

Inteligência Artificial



Fabrício Enembreck

PPGIA – Programa de Pós-Graduação
em Informática Aplicada

1

Definições de Inteligência Artificial

☞ “estudo de como fazer os computadores realizarem coisas que, atualmente, as pessoas fazem melhor.” Rich e Knight

☞ Quatro definições de Russel e Norvig:

- Sistemas que pensam como humanos
- Sistemas que pensam racionalmente
- Sistemas que agem como humanos
- Sistemas que agem racionalmente

2

Pensando Humanamente

- Pesquisar como pessoas pensam
 - Empiricamente
 - Baseando-se em psicologia cognitiva
- Resolvedores gerais de problemas
- Modelos Comportamentais (observação)
- **Modelos Cognitivos** (estímulos, crenças, metas, raciocínio, ações)

3

Pensando Racionalmente

- Aristóteles: pensar racionalmente é um processo de raciocínio irrefutável.
- Aristóteles propôs os padrões de argumentos que representam raciocínios corretos. Esses **Silogismos** fazem parte da **Lógica**.
- Sócrates é um homem. Todos os homens são mortais. Logo, Sócrates é mortal.

4

Agindo Humanamente

- Turing 1950: comportamento inteligente é a capacidade de alcançar a performance humana em todas as tarefas cognitivas, suficientes para enganar um interrogador
- Seis capacidades básicas de um computador inteligente:
 - Processamento de linguagem natural
 - Representação de conhecimento
 - Raciocínio automatizado
 - Aprendizagem de máquina
 - Visão computacional
 - Robótica

5

Agindo Racionalmente

- Um agente deve agir para alcançar sua meta
- Percepção e ação
- **Agentes racionais**
- Inteligência pode ser um fenômeno social, não apenas um modelo de raciocínio intrínseco ao indivíduo

6

Um pouco de história

🕒 1943-1956 (O início)

- McCulloch e Pitts (1943): Modelo de neurônios cerebrais e redes de neurônios para representar conectivos lógicos e o processo de aprendizagem
- Shannon e Turing (1950): Programas de xadrez para computadores de von Neumann
- Minsky e Edmonds (1951): Construíram a primeira rede neural para computador em Princetown (3000 válvulas para simular 40 neurônios)

7

Um pouco de história (cont.)

- 🕒 John McCarthy (1955): Desenvolvimento da primeira linguagem funcional (LISP) para prova de teoremas. Convenceu Minsky e colegas a trabalhar em **Inteligência Artificial**.
- 🕒 Workshop de Dartmouth (1956) reuniu pesquisadores em teoria dos autômatos, redes neurais e estudo da inteligência.
- 🕒 Newell e Simon apresentaram um programa de raciocínio baseado em Lógica
- 🕒 MIT, CMU, Stanford e IBM.

8

Um pouco de história (cont.)

🐣 1952-1969 (Entusiasmo)

- Limitação da tecnologia
- Newell e Simon: Provador geral de teoremas para puzzles com estratégias de raciocínio
- Samuel (1952): Provador de teoremas para jogo de damas
- McCarthy (1958): Mudou-se para o MIT, criou o LISP, criou o time-sharing, criou o Advice Taker
- McCarthy (1963): Mudou-se para o MIT e aprimorou o Advice Taker com o método de resolução introduzido por Robinson
- Minsky e os micro-mundos

9

Um pouco de história (cont.)

🐣 1966-1974 (Realismo)

- Apesar das aplicações potenciais, os sistemas “inteligentes” da época eram muito especializados e problemas muito “pequenos”
- Problema: Aplicações não utilizavam conhecimento, mas apenas substituições sintáticas (weak-methods)
- Weizenbaum 1965 – ELIZA
- Teoria dos problemas NP-completos
- Friedberg (1959) – Estudos sobre algoritmos genéticos
- Representações muito limitadas de comportamento inteligente (Minsky e redes neurais)

10

Um pouco de história (cont.)

🐼 1969-1979: Sistemas a base de conhecimento

- Suprir a necessidade de conhecimento para aplicações de domínios específicos
- Buchanan et al. (1969): DENDRAL – Dada uma fórmula molecular + massas o sistema previa todas as fórmulas derivadas + massas quando a fórmula era bombardeada por um elétron usando um conjunto de regras
- Feigenbaum, Buchanan e Shortliffe (1972): **Sistema Especialista MYCIN** – diagnóstico de infecções sanguíneas c/ **tratamento de incertezas**

11

Um pouco de história (cont.)

🐼 1969-1979: Sistemas a base de conhecimento

- Duda et al. (1979): PROSPECTOR - Sistema especialista para descoberta de jazidas de chumbo
- Shank, Alberson, Riesbeck, Dyer: Conhecimento é necessário para se construir sistemas que compreendem linguagem natural
- Woods (1973): LUNAR – Primeiro sistema de PLN que respondia questões sobre rochas trazidas da Lua
- Esquemas de representação de conhecimento: Prolog (1972), Frames (Minsky, 1975), Redes Semânticas (Woods), Grafos Conceituais (Shank), etc.

12

Um pouco de história (cont.)

🐼 Primeiras aplicações comerciais (1980-1988)

- Sistemas especialistas
- Sistemas de visão computacional

🐼 O retorno de redes neurais (1986 - ...)

🐼 Hoje em dia

- Reconhecimento de Padrões (voz, imagem,som)
- Raciocínio Incerto (Fuzzy, Probabilista)
- Processamento em Linguagem Natural
- Mineração e aquisição de conhecimento a partir de dados
- Inteligência Artificial Distribuída – Agentes Inteligentes
- Programação Genética/Algoritmos Genéticos
- Redes de sensores/coordenação de entidades autônomas
- Tecnologias da Informação – Pesquisa/Filtragem/Classificação
- Jogos Inteligentes, ...

13

Resolução de Problemas e Busca

Capítulo 2

14

Construção de um Sistema para Solução de um Problema

- Definir o problema precisamente
- Determinar o tipo de conhecimento necessário para resolver o problema.
- Criar um esquema da representação para tal conhecimento
- Escolher (ou desenvolver) técnicas de solução adequadas para solução do problema.

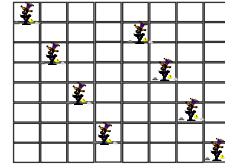
15

Definição do Problema

- Considerações:
 - Representação Computacional do problema
 - Objetivo (o que se pretende alcançar)
 - Onde Iniciar
 - Como modificar os estados
 - Como identificar modificações úteis na solução do problema

16

Problemas Difíceis



- Entendimento de Inglês.
- Jogar Xadrez.
- Resolver Integrais Indefinidas.
- Prever o clima.
- Prever mudanças no estoque de uma loja
- Organizar uma linha de produção
- Acomodar objetos dentro de um espaço físico limitado

17

Generalização do Problema

- É útil desenvolver métodos gerais para solução de problemas.
- Dessa forma torna-se fácil comparar problemas (especialmente o tamanho).
- Pode ser desenvolvido um conjunto de estratégias para solução de problemas e qualquer uma pode ser aplicada.
- É necessário, porém, escolher a metodologia mais adequada.

18

Definição do Problema como um Espaço de Estados (Cont.)

- Uma possível estratégia para solução de problemas é listar todos os estados possíveis.
- A solução do problema consiste em percorrer o espaço de estados a partir do estado inicial até o estado meta.
- É necessário desenvolver um conjunto de operadores que modifique um estado para um outro estado.

19

Definição do Problema como um Espaço de Estados (Cont.)

- Conjunto de Estados possíveis.
- Conjunto de operações possíveis que modifiquem um estado.
- Especificação de um estado *inicial(s)*.
- Especificação de um estado *meta(s)*.

20

Espaço de Estados

- Geralmente não é possível listar todos os espaços possíveis:
 - utilização de abstrações para descrever estados válidos.
 - pode ser mais fácil descrever estados inválidos.
 - algumas vezes é útil fornecer uma descrição geral do espaço de estados e um conjunto de restrições.

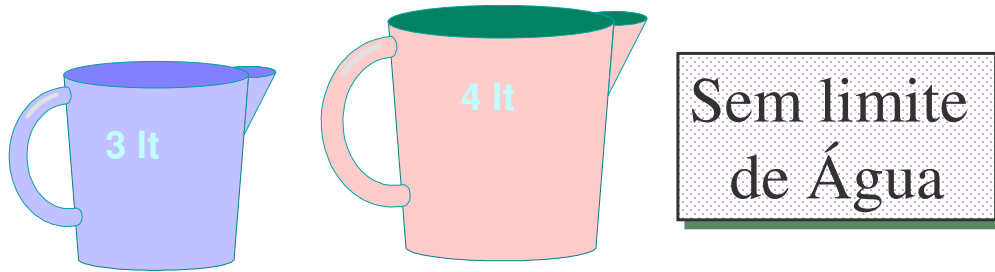
21

Operações

- O sistema de solução se move de um estado para outro de acordo com operações bem definidas.
- Geralmente estas operações são descritas como *regras*.
- Um sistema de controle decide quais regras são aplicáveis em um dado estado e resolve conflitos e/ou ambiguidades.

22

Exemplo: Problema dos Jarros de Água



Objetivo: 2 litros no jarro 4 lt

23

Espaço de Estados do Problema dos Jarros de Água

O espaço de estados pode ser representado por dois inteiros x e y :

x = litros no jarro de 4 litros

y = litros no jarro de 3 litros

Espaço de Estados = (x,y)

tal que $x \in \{0,1,2,3,4\}$, $y \in \{0,1,2,3\}$

24

Início dos Jarros de Água e Estados Meta

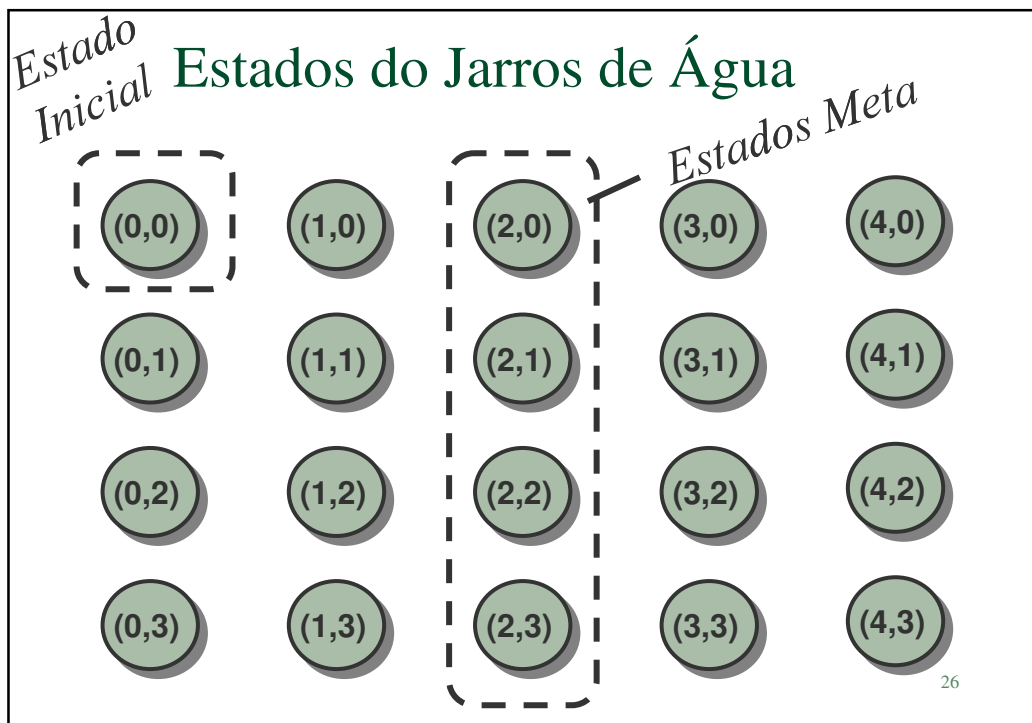
- O Estado Inicial ocorre quando ambos os jarros estão vazios:

$$(0,0)$$

- O Estado Meta é qualquer estado que possua 2 litros de água no jarro de 4 litros:

$$(2,n) \text{ para qualquer } n$$

25



Operações com Jarros de Água

- Colocar 3 lt. no jarro 3 $(x,y) \rightarrow (x,3)$
- Colocar 4 lt. no jarro 4 $(x,y) \rightarrow (4,y)$
- Esvaziar jarro 3 $(x,y) \rightarrow (x,0)$
- Esvaziar jarro 4 $(0,y) \rightarrow (y,0)$
- Coloca o conteúdo do jarro 3 no jarro 4
- Outros ???

27

Restrições

- Não é possível colocar água em um jarro cheio.
- Restrições são associadas para que uma operação possa ser aplicada sobre um estado

28

Regras de Produção

- Uma operação e as condições que devem ser satisfeitas (restrições) antes da operação poder ser aplicada é chamada de *regra*.
- Tipicamente é necessário mesclar regras gerais e regras específicas.
- Informação prévia sobre a solução tende a produzir regras específicas e aumentar a velocidade da busca.

29

Domínio de Regras Específicas

- Para o problema dos Jarros de Água :
 - $(0,2) \rightarrow (2,0)$
 - $(x,2) \rightarrow (0,2)$
- Utilizando estas regras soluções podem ser encontradas rapidamente!

30

Sistema de Produção

• Um *Sistema de Produção* é formado por:

- Um conjunto de regras.
- Algum mecanismo para representar o estado do sistema (uma base de dados).
- Uma estratégia de controle que controla a ordem na qual regras da base de dados são comparadas e a solução de conflitos.
- Um mecanismo que aplica as regras.

31

Estratégias de Controle

• Pré-requisitos Gerais:

- Movimentação:
 - aplicação da mesma regra sucessivamente não é provavelmente útil.
 - É necessário mudar um estado para fazer progresso.
- Sistemática
 - opera à medida que explora novas regiões no espaço de estados.
 - Evita busca circular (loops).

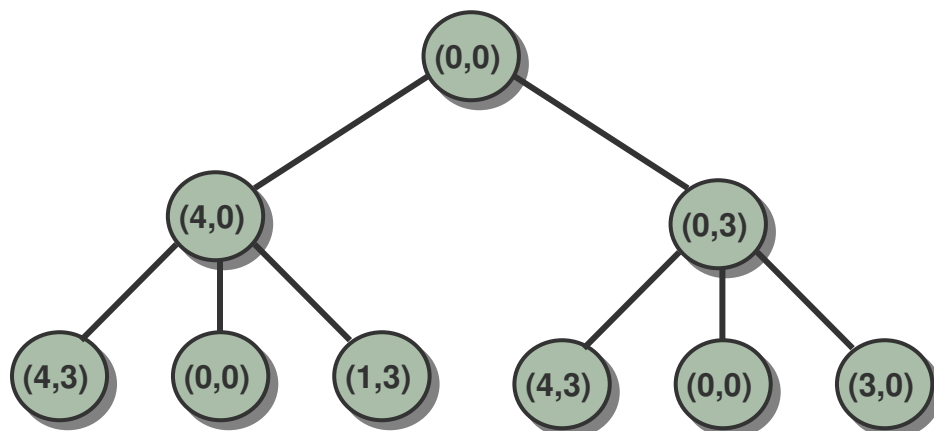
32

Estratégias de Controle para os Jarros de Água

- Lista ordenada de regras - aplica a primeira regra adequada para determinada situação.
- Escolhe qualquer regra (randomicamente) adequada para determinada situação.
- Aplica todas as regras adequadas, e armazena o caminho resultante de todos os estados obtidos. No próximo passo faz o mesmo para todos os estados.

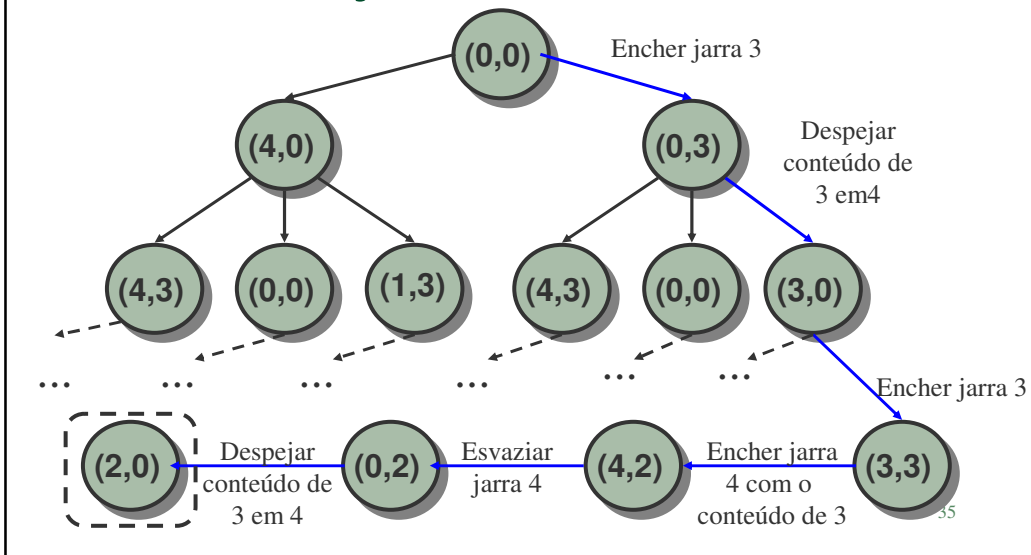
33

Árvore dos Jarros de Água



34

Árvore dos Jarros de Água – Exemplo de uma solução



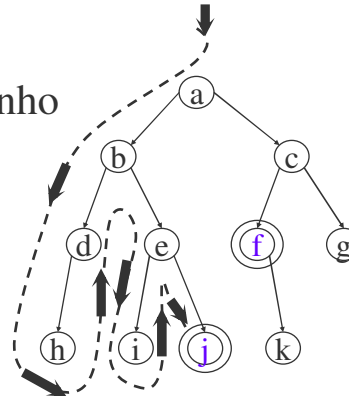
Busca em Profundidade

🐼 Algoritmo (BP)

- Estabeleça alguma ordenação às regras;
- **Enquanto** existem regras aplicáveis
 - Aplique a próxima regra e gere um novo estado;
 - Faça uma busca em profundidade (BP) no novo estado.
- **fim_enquanto**

Características da Busca em Profundidade

- Não necessita armazenar o caminho de uma grande lista de estados.
- Pode encontrar uma solução muito rapidamente.
- “Poda” é possível
Exemplo: utilização de heurísticas
- Pode facilmente encontrar problemas com ciclos (loops).



37

Busca em Profundidade em Prolog

```
profundidade(A,[A]) :- % se o estado é meta, então pára
    meta(A).
profundidade(A,[A|B]) :- % senão continua a busca no novo estado
    novo_estado(A,C),
    profundidade(C,B).
```

* *meta(A)* verifica se *A* é uma solução

Obviamente, o algoritmo acima não implementa o tratamento de ciclos, porém, ele pode ser facilmente modificado, armazenando o caminho percorrido, de forma a tratar o problema adequadamente.

38

Busca em Profundidade - Exercício

Baseado no programa Prolog de BP apresentado anteriormente, crie uma outra versão que resolva o problema dos ciclos.

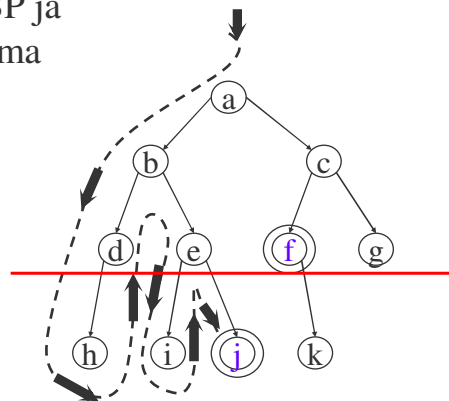
Solução:

```
profundidade(No,Solucao) :-  
    profundidade(No,[],Solucao).  
profundidade(No,_,[No]) :-  
    meta(No).  
profundidade(No,Caminho,[No|Solucao]) :-  
    \+ pertence(No,Caminho),  
    novo_estado(No,Novo),  
    profundidade(Novo,[No|Caminho],Solucao).
```

39

Busca em Profundidade - Exercício

Considerando os programas de BP já apresentados, escreva um programa Prolog que realize busca em profundidade com limitação de profundidade (fornecida como parâmetro)



40

Busca em Profundidade - Resolução do Exercício Anterior

```
profundidade(No,Solucao,Prof) :-  
    profundidade(No,[],Solucao,Prof).  
profundidade(No,_,[No],_) :-  
    meta(No).  
profundidade(No,Caminho,[No|Solucao],Prof) :-  
    Prof > 0,  
    Prof1 is Prof -1,  
    \+ pertence(No,Caminho),  
    novo_estado(No,Novo),  
    profundidade(Novo,[No|Caminho],Solucao,Prof1).
```

41

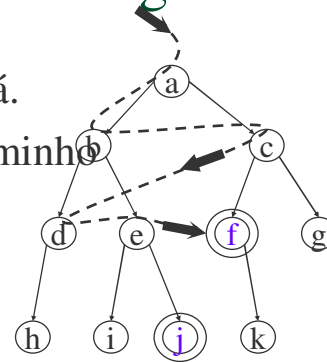
Busca em Largura

- A estratégia de árvore para o problema dos jarros de água é um exemplo de *busca em largura*.
- Algoritmo geral para BL:
 - crie *lista_nós* e a inicialize com o estado inicial;
 - enquanto um estado meta não é encontrado ou *lista_nós* != {};
 - remova o primeiro elemento de *lista_nós*, *primeiro_nó*;
 - aplique todas as regras possíveis em *primeiro_nó* e adicione os estados resultantes em *lista_nós*;
 - fim_enquanto

42

Características da Busca em Largura

- Se há uma solução, BL a encontrará.
- Encontrará a solução mínima (o caminho mais curto até a solução).
- Não terá problemas com ciclos.
- Requer espaço disponível para armazenar *lista_nós*, que pode ser muito grande!!!



43

Busca em Largura - Exercício

Escreva um programa Prolog que realize busca em largura num grafo semelhante à figura do grafo anterior

Exemplo:

```
?- largura([a],[f,j],Cam).
```

```
Cam = [a,b,c,d,e,f]
```

```
yes
```

44

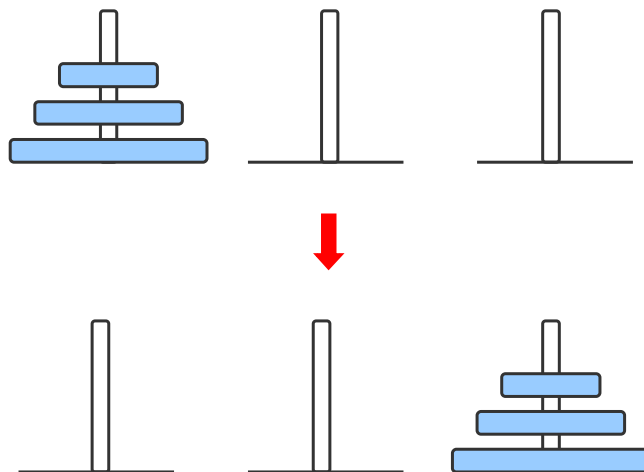
Busca em Largura - Resposta ao Exercício Anterior

```
largura([No|_],L,[No]) :-  
    pertence(No,L),!.  
largura([No|Lista],N,[No|L]) :-  
    filhos_sem_repeticoes(No,Lista,Filhos),  
    concatena(Lista,Filhos,Nos),  
    largura(Nos,N,L).
```

* *filhos_sem_repeticoes(No,Lista,F)* retorna em *F* os filhos de *No* que não pertencem a lista *Lista*

45

Outro problema: As torres de Hanói



46

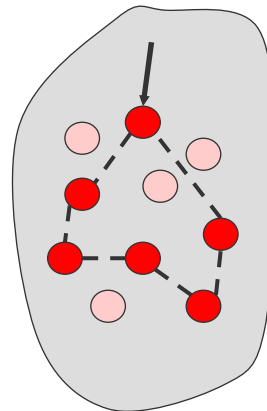
Exercício

- 1) Construa a árvore de busca para o problema das torres de Hanói
- 2) Responda a questão: Qual método encontrará a solução mais rapidamente (largura ou profundidade)? Demonstre usando a árvore da questão anterior.

47

Outro Problema

- Problema do Caixeiro Viajante
 - Lista de cidades para visitar
 - Lista de distâncias entre cada cidade
 - Visite cada cidade apenas uma vez
 - Encontre o menor trajeto
- Descreva estratégias de busca BL and BP que podem ser utilizadas neste problema



48

Busca Heurística



49

Busca Heurística

- Muitos problemas possuem espaços de busca que são muito grandes para serem examinados completamente.
- É possível construir estratégias que não prometem a melhor solução, mas que encontram uma “boa” resposta rapidamente.

50

Heurística

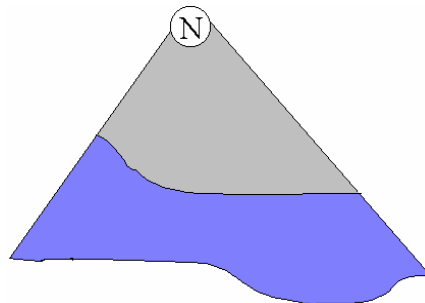
☛ Sobre busca Heurística:

- estimativa numérica da qualidade dos nós (em termos de possibilidade de dirigir a busca para uma solução)
- execução da busca sobre o nó mais promissor
- Avaliação dos nós pode ser baseada em uma informação incompleta.
- Uma *heurística* é usada para guiar o processo de busca.

51

Busca Heurística

- ## ☛ Estratégias de Busca Heurística usam
- informação do domínio para limitar a busca sobre áreas onde podem existir soluções.



52

Exercício

🦉 Crie heurísticas para:

- o que estudar para uma prova
- como resolver uma prova com questões objetivas
- qual será o clima de amanhã
- como vencer no tic-tac-toe.

53

Heurísticas para o problema do Caixeiro Viajante

🦉 Heurística do vizinho mais próximo:

1. Selecione qualquer cidade inicial
2. Enquanto houver cidades
escolha a cidade mais próxima à cidade corrente.

54

Utilização de Busca Heurística

- A solução não precisa ser ótima (qualquer solução é uma boa solução).
- O pior caso é raro.
- Entendimento de quando e como a busca heurística trabalha leva a um melhor entendimento do problema.

55

Usando conhecimento heurístico em um sistema baseado em regras

- Conhecimento Heurístico pode ser codificado diretamente em regras. Ex.: No problema dos jarros de água - não colocar água em um jarro cheio
- Pode ser uma função heurística que avalia estados do problema e impõe uma ordenação
- Xadrez - qual movimento deve ser explorado primeiro

56

Características de Problemas

• É útil identificar características para que um conjunto formado por estratégias aplicadas em problemas com características semelhantes possa ser obtido

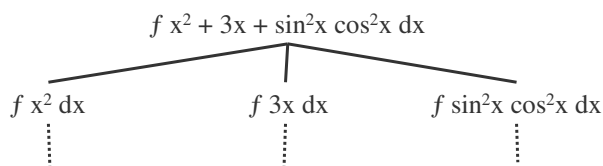
- Decomposição
- Passos ignorados ou desfeitos
- O universo é previsível?
- Precisa de uma solução ótima?
- A solução é um estado ou um caminho?
- Qual o papel do conhecimento?
- Humanos podem interagir com o sistema?

57

1. Decomposição

• O problema pode ser quebrado (decomposto) em vários problemas independentes de tamanho menor ?

• Sim - recursão pode ser utilizada, apenas os pequenos problemas são atacados diretamente



58

2. Passos podem ser ignorados ou desfeitos?

• Classes de problemas:

- Ignoráveis - passos da solução podem ser ignorados (a estratégia de busca não muda)
- Recuperáveis - passos podem ser desfeitos (retrocesso)
- Irrecuperáveis - passos não podem ser desfeitos

59

3. O Universo é previsível ?

- Algumas vezes o sistema controla completamente os estados e sabe exatamente a desvantagem de cada operação
- Outras vezes a desvantagem não é completamente previsível

60

4. Uma boa solução é relativa ou absoluta?

- Únicos vs. múltiplos caminhos para uma única solução
- Na busca pela solução absoluta não se pode utilizar heurística (todos os caminhos possuem possibilidades iguais e precisam ser percorridos)

61

5. A solução é um estado ou um caminho?

- Algumas vezes a solução é apenas um estado do espaço de estados
- Algumas vezes o caminho para o estado final é a resposta
- Todos os problemas podem ser reformulados para que a resposta seja apenas um estado

62

6. Qual é o papel do conhecimento?

- Alguns problemas usam o conhecimento para restringir a busca (a estratégia de busca faz parte do conhecimento)
- Outros problemas necessitam de conhecimento para avaliar estados (poder reconhecer uma solução)

63

7. Humanos podem interagir com o processo de busca?

- Algumas vezes humanos podem ajudar
- Humanos tendem a não confiar em sistemas se não entendem a lógica de como as respostas foram obtidas

64

Tipos de Problemas

- Várias características podem ser utilizadas para classificar problemas
- Existem técnicas adequadas para cada tipo de problema
- Exemplos de problemas genéricos:
 - classificação
 - Propor e refinar (problemas de projeto e planejamento)

65

Programas de Busca

- Busca pode ser vista como a travessia de uma árvore
- Árvore é formada pelas regras
- Representação Implícita vs. Explícita
- Algumas vezes é melhor pensar em busca como um grafo dirigido

66

Técnicas de Busca Heurística

Capítulo 3

- Gerar-e-testar
- Subida da Encosta
- Busca Best-First
- Redução de Problemas
- Satisfação de Restrições
- Análise Meios-Fins

67

Técnicas de Busca Heurística

- *Técnicas Diretas* nem sempre estão disponíveis (requerem muito tempo e memória)
- *Técnicas Fracas* podem ser suficientes se aplicadas corretamente (sobre o tipo correto de tarefas)
- Técnicas podem ser descritas genericamente

68

Gerar-e-testar

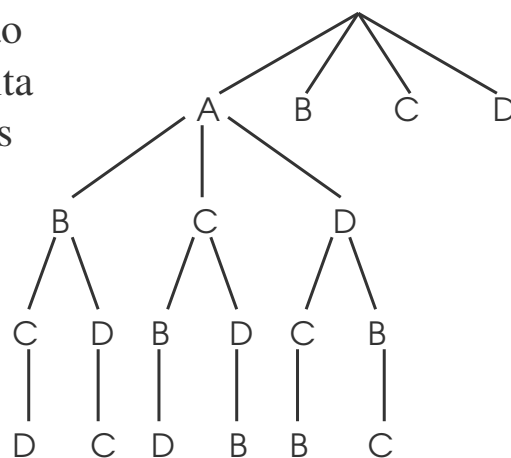
- Estratégia muito simples - baseada em previsão
 - enquanto a meta não for alcançada faça
 - gere uma solução possível
 - compare a solução com a meta
- Heurísticas podem ser usadas para determinar as regras específicas para geração de soluções

69

Exemplo de Gerar-e-Testar

- Caixeiro Viajante - geração de possíveis soluções é feita na ordem lexicográfica das cidades:

- 1. A - B - C - D
- 2. A - B - D - C
- 3. A - C - B - D
- 4. A - C - D - B
- ...



70

Problema dos Jarros de Água

- | | |
|--|--------------------|
| 1. $(x,y), x < 4 \rightarrow (4,y)$ | enche o jarro 4 |
| 2. $(x,y), y < 3 \rightarrow (x,3)$ | enche o jarro 3 |
| 3. $(x,y), x > 0 \rightarrow (0,y)$ | esvazia o jarro 4 |
| 4. $(x,y), y > 0 \rightarrow (x,0)$ | esvazia o jarro 3 |
| 5. $(x,y), x+y \geq 4, y > 0 \rightarrow (4, y-(4-x))$ | enche 4 com 3 |
| 6. $(x,y), x+y \geq 3, x > 0 \rightarrow (x-(3-y), 3)$ | enche 3 com 4 |
| 7. $(x,y), x+y \leq 4, y > 0 \rightarrow (x+y, 0)$ | conteúdo de 3 em 4 |
| 8. $(x,y), x+y \leq 3, x > 0 \rightarrow (0, x+y)$ | conteúdo de 4 em 3 |
| 9. $(0,2) \rightarrow (2,0)$ | 2 lt. de 3 para 4 |
| 10. $(x,2) \rightarrow (0,2)$ | esvazia 4 |

71

Subida da Encosta

• Variação sobre gerar e testar:

- *geração* do próximo estado depende do *procedimento de teste*
- *Teste* agora inclui uma função heurística que fornece uma previsão de quão bom é cada estado

• Há várias formas de utilizar a informação retornada pelo *procedimento de teste*

72

Subida da Encosta Simples

- Usa heurística para mudar para estados que são *melhores* que o estado corrente
- Sempre muda para o melhor estado quando possível
- O processo termina quando todos os operadores tiverem sido aplicados e nenhum dos estados resultantes são melhores que o estado corrente

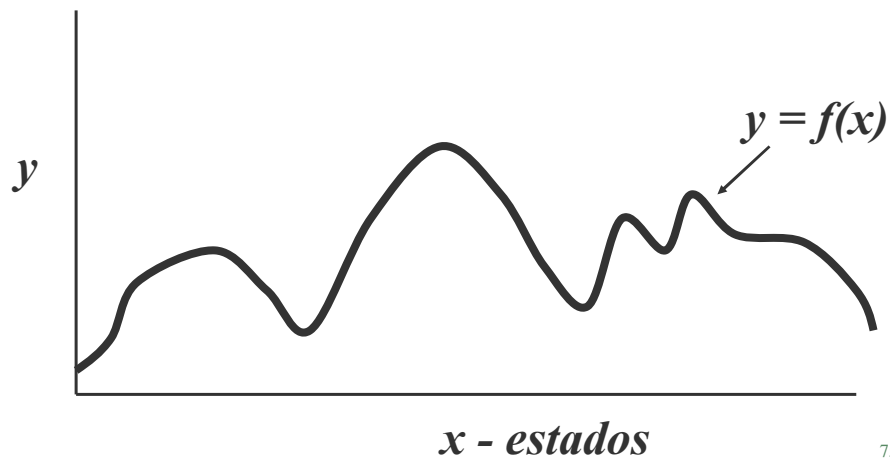
73

Subida da Encosta Simples - Algoritmo

- Se o estado inicial é a meta **então** termine
- **senão** *corrente* = estado inicial
- **enquanto** solução não encontrada **ou** não há novos operadores a serem aplicados
 - selecione um operador não aplicado sobre *corrente* para produzir um novo estado *novo*
 - **se** *novo* é meta **então** termine
 - **senão se** novo é melhor do que *corrente* **então** *corrente* = *novo*
- **fim_enquanto**

74

Subida da Encosta Função de Otimização



75

Potenciais Problemas com Subida da Encosta

- Terminar o processo em um ponto ótimo local
- A ordem de aplicação dos operadores pode fazer uma grande diferença
- Não pode ver estados anteriores ao estado corrente

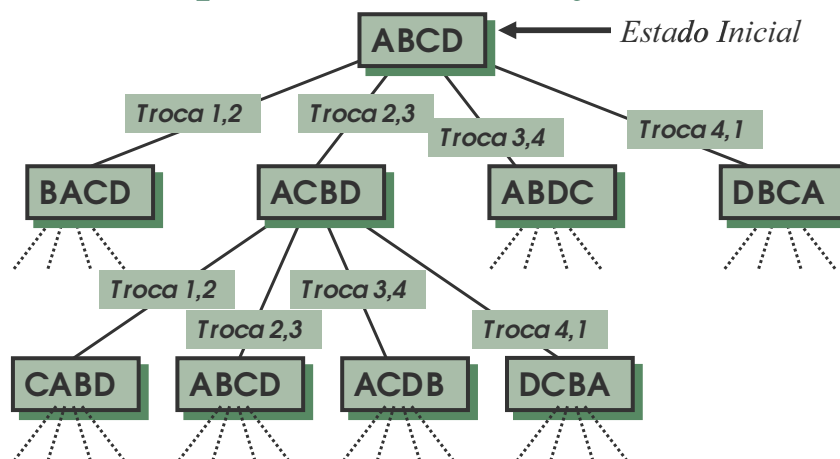
76

Exemplo de Subida da Encosta Simples

- Caixeiro Viajante - defina o espaço de estados como todos os trajetos possíveis
- Operadores trocam o estado de posição para cidades adjacentes em um trajeto
- Função Heurística é o tamanho do trajeto

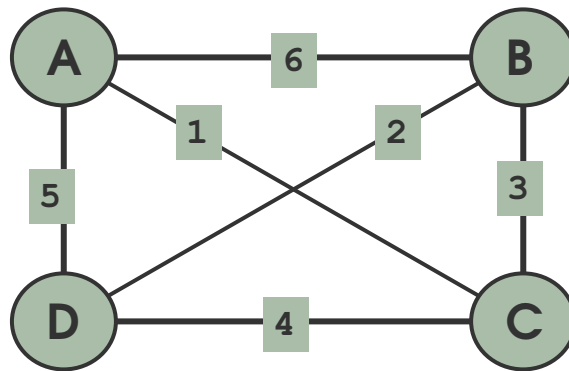
77

Espaço de Estados na Subida da Encosta para Caixeiro Viajante



78

Exemplo Caixeiro Viajante



79

Subida da Encosta pela Trilha mais Íngreme

- Uma variação da Subida da Encosta simples
- Ao invés de mover para o *primeiro* estado melhor, move-se para o melhor estado possível
- A ordem dos operadores não importa
- Como na subida simples, também não garante a subida para o melhor estado

80

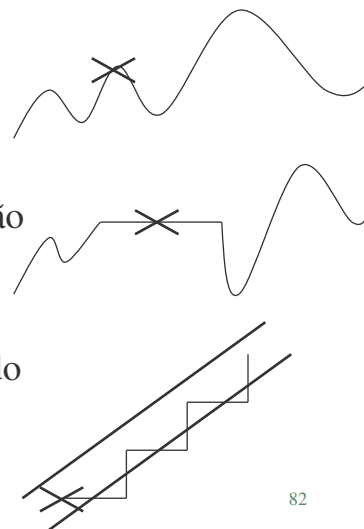
Subida da Encosta pela Trilha mais Íngreme - Algoritmo

- Se o estado inicial é a meta **então** termine
- **senão** *corrente* = estado inicial
- **enquanto** solução não encontrada **ou** uma iteração completa não modifique o estado *corrente*
 - *SUCC* = um estado tal que qualquer sucessor de *corrente* seja melhor que *SUCC*
 - **para** cada operador aplicável a *corrente* **faça**
 - aplique o operador e gere um novo estado *novo*
 - **se** *novo* é um estado meta **então** termine
 - **senão** se *novo* é melhor que *SUCC* então *SUCC* = *novo*
 - **fim_para**
 - Se *SUCC* é melhor que *corrente* então *corrente* = *SUCC*
- **fim_enquanto**

81

Perigos da Subida da Encosta

- Máximo Local: todos os estados vizinhos são piores que o atual
- Platô - todos os estados vizinhos são iguais que o atual
- Cume - máximo local que é causado pela incapacidade de aplicar 2 operadores de uma vez



82

Dependência Heurística

- Subida da encosta está baseada no valor associado aos estados por uma função heurística
- A heurística usada pelo algoritmo subida da encosta não precisa ser uma função estática de um simples estado
- A heurística pode indicar muitos estados ou pode usar outros meios para obter um valor para um estado

83

Cozimento Simulado

- Baseado no processo físico de cozimento de metal para obter o melhor estado (mínima energia)
- Subida da Encosta com um twist:
 - permite movimentos para baixo (para estados piores)
 - inicia permitindo grandes decidas (para estados piores) e gradualmente permite apenas pequenos movimentos de descida

84

Cozimento Simulado (Cont.)

- A busca inicialmente faz grandes saltos, explorando muitas regiões do espaço
- Os saltos são gradualmente reduzidos e a busca torna-se uma subida de encosta simples (busca por um ótimo local)

85

Cozimento Simulado (Cont.)

- O movimento dos estados é feito a partir dos estados com maior energia para os estados de menor energia
- Existe uma probabilidade de ocorrer um movimento para um estado de maior energia, calculada por:

$$p = e^{-\Delta E / kT}$$

- ΔE é a mudança positiva do nível de energia
- T é a temperatura
- k é a constante de Boltzmann

86

Cozimento Simulado – Algoritmo Simplificado

```
➤ candidato <- s0;  
  T <- T0;  
  repita  
➤ próximo <- vizinho de 'candidato' tomado aleatoriamente1;  
  deltaE <- energia(próximo) - energia(candidato);  
  se deltaE <= 0  
    então candidato <- próximo  
    senão faça candidato <- próximo com probabilidade  $\exp(-\text{deltaE}/T)$ 2;  
  T <- próximaTemperatura(T);  
➤ até T < Tfinal  
  retorna candidato;
```

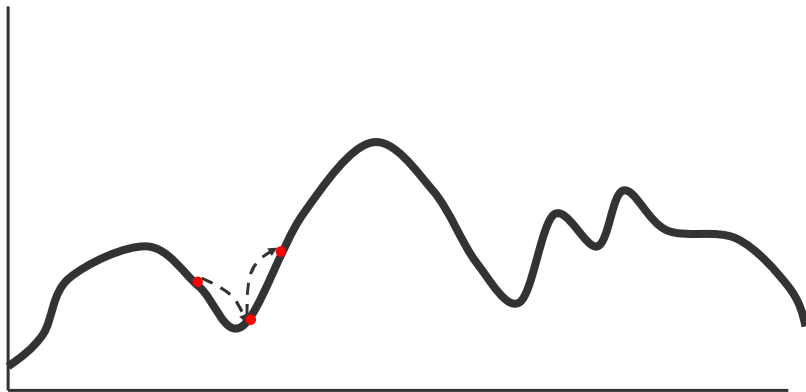
➤ onde:

- s0: estado (candidato a mínimo) inicial;
- T0/Tfinal: temperatura inicial/final;
- próximaTemperatura: função que calcula a temperatura vigente na próxima iteração;

Principais diferenças entre Cozimento Simulado e Subida da Encosta

- O cronograma da têmpora precisa ser mantido
- Movimentos para estados piores podem ser mantidos
- É interessante armazenar o melhor estado já encontrado até o momento (caso o estado final seja pior devido à escolha feita pelo algoritmo)

Cozimento Simulado (Cont.)



89

Cozimento Simulado - Algoritmo

- Se o estado inicial é a meta **então** termine
- **senão** *corrente* = estado inicial e *BEST_MOMENTO* = *corrente*
- Inicialize *T* de acordo com o anelamento
- **enquanto** solução não encontrada **ou** não houver mais operadores aplicáveis a *corrente*
 - selecione um operador não aplicado a *corrente* e gere o estado *novo*
 - $\Delta = (\text{valor do } \textit{corrente}) - (\text{valor do } \textit{novo})$
 - **se** o *novo* é meta **então** termine
 - **senão se** *novo* é melhor que *corrente*
 - **então** *corrente* = *BEST_MOMENTO* = *novo*
 - **senão** *corrente* = NOVO com probabilidade p'
 - Recalcule *T* de acordo com anelamento
- **fim_enquanto**
- **Retorne** *BEST_MOMENTO* como resposta

90

Busca Best-First

- Combina as vantagens das buscas em Largura e Profundidade
 - Profundidade: segue um caminho único, não precisa gerar todos os possíveis caminhos
 - Largura: não tem problemas com loops ou caminhos sem solução
- Busca Best First : explora o caminho mais promissor visto

91

Busca Best-First (Cont.)

- **enquanto** meta não alcançada:
 - Gere todos os potenciais estados sucessores e os adicione a uma lista de estados
 - Escolha o melhor estado na lista e realize busca sobre ele
- **fim_enquanto**
- Similar a encosta mais íngreme, mas não realiza busca em caminhos não escolhidos

92

Best First - Avaliação dos Estados

- Valor de cada estado é uma combinação de:
 - o custo do caminho para o estado
 - custo estimado de alcançar uma meta a partir um estado.
- A idéia é usar uma função para determinar (parcialmente) o ranking dos estados quando comparados a outros estados.
- Isto não ocorre em BL ou BF, mas é útil para busca Best-First.

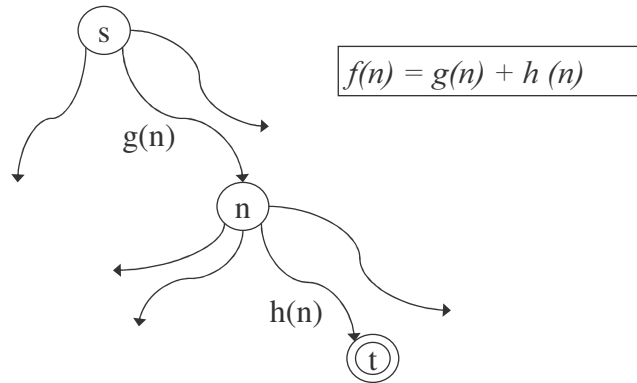
93

Busca Best First - Por que é necessária a avaliação

- Considere uma busca best-first que gera o mesmo estado muitas vezes.
- Quais dos caminhos que levam ao estado é o melhor?
- Lembre-se de que o caminho para uma meta pode ser a resposta (por exemplo, o problema dos jarros de água)

94

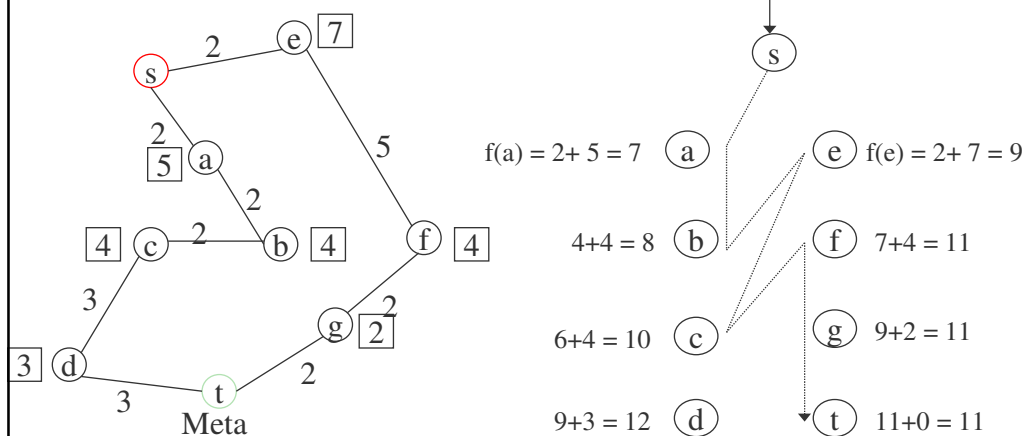
Busca Best-First



- $f(n)$ é uma função que estima o valor heurístico do nó n .
- s é o nó inicial, t é uma solução

95

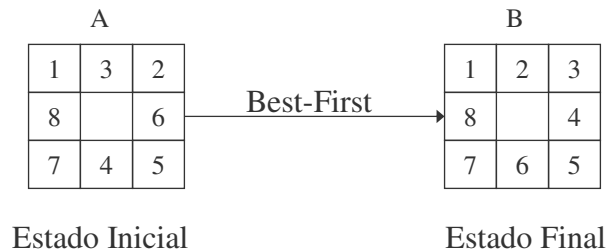
Busca Best-First - Caixeiro Viajante



- \square Representa a distância do nó até a solução (em linha reta)
- Os arcos representam as distâncias entre os nós

96

Best-First Aplicada ao Puzzle



Exercício:

Pense numa heurística que pode ser utilizada na busca Best-First para se encontrar uma solução rapidamente

97

Best-First Aplicada ao Puzzle - Solução do Exercício Anterior

Distancia_Total = somatório da diferença das distâncias do n. de cada quadrado e sua posição correta. Ex.: Na figura A do slide anterior

$$DT = |1 - 1| + |3 - 2| + |2 - 3| + |6 - 4| + |5 - 5| + |4 - 6| + |7 - 7| + |8 - 8| = 6$$

Sequência = somatório dos pesos associados a cada quadrado da seguinte forma:

se há valor no centro então some 1

se o quadrado não é central e é seguido pelo seu sucessor, some 0

se o quadrado não é central e não é seguido pelo seu sucessor, some 2

$$Seq = (0) + (0) + (2 + 2 + 2 + 2 + 2 + 2) = 12$$

$$H = DT + 3 * seq$$

98

Algoritmo A*

- O Algoritmo A* usa uma função de avaliação e a busca Best-First.
- A* minimiza o custo total do caminho.
- Sob as condições corretas A* fornece uma solução com custo menor em tempo ótimo!

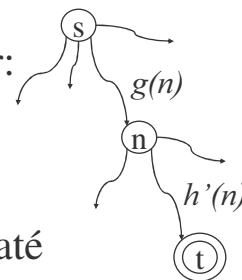
99

Função de Avaliação A*

- A função de avaliação f' é uma estimativa do valor de um nó dada por:

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

- $g(x)$ é o custo do estado inicial até x .
- $h'(x)$ é o custo estimado a partir de x até o estado meta (heurística).



100

Algoritmo A*

♣️ Idéia Geral:

- loops são desconsiderados - nenhum estado é expandido mais de uma vez.
- Informação sobre o caminho até o estado meta é retida.
- A lista de hipóteses é formada por caminhos de estados e não apenas por um estado

101

Algoritmo A*

1. Crie uma fila de caminhos parciais (inicialmente a raiz);
2. **enquanto** a fila não está vazia e meta não encontrada:
 - obtenha o estado x da fila;
 - se** x não é o estado meta **então**
 - Forme novos caminhos estendendo x a todos os caminhos possíveis;
 - Adicione os novos caminhos a fila;
 - Ordene a fila usando f*;
 - Remova duplicações* da fila (usando f*);
 - fim_se**
- fim_enquanto**

* duplicações são caminhos cujo final estão no mesmo estado.

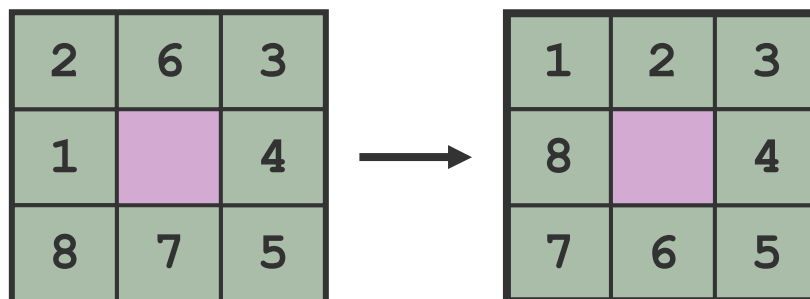
102

A* Otimização and Completude

- Se a função heurística h' é *admissível* o algoritmo encontrará o caminho mais barato para a solução no número mínimo de passos
- Uma heurística é *admissível* se jamais superestima o custo de um estado até o estado meta.

103

Exemplo A* - 8 Puzzle



104

Exemplo A*

🌳 A partir do mapa a seguir, crie a árvore de busca do A* gerada para a rota tendo como ponto inicial Arad e destino Bucharest (exemplo extraído de Norvig)

105

Exemplo A*

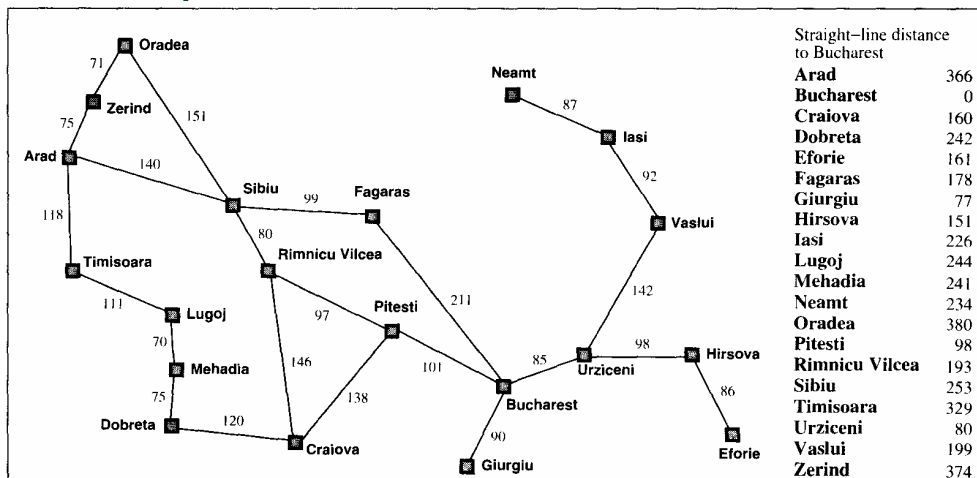


Figure 4.2 Map of Romania with road distances in km, and straight-line distances to Bucharest.

Redução de Problemas

- Problemas que podem ser decompostos requerem algoritmos especiais para tirar proveito de soluções parciais.
- Busca em árvores/grafos até aqui não empregavam redução de problema.
- Grafos AND/OR fornecem a base para muitos algoritmos de redução.
- Um grafo AND/OR é diferente de um espaço de estados

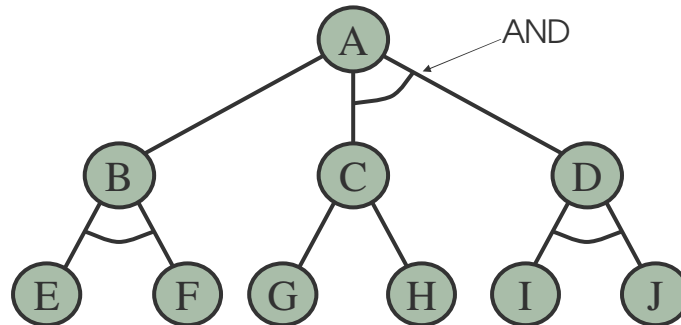
107

Árvores/Grafos AND/OR

- Cada nó (estado) na árvore de busca tem filhos que representam possíveis estados sucessores.
- Cada nó é AND ou OR.
- Um nó é resolvido se:
 - . AND: nó e todos os seus filhos são resolvidos.
 - . OR: nó e pelo menos 1 filho é resolvido

108

Árvore AND/OR



Para resolver A, é necessário resolver B *ou* C e D.
Para resolver B, é necessário resolver E e F.

...

109

Busca em Grafos AND/OR

🐼 Algoritmo básico de busca AND/OR

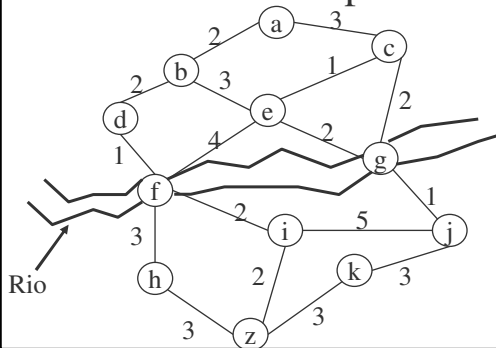
- **se** Nó é uma solução **então** pare
- **se** Nó tem sucessores OR *sucessoresOR*
 - **então** resolva *sucessoresOR* (tente encontrar pelo menos uma solução a partir de *sucessores*)
- **se** Nó tem sucessores AND *sucessoresAND*
 - **então** resolva *sucessoresAND* (tente encontrar pelo menos uma solução para cada nó de *sucessoresAND*)

* Se o algoritmo anterior não encontrou nenhuma resposta, então o problema não possui solução

110

Representação de Grafos AND/OR

- Decompõe problemas em subproblemas
- Subproblemas são independentes e podem ser resolvidos separadamente

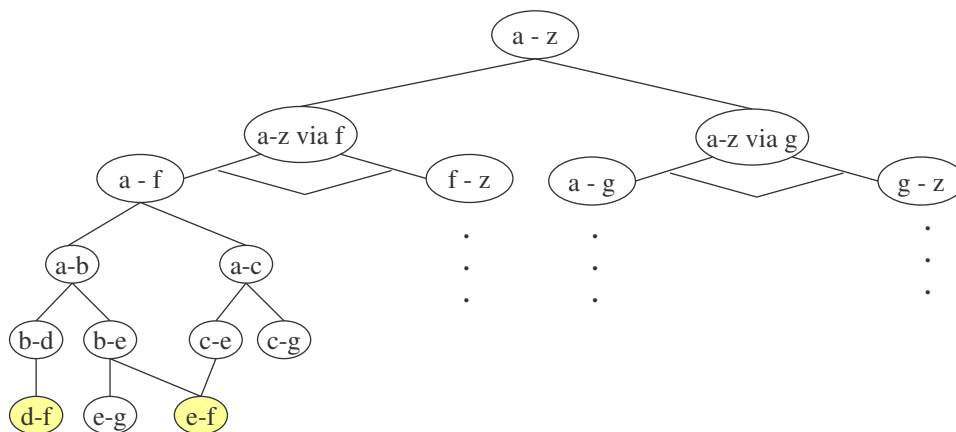


Exercício:

Encontre o grafo and/or com as rotas possíveis entre as cidades a e z que obrigatoriamente passem por f e g.

111

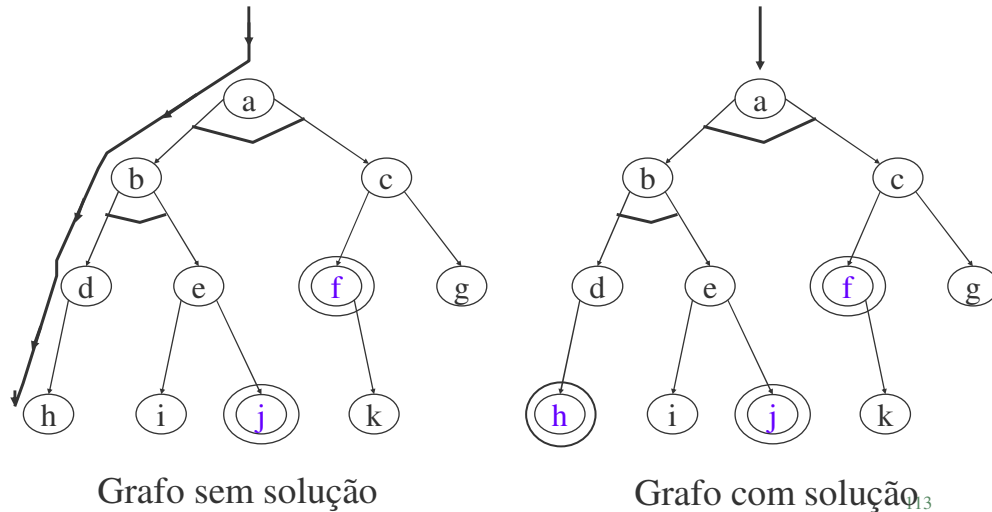
Representação de Grafos AND/OR - Solução do Exercício Anterior



112

Representação de Grafos AND/OR

(Cont.)



Busca AND/OR

Exercício: Escreva um programa Prolog que realize busca AND/OR simples em um grafo AND/OR qualquer

Solução:

```
and_or(No,N) :-
    pertence(No,N).
and_or(No,N) :-
    filhos_or(No,Filhos_or),
    solucao_or(Filhos_or,N).
and_or(No,N) :-
    filhos_and(No,Filhos_and),
    solucao_and(Filhos_and,N).
```

```
solucao_or(Nos,N) :-
    pertence(X,Nos),
    and_or(X,N),!.

solucao_and([],_).
solucao_and([No|Nos],N) :-
    and_or(No,N),
    solucao_and(Nos,N).
```

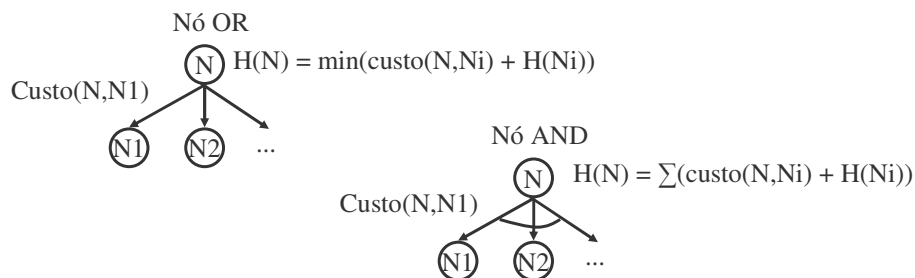
Algoritmo AO* (Encontra o caminho com menor custo)

- Aplicado sobre grafos AND/OR
- Cada nó do Grafo possui uma função h' (estimativa do custo do caminho entre o nó e um conjunto de soluções)
- Não é necessário armazenar g' (custo do nó inicial até o nó corrente)
- Apenas os nós do melhor caminho serão expandidos
- Cada nó do grafo aponta para seus sucessores e predecessores diretos

115

Algoritmo AO*

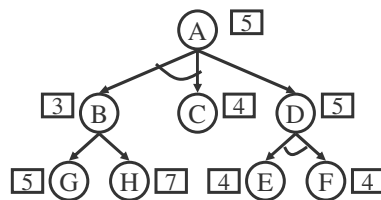
- A partir de um nó expande o filho mais promissor
- A qualidade de um nó é dada por uma função $h(n)$
- Exemplo considera apenas arcos com peso 1



116

Exemplo do AO*

- Dado o problema representado no grafo and/or a seguir, ilustre o processo de pesquisa pelo melhor caminho no grafo a partir do nó A

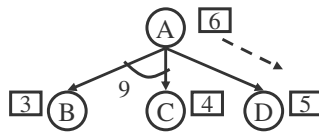


117

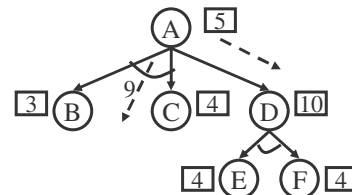
Exemplo do AO*



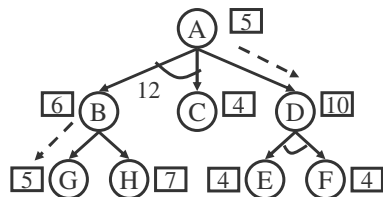
(a)



(b) A é expandido



(b) D é expandido



(b) B é expandido

118

Algoritmo AO* (cont.)

1. *Grafo* consiste apenas do nó inicial *Inicial*
2. **enquanto** *Inicial* não é resolvido **ou** $h' > limite$
 - a) selecione um dos sucessores de *Inicial*, *Nó*, que ainda não foi expandido
 - b) gere os sucessores de *Nó*
 - **se** *Nó* não tiver sucessor, atribua a *limite* o valor de $h'(Nó)$
 - **se** *Nó* tiver sucessores para cada nó *sucessor* que não seja antecessor
 - acrescente *sucessor* a *Grafo*
 - **se** *sucessor* é solução **então** marque-o como resolvido e $h'(sucessor) = 0$
 - **se** *sucessor* não é solução **então** calcule $h'(sucessor)$

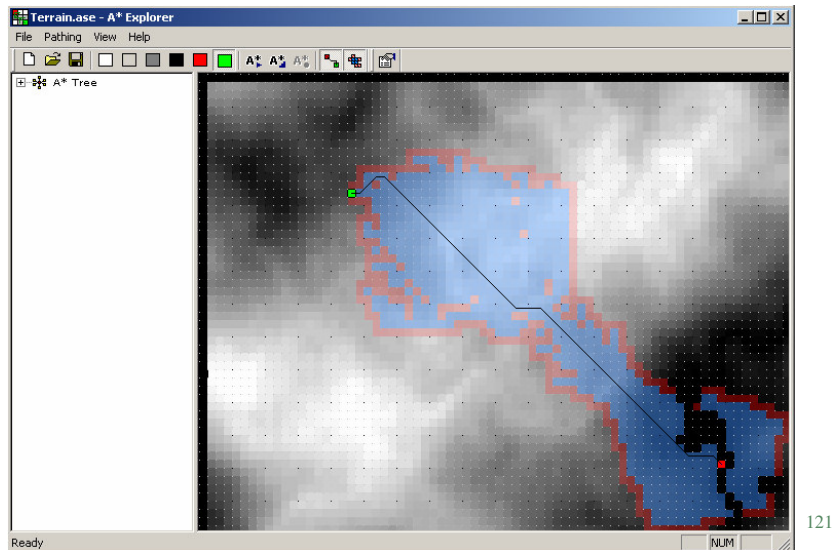
119

Algoritmo AO* (cont.)

- c) S = conjunto de Nós marcados como Resolvidos ou cujo valor de h' tenha sido alterado
 - **enquanto** S não está vazio
 - escolha, se possível, um nó que não tenha nenhum descendente em *GRAFO* ocorrendo em S , caso não exista, escolha qualquer nó (*corrente*) em S e o remova
 - $h'(corrente)$ = mínimo do somatório dos custos dos arcos que dele emergem
 - marque o melhor caminho que parte de *corrente* como aquele com o menor custo calculado anteriormente
 - marque *corrente* como resolvido se todos os nós ligados a ele pelo arco obtido tiverem sido marcados como resolvidos
 - **se** *corrente* está resolvido **ou** o custo de *corrente* foi alterado **então** acrescente os *ancestrais* de *corrente* a S

120

Ver demo do ASE



121

Satisfação de Restrição

- Alguns problemas têm metas que podem ser modeladas como um conjunto de restrições satisfeitas.
- Estados são definidos por valores de um conjunto de variáveis.
- Utilização de restrições para limitar movimentos no espaço de busca pode melhorar a eficiência.

122

Satisfação de Restrição - Algoritmo

- 1. Formule todas as restrições R a partir do estado inicial
- 2. Encontre um novo estado S
- 3. Aplique as restrições R sobre S gerando um novo conjunto de restrições R'
- 3. Se S satisfaz todas as restrições R' , retorne S
- 4. Se S produz uma contradição então retorne fracasso
- 5. Se não existir outro estado retorne fracasso senão retorne ao passo 2

123

Exemplo de Problemas de Satisfação de Restrições

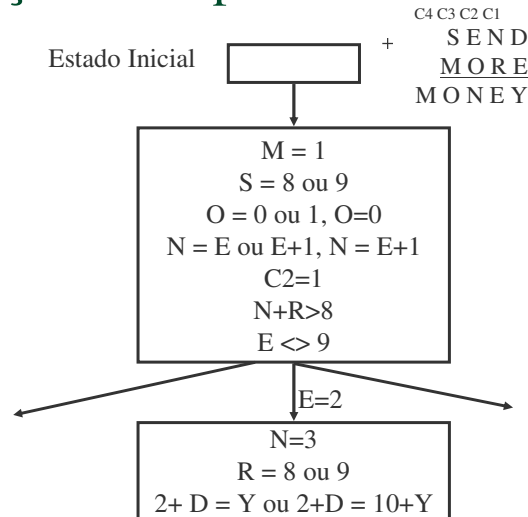
➤ Criptoaritmética:

- problemas aritméticos representados em letras. A meta é associar um dígito diferente a cada letra de modo a poder utilizá-las matematicamente.

$$\begin{array}{r} + \quad \text{S E N D} \\ \quad \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

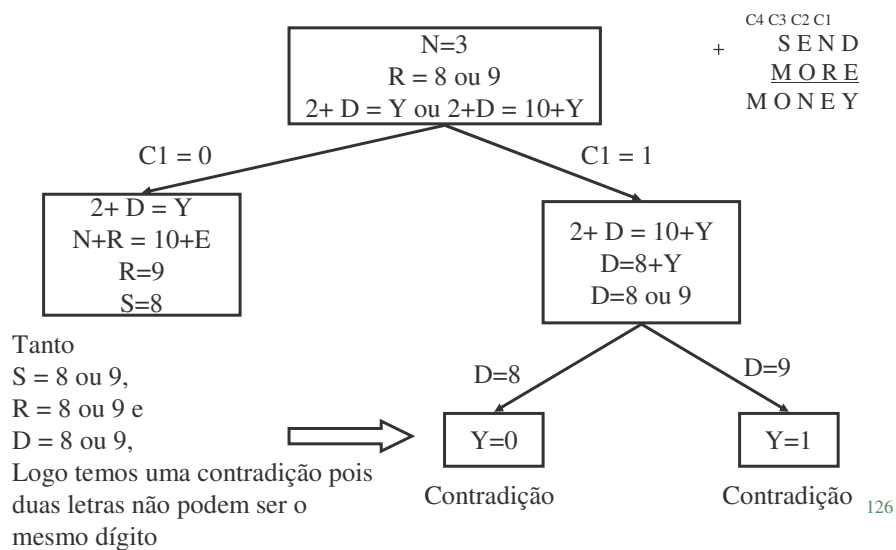
124

Exemplo de Problemas de Satisfação de Restrições - Criptoaritmética



125

Exemplo de Problemas de Satisfação de Restrições - Criptoaritmética

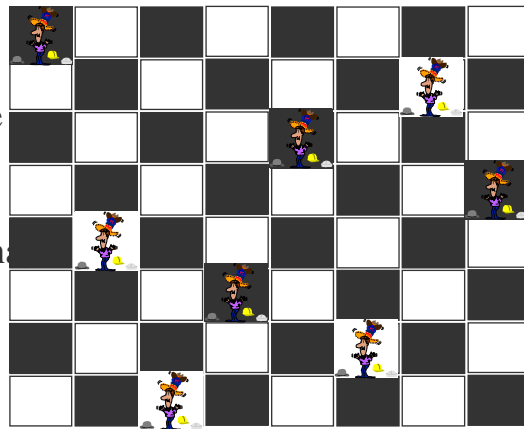


126

Exemplo de Problemas de Satisfação de Restrições (Cont.)

♟ 8 Rainhas:

- colocar 8 rainhas no tabuleiro de xadrez de modo que nenhuma rainha ataque outra rainha (na mesma linha, coluna ou diagonal).



127

Análise Meios-Fins

- ♟ Busca para frente (em direção à meta) e para trás (da meta para o estado corrente) ao mesmo tempo.
- ♟ Isola uma diferença entre o estado corrente e o estado meta - encontra um operador que reduz a diferença.
- ♟ É importante resolver primeiramente as diferenças maiores
- ♟ Motivada pela forma com que alguns problemas são resolvidos por pessoas.

128

Análise Meios-Fins – Algoritmo GPS

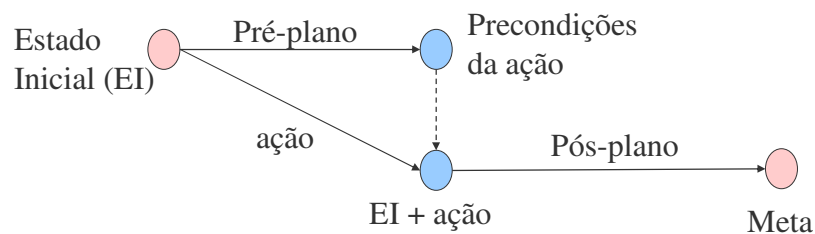
GPS(estado-inicial, meta)

1. Se $meta \subseteq estado-inicial$, então retorne *true*
2. Selecione uma diferença d entre *meta* e o *estado-inicial*
3. Selecione um operador O que reduz a diferença d
4. Se nenhum outro operador então *falhe*
5. $Estado = \mathbf{GPS}(estado-inicial, precondições(O))$
6. Se Estado, então retorne **GPS**(**apply**($O, estado-inicial$), *meta*)

129

Análise Meios-Fins - Esquema

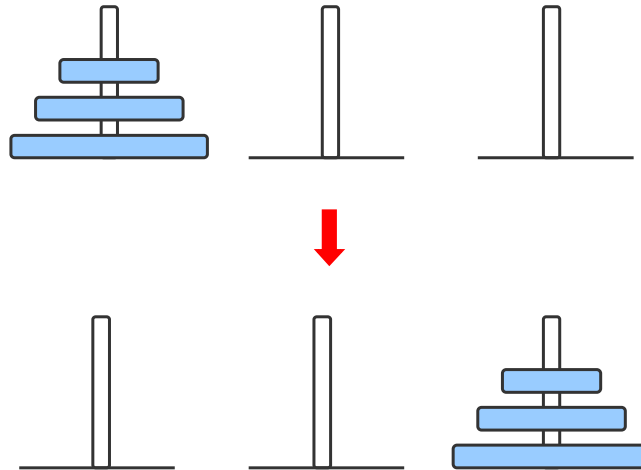
- Encontre uma ação útil (que reduza a diferença com a meta)



- Crie dois novos subproblemas:
 - tornar verdadeiras as precondições da ação a partir de EI
 - encontrar a meta a partir do resultado da ação sobre EI

130

Exemplo de análise meios-fins: as Torres de Hanoi



131

Solução em Prolog

```

hanoi(1,Start,End,_) :-
    write('move disc from '),
    write(Start), write(' to '),
    write(End), nl.

hanoi(N,Start,End,Aux) :-
    N>1, M is N-1,
    hanoi(M,Start,Aux,End),
    write('move disc from '), write(Start),
    write(' to '), write(End),nl,
    hanoi(M,Aux,End,Start).

```

Se existe apenas um disco,
Então mova-o para o pino final

Se existe mais de um disco,
Então

mova todos os N-1 pinos de cima da
esquerda para o meio com o auxílio do
pino direito

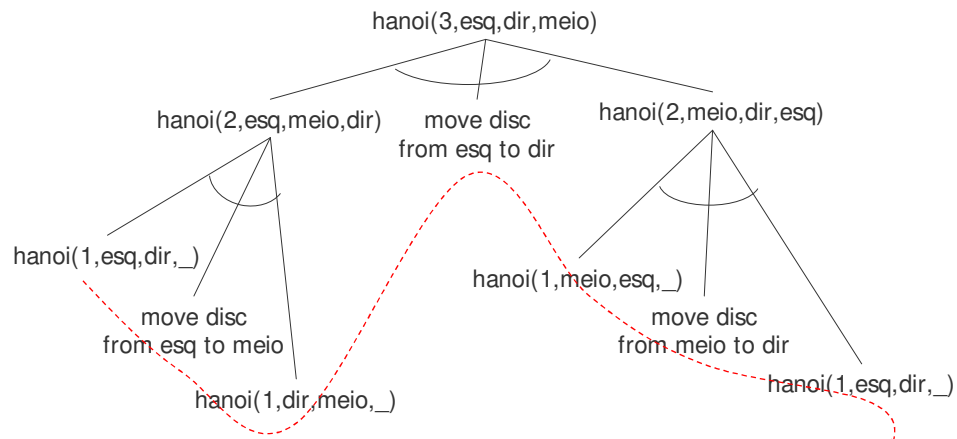
mova o pino de cima para o pino direito

mova todos os N-1 pinos do meio para a
direita com o auxílio do pino esquerdo

Questionamento: ?- hanoi(3,esquerda,direita,meio).

132

Execução do Programa Torres de Hanoi



133

Representação de Conhecimento

Capítulo 4

134

Representação de Conhecimento

- Técnicas genéricas de busca foram apresentadas.
- Geralmente inicia-se com uma técnica genérica que é aprimorada para ser aplicada em um domínio específico.
- A representação de conhecimento sobre o domínio é o maior problema.
- Escolher uma boa representação faz grande diferença.

135

Considerações sobre Representação de Conhecimento

- O conhecimento do mundo Real pode ser representado de várias formas
- Essas formas diferem no uso, expressividade e outras características
- Algumas formas de representação são as seguintes:
 - Linguagens Lógicas de Programação
 - Provadores de Teoremas
 - Sistemas baseados em regra ou de produção
 - Redes Semânticas
 - Linguagens de representação de frames
 - Bases de dados (relacionais, orientadas a objetos, etc.)
 - Sistemas de raciocínio sobre restrições
 - Lógicas de descrição (Modal, Fuzzy, Temporal, etc.)
 - Redes bayesianas
 - Raciocínio Evidencial

136

Conhecimento e Mapeamentos

- Conhecimento é uma coleção de fatos sobre o domínio.
- É necessário uma representação de fatos que possa ser manipulada por um programa.
 - Representação simbólica é necessária.
 - Precisa ser capaz de mapear fatos em símbolos.
 - Precisa ser capaz de mapear símbolos para fatos?

137

Representação de Fatos

- Representação lógica é comum em programas de IA:
 - Malhado é um cachorro
 - $\text{cachorro}(\text{Malhado})$
 - Todos os cachorros têm rabo
 - $\forall x: \text{cachorro}(x) \rightarrow \text{tem_rabo}(x)$
 - Malhado tem rabo
 - $\text{tem_rabo}(\text{Malhado})$

138

Representação de Propriedades

- Adequabilidade Representacional
- Adequabilidade Inferencial
- Eficiência na Inferência
- Eficiência na Aquisição

139

Bases de Dados Relacionais

- Uma forma de armazenar fatos declarativos é utilizar bases de dados relacionais:

Jogador	Altura	Peso	Posição
Érick	1,80	90	Direita
Marcos	1,75	85	Esquerda
Carlos	1,85	110	Direita

- Coleção de Atributos e Valores.

140

Herança

- É geralmente utilizada para fornecer uma estrutura de representação que suporta diretamente mecanismos de inferência.
- *Herança de Propriedades* é um mecanismo de herança comum.
- Objetos pertencem a classes.
- Classes possuem propriedades que são herdadas por objetos que pertencem à classe.

141

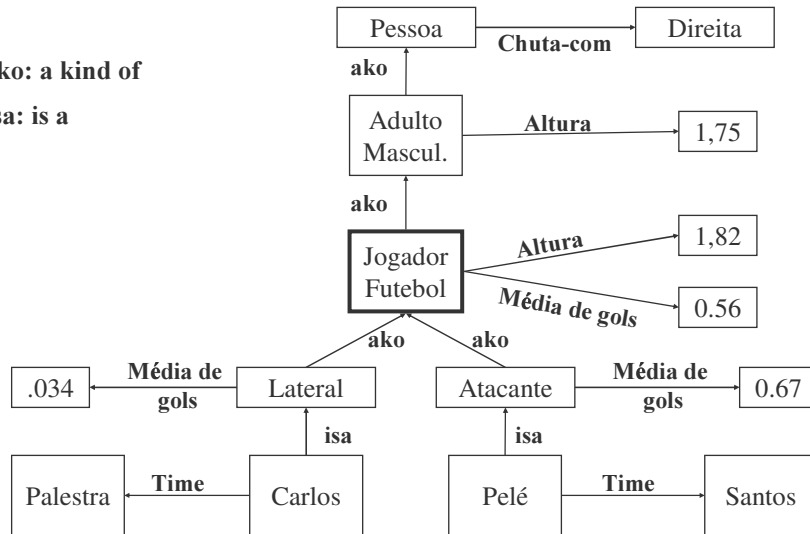
Hierarquia de Classes

- Classes são organizadas em uma hierarquia, dessa forma algumas classes são membros de classes mais gerais.
- Há grande variedade de estratégias de representação usadas em IA que são baseadas em herança:
 - regras de produção
 - redes semânticas
 - sistema de frames

142

Herança de Conhecimento

ako: a kind of
isa: is a



143

Algoritmo da Herança

- Nós queremos encontrar o valor do atributo **a** de um objeto **o**.
- Primeiro verificamos o próprio objeto **o**.
- A seguir nós verificamos um atributo *instância* em busca do valor de **a**.
- Se não há nenhum valor para **a**, verifique todos os atributos *is_a*.

144

Conhecimento Inferencial

- Herança não é o único mecanismo inferencial - fórmulas lógicas são sempre usadas:

$\forall x, y : atacante(x) \wedge chute(x, y) \wedge \text{não} - \text{defesa}(y) \rightarrow gbl(x)$

- Inferência baseada em Lógica será estudada posteriormente.

145

Conhecimento Procedimental

- O conhecimento está contido no código escrito em alguma linguagem.
- Como conhecimento procedimental trabalha com representação de propriedades:
 - Adequabilidade Representacional
 - Adequabilidade Inferencial
 - Eficiência de Inferência
 - Eficiência de Aquisição

146

Importância de Atributos

- Um atributo *ako* e *isa* suporta herança de propriedades.
- *ako* e *isa* podem ter outros nomes, ou podem estar implicitamente representados.
- O atributo *ako* (membro da classe) é transitivo.

147

Atributos como objetos

- Atributos podem ser objetos que possuem propriedades:
 - Inverso
 - Existência na hierarquia
 - Técnicas de raciocínio sobre valores
 - Atributos com um único valor

148

Granularidade da Representação

- Fatos de alto nível podem requerer grande armazenamento se representados como uma coleção de primitivas de baixo nível.
- Maior parte do conhecimento que está disponível em uma forma de alto nível (Português, Inglês).
- Nem sempre está claro quais primitivas de baixo nível utilizar.

149

Representação de Conjuntos de Objetos

- Definição Extensional: listar todos os membros de um conjunto.
 - Atletas = {Pedro, Carlos, Ana, Janete}
- Intensional: usa regras para definir membros de um conjunto:
 - Atleta = {x: boa_saude(x) e pratica_esporte(x) }

150

Busca e Representação de Estado

- Cada estado deve ser representado como uma coleção de fatos.
- O armazenamento de muitos estados na memória pode ser impraticável.
- A maioria dos fatos não é mudada quando a busca é movida de um estado para outro.

151

O Problema do Modelo

- Determinar como melhor representar fatos que mudam de estado para estado e aqueles que não mudam constitui o *Problema do Modelo*.
- Algumas vezes o mais difícil é determinar quais fatos mudam e quais não mudam.

152

Lógica em Representação de Conhecimento



153

Lógica

- Usa dedução matemática para derivar novo conhecimento.
- Lógica de Predicados é um poderoso esquema de representação para programas de IA.
- Lógica Proposicional é muito simples (menos poderosa).

154

Introdução à Lógica

🧠 O que é Lógica

“Linguagem que permite a representação de fatos, idéias ou conhecimento e, o mais importante, fornece um conjunto de métodos para a validação e inferência sobre essas informações.”

155

Introdução à Lógica (cont.)

🧠 Representando sentenças (conhecimento) no cálculo de predicados (Lógica de Predicados)

“Todos os membros da associação vivem na cidade. Quem é presidente da sociedade é membro da Associação. Sra Farias é presidente da Associação. Logo Sra. Farias vive na cidade.”

$\forall x(\text{membro}(x) \rightarrow \text{mora}(x))$ Premissa 1

$\forall x(\text{presidente}(x) \rightarrow \text{membro}(x))$ Premissa 2

$\text{presidente}(\text{sra_farias})$ Premissa 3

$\text{mora}(\text{sra_farias})$ Conclusão

Com a Lógica podemos representar e validar sentenças

156

Verificação de argumentos ou prova de teoremas

- ☛ Dadas as fórmulas $\beta_1, \beta_2, \dots, \beta_n$ e uma fórmula α , diz-se que essas informações formam um teorema ou o argumento é válido se α é **consequência lógica** de $\beta_1, \beta_2, \dots, \beta_n$, ou seja

$\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ é uma **tautologia**.

- ☛ Métodos de prova de teoremas

- Semântico
- Sintático
- Dedutivo
- Tableau
- Resolução



157

Teorema da Dedução ou Admissão de Premissas

Teorema 1. Dadas as fórmulas $\beta_1, \beta_2, \dots, \beta_n$ e uma fórmula α , α é **consequência lógica** de $\beta_1, \beta_2, \dots, \beta_n$ se e somente se a fórmula $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ é uma **tautologia**.

Prova: Seja I uma interpretação qualquer,

- 1) Se $\beta_1, \beta_2, \dots, \beta_n$ forem verdade em I, então α também será verdade em I, pois é consequência lógica dos β_i 's.
- 2) Se um dos β_i 's for falso em I, $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n$ também será falso em I. Independente do valor de α , $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ é verdade em I.

De 1 e 2 tem-se que $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ é verdade em qualquer interpretação, ou seja, $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ é uma tautologia.

158

Teorema da Redução ao Absurdo ou Prova por refutação

Teorema 2. Dadas as fórmulas $\beta_1, \beta_2, \dots, \beta_n$ e uma fórmula α , α é **consequência lógica** de $\beta_1, \beta_2, \dots, \beta_n$ se e somente se a fórmula $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \wedge \neg \alpha$ é uma **contradição**.

Prova: Sabe-se pelo teorema anterior que:

Dadas as fórmulas $\beta_1, \beta_2, \dots, \beta_n$ e α , α é consequência lógica se e somente se $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ for **válida**. Logo, sabe-se que α é consequência lógica se e somente se a negação de $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha$ for uma contradição. Assim

$$\begin{aligned}\neg(\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \rightarrow \alpha) &\equiv \\ \neg(\neg(\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n) \vee \alpha) &\equiv \\ \beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \wedge \neg \alpha\end{aligned}$$

ou seja, $\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n \wedge \neg \alpha$ é uma **contradição**

159

Prova por Resolução

- Método baseado em **Redução ao Absurdo**
- Aplicado sobre um conjunto de **Cláusulas Horn**
- Utiliza apenas uma regra de inferência: **Regra da Resolução**

160

Prova por Resolução

- Método baseado em **Redução ao Absurdo**
- Aplicado sobre um conjunto de **Cláusulas Horn**
- Utiliza apenas uma regra de inferência: **Regra da Resolução**

161

Obtenção de Cláusulas Horn

- Uma cláusula Horn é um caso particular da **Notação de Kowalski**
- Para a obtenção da Notação de Kowalski de uma fórmula devemos obter inicialmente a **Notação Clausal** seguindo alguns passos

162

Passos para a obtenção da Notação Clausal

Ex.: Dada a Fórmula:

$$\forall x \forall y (\exists z (p(x,z) \wedge p(y,z)) \rightarrow \exists u q(x,y,u))$$

Vamos obter a notação Clausal realizando os seguintes passos:

- 1 – Ligar existencialmente as variáveis livres
- 2 – Eliminar quantificadores redundantes

163

Obtenção da Notação Clausal

$$\forall x \forall y (\exists z (p(x,z) \wedge p(y,z)) \rightarrow \exists u q(x,y,u))$$

- 3 – Renomear variáveis quantificadas mais do que uma vez

- 4 – Remover equivalências (\leftrightarrow) e implicações (\rightarrow)

$$\forall x \forall y (\neg \exists z (p(x,z) \wedge p(y,z)) \vee \exists u q(x,y,u))$$

- 5 – Mover a negação para o interior da fórmula

$$\forall x \forall y (\forall z (\neg p(x,z) \vee \neg p(y,z)) \vee \exists u q(x,y,u))$$

- 6 – Eliminar os quantificadores existenciais

$$\forall x \forall y (\forall z (\neg p(x,z) \vee \neg p(y,z)) \vee q(x,y,g(x,y)))$$

- 7 – Obter a **Fórmula Normal Prenex** (FNP) e remover os quantificadores universais

$$\neg p(x,z) \vee \neg p(y,z) \vee q(x,y,g(x,y))$$

164

Obtenção da Notação Clausal (cont.)

$$\neg p(x,z) \vee \neg p(y,z) \vee q(x,y,g(x,y))$$

8 – Colocar a matriz da **FNP** na **Forma Conjuntiva**

9 – Eliminar os símbolos “ \wedge ” substituindo-se expressões da forma $(X1 \wedge X2)$ pelo conjunto de wffs $\{X1, X2\}$:

$$(\neg p(x,z) \vee \neg p(y,z)) \vee q(x,y,g(x,y))$$

10 – Notação clausal:

$$C1: \neg p(x,z) \vee \neg p(y,z) \vee q(x,y,g(x,y))$$

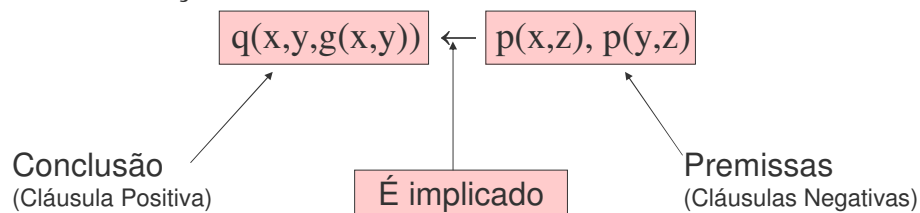
165

Notação Clausal e Notação de Kowalski

Notação Clausal:

$$C1: \neg p(x,z) \vee \neg p(y,z) \vee q(x,y,g(x,y))$$

11 – Notação de Kowalski



166

Notação de Kowalski

☞ Uma cláusula genérica na notação de Kowalsky é representada por:

$$A_1, A_2, \dots, A_m \leftarrow B_1, B_2, \dots, B_n$$

Quando

$m > 1$: as conclusões são indefinidas, ou seja, há várias conclusões;

$m \leq 1$: são as chamadas **Cláusulas de Horn**, que têm como casos particulares:

- $m = 1$ e $n > 0$: “ $A \leftarrow B_1, \dots, B_n$ ” (chamada cláusula definida, isto é, há apenas uma conclusão);
- $m = 1$ e $n = 0$: “ $A \leftarrow$ ” (chamada cláusula definida incondicional ou fato);
- $m = 0$ e $n > 0$: “ $\leftarrow B_1, \dots, B_n$ ” (negação pura de B_1, \dots, B_n) (não há conclusão);
- $m = 0$ e $n = 0$: “ \leftarrow ” chamada cláusula vazia, denotada \square .

167

Resolução

☞ Definição: Método de prova de teoremas que utiliza uma única regra de inferência (Regra da Resolução):

De $A \vee B$ e $\neg B \vee C$

Deduz-se $A \vee C$

De $A \vee \neg A$

Deduz-se \square (falso ou cláusula vazia)

168

Resolução

- Prova por Redução ao Absurdo através da negação da Conclusão
- Prova por Redução ao Absurdo através da negação do teorema
- Dado um conjunto de cláusulas $\beta_1, \beta_2, \dots, \beta_n$ e α onde cada β_i e α estão na **FNC**, aplique a regra da resolução até que a cláusula vazia seja obtida.

169

Exemplo de Resolução - Solução 1

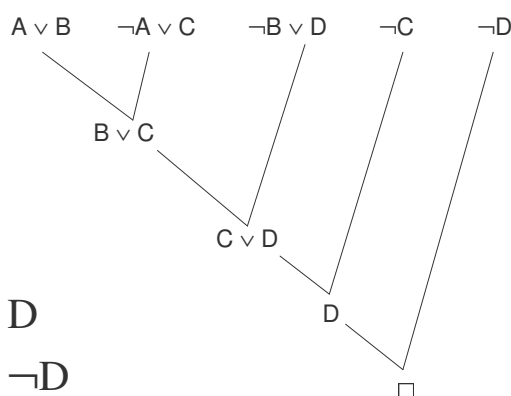
P1: $A \vee B$

P2: $\neg A \vee C$

P3: $\neg B \vee D$

Conclusão: $C \vee D$

\neg Conclusão: $\neg C$ e $\neg D$



170

Exemplo de Resolução - Solução 2

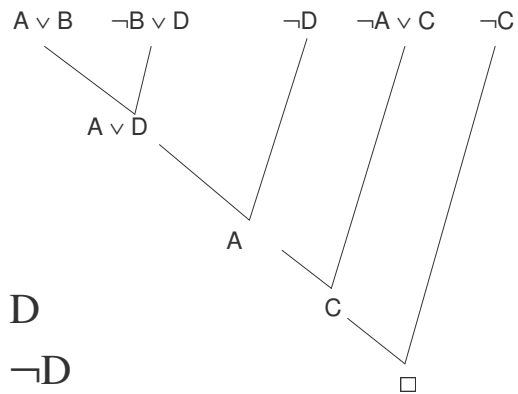
P1: $A \vee B$

P2: $\neg A \vee C$

P3: $\neg B \vee D$

Conclusão: $C \vee D$

\neg Conclusão: $\neg C$ e $\neg D$



171

Resolução-SLD*

☞ Trabalha com Cláusulas Horn:

1. " $A \leftarrow B_1, \dots, B_n$ "
2. " $A \leftarrow$ "
3. " $\leftarrow B_1, \dots, B_n$ " (negação pura de B_1, \dots, B_n)
4. " \leftarrow " (cláusula vazia, denotada \square)

onde 1 e 2 são cláusulas definidas e 3 e 4 são cláusulas objetivo

* Resolução linear com função de seleção para cláusulas definidas

172

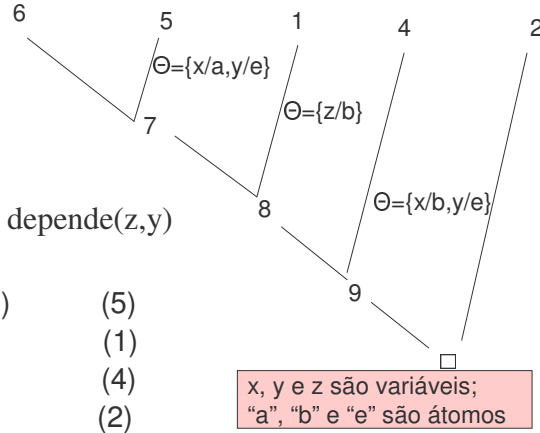
Resolução-SLD*

Conclusão negada: $\leftarrow \text{depende}(a,e)$

Exemplo:

Premissas

1. $\text{chama}(a,b) \leftarrow$
2. $\text{usa}(b,e) \leftarrow$
3. $\text{depende}(x,y) \leftarrow \text{chama}(x,y)$
4. $\text{depende}(x,y) \leftarrow \text{usa}(x,y)$
5. $\text{depende}(x,y) \leftarrow \text{chama}(x,z), \text{depende}(z,y)$
6. $\leftarrow \text{depende}(a,e)$
7. $\leftarrow \text{chama}(a,z), \text{depende}(z,e)$ (5)
8. $\leftarrow \text{depende}(b,e)$ (1)
9. $\leftarrow \text{usa}(b,e)$ (4)
10. \square (2)



x, y e z são variáveis;
"a", "b" e "e" são átomos

173

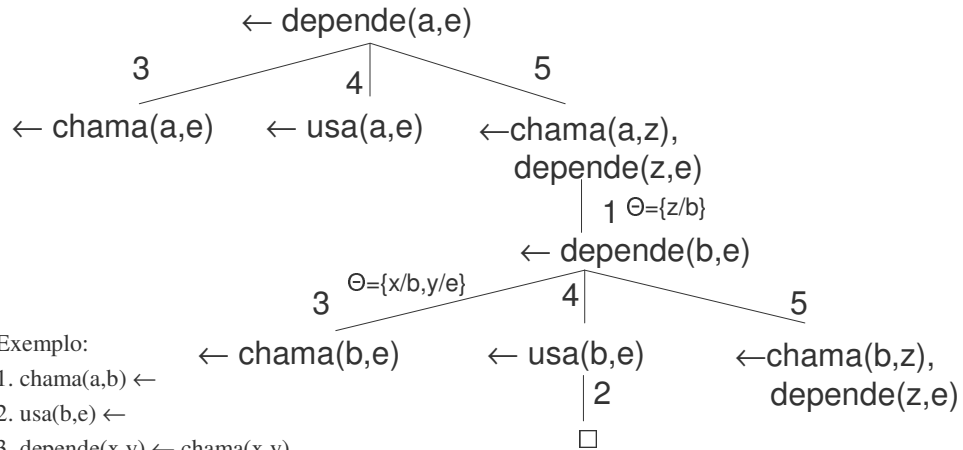
* Resolução linear com função de seleção para cláusulas definidas

Resolução-SLD (cont.)

- 👤 A linearização não é suficiente
- 👤 Para se formalizar um procedimento de Resolução-SLD é necessário utilizar uma função de escolha na seleção das cláusulas definidas
- 👤 Construimos então uma árvore de refutação da seguinte maneira:
 - Para cada nó com rótulo A construímos um conjunto de nós para os filhos na ordem em que eles aparecem usando a regra da resolução
 - Para cada nó criado, repete-se o procedimento até que a solução (cláusula vazia) seja encontrada

174

Resolução-SLD (cont.)



Exemplo:

1. $\text{chama}(a,b) \leftarrow$
2. $\text{usa}(b,e) \leftarrow$
3. $\text{depende}(x,y) \leftarrow \text{chama}(x,y)$
4. $\text{depende}(x,y) \leftarrow \text{usa}(x,y)$
5. $\text{depende}(x,y) \leftarrow \text{chama}(x,z), \text{depende}(z,y)$
6. $\leftarrow \text{depende}(a,e)$

x, y e z são variáveis;
"a", "b" e "e" são átomos

Semânticas de um Programa Lógico

Semântica Declarativa de um Programa Lógico (PL)

- Um programa lógico P é um conjunto de cláusulas definidas na forma:
$$“A \leftarrow B_1, \dots, B_n” \quad \text{ou} \quad “A \leftarrow”$$
- Uma consulta Q é uma conjunção de literais na forma
$$\leftarrow B_1, \dots, B_n$$
- Uma solução para Q é um conjunto de substituições das variáveis de Q por termos de P

177

Semântica Procedimental de um Programa Lógico (PL)

Dado um programa lógico P e uma consulta Q cuja cláusula objetivo é

$$\leftarrow B_1, \dots, B_n$$

então o procedimento de resolução-LSD com uma função que seleciona as cláusulas mais à esquerda pode ser utilizado para se obter as soluções sob a forma de um conjunto de substituições de variáveis

178

Exercício

☛ Dado o programa lógico à seguir, construa a árvore de resolução-SLD para a seguinte cláusula objetivo:

← tio(lucio, flavia)

```
casado(jose,carmem) ←  
pai(jose,fabio) ←  
pai(fabio,flavia) ←  
pai(jose,lucio) ←  
mae(X,Y) ← casado(Z,X), pai(Z,Y)  
irmao(X,Y) ← pai(Z,X), pai(Z,Y)  
tio(X,Y) ← pai(Z,Y), irmao(Z,X)  
tio(X,Y) ← mae(Z,Y), irmao(Z,X)
```

179

Exemplo

☛ Nevará OU será um teste.

☛ Dave é Darth Vader OU não nevará.

☛ Dave não é Darth Vader.

☛ Será um teste?

180

Solução

Nevará = a Teste = b Dave is D. Vader = c

Base de Conhecimento (Tudo é verdadeiro):

$$a \vee b, \quad c \vee \neg a, \quad \neg c$$

Por resolução nós sabemos que $b \vee c$ é verdade.

Por Unidade de Resolução nós sabemos que b é verdade.

Será um Teste!

181

Usando o método da Resolução

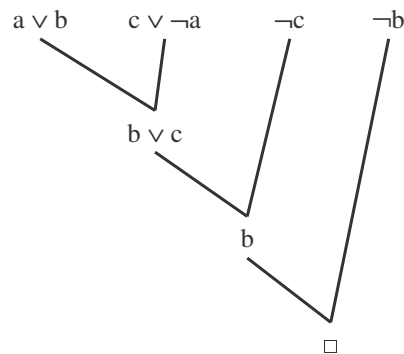
Premissa1: $a \vee b$

Premissa2: $c \vee \neg a$

Premissa3: $\neg c$

Conclusão: b

\neg Conclusão: $\neg b$



Utiliza uma única regra de inferência: de $A \vee B$, $C \vee \neg A$,
conclui-se $B \vee C$

182

Limites da Lógica Proposicional

- O poder expressivo da Lógica Proposicional é limitado, pois assume que qualquer coisa pode ser expressa em fatos.
- É muito mais fácil modelar objetos do mundo real usando *propriedades* and *relações*.
- Lógica de Predicados fornece estas habilidades mais formalmente e é utilizada na representação de muitos domínios na IA.

183

Lógica de Predicados

- *Termos* representam objetos específicos no *mundo* e podem ser constantes, variáveis ou funções.
- *Símbolos de Predicado* referem-se a relações particulares sobre objetos.
- *Sentenças* representam fatos, e são formadas de *termos*, *quantificadores* e símbolos de *predicados*.

184

Prolog – Uma linguagem de Programação Lógica

🐉 Características

- Provedor de teoremas (Verdade ou Falso)
- Linguagem declarativa
- Linguagem não tipada
- Linguagem é interpretada
- Não determinístico
- Diferente de programação procedimental (definição lógica dos problemas)
- Não existe variáveis globais
- Muito usado em IA (rápida prototipação)

185

Da Notação de Kowalski para um programa Prolog

Notação de Kowalski	Programa Prolog
chama(a,b) ← usa(b,e) ← depende(x,y) ← chama(x,y) depende(x,y) ← usa(x,y) depende(x,y) ← chama(x,z), depende(z,y)	chama(a,b). usa(b,e). depende(X,Y) :- chama(X,Y). depende(X,Y) :- usa(X,Y). depende(X,Y) :- chama(X,Z), depende(Z,Y).
← depende(a,e)	?- depende(a,e).

Fato ou
Cláusula Unitária

Regra

Questionamento

186

Raciocínio Monotônico

- Lógica de Predicados e Proposicional são *Monotônicas*: cada nova peça de informação estende a base de conhecimento, mas a semântica não é alterada
- Algumas vezes todo conhecimento para resolver um problema não está disponível - um sistema de raciocínio monotônico não pode trabalhar.

187

Raciocínio Não-Monotônico

- *Raciocínio Não-Monotônico* envolve inferências que podem reduzir ou modificar a base de conhecimento.
- Isto torna possível raciocinar com um conjunto incompleto de fatos.
- Em determinado momento, cada sentença é ou Verdadeira ou Falsa ou desconhecida. Isto não é diferente do Raciocínio Monotônico.

188

Problemas com Raciocínio Não-Monotônico

- Como é possível realizar inferências com falta de conhecimento?
- Como a base de conhecimento deve ser modificada quando um novo fato é adicionado?
- Como o conhecimento pode ser usado para resolver conflitos (contradições)?

189

Técnicas

- Sistemas Lógicos Formais que podem trabalhar com esses problemas têm sido desenvolvidos
- Algumas Definições:
 - *Interpretação* de um conjunto de sentenças: um conjunto de objetos e mapeamento de todos os predicados, funções e constantes.
 - *Modelo*: uma interpretação que satisfaz um conjunto de sentenças.

190

Raciocínio Default

• Lógica Não-Monotônica: Extensão da Lógica de Predicados:

- um operador modal M que representa “*consistente com qualquer coisa que é conhecida pelo sistema*”.

• Exemplo:

$\forall x,y: \text{Conhecido}(x,y) \wedge M \text{ Bom_Relacionamento}(x,y) \Rightarrow \text{Defenderá}(x,y)$

191

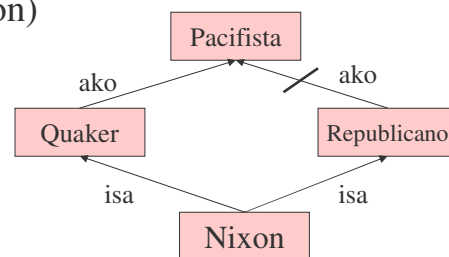
Problema com Lógica Não-Monotônica

• $\forall x: \text{Republicano}(x) \wedge M \neg \text{Pacifista}(x) \Rightarrow \neg \text{Pacifista}(x)$

• $\forall x: \text{Quaker}(x) \wedge M \text{ Pacifista}(x) \Rightarrow \text{Pacifista}(x)$

Republicano(Nixon)

Quaker(Nixon)



192

Lógica Default

• Nova regra de inferência:

$$\frac{A : B}{C}$$

- Se A é provável e é consistente acreditar em B (não há nada que prove que $\neg B$ é verdade), então C é verdade.

• Mesma Idéia, mas agora é usada como regra de inferência.

• A nova regra estende a base de conhecimento para um conjunto de extensões plausíveis, qualquer nova sentença que é verdadeira em todas as extensões é adicionada.

193

Abdução

• Dedução:

Dado $A(x) \Rightarrow B(x)$ e $A(x)$,
é assumido que $B(x)$ é verdade.

• Abdução:

Dado $A(x) \Rightarrow B(x)$ e $B(x)$,
é assumido que $A(x)$ verdade.

194

Herança

- É possível suportar herança usando Lógica Default:

Mamífero(x) : Patas(x,4)

Patas(x,4)

- Na falta de informação contraditória, pode-se assumir em qualquer ocasião que mamíferos têm 4 patas.
- *(também é necessário que uma regra indique que nenhum mamífero pode ter número de patas diferente)*

195

Sistemas de Produção



196

Regras de Produção

• Representam conhecimento com pares de *condição-ação*

- Se **condição** (ou **premissa** ou **antecedente**) ocorre
então ação (**resultado**, **conclusão** ou **conseqüente**) deverá ocorrer.
 - Se o agente percebe luz do freio do carro em frente acesa **então** ele deve frear o carro (**regra de ação**)
 - Se veículo tem 4 rodas e tem um motor
então veículo é um automóvel (**novo conhecimento**)

• São chamadas de **regras de produção** porque, quando utilizadas com *raciocínio progressivo*, **produzem** novos fatos a partir dos fatos e regras da BC.

- Esses novos fatos passam a fazer parte da BC

197

Regras de Produção

• Características:

- Representam conhecimento de forma modular
 - cada regra representa um “pedaço” de conhecimento independente
 - cuidado: a consistência deve ser mantida.
- São fáceis de compreender (legíveis) e de modificar
- Novas regras podem ser facilmente inseridas na BC
- Podem ser usadas tanto com raciocínio progressivo quanto com raciocínio regressivo.

198

Sistemas de Produção

• São sistemas baseados em **Regras de Produção**

• Consistem em 3 módulos principais:

- A **Base de Regras (BR)**: permanente
 - regras se-então e fatos conhecidos
- A **Memória de Trabalho (MT)**: temporária
 - base de fatos derivados durante a “vida” do agente
 - percepções do agente e fatos gerados a partir da BR pelo mecanismo de inferência
- O **Mecanismo (máquina) de Inferência (MI)**:
 - determina o método de raciocínio utilizado (progressivo ou regressivo)
 - utiliza estratégias de busca com casamento (unificação)
 - resolve conflitos e executa ações.

199

Arquitetura dos Sistemas de Produção

Base de Regras

Conhecimento Permanente

- fatos
- regras de produção

Meta-conhecimento

- estratégias para resolução de conflito

Memória de Trabalho

Conhecimento volátil

- descrição da instância do problema atual
- hipóteses atuais
- objetivos atuais
- resultados intermediários

Conjunto de conflito
conjunto de possíveis regras a serem disparadas

Mecanismo de Inferência

200

Exemplo de regras para veículos

- 🚲 Bicicleta: Se **veículoTipo=ciclo**
E **num-rodas=2**
E **motor=não**
Então **veículo=Bicicleta**
- 🚲 Triciclo: Se **veículoTipo=ciclo**
E **num-rodas=3**
E **motor=não**
Então **veículo=Triciclo**
- 🏍️ Motocicleta: Se **veículoTipo=ciclo**
E **num-rodas=2**
E **motor=sim**
Então **veículo=Motocicleta**

201

Exemplo de regras para veículos

- 🚗 CarroSport: Se **veículoTipo=automóvel**
E **tamanho=pequeno**
E **num-portas=2**
Então **veículo=CarroSport**
- 🚗 Sedan: Se **veículoTipo=automóvel**
E **tamanho=médio**
E **num-portas=4**
Então **veículo=Sedan**
- 🚗 MiniVan: Se **veículoTipo=automóvel**
E **tamanho=médio**
E **num-portas=3**
Então **veículo=MiniVan**

202

Exemplo de regras para veículos

- 🐼 UtilitárioSport: Se **veículoTipo=automóvel**
E **tamanho=grande**
E **num-portas=4**
Então **veículo=UtilitárioSport**
- 🐼 Ciclo: Se **num-rodas<4**
Então **veículoTipo=ciclo**
- 🐼 Automóvel: Se **num-rodas=4**
E **motor=sim**
Então **veículoTipo=automóvel**

203

Complementando os exemplos...

- 🐼 Meta-regras
 - Se **R1** e **R2** podem ser disparadas, escolha **R1**
 - Se **R1** e **R2** podem ser disparadas e **R1** foi disparada mais recentemente que **R2**, escolha **R2**
- 🐼 Fatos
 - **Veículo1**: tamanho=pequeno; num-portas=2; motor=sim
 - **Veículo2**: num-rodas=2; motor=não

204

Direções do raciocínio dedutivo

➤ Raciocínio progressivo (Forward)

- dos dados à conclusão - *data-driven inference*
- as regras da BC são usadas para gerar informação nova (novos fatos) a partir de um conjunto inicial de dados
- os fatos gerados passam a fazer parte da BC
 - ex.: *criminoso(West)*.

➤ Raciocínio regressivo (Backward)

- da hipótese aos dados - *goal-directed inference*
- usa as regras da BC para responder a perguntas
- prova se uma asserção é verdadeira
 - ex.: *criminoso(West)?*
- só processa as regras relevantes para a pergunta (asserção)

205

Raciocinando com Encadeamento Progressivo

➤ Dos dados à conclusão

- Parte dos fatos na BR e na memória de trabalho, buscando quais regras eles satisfazem, para produzir assim *novas conclusões (fatos)* e/ou *realizar ações*.

➤ Três etapas:

- Matching, Resolução de conflito, Ação

➤ É uma estratégia de inferência muito rápida

- usada em sistemas de monitoramento e diagnóstico em tempo real.

➤ Ferramentas comerciais que implementam esta estratégia: OPS5, OPS85, IBM, TIRS, JESS

206

Procedimento básico

Enquanto mudanças são realizadas na memória de trabalho faça:

- **Match:** Construa o conjunto de conflitos – o conjunto de todas os pares (R, F) tal que R é uma das regras e F é um subconjunto de fatos na MT que unificam com o antecedente (lado esquerdo) de R.
- **Resolução do Conflito:** Selecione um par para a execução.
- **Ação:** Execute as ações associadas com o conseqüente (lado direito) de R, depois de fazer as substituições usadas durante a unificação do antecedente com os fatos F. ²⁰⁷

Encadeamento progressivo: algoritmo

1. Armazena as regras da BR na máquina de inferência (MI) e os fatos na memória de trabalho (MT);
2. Adiciona os dados iniciais à memória de trabalho;
3. Compara o antecedente das regras com os fatos na MT.
 - Todas as regras cujo antecedente “**casa**” (**unifica**) com esses fatos podem ser **disparadas** e são colocadas no **conjunto de conflito**;
4. Usa o procedimento de **resolução de conflito** para selecionar uma única regra desse conjunto;
5. Dispara a regra selecionada e verifica o seu conseqüente:
 - a) se for um **fato**, atualiza a MT
 - b) se for uma **ação**, chama o procedimento que ativa os atuadores do agente e atualiza a MT
6. Repete os passos 3, 4 e 5 até o conjunto de conflito se tornar vazio. ²⁰⁸

Encadeamento progressivo: Busca e Casamento

🔍 Busca e Casamento (unificação)

- **Unifica** as premissas das regras com os fatos da memória de trabalho
- ex.: fatos e regras sobre automóveis
 - **MT1**: veloz(Kadet-2.0), veloz(BMW), veloz(Gol-2.0), veloz(Mercedes), importado(BMW), importado(Mercedes)
 - **BC**: Se veloz(x) e importado(x) **então** caro(x)
 - **MT2**: MT1 + {caro(BMW), caro(Mercedes)}

🔍 Geralmente, o antecedente de cada regra selecionada é **comparado** com os fatos na MT usando **busca gulosa (best-first)**

209

Encadeamento progressivo: Busca e Casamento (Matching)

🔍 Custo da busca-casamento

- Se a BR é muito grande, verificar todas as premissas de todas as regras a cada ciclo é **caro**

🔍 Solução (simples)

- uma vez realizadas as etapas iniciais de busca e casamento, o algoritmo atualiza o conjunto de conflitos levando em conta apenas o **conseqüente** da regra que foi disparada no último ciclo
 - ex1. conseqüente: **retract (número de rodas = 4)** verifica quais regras do conjunto de conflito deixam de ser válidas
 - ex2. conseqüente: **insert (número de rodas = 4)** verifica quais regras que contém esta premissa podem ser adicionadas ao ¹⁰ conjunto de conflito

Algoritmo Rete

- O **Algoritmo Rete** (“rede” em grego) é o algoritmo mais utilizado para implementação de sistemas de produção.
- Desenvolvido por Charles Forgy na Carnegie Mellon University em 1979.
 - Charles L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17-37, 1982.
- Rete é o único algoritmo cuja eficiência é independente do número de regras.
- É a base de vários shells de sistemas especialistas como: OPS5, ART, CLIPS e Jess.

211

Encadeamento progressivo: Busca e Casamento

- Outra solução: **algoritmo RETE** (rede).
 - elimina duplicações entre regras
 - minimiza o número de testes requeridos durante a fase de casamento
 - cria uma **rede de dependências** entre as regras da BR
 - que deve ser recriada sempre que as regras na base são modificadas

212

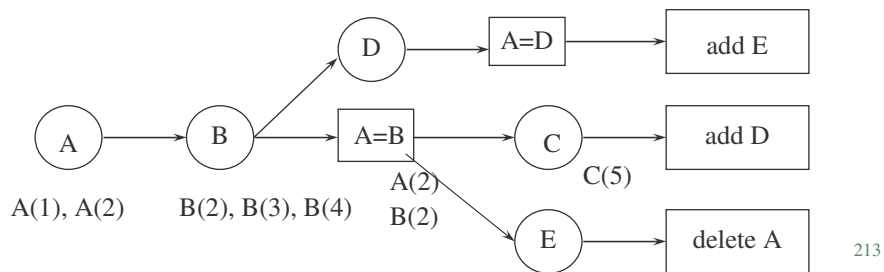
Algoritmo RETE: encadeamento progressivo

Base de Regras:

- $A(x) \wedge B(x) \wedge C(y) \Rightarrow \text{add } D(x)$
- $A(x) \wedge B(y) \wedge D(x) \Rightarrow \text{add } E(x)$
- $A(x) \wedge B(x) \wedge E(x) \Rightarrow \text{delete } A(x)$

Memória de Trabalho:

- $\{A(1), A(2), B(2), B(3), B(4), C(5)\}$



Encadeamento progressivo: Resolução de conflitos

Resolução de conflitos

- heurística geral para escolher um subconjunto de regras a disparar

Exemplos:

- **Não duplicação:** não executar a mesma regra com os mesmos argumentos duas vezes.
- **Prioridade de operação:** preferir ações com prioridade maior
 - semelhante aos sistemas ação-valor - LPO
- **Recency (“recenticidade”):** preferir regras que se referem a elementos da Memória de Trabalho criados recentemente.
- **Especificidade:** preferir regras que são mais específicas.

214

Encadeamento regressivo: Busca e Casamento

☛ Da hipótese aos dados

- Parte da **hipótese** que se quer provar, procurando regras na BR cujo **conseqüente** satisfaz essa hipótese.
- usa as regras da BR para responder a perguntas
- busca provar se uma asserção é verdadeira
 - ex.: **criminoso(West)?**
- só processa as regras relevantes para a pergunta

☛ Duas etapas:

- Busca e Casamento (unificação)

☛ Utilizado em sistemas de aconselhamento

- trava um “diálogo” com o usuário
- ex.: MYCIN

215

Encadeamento regressivo: algoritmo

1. Armazena as regras da BC no motor de inferência (MI) e os fatos na memória de trabalho (MT);
2. Adiciona os dados iniciais à memória de trabalho;
3. Especifica uma variável “*objetivo*” para o MI;
4. Busca o conjunto de regras que se referem à variável objetivo no conseqüente da regra
 - Isto é, seleciona todas as regras que atribuem um valor à variável objetivo quando disparadas.

Insere cada regra na pilha de objetivos;
5. Se a pilha de objetivos está vazia, pare.

216

Encadeamento regressivo: algoritmo

6. Selecione a regra no topo da pilha;
7. Tente provar que essa regra é verdadeira testando, um a um, se todos os seus antecedentes são verdadeiros:
 - a) se o 1o. antecedente é V, vá em frente para o próximo
 - b) se ele for F, desempilhe essa regra e volte ao passo 5
 - c) se o seu valor-verdade é desconhecido porque a variável do antecedente é desconhecida, vá para o passo 4 com essa variável como variável objetivo
 - d) se todos os antecedentes são V, dispare a regra, instancie a variável no conseqüente para o valor que aparece nessa regra, retire a regra da pilha e volte ao passo 5.

217

Encadeamento regressivo: Busca e Casamento

- O sistema percorre a BC em busca regras cujo conseqüente “casa” com a hipótese de entrada
- Se a hipótese de entrada é um fato (ex. criminoso(West)), a busca pára quando encontra a 1a. regra que casa com ele, e o sistema devolve uma variável booleana (V ou F).
- Se a hipótese tem alguma variável livre (ex. criminoso(X)), o sistema (programador) pode optar por devolver a 1a. instância encontrada, ou por devolver uma lista com todas as possíveis instâncias para aquela variável.
- Portanto, **não há conflito de execução de regras!**
- Unificação é realizada com busca em profundidade

218

Encadeamento regressivo: Exemplo no domínio dos veículos

➤ Carregar a BR de veículos no MI e os fatos na MT

➤ **Fatos iniciais:**

- num-rodas=4, motor=sim, num-portas=3, tamanho=médio

➤ Especificar variável objetivo

- **veículo=?**

➤ Pilha de objetivos

- regras com **variável objetivo** no **consequente**
 - as 7 primeiras regras da nossa BC

219

Encadeamento regressivo: Exemplo no domínio dos veículos

➤ Tenta provar verdadeiros os antecedentes da 1a. regra usando **busca em profundidade**

- **Bicicleta:** Se veículoTipo=ciclo
E num-rodas=2
E motor=não
Então veículo=*Bicicleta*

➤ *VeículoTipo=ciclo* não aparece na MT

- nova variável objetivo

➤ Atualiza pilha de objetivos

- inclui regras com nova variável objetivo no consequente
 - apenas a penúltima regra da nossa BC

220

Encadeamento regressivo

- *veículoTipo=ciclo* só é verdade em apenas uma regra
 - **Ciclo:** Se $\text{num-rodas} < 4$
Então $\text{veículoTipo} = \text{ciclo}$
- Verifica o valor verdade dos antecedentes da regra
 - $\text{num-rodas} < 4 \implies \text{FALSO!}$
- Donde se deduz que *veículo=Bicicleta* é **Falso!**

221

Encadeamento regressivo

- Se o **fato** a ser provado **não aparece** explicitamente na base e nem pode ser deduzido por nenhuma outra regra, **duas** coisas podem ocorrer, dependendo da implementação do sistema
 - o fato é considerado FALSO
 - ex. Prolog
 - o sistema consulta o usuário via sua interface
 - ex. Sistema ExpertSinta

222

Suposição do Mundo Fechado (SMF)

- SMF diz que os únicos objetos que satisfazem um predicado são aqueles que devem ser conhecidos pelo sistema.
- Muito poderosa em alguns domínios: Bases de Dados.
- Um problema:
 - A sintaxe indica que afirmações positivas têm prioridade sobre afirmações negativas. A afirmação usada deve ser selecionada cuidadosamente.

223

Encadeamento regressivo

- Desempilha as outras regras, uma a uma, até encontrar a regra abaixo - que vai dar certo!
 - **MiniVan:** Se veículoTipo=automóvel
E tamanho=médio
E num-portas=3
Então veículo=*MiniVan*
- *VeículoTipo=automóvel* não existe na MT
 - **Automóvel:** Se num-rodas=4 **OK! (1)**
E motor=sim **OK! (2)**
Então veículoTipo=*automóvel* **==> OK! (3)**
- Tenta provar os outros antecedentes da regra, que estão todos instanciados na MT, e são verdadeiros!
- **veículo=*MiniVan* é verdade!**

224

Regras com fator de incerteza

- ☛ Geralmente, é necessário associar-se um fator de incerteza (ou de confiança) a algumas regras na BR

- ☛ Incerteza nos dados e na aplicação das regras

If (previsão-do-tempo = chuva) > 80%
and (previsão-períodos-anteriores = chuva) = 85%
then (chance-de-chuva = alta) = 90%

- ☛ Infelizmente ...

- combinar as incertezas dos antecedentes neste caso propaga erros
- só uma abordagem probabilista pode tratar este tipo de incerteza corretamente

225

Vantagens e Limitações dos Sistemas de Produção

- ☛ Vantagens

- As regras são de fácil compreensão
- Inferência e explicações são facilmente derivadas
- Manutenção é relativamente simples, devido a modularidade
- São mais eficientes que os sistemas de programação em lógica, embora menos expressivos

- ☛ Desvantagens

- Conhecimento complexo requer muitas (milhares de) regras
- Esse excesso de regras cria problemas para utilização e manutenção do sistema
- Não são robustos (tratamento de incerteza)
- Não aprendem

226

Exemplo de Shell

☛ Shell: ambiente que facilita construção de bases de conhecimento

☛ ExpertSinta

- Construído por Ricardo Bezerra
- Lógica de ordem 0+ (atributo-valor)
- Usa encadeamento regressivo
- Implementado em Delphi

227

Exercício

☛ Utilize o Expert Sinta para criar um sistema especializado em prever se um artigo científico poderá ou não ser apresentado em uma conferência internacional.

228

Sistemas de Preenchimento de Slots

229

Estruturas de Preenchimento de Slots

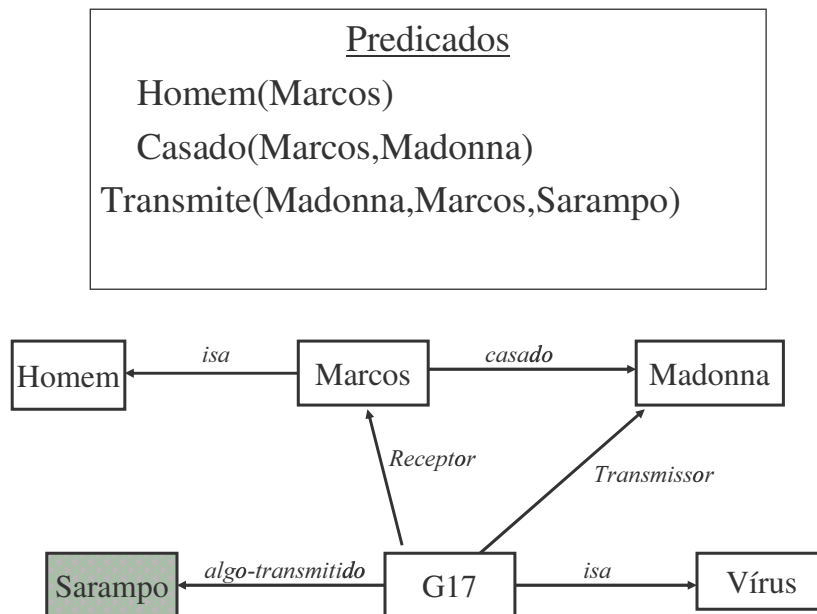
- Conhecimento é representado por um conjunto de entidades, seus atributos e relações entre entidades.
- *Estruturas de Slots Fracos* são modelos gerais que podem ser usados em uma grande variedade de domínios.
- *Estruturas de Slot Fracos* contém estratégias gerais de inferência que devem ser revisadas quando aplicadas em domínios específicos (Redes Semânticas, Frames)
- *Estruturas de Slot Fortes* contém extensões e/ou restrições que são específicas ao domínio (Dependência Conceitual)

230

Redes Semânticas

- Nós representam entidades e arcos representam relações entre nós.
- Rede de Herança é um bom exemplo.
- É possível transformar cada arco em um predicado binário que relaciona 2 nós.
- É possível, também, criar uma rede semântica para representar uma coleção de predicados.

231

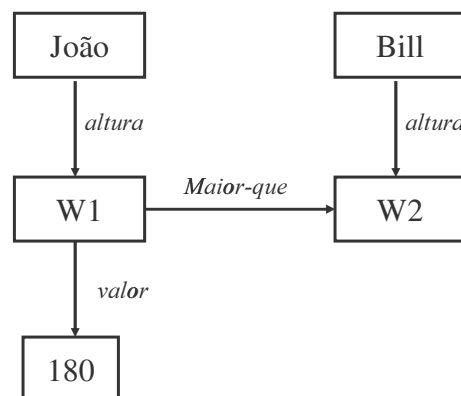


232

Arcos Entidade vs. Relações

- Alguns arcos definem novas entidades, outros relacionam entidades existentes.
- É possível criar novos nós para suportar relações entre os atributos de 2 entidades.
- Se é desejado ser flexível, é também possível associar atributos àqueles nós!

233



Redes Semânticas

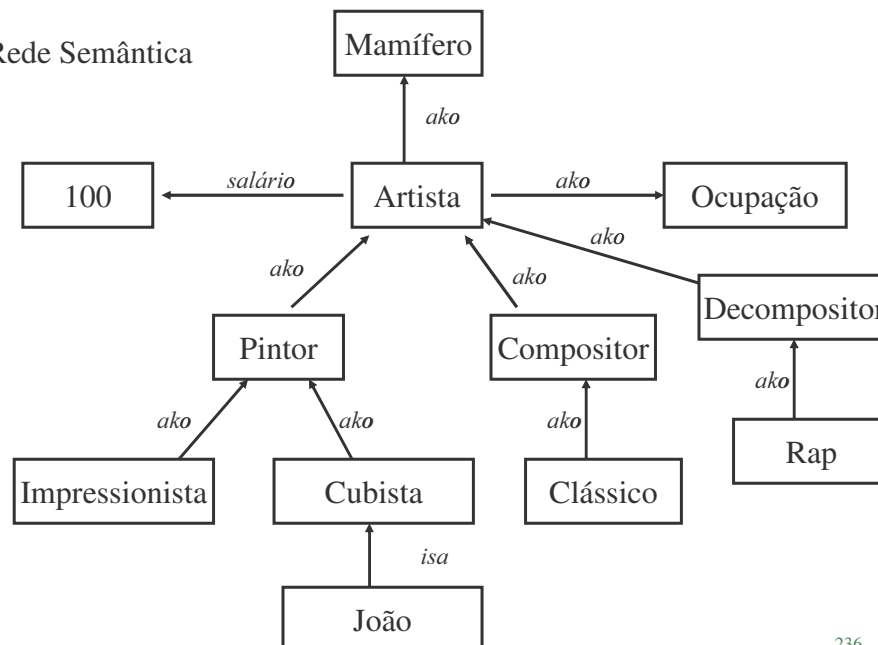
234

Tipos de Nós

- Alguns nós (nós classe) representam uma classe de objetos das quais todos compartilham (herdam) algum atributo default.
- Os atributos herdados podem não ser propriedades da classe pai!
- O próprio nó classe pode ter atributos ou relações.

235

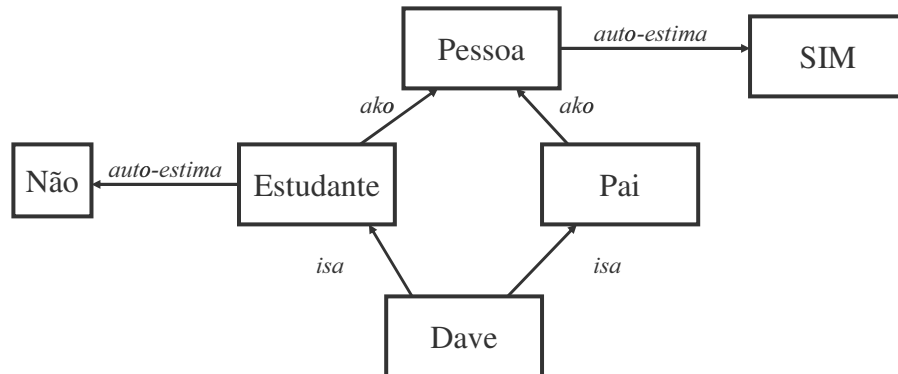
Rede Semântica



236

Múltipla Herança

- Redes Semânticas podem suportar múltipla herança, portanto, é possível revisar o algoritmo básico de herança.



Dave tem auto-estima?

237

Frames

- Um Frame é uma coleção de atributos e valores associados a um objeto (ou um conjunto de objetos).
- O valor de um atributo pode ser um outro frame (isto representa a ligação entre frames).
- Um Frame possui os atributos de um objeto e o próprio frame é um objeto.

238

Frames (Cont.)

- Objetos pertencem a Classes
- Um objeto pode pertencer a mais de uma classe
- Objetos podem estar dispostos em uma taxonomia que permite herança de propriedades
- Objetos podem possuir uma representação complexa

239

Proposta de Frames

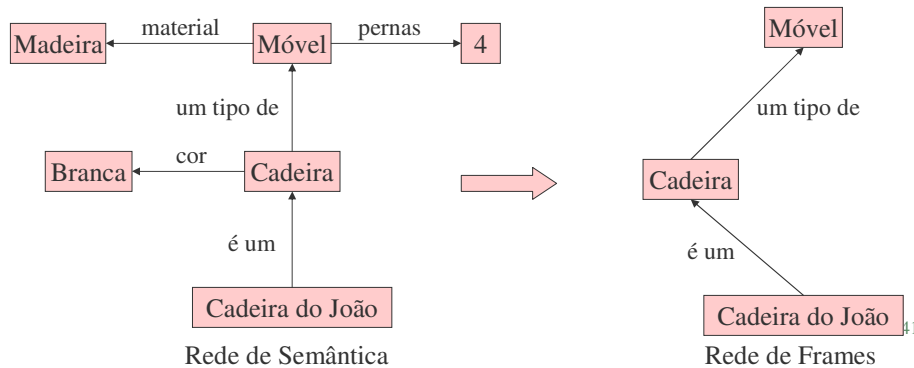
- Criada em 1974 por Marvin Minsky
- Objetivo de representar grandes quantidades de dados de forma estruturada
- Frames podem estar relacionados e compartilhar similaridades
- A disposição dos frames forma uma rede semântica

240

Proposta de Frames (cont.)

☞ A rede semântica deve:

- permitir a identificação de cada objeto
- permitir a identificação das propriedades de cada objeto
- permitir a navegação em busca de determinada informação



Estrutura dos Frames

☞ Identificado por um nome

☞ Formado por um conjunto de slots

- representam as propriedades do frame

☞ Slots obrigatórios

- *nome*: nome do frame
- *ako/isa*: identifica o frame hierarquicamente superior

☞ Slots possuem facetas que indicam o tipo do valor do slot

Estrutura dos Frames (Cont.)

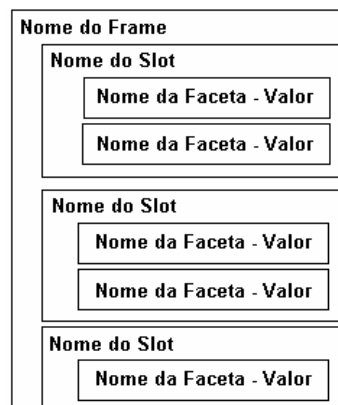
Tipos de Facetas

- **tipo**: tipo de dado (inteiro, real, string,...)
- **domínio**: valores possíveis para o slot (faixa entre valores, etc.)
- **valor**: indica o único valor presente no slot
- **default**: valor default, caso nenhum outro seja especificado
- **aponta frame**: permite que o slot seja descrito por outro frame
- **if-added, if-needed, if-removed**: procedimentos para determinar o valor do slot

243

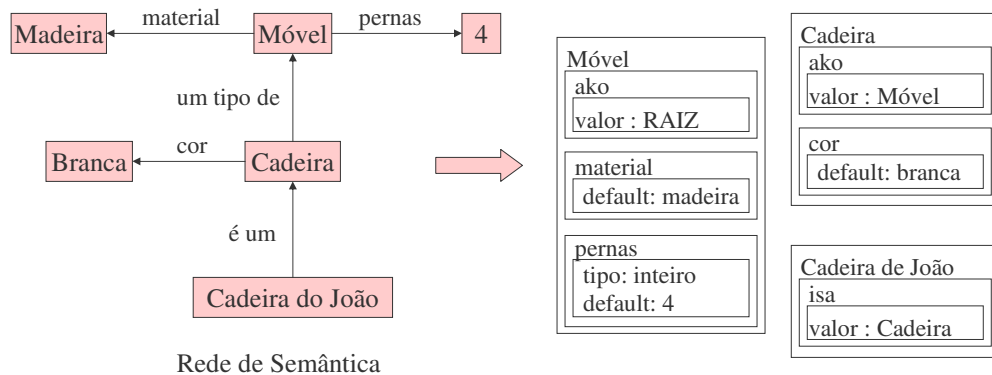
Estrutura dos Frames (Cont.)

- Estrutura genérica dos frames



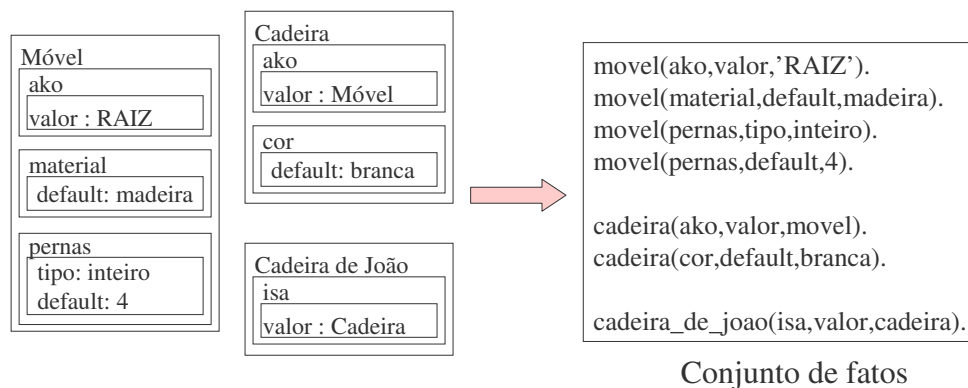
244

Estrutura dos Frames (Cont.)



245

Representação de Frames em Prolog



246

Herança em Frames

➤ Objetivos

- herança de características entre objetos
- simplificação e organização do conhecimento
- utilizar a hierarquia taxonômica do *conhecimento de senso comum*

➤ Objetos (subclasses) podem herdar características de classes hierarquicamente superior (superclasses)

247

Características da Herança em Frames

- Geralmente aplicada como um algoritmo de travessia em grafos
- Objetos semelhantes são agrupados em classes de objetos
- Ligações isa/ako entre objetos permitem realizar generalizações e deduzir novas informações a partir de uma *rede de herança*

248

Herança em Frames (Cont.)

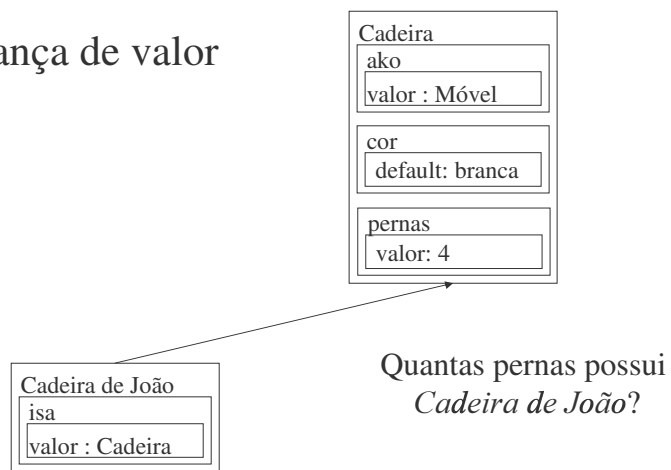
☛ Reconhecimento é feito tentando preencher os slots

- Herança de valor
- Herança de procedimentos
- Herança por valor default

249

Herança em Frames (Cont.)

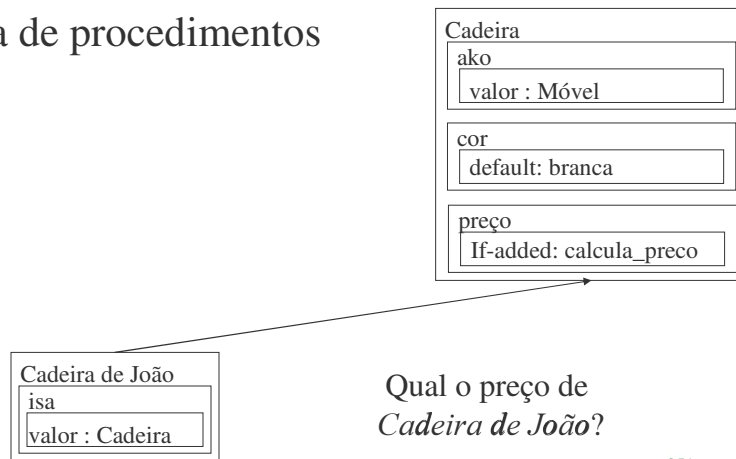
☛ Herança de valor



250

Herança em Frames (Cont.)

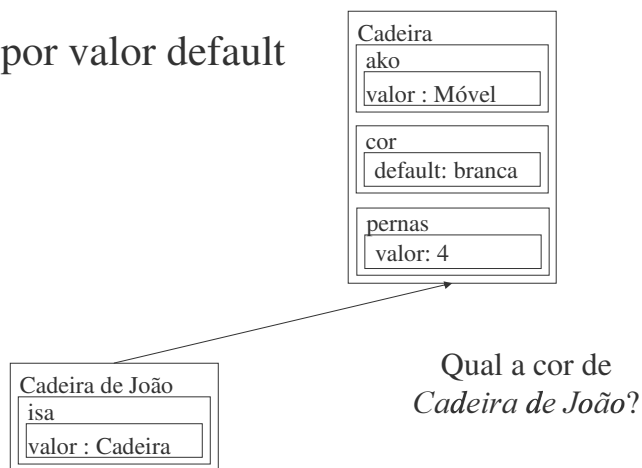
Herança de procedimentos



251

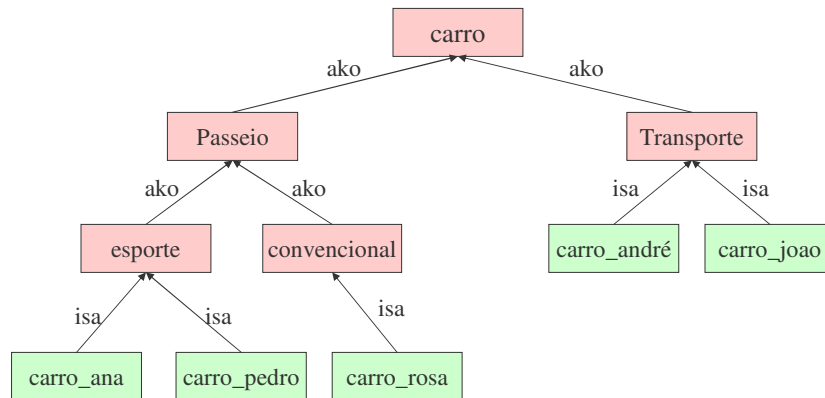
Herança em Frames (Cont.)

Herança por valor default



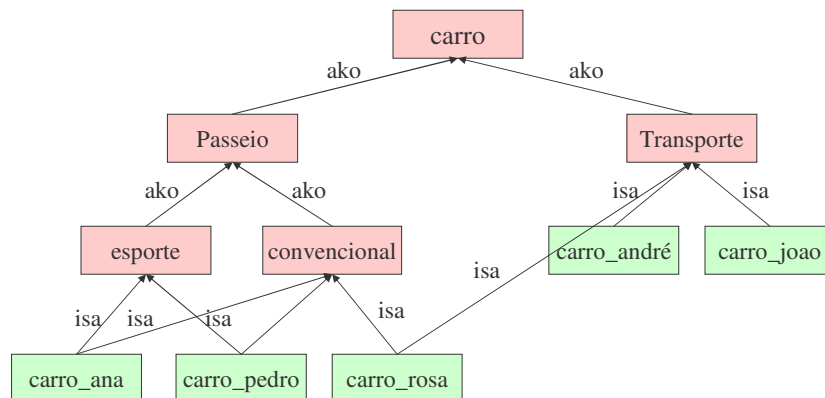
252

Redes de Herança Simples em Frames



253

Redes de Herança Múltipla em Frames



254

Raciocinadores de Herança

❧ Raciocinadores do Menor Caminho

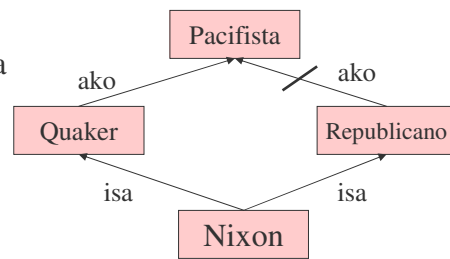
- solução mais próxima na hierarquia

❧ Raciocinadores Crédulos

- escolhe arbitrariamente uma solução

❧ Raciocinadores Céticos

- nenhuma solução é escolhida



255

Meta-Classes

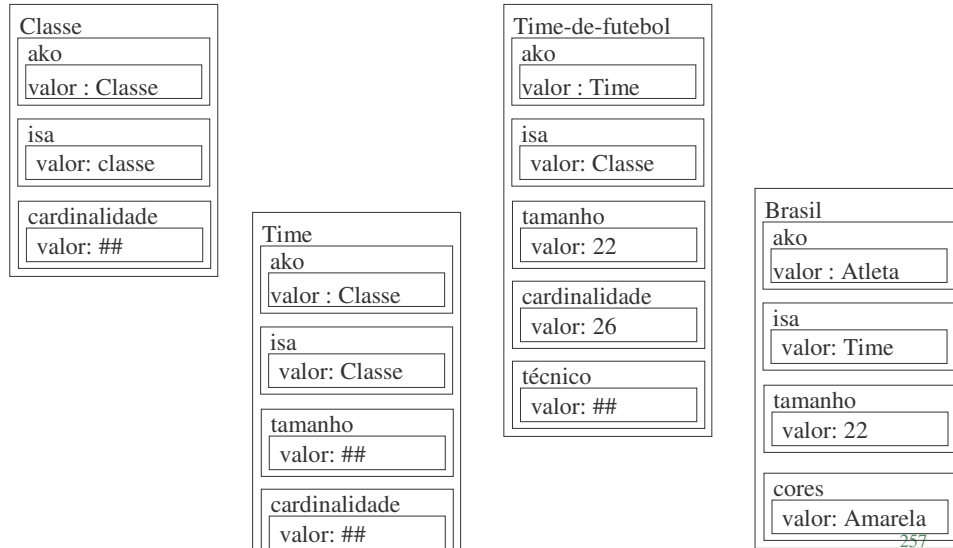
❧ Algumas vezes é desejável *herdar* propriedades de classes genéricas.

❧ É necessário um tipo especial de nó que possua propriedades de classes compartilhadas, este nó representa uma *meta-classe*.

256

Exemplo de Meta-Classe

As classes de todos os times de futebol



Outras Relações Entre Classes

- Classes podem estar relacionadas por ligações *ako*
- Objetos e Classes são relacionados por ligações *isa*
- É possível trabalhar com outras relações:
 - mutuamente-disjuntas: não compartilham elementos
 - coberta-por: todos os elementos estão contidos em um conjunto de subclasses.

258

Dependência Conceitual

259

Conceito

- Representação para expressar **eventos** em expressões lingüísticas
- A estrutura básica deste nível é a conceitualização: construção **ator-ação-objeto** ou **objeto-estado**
- Se uma ação está presente então suas circunstâncias também estão. Uma delas é o instrumental (também uma conceitualização)

260

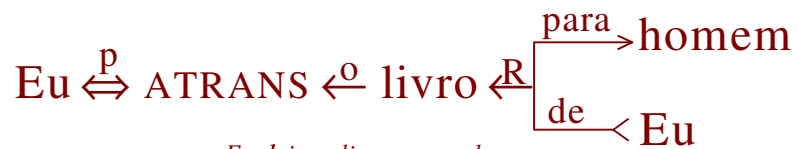
Objetivo

🎯 Representar o conhecimento de tal forma que:

- Seja **independente da linguagem** em que as frases foram originalmente expressas
- Sirva como base para programas de computador que entendem linguagem natural o suficiente para **fazer paráfrases e inferências** a partir de sentenças de entrada

261

Exemplo



Eu dei um livro para o homem

🎯 Onde

- \leftarrow indica a direção da dependência
- \Leftrightarrow indica dependência entre ator e ação (ACT)
- **p** indica tempo conceitual (passado)
- **ATRANS** indica transferência de posse (ação)
- **o** indica circunstância conceitual (Objeto)
- **R** indica circunstância conceitual (Recipiente)
- *Eu* e *Homem* são **PP's** (produtores de ação)
- *livro* é **PA** (modificador de ação)

262

Conceitualização

- Qualquer ação deve ser real e deve poder ser executada em algum **objeto** por um **ator**.
- Atores, ações e objetos no esquema conceitual devem corresponder a respectivas entidades do mundo real
- Conceitualizações possuem um esquema de representação

263

Esquema de Representação

- Dependência mútua entre ator-ação: \Leftrightarrow
- Relação objeto-estado:
 - \Leftrightarrow afirmação de um atributo
 - $\Leftarrow \boxed{\rightarrow}$ mudança de estado
 - \Uparrow relação causal indicada entre o causador e a causa, denota dependência temporal. Pode existir somente entre dependências duplas. Somente eventos ou estados podem causar eventos ou estados

264

Primitivas Conceituais

- 🐼 ACTs : Ações
- 🐼 PPs : Objetos (produtores de ações)
- 🐼 AAs : Modificadores de ações
- 🐼 PAs : Modificadores de PPs

265

Representação de uma sentença

- 🐼 “É possível representar uma grande parte dos significados das linguagens naturais pelo uso de um esquema de representação conceitual que inclui somente **quatorze ações** básicas, um conjunto infinito de objetos e um pequeno número de **estados**, em adição a aproximadamente **dezesseis regras** governando a combinação desses itens”

Schank (May, 1973)
266

Ações Primitivas

- O número total de ações (ACT) necessárias para dar o significado de qualquer sistema em linguagem natural é **quatorze**
- Entretanto, Schank em artigo de março de 1973, admite que podem ser necessárias outras ações (especificamente de sentimentos), cujo critério de decisão para criação baseia-se nas inferências

267

Ações Primitivas

- Cada ACT requer ainda três ou quatro circunstâncias conceituais (OBJETO, INSTRUMENTO, RECIPIENTE ou DIREÇÃO - O, I, R, D)
- Só é considerado ACT aquilo que pode ser executado por alguém. Por exemplo, *dormir* é considerado um estado.

268

Ações Primitivas

☛ Há quatro categorias de ACT nas quais as quatorze ações são divididas:

- Globais
- Físicas
- Mentais
- Instrumentais

269

Ações Primitivas

☛ Globais

- ATRANS - transferência de uma relação abstrata (*dar*)
- PTRANS - transferência de local físico de um objeto (*ir*)

270

Ações Primitivas

🐾 Físicas:

- PROPEL - aplicação de força física a um objeto (*empurrar*)
- MOVE - movimento de uma parte do corpo por seu dono (*chutar*)
- GRASP - domínio de um objeto por um ator (*agarrar*)
- INGEST - ingestão de um objeto por um animal (*comer*)
- EXPEL - expulsão de algo do corpo de um animal (*chorar*)

271

Ações Primitivas

🐾 Mentais:

- MTRANS - transferência de informação mental entre animais ou dentro de um animal (*contar*). Divide-se a memória em CP (processador consciente), LTM (memória de longo prazo) e órgãos dos sentidos.
- MBUILD - elaboração de novas informações a partir de informações antigas (*decidir*)
- CONC - ato de conceitualizar. Pode ser: focar atenção em algo ou executar processamento mental (*pensar, sonhar*)

272

Ações Primitivas

👂 Instrumentais:

- SPEAK - produção de sons (*dizer*)
- SMELL - ato de direcionar o nariz a um odor (*cheirar*)
- LOOK-AT - direcionar os olhos a um objeto físico (*olhar*)
- LISTEN-TO - direcionar os ouvidos a um objeto sonoro (*ouvir*)

273

Ações Primitivas

👂 ***ATTEND*** - *focagem de um órgão dos sentidos em um estímulo (escutar)*

👂 ***DO*** - *representa uma ação desconhecida*

274

Estados

- Podem ser descritos por escalas de valores numéricos
- Utilizados para detectar diferenças entre adjetivos, por exemplo *bravo* é um pouco menos que *furioso*

275

Estados

- Saúde: vai de -10 a +10
 - morto -10
 - gravemente doente -9
 - doente -9 a -1
 - bem 0
 - muito bem +7
 - perfeita saúde +10

276

Estados

☞ Estado Mental: vai de -10 a 10

- catatônico -9
- deprimido -5
- abalado -3
- triste -2
- Ok 0
- feliz +5
- extasiado +10

277

Estados

☞ Grau de Consciência: vai de 0 a +10

- inconsciente 0
- dormindo 5
- acordado 10
- “ percepção sob efeito de drogas” > 10

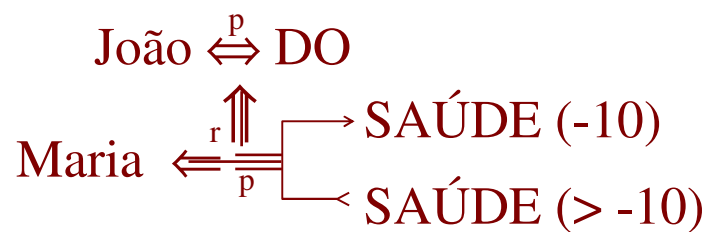
278

Estados

- Alguns estados têm valores absolutos e não em escalas: Length (Size), Color, Light Intensity, Mass, Speed, etc.
- Outros são relacionamentos entre objetos: Control, Part (possessão inalienável), Poss (possessão), Ownership, Contain, Proximity, Location, Physical Containment
- MFEEL expressa a relação entre duas pessoas e uma emoção

279

Exemplo



João matou Maria

280

Regras

1. **Atores** executam **ações**
2. **Ações** têm **objetos**
3. **Ações** têm **instrumentos**
4. **Ações** podem ter **recipientes**
5. **Ações** podem ter **direções**
6. **Objetos** podem estar relacionados a outros **objetos**. Essas relações são: *Possession*, *Location* e *Containment*

281

Regras

7. **Objetos** podem ter **atributos**
8. **Ações** podem ter **atributos**
9. **Atributos** têm **valores**
10. **Conceitualizações** podem ter **tempo**
11. **Conceitualizações** podem ter **localização**

282

Regras

- 12. Conceitualizações podem causar mudança no valor de atributos de objetos**
- 13. Conceitualizações podem habilitar a ocorrência de outras conceitualizações**
- 14. Conceitualizações podem servir de razão para conceitualizações**

283

Notação adicional

☞ **Existência de conceitualização:** \Leftrightarrow

☞ **Localizações:** \Leftrightarrow LOC

- ex.: $X \Leftrightarrow \text{LOC}(Y)$

X está localizado em Y

☞ **Tempo (marcado sobre \Leftrightarrow)**

- **p** - Passado
- **f** - Futuro
- **t** - Transição
- **t_s** - Iniciar transição
- **t_f** - Transição encerrada
- **k** - Continuidade
- Δ - Eterno
- ϕ - Presente

284

Notação adicional

🐼 Causas

- $\Uparrow r$ - Resultado
- $\Uparrow R$ - Razão
- \Uparrow - Causa física
- $\Uparrow E$ - Condição de permissão

🐼 ? - Interrogação

🐼 / - Negação

🐼 c - Condicional

🐼 \nsubseteq - incapaz de

285

Exemplos de regras

🐼 Relação entre um ator e o evento que ele causa

João \xRightarrow{p} PTRANS
João correu

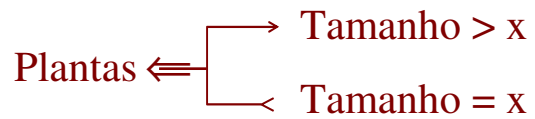
🐼 Relacionamento entre uma ação e o objeto (produtor de situação) que é objeto daquela ação.

João \xRightarrow{p} PROPEL \xleftarrow{o} carrinho
João empurrou o carrinho

286

Regras

- 🌱 Relação entre um ator e um estado no qual ele começou e terminou



As plantas cresceram

287

Inferências

- 🌱 “... uma inferência é uma conceitualização que é verdade com algum grau de probabilidade sempre que outra conceitualização ou conjunto de conceitualizações forem verdade.”

Schank (Mach, 1973)

288

Exemplo

- ☛ *João foi para São Paulo*
- ☛ Não está explícito que João, na realidade, chegou lá. A representação conceitual dos dois fatos é diferente.
- ☛ Mas podemos inferir isso se soubermos que essa possibilidade é válida
- ☛ Para cada ação há regras de inferência

289

Argumentos para o uso da DC

- ☛ Menos regras de inferência são necessárias
- ☛ Muitas inferências já estão presentes na própria representação
- ☛ As estruturas terão espaços que deverão ser preenchidos. Sobre estes pode recair o foco do programa que pretende compreender as frases

290

Argumentos contra seu uso

• Formalismo de representação

- O conhecimento precisa ser decomposto em primitivas de nível bastante baixo

• É apenas uma teoria da representação de eventos

- Não representa outros tipos de conhecimento

291

Incerteza em conhecimento



292

Causas de incerteza em representação de conhecimento

➤ Entrada de dados

- Dados Incompletos
- Dados com ruído

➤ Conhecimento é incerto

- Múltiplas causas podem levar a diferentes problemas
- Enumeração incompleta de condições e efeitos
- Relações de causabilidade incompletas no domínio
- Efeitos probabilísticos/estocásticos

➤ Saídas Incertas

- Abdução e indução são processos inerentemente incertos
 - Raciocínio default, mesmo em dedução, é incerto
 - Inferência dedutiva incompleta pode ser incerta
- Raciocínio probabilístico fornece apenas resultados probabilísticos (sumariza incerteza de várias fontes)

293

Tomada de decisão com incerteza

➤ Comportamento racional:

- Para cada ação identifique as possíveis desvantagens
- Calcule a probabilidade de cada desvantagem
- Calcule a utilidade de cada desvantagem
- Calcule a utilidade esperada para possíveis desvantagens para cada ação
- Selecione a ação com maior utilidade esperada (princípio da **Utilidade Esperada Máxima**)

294

Raciocínio Bayesiano

Teoria da Probabilidade

Inferência Bayesiana

- Usa teoria da probabilidade e informação sobre independência
- Da evidência (efeitos) para conclusões (causas) ou causalidade (de causas para efeitos)

Redes Bayesianas

- Representação compacta da distribuição de probabilidade sobre um conjunto de variáveis randômicas proposicionais
- Baseado em relações de independence

295

Outras representações de incerteza

Raciocínio default

- Lógica não-monotônica : permite a retração de crenças default se elas comprovadamente são falsas

Métodos baseados em regras

- Fatores de Certeza (Mycin): propaga simples modelos de crenças através de regras causais ou de diagnóstico

Raciocínio Evidencial

- Teoria de Dempster-Shafer : $Cr(P)$ é uma medida de evidência para P ; $Dr(\neg P)$ é uma evidência contra P (descrença); Juntos eles definem um intervalo de crença

Raciocínio Fuzzy

- Conjuntos Fuzzy : Grau de pertinência
- Lógica Fuzzy : “Quão verdadeira” é uma sentença lógica?

296

Exemplo de Raciocínio Probabilístico

- Identificar a doença $H \in \{H_1, H_2, H_3\}$ mais provável dados os sintomas do conjunto $E = \{E_1, E_2\}$
- Informações sobre o problema:

i	1	2	3
$p(H_i)$	0,5	0,3	0,2
$p(E_1 H_i)$	0,4	0,8	0,3
$p(E_2 H_i)$	0,7	0,9	0,0

297

Exemplo de Raciocínio Probabilístico

$$p(E_1 E_2 | H_1) = 0,4 \times 0,7 \times 0,5$$

$$p(E_1 E_2 | H_2) = 0,8 \times 0,9 \times 0,3$$

$$p(E_1 E_2 | H_3) = 0,3 \times 0,0 \times 0,2$$

i	1	2	3
$p(H_i)$	0,5	0,3	0,2
$p(E_1 H_i)$	0,4	0,8	0,3
$p(E_2 H_i)$	0,7	0,9	0,0

$$p(H_i | E_1 \dots E_n) = \frac{p(E_1 \dots E_n | H_i) \times p(H_i)}{\sum_{k=1}^m (p(E_1 \dots E_n | H_k) \times p(H_k))} \quad \Leftarrow \text{Fórmula de Bayes}$$

$$p(H_1 | E_1 E_2) = \frac{0,4 \times 0,7 \times 0,5}{(0,4 \times 0,7 \times 0,5) \times (0,8 \times 0,9 \times 0,3) \times (0,3 \times 0,0 \times 0,2)} = 0,393$$

$$p(H_2 | E_1 E_2) = \frac{0,8 \times 0,9 \times 0,3}{(0,4 \times 0,7 \times 0,5) \times (0,8 \times 0,9 \times 0,3) \times (0,3 \times 0,0 \times 0,2)} = 0,607$$

$$p(H_3 | E_1 E_2) = \frac{0,3 \times 0,0 \times 0,2}{(0,4 \times 0,7 \times 0,5) \times (0,8 \times 0,9 \times 0,3) \times (0,3 \times 0,0 \times 0,2)} = 0,0$$

298

Desafios sobre tratamento de incerteza

- **Redes bayesianas:** Boas propriedades teóricas combinadas com raciocínio eficiente fazem RB muito popular; expressividade limitada
- **Lógica não-monotônica :** representa raciocínio de senso comum, mas pode ser computacionalmente muito cara
- **Fatores de certeza:** Semanticamente não muito bem fundamentada
- **Teoria de Dempster-Shafer:** Tem boas propriedades formais, mas pode ser computacionalmente muito cara
- **Raciocínio Fuzzy:** Semântica não muito clara (fuzzy!), mas tem se mostrado muito útil para aplicações comerciais

299

Exemplo de tratamento de incerteza no Shell SINTA

Caso 1: exemplo

continuação

SE fumagina = sim 80%

ENTÃO suspeita de praga = mosca branca,
grau de confiança (CNF) 70%.

Cálculo:

- teremos que à variável **suspeita de praga** será atribuído o valor mosca branca, com o respectivo grau de confiança $0.80 * 0.70 = 0.56 = 56\%$.

300

Exemplo de tratamento de incerteza no Shell SINTA

Caso 2:

- Quando deseja-se calcular o grau de confiança envolvendo o operador **E**.

Cálculo :

- Se possuímos duas igualdades $var_1 = value_1$ e $var_2 = value_2$, com os respectivos graus de confiança c_1 e c_2 , temos que a sentença $var_1 = value_1$ E $var_2 = value_2$ retornará como valor de confiança $c_1 \times c_2$.

301

Exemplo de tratamento de incerteza no Shell SINTA

Caso 2: exemplo

continuação

- **SE** estados das folhas = esfurelam facilmente
E presença de manchas irregulares = sim...

Cálculo:

- Se o grau de confiança da igualdade estados das folhas = esfurelam facilmente é 80% e o grau de confiança da igualdade presença de manchas irregulares = sim é 70%, temos que a conjunção das duas sentenças retornará um valor CNF de 56%.

302

Exemplo de tratamento de incerteza no Shell SINTA

Caso 3:

- Quando deseja-se calcular o grau de confiança com o operador OU.

Cálculo :

- Se possuímos duas igualdades $var_1 = value_1$ e $var_2 = value_2$, com os respectivos graus de confiança c_1 e c_2 , temos que a sentença $var_1 = value_1$ OU $var_2 = value_2$ retornará como valor de confiança $c_1 + c_2 - c_1 \times c_2$.

303

Exemplo de tratamento de incerteza no Shell SINTA

Caso 3: exemplo

continuação

- SE besouros vermelhos = sim 80%
OU larvas marrons = sim 70%...

Cálculo :

- a disjunção das duas sentenças retornará um valor
 $CNF \text{ de } 0.70 + 0.80 - 0.70 \times 0.80 = 1.50 - 0.56$
 $= 0.94 = 94\%$.

304

Exemplo de tratamento de incerteza no Shell SINTA

Caso 4:

- Quando deseja-se atualizar o grau de confiança de uma variável que se encontra na memória de trabalho.

Situação

- A variável doença possuía valor **mofo preto** com grau de confiança 60%. Após a aplicação de outras regras chegou-se a uma outra atribuição **doença = mofo preto**, desta vez com CNF 50%.

Cálculo

- O cálculo se dá da mesma forma que a regra do OU:
 $0.60 + 0.50 - 0.60 * 0.50 = 1.10 - 0.30 = 0.80 = 80\%$.

305

Exemplo de tratamento de incerteza no Shell SINTA

Caso 5:

- Quando uma variável recebe duas vezes o mesmo valor em pontos diferentes da consulta.

Cálculo:

- Em momentos diferentes de uma consulta, uma mesma variável var pode receber o mesmo valor v, sendo que até a penúltima instanciación ela possuía grau de confiança c_1 , e a última atribuiu um CNF c_2 .

Sendo assim, temos que o valor final de confiança para $var = v$ será dado pela fórmula $c_a + c_n - c_a * c_n$, onde c_a representa o grau de confiança antes da última mudança e c_n o último grau de confiança atribuído.

306

Exemplo de tratamento de incerteza no Shell SINTA

Notas

- O sistema admite 50% como valor mínimo de confiança para que uma igualdade seja considerada verdadeira.
- Observe que as funções para conjunção e disjunção utilizadas seguem a Teoria das Possibilidades, não envolvendo nenhum tratamento estatístico mais aprofundado.
- É possível mudar as fórmulas utilizadas.

307

Aprendizado



Capítulo 5

308

O que é Aprendizado?

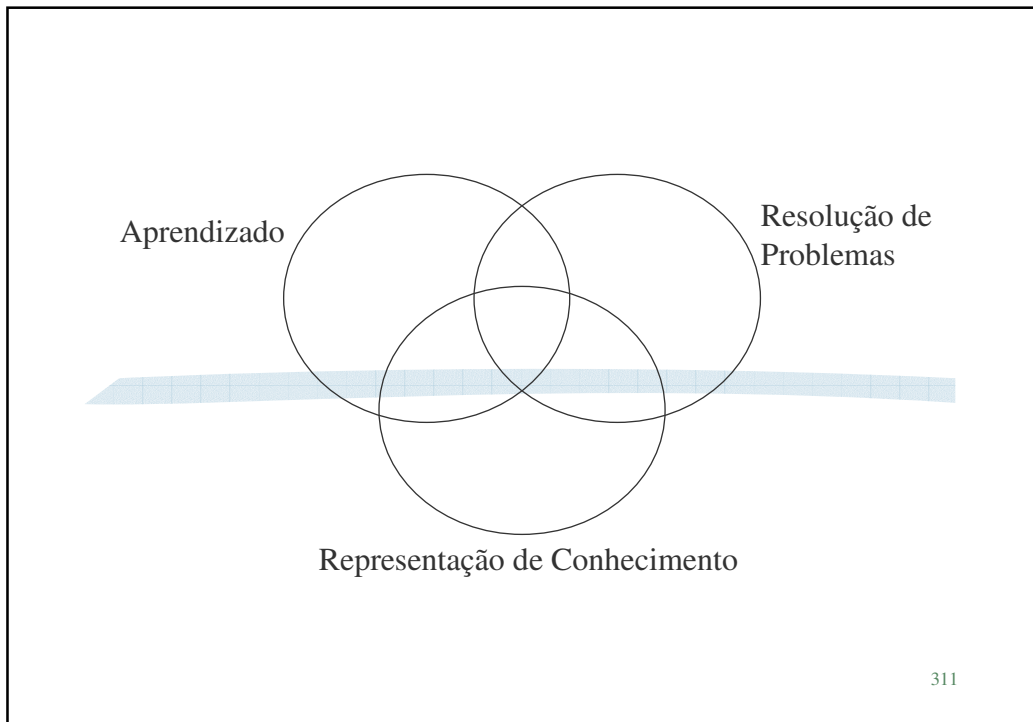
- ☛ “...mudanças que tornam sistemas capazes de adaptar-se e realizar tarefas semelhantes de maneira mais eficiente” [Simon]

309

Tipos de Aprendizado

- ☛ Refinamento de Habilidades vs. Aquisição do Conhecimento
- ☛ Aprendizagem por Memorização
- ☛ Aprendizagem através de Conselhos
- ☛ Aprendizagem através da Solução de Problemas
- ☛ Aprendizagem através de Exemplos

310



Aprendizagem por Memorização

- Armazenamento simples de valores
- Armazenamento e recuperação são mais baratos que o cálculo
- O armazenamento otimiza cálculos futuros
- Objetiva melhorar o desempenho em sistemas de IA

312

Aprendizagem por Aconselhamento

- Assim como pessoas, programas também podem aprender através de conselhos dados por seres humanos
- Os conselhos precisam ser codificados para que possam ser interpretados pelo sistema
- Novos conselhos podem ser utilizados pelo sistema com o objetivo de melhorar seu desempenho

313

Aprendizado por Exemplos: Indução

- Na estratégia de aprendizado por indução, o sistema adquire os conceitos através de inferências indutivas realizadas sobre fatos fornecidos ou observados.

314

Indução & Dedução

☛ Exemplo Dedução:

- todos os homens são fortes
- **Se** Pedro é homem **Então** Pedro é forte

☛ Exemplo de Indução:

- A maioria dos homens é forte
- **Se** Pedro é homem **Então** Pedro é forte

315

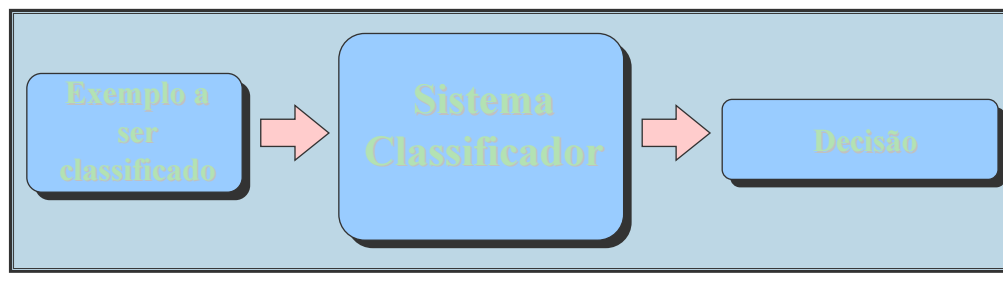
Aprendizado por Indução: Classificação

- ☛ O objetivo é associar a cada exemplo, ou observação, uma classe à qual ele pertence
- ☛ Os conceitos construídos estão representados na forma de um classificador

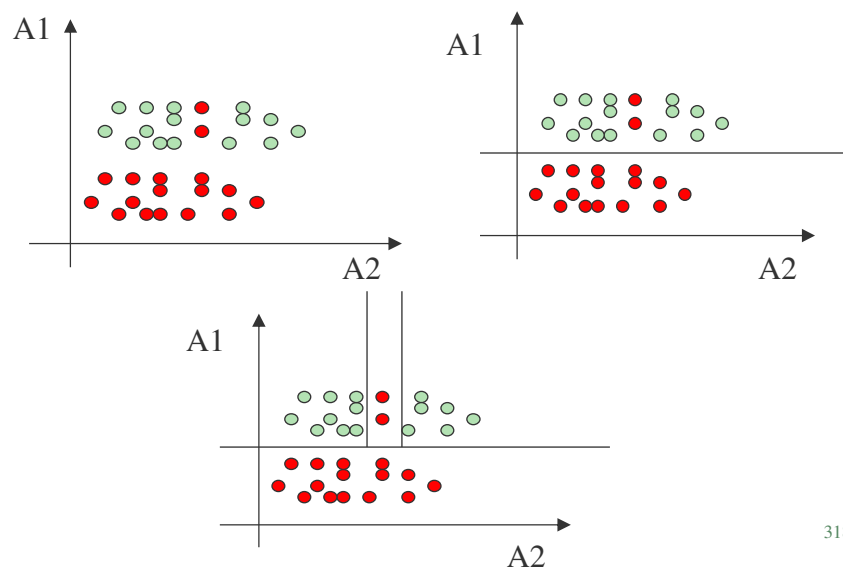


Aprendizado por Indução: Classificação (Cont.)

- O exemplo a ser classificado é submetido aos conceitos adquiridos, e uma decisão sobre a sua classe é devolvida pelo classificador



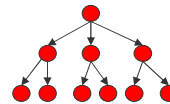
Problemas com Classificação



318

Aprendizado - Árvores de Decisão

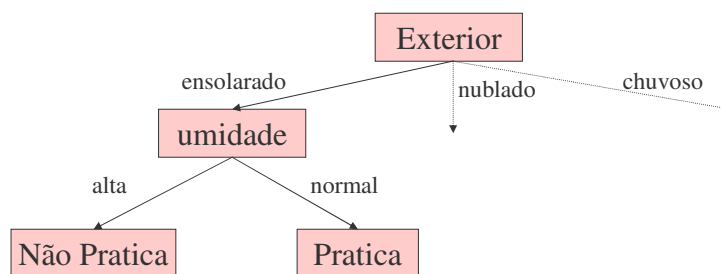
- Representação



- formada por nós folha que representam as classes de cada exemplo contido na folha
- formada por nós internos que representam condições sobre os valores de um determinado atributo
- formada por ligações entre nós que representam os valores possíveis para um determinado atributo

319

•Aprendizado - Árvores de Decisão (Cont.)



- Algoritmo de Construção

- Entrada: Conjunto de Exemplos E
- Saída: Árvore de Decisão T

320

Aprendizado - Árvores de Decisão

(Cont.)

$E = \{E_1, \dots, E_n\}$ = Conjunto de Exemplos

{ Inicialmente todos os exemplos pertencem a raiz }

Se todas as tuplas pertencem a uma mesma classe **então** parar

senão

 escolha um atributo;

 divida o conjunto de tuplas em subconjuntos, de acordo com cada valor do atributo escolhido;

 repita o procedimento recursivamente em cada subconjunto criado;

fim_senão

fim_se

321

Aprendizado - Árvores de Decisão

(Cont.)

☛ Escolha do Atributo

. Entropia

- Entropia é uma grandeza da teoria da informação que mede a desordem de uma determinada coleção de exemplos
- Quanto maior a entropia, maior a desordem dos exemplos

322

Aprendizado - Árvores de Decisão

(Cont.)

☛ Fórmula da Entropia

$$E(S) = - \sum_{i=1}^k p(i) \log_2 p(i)$$

onde:

- S é um conjunto de exemplos
- i é uma determinada classe
- $p(i)$ é a probabilidade de ocorrência da classe i no conjunto S
- k é o número de classes

323

Aprendizado - Árvores de Decisão

(Cont.)

☛ Entropia de um atributo

$$E(S, A) = - \sum_{i=1}^v \frac{|S_i|}{S_n} E(S_i)$$

onde:

- A é um determinado atributo
- i é um determinado valor para o atributo A
- S_i é o subconjunto de exemplos que possuem valor i para o atributo A
- S_i é o número de exemplos de S_i
- v é o número de valores para o atributo A

324

Aprendizado - Árvores de Decisão

(Cont.)

- Escolha baseada no ganho de informação

$$\text{ganho}(S, A) = E(S) - E(S, A)$$

- Informação da Divisão

$$\text{Inf} (S, A) = - \sum_{i=1}^v \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

325

Aprendizado - Árvores de Decisão

(Cont.)

- Escolha baseada no ganho proporcional de informação

$$\text{ganho relativo}(S, A) = \frac{\text{ganho}(S, A)}{\text{inf}(S, A)}$$

- Melhor atributo é aquele que maximiza o ganho proporcional

326

Aprendizado - Árvores de Decisão

(Cont.)

Exemplo

- Dado um conjunto de exemplos, verificar se um determinado dia está próprio para prática de esportes

327

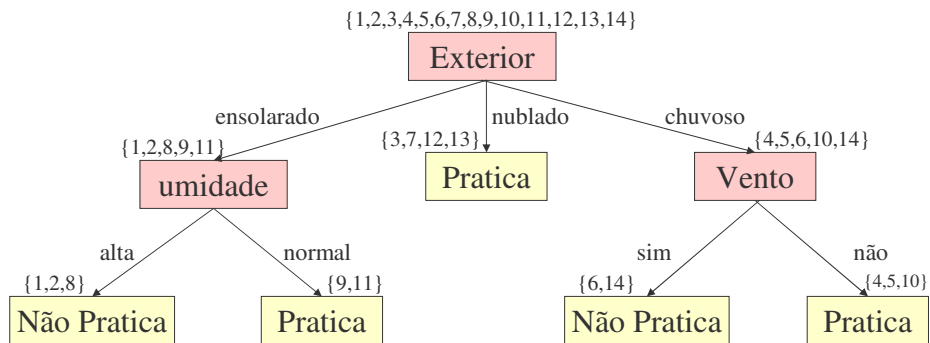
N	Atributos (A)				Classe
	Exterior	Temperatura	Umidade	Vento	(C)
1	ensolarado	alta	alta	não	np
2	ensolarado	alta	alta	sim	np
3	nublado	alta	alta	não	pe
4	chuvoso	média	alta	não	pe
5	chuvoso	baixa	normal	não	pe
6	chuvoso	baixa	normal	sim	np
7	nublado	baixa	normal	sim	pe
8	ensolarado	média	alta	não	np
9	ensolarado	baixa	normal	não	pe
10	chuvoso	média	normal	não	pe
11	ensolarado	média	normal	sim	pe
12	nublado	média	alta	sim	pe
13	nublado	alta	normal	não	pe
14	chuvoso	média	alta	sim	np

328

Aprendizado - Árvores de Decisão

(Cont.)

- Árvore de Decisão criada

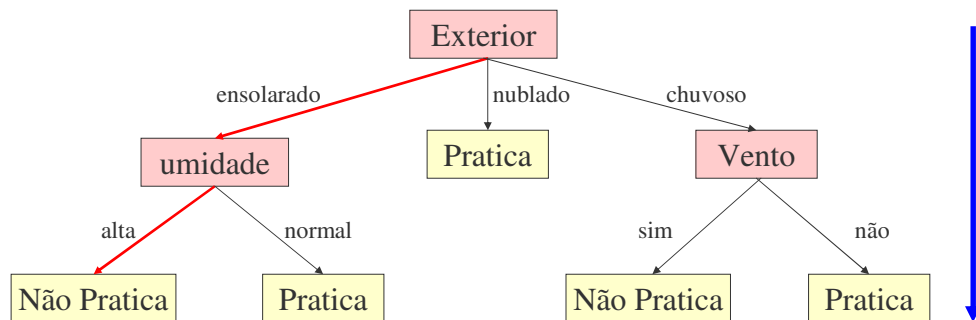


329

Aprendizado - Árvores de Decisão

(Cont.)

- Classificação do exemplo <ensolarado média alta não>



330

Aprendizado - Árvores de Decisão

(Cont.)

🌿 Regras de produção geradas

1. **Se** exterior = ensolarado
 e umidade = alta
 Então classe = np
2. **Se** exterior = ensolarado
 e umidade = normal
 Então classe = pe

331

Aprendizado - Árvores de Decisão

(Cont.)

🌿 Regras de produção geradas

3. **Se** exterior = nublado
 Então classe = pe
4. **Se** exterior = chuvoso
 e vento = sim
 Então classe = np

332

Aprendizado - Árvores de Decisão

(Cont.)

• Regras de produção geradas

5. **Se** exterior = chuvoso

e vento = não

Então classe = pe

333

Aprendizado - Árvores de Decisão

(Cont.)

• Principais problemas encontrados

- tamanho da árvore
- superadequação aos dados
- presença de informações inconsistentes

• Possíveis estratégias para a solução dos problemas encontrados

- poda (pré-poda e pós-poda)
- utilização de conceitos da Lógica Paraconsistente para manipulação adequada de dados inconsistentes

334

Aprendizado - Árvores de Decisão (Cont.)

- Vantagens
 - Podem adquirir e representar facilmente conceitos disjuntos
 - Podem manipular dados com ruídos
 - Generalizam conceitos a partir de poucos atributos
 - Possuem representação simples e rápida classificação
 - Têm obtido muito sucesso em diversas áreas de aplicação — vários sistemas estão disponíveis comercialmente

335

Aprendizado - Árvores de Decisão (Cont.)

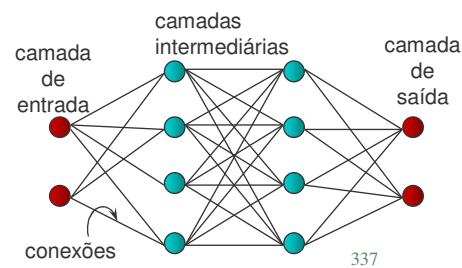
- Desvantagens
 - Vários conceitos para representar uma mesma classe podem ser criados
 - A maioria dos sistemas de árvore de decisão não implementam *backtraking* (retrocesso) na escolha de um atributo (sistemas determinísticos)
 - Poda não adequada pode generalizar conceitos de forma excessiva

336

Outros métodos de Aprendizado por Exemplos

🧠 Redes Neurais

- também conhecidas como o modelos conexionistas
- são redes interconectadas formadas por elementos computacionais muito simples
- baseadas no modelo de funcionamento do cérebro



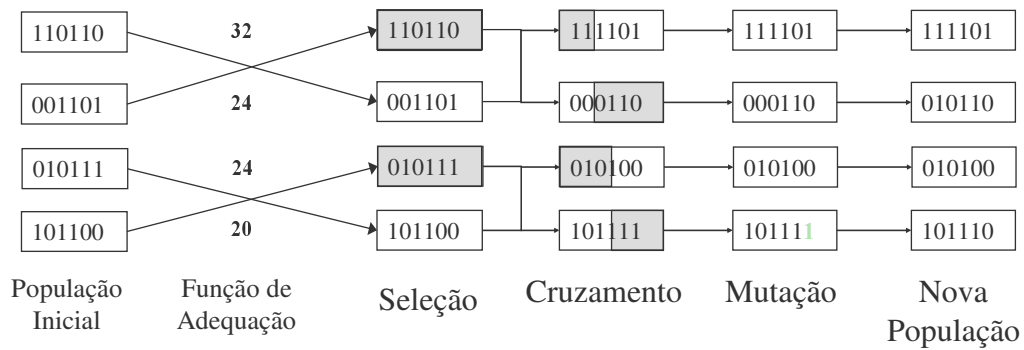
Outros métodos de Aprendizado por Exemplos

🧠 Algoritmos Genéticos

- tiveram origem nos estudos realizados sobre autômatos celulares (Holland)
- baseado no mecanismo de seleção natural e não no processo de raciocínio simulado
- procura gerar novas descrições, a partir de descrições iniciais, que possuam maior qualidade, criando novas gerações de descrições

338

Algoritmos Genéticos



339

Planejamento

Capítulo 6

340

Planejamento

- Para muitos problemas, a noção de planejamento é a mesma de *busca por solução*
- Exemplo: 8-puzzle
- Outros domínios de problemas fazem clara distinção entre busca e planejamento:
 - aqueles cujo universo não é previsível
 - quando passos não podem ser desfeitos (não é possível backtracking).

341

Navegação de Robôs

- O objetivo é movimentar um robô de uma sala para outra sem bater em obstáculos.
 - O que fazer se uma cadeira é movimentada?
 - O que fazer se uma criança salta em frente ao computador?
 - O que fazer se uma pessoa tenta parar o robô e o robô é forçado a matá-la, e o corpo está bloqueando a porta?

342

Planejamento & Decomposição

- 🤖 Problemas que são decomponíveis ou *parcialmente decomponíveis* podem sempre ser resolvidos por um sistema que:
 - Possa resolver cada subproblema de forma independente
 - Saiba quais subproblemas não são independentes
 - Possa revisar a solução dos subproblemas quando algo não esta de acordo com o *plano*.
- 🤖 Esta técnica é chamada *planejamento*

343

Exemplo de Planejamento

- 🤖 Robô desenvolve um *plano* para ir a outra sala.
O plano envolve:
 - Movimentação em torno do sofá
 - Movimentação do sofá para a porta
 - Abertura da porta
 - Movimentação através da porta

344

Representação do Estado

- Este tipo de problema sempre inclui descrições de estado muito complexas
- Pode não ser possível criar uma cópia *completa* de toda informação para cada estado considerado durante a busca
- O estado sucessor pode ser descrito pelas mudanças que são feitas por um operador
- É necessário armazenar o caminho dessas mudanças para permitir backtracking e evitar loops.

345

Espaço de Estados do Robô

- Localização de todos os obstáculos, paredes, portas, salas, etc.
- Localização do robô
- Velocidade e direção do movimento do robô
- Posição de todos os periféricos robóticos (braços, sensores, câmeras).

346

Como resolver problemas?

- Nós criamos um plano que inclui as soluções a um determinado número de subproblemas
- Depois de executar a solução para o subproblema 1, algo vai mudar
- É necessário saber quais outros subproblemas serão afetados, e revisar cada uma de suas soluções
- A chave é que nós não queremos recomeçar do nada

347

Planejando um conjunto de soluções

- Algumas vezes é possível criar um número de planos, cada um pode trabalhar com obstáculos específicos que podem surgir
- Quando algo dá errado, nós já temos um plano de contingência pronto
- Nem sempre isto é possível ou prático (o conhecimento necessário não está disponível ou é muito grande).

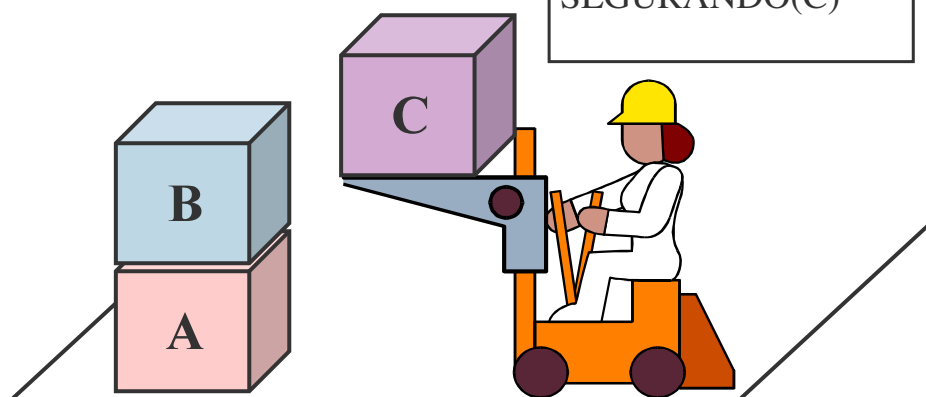
348

Dependências entre Subtarefas

- 👉 O plano inclui informação que descrevem as dependências entre todas as subtarefas
- 👉 A utilização desta informação para revisar planos é chamada de *backtraking dirigido a dependência*
- 👉 É sempre mais fácil determinar estas dependências se o processo de planeamento trabalha a partir dos estados meta anteriores

349

Mundo dos Blocos



350






Mundo dos Blocos (Cont.)

Predicados usados para descrever o estado:

SOBRE(A,B)	Bloco A está sobre B
SOBRE_MESA(A)	Bloco A está sobre a mesa
LIVRE(A)	Hão há nada sobre A
SEGURANDO(A)	O robô está segurando A
BRAÇOVAZIO	O braço do robô está vazio

351

Operações no Mundo dos Blocos

 <i>retira(A,B)</i>	retira A de cima de B
 <i>coloca(A,B)</i>	coloca A sobre B
 <i>levanta(A)</i>	levanta o bloco A.
 <i>abaixa(A)</i>	coloca A sobre a mesa
 <i>larga(A)</i>	larga A.

352

Exemplo de Restrições de Operação

retira(A,B):

BRAÇOVAZIO	o braço deve estar vazio
LIVRE(A)	nada sobre A
SOBRE(A,B)	A deve estar sobre B

353

Alguns Axiomas do Mundo dos Blocos

$$[\forall x : \text{SEGURANDO}(x)] \Rightarrow \neg \text{BRAÇOVAZIO}$$
$$\forall x : \text{SOBRE_MESA}(x) \Rightarrow \neg \forall y : \text{SOBRE}(x,y)$$
$$\forall x : [\neg \forall y : \text{SOBRE}(y,x)] \Rightarrow \text{LIVRE}(x)$$

354

Raciocinando no Mundo dos Blocos

- Representação do estado é um conjunto de predicados que são verdadeiros para aquele estado
- Cada predicado se refere a um estado do objeto:

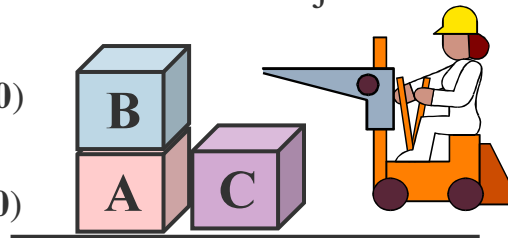
SOBRE(B,A,estado0)

SOBRE_MESA(A, estado0)

LIVRE(B, estado0)

SOBRE_MESA(C, estado0)

LIVRE(C, estado0)



355

Mundo dos Blocos & Resolução

- O efeito de cada operador é codificado em lógica de predicados:

$$\begin{aligned} \text{LIVRE}(x,s) \wedge \text{SOBRE}(x,y,s) \Rightarrow \\ \text{SEGURANDO}(x, \text{Faça}(\text{retire}(x,y),s)) \wedge \\ \text{LIVRE}(y, \text{Faça}(\text{retire}(x,y),s)) \end{aligned}$$

Faça é uma função que especifica o estado resultante da aplicação de um operador

356

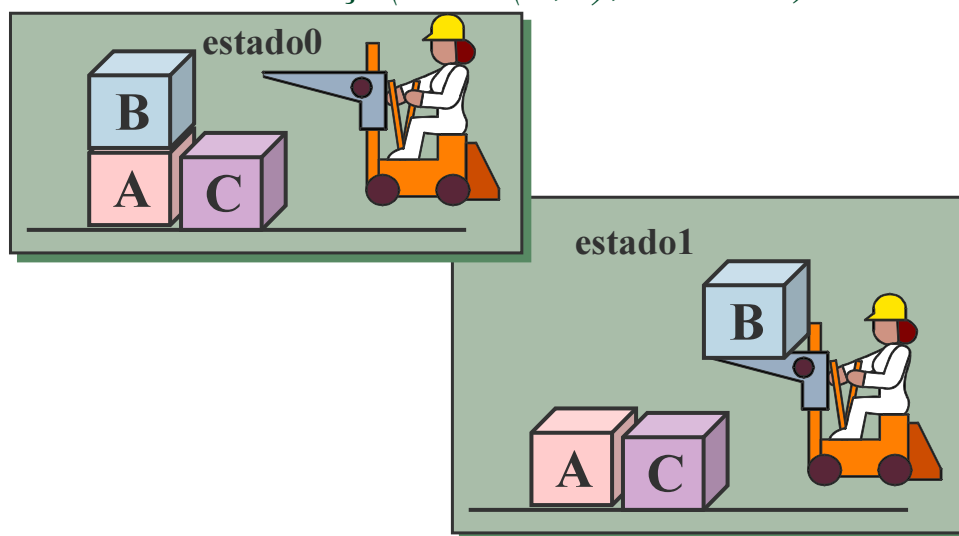
<p><u>Estado0:</u></p> <p>SOBRE(B,A,estado0)</p> <p>SOBRE_MESA(A, estado0)</p> <p>LIVRE(B, estado0)</p> <p>SOBRE_MESA(C, estado0)</p> <p>LIVRE(C, estado0)</p>	<p><u>Operador Retire</u></p> <p>$LIVRE(x,s) \wedge SOBRE(x,y,s) \Rightarrow$</p> <p>$SEGURANDO(x, Fa\c{c}a(retire(x,y),s))$</p> <p>$\wedge$</p> <p>$LIVRE(y, Fa\c{c}a(retire(x,y),s))$</p>
--	---

estado1 = $Fa\c{c}a(retire(B,A), estado0)$

$SEGURANDO(B, estado1) \wedge LIVRE(A, estado1)$

357

estado1 = $Fa\c{c}a(retire(B,A), estado0)$



358

SOBRE_MESA(A,estado1) é verdade ?

- O operador *retire* que produziu o estado1 não diz nada sobre o que foi modificado
- Nós precisamos adicionar algum *axioma* que especifique o que não mudou quando um operador foi aplicado

$$\text{SOBRE_MESA}(z,s) \Rightarrow \text{SOBRE_MESA}(z, \text{Faça}(\text{retire}(x,y),s))$$

359

Usando Resolução

- Em geral, nós precisamos adicionar muitos *axiomas* para ter certeza que a informação pode ser derivada
- Se nós especificamos todos os axiomas necessários, Resolução é o único mecanismo necessário
- Nem sempre é possível especificar todos os axiomas necessários

360

Operadores Híbridos

Para cada Operador:

precondições: predicados que devem ser verdadeiros antes que o operador possa ser aplicado (restrições).

adições: predicados que o operador torna verdadeiros

deleções: predicados que o operador torna falsos

Qualquer coisa não especificada é assumida ser não modificada

361

coloca(x,y)

Precondições: $LIVRE(y) \wedge SEGURANDO(x)$

Adições: $BRAÇOVAZIO \wedge SOBRE(x,y)$

Deleções: $LIVRE(y) \wedge SEGURANDO(x)$

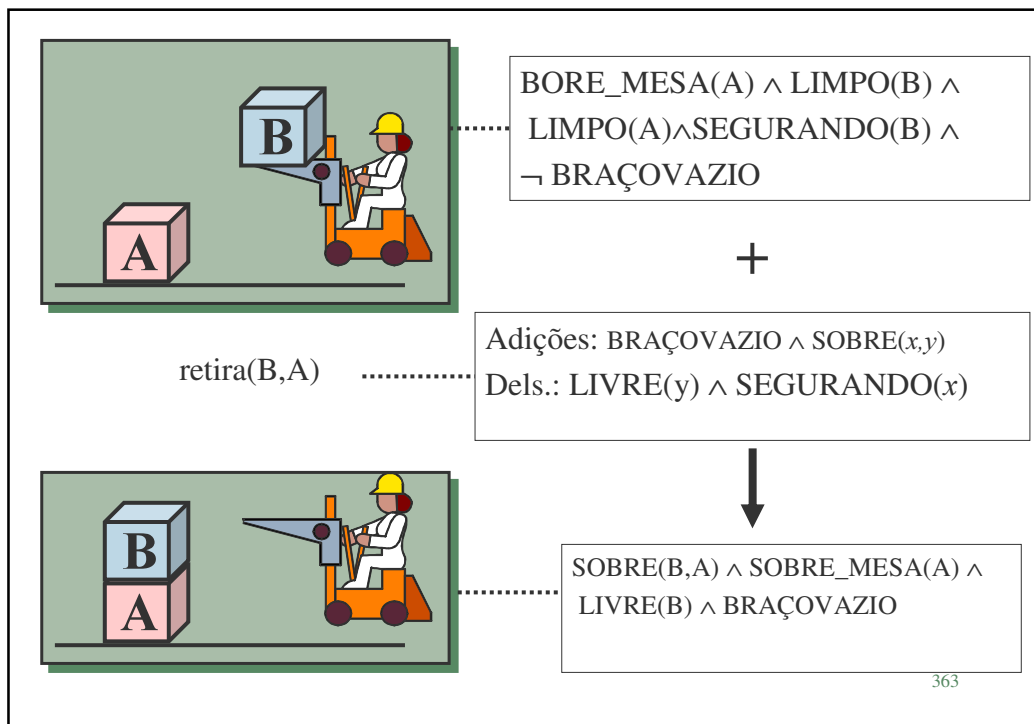
retira(x,y)

Precondições:

Adições:

Deleções:

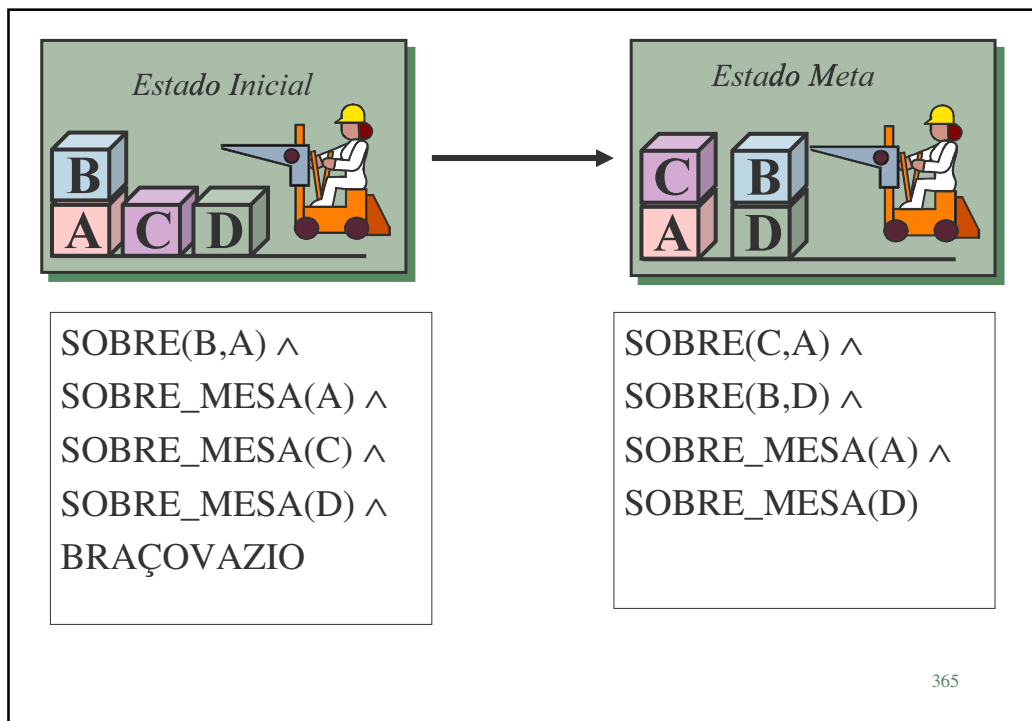
362



Planejando Metas Empilhadas

- Divide o problema em subproblemas
- Para cada subproblema:
 - Coloque a descrição da submeta em uma pilha
- Enquanto não realizada
 - obtenha a submeta da pilha
 - Encontre a sequência de operadores que alcançarão a submeta. Aplique operadores no corrente estado

364



Pilha de Metas

1. $\text{SOBRE}(\text{C}, \text{A}) \wedge \text{SOBRE}(\text{B}, \text{D}) \wedge \text{SOBRE_MESA}(\text{A}) \wedge \text{SOBRE_MESA}(\text{D})$

Obtenha a meta da pilha, divida-a em submetas, coloque a meta original na pilha, seguida por cada meta insatisfeita

1. $\text{SOBRE}(\text{C}, \text{A})$
 2. $\text{SOBRE}(\text{B}, \text{D})$
 3. $\text{SOBRE}(\text{C}, \text{A}) \wedge \text{SOBRE}(\text{B}, \text{D}) \wedge \text{SOBRE_MESA}(\text{A}) \wedge \text{SOBRE_MESA}(\text{D})$

Submeta: SOBRE(C,A)

- Examine operadores para uma submeta que tenha um predicado SOBRE na lista *Adds*
- Encontre aquele *coloca*(C,A) que realiza a tarefa
- Coloque *coloca*(C,A) na pilha de metas
 - Não precisa colocar SOBRE(C,A) na pilha, uma vez que *coloca*(C,A) torna SOBRE(C,A) verdade

367

Submeta: *coloca*(C,A)

- *coloca*(C,A) tem precondições, coloque-as na pilha de metas

1. LIVRE(A) \wedge SEGURANDO(C)
2. *coloca*(C,A)
2. SOBRE(B,D)
3. SOBRE(C,A) \wedge SOBRE(B,D) \wedge SOBRE_MESA(A) \wedge SOBRE_MESA(D)

368

Submeta: $LIVRE(A) \wedge SEGURANDO(C)$

☞ Separe em 2 submetas e coloque-as na pilha. Neste caso é claro que a ordem pode ser muito importante!

1. $LIVRE(A)$
2. $SEGURANDO(C)$
3. $LIVRE(A) \wedge SEGURANDO(C)$
2. $coloca(C,A)$
2. $SOBRE(B,D)$
3. $SOBRE(C,A) \wedge SOBRE(B,D) \wedge SOBRE_MESA(A) \wedge SOBRE_MESA(D)$

369

Submeta: $LIVRE(A)$

- ☞ $LIVRE(A)$ não é verdade neste momento, mas $retira(B,A)$ a tornará verdade
- ☞ Coloque $retira(B,A)$ na pilha no lugar de $LIVRE(A)$
- ☞ $retira(B,A)$ tem precondições:
 - $SOBRE(B,A) \wedge LIVRE(B) \wedge BRAÇOVAZIO$

370

Backtracking

- A Divisão de uma meta em submetas e a colocação de cada uma delas em uma pilha é uma forma de planejamento
- Quando a solução para uma submeta aponta para um próximo passo, backtracking é necessário
- Backtracking não é feito explicitamente, mas implicado por processos
 - cada submeta tomada da pilha é resolvida em relação ao estado corrente

371

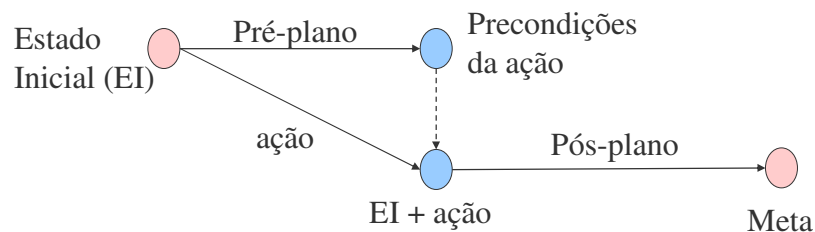
A Solução

- A solução é a seqüência de operadores que foram aplicados
- Uma vez que nós sabemos o que cada operador faz (das *adições* e *deleções*) é possível reduzir uma seqüência de operadores
- Exemplo: coloca(A,B), retira(A,B) => { }

372

Análise Meios-Fins em Planejamento

- Encontre uma ação útil (que reduza a diferença com a meta)



- Crie dois novos subproblemas:

- tornar verdadeiras as precondições da ação a partir de EI
- encontrar a meta a partir do resultado da ação sobre EI

373

Análise Meios-Fins – Algoritmo GPS

GPS(estado-inicial, meta)

- Se $meta \subseteq estado-inicial$, então retorne *true*
- Selecione uma diferença d entre *meta* e o *estado-inicial*
- Selecione um operador O que reduz a diferença d
- Se nenhum outro operador então *falhe*
- $Estado = \mathbf{GPS}(estado-inicial, precondições(O))$
- Se Estado, então retorne $\mathbf{GPS}(\mathbf{apply}(O, estado-inicial), meta)$

374

Resolvendo o mundo dos blocos com Meios-Fins

```

meta([livre(a),sobre(a,b),sobre(b,c),
      sobre(c,d),sobre_a_mesa(d),braco_vazio])).

operador(pegar,
          [livre(X),sobre_a_mesa(X),braco_vazio],
          [sobre_a_mesa(X),braco_vazio],
          [segurando(X)],true).

atual([livre(d),sobre(b,a),sobre(c,b),sobre(d,c),
      sobre_a_mesa(a),braco_vazio])).

operador(largar,
          [segurando(X)],
          [segurando(X)],
          [sobre_a_mesa(X),braco_vazio],true).

operador(desempilhar,
          [sobre(X,Y),livre(X), braco_vazio],
          [sobre(X,Y),braco_vazio],
          [segurando(X),livre(Y)],(X\=Y)).

operador(empilhar,
          [livre(Y),segurando(X)],
          [livre(Y),segurando(X)],
          [braco_vazio,sobre(X,Y)],(X\=Y)).

```

375

Resolvendo o mundo dos blocos com Meios-Fins

```

seleciona_operador1(Diferenca,Atual,Meta) :-
    Diferenca \= [],
    operador(O,_,_,_), % Encontra um operador
    faca(O,Atual,S,P), % Executa o operador sobre atual
    nao_visitado([O,S]), % Verifica se estado não foi visitado
    diferenca(S,Diferenca,L),length(L,X),length(Diferenca,Y), X <= Y,
    \+ go_back_action([O,S]),
    insert_last_action([O,Atual]),
    write([O,S]),nl,
    meios_fins(Atual,P),
    meios_fins(S,Meta).

```

Verifica se a diferença diminuiu

Verifica se ação não irá desfazer a ação anterior

376

Planejamento Não-Linear

- Algumas vezes é necessário atacar múltiplas submetas ao mesmo tempo
- O desenvolvimento de um plano que pode fazer isto é chamado *planejamento não-linear*
- O uso de listas de precondição, adição e deleção torna possível determinar o efeito de intercalar soluções (operadores).

377

Planejamento Hierárquico

1. Tente estabelecer, primeiramente, um plano geral
2. Refine cada um dos passos do plano
3. Refine cada refinamento (e assim sucessivamente...)

378

Planejamento Hierárquico (Cont.)

Duas Técnicas:

- 🐼 Macro Operadores - constrói operadores complexos além dos pequenos operadores
- 🐼 Associa pesos a precondições. Inicialmente procure apenas satisfazer precondições importantes (mas resolva o problema inteiro). Então se preocupe com precondições menos importantes

379

Sistemas Reativos

- 🐼 Desenvolva uma submeta, e entre em ação para alcançar a submeta
- 🐼 Dê uma olhada no resultado, invente uma nova submeta
- 🐼 Continue indo até alcançar a meta

380

Sistemas Reativos (Cont.)

- Trabalha bem para muitos tipos de tarefas simples.
- Frequentemente uma combinação de reação e planejamento é requerida:
 - Não pode esperar pelo melhor plano antes de tomar alguma em ação
 - Tomada de uma ação realmente ruim pode conduzir a desastre

381

Arquitetura de um sistema de diálogo com um Agente Assistente



382

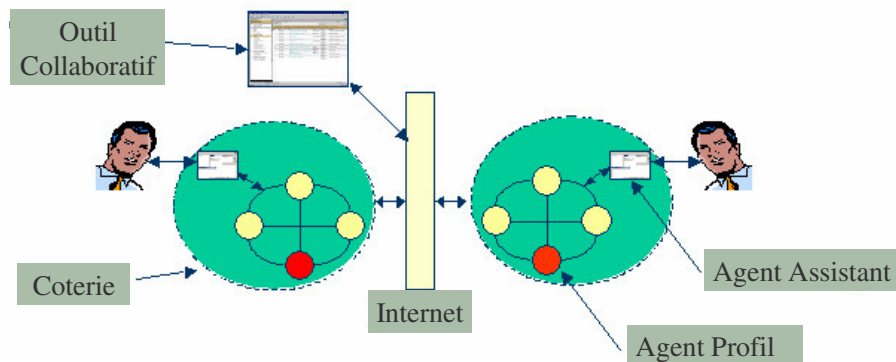
Contexto

- 👤 Agentes assistentes pessoais
- 👤 Interfaces em Linguagem Natural
- 👤 Utilização da técnica Morfológica
- 👤 Sobre sistemas de diálogo
- 👤 Considerações

383

O projeto AACC*

- 👤 Melhorar a interação entre grupos de estudantes franceses e americanos



*Agentes para o auxílio à concepção colaborativa

384

Agentes Assistentes Pessoais

🤖 Softbots (Etzioni)

- Serviços Web

🤖 Agentes interface (Lieberman)

- Auxílio à utilização de diferentes sistemas
- Ferramentas de uso corrente (emails, editores, navegação, pesquisa de informação, personalização de produtos)

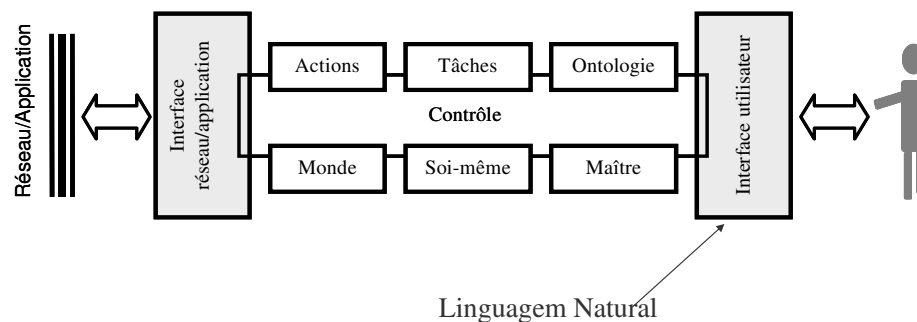
🤖 Agentes de informação (CIA) (8^{ème})

- Distribuição de Informação e serviços (Computação pessoal, Wireless Computing)

385

Agentes Assistentes Pessoais (cont.)

🤖 Modelo de um Agente Assistente Pessoal (Ramos)



386

Interfaces em Linguagem Natural

👤 Motivações

- Permitir ao usuário interrogar o assistente e solicitar informações e serviços
- Tornar a comunicação mais natural e simples
- Diminuir a carga cognitiva sobre o usuário

387

Objetivos

- 👤 Melhorar a interação entre homem e computador através de um meio mais intuitivo de comunicação



388

Técnicas Correntes

🐼 1- Palavras-chave

- Simples a desenvolver
- Solicitações de complexidade limitada

🐼 2- Semântica (KL-ONE, FRL-0, Gr. Conceituais, DMAP, Dependência Conceitual, etc.)

- Permite a interrogação e interpretação de frases complexas
- Utiliza uma linguagem de descrição (complexa)
- Todas as interpretações semânticas devem ser fornecidas antes que o sistema seja executado (domínio fixo)
- Sintaxe e semântica (conhecimento) podem estar misturados

389

Técnicas Correntes

🐼 3- Morfológica

- A sintaxe é separada da semântica
- A sintaxe é fornecida por uma gramática
- A semântica (conhecimentos) é fornecida por uma hierarquia de conceitos (ontologia)
- O conhecimento do sistema pode ser atualizados mais facilmente (mudanças na ontologia)

390

Interação em Linguagem Natural

☛ Tipos de Interação

- Sistemas de questão/resposta
 - O usuário solicita informações sobre objetos
 - O sistema deve pesquisar uma base de objetos e retornar uma solução
- Sistemas de Diálogo
 - Sequência de interações onde o sistema deve adquirir informações do usuário, responder questões e executar ordens

391

Interação em Linguagem Natural (cont.)

☛ Por palavras-chave

- Filtragem e reconhecimento de tokens

☛ Semântica

- Linguagem de representação complexa

☛ Morfológica

- Análise léxica, sintática, e semântica

392

Aplicações baseadas em Palavras-chave

Ex.: “Encontre páginas **em Português** que contém informações **sobre Pelé**”

Algoritmo

Entrada: *Dicionário do Domínio, Frase*

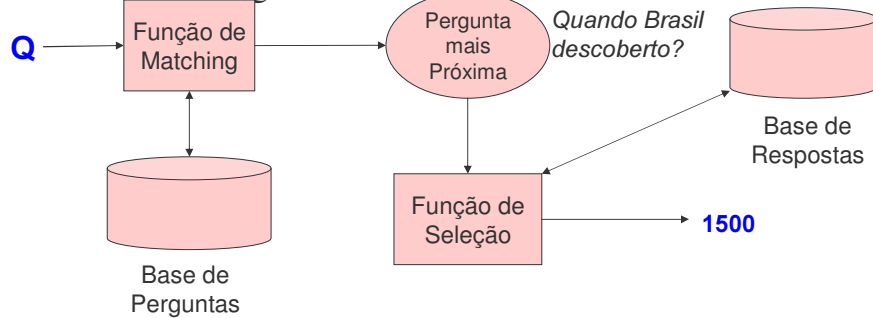
- Eliminar stop words
- Encontrar termos da frase presentes no dicionário
- Disparar um motor de pesquisa

393

Interação baseada em Palavras-chave

O sistema deve pesquisar a pergunta do usuário dentro de uma base de perguntas e outra de respostas previamente modeladas

Ex.: **Q** = “Quando o Brasil foi descoberto?”



394

Interação baseada em Palavras-chave

- Medidas de matching simples para representação textual
 - Frequência de palavras
 - Frequência ponderada
- Medidas de matching simples para representação vetorial
 - Distância Euclidiana
 - Média ponderada
 - TF-IDF (Term Frequency - Inverse Document Frequency)

395

The screenshot shows the PRECISE web interface in a Microsoft Internet Explorer browser window. The page title is "PRECISE: A Reliable Natural Language Interface To Databases". The interface is divided into a left sidebar with navigation links and a main content area titled "PRECISE Demonstration".

Left Sidebar:

- General:** Reliable NLI Home, Precise Demo
- Faculty:** Oren Etzioni, Henry Kautz, Dan Weld
- Students:** Ana-Marie Popescu, Alexander Yates
- UW Natural Language Community:** Mari Ostendorf (EE Dept.), Speech Group (EE Dept.), Wanda Pratt (Information School), John Goldsmith (Linguistics Dept.), Eric Brill (Microsoft Research)

Main Content Area:

PRECISE Demonstration

Please select a database: **ATIS**

Enter your query for the database here:

User Query:
How long does a flight from Baltimore to San Francisco take?

Result:
(405) (414) (416) (417) (421) (425) (428) (435) (437) (438) (442) (444) (445) (448) (450) (456) (460) (468) (470) (475) (487) (502) (509) (573)

SQL query:
SELECT DISTINCT f1.time_elapsed FROM city c4, flight f2, airport_service as6, airport_service as7, flight f3, flight f1, city c5 WHERE f1.flight_id = f2.flight_id AND f3.from_airport = as6.airport_code AND as6.city_code = c4.city_code AND c4.city_name = 'BALTIMORE' AND f3.to_airport = as7.airport_code AND as7.city_code = c5.city_code AND c5.city_name = 'SAN FRANCISCO' AND f2.flight_id = f3.flight_id

Logical Form:
how long(s0) ^ does(s0, s1) ^ flight(s1) ^ from airport(s2, s3) ^ baltimore(s3) ^ to airport(s2, s4) ^ san francisco(s4) ^ take(s1, s2)

Department of Computer Science & Engineering

The browser's taskbar at the bottom shows several open applications, including "2 Bloco de n...", "SWI-Prolog (...)", "PRECISE: A ...", "Folha Online - ...", "C:\Document...", "Personalizaçã...", "Aula6-Interfa...", and "FE-Journal-A...". The system clock shows 15:58.

Interação baseada em Palavras-chave

(cont.)

🐼 Vantagens

- Simplicidade
- Processamento rápido

🐼 Desvantagens

- Muito esforço de modelagem pois todas as questões devem ser previstas *apriori*
- Limitada à interpretação de questões simples

397

Sistemas de Questão/Resposta

🐼 Aplicações

- Interrogação a bases de dados em geral
- Solicitação de informações (turísticas, telefônicas, etc.)
- Recuperação de documentos baseada no conteúdo (e.g.: encontre todas as histórias sobre Pelé)
- Busca na Web

398

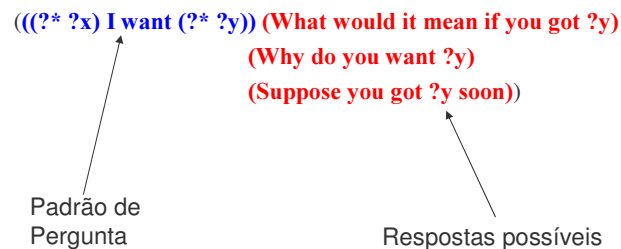
O sistema ELIZA

- Sistema desenvolvido por Joseph Weizenbaum no MIT e publicado em janeiro de 1966 na *Communications of the Association for Computing Machinery*
- Funcionamento
 - uma entrada, que consiste na leitura de uma frase via teclado ;
 - um processamento, envolvendo duas fases :
 - o **casamento de padrões**, que corresponde encontrar numa base de padrões um padrão que case com a frase de entrada ;
 - a transformação da frase de entrada, utilizando o resultado do passo (2), numa segunda frase, que corresponde a resposta à entrada (1) ;
 - uma saída, que consiste simplesmente na impressão da resposta

399

O sistema Eliza (cont.)

Exemplo de Regra



400

Interação baseada em Representações Semânticas

- Utiliza linguagens complexas para representar conhecimento
- Essas linguagens permitem a representação e inferência sobre conceitos complexos
- Exemplos: Rede Semântica, Frames, Grafos Conceituais, Dependência Conceitual

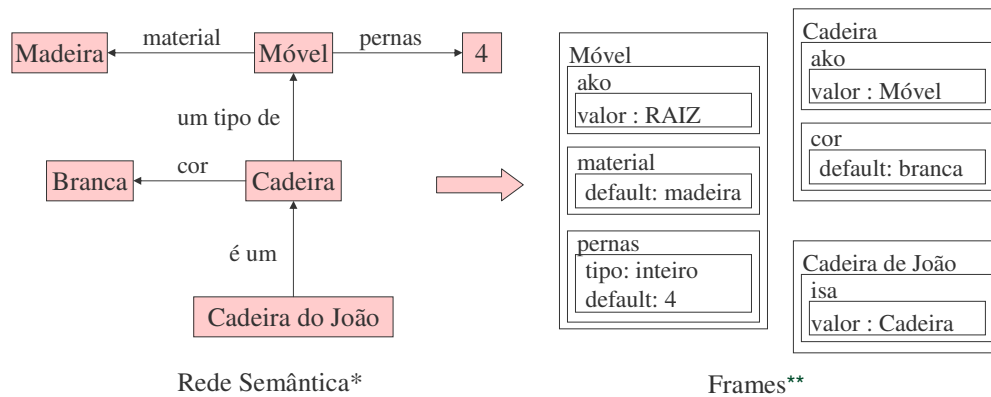
401

Interação em Linguagem Natural (cont.)

- Por palavras-chave
 - Filtragem e reconhecimento de tokens
- Semântica
 - Linguagem de representação complexa
- Morfológica (Artigo SBIA, Cap. III da tese)
 - Análise léxica, sintática, e semântica

402

Interação baseada em Representações Semânticas (cont.)



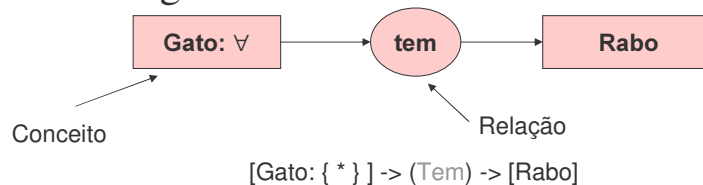
* Woods

**Marvin Minsky

403

Interação baseada em Representações Semânticas – Grafos Conceituais* (cont.)

🐱 “Todo gato tem rabo”



Conjunto de relações:
é um, tem, parte de, agente, loc, etc.

🐼 “Pedro mora em Curitiba”

$[Pessoa: Pedro] \leftarrow (Expr) \leftarrow [Mora] \rightarrow (Loc) \rightarrow [Cidade: Curitiba]$

*John Sowa

404

Interação baseada em Representações Semânticas (cont.)

• Outras formas de representação

- Dependências Conceituais*
- DMT* (Dynamic Modeling Theory)
- ...

• Utilização de Representações semânticas

- Construção de parsers semânticos

*Roger C. Schank

405

Interação baseada em Representações Semânticas (cont.)

• Vantagens

- Podemos representar conceitos, relações e idéias complexas

• Desvantagens

- Utiliza linguagens complexas
- Necessita um grande esforço de modelagem do domínio

406

Interação baseada em Representações Semânticas (cont.)

👉 Exemplos

- Avaliação do grau de compreensão de textos*
- Sumarização de textos e documentos **
- Sistemas de tradução multi-língua

* prof. Bráulio

** prof. Celso

407

A Técnica Morfológica

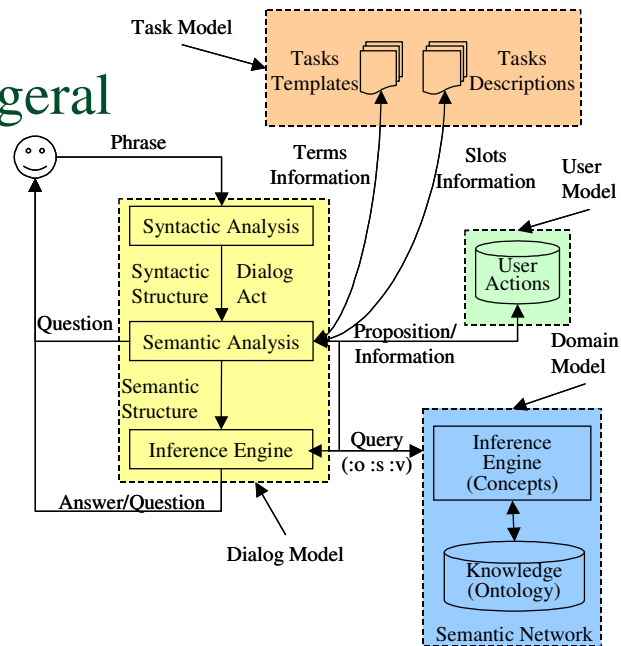
👉 Técnica Morfológica: (Gramática + Ontologia)

👉 Comunicação baseada em sistemas de diálogo

- Diálogo orientado a tarefas e questão/resposta
- Um motor de diálogo interpreta atos de diálogo

408

Arquitetura geral



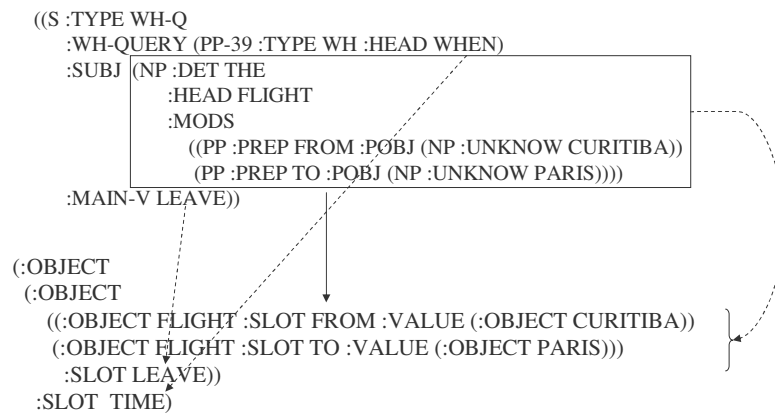
Análise Sintática

(s '(When does the flight from Curitiba to Paris leave))



```
((S :TYPE WH-Q
:WH-QUERY (PP-39 :TYPE WH :HEAD WHEN)
:SUBJ (NP :DET THE
:HEAD FLIGHT
:MODS
((PP :PREP FROM :POBJ (NP :UNKNOWN CURITIBA))
(P :PREP TO :POBJ (NP :UNKNOWN PARIS))))
:MAIN-V LEAVE))
```

Análise Semântica



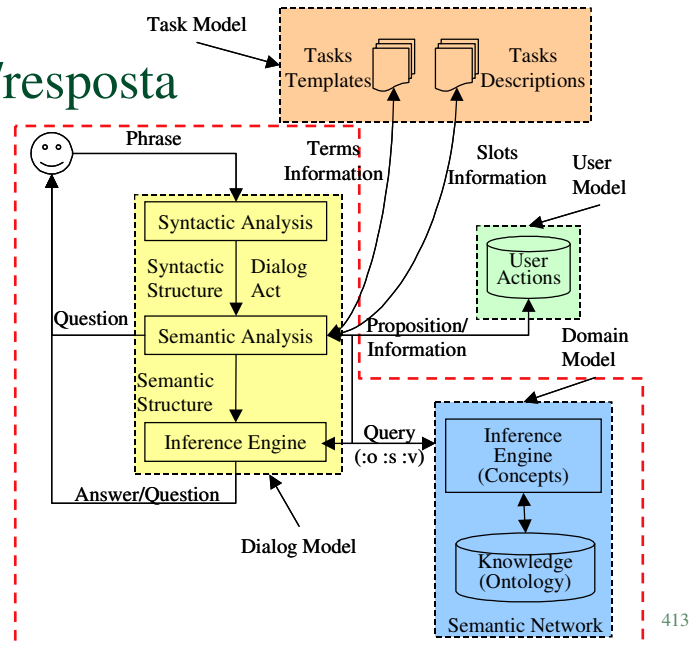
411

Motor de Inferência

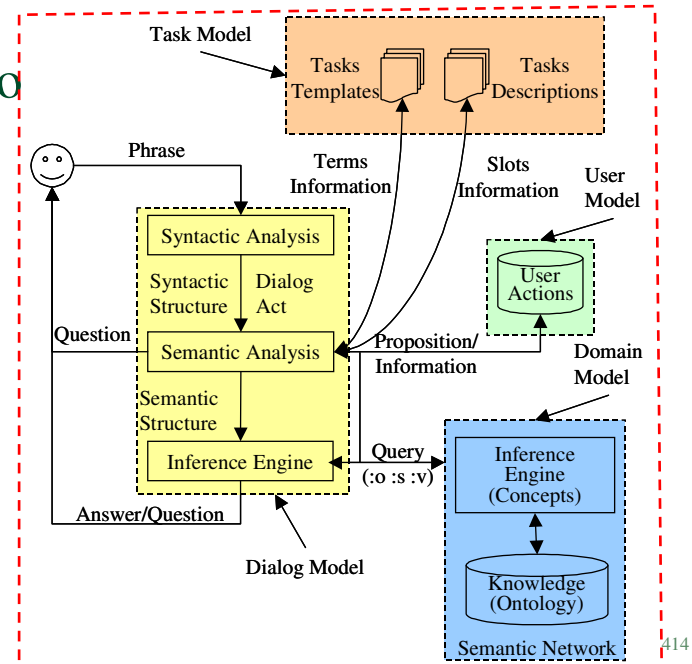
- A partir da representação semântica, procura na ontologia os valores e objetos solicitados
- A ontologia é representada na forma de uma rede semântica (MOSS)
- Cada tipo de enunciado possui uma semântica bem determinada

412

Diálogo questão/resposta



Diálogo orientado a tarefas



Diálogo orientado a tarefas

- 🐼 Serve a solicitar serviços
- 🐼 Informações terminológicas

Template de la tâche : “Recherche de document”

```
:verbs-synonyms (locate find search give look)
:nouns-synonyms (document documents paper file article)
author (author researcher)
year (year date when)
title (subject title theme)
subject (about)
```

415

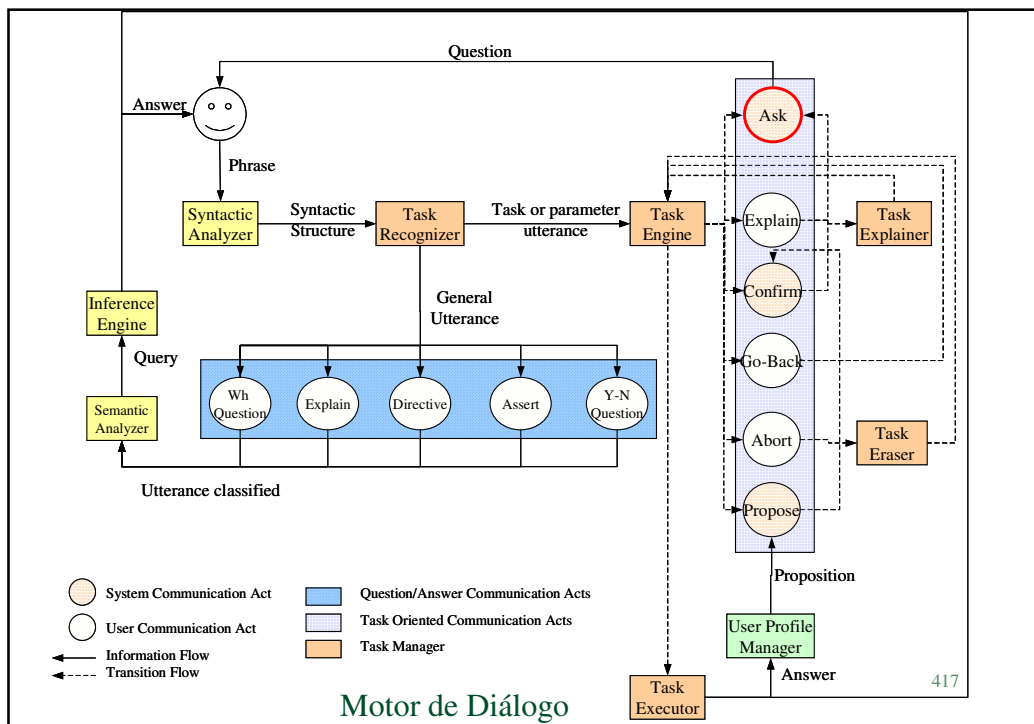
Diálogo orientado a tarefas_(cont.)

- 🐼 Informações Estruturais

Structure de la tâche : “Recherche de document”

```
:name 'find-document
:params '(author year title subject)
:params-values '(nil nil nil nil)
:semantic-value '(name nil nil nil)
:params-confirm '(nil nil nil nil)
:params-labels '("Who is the author of the document?"
                 "When the document was written?"
                 "What is the title of the Document?"
                 "What is the subject?")
:params-save '(t t t)
:params-explain '("The person who wrote the document."
                  "The parameter FROM represents who goes
                    to receive a Carbon Copy of the message."
                  "All the messages have a subject for identification."
                  "This is the body of the message, the text that will
                    be sent.")
:global-confirm "Confirm search?"
```

416



Explorando conhecimento com Linguagem Natural

- 1 User: Joe is a professor of Artificial Intelligence at UTC.
- 2 System: OK.
- 3 User: What does Joe do?
- 4 System: Joe is a professor of Artificial Intelligence at UTC.
- 5 User: What is Joe's profession?
- 6 System: Joe is a professor of Artificial Intelligence at UTC.
- 7 User: Where does Joe works?
- 8 System: At UTC
- 9 User: Is Joe a professor of Artificial Intelligence?
- 10 System: Yes
- 11 User: Is Joe a professor of artificial intelligence at UTC?
- 12 System: Yes
- 13 User: Is Joe a professor of artificial intelligence at PUC?
- 14 System: I don't know. None information about this.
- 15 User: What does Joe teaches?
- 16 System: Artificial Intelligence

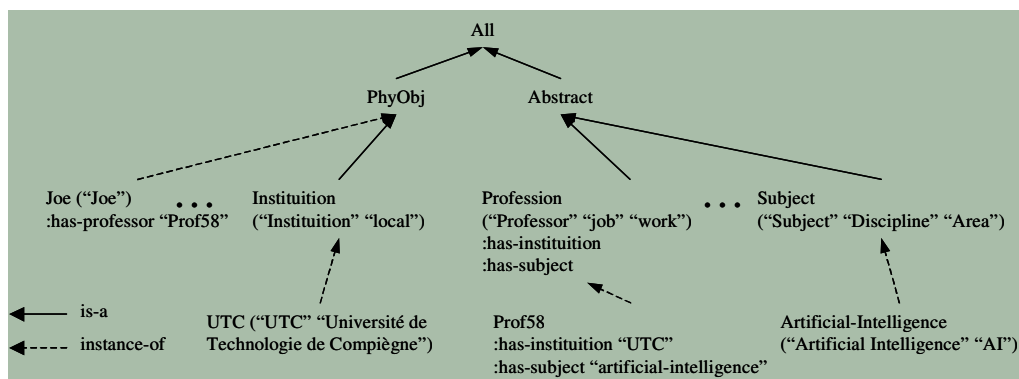
418

Explorando conhecimento com Linguagem Natural_(cont.)

1 User: Send a mail to Marco for me.
2 System: Who are the Carbon Copy receivers?
3 User: What is Leila's address?
4 System: The address of Leila is "25 rue de Paris".
5 System: Who are the Carbon Copy receivers?
6 User: none
7 System: What is the subject?
8 User: Ohh I'm sorry I made mistake!
9 System: Who are the Carbon Copy receivers?
10 User: Mary's husband.
11 System: What is the subject?
...

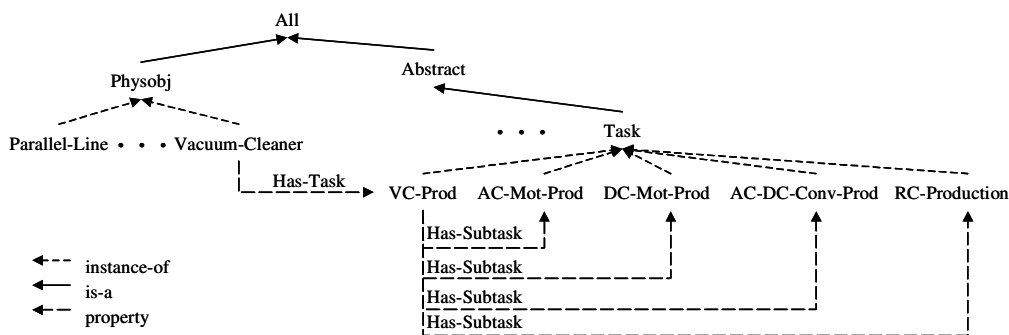
419

Base de conhecimentos gerais



420

Base de conhecimentos sobre mecânica



421

Questões específicas

- 1 User: How can I create a vacuum cleaner?
- 2 System: Firstly, a switch is needed to physically alternate the polarities from the power supply so the motor runs in a clockwise or CCW direction. If you use a Remote control, it must be... The production of an AC motor must...
- 3 User: Why must I create an ac motor?
- 4 System: The production of an AC motor for a suction cleaner needs take into account a relative speed, high torque for turning a fan unit and the belt drive.

422

Sobre sistemas de diálogo

- Atos de diálogo podem ser utilizados para construir sistemas capazes de responder a sentenças em linguagem natural
- Independência entre modelo de diálogo e conhecimentos do domínio
- Motor de diálogo genérico + estrutura explícita das tarefas torna o sistema escalável

423

Introdução à IAD



424

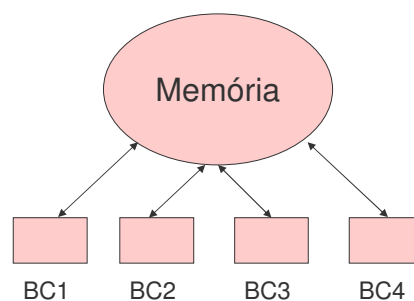
No Início...

- Desde os anos 50, pesquisadores estão interessados em construir sistemas inteligentes...
- Alguns pesquisadores como Erman e Lesser (1975), Hewitt (1977) e Smith (1979) apontaram para a necessidade de se fabricar entidades inteligentes distribuídas capazes de comunicar e resolver problemas que não podem ser facilmente resolvidos de forma centralizada

425

Trabalhos Pioneiros

- Erman e Lesser desenvolveram o ***Blackboard*** com o sistema Hearsay



426

Trabalhos Pioneiros (cont.)

- Hewitt propôs o modelo *Actor*
- Atores têm uma duração de vida limitada à realização de uma tarefa codificada na forma de scripts
- Permitia principalmente load-balancing

427

Trabalhos Pioneiros (cont.)

- Smith propôs um dos primeiros protocolos de alocação de tarefas utilizadas por agentes, o protocolo *Contract Net*

428

Divisão da IAD

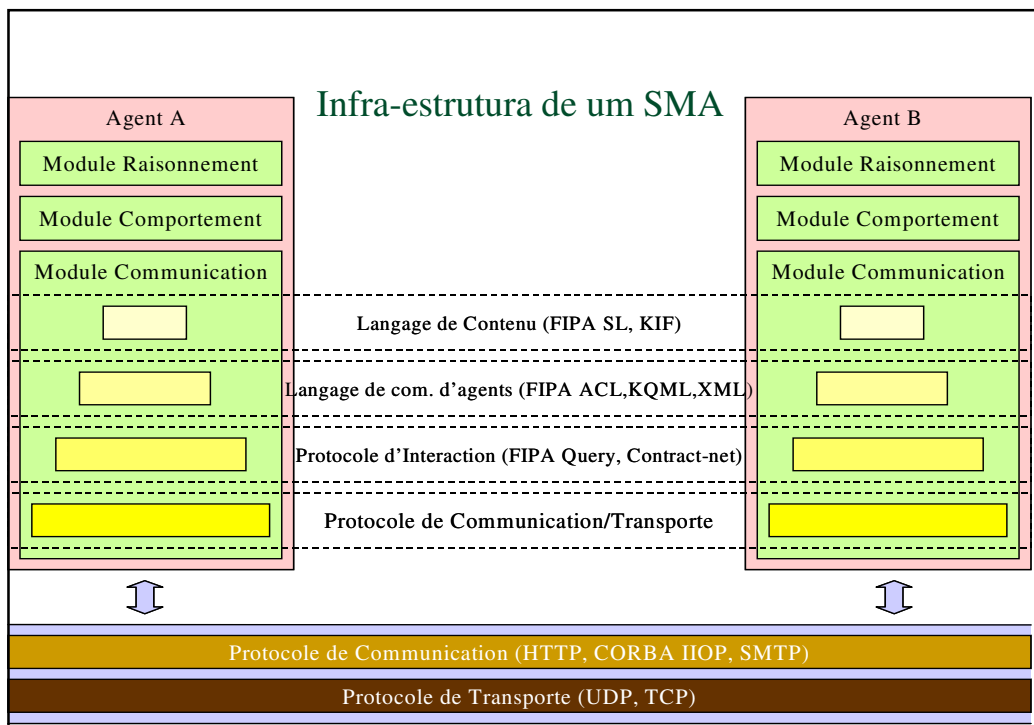
• Resolução distribuída de problemas

- Ex.: *l'éco résolution* proposta por Ferber (1995)
- Cooperation, Collaboration, Competition
- Satisfação distribuída de restrições, Relaxamento de restrições, Alocação de Tarefas, Planejamento Distribuído,...

• Sistemas Multi-Agentes (SMA)

- Mecanismos para a concepção de SMAs como : modelo de comportamento de agentes (autônomos, sociais, comunicantes), métodos de raciocínio (reativos, deliberativos), protocolos de interação (coordenação, cooperação, negociação), infra-estrutura de comunicação ou ainda métodos e ambientes de desenvolvimento

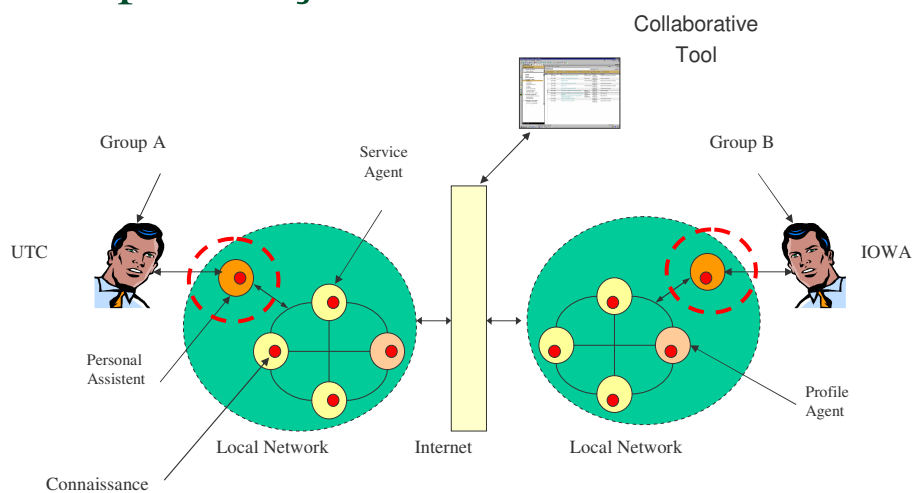
429



Aplicações SMA desenvolvidas durante a tese

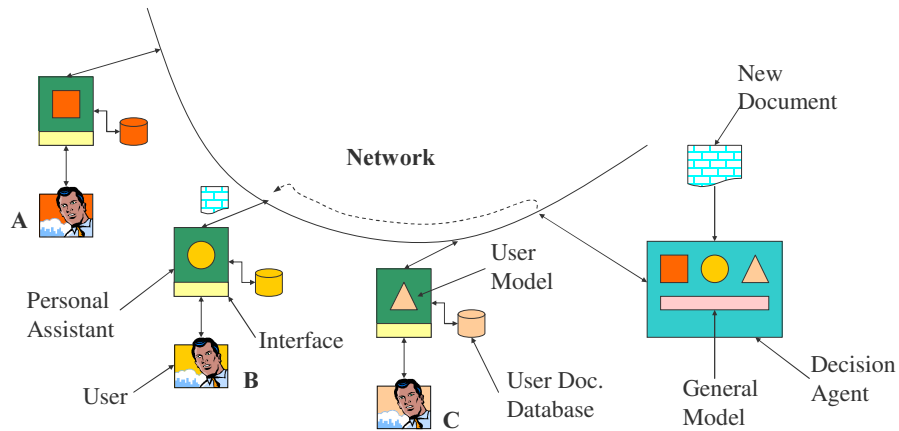
431

Capitalização de conhecimento



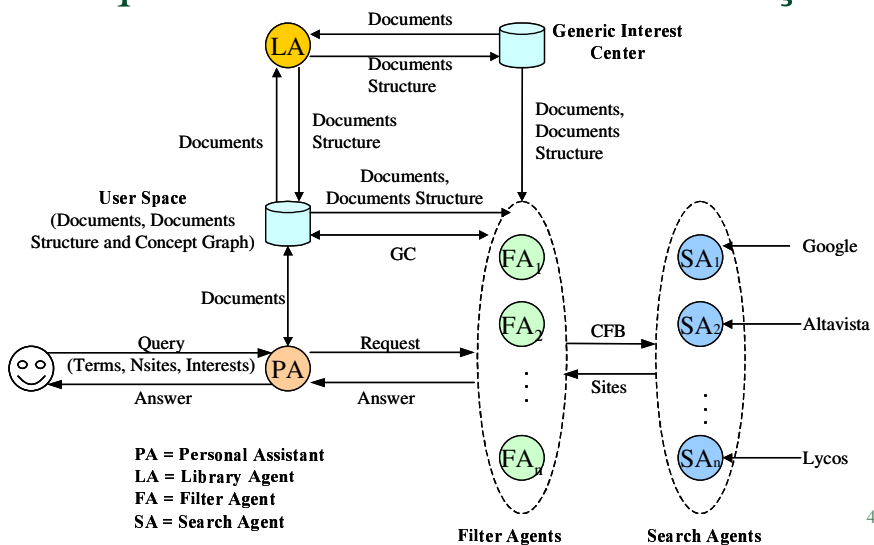
432

Filtragem de documentos



433

Pesquisa Personalizada de Informação



434