

Linguagem de Programação

Introdução à Linguagem Prolog

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

SLIDES MONTADOS A PARTIR DE SLIDES PREPARADOS PARA A
JAI96 E DE SLIDES OBTIDOS NA INTERNET.

Introdução à Linguagem Prolog



Referência

- SWI-Prolog versão 8.2.1-1 (Windows 64)

Autor

- Jan Wielemaker (jan@swi-prolog.org)
 - Universidade de Amsterdam

Distribuição e informações

- <http://www.swi-prolog.org/>

Mais de um milhão de downloads

O que é Prolog?

Linguagem declarativa

- Não tem instruções

Primeira e mais usada linguagem do paradigma programação em lógica

- Proposta em 1970 como provador de teoremas especializado
 - Alain Colmerauer e Phillipe Roussel (Universidade de Aix-Marseille)
 - Robert Kowalski (Universidade de Edinburgh)
 - Outras pessoas importantes ... **Helder Coelho**, Universidade de Lisboa.

Não usada extensivamente na prática de engenharia de software

- Seriamente avaliada nos 80s como linguagem de propósito geral
 - linguagem do Projeto V Geração do Japão
- Hoje é usada em contextos específicos
 - maioria relacionados a IA
 - mais utilizada na Europa
 - nos EUA a linguagem LISP é mais utilizada em IA

O que é Prolog?

Metáfora da programação em lógica

Teoria Lógica (TL) = programa = BD dedutivo = BC

Programar é declarar **axiomas e regras**

Axiomas **da TL**

- fatos da BC
- parte extensional do BD dedutivo
- dados explícitos de um BD tradicional

Regras **da TL (e da BC):**

- parte intencional do BD dedutivo

Teoremas **da TL**

- deduzidos a partir dos axiomas e das regras
- dados implícitos do BD dedutivo

O que é Prolog?

Uma linguagem de programação para computação simbólica não numérica

- interpretador
- base de conhecimentos, memória de trabalho e máquina de inferência

Com o que se parece programar em Prolog?

- Definir relações e formular perguntas sobre as relações.

O que é o SWI-PROLOG?

- um interpretador
 - ?- prompt principal
 - | prompt secundario

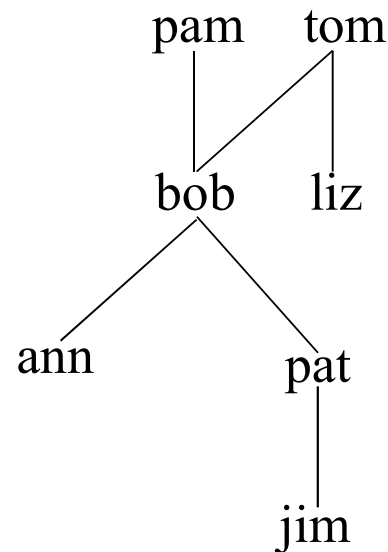
O que é Prolog?

Exemplo de Programa em Prolog

Fatos

```
parent(pam,bob).  
parent(tom,bob).  
parent(tom,liz).  
parent(bob,ann).  
parent(bob,pat).  
parent(pat,jim).
```

parent/2



Perguntas (queries)

?- parent(bob,pat).

true.

?- parent(liz,pat).

fail.

?- parent(X,liz).

X = tom

?- parent(bob, X).

X = ann ;

X = pat.

O que é Prolog?

Exemplo de Programa em Prolog

Quem é pai ou mãe de quem?

?- parent(X,Y).

X = pam,
Y = bob ;
X = tom,
Y = bob ;
X = tom,
Y = liz

Conectivo AND

- Quem é um pai ou mãe X de Ann? Também o é de Pat?
 - ?- parent(X,ann),parent(X,pat).
 - X = bob

O que é Prolog?

Exemplo de Programa em Prolog com Regras

Para todo X e Y, Y é filho de X se X é um pai ou mãe de Y

`offspring(Y,X) :- parent(X,Y).`

- Semântica: se `parent(X,Y)` então `offspring(Y,X)`.

Definição recursiva

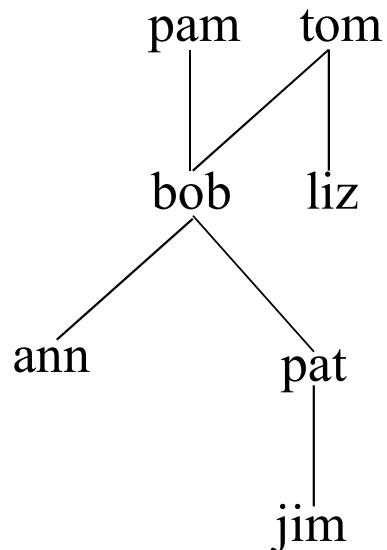
- Para todo X e Z, X é um antepassado de Z se X é pai ou mãe de Z.
- Para todo X e Z, X é um antepassado de Z se existe Y tal que:
 - X é pai ou mãe de Y e
 - Y é um antepassado de Z.

O que é Prolog?

Exemplo de Programa em Prolog com Regras

```
predecessor(X,Z) :-  
parent(X,Z).
```

```
predecessor(X,Z) :-  
parent(X,Y),  
predecessor(Y,Z).
```



Perguntas (queries)

```
?- predecessor(pam,X).
```

X = bob ;

X = ann ;

X = pat ;

X = jim ;

fail.

Regras de Produção

Esquema de representação no qual o conhecimento é representado por regras situação-ação

SE <condição> ENTÃO <ação>.

Especialistas tendem a expressar técnicas de solução de problemas em termos destas regras

- A *condição* estabelece o contexto para aplicação da regra.
- A *ação* corresponde ao procedimento que acarreta uma conclusão ou mudança (no estado corrente)

Regras de Produção

Descrevem relações entre objetos do domínio de acordo com os valores que os atributos podem ter

- incorporam conhecimento prático (heurístico).

Cada regra aproxima um fragmento independente do conhecimento

- o conhecimento existente pode ser refinado com a adição de regras, permitindo um crescimento incremental da base de conhecimentos.

A performance do sistema cresce proporcionalmente ao crescimento da base de conhecimentos

Regras de Produção

As definições lógicas do problema são vistas como uma **representação de conhecimento procedimental**

- aquela em que as **informações de controle** necessárias ao uso do conhecimento **estão embutidas** no próprio conhecimento

Exemplo

SE animal de estimação \wedge pequeno **ENTÃO** bicho de apartamento.

SE gato \vee cachorro **ENTÃO** animal de estimação.

SE poodle **ENTÃO** cachorro \wedge pequeno.

Fido **é** poodle.

Regras de Produção

É necessário ampliar a representação de conhecimento procedimental com um interpretador que siga as instruções (de controle) embutidas

Um sistema típico de regras consiste de uma base de conhecimentos (BC), uma memória de trabalho (MT) e uma máquina de inferências

- A BC é composta de fatos e regras
 - Regras: declarações sobre **classes** de objetos
 - Definem a lógica de processamento.
 - SE condição (antecedente) ENTÃO ação (consequente)
 - Fatos: declarações sobre **objetos específicos**

Regras de Produção

Memória de Trabalho

Representa o estado do problema em um dado instante.

- Permite a comunicação entre regras.

Possui dados dinâmicos de curta duração que existem enquanto uma regra estiver sendo interpretada

- Variáveis da regra **não são estáticas**
 - o escopo é a própria regra.

Ação da regra implica modificações na memória de trabalho ou efeitos colaterais eventuais

Regras de Produção

Máquina de inferências

Responsável pela

- execução das regras.
- determinação de quais são relevantes, dada uma configuração da memória de trabalho.
- escolha de quais regras aplicar.

Ciclo de execução

- seleção das regras
- resolução de conflitos
 - na existência de mais de uma regra a aplicar.
- ação
 - execução da(s) regra(s) selecionada(s) com as consequentes alterações na MT ou efeitos colaterais.

Regras de Produção

Máquina de Inferências: Ciclo de Execução

SELEÇÃO DAS REGRAS (matching)

- Casamento das regras com **dados** da MT
- Conjunto de **conflito** (regras casadas)

ESTRATÉGIAS UTILIZADAS

- guiada por **dados** (forward-chaining)
- guiada por **metas** (backward-chaining)

ESTADO INICIAL (dados da MT)

RACIOCÍNIO PARA FRENTE

- Casamento: MT com **condições** das regras.
- Estados: gerados com **disparo das ações**.
- Meta: permanece a **mesma**.
- Parada: **casamento** com meta.

RACIOCÍNIO PARA TRÁS

- Início: configuração objetivo final (**meta**).
- Casamento: MT com **ação** da meta (mesmo em parte).
- Estados: gerados com as **condições**.
- Meta: **estado atual**
- Parada: casamento com **estado inicial**.

Máquina de Inferências

Exemplo de Raciocínio

Fatos

marco é homem.
césar é homem.

Regra 1:

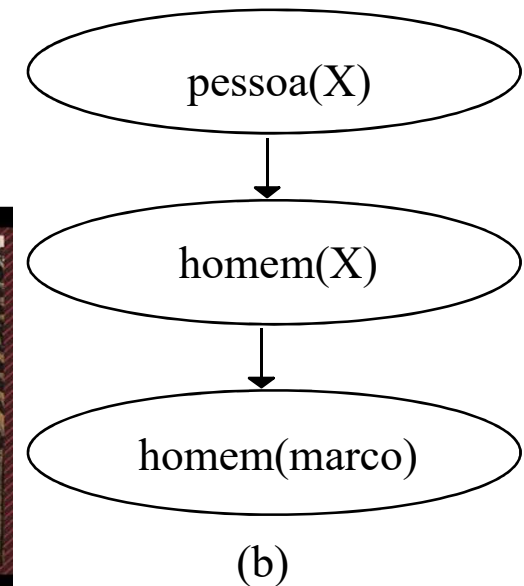
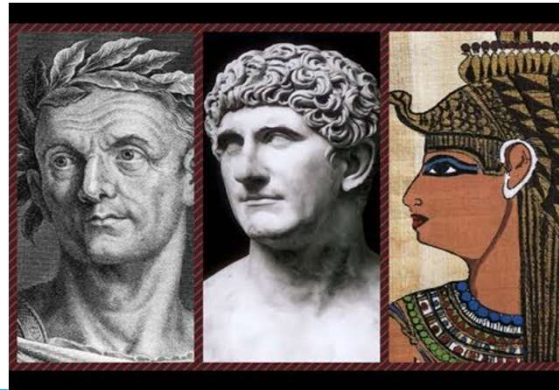
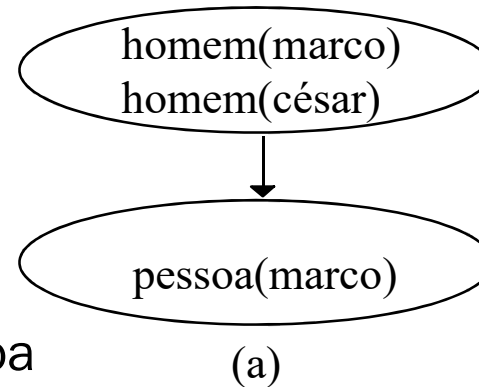
SE X é homem ENTÃO X é pessoa

Meta

Existe pessoa?

Solução

- estado inicial: fatos
- Figura (a): forward-chaining
- Figura (b): backward-chaining



Regras de Produção

Máquina de Inferências Ciclo de execução

Resolução de conflitos

Seleção da ordem de aplicação das regras do conjunto de conflito.

Preferência baseada na:

- **ordem** em que as regras aparecem (Prolog)
- **importância** dos objetos que foram casadas (ELIZA, Weizenbaum, 1966)

Talvez **eu** pudesse aprender a conviver com a minha **mãe**.

- Talvez **você** pudesse aprender a conviver com a sua mãe.
- Fale mais sobre a sua **família**.

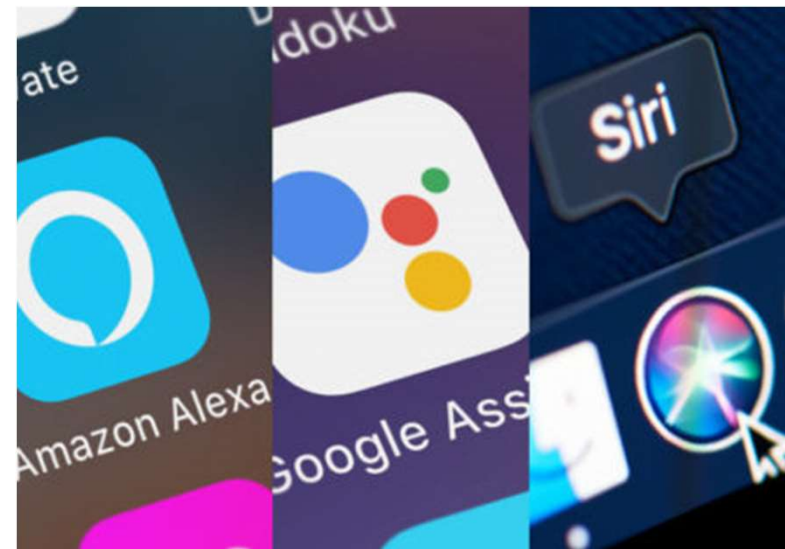
- **em estados**

- Consiste em, temporariamente, disparar **todas as regras selecionadas** e utilizar uma **função heurística para avaliar os resultados** de cada uma delas, e **priorizá-las** segundo o seu mérito.

Joseph Weizenbaum, o pai da ELIZA, a avó da Siri !

Weizenbaum was shocked with users' reaction to ELIZA. He found it shocking that users opened their hearts to ELIZA, revealed secrets/personal problems with her(or should I say it), thought it was a real therapist and that it really understood them and believed that it could help. Upon informing his secretary that he had logs of all the conversations, she reacted with outrage at the invasion of her privacy.

```
> Hello, I am Eliza.  
* I am afraid.  
> How long have you been afraid?  
* Since midterms.  
> Tell me more...  
* I am afraid that I will fail  
my course.  
> Do you enjoy being afraid that  
you will fail your course?
```



Linguagem Prolog

Programação em lógica

- definições lógicas são vistas como programas.
 - Em Prolog definições lógicas são cláusulas de Horn.

É praxe representar apenas o conhecimento positivo

- asserções afirmativas

Hipótese do Mundo Fechado

- as declarações relevantes e verdadeiras estão contidas na BC ou podem ser derivadas a partir de fatos, regras e a BC.

Negação

- ausência da declaração.
 - Não é uma negação lógica.

Resolução por refutação

P é consistente se falhar a prova de $\neg P$.

Linguagem Prolog

Implicação

Representação da implicação

Tabela verdade

A	B	$A \rightarrow B$
V	V	V
V	F	F
F	V	V
F	F	V

$\neg A$	B	$\neg A \vee B$
F	V	V
F	F	F
V	V	V
V	F	V

$$A \rightarrow B \Leftrightarrow \neg A \vee B$$

Linguagem Prolog

Cláusulas de Horn

Cláusula que tem, no máximo, um literal positivo

“Well Formed Formula” em forma conjuntiva normal e sem conectivo \wedge

- prefixo quantificadores universais (\forall) aplicado a predicados conectados por \vee

Lógica decidível

Notação Lógica

$\forall x(\text{estimação}(x) \wedge \text{pequeno}(x) \rightarrow \text{bichoapt}(x))$

$\forall x(\text{gato}(x) \vee \text{cachorro}(x) \rightarrow \text{estimação}(x))$

$\forall x(\text{poodle}(x) \rightarrow \text{cachorro}(x) \wedge \text{pequeno}(x))$

Cláusulas de Horn

$\neg \text{estimação}(x) \vee \neg \text{pequeno}(x) \vee \text{bichoapt}(x)$

$\neg \text{gato}(x) \vee \text{estimação}(x)$

$\neg \text{cachorro}(x) \vee \text{estimação}(x)$

$\neg \text{poodle}(x) \vee \text{cachorro}(x)$

$\neg \text{poodle}(x) \vee \text{pequeno}(x)$

poodle(fido)

poodle(fido)

UMA TEORIA EM UM SISTEMA LÓGICO É DECIDÍVEL SE EXISTE UM ALGORITMO EFICIENTE
PARA DETERMINAR SE FÓRMULAS ARBITRÁRIAS PERTENCEM A ELA.

Linguagem Prolog

Características

Quantificadores universais não explicitados.

Conectivos ‘,’ e ‘;’ (respectivamente \wedge e \vee)

A regra $p \rightarrow q$ é representada como $q :- p$.

cabeça :- lista de predicados.

Cláusulas de Horn

\neg **estimação(x)** \vee \neg **pequeno(x)** \vee **bichoapt(x)**

\neg **gato(x)** \vee **estimação(x)**

\neg **cachorro(x)** \vee **estimação(x)**

\neg **poodle(x)** \vee **cachorro(x)**

\neg **poodle(x)** \vee **pequeno(x)**

poodle(fido)

Notação Prolog

bichoapt(X):- estimação(X), pequeno(X).

estimação(X):-gato(X).

estimação(X):-cachorro(X).

cachorro(X):-poodle(X).

pequeno(X):-poodle(X).

poodle(fido).

Linguagem Prolog

Cláusulas de Horn

Fatos Prolog

- cláusulas de Horn com premissa única **True** implícita

$$C. \Leftrightarrow \text{True} \rightarrow C$$

Regras Prolog

- cláusulas de Horn

$$C :- P1, \dots, Pn. \Leftrightarrow P1 \& \dots \& Pn \rightarrow C$$

Premissas de cláusulas com a mesma conclusão são implicitamente disjuntivas:

$$C :- P1, \dots, Pn.$$

$$C :- Q1, \dots, Qm.$$

$$\Leftrightarrow (P1 \& \dots \& Pn) \vee (Q1 \& \dots \& Qm) \rightarrow C$$

Linguagem Prolog

Máquina de Inferências

Inferência

- resolução por refutação

Seleção das regras

- raciocínio para trás com indexação das regras pelo predicado e casamento (*matching*) precedido pelas instanciações das variáveis das regras (unificação)

Resolução de conflitos

- preferência baseada em regras (ordem)

Interpretador Prolog

Controle e busca

Aplica regra de resolução

- com estratégia *linear* (sempre tenta unificar *ultimo fato* a provar com a *conclusão de uma cláusula do programa*),
- na *ordem de escrita* das cláusulas *no programa*,
- com encadeamento de regras *para trás*,
- *busca em profundidade* e
- *da esquerda para direita* das premissas das cláusulas,
- e com *backtracking sistemático* e *linear* quando a *unificação falha*,
- e sem *occur-check* na unificação.

Estratégia eficiente mas incompleta.

Interpretador Prolog

Verificação de ocorrência

unificação de Prolog é sem *occur-check*, quando chamado com uma variável X e um literal l , instancia X com l , *sem verificar* antes se X ocorre em l .

Junto com a busca em profundidade:

- faz com que Prolog possa entrar em loop com regras recursivas,
 - $c(X) :- c(p(X))$. gera lista infinita de objetivos:
 - $c(p(U)), c(p(p(U))), c(p(p(p(U))))$, ...
- *cabe ao programador* não escrever tais regras,
- torna a *unificação linear* no lugar de quadrática no tamanho dos termos a unificar

Vantagens

Ampla expressividade

Representação de associações empíricas (heurísticas) em domínios não estruturados

Codificação da experiência de especialistas na resolução de problemas

Possui sintaxe e semântica simples

Aplicação

- sistemas especialistas, em especial, de diagnóstico

Prototipação

- crescimento incremental da BC.

Desvantagens

Falta de estruturação da BC dificulta

- introduzir modificações na BC.
 - localizar informações desejadas.
 - representar estruturas inerentes ao domínio tais como:
 - taxonomia de classes
 - relações temporais
 - relações estruturais
 - herança de atributos
- ⇒ Regras podem ser modularizadas de forma a se obter subconjuntos independentes e complementares, o que facilita o processo de resolução de conflitos

Não facilita a distinção semântica entre propriedades essenciais e propriedades complementares dos objetos