Fundamentos

Paradigma de Programação Lógico

Marco A L Barbosa

http://github.com/malbarbo/na-func

Conteúdo

Tutorial

- Fatos
- Consultas
- Variáveis
- Conjunções
- Regras

Visão mais detalhada

- Definições
- Constantes, variáveis e estruturas
- Operadores
- Unificação
- Aritmética

Escrevendo código em Prolog

Tutorial

Tutorial

- Neste tutorial não seremos muito formais
- Programar em Prolog consiste em
  - Especificar fatos sobre objetos e suas relações
  - Definir regras sobre objetos e suas relações
  - Fazer consultas (perguntas) sobre objetos e suas relações

Fatos

Fatos

- Um fato é algo que é verdadeiro sobre uma relação de objetos
- Fato: João utiliza o editor vim.

```
editor(joao, vim).
```

- `joao` e `vim` são objetos, `editor` é uma relação
- Os nomes das relações e dos objetos devem começar com letras minúsculas
- A ordem dos objetos é arbitrária, mas você deve ser consistente
- Os objetos de uma relação são chamados de argumentos
- O nome da relação é chamado de predicado

Fatos

- Uma relação pode ter qualquer quantidade de argumentos
- Fato: Está chovendo.

```
chovendo.
```

- Fato: Maria comprou um livro do Jorge.

```
comprou(maria, livro, jorge).
```

Consultas

Consultas

- Podemos fazer consultas sobre os fatos que foram definidos
- Dado os seguintes fatos

```
editor(joao, vim).
editor(pedro, emacs).
```

- Podemos fazer algumas consultas
  - É verdade que o João utiliza o editor vim?

```
?- editor(joao, vim).
true.
```

  - É verdade que o João utiliza o editor emacs?

```
?- editor(joao, emacs).
false.
```

- Observe que a forma de uma consulta é a mesma de um fato

Consultas

- Quando uma consulta é realizada o Prolog faz uma busca por fatos que unificam com o fato que está sendo consultado
  - Dois fatos unificam se os predicados são os mesmos e cada argumento correspondente é o mesmo
- Se um fato que unifica com a consulta for encontrado, o Prolog irá responder `true`, caso contrário o Prolog responderá `false`
- A resposta `false` significa que não foi encontrado um fato que unifica com a questão

Consultas

- Fatos

```
humano(socrates).
humano(aristoteles).

ateniense(socrates).
```

- Consulta

```
?- ateniense(aristoteles).
false.
```

- Apesar de poder ser verdade no mundo real que Aristóteles era ateniense (viveu em Atenas), nós não podemos provar isto a partir dos fatos dados

Variáveis

Variáveis

- Para fazer perguntas que as respostas não sejam apenas `true` e `false` usamos variáveis
- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
```

- Consulta
  - Existe algum E tal que Pedro utiliza o editor E?

```
?- editor(pedro, E).
E = emacs.
```

- Observe que as variáveis começam com letra maiúscula

Variáveis

- O Prolog realiza uma busca da mesma forma que antes, mas considera que uma variável não instanciada unifica com qualquer objeto
- Quando o Prolog encontra um fato que unifica com a consulta, ele marca o fato e exibe os valores unificados com as variáveis
  - Se o utilizador pressionar a tecla enter, a busca é finalizada
  - Se o utilizador pressionar a tecla ; a busca é reiniciada a partir da marca

Variáveis

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
```

- Consulta
  - Existe algum E tal que João utiliza o editor E?

```
?- editor(joao, E).
E = vim ;
E = emacs.
```

Conjunções

Conjunções

- Também é possível fazer consultas mais elaboradas usando conjunções (e)
- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consultas
  - João e Pedro utilizam o editor emacs?
  - João utiliza o editor emacs e Pedro utiliza o editor emacs?

```
?- editor(joao, emacs), editor(pedro, emacs).
true.
```

- O símbolo "," é pronunciado "e"

Conjunções

- Quando uma sequência de metas separadas por vírgula é dada para o Prolog, ele tenta satisfazer uma meta por vez
- Todas as metas devem ser satisfeitas para a consulta ser satisfeita

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existe algum E tal que João e Maria utilizam o editor E?
  - Existe algum E tal que João utiliza o editor E e Maria utiliza o editor E?

```
?- editor(joao, E), editor(maria, E).
E = vim ;
false.
```

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existem X e Y tal que X e Y utilizam o editor emacs?
  - Existem X e Y tal que X utiliza o editor emacs e Y utiliza o editor emacs?

```
?- editor(X, emacs), editor(Y, emacs).
X = Y, Y = joao ;
X = joao,
Y = pedro ;
X = pedro,
Y = joao ;
X = Y, Y = pedro.
```

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim)
```

- Consulta
  - Existem X e Y tal que X e Y utilizam o editor emacs e o nome de X "vem antes" que o de Y?

```
?- editor(X, emacs), editor(Y, emacs), X @< Y.
X = joao,
Y = pedro ;
false.
```

Conjunções

- Fatos

```
editor(joao, vim).
editor(joao, emacs).
editor(pedro, emacs).
editor(maria, vim).
```

- Consulta
  - Existem X, Y e Z tal que X e Y utilizam o editor Z?

```
?- editor(X, Z), editor(Y, Z).
```

  - Qual é a resposta?

Regras

Regras

- Forma de abstração utilizada pelo Prolog
- Usamos regras para dizer que um fato depende de um outro grupo de fatos
- Uma **regra** é um sentença genérica sobre objetos e suas relações
- Exemplo: dois programadores podem fazer um par para programação se eles utilizam o mesmo editor

```
par(X, Y) :-
    editor(X, Z),
    editor(Y, Z),
    X @< Y.
```

Visão mais detalhada

Definições

Definições

- Um **predicado** é a coleção de fatos e regras com o mesmo nome e aridade
- Uma **cláusula** é um fato ou uma regra

Constantes, variáveis e estruturas

Tipos de dados

- Um programa em Prolog é construído com termos
- Termos
  - Constantes
    - Átomos
    - Números
  - Variáveis
  - Estruturas
- Cada termo é definido com uma sequência de caracteres
  - Letras maiúsculas: `A .. Z`
  - Letras minúsculas: `a .. z`
  - Dígitos: `0 .. 9`
  - Símbolos: `+ - * /\ ~ ^ < > : . ? @ ### $`

Constante

- Constantes nomeiam objetos ou relações específicas
- Átomos começam com letra minúscula ou símbolo, ou entre apóstrofos '

```
casa
-->
'Jose da Silva'
'123'
```

- Números

```
89
-17
2.67e10
```

Variáveis

- Parecem átomos mas começam com letras maiúsculas ou _

```
X
Reposta
Nome_longo
```

- Variáveis anônimas são definidas com o caractere _
- Cada ocorrência de uma variável anônima refere-se a um valor (que pode ser diferente das ocorrências anteriores
- Usamos variáveis anônimas quando não estamos interessados no valor

```
?- gosta(joao, _). % existe alguém que joao gosta?
true.
```

Estruturas

- Estruturas também são chamadas de termos compostos
- Uma **estrutura** é um único objeto composto de outros objetos chamados de argumentos (ou componentes)
- Semelhante a registros em linguagens imperativas, mas os componentes não são nomeados
- Fato: João possui o livro Algoritmos escrito pelo Cormem

```
possui(joao, livro(algoritmos, cormem)).
```

- O nome da estrutura é chamado de **functor**, no exemplo o functor é `livro`
- `algoritmos` e `cormem` são os **componentes** da estrutura `livro(algoritmos, cormem)`
- A quantidade de componentes em uma estrutura é a **aridade** da estrutura

Operadores

Operadores

- As vezes é conveniente escrever functors como operadores
- `x + y` ao invés de `+(x, y)`
- Observe que os dois exemplos descrevem o mesmo objeto, a estrutura com o functor `+` e os componentes `x` e `y`
- O Prolog usa regras de precedência e associatividade semelhantes e de outras linguaguens
  - `2 + 4 * 3` é o mesmo que `+(2, \*(4, 3))`
  - `8/2/2` é o mesmo que `/(/(8,2), 2)`
- Lembre-se: estas construções descrevem estruturas, elas só são interpretadas (e avaliadas) como expressões aritméticas em alguns contexto (veremos a seguir)

```
?- X = 2 + 4 * 3, write_canonical(X).
+(2,*(4,3))
X = 2+4*3.
```

Igualdade

- Predicados pré-definidos
  - = (negação \=) Verifica se dois termos podem ser unificados

```
?- casa == casa.
true.
?- X = 4.
X = 4.
?- livro(X, alg) = livro(autor(thomas, cormem), Y).
X = autor(thomas, cormem),
Y = alg.
```

  - == (negação \==) Verifica se dois termos são iguais (não tenta fazer a unificação)

```
?- casa == casa.
true.
?- X == 4.
false.
?- livro(X, alg) == livro(autor(thomas, cormem), Y).
false.
```

Unificação

Unificação

- Ideia: dois termos unificam se eles são os mesmos termos ou se eles contêm variáveis que podem ser instanciadas de maneira que os termos resultantes sejam iguais

Unificação

- Dois termos constantes unificam se eles são iguais
- Um termo que é uma variável não instanciada unifica com qualquer outro termo. No caso de duas variáveis não instanciadas é criado um co-referência, neste caso, quando uma das variáveis é instanciada, a outra também é
- Duas estruturas unificam se
  - Elas têm o mesmo functor e a mesma aridade
  - Todos os argumentos correspondentes unificam
  - A instanciação das variáveis são compatíveis

Unificação

- Exemplos

```
?- casa = casa.
true.

?- casa = carro.
false.

?- X = Y, gosta(joao, Y) = gosta(joao, pizza).
X = Y, Y = pizza.

?- livro(X, algoritmos) = livro(autor(thomas, cormem), Y).
X = autor(thomas, cormem),
Y = algoritmos.
```

Aritmética

Aritmética

- Termos que representam expressões aritméticas são avaliados quando usados com os predicados =:=, =\=, >, >=, <, =<

```
?- 3 + 4 =:= 10 - 3.      % igual
true.

?- 4 * 3 =\= 4 + 4 + 4.   % diferente
false.

?- X = 3, Y = 5, X + Y > 2 * X.
X = 3,
Y = 5.

?- X = 3, Y = 5, X + Y < 2 * X.
false.

?- X > 2.
ERROR: >/2: Arguments are not sufficiently instantiated
```

- Observe que os termos não podem ter variáveis não instanciadas

Aritmética

- O operador `is` pode ser usado para instanciar uma variável com o resultado de uma expressão aritmética
- O termo da direita é interpretado como um expressão aritmética e o resultado da avaliação da expressão é unificado com o termo da esquerda

```
?- X is 3 + 4 * 2.
X = 11.

?- 2 + 2 is 2 + 2.
false.

?- 2 is X.
ERROR: is/2: Arguments are not sufficiently instantiated
```

Escrevendo código em Prolog

Escrevendo código em Prolog

- Documentação
  - De acordo com PLdoc
  - Um predicado pode ser
    - `det` (determinístico) satisfeito uma vez sem escolha
    - `semidet` (semi determinístico) falha ou é satisfeito uma vez sem escolha
    - `nondet` (não determinístico) sem limite de vezes que o predicado é satisfeito e pode deixar escolha na última vez que é satisfeito
  - Padrões de instanciação
    - `+` argumento precisa estar completamente instanciado
    - `-` argumento não pode estar instanciado
    - `?` argumento pode ou não estar instanciado
- Testes
  - Usaremos a biblioteca plunit

Exemplo 1 - quadrado

```
:- use_module(library(plunit)).

% quadrado(+X, ?Y) is semidet
%
% Verdadeiro se Y é o quadrado de X.

:- begin_tests(quadrado).

test(quadrado3, Q == 9) :- quadrado(3, Q).
test(quadrado4) :- quadrado(4, 16).

:- end_tests(quadrado).

quadrado(X, Y) :-
    Y is X * X.
```

Para executar os utilize o predicado `run_tests`

```
?- run_tests.
% PL-Unit: quadrado .. done
% All 2 tests passed
true.
```

Exemplo 2 - fatorial

```
:- use_module(library(plunit)).

% fat(+N, ?F) is semidet
%
% Verdeiro se F é o fatorial de N.

:- begin_tests(fatorial).

test(f0) :- fat(0, 1).
test(f1) :- fat(1, 1).
test(f2) :- fat(2, 2).
test(f3) :- fat(3, 6).
test(f4) :- fat(4, 24).
test(f5, F == 120) :- fat(5, F).

:- end_tests(fatorial).
```

Exemplo 2 - fatorial

```
fat(N, F) :-
    N >= 1,
    N0 is N - 1,
    fat(N0, F0),
    F is N * F0.

fat(0, 1).
```