Manipulação da Base de Conhecimento

Um programa Prolog pode ser visto como uma base de dados que especifica um conjunto de relações (de forma explícita através dos factos e implícita através das regras).

O Prolog tem predicados pré-definidos que permitem fazer a manipulação da base de conhecimento. Ou seja, predicados que permitem acrescentar e / ou retirar factos e regras da base de conhecimento, durante a execução de um programa.

Os predicados que estão definidos nos ficheiros que são carregados (por `consult`) são, por omissão, ***estáticos***. Ou seja, é impossível o programa alterar dimâmicamente tais predicados.

Para que seja possível alterar dinâminamente predicados carregados de ficheiro, esses predicados deverão ser declarados como ***dinâmicos*** no ficheiro em que está definido. Isso é feito incluindo no ficheiro a directiva:

```
:- dynamic PredicateName/Arity.
```

Os predicados que não estão, à partida, definidos são considerados dinâmicos

Adicionar factos e regras

Para adicionar factos e regras à base de conhecimento podem-se usar os predicados:

`assert/1`	acrescenta o facto/regra como *último* facto/regra do predicado
`asserta/1`	acrescenta o facto/regra como *primeiro* facto/regra do predicado
`assertz/1`	igual a `assert/1`

Exemplos:

amigos.pl

```
:- dynamic amigos/2.
:- dynamic mas/1.

amigos(ana,rui).
amigos(pedro,rui).
amigos(maria,helena).
amigos(pedro,ana).

mas(rui).
mas(pedro).

fem(ana).
fem(maria).
fem(helena).
```

```
| ?- ['amigos.pl'].
% consulting ...
yes
| ?- asserta(amigos(joao,helena)).
yes
| ?- assertz(mas(joao)).
yes
| ?- assertz(fem(isabel)).
! Permission error: cannot assert static user:fem/1
! goal:  assertz(user:fem(isabel))

| ?- assertz((amiga(X,Y) :- amigos(X,Y), fem(X))).
true ?
yes
| ?- assert((amiga(X,Y) :- amigos(Y,X), fem(X))).
true ?
yes
```



```
| ?- listing.
amiga(A, B) :- amigos(A, B), fem(A).
amiga(A, B) :- amigos(B, A), fem(A).

amigos(joao, helena).
amigos(ana, rui).
amigos(pedro, rui).
amigos(maria, helena).
amigos(pedro, ana).

fem(ana).
fem(maria).
fem(helena).

mas(rui).
mas(pedro).
mas(joao).

yes
| ?-
```

Se quiser ver apenas alguns predicados pode usar o `listing/1`.

Exemplos:

```
| ?- listing(fem).
```

```
| ?- listing(amigos/2).
```

```
| ?- listing([amiga,mas/1]).
```

Note que *este programa é volátil*. Quando sair do interpretador os novos factos e regras perdem-se.

Remover factos e regras

Para remover factos e regras da base de conhecimento podem-se usar os predicados:

`retract/1`	remove da base de conhecimento a *primeira* cláusula (facto ou regra) que unifica com o termo que é passado como parâmetro.
`retractall/1`	remove da base de conhecimento *todos* os factos ou regras cuja **cabeça** unifique com o termo que é passado como parâmetro.
`abolish/1`	remove da base de conhecimento *todos* os factos e regras com o functor/aridade que é passada como parâmetro.
`abolish/2`	semelhante a `abolish/1`, mas passando o nome do functor e a sua aridade separadamente.

Exemplo:

Considere o novo ficheiro

AMIGOS.pl

```
:- dynamic amigos/2, mas/1.

amigos(ana,rui).
amigos(pedro,rui).
amigos(maria,helena).
amigos(pedro,ana).
amigos(X,Y) :- amigos(Y,X).

mas(rui).
mas(pedro).

fem(ana).
fem(maria).
fem(helena).
```

```
| ?- ['AMIGOS'].
% consulting ...
| ?- retract(amigos(_,rui)).
yes
| ?- retract(fem(_)).
! Permission error: cannot retract static user:fem/1
! goal:  retract(user:fem(_79))
| ?- abolish(fem/1).
yes
| ?- listing.
amigos(pedro, rui).
amigos(maria, helena).
amigos(pedro, ana).
amigos(A, B) :- amigos(B, A).

mas(rui).
mas(pedro).
yes
| ?- retractall(amigos(pedro,_)).
yes
| ?- listing.
amigos(maria, helena).

mas(rui).
mas(pedro).
yes
| ?- abolish(mas,1).
yes
```

Nota: `abolish` remove mesmo predicados estáticos !

Exercícios:

1. A informação referente aos horários das salas de aula pode estar guardada na base de conhecimento em factos da forma `sala(num,dia,inicio,fim,discipl,tipo)`

```
:- dynamic sala/6.

sala(cp1103, seg, 10, 13, aaa, p).
sala(cp2301, ter, 10, 11, aaa, t).
sala(di011, sab, 12, 10, xxx, p).  % com erro
sala(cp3204, dom, 8, 10, zzz, p).
sala(di011, sex, 14, 16, xxx, p).
sala(cp204, sab, 15, 17, zzz, tp).
sala(di011, qui, 14, 13, bbb, tp). % com erro
sala(di104, qui, 9, 10, aaa, tp).
sala(dia1, dom, 14, 16, bbb, t).
sala(cp1220, sab, 14, 18, sss, p).
```

a) O seguinte predicado, permite retirar da base de dados todas marcações de sala em que, erradamente, a hora de início da aula é superior à hora de fim.

```
apaga_erros :- sala(N,D,Hi,Hf,C,T), Hf =< Hi,
               retract(sala(N,D,Hi,Hf,C,T)), fail.
apaga_erros.
```

Qual é o efeito da segunda clausula `apaga_erros` ?

b) Execute um programa que retire todas as marcações de salas para os domingos.

c) Execute um programa que retire todas as marcações de salas para os sábados depois das 13 horas.

d) Defina o predicado `ocupada(+NSala,+Dia,+Hora)` que sucede se a sala número `Nsala` está ocupada no dia `Dia` à hora `Hora`.

e) Defina o predicado `livre(?NSala,+Dia,+Hora)` que sucede se a sala número `NSala` está livre no dia `Dia` à hora `Hora`.

f) Defina o predicado `livres(-Lista,+Dia,+Hora)` que sucede se `Lista` é a lista dos números das salas que estão livres no dia `Dia` à hora `Hora`.

g) Defina o predicado `total_ocupacao(+NSala,-Total)` que sucede se `Total` é o número total de horas que a sala `NSala` está ocupada.

h) Defina o predicado `cria_total(+NSala)` que acrescenta à base de dados um facto, associando à sala `NSala` o número total horas de ocupação dessa sala por semana.

i) Defina o predicado `intervalo_livre(?NSala,+Dia,+Inicio,+Fim)` que sucede se `NSala` está livre no dia `Dia` durante o periodo de tempo entre `Inicio` e `Fim`.

2. Assuma que a informação referente às notas dos alunos à disciplina de *Lógica Computacional 2005/06* está guardada na base de conhecimento em factos da forma:

`modalidadeA(`*numero, nome, fichas, exame*`)`
`modalidadeB(`*numero, nome, fichas, trabalho, exame*`)`

Por exemplo:

```
modalidadeA(1111, 'Maria Campos' , 14, 12).
modalidadeA(3333, 'Rui Silva', 13, 15).
modalidadeA(4444, 'Paulo Pontes', 17, 12).
modalidadeA(8888, 'Antonio Sousa', 14, 8).

modalidadeB(2222, 'Ana Miranda', 14, 15, 12).
modalidadeB(5555, 'Joao Ferreira', 15, 16, 11).
```

a) Escreva o predicado `gera_notas/0` que cria factos `nota/3`, que definem a relação entre o aluno (número e nome) e a sua nota final (calculada de acordo com o estabelecido para esta disciplina).

b) Defina o predicado `aprovados(-Lista)` que sucede se `Lista` contem o número dos alunos aprovados à disciplina.