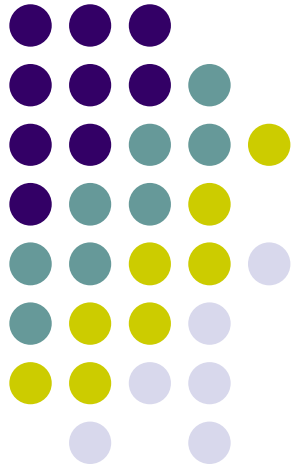
Sistemas de Representação e Raciocínio – Parte 5

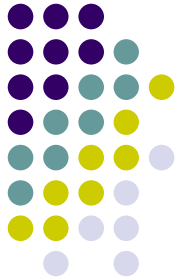
Introdução à Inteligência Artificial

Profa. Josiane

Baseado no material de David Poole,
Alan Mackworth e Randy Goebel
Abril/2007



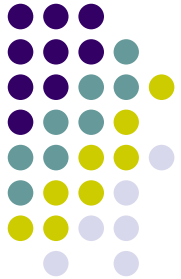
Lógica e Base de Dados



- Uma **base de dados relacional** é uma **base de conhecimento de fatos básicos** (sem variáveis).

- Pode-se usar regras de Datalog para definir as operações da álgebra relacional:
  - Seleção
  - União
  - Junção
  - Projeção

Base de dados

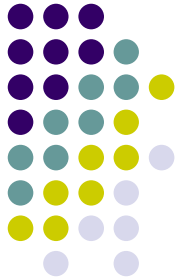


- % *curso*(C) é verdadeiro se C é um curso universitário.
  - *curso(321)*
  - *curso(322)*
  - *curso(315)*
  - *curso(371)*

- % *departamento*(C,D) é verdadeiro se C é um curso oferecido pelo departamento D.
  - *departamento(312, cien_comp)*
  - *departamento(322, cien_comp)*
  - *departamento(315, matem)*
  - *departamento(371, física)*

- % *estudante*(E) é verdadeiro se E é um estudante.
  - *estudante(maria).*
  - *estudante(jane).*
  - *estudante(joão).*
  - *estudante(aroldo).*

Base de dados



- % *feminino*(P) é verdadeiro se P é feminino.
  - *feminino(maria).*
  - *feminino(jane).*

- % *matriculado*(E,C) é verdadeiro se o estudante E está matriculado no curso C.
  - *matriculado(maria, 322).*
  - *matriculado(maria, 312).*
  - *matriculado(joão, 322).*
  - *matriculado(joão, 315).*
  - *matriculado(aroldo, 322).*
  - *matriculado(maria, 315).*
  - *matriculado(jane, 312).*
  - *matriculado(jane, 322).*

Operações relacionais em Datalog

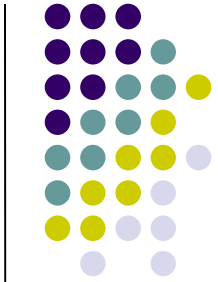
- Operação de **seleção**:
  - Uso de constantes ou variáveis repetidas em uma pergunta ou corpo de uma regra.

    - *curso_CC*(C) ← *departamento*(C, *cien_comp*).
    - *curso_mat*(C) ← *departamento*(C, *matem*).

- Operação de **união**:
  - Uso de regras múltiplas com a mesma conclusão (cabeça)

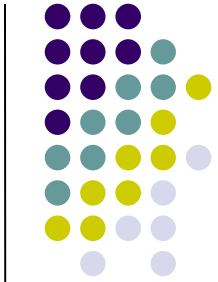
    - *curso_CC_ou_matem*(C) ← *curso_CC*(C).
    - *curso_CC_ou_matem*(C) ← *curso_matem*(C).

Operações relacionais em Datalog



- Operação de **junção**:
  - A conjunção de duas relações no corpo de uma clausula. A junção é sobre a variável compartilhada pelas relações.
    - ? *matriculado*(E,C) $\wedge$ *departamento*(C,D)
- Operação de **Projeção**:
  - Quando existem variáveis no corpo da cláusula que não aparecem na conclusão (cabeça) da regra se diz que a relação está projetada sobre estas variáveis na conclusão.
    - *no_departamento*(E,D) $\leftarrow$ *matriculado*(E,C) $\wedge$ *departamento*(C,D)

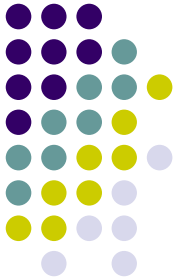
Recursão e Indução Matemática



- *sobre(X,Y)* ← *em(X,Y).*
- *sobre(X,Y)* ← *em(X,Z)* ∧ *sobre(Z,Y).*

- Isto pode ser visto como:
  - Uma definição recursiva de ***sobre***: prove *sobre* em termos de um caso base (***em***) ou um instância simples dele mesmo; ou
  - Como uma forma de provar ***sobre*** por indução matemática: o caso base ocorre quando não existe bloco algum entre X e Y, e se você pode provar ***sobre*** quando existe *n* blocos entre eles, você pode prová-lo quando existe *n+1* blocos.

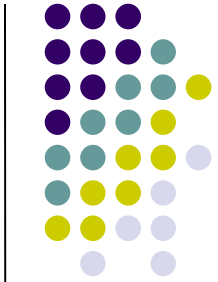
Representando conceitos abstratos - Listas



- Uma lista pode ser:
  - Uma lista vazia → nil.
  - Ou ter uma cabeça e uma cauda → cons(H,T).

- Uma forma de representar mais simples:
  - nil → [].
  - cons(H,T) → [H|T]
  - [A|[B]] → [A,B]
  - [A|[]] → [A]
  - cons(**a**,cons(**b**,cons(**c**, nil))) = [a,b,c]

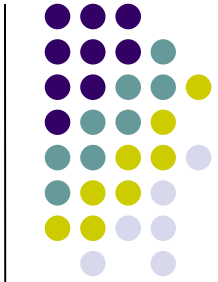
Representando conceitos abstratos - Listas



- Listas quase sempre usam recursão sobre a estrutura da lista

- Funções exemplo:
  - list(L)
  - member(X,L)
  - not_in(X,L)
  - append(X,Y,Z)
  - reverse(L,R)

Representando conceitos abstratos - Listas

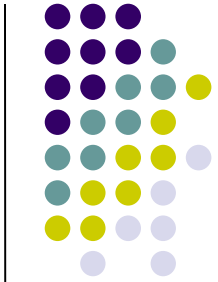


```
% list(L) é verdadeiro se L é uma lista
  list([]).
  list(H|T) :- list(t).

% member(X,L) é verdadeiro se X é um elemento da lista L
  member(X,[X|L]).
  member(X,[H|R]):- member(X,R).

% not_in(X,L) é verdadeira se X não é um elemento da
% lista L --- ou X é diferente de todo membro de L
  not_in(A,[]).
  not_in(A,[H|T]):-
      different(A, H), not_in(A,T).
```

Representando conceitos abstratos - Listas



```
% append(X,Y,Z) é verdadeiro quando X, Y, Z são listas
% e Z contém os elementos de X (em ordem) seguidos
% pelos elementos de Y (em ordem)
  append([],Z,Z).
  append([A|X],Y,[A|Z]) :- append(X,Y,Z).

% rev(L,R) é verdadeiro se a lista R contém os
% elementos da lista L, em ordem contrária.
  rev([],[]).
  rev([H|T],R):-
      rev(T,RT), append(RT,[H],R).
```

Como funciona o *append*

- **append([],Z,Z).**
- **append([A|X],Y,[A|Z]) :- append(X,Y,Z).**
- **?append([a,b], [c,d], Z).**

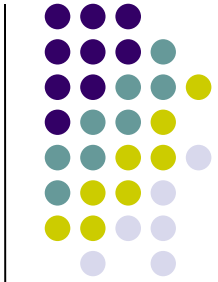
- **yes (Z) <- append([a,b], [c,d], Z).**
  - resolve com append([A1|X1],Y1,[A1|Z1]) :- append(X1,Y1,Z1).
  - unificação {A1/a, X1/[b], Y1/[c,d], Z/[A1|Z1]}
- **yes ([a|Z1]) <- append([b],[c,d],Z1).**
  - resolve com append([A2|X2],Y2,[A2|Z2]) :- append(X2,Y2,Z2).
  - unificação {A2/b, X2/[], Y2/[c,d], Z1/[A2|Z2]}
- **yes ([a,b|Z2]) <- append([],[c,d],Z2).**
  - resolve com append([],Z3,Z3).
  - unificação {Z3/[c, d], Z2/[c,d]}
- **yes ([a,b,c,d]) <- .**

Como funciona o *append*

- `append([],Z,Z).`
- `append([A|X],Y,[A|Z]) :- append(X,Y,Z).`
- `?append([a,b], Y, [a,b,c,d]).`

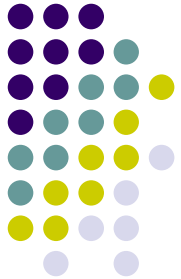
- `yes (Y) <- append([a,b], Y, [a,b,c,d]).`
  - resolve com append([A1|X1],Y1,[A1|Z1]) :- append(X1,Y1,Z1).
  - unificação {A1/a, X1/[b], Y1/Y, Z1/[b,c,d]}
- `yes (Y1) <- append([b],Y1,[b,c,d]).`
  - resolve com append([A2|X2],Y2,[A2|Z2]) :- append(X2,Y2,Z2).
  - unificação {A2/b, X2/[], Y2/Y1, Z2/[c,d]}
- `yes (Y2) <- append([],Y2,[c,d]).`
  - resolve com append([],Z3,Z3).
  - unificação {Z3/[c,d], Y2/[c,d]}
- `yes ([c,d]) <- .`

Exercício 1:



- Considere os seguintes fatos:
  - Cachorros são animais. Todos os animais têm prazer com a alimentação. As pessoas gostam dos animais que elas têm. As pessoas fazem coisas que tragam prazer às coisas que elas gostam. Fido é o cachorro da Maria.
- A) Escreva os fatos acima como um conjunto de cláusulas definidas.
- B) Mostre como definir uma pergunta que questione: “O que Maria faz?”
- C) Mostre a uma derivação SLD para a pergunta acima, mostrando as ligações das variáveis.

Exercício 2:



- Considere o seguinte programa lógico:
  - *f*(*empty*, X,Y).
  - *f*(*cons*(X,Y),W,Z) ← *f*(Y,W,*cons*(X,Z)).
- Apresente cada uma das derivações *top-down*, mostrando as substituições para a pergunta:
  - ?*f*(*cons*(a, *cons*(b, *cons*(c, *empty*))), L, *empty*).