Introdução à Linguagem Prolog

Prof. Dr. Silvio do Lago Pereira

slago@ime.usp.br

1 Introdução

Prolog (**pro***gramming in* **log***ic*) [1,2,5] é uma linguagem de programação declarativa, baseada em cláusulas de Horn, que tem o algoritmo Sld-Resolução embutido. Enquanto numa linguagem de programação imperativa um programa consiste numa descrição detalhada de como um determinado problema deve ser solucionado, em Prolog, um programa é apenas um conjunto de sentenças que declaram o conhecimento que temos a cerca de um determinado contexto. Uma vez que essas sentenças sejam fornecidas ao sistema Prolog, o usuário pode consultá-lo, fazendo perguntas cujas respostas serão deduzidas, automaticamente, a partir do conhecimento que foi declarado pelas sentenças.

1.1 Fatos, regras e consultas

Para introduzir os elementos básicos da linguagem Prolog, vamos usar como exemplo um famoso argumento de lógica clássica:

Sócrates é um homem.
Todo homem é mortal.

Logo, Sócrates é mortal.

Nesse argumento, as duas primeiras sentenças são *premissas*, enquanto a última delas é uma *conclusão*. Claramente, essa conclusão pode ser deduzida das premissas. Então, como o sistema Prolog implementa um algoritmo de raciocínio dedutivo, se fornecermos a ele essas premissas e perguntarmos se Sócrates é mortal ele deverá responder que sim. Para tanto, a primeira coisa que temos a fazer é escrever as premissas numa forma que elas possam ser entendidas pelo sistema. Escrevendo as premissas em Prolog, obtemos o seguinte programa:

Programa 1 Filósofos

```
homem(sócrates).
mortal(X) :- homem(X).
```

Com relação à notação lógica usada nas cláusulas de Horn, as diferenças sintáticas em Prolog são pequenas: primeiro, todas as cláusulas devem finalizar com ponto; segundo, o operador ← é substituído pelo operador `:-`; e, terceiro, nos fatos, o operador ← é omitido.

No Programa 1, a cláusula

```
homem(sócrates).
```

estabelece um *fato* e deve ser lida como "*Sócrates é um homem*". Por outro lado, a cláusula

```
mortal(X) :- homem(X).
```

estabelece uma *regra* e deve ser lida como "*X é mortal se X é um homem*". Um *programa* em PROLOG nada mais é do que um conjunto de fatos e regras.

Para executar o Programa 1, entramos no sistema SWI-PROLOG[1] e, ao sinal de prontidão desse sistema, digitamos `emacs('filósofos.pl')`, seguido de `ponto` e `enter`. Isso fará com que o editor EMACS seja aberto para a criação do arquivo[2] `filósofos.pl`. Em seguida, digitamos as cláusulas que compõem o programa e compilamos (usando a opção *compile buffer*, no menu *compile*, do editor EMACS). Se nenhum erro de digitação tiver sido cometido, o sistema informará que o programa foi corretamente compilado e que está pronto para ser consultado.

```
filósofos.pl compiled, 0.01 sec, 588 bytes.
yes
?-
```

Para testar o sistema, podemos começar com a seguinte consulta:

```
?- homem(sócrates).
```

ou seja, "*Sócrates é um homem?*". A essa consulta o sistema responderá "*yes*", já que esse fato foi explicitamente estabelecido pela primeira cláusula do Programa 1. Outra consulta que podemos fazer é "*Sócrates é mortal?*":

```
?- mortal(sócrates).
```

Dessa vez, embora o fato `mortal(sócrates)` não tenha sido explicitamente estabelecido, o sistema pode deduzi-lo a partir das cláusulas do programa `filósofos.pl`. Sendo assim, a resposta para essa segunda consulta também será "*yes*".

1.2 A hipótese do mundo fechado e a negação por falha finita

Conforme vimos, o sistema PROLOG é capaz de responder positivamente a respeito de informações que lhe comunicamos explicitamente, através de *fatos*, ou implicitamente, através de *regras*. Mas o que acontece se lhe consultarmos sobre algo que não lhe foi comunicado? Por exemplo, ainda considerando o Programa 1, poderíamos fazer a seguinte consulta:

[1] Download disponível em `http://www.swi-prolog.org`.
[2] Apenas arquivos com extensão `.pl` são reconhecidos como programas em PROLOG.

```
?- mortal(platão).
```

ou seja, "*Platão é mortal?*". A essa pergunta o sistema responderá "*no*". Entretanto, essa resposta não significa que Platão seja imortal, mas apenas que o sistema não dispõe de conhecimento suficiente para afirmar que Platão é mortal. Esse tipo de resposta negativa é baseada na *hipótese do mundo fechado* [4], segundo a qual *o sistema pode supor que conhece todos os fatos verdadeiros a respeito do mundo*. De acordo com essa hipótese, se um fato não foi comunicado ao sistema, seja explícita ou implicitamente, o sistema pode assumir que esse fato é falso.

Devido à implementação da hipótese do mundo fechado, o PROLOG é um sistema lógico não-monotônico, *i.e.*, a adição de novas premissas pode descartar conclusões anteriormente obtidas. Por exemplo, se adicionarmos ao programa `filósofos.pl` o fato `homem(platão)`, e repetirmos a consulta `?- mortal(platão)`, o sistema responderá "*yes*". Ou seja, a resposta do sistema a uma consulta depende do conhecimento que ele tem a respeito do contexto de discurso: se esse conhecimento muda, suas respostas também podem mudar.

A negação em PROLOG, conhecida como *negação por falha*[3], é também implementada com base na hipótese do mundo fechado. Quando o sistema tem que responder a respeito de um fato negativo, primeiro ele verifica se esse fato é verdadeiro. Caso o fato seja verdadeiro, ele responde "*no*"; senão, ele responde "*yes*". Por exemplo, à consulta:

```
?- not( mortal(sócrates) ).
```

o sistema responderá "*no*", já que o fato `mortal(sócrates)` é verdadeiro. Por outro lado, à consulta:

```
?- not( mortal(zeus) ).
```

o sistema responderá "*yes*", já que o fato `homem(zeus)` não foi comunicado ao sistema e, portanto, de acordo com a hipótese do mundo fechado, deve ser assumido como sendo falso.

1.3 Consultas com variáveis

Veremos agora que o PROLOG é capaz de responder mais do que simplesmente "*yes*" ou "*no*". Por exemplo, suponha que desejamos saber *quem* (`X`)[4] é mortal:

```
?- mortal(X).
```

[3] A negação de um fato é considerada verdadeira se o sistema *falha* ao tentar provar que esse fato é verdadeiro.

[4] No PROLOG, todo identificador iniciando com maiúscula é considerado uma variável.

Nesse caso, não basta que o sistema responda "*yes*". Será preciso que ele informe *quem* é mortal, ou seja, ele deverá encontrar um valor apropriado para a variável `X`, que será a resposta à nossa pergunta. Portanto, ele responderá `X = sócrates`.

Caso haja mais de uma resposta possível, o sistema exibirá a primeira delas e ficará aguardando instruções: se digitarmos *ponto-e-virgula*, ele tentará encontrar uma resposta alternativa; se digitarmos *enter*, ele encerrará a consulta.

Por exemplo, supondo que o fato `homem(platão)` tenha sido incluído no programa `filósofos.pl`, o sistema fornecerá as seguintes respostas:

```
?- mortal(X).
X = sócrates ;
X = platão
yes
```

Exercício 1 *Codifique as cláusula a seguir em* Prolog *e consulte o sistema para descobrir o que é saudável.*

$bebe(ze, pinga) \leftarrow$
$bebe(mane, agua) \leftarrow$
$vivo(mane) \leftarrow$
$saudavel(X) \leftarrow bebe(Y, X), vivo(Y)$ □

Exercício 2 *Codifique as cláusula a seguir em* Prolog *e consulte o sistema para descobrir que idiomas a Ana fala e que idiomas o Yves fala.*

$nasceu(ana, brasil) \leftarrow$
$nasceu(yves, france) \leftarrow$
$idioma(brasil, portugues) \leftarrow$
$idioma(franca, frances) \leftarrow$
$idioma(inglaterra, ingles) \leftarrow$
$estudou(ana, frances) \leftarrow$
$estudou(ana, ingles) \leftarrow$
$estudou(yves, ingles) \leftarrow$
$fala(A, C) \leftarrow nasceu(A, B), idioma(B, C)$
$fala(D, E) \leftarrow estudou(D, E)$ □

Exercício 3 *Codifique as cláusula a seguir em* Prolog *e consulte o sistema para descobrir quem é irmão de Cain (no* Prolog *o predicado de desigualdade é escrito como* `\=`*).*

$pai(adao, cain) \leftarrow$
$pai(adao, abel) \leftarrow$
$pai(adao, seth) \leftarrow$
$irmao(X, Y) \leftarrow pai(Z, X), pai(Z, Y), X \neq Y$ □

Exercício 4 *Codifique as cláusula a seguir em* PROLOG *e consulte o sistema para descobrir quem namora com quem e quem é infiel.*

$gosta(ary, eva) \leftarrow$
$gosta(ary, bia) \leftarrow$
$gosta(ivo, ana) \leftarrow$
$gosta(ivo, eva) \leftarrow$
$gosta(eva, ary) \leftarrow$
$gosta(ana, ary) \leftarrow$
$namora(A, B) \leftarrow gosta(A, B), gosta(B, A)$
$infiel(C) \leftarrow namora(C, D), gosta(C, E), D \neq E$ □

Exercício 5 *Imagine um contexto definido pelos seguintes predicados:*

- $mora(Pessoa, Bairro)$ *: relaciona uma pessoa ao bairro em que ela mora*
- $pertence(Bairro, Zona)$ *: relaciona um bairro à zona à qual ele pertence*
- $amigo(Pessoa, Pessoa)$ *: relaciona duas pessoas que são amigas*
- $tem\_carro(Pessoa)$ *: discrimina as pessoas que têm carro*

Crie uma coleção de fatos a respeito desses predicados (invente os dados) e defina uma regra estabelecendo que uma pessoa pode dar carona a outra se essa pessoa tem carro e ambas moram em bairros que ficam na mesma zona. Em seguida, faça consultas ao sistema PROLOG *e verifique se as respostas obtidas são coerentes com os dados declarados.* □

Referências

1. AMBLE, T. *Logic Programming and Knowledge Engineering*, Addison-Wesley, 1987.
2. BRATKO, I. *Prolog - Programming for Artificial Intelligence*, Addison-Wesley, 1990.
3. GENESERETH, M. R. AND NILSSON, N. J. *Logical Fundations of Artificial Intelligence*, Morgan Kaufmann Publishers, 1988.
4. REITER, R. *On Closed Word Data Bases*, In H. Gallaire and J. Minker, Logic and Data Bases, pages 55-76, Plenum Press, NY, 1978.
5. STERLING, L. AND SHAPIRO, E. *The Art of Prolog*, MIT Press, 1986.