Fundamentos

Marco A L Barbosa
malbarbo.pro.br

Departamento de Informática
Universidade Estadual de Maringá

Definições

Definições

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

Termos

Termos

- Um programa em Prolog é construído com termos
- Um termo pode ser
  - Constante
    - Átomo
    - Número
  - Variável
  - Estrutura

Tipos de dados

- 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

- 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

Estruturas

- 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`

Operadores

- O Prolog usa regras de precedência e associatividade semelhantes a de outras linguagens
  - `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

- = (negação \=) Verdadeiro se os 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.
```

Igualdade

- == (negação \==) Verdadeiro 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 uma 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 (observe que a definição é recursiva)
  - A instanciação das variáveis são compatíveis

Unificação

```
?- 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

Documentação

- De acordo com PLdoc

Documentação

- 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

Documentação

- Padrões de instanciação (descrição não muito precisa)
  - `+` argumento precisa estar completamente instanciado
  - `-` argumento não pode estar instanciado
  - `?` argumento pode ou não estar instanciado
  - `@` nenhum instanciação é feita

Testes

- Usaremos a biblioteca plunit

Exemplo 2.1

Defina um predicado `quadrado(X, Y)` que é verdadeiro se $Y = X^2$.

Exemplo 2.1

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

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

:- begin_tests(quadrado).

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

:- end_tests(quadrado).
```

Exemplo 2.1

```
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 2.2

Defina um predicado `fatorial(N, F)` que é verdadeiro se o fatorial de N é F.

Exemplo 2.2

```
:- 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, [fail]) :- fat(4, 22).
test(f5, F == 120) :- fat(5, F).

:- end_tests(fatorial).
```

Exemplo 2.2

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

fat(0, 1).
```

Referências

Referências básicas

- Capítulos 2 e 3 do livro The Power of Prolog
- Capítulos 1 e 2 do livro Programming in Prolog
- Capítulos 2 e 3 da apostila Paradigmas de programação - Prolog
- Capítulos 1 , 2 e 5 do livro Learn Prolog Now.

Referências complementares

- Introduction to Prolog