Escola Politécnica da USP - Depto. de Enga. Mecatrônica

PMR-3510 Inteligência Artificial

AULA 12 - "PLANNING" E RESOLUÇÃO DE PROBLEMAS

Prof. José Reinaldo Silva

reinaldo@usp.br

Nesta aula completaremos a discussão sobre a estrutura dos sistemas inteligentes (clássicos), e discutiremos o método de unificação/resolução (a base da programação lógica), aplicado à resolução de problemas, agora usando recursos de IA mais elaborados.

CHAPTER 9

INFERENCE IN FIRST-ORDER LOGIC

In which we define effective procedures for answering questions posed in first-order logic.

chaining (Section 9.3), **backward chaining** (Section 9.4), and **resolution-based theorem proving** (Section 9.5).

9.1 Propositional vs. First-Order Inference

One way to do first-order inference is to convert the first-order knowledge base to propositional logic and use propositional inference, which we already know how to do. A first step is eliminating universal quantifiers. For example, suppose our knowledge base contains the standard folkloric axiom that all greedy kings are evil:

$$\forall x \; King(x) \wedge Greedy(x) \Rightarrow Evil(x) .$$

From that we can infer any of the following sentences:

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John)) .$$
$$\vdots$$

In general, the rule of **Universal Instantiation** (UI for short) says that we can infer any

$\{y/M_1\}$ $\{\}$ $\{\}$ $\{\}$

Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove *Criminal*(*West*), we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally *Hostile*(*z*), *z* is already bound to *Nono*.

unify with the goal, standardizing the variables in the clause to be brand-new variables, and then, if the *rhs* of the clause does indeed unify with the goal, proving every conjunct in the *lhs*, using FOL-BC-AND. That function works by proving each of the conjuncts in turn, keeping track of the accumulated substitution as it goes. Figure 9.7 is the proof tree for deriving *Criminal*(*West*) from sentences (9.3) through (9.10).

Backward chaining, as we have written it, is clearly a depth-first search algorithm. This means that its space requirements are linear in the size of the proof. It also means that backward chaining (unlike forward chaining) suffers from problems with repeated states and incompleteness. Despite these limitations, backward chaining has proven to be popular and effective in logic programming languages.

9.4.2 Logic programming

Logic programming is a technology that comes close to embodying the declarative ideal described in Chapter 7: that systems should be constructed by expressing knowledge in a formal language and that problems should be solved by running inference processes on that knowledge. The ideal is summed up in Robert Kowalski's equation,

$$Algorithm = Logic + Control.$$

Prolog is the most widely used logic programming language. It is used primarily as a rapid-prototyping language and for symbol-manipulation tasks such as writing compilers (Van Roy, 1990) and parsing natural language (Pereira and Warren, 1980). Many expert systems have been written in Prolog for legal, medical, financial, and other domains.

Prolog programs are sets of definite clauses written in a notation somewhat different from standard first-order logic. Prolog uses uppercase letters for variables and lowercase for constants—the opposite of our convention for logic. Commas separate conjuncts in a clause,


Lógica de primeira ordem
Lógica de proposicional
Cláusulas de Horn

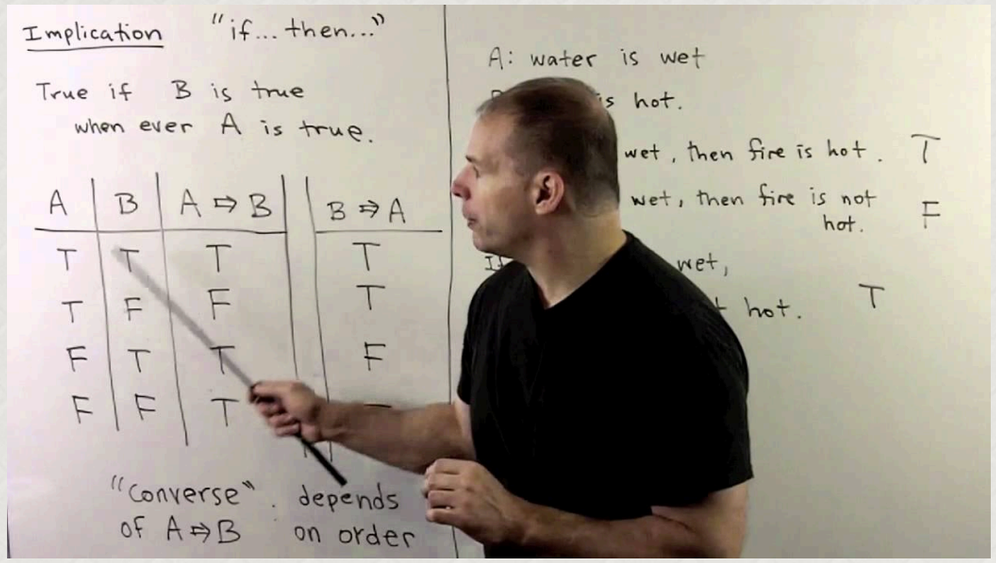
A estratégia de resolução implica em aplicar um processo de proposicionalização das sentenças, um processo conhecido como "Skolenização".

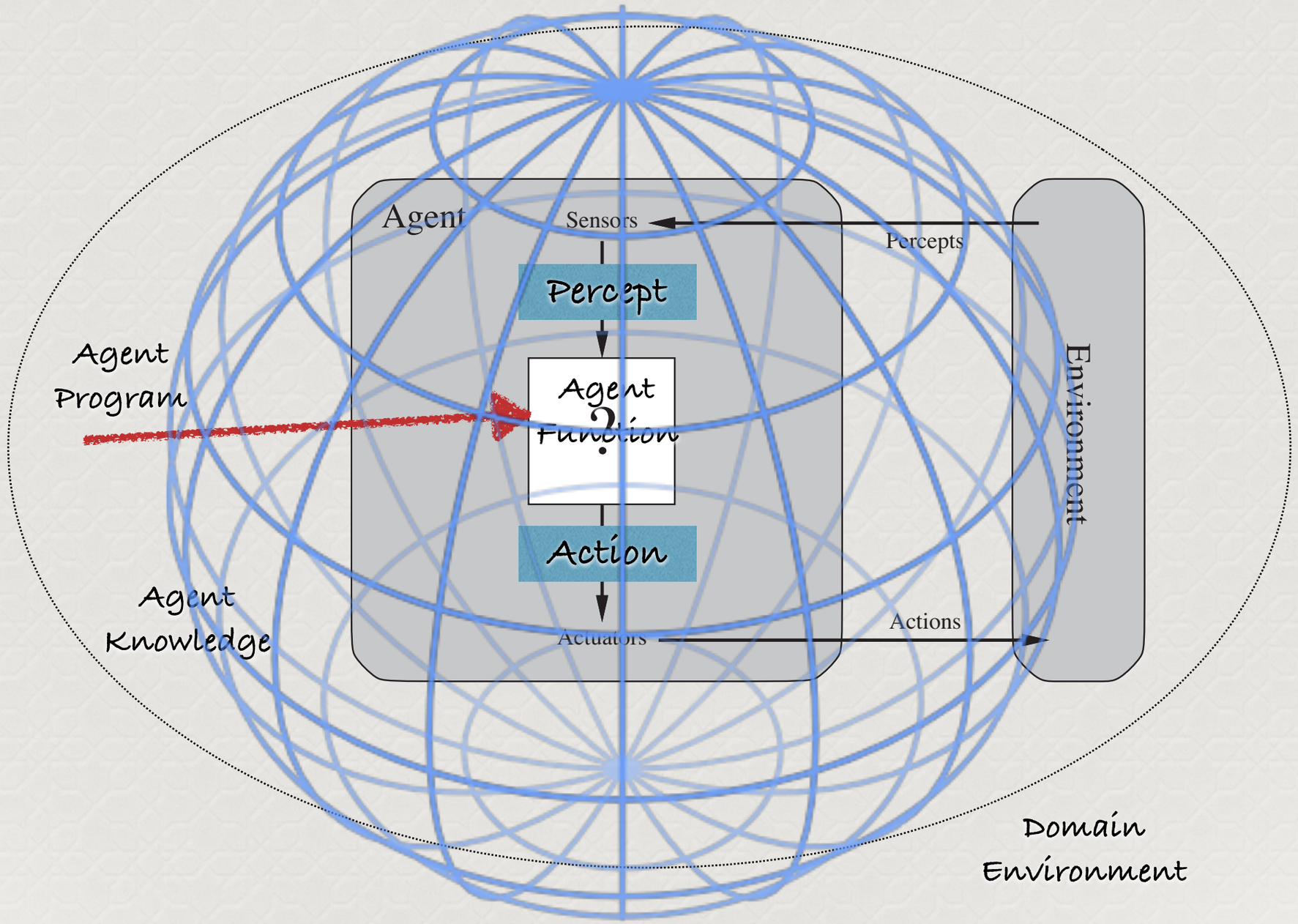
$$\forall King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

Skolen

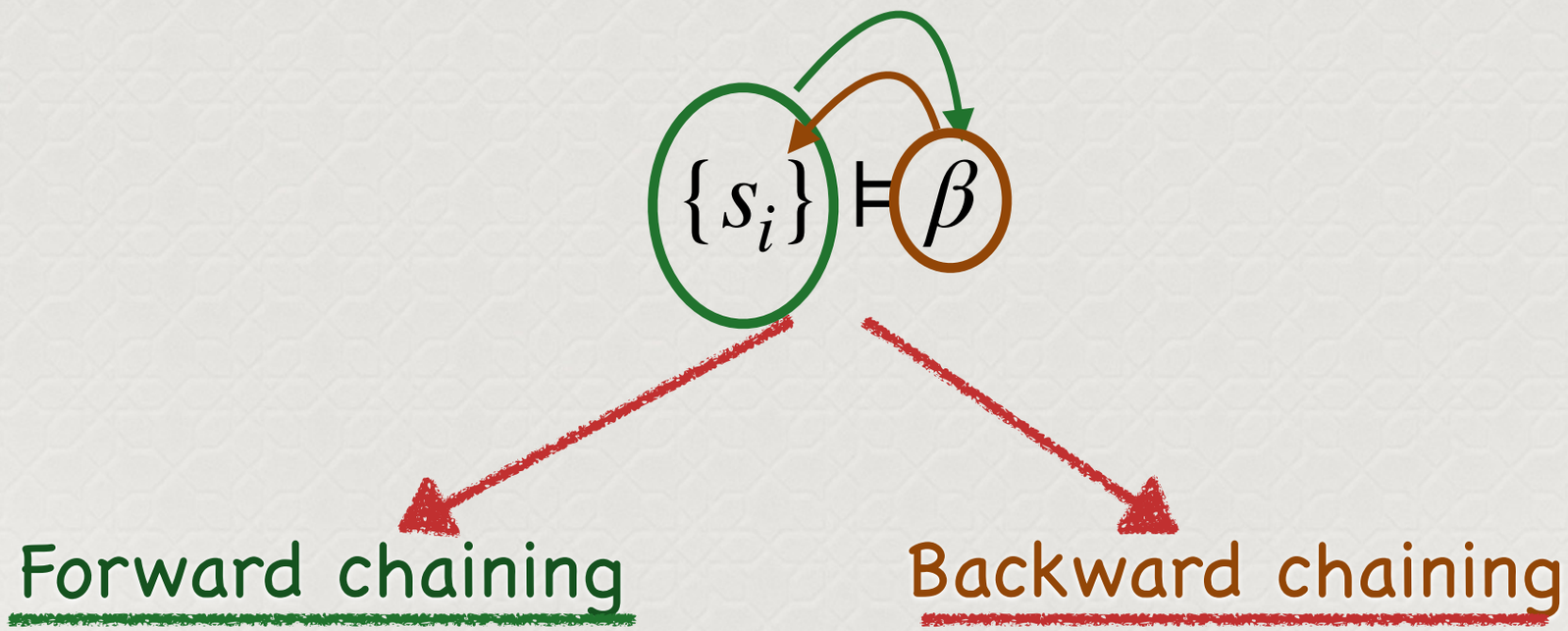
$$King(John)$$
$$Greedy(John)$$
$$Brother(John, Richard)\ .$$

Achar uma interpretação que valide uma fórmula (genérica ou em cláusula de Horn) significa achar uma substituição θ que instancie todas as variáveis e valide a sentença alvo por Modus Ponens Generalizado.




Agent
Sensors
Percepts
Percept
Agent Program
Agent Function
?
Action
Actuators
Actions
Environment
Agent Knowledge
Domain Environment

É possível encadear (entail) sentenças logicamente, e deduzir novas sentenças que sejam verdadeiras sob a mesma interpretação.



$$Americano(x) \land Arma(y) \land Vende(x, y, z) \land Hostil(z) \Rightarrow Criminoso(x)$$

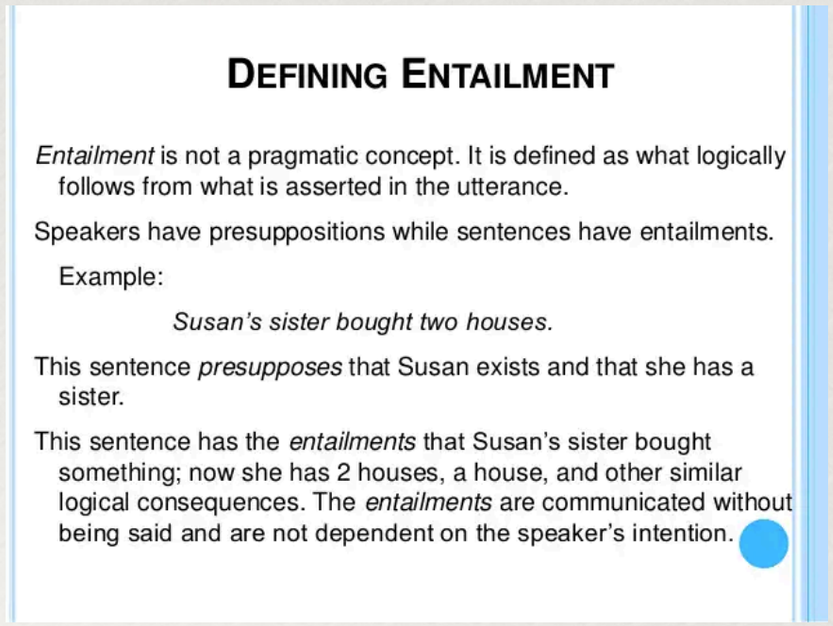
$$Possui(Nono, M1)$$

$$Missel(M1)$$

$$Missel(x) \Rightarrow Arma(x) \qquad Inimigo(x, EUA) \Rightarrow Hostil(x)$$

$$Possui(Nono, x) \land Missel(x) \Rightarrow Vende(West, x, Nono)$$

$$Americano(West)$$



DEFINING ENTAILMENT

Entailment is not a pragmatic concept. It is defined as what logically follows from what is asserted in the utterance.

Speakers have presuppositions while sentences have entailments.

Example:

Susan's sister bought two houses.

This sentence *presupposes* that Susan exists and that she has a sister.

This sentence has the *entailments* that Susan's sister bought something; now she has 2 houses, a house, and other similar logical consequences. The *entailments* are communicated without being said and are not dependent on the speaker's intention.

$$\mathfrak{F}\{x/West, y/M1, z/Nano\}$$

$$Criminoso(West)$$

$$Criminoso(x)$$

$$\mathfrak{I}\{x/West, y/M1, z/Nano\}$$

$$Americano(x) \wedge Arma(y) \wedge Vende(x, y, z) \wedge Hostil(z) \Rightarrow Criminoso(x)$$

$$Possui(Nono, M1)$$

$$Missel(M1)$$

$$Missel(x) \Rightarrow Arma(x) \qquad Inimigo(x, EUA) \Rightarrow Hostil(x)$$

$$Possui(Nono, x) \wedge Missel(x) \Rightarrow Vende(West, x, Nono)$$

$$Americano(West)$$

1. First, we observe that if S is unsatisfiable, then there exists a particular set of *ground instances* of the clauses of S such that this set is also unsatisfiable (Herbrand's theorem).
2. We then appeal to the **ground resolution theorem** given in Chapter 7, which states that propositional resolution is complete for ground sentences.
3. We then use a **lifting lemma** to show that, for any propositional resolution proof using the set of ground sentences, there is a corresponding first-order resolution proof using the first-order sentences from which the ground sentences were obtained.

SWISH File Edit Examples Help

Mundo dos blocos - Rework 3.0 | Program

```
1  % Teste of unification
2
3  king(john).
4  king(richard).
5  king(henryII).
6  king(henryVIII).
7  king(george).
8
9  queen(vitoria).
10
11 greedy(john).
12 greedy(henryVIII).
13
14 father(henryII, john).
15 father(henryII, reichard).
16
17 brother(john, richard).
18
19
```


king(X).

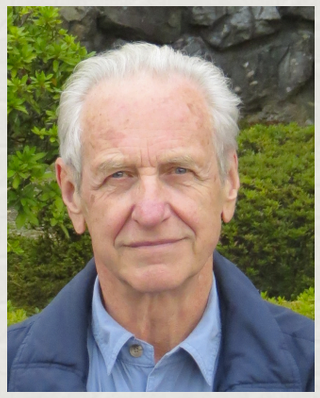
X = john

Next 10 100 1,000 Stop

?- king(x).

Association for Computing Machinery

Communication of ACM, January 1988, vol. 31, no. 1.



Robert Kowalski

ARTICLES

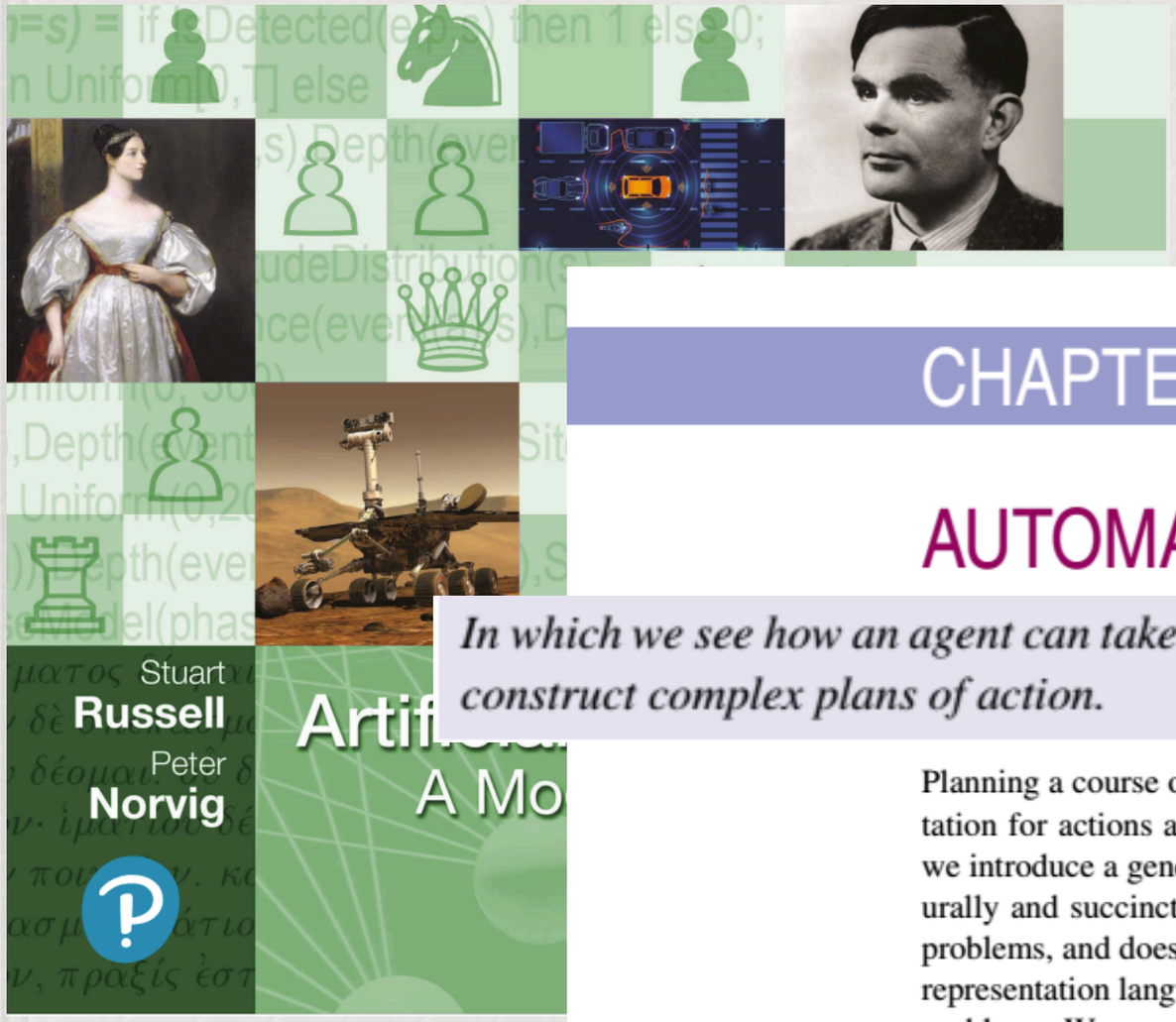
THE EARLY YEARS OF LOGIC PROGRAMMING

This firsthand recollection of those early days of logic programming traces the shared influences and inspirations that connected Edinburgh, Scotland, and Marseilles, France.

ROBERT A. KOWALSKI

The name *Prolog* is ambiguous. It was originally intended as the name for the programming language developed by Alain Colmerauer and Phillipe Roussel in the summer of 1972. The name was suggested by Roussel's wife, Jacqueline, as an abbreviation for *programmation en logique*. In time, however, this abbreviation has been used to refer to the concept of logic pro-

clauses and deduction is performed by backwards reasoning embedded in resolution [29]. But logic programming can also be understood more generally, for example, to include negation by failure [3], set construction [4, 32], or goal-directed reasoning with equations. The advantage of the more liberal notion of logic programming is that it points the way for further developments



CHAPTER 11

AUTOMATED PLANNING

In which we see how an agent can take advantage of the structure of a problem to efficiently construct complex plans of action.

Planning a course of action is a key requirement for an intelligent agent. The right representation for actions and states and the right algorithms can make this easier. In Section 11.1 we introduce a general **factored** representation language for planning problems that can naturally and succinctly represent a wide variety of domains, can efficiently scale up to large problems, and does not require ad hoc heuristics for a new domain. Section 11.4 extends the representation language to allow for hierarchical actions, allowing us to tackle more complex problems. We cover efficient algorithms for planning in Section 11.2, and heuristics for them in Section 11.3. In Section 11.5 we account for partially observable and nondeterministic domains, and in Section 11.6 we extend the language once again to cover scheduling problems with resource constraints. This gets us closer to planners that are used in the real world for planning and scheduling the operations of spacecraft, factories, and military campaigns. Section 11.7 analyzes the effectiveness of these techniques.

11.1 Definition of Classical Planning

Classical planning

Classical planning is defined as the task of finding a sequence of actions to accomplish a

Agentes baseados em objetivos (Goal-based agents)

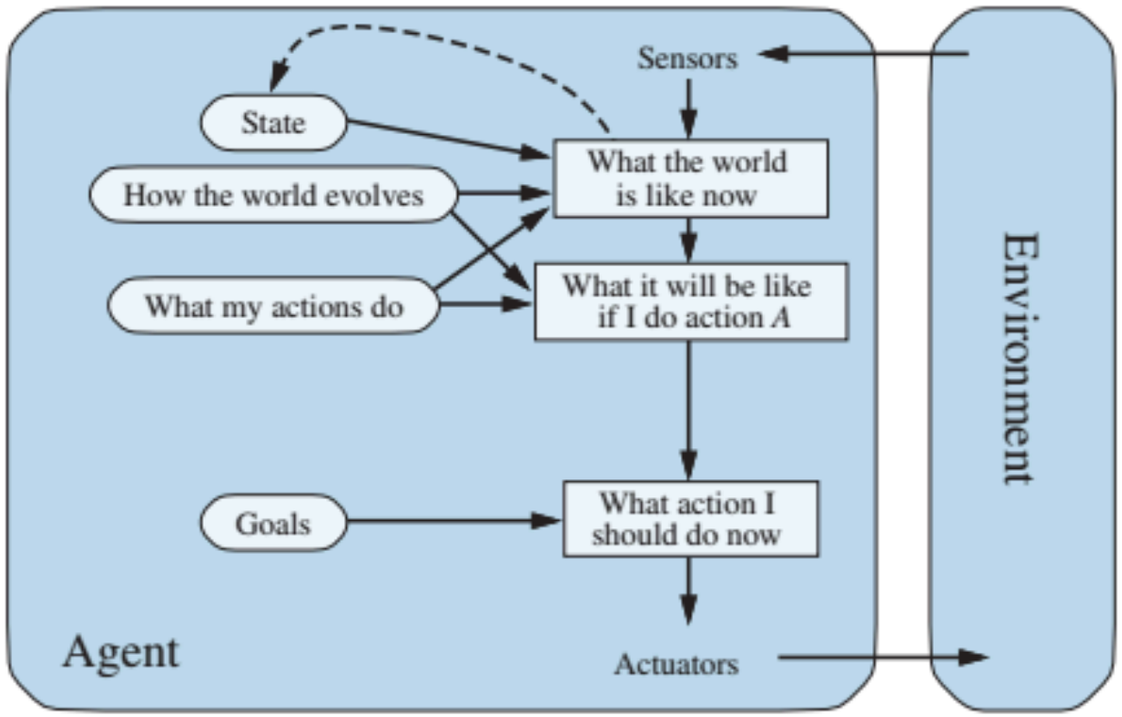


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

"Planning" denota uma classe de problemas com áreas de aplicação bem diversificadas: seja na logística, na manufatura, nos sistemas espaciais, sistemas de saúde, etc.

O importante é saber qual a essência desse tipo de resolução de problema, e como se aplicam as técnicas de inteligência artificial.

Na aula passada mencionamos que

Uma possibilidade é gerar o espaço de estados e fazer uma busca, informada ou não, sobre esse espaço. Essa é a abordagem clássica que vimos no início do curso.

Outra abordagem seria detectar no "environment" o estado inicial, e um estado final como objetivo. Com isso podemos usar a base de conhecimento do agente para determinar uma sequência de ações consistentes que leve o environment a este estado final.

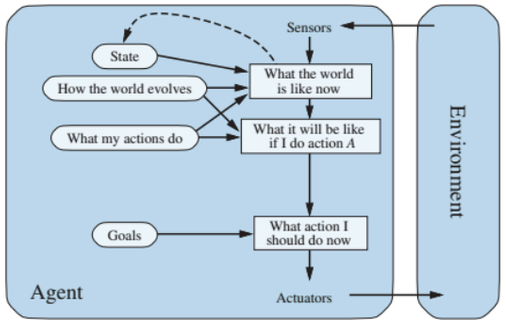


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Em que isso é diferente de um processo de busca?

- Temos agora um "transition system", e é preciso fazer as mudanças de estado com precisão;
- Temos um conjunto de ações, onde podemos escolher uma ação é admissível, e avaliar se o resultado desta ação aproxima o sistema do seu objetivo;
- Podemos achar uma heurística para escolher a "melhor" ação.

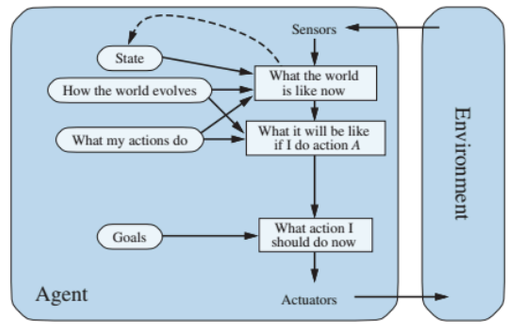
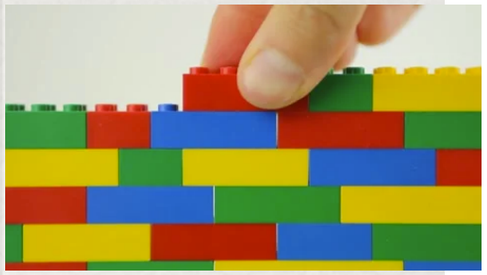


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

A pergunta de fato é : existe um método geral, independente de domínio, para resolver problemas de "planning", isto é, o problema dos agentes com objetivo?.

A "dependência de domínio" nos obriga a obter uma quantidade maior de informação sobre o environment, e também a ter um processamento mais elaborado para os percepts, gerando soluções que normalmente só valem para uma configuração específica do domínio.



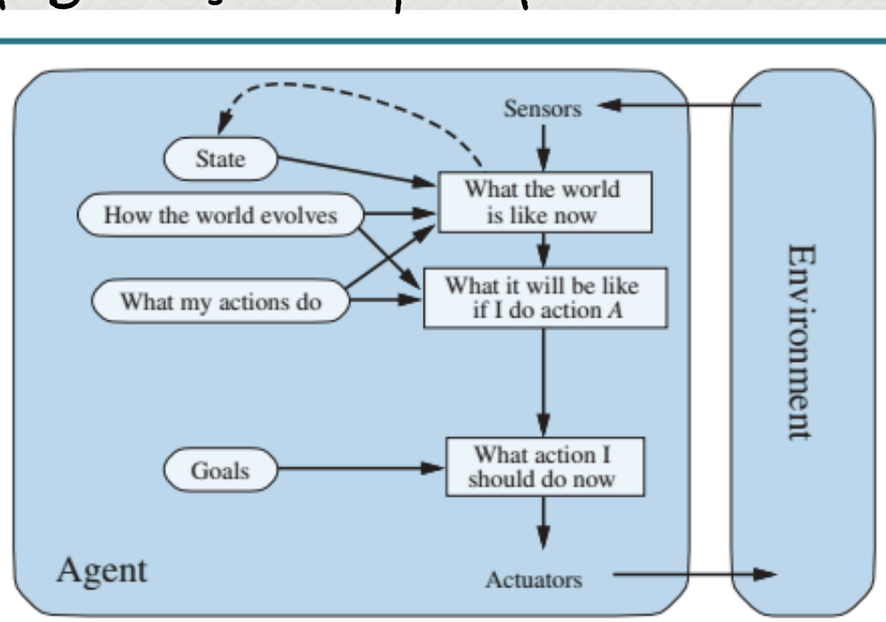


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

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

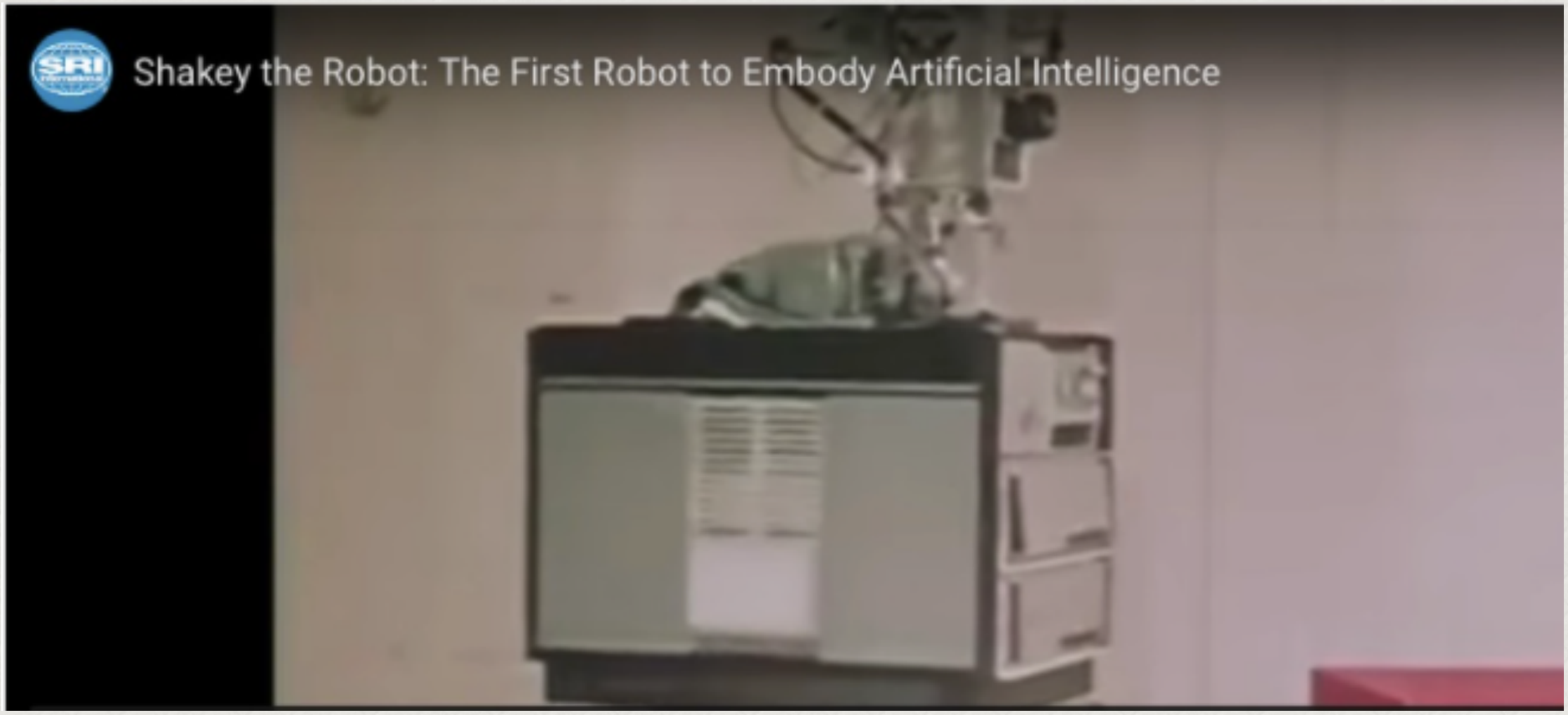
We describe a new problem solver called STRIPS that attempts to find a sequence of operators in a space of world models to transform a given initial world model into a model in which a given goal formula can be proven to be true. STRIPS represents a world model as an arbitrary collection of first-order 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



O método STRIPS para resolução de problemas foi criado para automação (robótica) e testado no robô Shakey.

estado inicial

estado objetivo

Conjunto de Ações

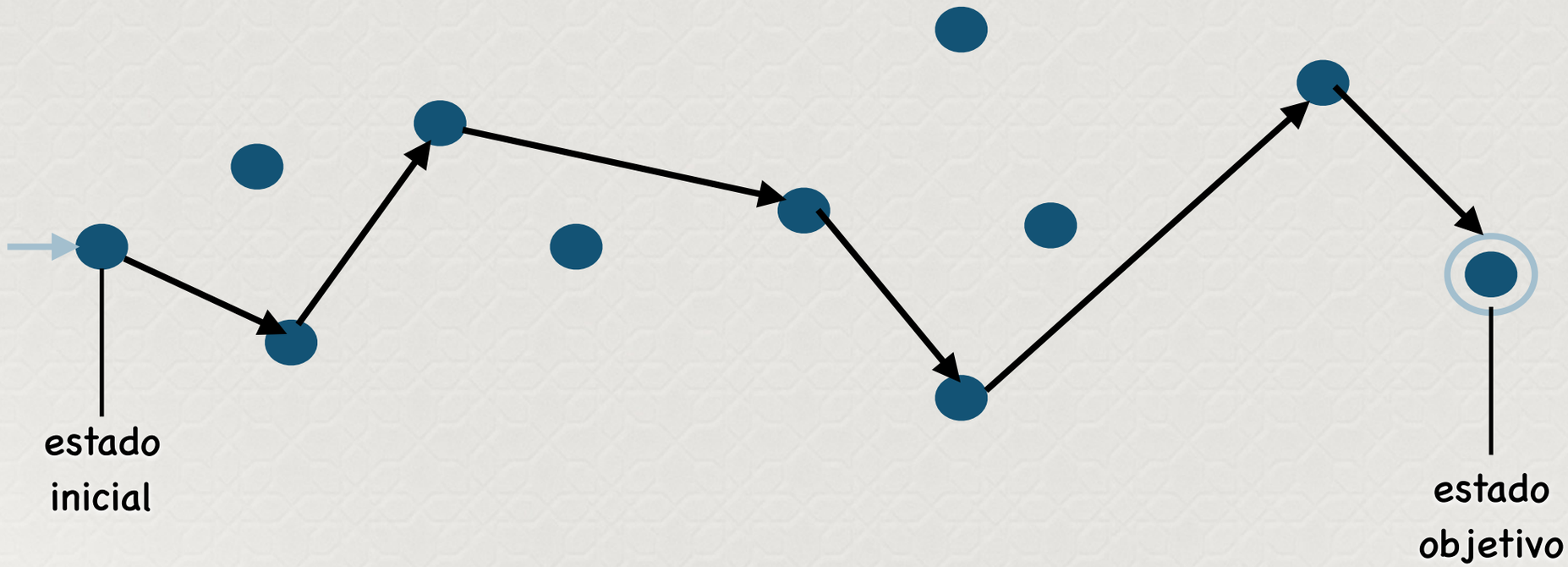
Problema de planejamento

Critério de aplicação das ações

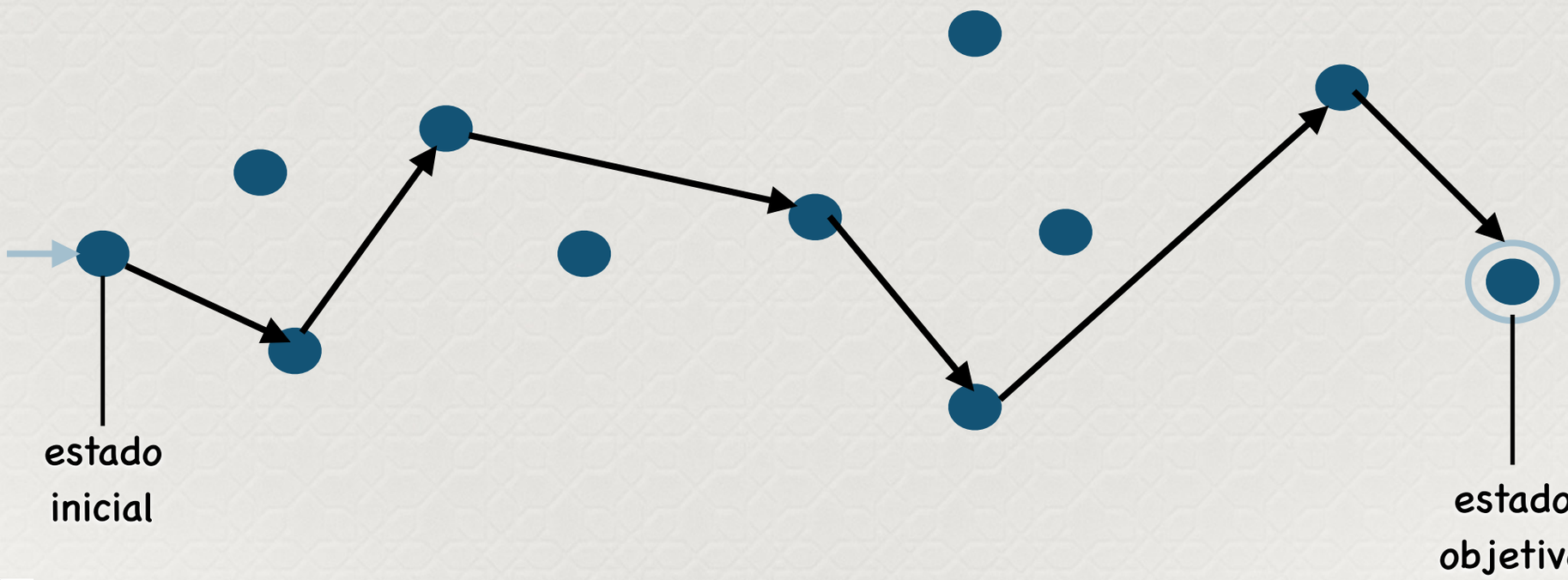
Percepção das mudanças de estado

Análise de custo

Base de conhecimento



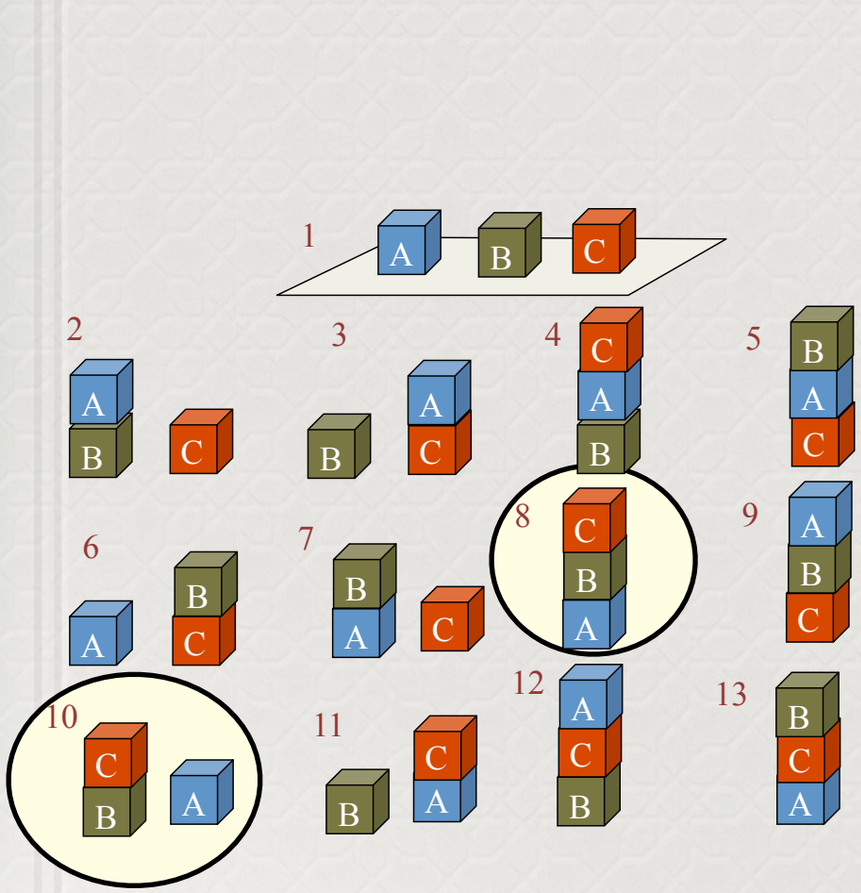
Cada estado é definido por um conjunto de sentenças lógicas, e mudar de estado significa remover algumas destas sentenças e acrescentar outras.

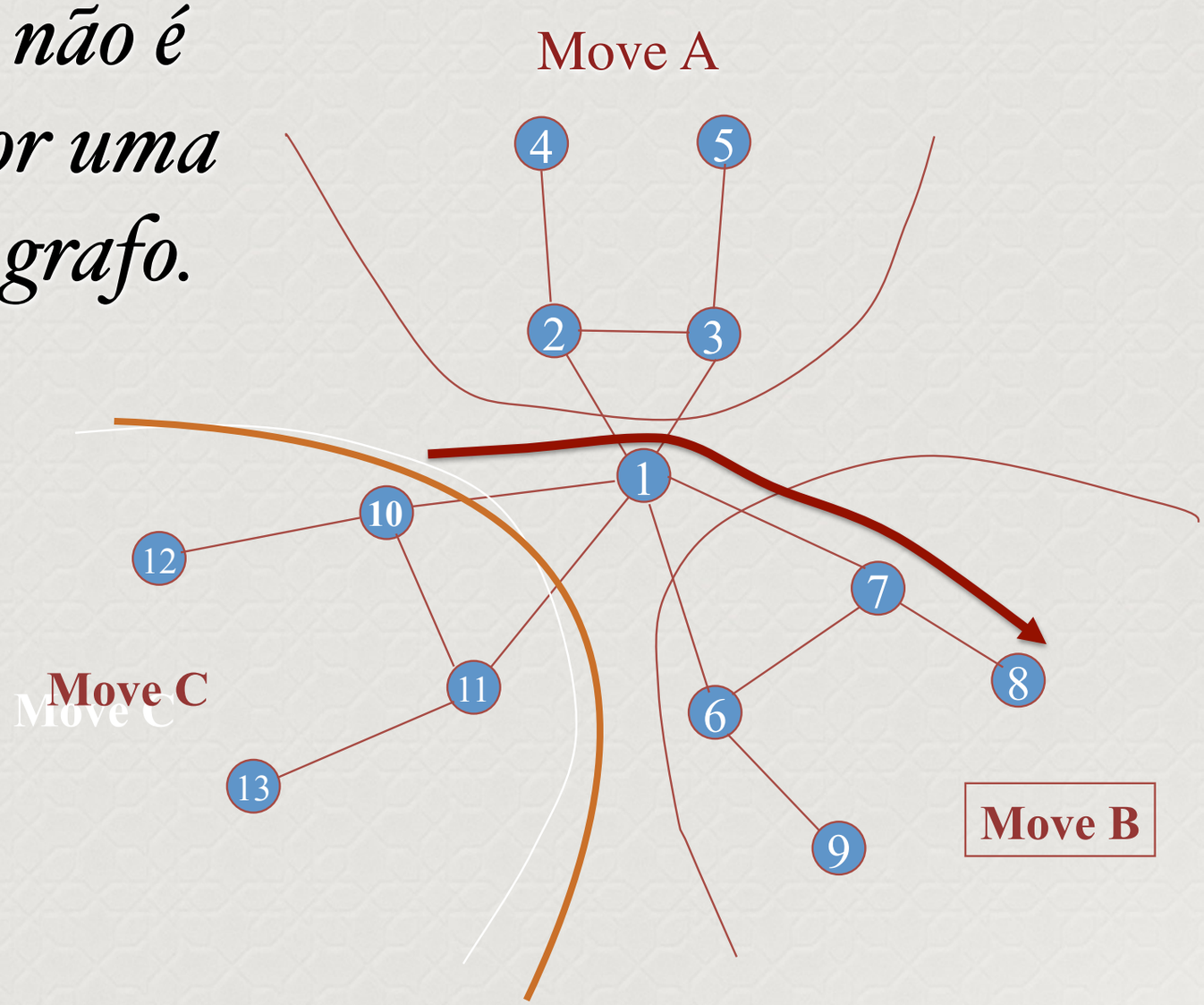


Vamos usar o problema modelo do mundo de blocos para "construir" mostrar o algoritmo geral de resolução de problemas.

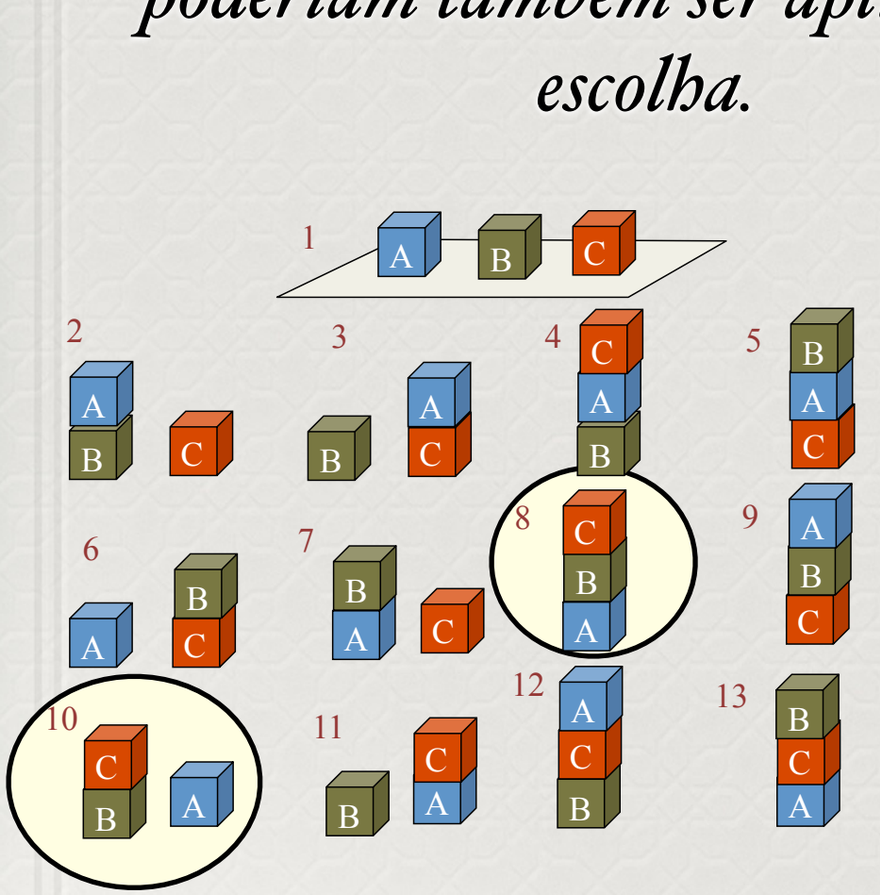
O Mundo de blocos

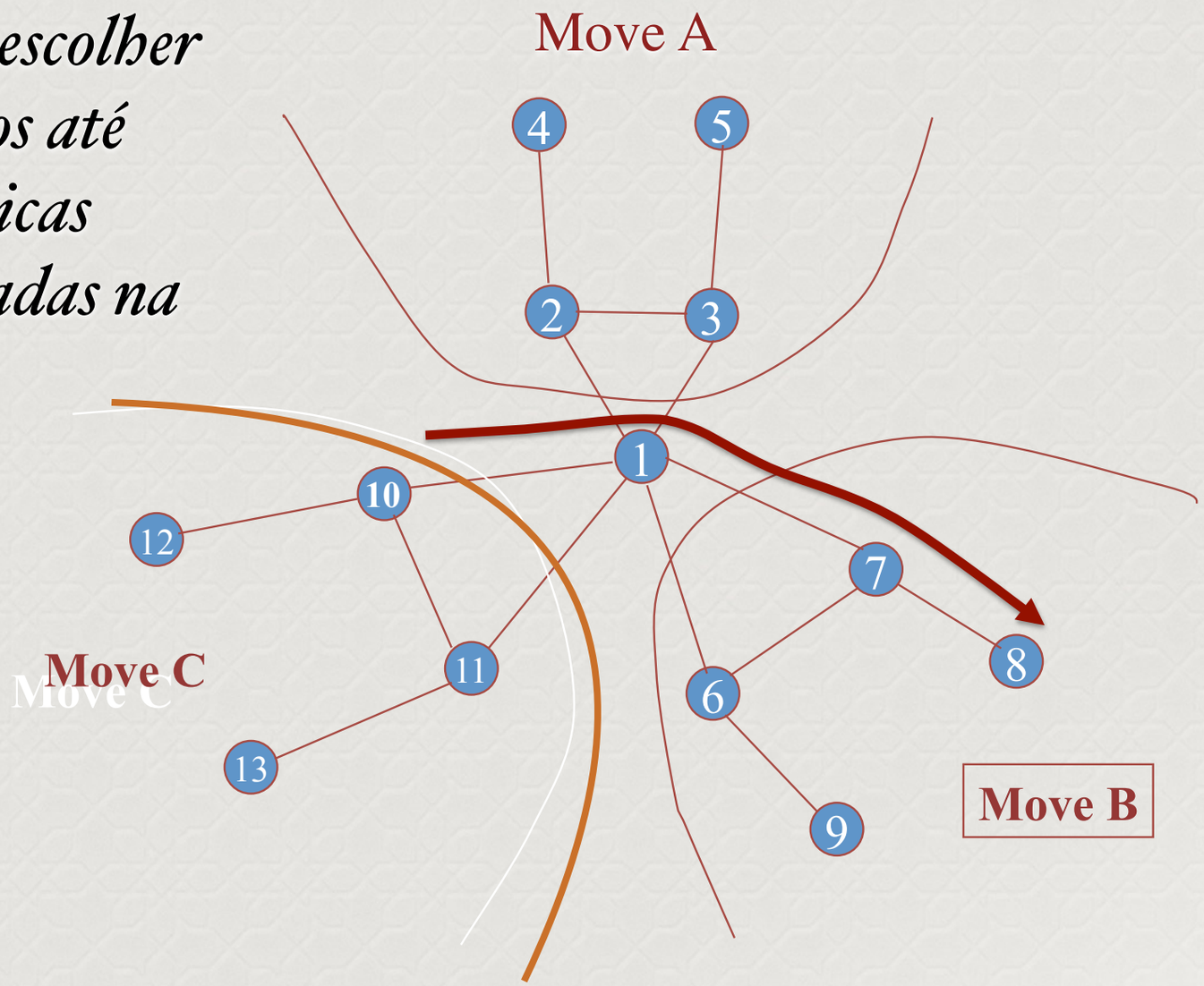
...o espaço de estados não é mais caracterizado por uma árvore e sim por um grafo.



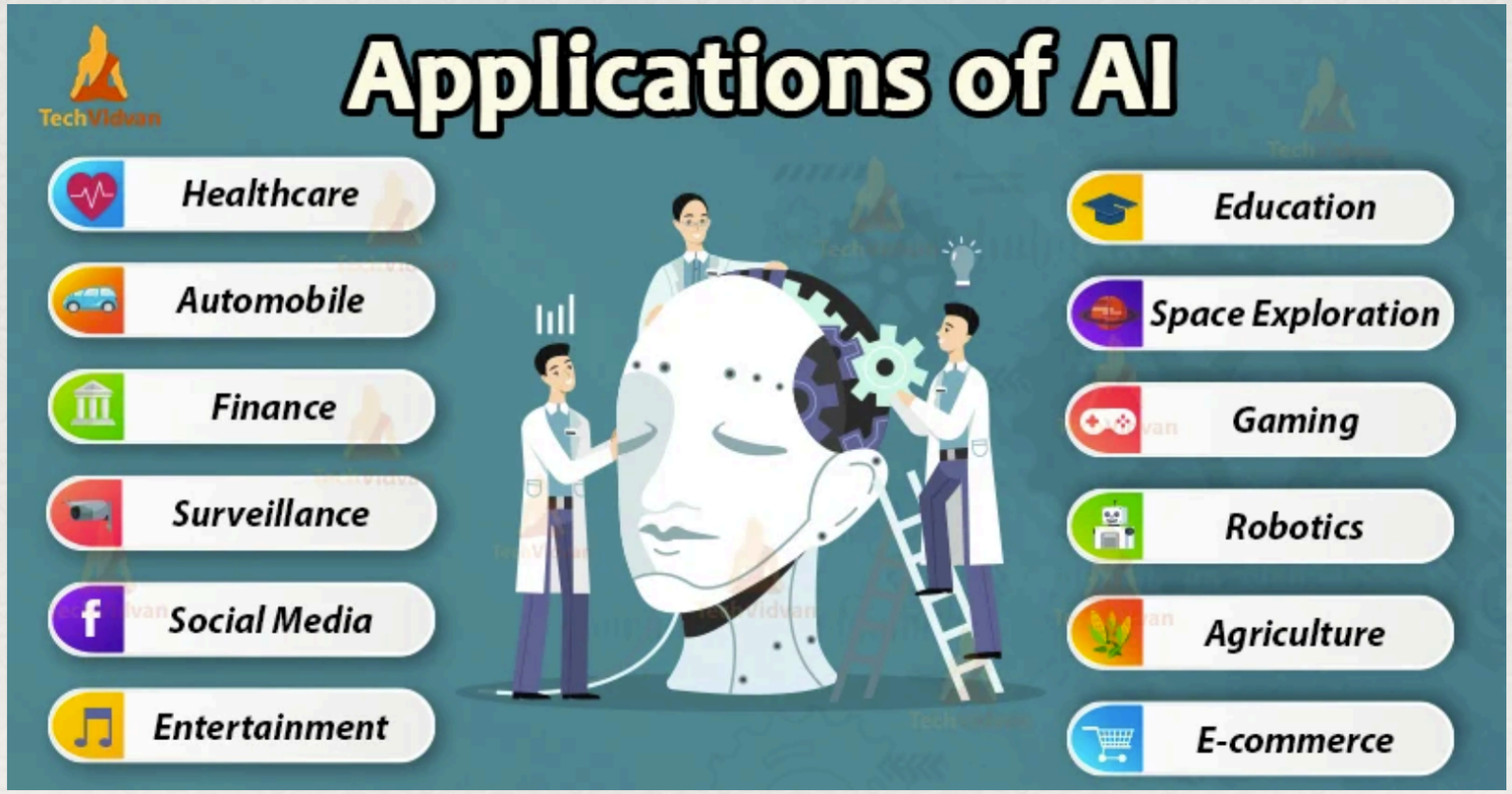


Resolver esse problema usando somente "busca" significaria escolher todas os possíveis caminhos até achar a solução. Heurísticas poderiam também ser aplicadas na escolha.

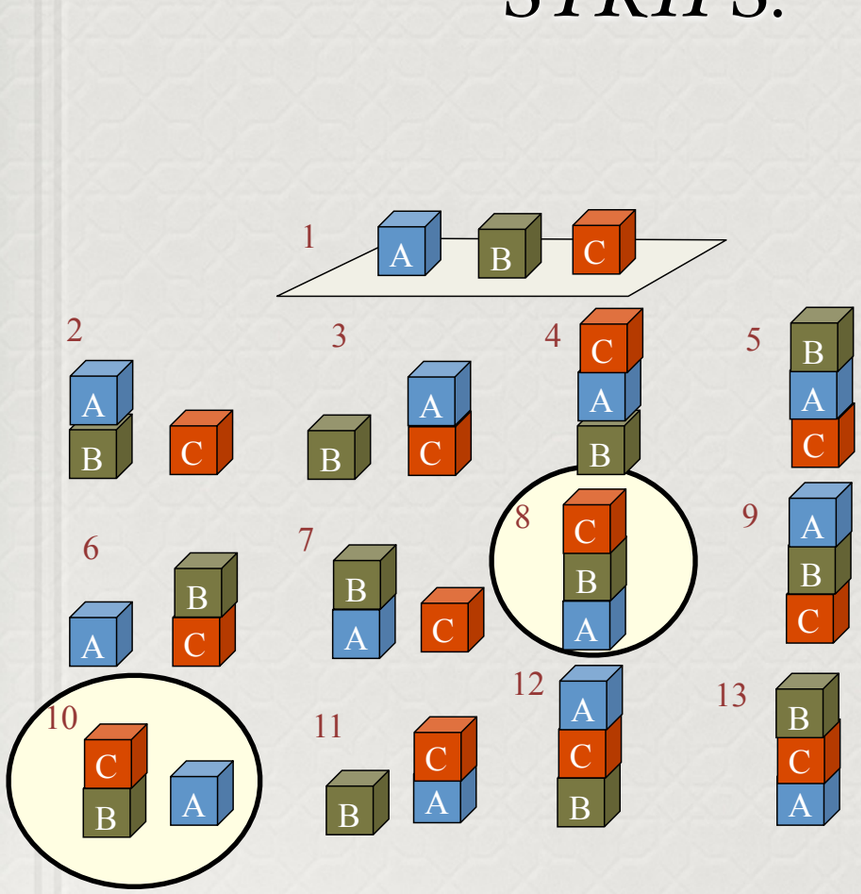


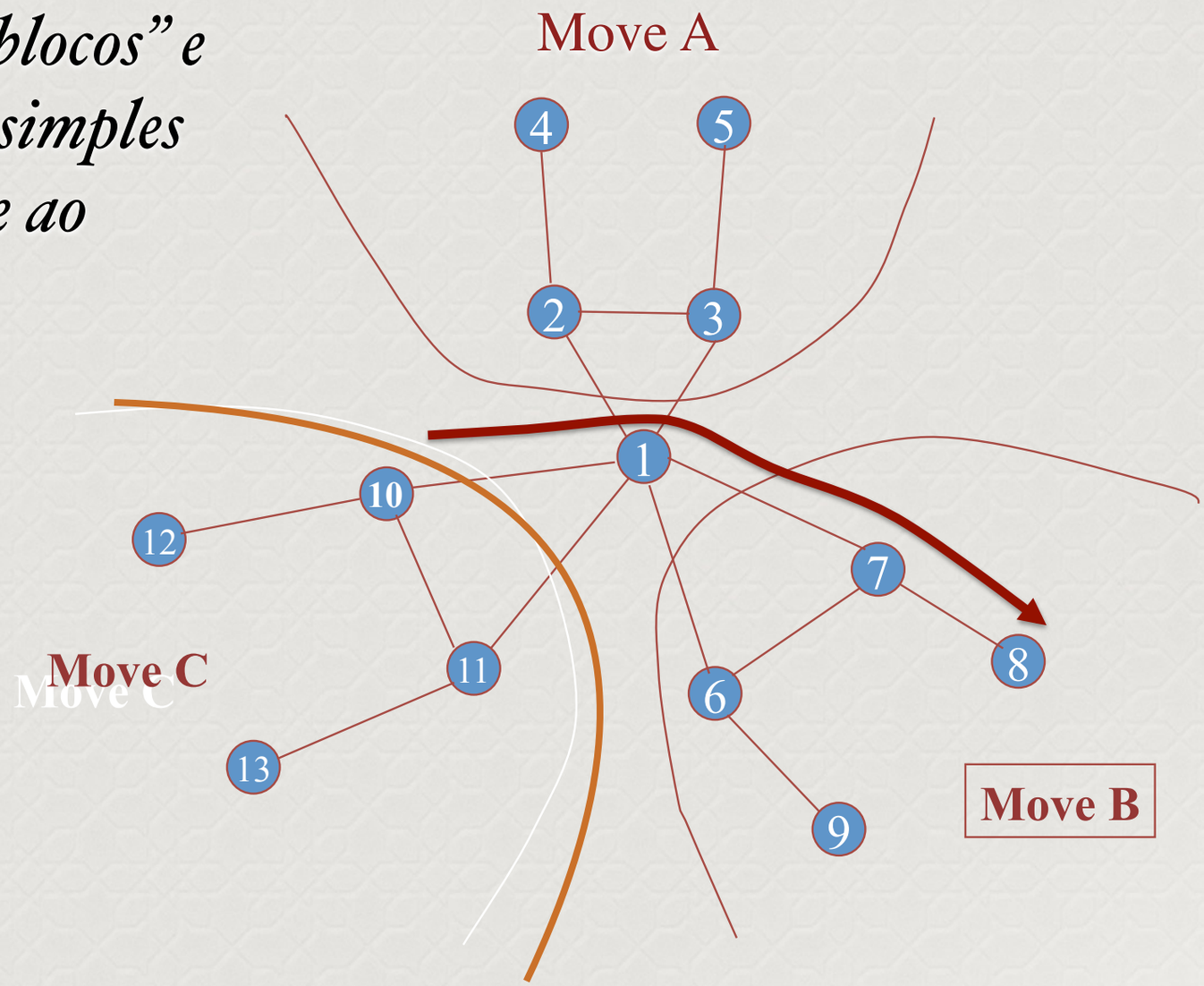


Será necessário considerar as diferenças de performance em cada método e também os critério de criatividade das aplicações.

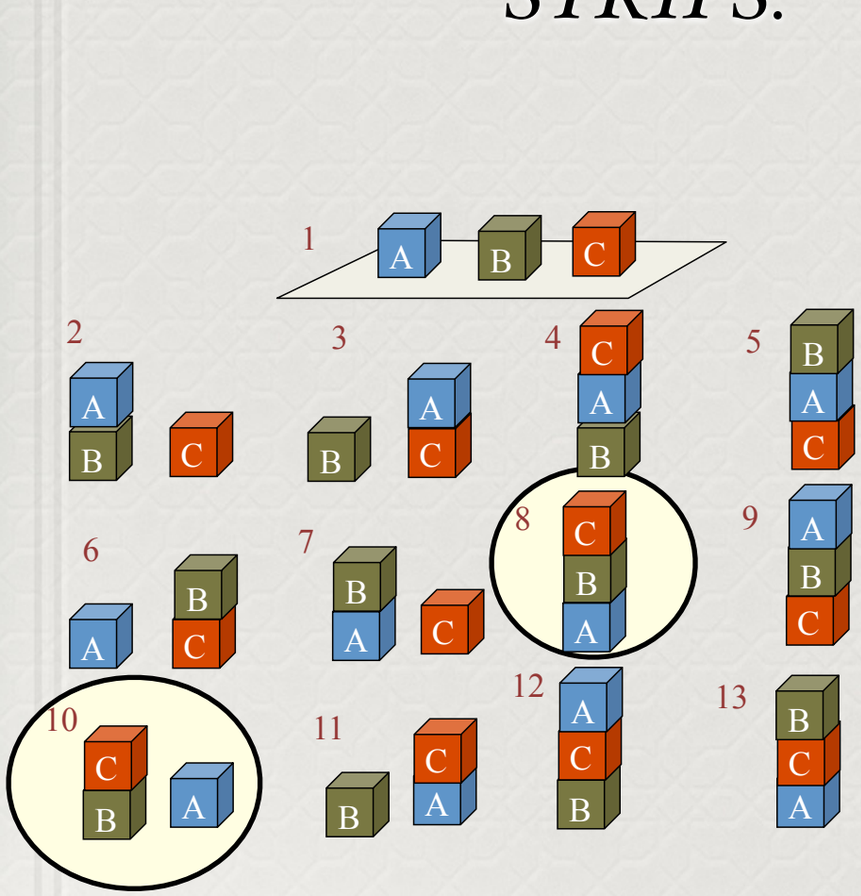


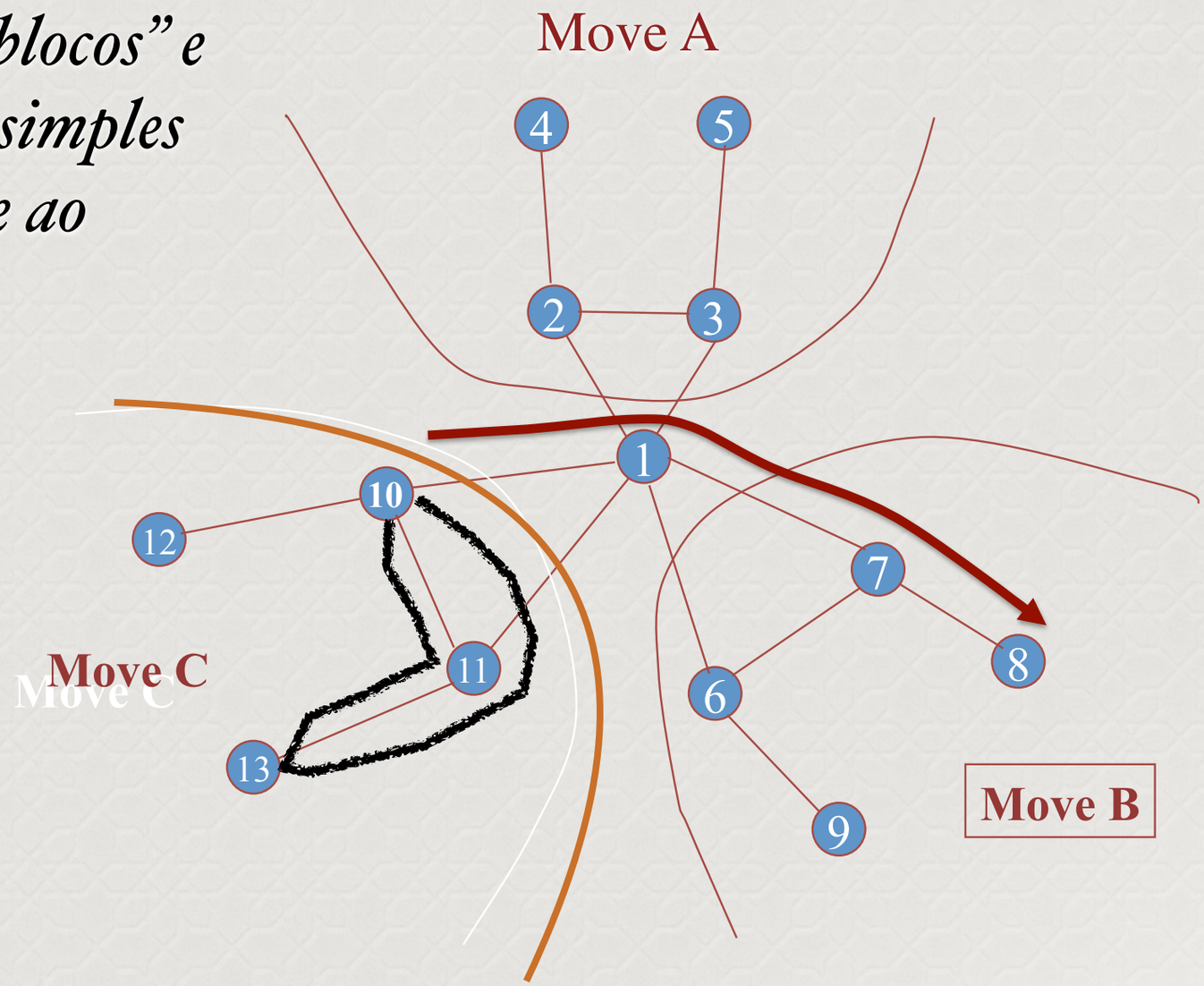
Vamos voltar ao "mundo de blocos" e considerar uma abordagem simples mas realística e aderente ao STRIPS.



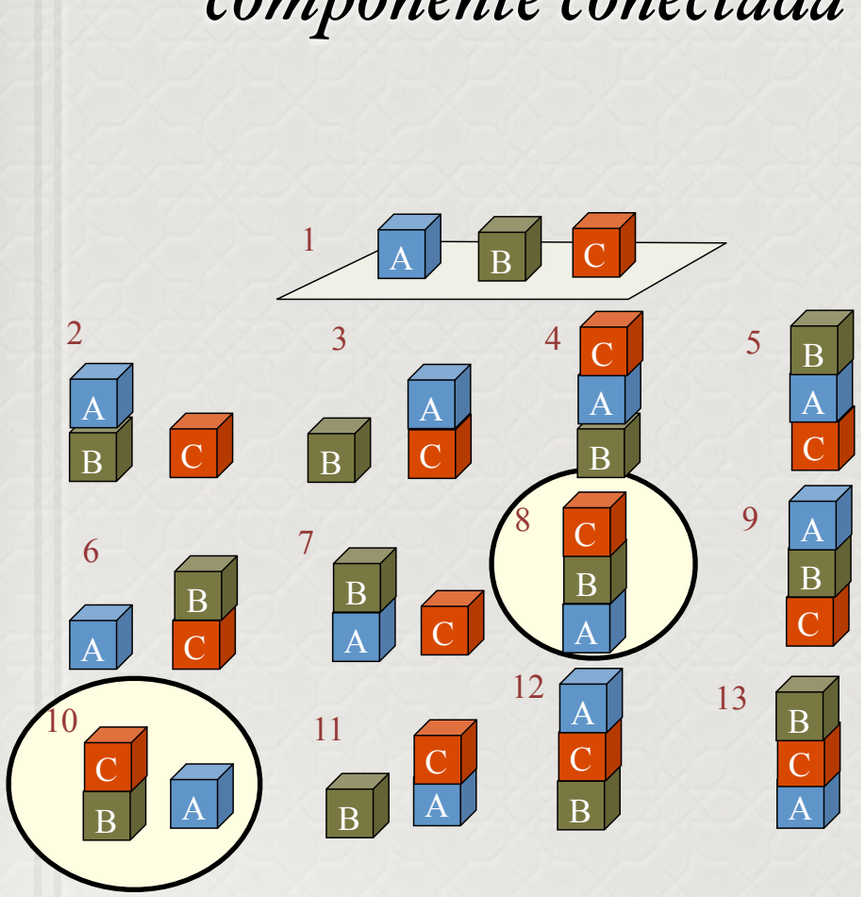


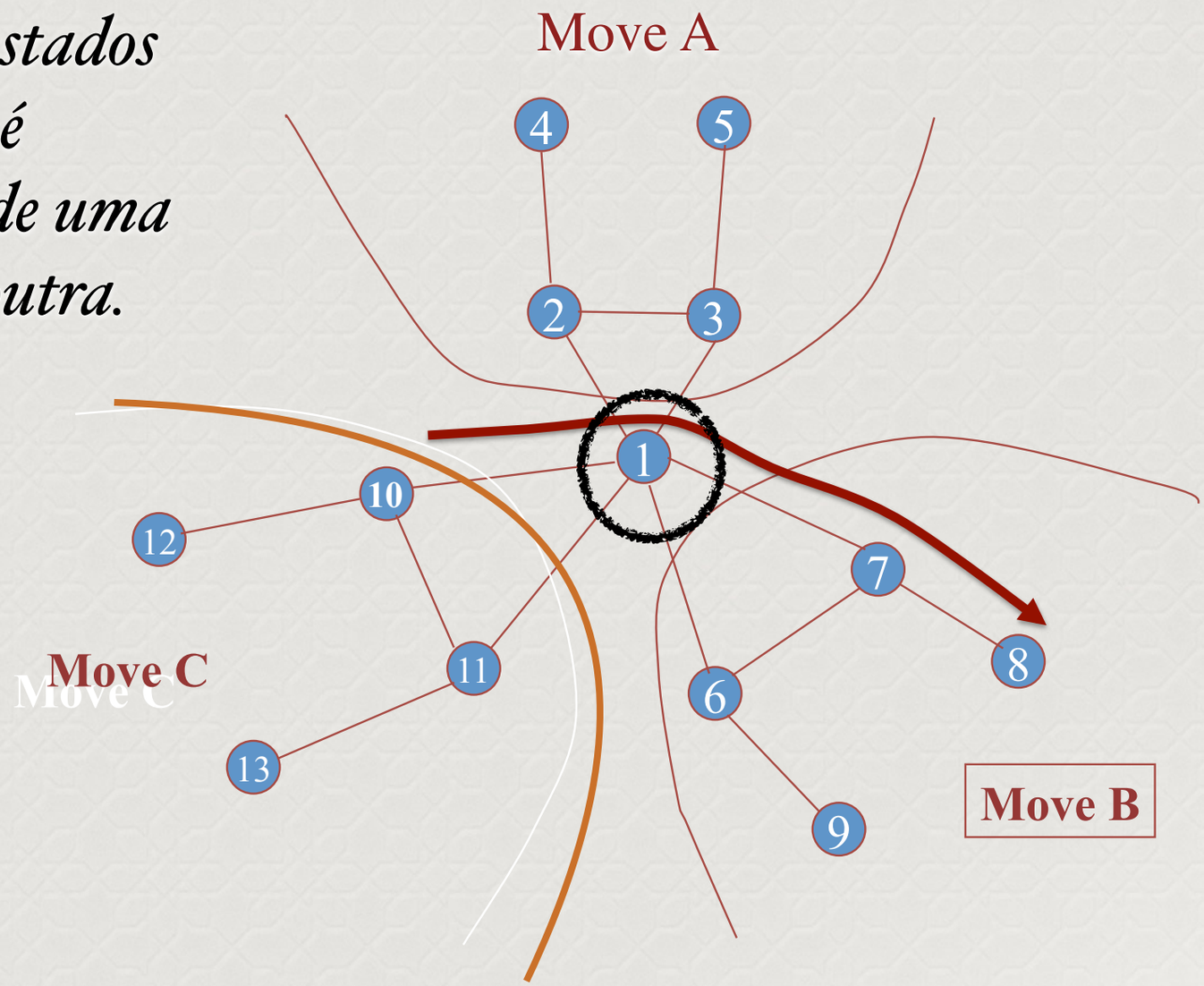
Vamos voltar ao "mundo de blocos" e considerar uma abordagem simples mas realística e aderente ao STRIPS.



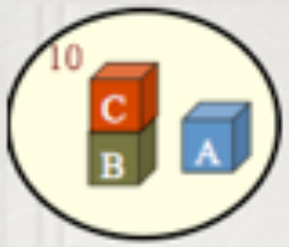


Uma análise do espaço de estados mostra que o estado 1 é "preferencial" para passar de uma componente conectada a outra.

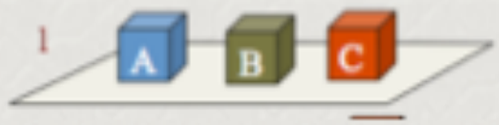




Representação dos "estados" em lógica: estado inicial e estado "preferencial.



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 inicial

estado objetivo

Conjunto de Ações

Problema de planejamento

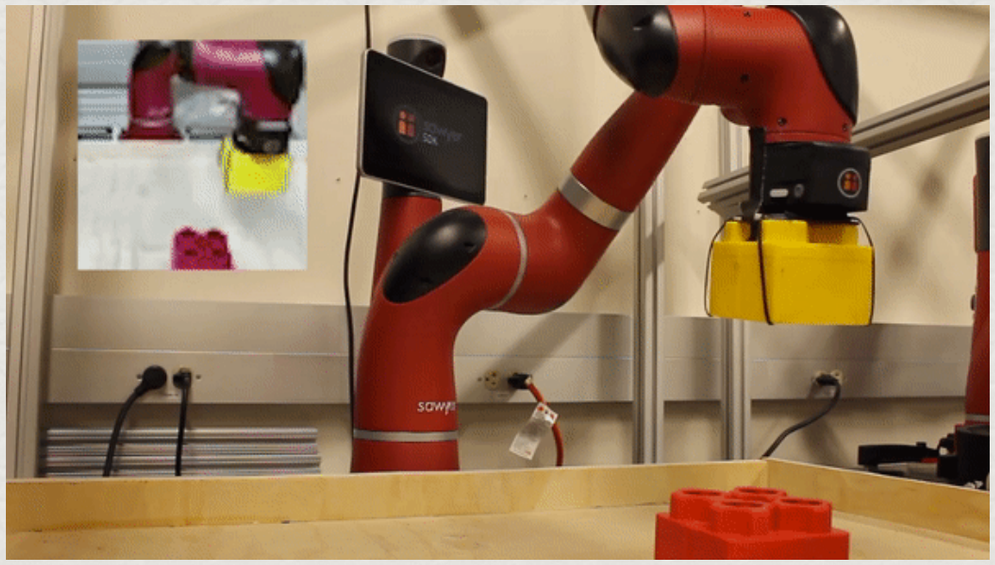
Critério de aplicação das ações

Percepção das mudanças de estado

Análise de custo

Base de conhecimento

O conjunto de ações é baseado nos movimentos de um braço robótico onde o efetuador (garra) só pode segurar um bloco de cada vez. A deposição só pode ser feita sobre um bloco livre, que não tenha nenhum outro bloco em cima dele.

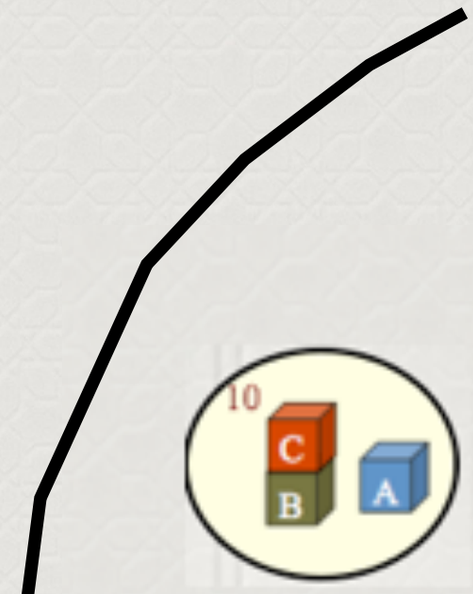


Uma ação desse tipo pode ser definida como $move(X, Y)$, onde um robô pode mover o bloco X para cima do bloco Y ou para a mesa.

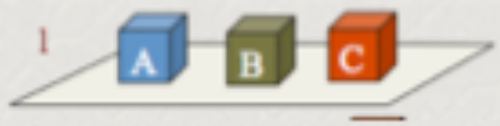
Pré-condições

Pós-condições

$move(c, mesa)$.



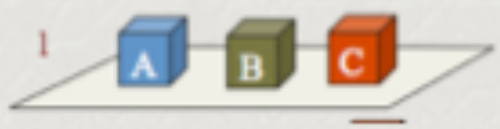
sobre(c, b)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(a)



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

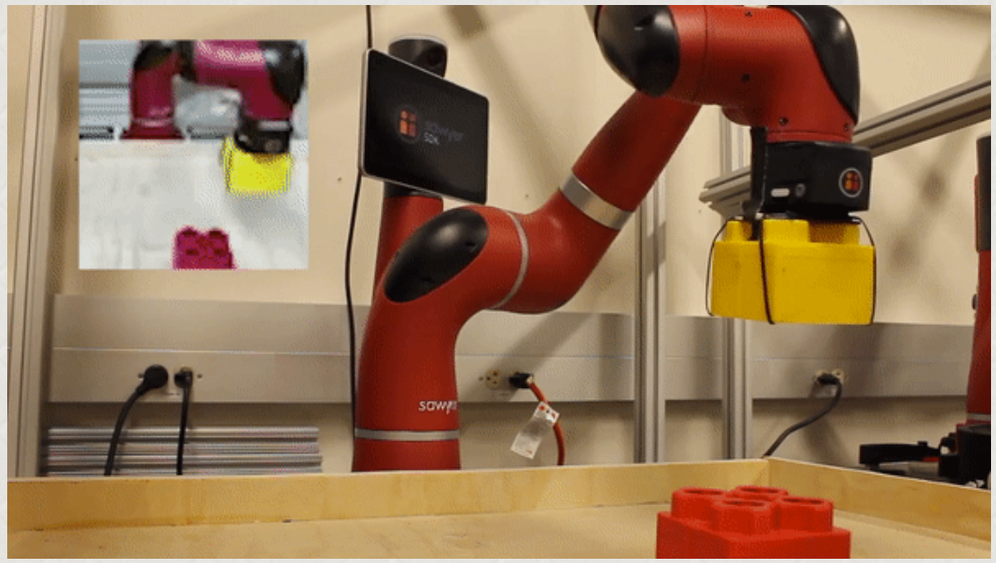
$$move(c, mesa)\ .$$

remove([sobre(c,b)] add([sobre(c,mesa), livre(b)]



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

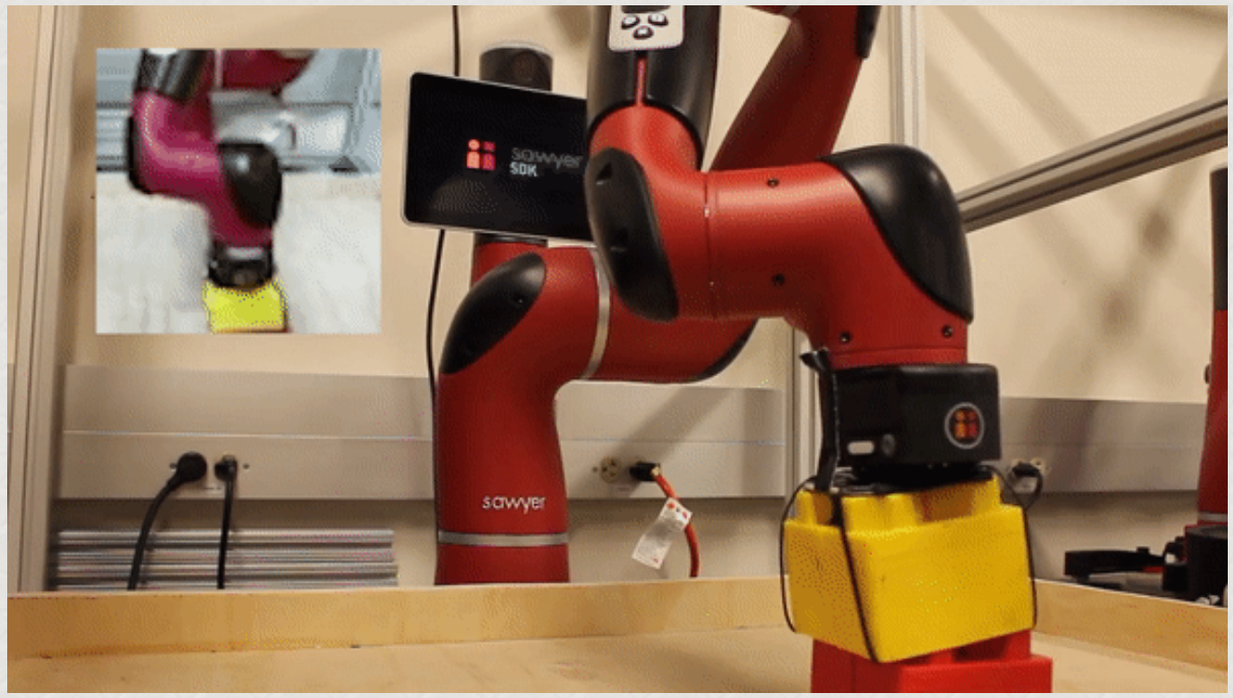
sobre(c, mesa)
sobre(b, mesa)
sobre(a, mesa)
livre(c)
livre(b)
livre(a)



$$move(X,Y) :- livre(X), livre(Y), livre(garra), remove(livre(Y)), add(sobre(X,Y)).$$

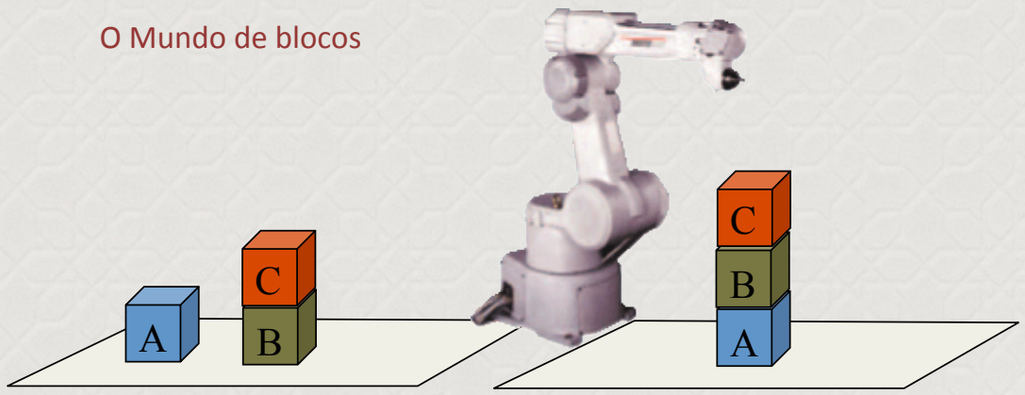
Intuitivamente as ações não são somente enumeradas, mas definidas como predicados, com regras para sua aplicação. O evento é aplicado identificando uma lista de termos que deve ser removida da descrição do estado origem, e acrescentando uma nova lista a esse estado para obter o estado destino.

Será que isso pode ser generalizado como um método para resolver uma classe ampla de problemas de planejamento?.



Na aula que vem vamos discutir essa possibilidade e apresentar mais detalhadamente o algoritmo genérico para resolver problemas conhecido como STRIPS.

Na aula que vem também aprofundaremos os aspectos práticos do problema modelo do mundo de blocos que será o nosso próximo exercício-programa.



O Mundo de blocos

Equipe 1

Bernardo Moredo Rocco
Victor Benito Pereira
Rahul Casanovas
Luan Gustavo de Brito

Equipe 2

João Victor Lins Fregnan
Victor Kendy Kamia
Vinicius Araújo da Costa
Gustavo Marangoni Rubo

Equipe 3

Lucas Nascimento Tulha
Marcelo Alonso Cordova
Vidal Gonzalo Rojas

Equipe 4

Daniel Willian Domingues
Gabriel Antonio Ken Chang
Mauricio Hiroki Tomida
Maykon Souza Cruz

Equipe 5

Giulia Duo Cardella
Guilherme Rodrigues Monteiro
Luis Fernando Ferreira da Silva
Yae Do Choi

Equipe 6

Gabriela Fonseca Fanucchi
Guilherme Henrique de Oliveira
Yuri Lopes Pamplona

Equipe 7

Lucas Oliveira Reis
Pablo Nabil Villar Bou Assi
Samuel Flávio de Paiva Araújo
Vinicius Rocha Paiva

Perguntas?