	0	0	0
13	0	1	0	0



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

Search Techniques for Artificial Intelligence

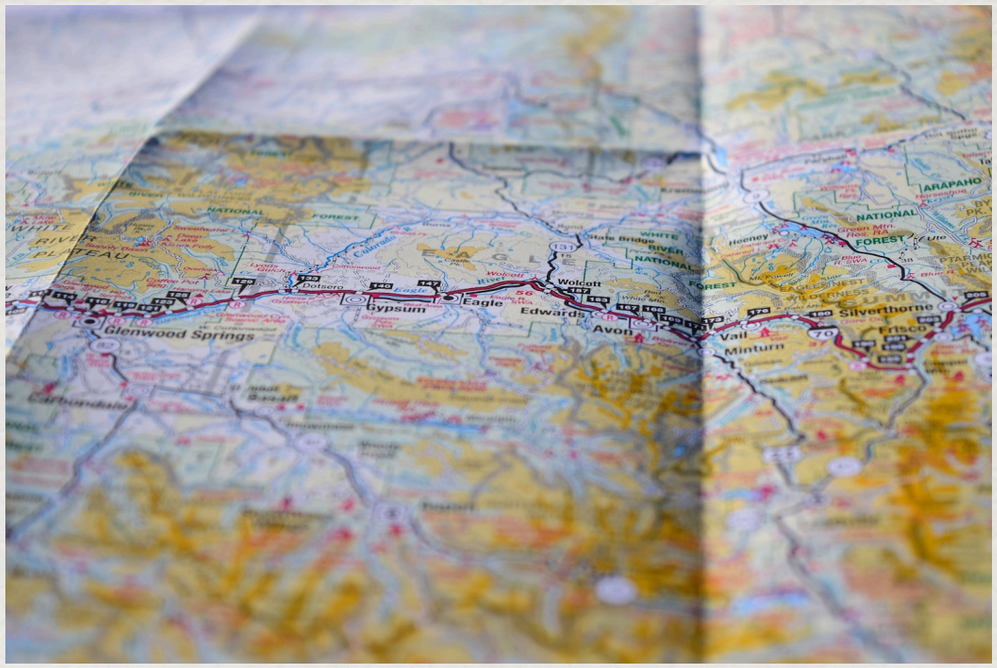
Search is a central topic in Artificial Intelligence. This part of the course will show why search is such an important topic, present a general approach to representing problems to do with search, introduce several search algorithms, and demonstrate how to implement these algorithms in Prolog.

- Motivation: Applications and Toy Examples
- The State-Space Representation
- Uninformed Search Techniques:
  - Depth-first Search (several variations)
  - Breadth-first Search
  - Iterative Deepening
- Best-first Search with the A* Algorithm

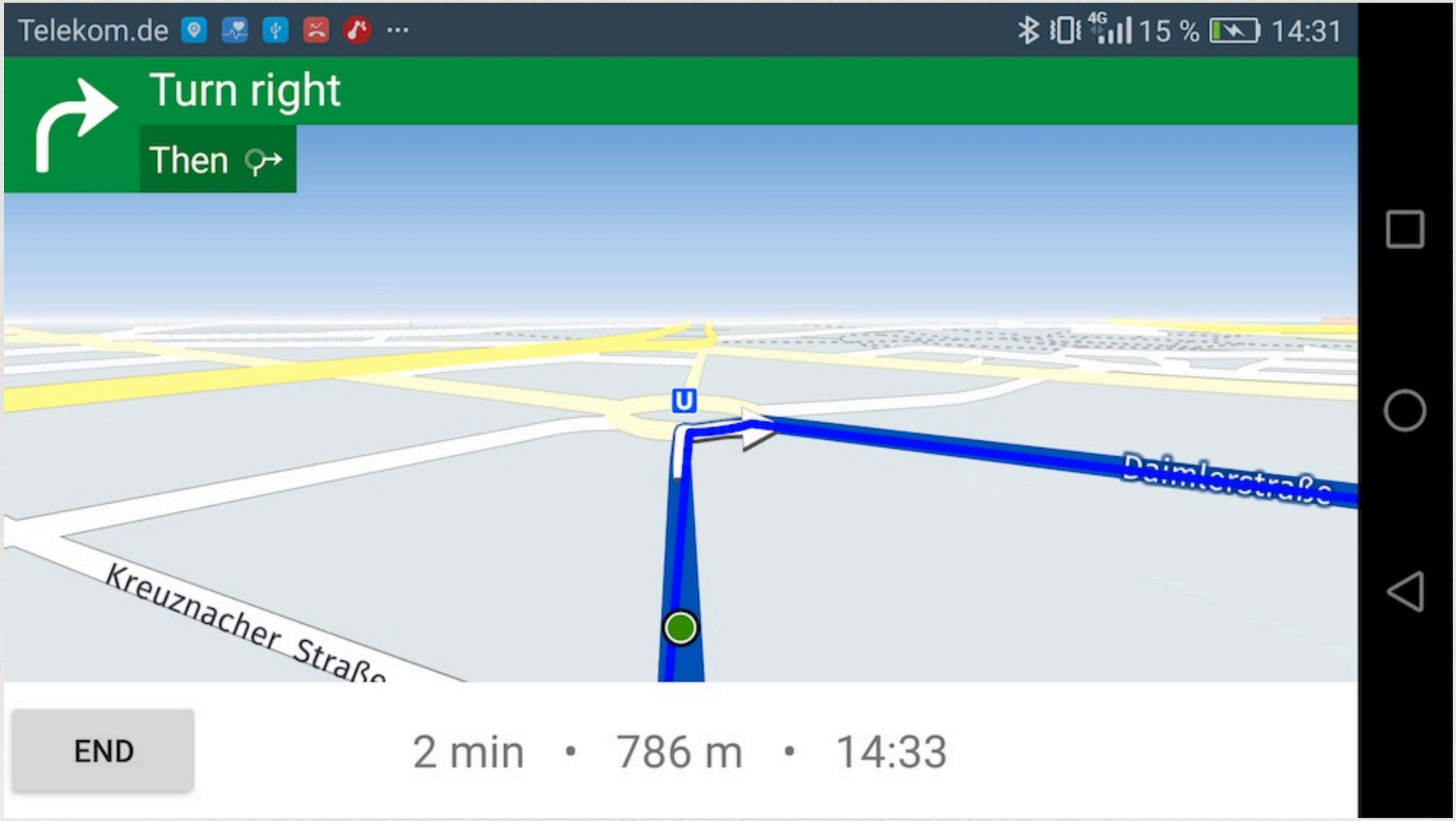
Exemplo de problemas que podem ser resolvidos com busca:

Route Planning

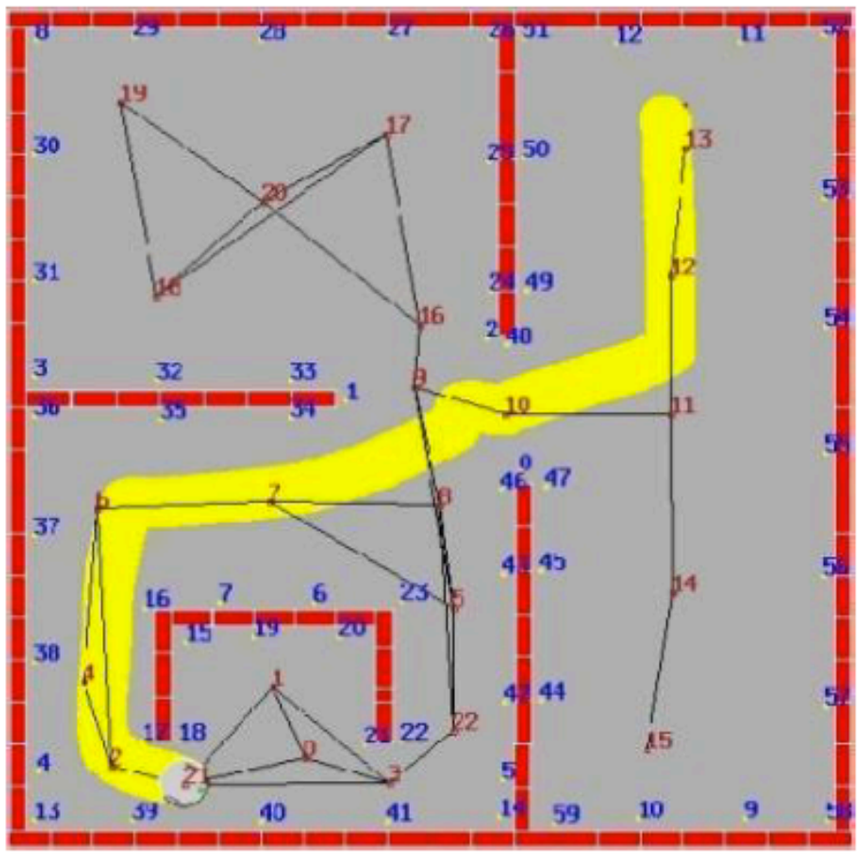
Aplicações práticas de "intelligent routing planning"



A medium-sized town somewhere in Germany, where residents separate their rubbish into glass, paper, food waste, plastic, old clothes and various other categories. All the bins are fitted with sensors to measure the fill level, and all the data is transmitted to a central server. We're talking here about thousands of bins. Millions, even, in large urban areas. Reliably transmitting signals, day in, day out. But who's supposed to process this information? The sheer amount of data generated would be way too much for even the brainiest human being to cope with.


Telekom.de
15 %
14:31
Turn right
Then
U
Daimlerstraße
Kreuznacher Straße
END
2 min • 786 m • 14:33

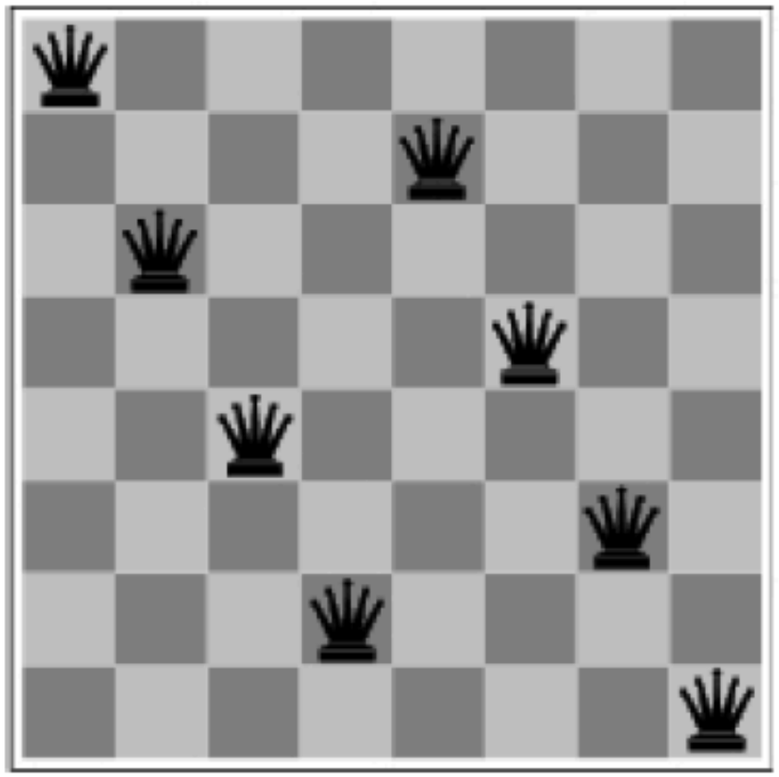
Robot Navigation



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

Eight-Queens Problem

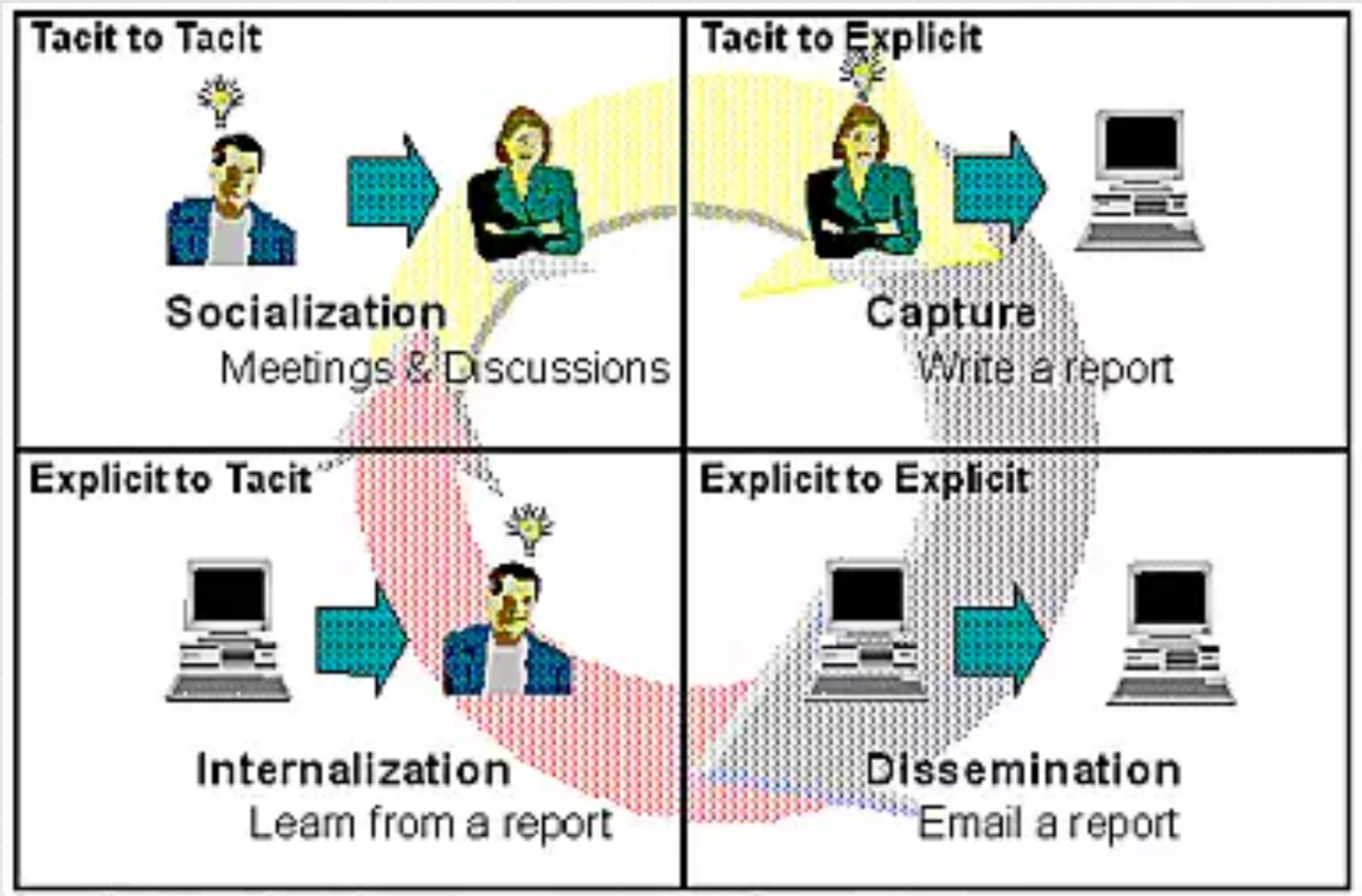
Arrange eight queens on a chess board in such a manner that none of them can attack any of the others!



Source: Russell & Norvig, *Artificial Intelligence*

The above is *almost* a solution, but not quite ...

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 uso de funções de avaliação

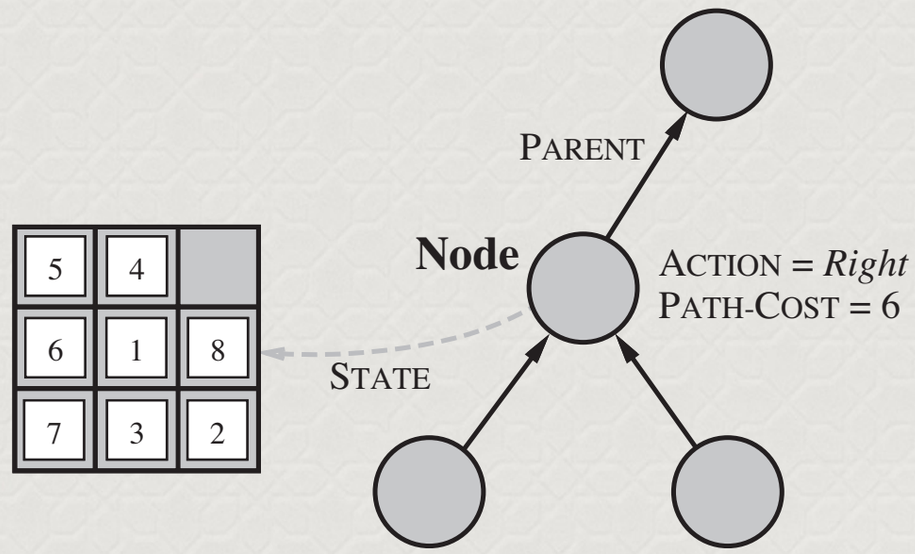
- Vamos assumir que existe uma função (de avaliação) heurística que pode ser usada para escolher o melhor nó para expandir a busca.

- Assim, podemos expandir um nó n que tenha o menor valor depois de aplicada a função de avaliação f(n). Os "empates" podem ser resolvidos arbitrariamente.

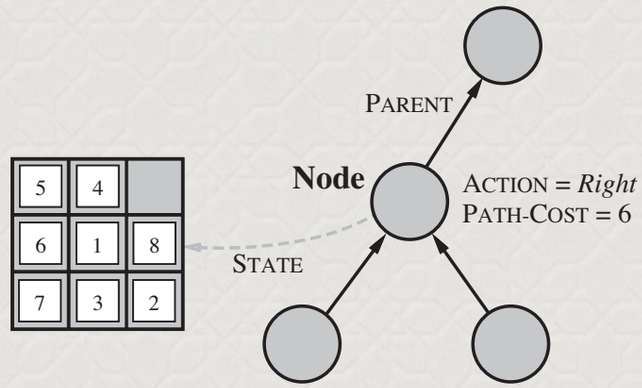
- A busca termina quando o nó a ser expandido for o nó objetivo.

A tabela de tiles

A tabela de tiles tem NOVE posições e o conteúdo de cada uma destas posições é um inteiro de zero a oito, onde o zero significa o tile para onde se pode mover os demais vizinhos.



A função de avaliação traduz portanto o fato que gostaríamos de encontrar uma solução "rápida", isto é, em baixa profundidade da árvore de busca, e que quanto mais nos aproximamos do objetivo mais "organizada" fica a tabela de tiles, isto é, o número de tiles fora do lugar diminui. Assim, a função de avaliação é



$$g(n) = p + h(n)$$

onde o fator p traduz a profundidade do nó n na árvore de busca e h(n) *é o número de tiles fora do lugar.*

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.

Best-first Search and Heuristic Functions

- For both *depth-first* and *breadth-first* search, which node in the search tree will be considered next only depends on the structure of the tree.
- The rationale in *best-first* search is to expand those paths next that seem the most "promising". Making this vague idea of what may be promising precise means defining *heuristics*.
- We fix heuristics by means of a *heuristic function* h that is used to *estimate* the "distance" of the current node n to a goal node:

$$h(n) \quad = \quad \textit{estimated cost from node } n \textit{ to a goal node}$$

Of course, the definition of h is highly application-dependent. In the *route-planning* domain, for instance, we could use the straight-line distance to the goal location. For the *eight-puzzle*, we might use the number of misplaced tiles.

Até a próxima aula!