Inteligência Artificial

Apontamentos para as aulas

Luís Miguel Botelho

Departamento de Ciências e Tecnologias da Informação
Instituto Superior de Ciências do Trabalho e da Empresa

Julho de 2015

Programação em Lógica para Representar Conhecimento Declarativo

Índice

Programação em Lógica para Representar Conhecimento Declarativo

A linguagem de programação mais disseminada que representa o paradigma da programação em lógica é o Prolog ("Programming in Logic"). O Prolog é uma linguagem baseada na lógica de predicados de primeira ordem (FOL, "First Order Logic", ou FOPC, "First Order Predicate Calculus"). A ideia mais importante subjacente ao Prolog é a chamada programação declarativa, isto é, uma forma de programar em que se representam relações entre entidades em vez de se representar os processos que permitem criar e transformar entidades. Há várias abordagens à programação declarativa, incluindo a programação em lógica de que o Prolog é exemplo e a programação por regras.

Apesar de ter sido criada como uma linguagem essencialmente declarativa, o Prolog também possui mecanismos de controlo semelhantes aos das outras linguagens (e.g., JAVA e C++), incluindo mecanismos de repetição e outros.

De momento, estes apontamentos não incluem a explicação da linguagem Prolog, a qual deve ser procurada em livros aconselhados para o efeito, por exemplo, "Programming in Prolog: Using the ISO Standard (Fifth Edition)" de William Clocksin e Christopher Mellish [Clocksin e Mellish 2003] e "Prolog Programming for Artificial Intelligence" de Ivan Bratko [Bratko 2000].

Neste ponto do programa, será apresentada apenas uma porção simples (sem recursividade) da parte declarativa da linguagem Prolog. Mais adiante, no programa, será apresentado o resto da parte declarativa do Prolog (essencialmente, recursividade). Depois serão ainda apresentados os mecanismos de controlo e os mecanismos de *input/output* da linguagem.

1 O Paradigma da Programação em Lógica

Apenas a título introdutório, seguem-se algumas noções de representação explícita de conhecimento declarativo em Prolog, como exemplo de linguagem de programação em lógica.

A Programação em Lógica é método de representação de conhecimento baseado na lógica matemática. A linguagem mais representativa da Programação em Lógica é o Prolog (*Programming in Logic*), criada em 1972 na Universidade de Marselha com base em teoria desenvolvida na Universidade de Edimburgo.

Antes de passar à representação de conhecimento através da Programação em Lógica, usando Prolog, apresenta-se um breve enquadramento da Programação em Lógica no panorama dos paradigmas de programação.

Há vários critérios para classificar linguagens e paradigmas de programação. Sem discutir o interesse de cada um dos critérios e das classificações que origina, pode dizer-se que é comum falar dos seguintes tipos de programação:

- Programação imperativa;
- Programação centrada em objetos; e

- Programação declarativa.

Apesar de bastante comum, esta classificação pode não ser considerada exaustiva e pode mesmo ser considerada errada. Há quem defenda que existem outros tipos de programação, por exemplo, a programação centrada em eventos. Há ainda quem ache que a programação centrada em objetos é um caso particular da programação imperativa.

Na programação imperativa, um programa é uma sequência de comandos, de ordens ao computador: *faz isto, faz aquilo, se isto for verdade faz assim, se não for, faz assado*. Os conceitos fundamentais aqui são sequência e comandos, i.e., ordens. Um programa, numa linguagem imperativa (por exemplo em C, mas não em C++ nem em C#), depois de traduzida em linguagem máquina, é diretamente executado pelo computador.

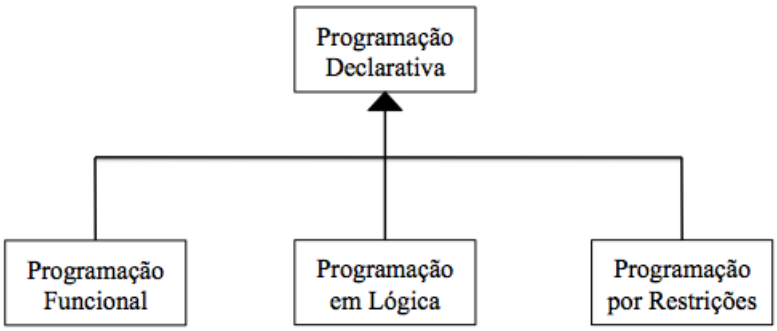
Na programação declarativa, um programa é um conjunto de declarações: *isto define-se desta forma, isto é verdade, isto é falso, isto satisfaz esta restrição, etc*. O conceito fundamental é a declaração; a sequência não faz sentido. De facto, uma afirmação, uma definição, ou o enunciado de uma restrição não contém em geral a noção de sequência.

Também é bastante comum dizer-se que programação declarativa se distinguee da programação imperativa no sentido em que apenas especifica *o quê* em vez de especificar *como*.

Na programação centrada em objetos, um programa é uma coleção de objetos que se relacionam de alguma forma, nem que seja através da sua ação. Cada objeto define-se pelos seus atributos ou propriedades, e pelos seus comportamentos. Quando se define um objeto, indicando e caracterizando os seus atributos ou propriedades, está a usar-se um tipo de abordagem declarativa à programação. Quando se diz que um dado objeto ou uma dada classe de objetos tem um conjunto comportamentos, também se está no âmbito da definição e não no âmbito das sequências de comandos. No entanto, a definição dos processos computacionais que implementam esses comportamentos é, em várias linguagens (e.g., C++, C#, Java), feita recorrendo a uma sequência de ordens / comandos, i.e., recorrendo a uma abordagem imperativa; e é, noutras linguagens (e.g., CLOS, Common Lisp Object System), feita através de uma abordagem declarativa.

Na programação centrada em eventos, pode dizer-se que um programa especifica os processos computacionais que devem ser desencadeados quando ocorrem determinados eventos: se ocorrer o evento x, desencadeias o processo computacional p, quando ocorrer o evento y, desencadeias o processo computacional q. Neste tipo de especificação, a noção de sequência não é importante. No entanto, há ordens que se dão (“desencadeias o processo p”). A especificação de cada um dos processos computacionais a serem desencadeados quando determinado evento ocorre recorre frequentemente a< um tipo de programação imperativa, embora haja também abordagens diferentes.

Com o intuito de nos aproximarmos da Programação em Lógica, diremos que se trata de um dos tipos de programação declarativa. A figura seguinte mostra a organização habitualmente apresentada da programação declarativa.



Na programação funcional pura, um programa é constituído apenas por um conjunto de definições de funções. Cada função é definida sempre à custa de outras funções. O Lisp (LISt Processing) é a linguagem funcional mais disseminada.

Na programação em lógica, um programa é um conjunto de afirmações sobre as entidades do domínio da aplicação. Em geral, cada conceito é definido por um subconjunto dessas afirmações. Por exemplo, "uma mãe é uma mulher que tem pelo menos um filho", "o Rui é um homem", e "um descendente de uma pessoa é um filho dessa pessoa, ou é um descendente de um filho". O Prolog (PROgramming in LOGic) é a linguagem mais representativa e conhecida de programação em lógica. O Prolog baseia-se num subconjunto da forma clausal da lógica de predicados de primeira ordem. As cláusulas do Prolog chamam-se cláusulas de Horn (cláusulas com, no máximo, um literal positivo).

Na programação por restrições, um programa é um conjunto de restrições aplicáveis às várias entidades do domínio da aplicação. A programação por restrições não é tão divulgada como as programações funcional e em lógica. O Prolog III é uma das linguagens de uma variedade de Programação por Restrições chamada Programação em Lógica com Restrições (Constraint Logic Programming).

Depois desta breve contextualização, passamos á apresentação da representação de conhecimento através da programação em lógica, usando Prolog.

2 Representação de conhecimento em Prolog

2.1 Apresentação inicial da linguagem Prolog

Base de conhecimentos

Uma base de conhecimentos é um conjunto de cláusulas

Cláusulas: factos e regras

Execução

Objetivos (*"goals"*, perguntas)

Exemplo

No ficheiro

```
  homem(joao).                  mulher(ana).
  homem(antonio).               mulher(maria).
  homem(luis).                  mulher(rita).
  filho(ana, joao).             filho(ana, maria).
% X é pai se X for um homem e existir pelo menos um filho de X
pai(X):- homem(X), filho(Y, X).
```

No interpretador...

```
?- homem(joao)
yes
?- homem(X).
X = joao;
X = antonio;
X = luis;
no
?- filho(X, joao).
X = ana
yes
?- pai(X).
X = joao
yes
```

2.2 Exemplos de representação

Procura-se que os alunos percebam o funcionamento do Prolog, em especial os conceitos de *backward chaining* e de *backtracking*, através destes exemplos.

A capital de um pais é uma cidade

$\forall_x, \forall_p$ Pais(p) $\wedge$ Capital(p) = x $\Rightarrow$ Cidade(x)

Em Prolog não há funções...

cidade(X):- pais(P), capital(X, P).

pais(portugal).

pais(franca).

pais(etiopia).

capital(portugal, lisboa).

capital(franca, paris).

capital(etiopia, adis_abeba).

?- cidade(X).

Maior de dois números

$\forall_{x,y}$ Numero(x) $\wedge$ Numero(y) $\wedge$ x $\geq$ y $\Rightarrow$ Maior(x, y) = x

$\forall_{x,y}$ Numero(x) $\wedge$ Numero(y) $\wedge$ x < y $\Rightarrow$ Maior(x, y) = y

O Prolog não tem funções; e há predicados que não geram soluções; apenas verificam se algo é verdade.

% maior(X, Y, Z) – Z é o maior dos dois números X e Y.

% number(X) – Predicado pré-construído no Prolog que verifica se X é um número; não gera números.

maior(X, Y, X):- number(X), number(Y), X >= Y.

maior(X, Y, Y):- number(X), number(Y), X < Y.

?- maior(7, 2, X).

X = 7

?- maior(2, 7.5, X).

X = 7.5

?- maior(X, Y, Z).

no

Mas...

termo(1).

termo(7).

termo(a).

termo([10, 30, 4]).

termo(5).

?- termo(X), termo(Y), maior(X, Y, Z).

2.3 Interpretadores Prolog

Existem vários interpretadores / compiladores Prolog. Por seguir o *standard* ISSO Prolog e por ser *Open Source*, o SWI Prolog é atualmente o interpretador usado nas aulas de Inteligência Artificial e de Tecnologias para Sistemas Inteligentes. Existe SWI Prolog para ambientes Windows, Mac OS X, e Linux. No entanto há muitos outros interpretadores pagos e gratuitos da linguagem Prolog.

2.4 Exemplos de sistemas

Pretende mostrar-se a aplicação da Programação em Lógica a problemas que poderiam ser bem resolvidos pela utilização de Sistemas Baseados em Conhecimento.

Pen USB e documentos

```
% cabe(Doc, Pen) – Doc cabe na Pen
cabe(Doc, Pen):-
  penUSB(Pen, ELivre),
  ficheiro(Ficheiro, Tamanho),
  ELivre >= Tamanho.
penUSB(pen1, 2000).% 2000 MBytes (2 GBytes)
penUSB(pen2, 3).   % 3 MBytes
ficheiro(doc1, 0.5).  % 5.5 MBytes (500 KBytes)
ficheiro(doc2, 3.5).
ficheiro(doc3, 0.25).
?- cabe(Doc, Pen).
```

Podemos tornar este sistema mais realista se o predicado *penUSB/2* computar o espaço livre na *pen*, com base nos tamanhos dos ficheiros armazenados.

Os seguintes factos ilustram o que se pretende

armazenado(doc1, pen1).

armazenado(doc4, pen1).

armazenado(imagem1, pen1).

....

A solução do problema passaria por criar um sistema que computasse o espaço ocupado por todos os ficheiros armazenados, mas isso não pode ser feito hoje porque não foi dada a matéria suficiente.

Sistema de avaliação de candidatos

Numa empresa, pretende desenvolver-se um sistema para ajudar a identificar os trabalhadores mais adequados para determinadas tarefas. Esse sistema é constituído por várias partes. Aqui, analisa-se uma parte do sistema que apenas identifica trabalhadores para a tarefa, mas não os ordena.

```
% Regras para selecionar candidatos
candidato(A, T) :-
  trabalhador(A),
  tarefa(T),
  disponivel(A, T),
  capacidade(A, T),
  interesse(A, T).
disponivel(A, T) :-
  tempo_disp(A, T),% Tempo disponivel
  permitido(A, T).   % A pode efectuar T
```

```
tempo_disp(A, T) :-
  horas_contratuais(A, N),
  horas_atribuidas(A, M),
  Disponivel is N - M,
  nHoras(T, HorasTarefa),
  Disponivel >= HorasTarefa.
% Factos
% Tarefas
tarefa(testp)  % Teste sistemático de programas de computador
tarefa(intrd)  % Introdução de dados num sistema de documentação
tarefa(secr) % Secretariado
% Horas necessárias para as tarefas
nHoras(testp, 20).
nHoras(intrd, 25).
nHoras(secr, 15).
% Empregados: Alves, Matos
% Alves
trabalhador(alves).
horas_contratuais(alves, 40).
horas_atribuidas(alves, 20).
permitido(alves, testp).
capacidade(alves, testp).
interesse(alves, testp).
% Matos
trabalhador(matos).
horas_contratuais(matos, 40).
horas_atribuidas(matos, 15).
permitido(matos, secr).
capacidade(matos, secr).
interesse(matos, secr).
```

Este sistema constitui una simplificação significativa da realidade. De acordo com o conhecimento explicitamente representado, um trabalhador tem disponibilidade para a tarefa se tem tempo disponível e se for permitido. A permissão é, nesta simplificação, representada por factos. Num sistema mais realista, seria necessário confrontar a tarefa e as atividades que ela envolve com o contrato de trabalho e eventualmente com a lei geral.

A capacidade de um candidato para uma tarefa é também um facto. Talvez um sistema mais realista tivesse de confrontar os requisitos da tarefa com as aptidões e características do candidato.

3 Referências

[Bratko 2000] Bratko, I. 2000. “Prolog Programming for Artificial Intelligence. Third Edition”. Pearson Addison Wesley

[Clocksin e Mellish 2003] Clocksin, W.F.; e Mellish, C.S. 2003. Programming in Prolog: Using the ISO Standard (Fifth Edition). Springer