Más functores

En muchos problemas tenemos que manejar un grupo de datos juntos, relacionados. Tenemos una base de hechos sobre recursos guardados en dos servidores: Gutenberg y Perseus.

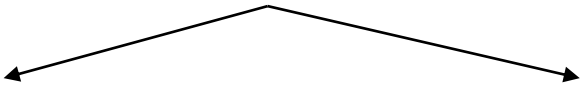
```
hostea(gutenberg, libro, leyendas, becquer, espaniol).
hostea(gutenberg, libro, alice_in_wonderland, tenniel, ingles).

hostea(gutenberg, imagen, ilustraciones_alicia, tenniel).
hostea(gutenberg, imagen, arte_rupeste_frances, pedro_picapiedra).

hostea(perseus, libro, iliada, homero, griego).
hostea(perseus, libro, metamorfosis, kafka, ingles).
```

La primera línea se lee: *se hostea en el servidor Gutenberg el libro digital "Leyendas" (de Bécquer) en español.*

Y tiene la forma:



Si es un libro: *hostea(Server, Tipo de Recurso, Titulo, Autor, Idioma)*	Si es una imagen: *hostea(Server, Tipo de Recurso, Titulo, Autor).*

etc... para cada tipo de recurso que haya.

Esta forma de representar la información es incómoda, tengo un montón de datos relacionados pero dispersos. Para mí el átomo "espaniol" no tiene sentido en sí mismo, sino como parte de la información relacionada con un libro. Esta forma de representar está lejos de lo que yo quiero decir. Por ejemplo, ¿cómo hago una que me asocie contenidos iguales hosteados tanto por gutenberg como por perseus?

```
estaEnAmbos1(libro, Titulo, Autor, Idioma) :-
    hostea(gutenberg, libro, Titulo, Autor, Idioma),
    hostea(perseus, libro, Titulo, Autor, Idioma).

estaEnAmbos2(imagen, Titulo, Autor) :-
    hostea(gutenberg, imagen, Titulo, Autor),
    hostea(perseus, imagen, Titulo, Autor).
```

No está copado, son dos predicados, no puedo resolverlo con un solo predicado.

- ¿Cómo puedo asociar los átomos/la información que está relacionada? ¿Cómo puedo hacer un individuo compuesto? Puedo usar **Functores**.

```
hostea(gutenberg, libro(leyendas, becquer, espaniol)).
hostea(gutenberg, libro(metamorfosis, kafka, ingles)).
hostea(gutenberg, libro(alice_in_wonderland, tenniel, ingles)).
hostea(gutenberg, imagen(ilustraciones_alicia, tenniel)).
hostea(gutenberg, imagen(arte_rupeste_frances, pedro_picapiedra)).
hostea(perseus, libro(iliada, homero, griego)).
hostea(perseus, libro(metamorfosis, kafka, ingles)).
```

De esta forma el predicado hostea tiene la forma: *hostea(Server, Recurso).* Los functores me permiten tener un grupo de individuos *relacionados* y ponerle un nombre.

Un functor no es un predicado, aunque su notación se le parezca. Un functor denota un individuo (compuesto, si se quiere), no una regla o un hecho. Los functores relacionan átomos, otros functores o listas…

Functores y polimorfismo

Puedo definir una regla que asocie contenidos iguales hosteados tanto por gutenberg como por perseus:

```
estaEnAmbos(X):-
        hostea(gutenberg, X),
        hostea(perseus, X).
```

X vale para libros o para imágenes. Entonces trato a los libros y a las imágenes en un determinado contexto como “intercambiables” o **polimórficos**. Recuerden ese concepto para cuando lleguemos a objetos.

Consultamos:

```
?- estaEnAmbos(X).
X = libro(metamorfosis, kafka, ingles).
```

¿Para qué nos sirve trabajar juntos a las imágenes y a los libros? Bueno, en principio podemos saber qué servidor tiene más recursos, sin necesidad de distinguir si son libros o imágenes.

- ¡¡¡¡Importante!!!! *¡Puedo usar pattern matching dentro de un functor!*

Pregunto por los libros en español que están en el servidor Gutenberg:

```
?- hostea(gutenberg, libro(Titulo, Autor, espaniol)).
```

Como espaniol es el tercer componente del functor debemos colocar variables (pueden ser anónimas) en los primeros dos lugares. No hay separación en cabeza/cola como las listas. Los functores NO son listas. **La cantidad de argumentos, o sea la aridad, es importante**.

Si tuviese:

```
hostea(gutenberg, imagen(ilustraciones_alicia, tenniel)).
hostea(gutenberg, imagen(arte_rupeste_frances))
```

tendría que consultar de dos maneras posibles para preguntar por los nombres de las imágenes en Gutenberg:

```
?- hostea(gutenberg,imagen(X,_))
X = arte_rupeste_frances
?- hostea(gutenberg,imagen(X))
X = ilustraciones_alicia
```

Por eso es importante fijarse en la aridad de los functores. No conservar uniformidad en los componentes de un functor hará que las consultas necesiten distinguir por un functor u otro.

Otros ejemplos:

Desarrollamos un predicado que indica en qué idiomas hay material en un determinado servidor:

```
idiomaDisponible(Server, Idioma):- hostea(Server, libro(_, _, Idioma)).
```

Desarrollamos ahora un predicado que resuelve qué autor es dueño tanto de un libro como de una imagen (como es en el caso que ilustre su propio libro):

```
autorIlustracionYLibro(Autor):-
    hostea(_, libro(_, Autor, _)),
    hostea(_, imagen(_, Autor)).

?- autorIlustracionYLibro(Autor).
   Autor = tenniel
```

Predicados de orden superior

Si la cuantificación es sobre una variable → el predicado es de primer orden.
"¿Cuántos años tiene Juan?",
"¿Hay algún alumno que tenga más de 60 años?",
"¿Nicolás aprobó Paradigmas?"

Si la cuantificación es sobre un predicado → estamos hablando de un predicado de orden superior:
"¿es falso que (hay un alumno que tiene más de 60 años)?"
→ `alumno(Alumno), not((edad(Alumno, Edad), Edad > 60))` [1]

¿**not/1** es sobre un individuo o sobre un predicado? Sobre un predicado. Entonces hay un nivel más de abstracción. Predicados que trabajan sobre predicados.

También vimos otra consulta que nos lleva a un predicado de orden superior:
"Armame una lista con (todas las chicas menores de 25)" → `findall/3`

Y más adelante vamos a conocer la forma de resolver otra consulta:
"quiero saber si todos los programas de televisión cumplen que (son sensacionalistas)" → `forall/2`

2 cosas que nos interesa decir sobre los predicados de orden superior:

- Asociamos el orden superior con un grado mayor de abstracción.
- Yo puedo definir mis propios predicados de orden superior, pero por ahora nos vamos a quedar con estos, que son los más importantes y vienen predefinidos.

Pero vamos por partes…

Tengo una base de conocimientos con la siguiente información:

```
mujer(maria).
mujer(juana).
mujer(ana).
mujer(luisa).
mujer(ivanna).
```

[1] Los paréntesis son requeridos, el not es de aridad 1

```
edad(maria, 20).
edad(juana, 17).
edad(ana, 36).
edad(luisa, 52).
edad(ivanna, 32).
```

Quiero definir la regla salirCon(Edad, MujeresPosibles), donde voy a salir con chicas que me lleven una diferencia menor a seis años.

Primer dilema: no tengo la información en formato de lista. ¿Qué puedo hacer?

Tengo que pasar de:

```
mujer(maria).
mujer(juana). etc.
```

A algo del estilo:

```
mujeres([maria, juana, … ]).
```

Bueno, para eso tengo:

findall/3

Busco todos los x que cumplen p(x). O sea, busco todas las variables que cumplan un predicado. El tercer argumento es unificable a la lista de las soluciones posibles:

```
?- findall(Mujer, mujer(Mujer), Mujeres)
Mujeres = [maria, juana, ana, luisa, ivanna] .
```

Puedo también pedir
`findall(ivanna, mujer(ivanna), Mujeres).`
Eso me devuelve la lista compuesta por
`Mujeres = [ivanna].`

Y qué pasa si pido una mujer que no existe en la base:

```
?- findall(laura, mujer(laura), Mujeres).
[]
```

Devuelve una lista vacía.

Bien, una vez que tengo la lista de mujeres, sólo debería filtrar aquellas que no cumplen la edad correspondiente:

```
salirCon(Edad, MujeresPosibles):-
  findall(Mujer,
          mujerJoven(Edad, Mujer),
          MujeresPosibles).
```

¿Qué es esto de `mujerJoven`? Una cláusula que nos permita definir cuándo es una mujer lo suficientemente joven como para que yo salga con ella:

```
mujerJoven(Edad, Mujer):-
  edad(Mujer, EdadMujer),
  Diferencia is abs(Edad - EdadMujer),
  Diferencia < 6.
```

Predicados de orden superior con functores

Ahora veremos un par de ejemplos usando functores, listas y predicados de orden superior (ya que estamos). Para ver bien los ejemplos, agregamos esto a nuestra base de conocimiento:

```
hostea(gutenberg, libro(iliada, homero, ingles)).
hostea(gutenberg, libro(odisea, homero, ingles)).
```

Entonces:

Quiero obtener la lista de los libros (el functor completo) disponibles de un autor dado en un server dado, ej:

```
?- librosAutorEnServer(homero, gutenberg, Libros).
Libros = [libro(iliada, homero, ingles), libro(odisea, homero, ingles)]
```

Lo podemos resolver así:

```
librosAutorEnServer(Autor, Server, Lista):-
      findall(libro(Titulo, Autor, Idioma),
      hostea(Server, libro(Titulo, Autor, Idioma)), Lista).
```

¿Y si sólo quiero los títulos? Entonces, si quiero obtener la lista de los títulos disponibles de un autor dado en un server dado, ej:

```
?- titulosAutorEnServer(homero, gutenberg, L).
L = [iliada, odisea]
```

Puedo hacer:

```
titulosAutorEnServer(Autor, Server, Lista):-
      findall(Titulo, hostea(Server, libro(Titulo, Autor, _ )),Lista).
```

Como vemos, se puede usar pattern matching.

Ahora podemos relacionar los libros en un idioma hosteados en un servidor, pero generando un functor nuevo que tenga:

- El título del libro
- El autor

Ejemplo:

```
?- titulosAutorEnIdiomaEnServer(ingles, perseus, L).
L = [tituloAutor(metamorfosis,kafka)]
```

Mediante pattern matching es posible armar un functor nuevo: `tituloAutor`

```
titulosAutorEnIdiomaEnServer(Idioma, Server, Lista):-
    findall(tituloAutor(Titulo, Autor),
            hostea(Server, libro(Titulo, Autor,Idioma)),
            Lista).
```