Lógica Computacional

Aula Teórica 20: Introdução ao Prolog

Ricardo Gonçalves

Departamento de Informática

23 de novembro de 2023

Prolog

- **PRO**gramação em **LÓG**ica
- Declarativa
- Muito diferente das outras linguagens de programação (imperativas ou funcionais)
- Adequada para tarefas baseadas em conhecimento

História do Prolog

- 1972 - primeiro interpretador de Prolog (Colmerauer e Roussel)
- 1977 - Implementação do compilador DEC10 (Warren)
- 1980 - implementação de Gramáticas de Cláusulas Definidas (**Pereira** e Warren)
- 1980s/90s - Prolog ganha popularidade especialmente na Europa e no Japão
- 2005 - Prolog usado para programar interface de língua natural Estação Espacial Internacional (NASA)
- 2011 - Prolog usado para programar parte do Watson

Ideia básica

- Descrever o problema numa base de conhecimento em lógica
- Fazer interrogações
- O Prolog deduz novos factos que são consequência lógica
- O Prolog devolve as suas deduções como respostas às interrogações

Novo paradigma de programação

- Pensar declarativamente, não procedimentalmente
  - Difícil
  - Exige uma abordagem diferente
- Linguagem de alto-nível
  - Não tão eficiente como, por exemplo, C
  - Adequada para modelação rápida
  - Útil em muitas aplicações em IA

Exemplo - Base de conhecimento e consultas

Base de conhecimento 1

```
estudante(tomas).
estudante(rodrigo).
estudante(lurdes).
cantaAlto(rodrigo).
festa.
```

Consultas

```
?-estudante(tomas).
true
?-cantaAlto(rodrigo).
true
?-cantaAlto(tomas).
false
```

Exemplo - Base de conhecimento e consultas

Base de conhecimento 1

```
estudante(tomas).
estudante(rodrigo).
estudante(lurdes).
cantaAlto(rodrigo).
festa.
```

Consultas

```
?-tocaInstrumento(tomas).
procedure ‘tocaInstrumento(A)’ does
not exist

?-festa.
true

?-concertoRock.
procedure ‘concertoRock’ does not exist
```

Factos e regras

Factos

A base de conhecimento anterior só tinha factos, tais como:

estudante(tomas).

Regras

Uma base de conhecimento pode também ter regras:

ouveMusica(lurdes):- feliz(lurdes), em_casa(lurdes).

O lado esquerdo da regra é chamado de **cabeça** da regra.

O lado direito da regra é chamado de **corpo** da regra.

Exemplo - Base de conhecimento e consultas

Base de conhecimento 2

feliz(lurdes). — facto
ouveMusica(tomas). — facto
ouveMusica(lurdes):- feliz(lurdes). — regra
cantaAlto(tomas):- ouveMusica(tomas). — regra
cantaAlto(lurdes):- ouveMusica(lurdes). — regra

Consultas

?-cantaAlto(tomas).
true
?-cantaAlto(lurdes).
true

Cláusulas

Base de conhecimento 2

feliz(lurdes). — facto
ouveMusica(tomas). — facto
ouveMusica(lurdes):- feliz(lurdes). — regra
cantaAlto(tomas):- ouveMusica(tomas). — regra
cantaAlto(lurdes):- ouveMusica(lurdes). — regra

Há 5 cláusulas nesta base de conhecimentos: 2 factos e 3 regras.
O fim de uma cláusula é assinalado com um ponto.

Cláusulas

Base de conhecimento 2

```
feliz(lurdes).
ouveMusica(tomas).
ouveMusica(lurdes):- feliz(lurdes).
cantaAlto(tomas):- ouveMusica(tomas).
cantaAlto(lurdes):- ouveMusica(lurdes).
```

Há 3 predicados nesta base de conhecimentos:

feliz, ouveMusica, e cantaAlto.

Representação da conjunção

```
feliz(max).
ouveMusica(rodrigo).
cantaAlto(max):- ouveMusica(max), feliz(max).
cantaAlto(rodrigo):- feliz(rodrigo).
cantaAlto(rodrigo):- ouveMusica(rodrigo).
```

A virgula “,” representa a conjunção em Prolog

```
?- cantaAlto(max).
false
?- cantaAlto(rodrigo).
true
```

Representação da conjunção

```
feliz(max).
ouveMusica(rodrigo).
cantaAlto(max):- ouveMusica(max), feliz(max).
cantaAlto(rodrigo):- feliz(rodrigo).
cantaAlto(rodrigo):- ouveMusica(rodrigo).
```

As consultas também podem ser conjuntivas.

```
?- cantaAlto(max), cantaAlto(rodrigo).
false
?- cantaAlto(rodrigo),ouveMusica(rodrigo).
true
```

Representação da disjunção

```
feliz(max).
ouveMusica(rodrigo).
cantaAlto(max):- ouveMusica(max), feliz(max).
cantaAlto(rodrigo):- feliz(rodrigo).
cantaAlto(rodrigo):- ouveMusica(rodrigo).
```

A disjunção pode ser representada pela existência de várias regras para definir o mesmo predicado, ou usando o ponto e vírgula ";"

```
feliz(max).
ouveMusica(rodrigo).
cantaAlto(max):- ouveMusica(max), feliz(max).
cantaAlto(rodrigo):- feliz(rodrigo); ouveMusica(rodrigo)
```

Consultas

Só podemos colocar questões de sim/não?

Não! A linguagem de consulta é mais expressiva do que isso.
Podemos usar variáveis.

Outras consultas

Base de conhecimento

```
estudante(tomas).
estudante(rodrigo).
estudante(lurdes).
filhoDe(rodrigo, ricardo).
filhoDe(tomas, ricardo).
filhoDe(rodrigo, lurdes).
filhoDe(lurdes, miguel).
```

Consultas

```
?-estudante(X).
X=tomas;
X=rodrigo;
X=lurdes.

?-filhoDe(rodrigo,X), estudante(X).
X=lurdes.

?-filhoDe(tomas,X), estudante(X).
false
```

Outras consultas

Base de conhecimento

```
filhoDe(tomas, ricardo).
filhoDe(rodrigo, ricardo).
filhoDe(tomas, lurdes).
filhoDe(lurdes, miguel).
irmao(X,Y):- filhoDe(X,Z), filhoDe(Y,Z).
```

Consultas

```
?-irmao(tomas,W).
W=tomas;
W=rodrigo;
W=tomas.
```

Unificação

- A unificação é um dos mecanismos mais poderosos do Prolog
- Prolog tem um algoritmo de unificação
- Podemos usar "=" para testar unificação e obter unificador

```
?- lurdes=lurdes
true
?-lurdes=max
false
?- lurdes = X.
X=lurdes.
?- X=lurdes, X=max.
false
```

O X do primeiro objetivo é unificado com lurdes, logo não pode depois unificar com max.

Unificação

Mais alguns exemplos de unificação

```
?- k(s(g),Y) = k(X,t(k)).
X=s(g),
Y=t(k).

?- k(s(g),t(k)) = k(X,t(Y)).
X=s(g),
Y=k.

?- filhoDe(X,X) = filhoDe(max,lurdes).
false
```

Unificação - caso problemático

Diferentes implementações de Prolog usam diferentes algoritmos de unificação, muitos deles diferem do algoritmo standard quando há partilha de variáveis. Exemplo:

```
?- pai(X) = X.
```

Qual deveria ser a resposta, segundo o algoritmo que estudámos? Em algumas implementações:

```
?- pai(X) = X.
X=pai(pai(pai(...))).
```

Em outras existe occurs check, o que resulta em:

```
?- pai(X) = X.
false
```

Unificação - caso problemático

Na implementação que vamos usar:

```
?- pai(X) = X.
X=pai(X).
```

Mas podemos forçar a utilização de occurs check:

```
?- unify_with_occurs_check(X,pai(X)).
false
```

Variáveis anónimas

O underscore "_" representa uma variável anónima
É usada quando não interessa o valor dessa variável
Cada ocorrência de "_" é independente
Podemos ver na unificação, em que a unificação de "_" é ignorada:

```
?- primos(mae(X),Y)=primos(_,jose).
Y=jose
```

E usar na modelação

```
temNota(jose,12).
foiAoTeste(X):- temNota(X,_).
```

Obter resultados só com unificação

Obter termos complexos a partir da unificação é um dos mecanismos mais poderosos do Prolog.

```
vertical(linha(ponto(X,_),ponto(X,_))).
horizontal(linha(ponto(_,Y),ponto(_,Y))).
```

```
?- vertical(linha(ponto(1,1),ponto(1,3))).
true
?- vertical(linha(ponto(1,1),ponto(3,2))).
false
?- horizontal(linha(ponto(1,1),ponto(2,Y))).
Y = 1
?- horizontal(linha(ponto(2,3),X)).
X = ponto(_1714,3).
```

Predicados e constantes

Como se devem escrever predicados e constantes?

- Uma sequência de caracteres de letras maiúsculas, minúsculas, números ou 'underscore', começando com uma letra minúscula
- Exemplos: rodrigo, feliz, cantaAlto
- Uma sequência arbitrária de caracteres entre plicas
- Exemplos: 'Rodrigo', 'O tal', '@$%'
- Predicados com o mesmo nome, mas aridades diferentes, são considerados diferentes.

Variáveis

Como se devem escrever variáveis?

- Uma sequência de caracteres de letras maiúsculas, minúsculas, números ou 'underscore', começando com uma letra maiúscula ou 'underscore'.
- Exemplos: X, Y, Variavel, Rodrigo, _aluno

Aritmética

Podemos representar números inteiros: 12, -34, 22342,...
Números reais: 12.42, -34.1, ...
Temos as operações aritméticas: +, -, / e *
Mas as expressões só são avaliadas quando usamos o predicado "is"

```
?- 10 = 5+5.
false
?- 10 is 5+5.
true
```

```
?- X is 3 * 4.
X=12.
?- R is mod(7,2).
R=1.
```

Atenção: "is" só avalia a expressão aritmética da direita, e depois compara-a com o número à esquerda.
Temos também os comparadores de números: <, >, <=, >=,=/=

Definir predicados usando aritmética

```
adionar3eDuplicar(X,Y):- Y is (X+3)*2.
```

```
?- adionar3eDuplicar(1,X).
X=8.

?- adionar3eDuplicar(2,X).
X=10.
```

Listas

- Uma lista é uma sequência finita de elementos
- Os elementos das listas estão entre parêntesis rectos
- Exemplos de listas em Prolog:

- [lurdes, max, jose, maria]
- [lurdes, feliz(jose), X, 2, lurdes]
- []
- [lurdes, [max, jose], [rodrigo, triste(rodrigo)]]
- [[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

- O comprimento de uma lista é o seu número de elementos
- Qualquer termo Prolog pode ser um elemento de uma lista
- Existe uma lista especial: a lista vazia [ ]

Cabeça (Head) e Resto (Tail)

- Uma lista não vazia tem duas partes
  - A cabeça
  - O resto
- A cabeça é o primeiro elemento da lista
- O resto é tudo menos o primeiro elemento
- O resto de uma lista é sempre uma lista

[lurdes, max, jose, maria]
Head: lurdes
Tail: [max, jose, maria]

[[ ], triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
Head: [ ]
Tail: [triste(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

[triste(z)]
Head: triste(z)
Tail: [ ]

Cabeça e resto da lista vazia

- A lista vazia não tem nem cabeça nem resto
- Em Prolog, [ ] é uma lista simples especial sem nenhuma estrutura interna
- A lista vazia tem um papel importante nos predicados recursivos para o processamento de listas em Prolog

Cabeça e resto da lista vazia

- O Prolog tem um operador especial "|" que pode ser usado para decompor a lista em duas partes: cabeça e resto
- O operador "|" é essencial para escrever predicados de manipulação de listas

```
?- [X|Y] = [lurdes, max, jose, maria].
X = lurdes,
Y = [max, jose, maria].

?- [X|Y] = [ ].
false

?- [X,Y|Tail] = [[ ], triste(z), [2, [b,c]], [], Z, [2,[b,c]]]
X=[ ],
Y = triste(z),
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]].
```

Experimentar Prolog

- SWI Prolog é a mais popular implementação de Prolog
- Site: https://www.swi-prolog.org/
- Versão online: https://swish.swi-prolog.org/