Ficha Prática 10

10.1 Objectivos

1. Praticar a utilização de `assert` e `retract`.

10.2 Conceitos

O Prolog permite a manipulação da base de conhecimento durante a execução de um programa. Para tal disponibiliza dois meta-predicados base: `assert` e `retract`.

Outra utilização que pode ser dada a estes meta-predicados é na simulação de variáveis globais. Em Prolog existem apenas variáveis locais (ao termo em que são utilizadas). Em situações muito específicas poderá ser útil recorrer a variáveis globais. Os meta-predicados `assert` e `retract` podem ser utilizados para simular estas variáveis.

10.2.1 `dynamic/1`

Antes de perceber os meta-predicados `assert` e `retract`, convém entender o conceito de predicados estáticos e predicados dinâmicos.

Por omissão, um predicado que seja carregado através do `consult` fica definido como estático. Ou seja, a sua definição não poderá ser alterada, não sendo possível adicionar ou remover factos e/ou regras a ela associados[11]. Para indicar que a definição de um dado predicado poderá ser alterada (através da adição/remoção de factos e/ou regras), o predicado deverá ser declarado como dinâmico. Isso é conseguido através da directiva `dynamic`.

Por exemplo, a inclusão da directiva:

```
:- dynamic primo/1.
```

no início de um ficheiro (em que o predicado `primo/1` é definido) permite que, após o *consult* do ficheiro, a definição de `primo/1` seja alterada.

10.2.2 `assert/1` (`asserta/1` e `assertz/1`)

O predicado `assert/1` permite adicionar factos ou regras à base de conhecimento. O facto/regra é adicionado como o último facto/regra do predicado em causa. O predicado, caso exista, tem que estar definido como dinâmico.

Por exemplo (e assumindo que os predicados `primo/1` e `impar/1` são dinâmicos ou não existem na base de conhecimento), após

```
assert(primo(5))
```

o facto "`primo(5).`" passará a constar na base de conhecimento (como o último facto referente ao predicado `primo/1`). Após

[11] Existe, no entanto, uma excepção a esta regra. Ver mais à frente.

```
assert(impar(N):-primo(N))
```

a regra "`impar(N):-primo(N).`" passará a constar na base de conhecimento (como a última regra referente ao predicado impar).

Predicados adicionados à base de conhecimento utilizando `assert/1` ficam definidos como dinâmicos.

`asserta/1`

Semelhante a `assert/1`, mas o facto/regra é adicionado como o primeiro facto/regra do predicado.

`assertz/1`

Equivalente a `assert/1`.

10.2.3 `retract/1` (`retractall/1` e `abolish/1`)

O predicado `retract/1` permite remover factos ou regras da base de conhecimento. O predicado a que diz respeito o facto/regra a ser removido tem que estar definido como dinâmico. Como parâmetro deverá ser passado um termo que unifique com o facto ou regra a remover.

Considere

```
retract(primo(X))
```

o primeiro facto que unifique com "`primo(X)`" será removido da base de conhecimento.

`retractall/1`

Semelhante a `retract/1`, mas **todos** os factos ou regras cuja **cabeça** unifique com o termo passado como parâmetro são removidos da base de conhecimento. Note que neste caso, para remover regras, não é preciso indicar toda a regra, mas apenas a sua cabeça.

`abolish/1`

A expressão `abolish(impar/1)` remove da base de conhecimento todos os factos e regras com functor `impar` e aridade 1. Os atributos definidos para o predicado (por exemplo, se é dinâmico) são também removidos.

Atenção: no SWI-Prolog este meta-predicado remove factos e regras mesmo de predicados estáticos.

10.3 Exercícios

1. Com base no que sabe sobre os predicados `assert`, `retract`, `retractall` e `abolish` procure prever o resultado das queries sublinhadas:

   (a)
   ```
   ?- [user].
   impar(1). impar(2). impar(3).
   ^D
   ?- assert(par(2)).
   ?- assert(par(3)).
   ?- assert(par(4)).
   ?- listing(impar).
   ?- listing(par).
   ?- assert(impar(5)).
   ```

```
?- retract(par(3)).
?- retract(impar(2)).
```

(b)

```
?-  [user].
:- dynamic m2/1, m3/1.
m2(2). m2(4). m2(6).
m3(3). m3(6). m3(8).
^D
?- retract(m3(8)).
?- assert(m2(8)).
?- assert(m3(9)).
```

(c)

```
?-  [user].
:- dynamic m23/1.
m23(X) :- m2(X).
m23(X) :- m3(X).
nãoComum(X) :- m2(X), not(m3(X)).
nãoComum(X) :- m3(X), not(m2(X)).
^D
?- assert(comum(X):- m2(X), m3(X)).
?- retract(comum(_)).
?- retract(comum(_):- m2(X), m3(X)).
?- retractall(m23(_)).
?- retractall(nãoComum(_)).
?- abolish(nãoComum(_)).
```

2. Escreva o predicado `regista_pares/1` que, dada uma lista de número inteiros, cria na base de conhecimento o facto `pares/1` cujo parâmetro é uma lista com os números pares presentes na lista original.

3. Escreva o predicado `regista_repetidos/1` que, dada uma lista, cria na base de conhecimento os factos `repetidos/1` e `repeticao/2`. O parâmetro de `repetidos/1` é uma lista com os elementos repetidos da lista original (a lista gerada não deverá ter repetições). Os factos do predicado `repeticao/2` registam, para cada um dos elementos repetidos, quantas vezes ele aparece repetido.

4. Relembre a base de conhecimento da secção 1.3. Escreva o predicado `gera_tios/0` que cria factos `tio/2`, que definem a relação "tio de", com base na informação presente na base de conhecimento.

5. Relembre o predicado `aluno/2` definido na secção 1.4.2. Escreva o predicado `gera_turmas/0` que cria, na base de conhecimento, factos `alunos/2` associando, a cada disciplina, a lista de alunos nela inscritos.

6. Relembre os exercícios da secção 4.3.2. Refaça os predicados para as funções de Fibonacci e Ackerman por forma a que reutilizem valores já calculados. Compare as capacidades de cálculo das versões originais com as das versões agora desenvolvidas.