

Programação Lógica

Fundamentos

Prof. Edson Alves

Faculdade UnB Gama

2020

Sumário

1. Paradigma Lógico
2. Fatos
3. Consultas simples

Paradigma Lógico

- ▶ O paradigma lógico foi elaborado a partir de um teorema proposto no contexto do processamento de linguagens naturais
- ▶ Ele permite prototipamento e desenvolvimento rápidos devido sua proximidade semântica com a especificação lógica do problema ser resolvido
- ▶ A programação, sob o viés do paradigma lógico, é declarativa
- ▶ Os programas são compostos por uma série de relações lógicas e consultas, as quais questionam se uma determinada proposição é ou não verdadeira
- ▶ Caso seja verdadeira, também é possível determinar qual atribuição de valores lógicos às variáveis da proposição a torna verdadeira
- ▶ Para se tornar um paradigma prático, é necessário o apoio de subrotinas extra-lógicas que controlem os processos de entrada e saída de dados e que manipulem o fluxo de execução do programa

Prolog

- ▶ A linguagem de programação lógica Prolog foi proposta em 1972 por Alain Colmerauer e Philippe Roussel, tendo como base os trabalhos de Robert Kowalski
- ▶ Prolog é uma contração da expressão “*PRO*gramming in *LOG*ic”
- ▶ Ela tem raízes na lógica de primeira ordem
- ▶ O SWI Prolog pode ser instalado em distribuições com suporte ao apt por meio do comando

```
$ sudo apt-get install swi-prolog
```

- ▶ O interpretador (*listener*) Prolog pode ser invocado com o comando

```
$ prolog
```

- ▶ Para validar a instalação, utilize o comando

```
$ prolog -v
```

Programas em Prolog

- ▶ Um programa em Prolog é composto de uma coleção de pequenas unidades modulares, denominadas **predicatos**
- ▶ Os predicatos são semelhantes às subrotinas de outras linguagens
- ▶ Eles podem ser testados e adicionados separadamente em um programa, de modo que é possível construir programas incrementalmente
- ▶ O ato de inserir de códigos no interpretador Prolog é denominado **consultar**
- ▶ É possível consultar diretamente um arquivo-fonte no interpretador Prolog por meio do predicado `consult()`:

```
?- consult(source)
```

Programas em Prolog

- ▶ Se o arquivo for modificado, ele deve ser relido através do predicado `reconsult()`:

```
?- reconsult(source)
```

- ▶ A extensão dos arquivos-fonte deve ser `'.pl'`
- ▶ Usando o comando

```
$ prolog -s source.pl
```

o conteúdo de `source.pl` é carregado no interpretador e o terminal fica pronto para consultas

- ▶ Para fazer uma consulta sem entrar no modo iterativo, use a opção `'-g'` para estabelecer o objetivo e a opção `'-t'` para encerrar o Prolog:

```
$ prolog -s source.pl -g "goal(X),print(X),nl,fail." -t halt
```

Exemplo de programa em Prolog

- ▶ A proposição “*Todo estudante da FGA é estudante da UnB*” pode ser declarada em Prolog da seguinte forma:

```
unb(X) :- fga(X).
```

- ▶ Outra forma de ler esta mesma proposição seria “*Para todo X, X é aluno da UnB se X estuda na FGA*”
- ▶ Afirmações sobre estudantes da FGA pode ser feitas por meio do predicado **fga**/1:

```
fga(ana).  
fga(beto).  
fga(carlos).
```

- ▶ Observe o ponto final (‘.’) que encerra cada proposição/predicado

Exemplo de arquivo-fonte em Prolog

```
1 % Exemplo de arquivo-fonte Prolog
2 unb(X) :- fga(X).
3
4 fga(ana).
5 fga(beto).
6 fga(carlos).
```


Exemplo de programa em Prolog

- ▶ Se estes predicados forem inseridos em um arquivo denominado 'students.pl', ele pode ser consultado no interpretador Prolog através do predicado `consult/1`:

```
?- consult(students).  
true.
```

- ▶ Para saber checar se Ana é estudante da UnB, basta utilizar a consulta

```
?- unb(ana).  
true.
```

- ▶ Para saber se Diana também é estudante da UnB, a consulta deve ser

```
?- unb(diana).  
false.
```

Exemplo de programa em Prolog

- ▶ Para listar todos os estudantes conhecidos da UnB, faça a consulta

```
?- unb(X).  
X = ana .
```

- ▶ Observe que a consulta atribuiu corretamente $X = \text{ana}$, pois Ana é estudante da UnB
- ▶ A consulta retorna o primeiro valor encontrado para X que torna a sentença aberta $\text{unb}(X)$ verdadeira
- ▶ Para obter os demais valores de X que também tornam tal sentença verdadeira, utilize o operador ponto-e-vírgula (;) após cada retorno:

```
?- unb(X).  
X = ana ;  
X = beto ;  
X = carlos.
```

Exemplo de programa em Prolog

- ▶ É possível formatar a lista de todos os estudantes da UnB por meio dos predicados `write/1` e `nl/0`
- ▶ O predicado `write/1` escreve seu argumento no terminal
- ▶ O predicado `nl/0` inicia uma nova linha no terminal
- ▶ Estes predicados podem ser combinados com a consulta sobre os estudantes da UnB para formar o predicado `unb_report/0`:

```
unb_report :-  
    write('Estudantes da UnB: '), nl,  
    unb(X),  
    write(X), nl,  
    fail.
```

- ▶ Este novo predicado por ser consultado no interpretador da seguinte forma:

```
?- unb_report.
```

Arquivo-fonte completo do exemplo

```
1 % Exemplo de arquivo-fonte Prolog
2 unb(X) :- fga(X).
3
4 fga(ana).
5 fga(beto).
6 fga(carlos).
7
8 unb_report :-
9     write('Estudantes da UnB: '), nl,
10    unb(X),
11    write(X), nl,
12    fail.
```

Terminologia

- ▶ O jargão de Prolog é composto por termos de programação, termos de bancos de dados e termos lógicos
- ▶ Não há uma divisão clara, em Prolog, entre dados e procedimentos
- ▶ Um programa em Prolog é um banco de dados Prolog
- ▶ Sinônimos para um termo são introduzidos entre parêntesis
- ▶ Por exemplo, no nível mais alto temos `program(database)`
- ▶ Um programa é composto por predicados (procedimentos, registros, relações)
- ▶ Cada predicado é definido um nome e um número (**aridade**)
- ▶ A aridade é o número de argumentos (atributos, campos) do predicado
- ▶ Dois predicados com nomes diferentes, mais aridades distintas, são considerados distintos

Terminologia

- ▶ No exemplo anterior são três os predicados: `unb/1`, `unb_report/0` e `fga/1`
- ▶ `fga/1` lembra um registro com um campo de outras linguagens
- ▶ `unb_report/0` se assemelha a uma subrotina sem argumentos
- ▶ `mortal/1` remete a uma regra ou proposição, e está em algum lugar entre dados e procedimentos
- ▶ Cada predicado do programa é definido pela existência de uma ou mais cláusulas no banco de dados
- ▶ No exemplo, `fga/1` tem 3 cláusulas, os demais predicatos apenas uma cláusula
- ▶ Cada cláusula pode ser uma **regra** ou um **fato**
- ▶ As quatro cláusulas de `fga/1` são fatos
- ▶ As demais são regras

Fatos em Prolog

- ▶ Os fatos são os predicados mais simples do Prolog
- ▶ Eles se assemelham a registros em um banco de dados relacional
- ▶ A sintaxe para a declaração de um fato é

`pred(arg1, arg2, ..., argN).`

onde `pred` é o nome do fato, e `arg1`, `arg2`, ..., `argN` são os argumentos, sendo N a aridade

- ▶ O ponto final (‘.’) encerra todas as cláusulas de Prolog
- ▶ Se a aridade do predicado for zero, a sintaxe se reduz a

`pred.`

Termos

- ▶ Os argumentos podem ser quaisquer termos válidos de Prolog
- ▶ Os termos básicos do Prolog são
 - ▶ **inteiro**: número positivo, negativo ou zero, com valor absoluto máximo dependendo da implementação
 - ▶ **átomo**: uma constante de texto iniciada com letra minúscula
 - ▶ **variável**: começa com letra maiúscula ou sublinhado ('_')
 - ▶ **estrutura**: termos complexos
- ▶ Algumas implementações podem estender esta lista, com strings e ponto flutuante, por exemplo
- ▶ O uso de aspas simples permitem a construção de átomos por meio de qualquer combinação válida de caracteres
- ▶ Os nomes dos predicados seguem as mesmas regras dos átomos

Fatos em programas em Prolog

- ▶ Os fatos são frequentemente utilizados para inserir informações no programa
- ▶ Por exemplo, para o predicado `paciente/3` podem ser atestados os seguintes fatos:

```
paciente('Maria Rita', 35, sus)
paciente('Pedro Silva', 70, amil)
```

- ▶ As aspas foram usadas nos nomes porque começam em maiúsculas e porque tem espaços
- ▶ Um interpretador Prolog deve fornecer meios de inserção de fatos e regras em uma base de dados dinâmica, a qual pode ser consultada
- ▶ A base de dados é atualizada por meio de consultas (`consult/1` ou `reconsult/1`)

Fatos em programas em Prolog

- ▶ Os predicados podem ser inseridos diretamente no interpretador, mas não são gravados entre as sessões
- ▶ Isto pode ser feito por meio do predicados `asserta/1` e `assertz/1`
- ▶ O primeiro insere um novo fato como primeiro dentre os fatos declarados para o predicado
- ▶ O segundo insere o novo fato como o último dentre os já declarados
- ▶ Os nomes utilizados no fato são indiferentes para o Prolog, mas para a aplicação as relações devem ser compatíveis com a semântica dos identificadores escolhidos
- ▶ Prolog considera distintos os fatos `fato(a, b)` e `fato(b, a)`

Consultas simples

- ▶ Uma vez que a base de dados do interpretador Prolog seja alimentado com fatos, o este interpretador pode responder a consultas (*queries*) a respeito dos fatos
- ▶ As consultas em Prolog funcionam por meio do casamento de padrões (*pattern matching*)
- ▶ O padrão de uma consulta é denominado **objetivo** (*goal*)
- ▶ Se algum fato atinge o objetivo, a consulta é bem sucedida e o interpretador responde “Sim” (**true.**)
- ▶ Caso contrário, a consulta falha e o interpretador responde “Não” (**false.**)
- ▶ O casamento de padrões do Prolog é denominado **unificação**

Unificação

Unificação (versão simplificada)

Se o programa contém apenas fatos, a unificação é bem sucedida se as três condições abaixo são satisfeitas:

1. o predicado citado no objetivo e na base de dados é o mesmo,
2. ambos tem a mesma aridade,
3. todos os argumentos são os mesmos.

Exemplos de unificação

```
1 % Os fatos abaixo devem ser carregados no interpretador Prolog
2 f(1, 2).
3 f(1, 3).
4 f(2, 3).
5
6 g(1).
7 g(2).
8
9 %% Exemplos de consultas
10 % ?- f(1, 2).
11 % true.
12 %
13 % ?- f(2, 1).
14 % false.
15 %
16 % ?- f(1).
17 % false.
18 %
19 % ?- g(1, 2).
20 % false.
```

Referências

1. **MERRIT**, Dennis. *Adventure in Prolog*, Amzi! Inc, 191 pgs, 2017.
2. **SHALOM**, Elad. *A Review of Programming Paradigms Throughout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.
3. SWI Prolog. [Swi Prolog](#), acesso em 10/11/2020.
4. Wikipédia. [Prolog](#), acesso em 10/11/2020.