Representação de Conhecimento

Capítulo 4

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.

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

137

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?

138

Representação de Fatos

- Representação lógica é comum em programas de IA:
  - Malhado é um cachorro
    - ca*chorro(Malhado)*
  - Todos os cachorros têm rabo
    - $\forall x$*:cachorro(x)->tem_rabo(x)*
  - Malhado tem rabo
    - *tem_rabo(Malhado)*

139

Representação de Propriedades

- Adequabilidade Representacional
- Adequabilidade Inferential
- Eficiência na Inferência
- Eficiência na Aquisição

140

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.

141

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.

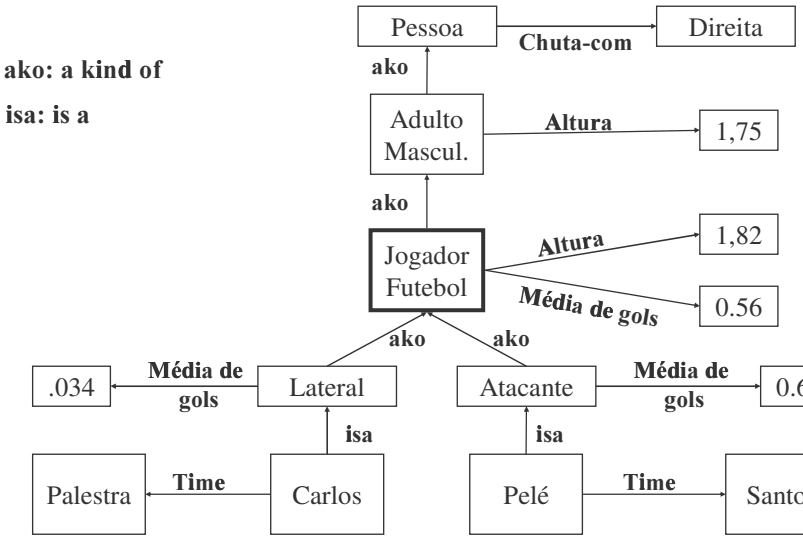
142

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
  - sistemas de frames

143

Herança de Conhecimento



144

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*.

145

Conhecimento Inferencial

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

$$\forall x, y : atacante\,(x) \wedge chute\,(x, y) \wedge não - defende\,(y) \rightarrow gol\,(x)$$

- Inferência baseada em Lógica será estudada posteriormente.

146

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

147

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.

148

Atributos como objetos

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

149

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 está disponível em uma forma de alto nível (Português, Inglês).
- Nem sempre está claro quais primitivas de baixo nível utilizar.

150

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) }

151

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.

152

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.

153

Lógica em Representação de Conhecimento

154

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).

155

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*."

156

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."

$\forall x(membro(x) \rightarrow mora(x))$	Premissa 1
$\forall x(presidente(x) \rightarrow membro(x))$	Premissa 2
presidente(sra_farias)	Premissa 3
mora(sra_farias)	Conclusão

Com a Lógica podemos representar e validar sentenças

Verificação de argumentos ou prova de teoremas

- Dadas as fórmulas β_1, β_2,..., β_n e uma fórmula α, diz-se que essas informações formam um teorema ou o argumento é válido se α é **conseqüência lógica** de β_1, β_2,..., β_n, ou seja

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

- Métodos de prova de teoremas
  - Semântico
  - Sintático
  - Dedutivo
  - Tableau
  - Resolução

Teorema da Dedução ou Admissão de Premissas

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

Prova: Seja I uma interpretação qualquer,

1) Se $\beta_1, \beta_2,...$e β_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, $\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_n$ também será falso em I. Independente do valor de α, $\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_n \rightarrow \alpha$ é verdade em I.

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

159

Teorema da Redução ao Absurdo ou Prova por refutação

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

Prova: Sabe-se pelo teorema anterior que:

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

$\neg(\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_n \rightarrow \alpha) \equiv$

$\neg(\neg(\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_n) \vee \alpha) \equiv$

$\beta_1 \wedge \beta_2 \wedge ... \wedge \beta_n \wedge \neg\alpha$

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

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

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**

162

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

163

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

164

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

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

165

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:

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

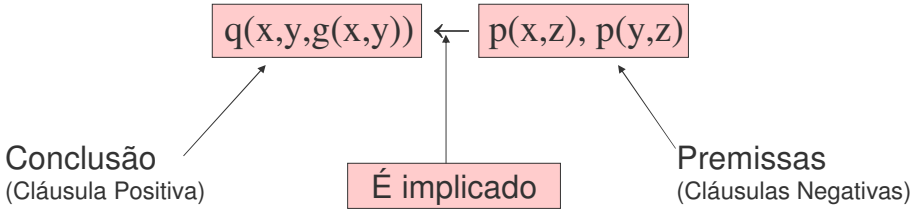
166

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



Notação de Kowalski

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

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

Quando

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

m <= 1: são as chamadas **Cláusulas de Horn**, que têm como casos particulares:

- m = 1 e n > 0: "$A \leftarrow B_1,...,B_n$" (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: "$\leftarrow B_1,...,B_n$" (negação pura de $B_1,...,B_n$) (não há conclusão);
- m = 0 e n = 0: "$\leftarrow$" chamada cláusula vazia, denotada □.

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)

169

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 $\beta_1, \beta_2, ..., \beta_n$ e α onde cada β_i e α estão na **FNC**, aplique a regra da resolução até que a cláusula vazia seja obtida.

170

Exemplo de Resolução - Solução 1

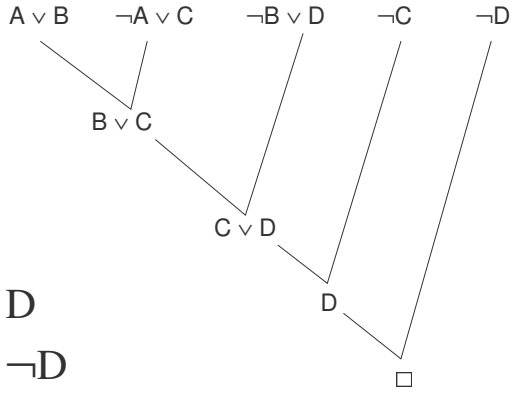
P1: $A \vee B$

P2: $\neg A \vee C$

P3: $\neg B \vee D$

Conclusão: $C \vee D$

$\neg$Conclusão: $\neg C$ e $\neg D$



171

Exemplo de Resolução - Solução 2

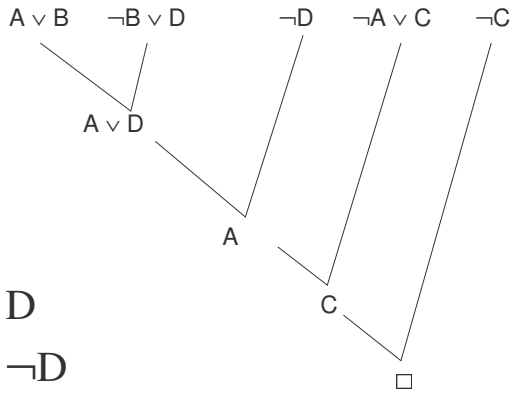
P1: $A \vee B$

P2: $\neg A \vee C$

P3: $\neg B \vee D$

Conclusão: $C \vee D$

$\neg$Conclusão: $\neg C$ e $\neg D$



172

Resolução-SLD*

- Trabalha com Cláusulas Horn:
  1. "$A \leftarrow B_1,...,B_n$"
  2. "$A \leftarrow$"
  3. "$\leftarrow B_1,...,B_n$" (negação pura de $B_1,...,B_n$)
  4. "$\leftarrow$" (cláusula vazia, denotada □)

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

* Resolução linear com função de seleção para cláusulas definidas

173

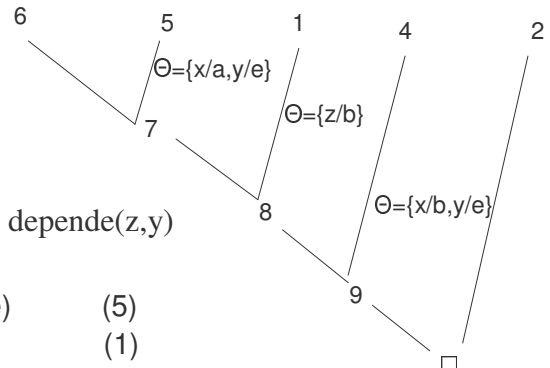
Resolução-SLD*

Conclusão negada: ← depende(a,e)

Exemplo:

Premissas

1. chama(a,b) ←
2. usa(b,e) ←
3. depende(x,y) ← chama(x,y)
4. depende(x,y) ← usa(x,y)
5. depende(x,y) ← chama(x,z), depende(z,y)
6. ← depende(a,e)
7. ← chama(a,z), depende(z,e) (5)
8. ← depende(b,e) (1)
9. ← usa(b,e) (4)
10. □ (2)



174

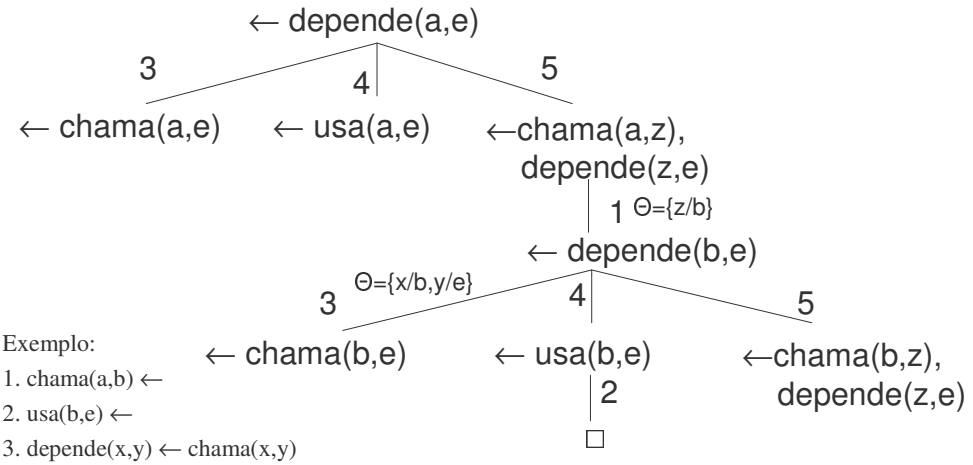
* Resolução linear com função de seleção para cláusulas definidas

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
- Construímos 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

175

Resolução-SLD (cont.)



Exemplo:

1. chama(a,b) ←
2. usa(b,e) ←
3. depende(x,y) ← chama(x,y)
4. depende(x,y) ← usa(x,y)
5. depende(x,y) ← chama(x,z), depende(z,y)
6. ← depende(a,e)

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

Semânticas de um Programa Lógico

177

Semântica Declarativa de um Programa Lógico (PL)

- Um programa lógico P é um conjunto de cláusulas definidas na forma:

  "$A \leftarrow B_1,...,B_n$" ou "$A \leftarrow$"

- Uma consulta *Q* é uma conjunção de literais na forma

  $\leftarrow B_1,...,B_n$

- Uma solução para *Q* é um conjunto de substituições das variáveis de *Q* por termos de *P*

178

Semântica Procedimental de um Programa Lógico (PL)

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

$$\leftarrow B_1,\ldots,B_n$$

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

179

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)

180

Exemplo

- Nevará OU será um teste.
- Dave é Darth Vader OU não nevará.
- Dave não é Darth Vader.

- Será um teste?

181

Solução

- Nevará = a Teste = b Dave is D. Vader = c
- Base de Conhecimento (Tudo é verdadeiro):
  - $a \vee b$, $c \vee \neg a$, $\neg c$
- Por resolução nós sabemos que $b \vee c$ é verdade.
- Por Unidade de Resolução nós sabemos que b é verdade.

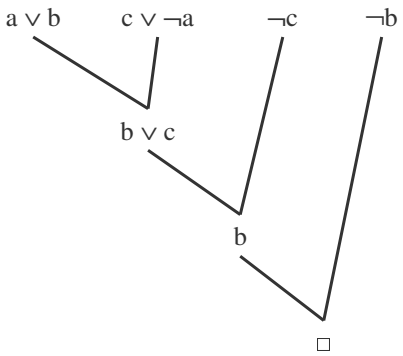
Será um Teste!

182

Usando o método da Resolução

Premissa1: $a \vee b$
Premissa2: $c \vee \neg a$
Premissa3: $\neg c$

Conclusão: b
¬Conclusão: ¬b



Utiliza uma única regra de inferência: de A ∨ B, C ∨ ¬A, conclui-se B ∨ C

183

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.

184

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*.

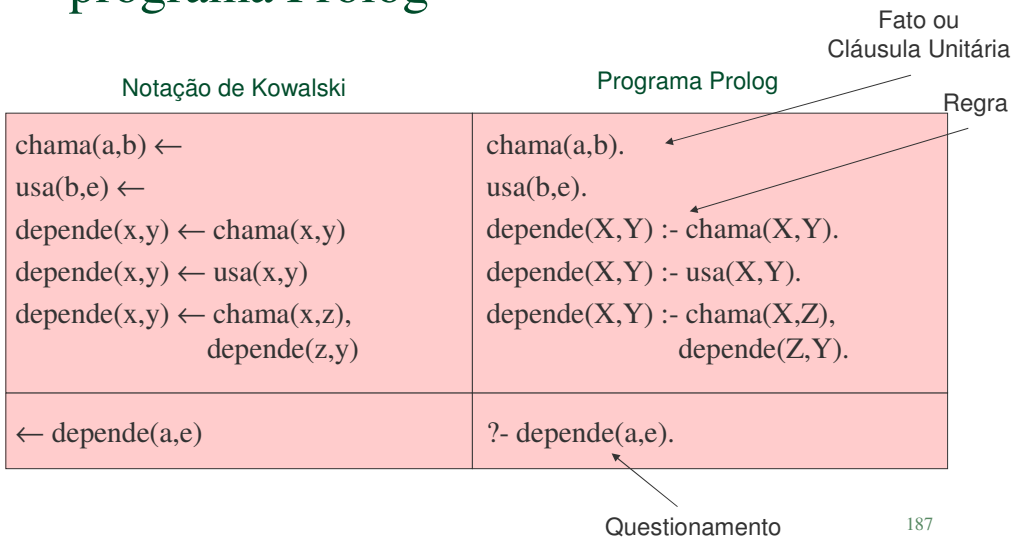
185

Prolog – Uma linguagem de Programação Lógica

- Características
  - Provador 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)

186

Da Notação de Kowalski para um programa Prolog



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.

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.

189

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)?

190

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.

191

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:

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

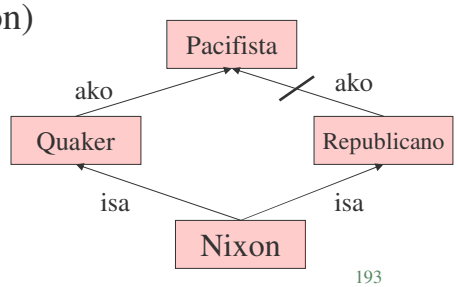
192

Problema com Lógica Não-Monotônica

- $\forall x$: Republicano$(x) \wedge$ M $\neg$ Pacifista$(x) \Rightarrow \neg$ Pacifista(x)
  - $\forall x$: Quaker$(x) \wedge$ M Pacifista$(x) \Rightarrow$ Pacifista(x)

Republicano(Nixon)

Quaker(Nixon)



193

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 ¬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.

194

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.

195

Herança

- É possível suportar herança usando Lógica Default:

$$\frac{\text{Mamífero(x) : Patas(x,4)}}{\text{Patas(x,4)}}$$

  - 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)*

196

Sistemas de Produção

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

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.

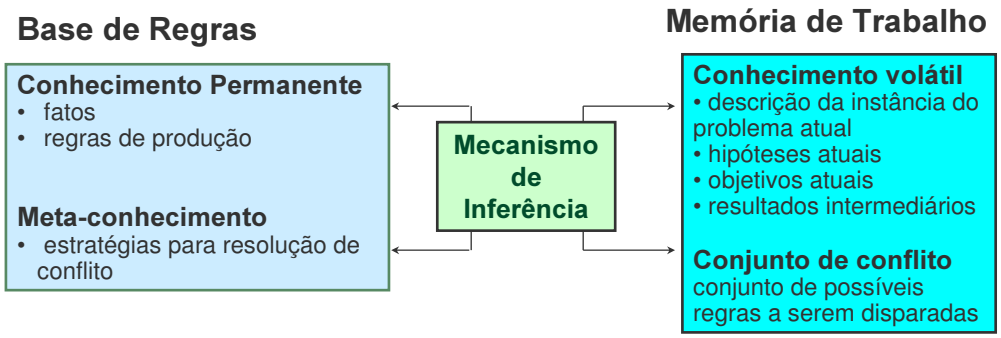
199

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.

200

Arquitetura dos Sistemas de Produção



201

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***

202

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***

203

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***

204

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

205

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)

206

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

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.

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.

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)**

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 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.

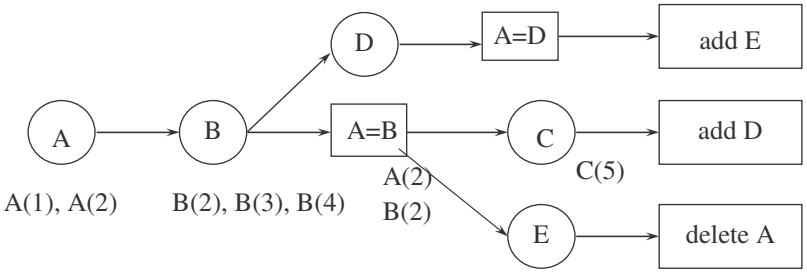
Encadeamento progressivo: Busca e Casamento

- 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

213

Algoritmo RETE: encadeamento progressivo

- Base de Regras:
  - A(x) ^ B(x) ^ C(y) => add D(x)
  - A(x) ^ B(y) ^ D(x) => add E(x)
  - A(x) ^ B(x) ^ E(x) => delete A(x)
- Memória de Trabalho:
  - {A(1), A(2), B(2), B(3), B(4), C(5)}



214

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.

215

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

216

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.

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.

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. instanciação 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

219

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 conseqüente
    - as 7 primeiras regras da nossa BC

220

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 conseqüente
    - apenas a penúltima regra da nossa BC

221

Encadeamento regressivo

- *veículoTipo=ciclo* só é verdade em apenas uma regra
  - **Ciclo: Se** num-rodas < 4
    Então veículoTipo=*ciclo*

- Verifica o valor verdade dos antecedentes da regra
  - num-rodas < 4 ===> FALSO!

- Donde se deduz que *veículo=Bicicleta* é Falso!

222

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

223

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.

224

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!

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

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

227

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

228

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.

229