Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

Aula 8- Algoritmo de busca A*

Prof. José Reinaldo Silva

reinaldo@usp.br

Cronograma de entrega do trabalho em grupo

O cronograma de trabalho será dividido em "milestones": **o primeiro trabalho consiste em desenvolver um programa em Prolog para resolver o mundo de blocos.**

25/09 - início do processo, definição do domínio

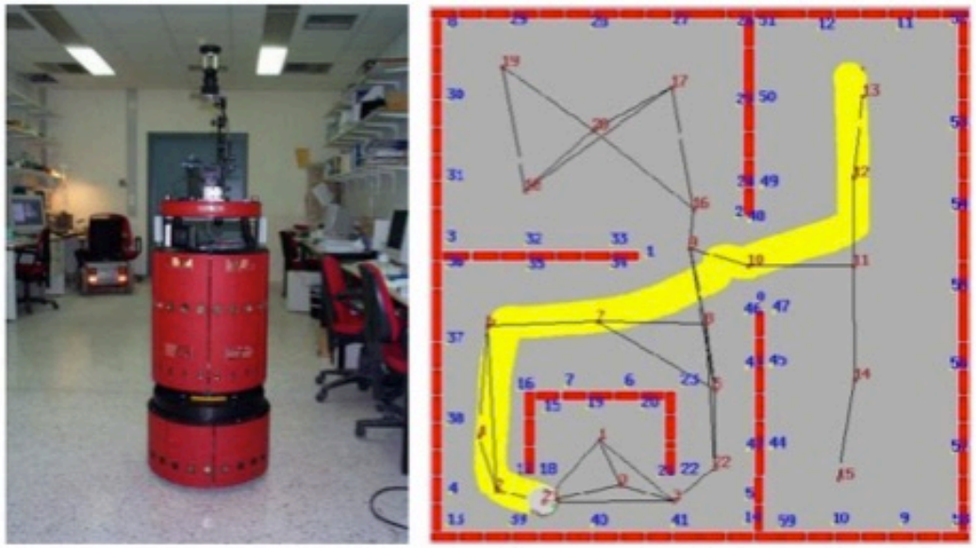
02/10 - definição do algoritmo e implementação

09/10 - entrega do programa final

16/10 - competição

O STRIPS e o Frame Problem

Editar



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

Vamos resumir o que vimos até aqui sobre os processos para resolver problemas genéricos usando Inteligência Artificial para ter uma noção mais precisa da evolução desta discussão desde a criação da IA, e de como esta abordagem dominou a cena nos principais centros de pesquisa. Veremos que o Frame Problem, tem, além de aspectos filosóficos uma relação bem próxima com a representação de problemas para robótica e, em especial, para usar a abordagem estado-transição. Espera-se que isso traga algum insight para o trabalho de resolver o mundo de blocos.

Começaremos a discussão pelo best-first que será detalhado na próxima aula.

- Search Techniques — Editar
- Aula7 — Editar
- 2o. milestone: algoritmo para resolver o mundo de blocos — Editar

Acrescentar recurso... | Adicionar uma atividade...

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

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

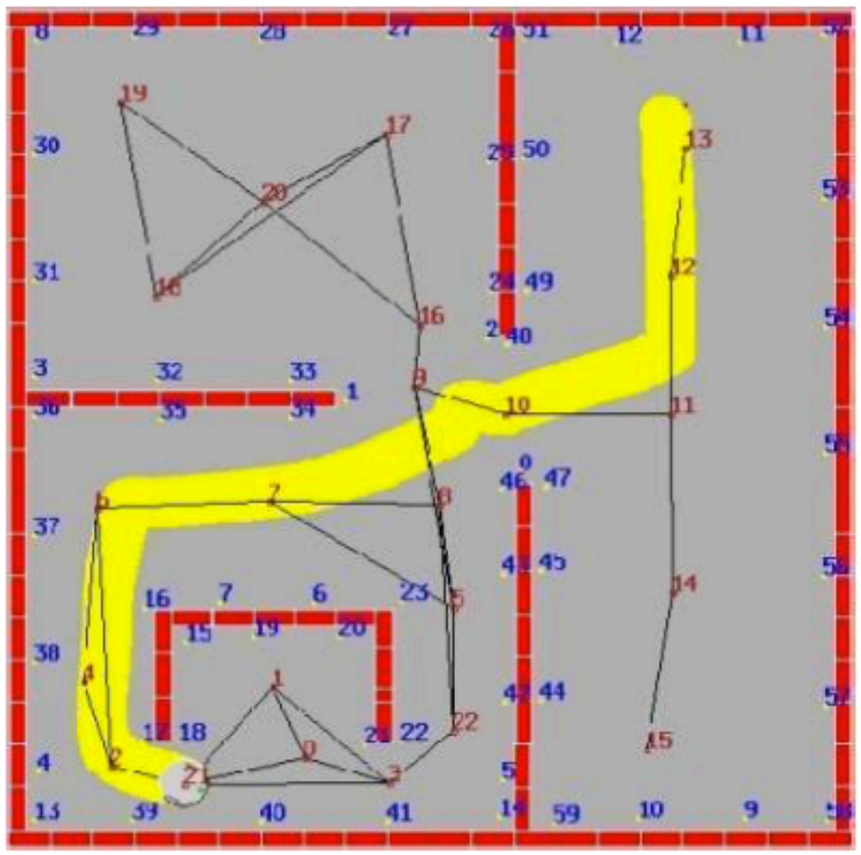
Kaio Takase - 9345690

Matheus Ramalho - 9345710

Grupo 8(?):

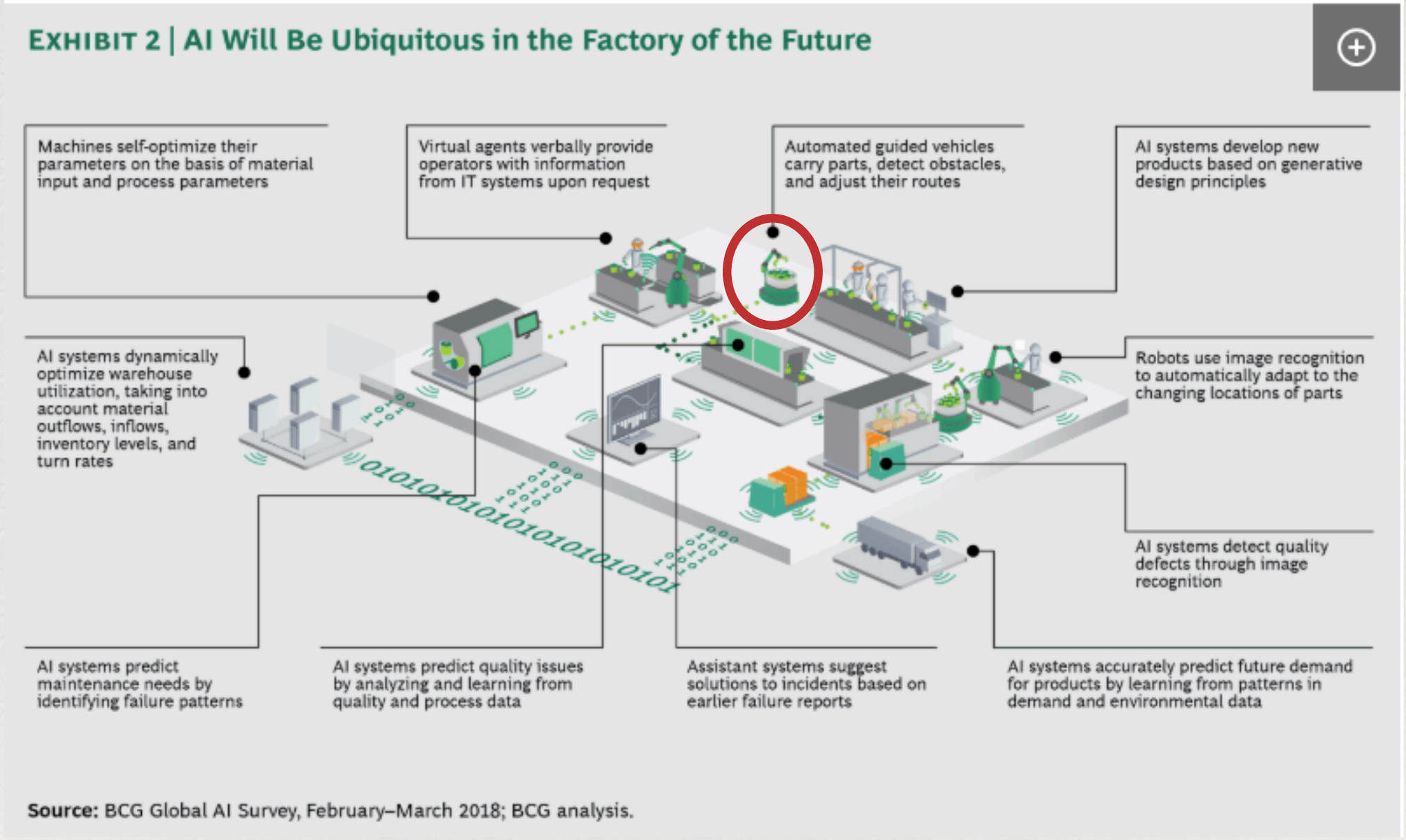
Danilo Polidoro - 8582982

Robot Navigation



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

AI in the Factory of the Future: The Ghost in the Machine



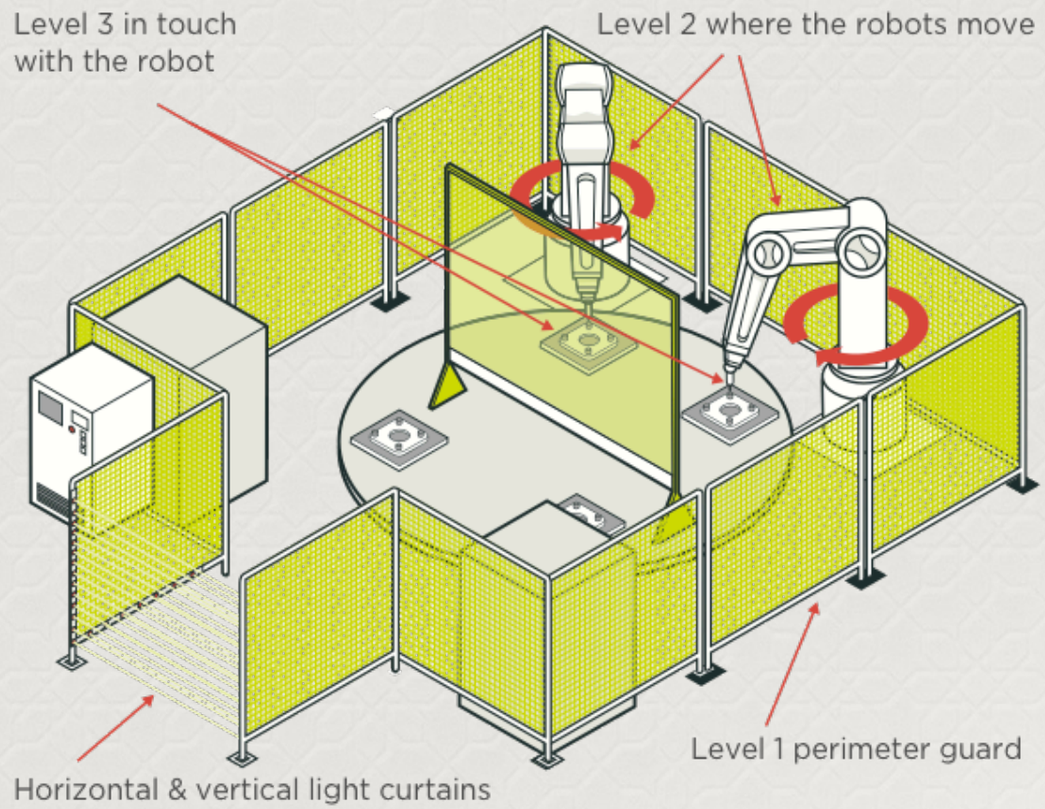
https://www.bcg.com/publications/2018/artificial-intelligence-factory-future.aspx

Na prática são poucos os casos onde se pode mapear o espaço de estados de um robô móvel em ambiente real.

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

Outro problema são as "interferências", especialmente as causadas pela presença de humanos.

Um dos grandes desafios é guiar um veículo móvel sem referências, isto é, em ambiente não estruturado e aberto.



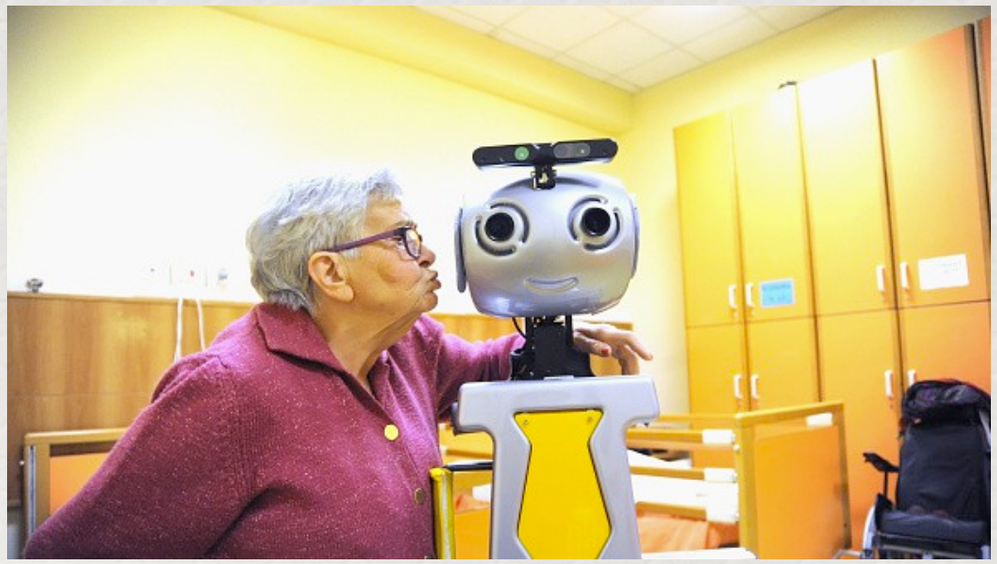
This is Pepper

Novas aplicações em arquitetura de precisão

Novas aplicações com drones

Os drones podem fazer vôos longos e ter uma autonomia bastante razoável, o que amplia sua aplicação. Mas precisam de referências para orientar o vôo ou de sistemas inteligentes para se orientar com poucas referências.

Estamos portanto a uma grande distância destas aplicações de grande porte e de grande efeito, seja na Manufatura 4.0, seja na moderna agricultura de precisão, seja na quebra da barreira que ainda existe entre homens e máquinas. Mas toda grande trajetória começa com os primeiros passos.



E falando em... primeiros passos, vamos voltar a eles, discutindo agora os métodos de busca, que podem ser aplicados ao problema modelo do mundo de blocos, que ilustra o método STRIPS de aplicação de provadores de teorema à resolução de problemas em IA.

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

Heuristic-guided Search

- Our complexity analysis of the various basic search algorithms has shown that they are unlikely to produce results for slightly more complex problems than we have considered here.
- In general, there is no way around this problem. In practice, however, good *heuristics* that tell us which part of the search tree to explore next, can often help to find solutions also for larger problem instances.
- In this final chapter on search techniques for AI, we are going to discuss one such heuristic, which leads to the well-known A* algorithm.

Estratégias de Busca Informada

Análise de custo

$f(x) = g(x)$

Heurística

$f(x) = h(x)$

Algorítmo A*

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

The A* Algorithm

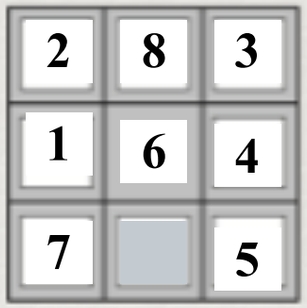
The central idea in the so-called *A* algorithm* is to guide best-first search both by

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

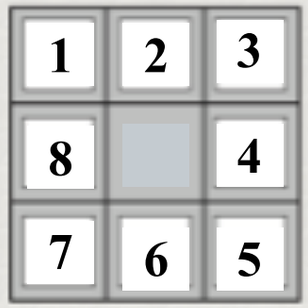
Let n be a node, $g(n)$ the cost of moving from the initial node to n along the current path, and $h(n)$ the estimated cost of reaching a goal node from n. Define $f(n)$ as follows:

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

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.*



f(0) = 0 + 4

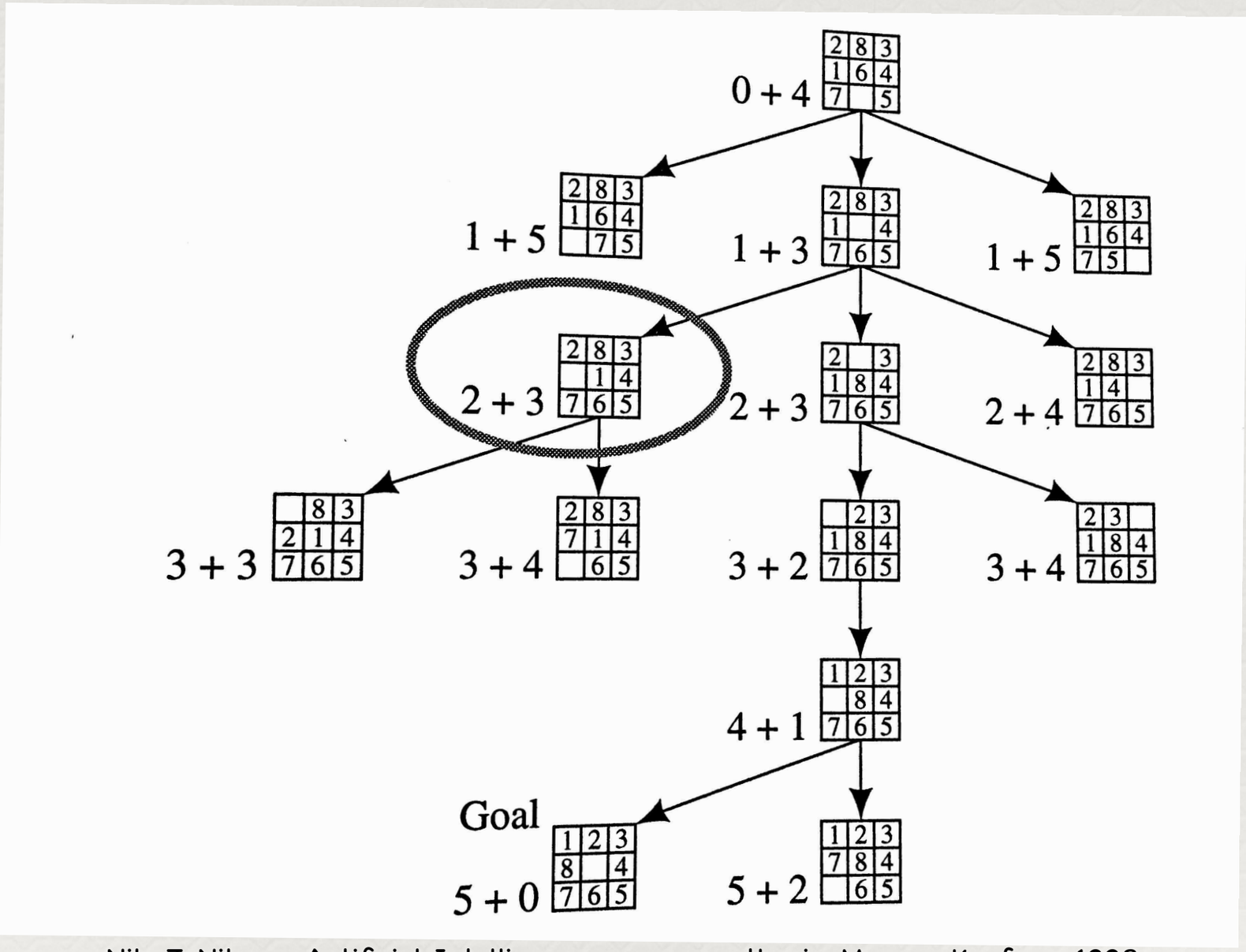


estado final

comparando o estado inicial (a raiz da árvore de busca) com o estado final mostrado à direita, temos que os tiles 1, 2, 8 e 6 estão fora do lugar, portanto neste estado (o nível zero ou raiz, está a uma distância avaliada pelo número mínimo de movimentos para atingir o estado final) h(0) = 4, e a função de avaliação é f(0) = 0 + 4 = 4

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

Árvore de busca e a busca informada



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

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.

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.*

Dado um nó n, vamos definir o custo para ir deste nó para o seu sucessor n', realizando a ação a, como sendo C(n, a, n').

Uma heurística h é dita consistente, se para qualquer sucessor n' do nó n, vale a seguinte inequação:

$$h(n) \leq C(n, a, n') + h(n')$$

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.

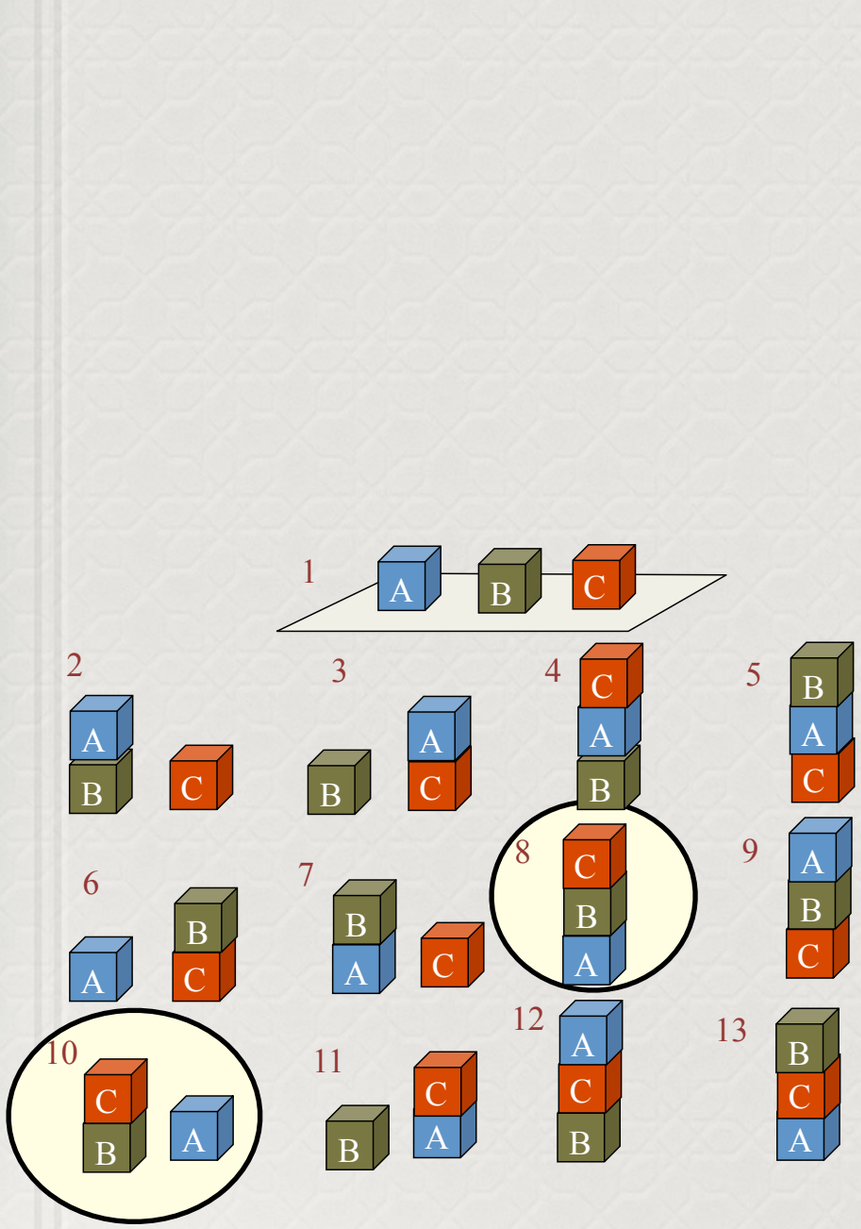
Portanto o A* é um algoritmo de busca informada "otimizante", que, em situações práticas admite uma abordagem heurística menos rigorosa, para obter uma reposta mais rápida.

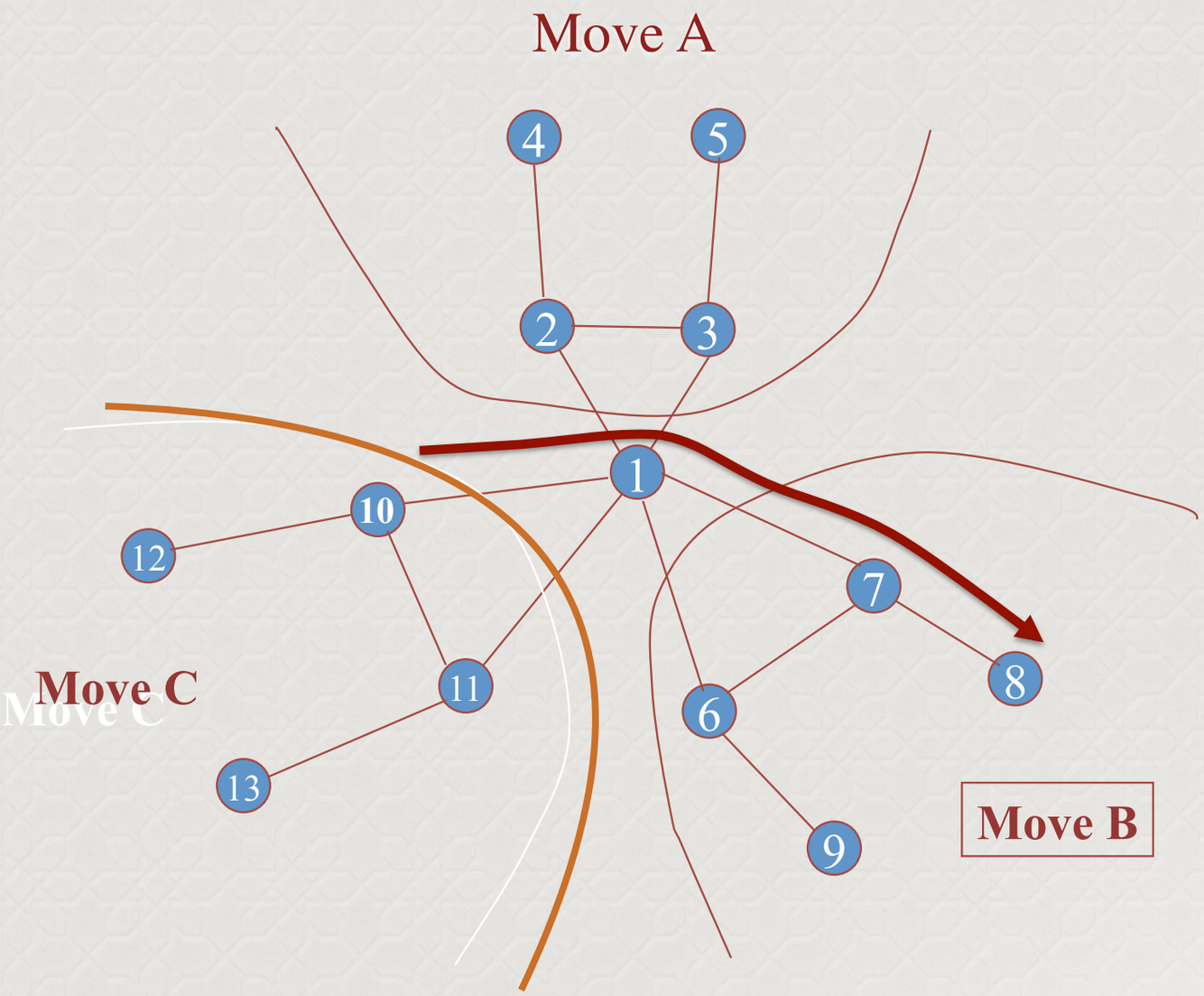
Como seria a aplicação de heurísticas ao mundo de blocos?

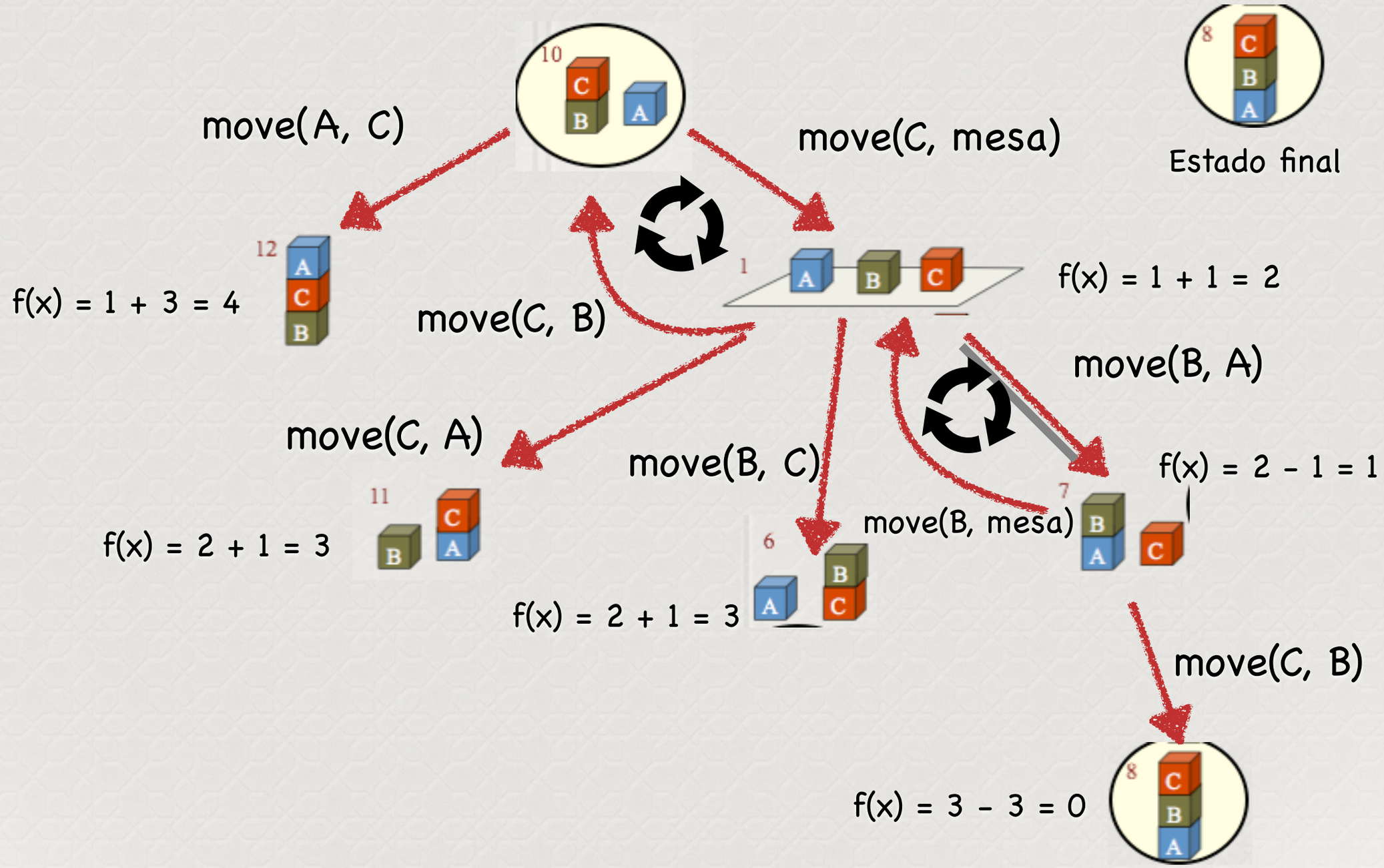
Uma idéia seria, se forma similar ao jogo de tiles, incrementar e associar os movimentos a uma função

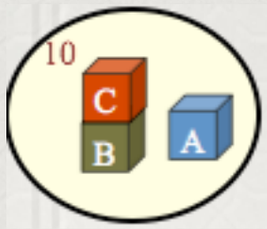
$$f(x) = g(x) + h(x)$$

Onde g(x) é a função custo, dado pela profundidade da árvore de busca, e h(x) uma ponderação entre os blocos fora de lugar (+1) e os blocos que já estão no devido lugar (-1), tendo em vista o estado final.

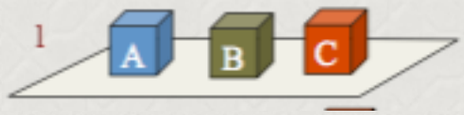

1
A
B
C
2
3
4
5
6
7
8
9
10
11
12
13


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


10
C
B
A
move(A, C)
move(C, mesa)
8
C
B
A
Estado final
12
A
C
B
f(x) = 1 + 3 = 4
1
A
B
C
f(x) = 1 + 1 = 2
move(C, B)
move(B, A)
move(C, A)
move(B, C)
f(x) = 2 - 1 = 1
11
C
B
A
f(x) = 2 + 1 = 3
7
B
A
C
move(B, mesa)
6
B
C
A
f(x) = 2 + 1 = 3
move(C, B)
8
C
B
A
f(x) = 3 - 3 = 0

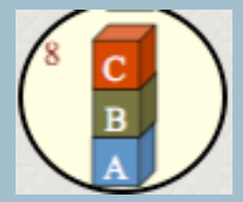


sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)



sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)

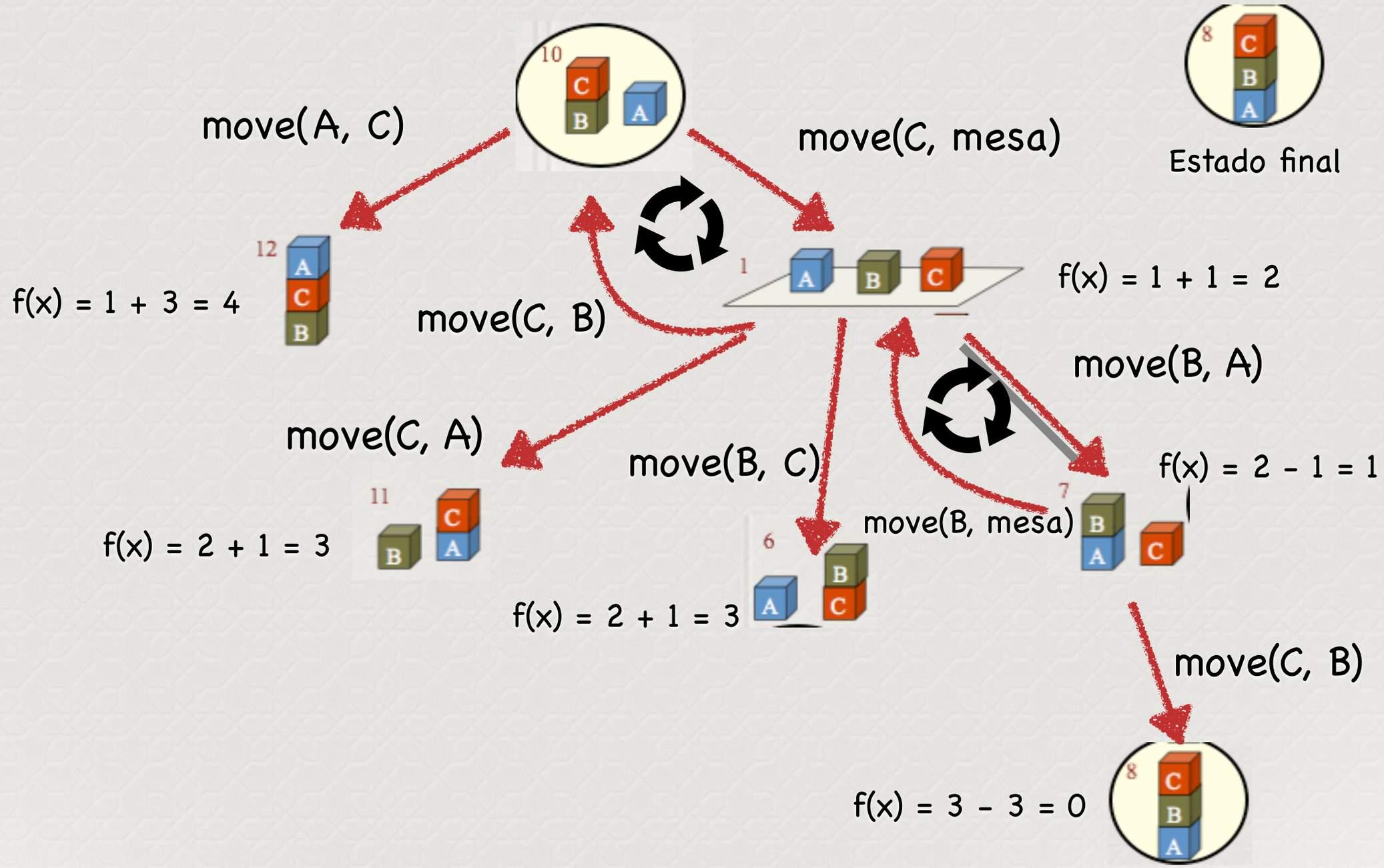
move(C, B)



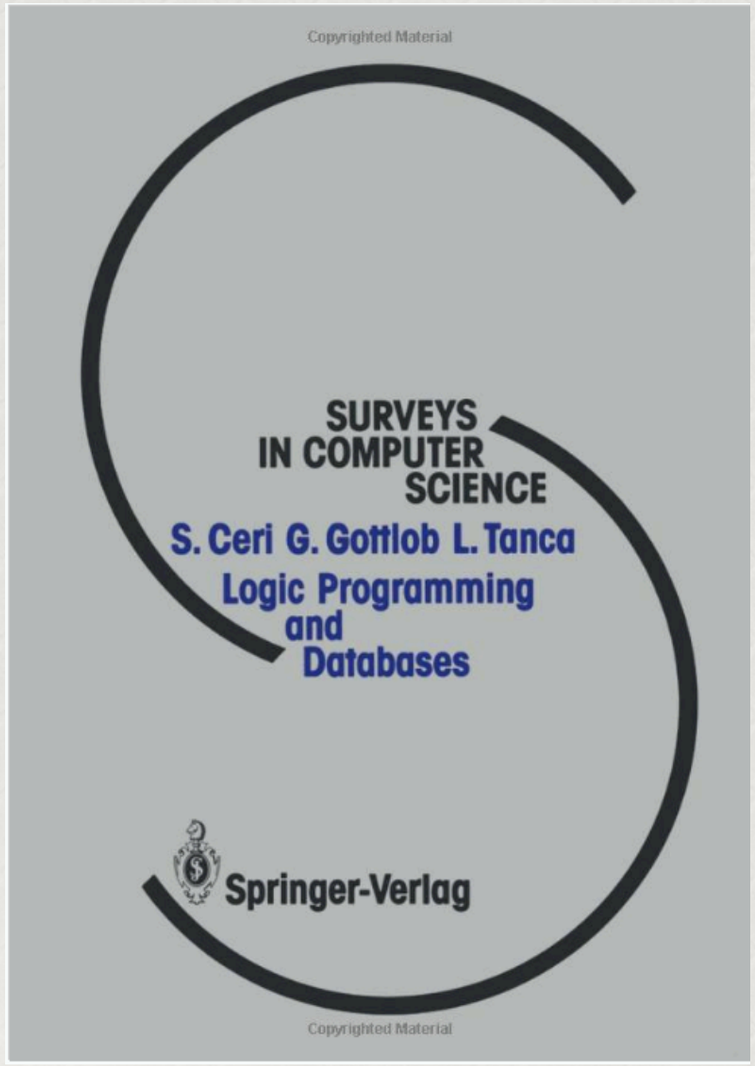
Estado final

sobre(c, b)
sobre(b, a)
sobre(a, mesa)
livre(c)

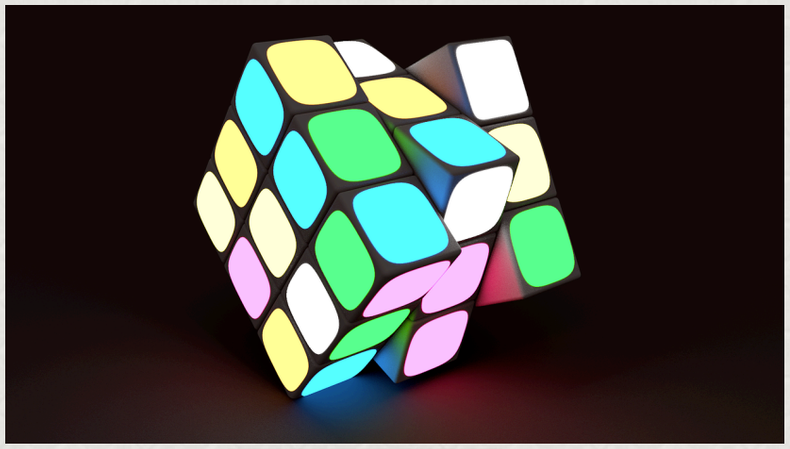
Gerald J. Sussman

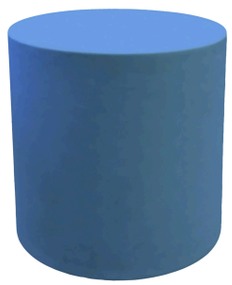

10
C
B
A
move(A, C)
move(C, mesa)
8
C
B
A
Estado final
12
A
C
B
f(x) = 1 + 3 = 4
1
A
B
C
f(x) = 1 + 1 = 2
move(C, B)
move(B, A)
move(C, A)
move(B, C)
f(x) = 2 - 1 = 1
11
C
B
A
f(x) = 2 + 1 = 3
7
B
A
C
move(B, mesa)
6
B
C
A
f(x) = 2 + 1 = 3
move(C, B)
8
C
B
A
f(x) = 3 - 3 = 0

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.





A programação lógica, especialmente em Prolog é orientada a sequências enquanto os BDs (relacionais) são orientados a conjuntos.

https://www.swi-prolog.org/howto/database.html

How to deal with the Prolog dynamic database?

The short answer is: *if you can avoid using it, consider this first.*

Why is the dynamic database evil?

The database is a non-logical extension to Prolog. The core philosophy of Prolog is that is searches for a set of variable bindings that makes a query true. It does this using depth-first search, binding variables to values. If an inconsistent variable binding is detected, the system backtracks: it finds the last choice it took, reverts all actions done since and continues with the next alternative. This is the core power of Prolog and gives it its logical basis.

Even if you cannot be bothered by logic, consider what this does for debugging: you can optimistically step over (skip) goals. If the answer is wrong, you hit **retry**, which takes the execution back to the start and you inspect the goal at a deeper level. In other words, the backtracking machinery provides you with a time-machine that allows you to go back in history.

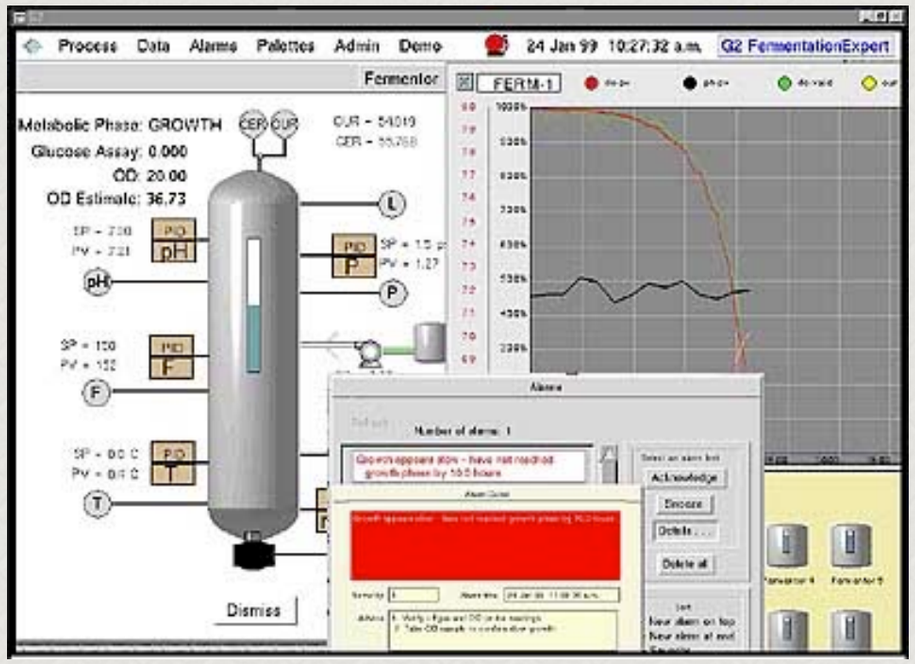
Changes to the dynamic database are not reverted on backtracking, destroying all these nice goodies. Second, dynamic predicates are like destructible **global variables** and therefore subject to all the common problems with such objects: name-space pollution, state-full computation, initialization and cleanup requirements and lack of *thread safety*.

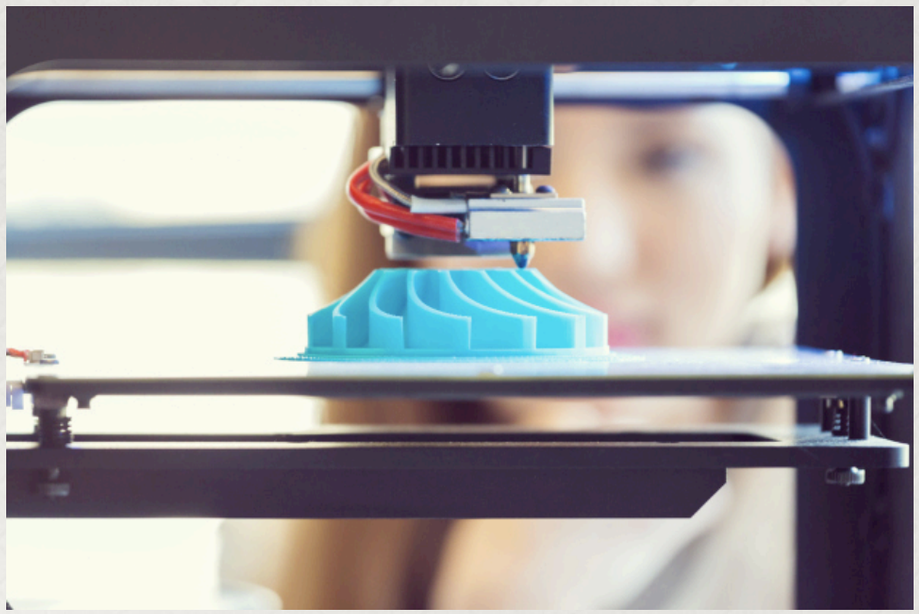
I need to preserve data over backtracking

Sometimes, results are computed and stored in the database to preserve them over backtracking. E.g.,

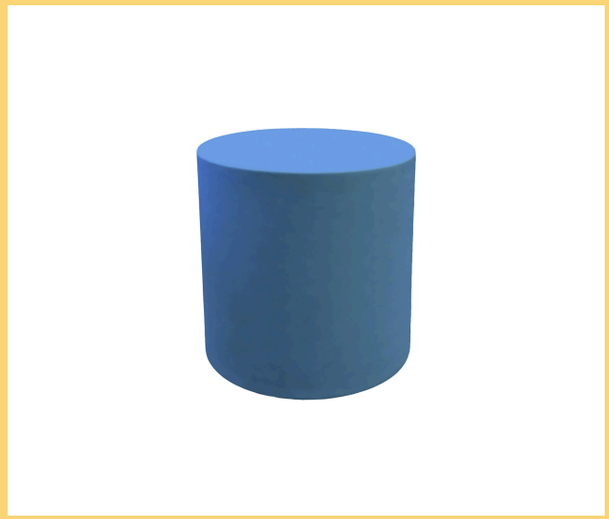
```
...,
assertz(result(X)),
...,
fail


...,
result(X),
```





Com a perspectiva atual de associar a programação lógica e as aplicações de AI (clássicas ou baseadas em ML) com grandes volumes de dados (I4.0) é bastante tentador associar BDs à programação lógica.



Database Manipulation Language

DML

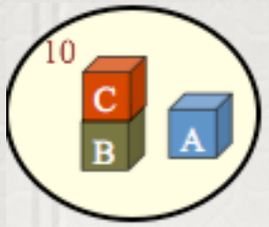
Database Definition Language

DDL

SGBD

Existem basicamente quatro comandos (de manipulação) em Prolog:

- assert,
- retract,
- asserta,
- assertz.

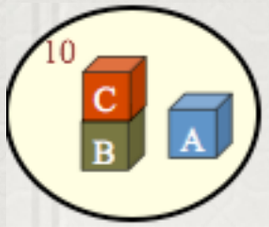


sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)

sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)

```
sobre(c, mesa).
sobre(b, mesa).
sobre(a, mesa).
livre(c).
livre(a).
```

```
:- assert(sobre(c, mesa)).
:- assert(sobre(b, mesa)).
:- assert(sobre(a, mesa)).
:- assert(livre(c)).
:- assert(livre(a)).
```



sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)

sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)

```
estado_corrente([sobre(c, mesa),
                 sobre(b, mesa),
                 sobre(a, mesa)),
                 livre(c),
                 livre(a)]).
```

```
:- assert(estado_corrente([sobre(c, mesa),
          sobre(b, mesa),
          sobre(a, mesa)),
          livre(c),
          livre(a)])).
```

Existem várias possibilidades de implementação e a prioridade é pensar por enquanto no algoritmo. Voltaremos a esta discussão na aula que vem ou pela página da disciplina (depois que tivermos o algoritmo definido).

Até a próxima aula!