Prácticas de Programación Declarativa

Pascual Julián Iranzo.

Departamento de Tecnologías y Sistemas de Información

Universidad de Castilla–La Mancha.

Primera versión: Primavera de 1998

Segunda versión: Otoño de 2004

Tercera versión: Verano de 2009

Cuarta versión: Primavera de 2014

LABORATORIO 1 (Prolog Puro, estructuras y unificación)

Para medir lo aprendido durante la Práctica 1, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab1.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab1.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab1.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 1.
El siguiente programa Prolog especifica una base de datos (deductiva) sobre relaciones familiares:

```
%%% HECHOS
padre(abraham,isaac).
padre(haran,lot).
padre(haran,melca).
padre(haran,jesca).
hombre(isaac).
hombre(lot).
mujer(melca).
mujer(jesca).

%%% REGLAS
ascendiente_directo(X, Y) :- (padre(X, Y); madre(X, Y)).

ascendiente(X, Z) :- ascendiente_directo(X, Z).
ascendiente(X, Z) :- ascendiente_directo(X, Y), ascendiente(Y, Z).

hijo(X,Y) :- hombre(X), ascendiente_directo(Y,X).
hija(X,Y) :- mujer(X), ascendiente_directo(Y,X).
```

Ampliar la base de conocimiento (hechos) anterior mediante la atenta lectura de estos fragmentos del Génesis:

``He aquí la descendencia de Teraj: Teraj engendró a Abram (posteriormente llamado `Abraham', que significa `Padre de multitud'), Najor, y Harán. Harán engendró a Lot ... La mujer de Abram se llamaba Sarai (o Sara) y la de Najor Melca, hija de Harán, padre de Melca y de Jesca."

``Sarai, la mujer de Abram, no le había dado hijos, pero ella tenía una esclava egipcia de nombre Agar. ... tomó a Agar y se la dio por mujer a Abram, ... Agar parió un hijo a Abram y a este hijo tenido de Agar, Abram le llamó Ismael."

Estando en la tierra de Guerar, Abram confeso que: `` es verdad que ella (Sarai) también es mi hermana, hija de mi padre pero no de mi madre, y ahora es mi mujer."

``Sara, pues, concibió y parió un hijo en su vejez, en el tiempo predicho por Dios. Y Abraham llamó al hijo que le nació Isaac (que significa `el que ríe')." Isaac casó con Rebeca, ``hija de Batuel, el que Melca parió a Najor" y hermana de Labán. Isaac y Rebeca tuvieron dos hijos, Esaú y Jacob, pero ésta es otra historia.

Sólo incluir como hechos los datos aportados sobre quien es hombre o mujer, padre o madre de alguien o casado con alguien, que se consideran los *símbolos primitivos* de nuestra representación. El resto de relaciones familiares definirlas como reglas, en función de las anteriores. Tomar como modelo las definiciones de los predicados hijo e hija.

a) Definir las relaciones: ascendiente de; descendiente de; abuelo(/a) de; hermano(/a) de; tío(/a) de; sobrino(/a) de; y primo(/a) de.

b) Definir un predicado, incestuosos(X,Y), que nos informe de las relaciones que hoy consideramos incestuosas.

c) Señalar las relaciones que sean directamente recursivas y las que sean indirectamente recursivas.

d) Qué predicados son inversibles?

Ejercicio 2.
Una base de datos puede representarse de forma natural en PROLOG como un conjunto de hechos. Por ejemplo, la información sobre una familia puede estructurarse como

```
familia(persona(antonio, foix, fecha(7, febrero, 1950), trabajo(renfe, 1200)),
        persona(maria, lopez, fecha(17, enero, 1952), trabajo(sus_labores, 0)),
       [persona(patricia, foix, fecha(10, junio, 1970), trabajo(estudiante, 0)),
        persona(juan, foix, fecha(30, mayo, 1972), trabajo(estudiante,0))] ).
```

Podemos ver la anterior relación como una de las filas de una tabla que almacena las informaciones de las familias. La tabla FAMILIA estaría compuesta por los campos: marido, esposa e hijos. Estos campos serían a su vez estructuras. Los campos marido y esposa son estructuras de tipo persona. El campo hijos está constituido por una lista de personas. Cada persona es una estructura de cuatro componentes: nombre, apellido, fecha de nacimiento y trabajo que desempeña.

Si representamos la información mediante estructuras, una propiedad agradable de PROLOG y su mecanismo de unificación es que podemos recuperar información de la base de datos mediante la formulación de objetivos en los que sus componentes no están completamente instanciados. Por ejemplo, el objetivo:

```
?- familia(P, M, H).
```

recupera la información sobre todas las familias en la base de datos y el objetivo

```
?- familia(P, M, H), P = persona(_, foix, _, _).
```

recupera la información sobre la familia Foix.

Ampliar la base de datos con los siguientes hechos:

```
familia( persona(manuel, monterde, fecha(15, marzo, 1934), trabajo(profesor, 2000)),
        persona(pilar, gonzalez, fecha(9, julio, 1940), trabajo(maestra, 1900)),
        [ persona(manolo, monterde, fecha(10, febrero, 1964), trabajo(arquitecto, 5000)),
          persona(javier, monterde, fecha(24, noviembre, 1968), trabajo(estudiante, 0)) ] ).
familia( persona(jose, benitez, fecha(3, septiembre, 1958), trabajo(profesor, 2000)),
        persona(aurora, carvajal, fecha(29, agosto, 1972), trabajo(maestra, 1900)),
        [ persona(jorge, benitez, fecha(6, noviembre, 1997), trabajo(desocupado, 0))] ).
```

```
familia( persona(jacinto, gil, fecha(7, junio, 1958), trabajo(minero, 1850)),
        persona(guillermina, diaz, fecha(12, enero, 1957), trabajo(sus_labores, 0)),
        [ persona(carla, gil, fecha(1, agosto, 1958), trabajo(oficinista, 1500)),
          persona(amalia, gil, fecha(6, abril, 1962), trabajo(deliniante, 1900)),
          persona(irene, gil, fecha(3, mayo, 1970), trabajo(estudiante, 0)) ] ).
familia( persona(ismael, ortega, fecha(7, junio, 1966), trabajo(carpintero, 2350)),
        persona(salvadora, diaz, fecha(12, enero, 1968), trabajo(sus_labores, 0)),
        [] ).
familia( persona(pedro, ramirez, fecha(7, junio, 1966), trabajo(en_paro,0)),
        persona(teresa, fuentes, fecha(12, enero, 1968), trabajo(administrativa, 1250)),
        [] ).
```

y responder a las siguientes preguntas planteando objetivos:
a) encontrar los nombres y apellidos de las mujeres casadas que tienen tres o más hijos;
b) encontrar los nombres de las familias que no tienen hijos;
c) nombres de las familias en las que la mujer trabaja pero el marido no.

[**Observación.**
Convertir los objetivos planteados en los anteriores subapartados en los predicados: ej2a(Nombre, Apellidos); ej2b(Apellidos_Padre, Apellidos_Madre) y ej2c(Apellidos_Padre, Apellidos_Madre). Introducir estos predicados en el fichero **Apellidos_Lab3.pl** .]

Ejercicio 3.
Utilizar la base de datos de la anterior forma supone que el usuario sea conocedor de la estructura interna de la representación. Para paliar esta situación, definir los procedimientos que facilitan la interacción con la base de datos:

PREDICADOS	SIGNIFICADO ESPERADO
padre(P)	**P** es un hombre casado. Queremos que **P** se instancie con los datos de la primera persona de la familia;
madre(P)	Queremos que **P** se instancie con los datos de la madre;
hijo(P)	Queremos que **P** se instancie con los datos de uno de los hijos;
existe(P)	**P** es una persona que está en la base de datos;
f_nacimiento(P, F)	**F** es la fecha de nacimiento de la persona **P**;
salario(P, S)	**S** es el salario de la persona **P**.

Una vez definidos los anteriores predicados y relaciones, contestar a las siguientes preguntas (formulando los objetivos correspondientes)

a) Encontrar los nombres de todas las personas en la base de datos;
b) Encontrar los hijos nacidos después de 1980;
c)Encontrar todas la mujeres empleadas;
d) Encontrar los nombres de las personas desempleadas nacidas antes de 1960;
e) Encontrar las personas nacidas después de 1950 cuyo salario esté comprendido entre las 800 y 1300 euros.
[**Observación.**
Convertir los objetivos planteados en los anteriores subapartados en los predicados: ej3a(Nombre, Apellidos); ej3b(Nombre_Hijo, Apellidos_Hijo), ej3c(Nombre_Mujer, Apellidos_Mujer), ej3d(Nombre, Apellidos), ej3e(Nombre, Apellidos). Introducir estos predicados en el fichero **Apellidos_Lab3.pl** .]

LABORATORIO 2 (Listas y Aritmética)

Para medir lo aprendido durante la Práctica 2, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab2.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab2.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab2.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 4.

Definir un predicado **sus(X,Y,L1,L2)** que sea capaz de sustituir un elemento **X** por otro **Y** en la lista **L1**, para dar **L2**. Procurar que el predicado definido sea inversible.

Ejercicio 5.

Definir la relación **aplanar(Lista, Aplanada)**, donde **Lista** es en general una lista de listas, tan compleja en su anidamiento como queramos imaginar, y **Aplanada** es la lista que resulta de reorganizar los elementos contenidos en las listas anidadas en un único nivel, i.e. una lista plana. Por ejemplo

```
?- aplanar([[a, b], [c, [d, e]], f], L).
L = [a, b, c, d, e, f]
```

Ejercicio 6.

Definir un predicado **igualesElem(L1,L2)** que compruebe que **L1** y **L2** son listas que contienen los mismos elementos independientemente del orden de aparición.
[**Ayuda:** utilizar el predicado **length/2**]

Ejercicio 7.

Definir un predicado **descomponer(N,A,B)** que permita resolver el problema de descomponer un número natural **N** en la suma de dos pares **A** y **B**. Esto es, **descomponer(N,A,B)** debe tomar como entrada un natural **N** y devolver dos naturales **A** y **B** tales que **N** = **A** + **B**.
[**Ayuda:** utilizar el predicado **between/3**]

Ejercicio 8.

Rompecabezas de Brandreth. El cuadrado de 45 es 2025. Notad que si partimos el número en dos obtenemos los números 20 y 25 cuya suma es, precisamente, 45. Obtener que otros números cuyo cuadrado es un número de cuatro cifras cumplen esta propiedad. Con este fin, definir un predicado **numBrandreth (N, C)** que devuelva uno de estos números **N** y su cuadrado **C**.
[**Ayuda:** los números N cuyo cuadrado es de cuatro cifras pueden generarse mediante una llamada al predicado **between(32, 99, N)**].

LABORATORIO 3 (Más sobre listas, aritmética y computación simbólica)

Para medir lo aprendido durante la Práctica 3, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab3.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab3.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab3.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 9.
Defina un predicado partir(L, L1, L2) que divida la lista L en dos partes L1 y L2, tales que los elementos de L1 son menores o iguales que un cierto elemento N perteneciente a L y los de L2 son mayores que ese elemento N. El elemento N seleccionado no se incluye en las listas partidas L1 y L2.

Ejercicio 10. (Ordenación rápida—quicksort—)
El algoritmo de ordenación rápida aplica la estrategia de "divide y vencerás" a la tarea de ordenar una lista. La idea es particionar una lista con respecto a uno de sus elementos (en principio elegido al azar), llamado el pivote, de forma que los elementos menores o iguales que el pivote queden agrupados a su izquierda, en una de las listas, y los elementos mayores que el pivote queden agrupados a su derecha, en la otra lista. Observe que, tras la partición, lo único seguro es que el pivote está en el lugar que le corresponderá en la ordenación final. Entonces, el algoritmo se centra en la ordenación de las porciones de la lista (que no están necesariamente ordenadas), lo que nos remite al problema original. Utilizando el predicado partir(L, L1, L2) del Ejercicio 10, dé una implementación recursiva del algoritmo de ordenación rápida, mediante la definición de un predicado: quicksort(Lista, ListaOrdenada).

Ejercicio 11. (Mezcla ordenada —merge–sort—)
Implemente en Prolog el algoritmo de mezcla ordenada para la ordenación de una lista de elementos, mediante la definición de un predicado: merge_sort(Lista, ListaOrdenada).
Informalmente, este algoritmo puede formularse como sigue: Dada una lista, divídase en dos mitades, ordene cada una de las mitades y, después, "mezcle" apropiadamente las dos listas ordenadas obtenidas en el paso anterior.

Ejercicio 12.
El polinomio $Cnx^n + : : : + C2x^2 + C1x + C0$, donde Cn; ... ;C1;C0 son coeficientes enteros, puede representarse en Prolog mediante el siguiente término:

cn * x ** n + ... + c2 * x ** 2 + c1 * x + c0.

El operador "**" es un operador binario infijo. Cuando se evalúa la expresión "X ** n", computa la potencia enésima de X. Observe que el operador "**" liga más que el operador binario infijo "*", que a su vez liga más que el operador binario infijo "+" (por lo tanto no se requiere el uso de paréntesis). Observe también que, en la representación anterior, la variable x se trata como una constante. Defina un predicado eval(P, V, R) que devuelva el resultado R de evaluar un polinomio P para un cierto valor V de la variable x. A modo de ejemplo, el objetivo ?- eval(5 * x ** 2 + 1, 4, R). debe tener éxito con respuesta R = 81.

Ejercicio 13.
Utilizando la representación de los polinomios propuesta en el ejercicio anterior, defina un predicado d(P, D) que compute la derivada D de un polinomio P con respecto a x. Se requiere que el polinomio D, que se obtiene como resultado, se presente en un formato simplificado. Esto es, si lanzamos el objetivo:

```
?- d(2 * x ** 2 + 3, D).
```

debe devolver la respuesta “D = 4 * x “ y no “D = 2* 2 * x ** 1 + 0“.

LABORATORIO 4 (Arboles)

Para medir lo aprendido durante la Práctica 4, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab4.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab4.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab4.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Los árboles n–arios de tipo T o rosadelfas se caracterizan por ser:

1. Bien una estructura vacía,
2. o bien una estructura constituida por un elemento de tipo T, denominado nodo, junto con m (siendo 0 =< m =< n) subconjuntos disjuntos de elementos; estos subconjuntos son a su vez árboles n–arios de tipo T, que se denominan subárboles del árbol original.

El "grado" de un nodo es su número de hijos y el "grado" un árbol, el grado máximo de sus nodos.

Ejercicio 14.
Las rosadelfas se han representado en Prolog mediante los constructores:

1. nil;
2. hoja(X), donde X elemento de tipo T;
3. nodo(X, [T_1, ..., T_M]), donde X es un elemento de tipo T, T_i es un árbol n--ario de tipo T, y M =< N.

El predicado esRosadelfa(O) permite determinar si un objeto O es o no una rosadelfa.

```
% esRosadelfa(O), el objeto O es una rosadelfa.
esRosadelfa(nil).
esRosadelfa(hoja(_)).
esRosadelfa(nodo(_, Rosadelfas)) :-  esListaRosadelfas(Rosadelfas).

esListaRosadelfas([R]) :- esRosadelfa(R).
esListaRosadelfas([R|Rosadelfas]) :- esRosadelfa(R), esListaRosadelfas(Rosadelfas).
```

Sin embargo, en la definición anterior no se hace ninguna comprobación del tipo de los elementos o sobre el número de hijos (o grado) de cada nodo. Modificar el predicado anterior y definir un predicado esRosadelfa(O, N) que determine si un objeto O es una rosadelfa de grado N y realice las comprobaciones de tipo oportunas.

Ejercicio 15.
Defina los siguientes predicados acerca de los árboles n–arios:

a) peso(A, P) que calcule el peso P de un árbol n–ario A, entendiendo por "peso" el número de nodos que contiene dicho árbol. Esta magnitud también recibe el nombre de "tamaño" del árbol.

b) grado(A, G) que calcule el grado G de un árbol n–ario A; el "grado" de un nodo es el número de hijos de ese nodo y el "grado" de un árbol es el grado máximo de los nodos que lo componen.

c) frontera(A, F) que permite determinar la frontera F del árbol n–ario A. La frontera

de un árbol es la lista de sus hojas.
d) preorden(A, L) que permita recorrer los nodos del árbol n–ario A en orden preorden, obteniendo la lista L de nodos visitados y en el orden que fueron visitados. Un recorrido preorden consiste en visitar primero la raíz y después los subárboles de izquierda a derecha.

Ejercicio 16.
Al definir el predicado construirRosadelfa(L, G, R) se pretende construir una rosadelfa R, de grado G a partir de los elementos de una lista L. Más abajo se presenta una solución que aparece en la página 218 del libro **PROGRAMACIÓN LÓGICA, TEORÍA Y PRÁCTICA**, cuyos autores son María Alpuente y Pascual Julián. En realidad, dicha solución no hace lo que se pretende. Ante un objetivo como:

?- construirRosadelfa([1,2,3,4,5,6,7,8,9,10,11,12,13], 2, R).

El interprete responde construyendo el árbol:

```
R =
nodo(1,
    [nodo(2, [hoja(3)]),
     nodo(4, [hoja(5)]),
     nodo(6, [hoja(7)]),
     nodo(8, [hoja(9)]),
     nodo(10, [hoja(11)]),
     nodo(12, [hoja(13)]),
     nil
    ]
   )
```

que no es de grado 2.

El objetivo de este ejercicio es modificar el código del programa para que se atenga a su especificación. Esto es construya una rosadelfa de grado G.

```
% construirRosadelfa(L, G, R), construye una rosadelfa R, a partir de los elementos de una lista L.
construirRosadelfa([], _, nil).
construirRosadelfa([X], _, hoja(X)).
construirRosadelfa([X|L], G, nodo(X, [R|Rosadelfas])):-
                                partir(L, G, [L1|GListas]),
                                construirRosadelfa(L1, G, R),
                                construirRosadelfas(GListas, G, Rosadelfas).

% inspecciona una lista de listas y por cada lista inspeccionada crea una rosadelfa
construirRosadelfas([], _, []).
construirRosadelfas([L|GListas], G, [R|Rosadelfas]) :-
                                construirRosadelfa(L, G, R),
                                construirRosadelfas(GListas, G, Rosadelfas).
% partir(L, G, [L1|GListas]), divide la lista L en una secuencia de listas de longitud
% menor o igual que G, que se devuelve como una lista de listas en el segundo argumento.
partir(L, G, [L]):- length(L,N), N < G.
partir(L, G, [L1|GListas]):-   length(L,N), N >= G,
```

```
    length(L1, G),
    append(L1, LL, L),
    partir(LL, G, GListas).
```

La definición del predicado partir, en su formulación actual, trata dos casos: i) si la longitud de la lista L es menor que G, entonces es un resto que debe unirse directamente a la secuencia de listas; ii) en caso contrario, se fragmenta la lista L en un prefijo de longitud G y un resto de lista LL, que se sigue partiendo, mediante la llamada recursiva a partir. Observe que la llamada length(L1,G) puede utilizarse para crear una lista L1 que contenga G variables, que posteriormente se instanciarán. Si lanzamos el objetivo "?- partir([1,2,3,4,5,6,7,8,9,10,11,12,13], 2, L).", se obtiene como respuesta:

L = [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12], [13]]

Ahí podría estar la fuente del error. Se sugiere modificar el comportamiento del predicado "partir" para que construya la lista:

L = [[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13]]

y el resto de predicados pueda generar una rosadelfa equilibrada de grado G=2.

LABORATORIO 5 (Parámetros de acumulación y eficiencia)

Para medir lo aprendido durante la Práctica 5, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab5.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab5.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab5.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 17.
Sean P1 y P2 dos programas que implementan una misma especificación, es decir, pueden considerarse semánticamente equivalentes. Si el programa P1 se ejecuta en un tiempo T1 y el programa P2 en un tiempo T2, el speedup S de P1 con respecto a P2 se define como: S=T2/T1. Implementar un predicado speedup(S, G1, G2) que calcule el speedup S de un programa Prolog P1, en el que se lanza el objetivo G1, con respecto al programa P2, en el que se lanza el objetivo G2. (Ayuda: utilizar el operador cputime, que se evalúa el número de segundos utilizados por Prolog hasta el momento de su ejecución. Así pues, un objetivo como el siguiente:

?- T1 is cputime, Goal, T2 is cputime, T is T2 – T1.

evalúa el tiempo T consumido durante la ejecución del objetivo Goal.)

Ejercicio 18.
La relación "invertir una lista" se puede definir de forma directa en términos del predicado append:

```
% invertir(L,I), I es la lista que resulta de invertir L
invertir1([],[]).
invertir1([H|T],L) :- invertir1(T,Z), append(Z,[H],L).
```

El problema con esta versión es que es muy ineficiente debido a que el número de llamadas al operador "[.|..]" es cuadrático con respecto al número de elementos de la lista que se está invirtiendo, si también contamos las llamadas al operador "[.|..]" producidas por una llamada al predicado append. Se puede lograr que el número de llamadas al operador "[.|..]" sea lineal con respecto al número de elementos de la lista utilizando un parámetro de acumulación. Un parámetro de acumulación es un argumento de un predicado que, como indica su nombre, se utiliza para almacenar resultados intermedios. En el caso que nos ocupa, se almacena la lista que acabará por ser la lista invertida, en sus diferentes fases de construcción.

```
% invertir2(L,I), I es la lista que resulta de invertir L
% Usando un parametro de acumulacion.
invertir2(L,I) :- inv(L, [], I).
% inv(Lista, Acumulador, Invertida)
inv([], I, I).
inv([X|R], A, I) :- inv(R, [X|A], I).
```

Medir el speedup del programa "invertir2" con respecto al programa "invertir1". Compare también

la eficiencia de estos programas con respecto al predicado predefinido “reverse”. (**Ayuda**: definir un predicado capaz de generar una lista de miles de elementos.)

(**Observación**: En el libro **PROGRAMACIÓN LÓGICA, TEORÍA Y PRÁCTICA**, de María Alpuente y Pascual Julián, página 193, puede encontrar más información sobre este ejercicio y el uso de parámetros de acumulación.)

Ejercicio 19.
La siguiente definición de “longitud1” es una versión ingenua que permite calcular la longitud de una lista:

```
% longitud1(L,N), N es la longitud de la lista L
longitud1([],0).
longitud1([H|T],N) :- longitud1(T,Z), N is Z +1.
```

Utilizando parámetros de acumulación, definir una versión “longitud2(L,N)” más eficiente.

Medir el speedup del programa “longitud2” con respecto al programa “longitud1”. Compare también la eficiencia de estos programas con respecto al predicado “length” predefinido en Prolog. (**Ayuda**: definir un predicado capaz de generar una lista de miles de elementos.)

Ejercicio 20.
La siguiente definición de “suma1” es una versión ingenua que permite calcular la suma de los elementos de una lista de enteros:

```
% suma1(L,N), N es  la suma de los elementos de una lista de enteros.
suma1([],0).
suma1([H|T],N) :- suma1(T,Z), N is Z +H
```

Utilizando parámetros de acumulación, definir una versión “suma2(L,N)” más eficiente.

Medir el speedup del programa “suma2” con respecto al programa “suma1”.

Ejercicio 21.
La sucesión de Fibonacci está compuesta por los números: a1=1, a2=1, a3=2, a4=3, a5=5, a6=8, ...
Esta sucesión puede definirse por intensión en los siguientes términos

a1=1;
a2=1;
an = an-1 + an-2 si n>2.

La anterior definición puede trasladarse a sintaxis Prolog de forma inmediata

```
fib(1,1).
fib(2,1).
fib(N,F) :- N>2, H1 is N-1, H2 is N-2,
            fib(H1,F1),fib(H2,F2),
            F is F1+F2.
```

Sin embargo, este programa es muy ineficiente, pues tiende a rehacer muchos cálculos previamente efectuados. Podemos confirmar lo anterior sin más que inspeccionar la traza del objetivo "?- fib(6, F).'', por ejemplo, puede verse que el número fib(3, _) se calcula en tres ocasiones. Utilizando una función auxiliar con parámetros de acumulación, definir un predicado fib2(N, F) que compute eficientemente el N-esimo número de Fibonacci evitando cómputos redundantes. Medir el speedup del programa "fib2" con respecto al programa "fib".

LABORATORIO 6 (Estructuras de control y Entrada/Salida)

Para medir lo aprendido durante la Práctica 6, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab6.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab6.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab6.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 22.

Estudiar el comportamiento del programa

```
p(X) :- q,r(X).          t.
p(X) :- s(X),t.          a.
q :- a,!,b.              b :- fail.
q :- c,d.                c.
r(uno).                  d.
s(dos).
```

a) Realizar una traza y generar el árbol correspondiente al objetivo "?- p(X).". b) Eliminar el corte del programa y repetir la experiencia. ¿Qué ramas fueron podadas?. c) ¿ El corte introducido en el programa es rojo o verde ?.

Observaciones.

a) Se dice que un corte introducido en un programa es **verde**, cuando no altera su significado declarativo. Esto es, el programa produce las mismas respuestas con y sin el corte. Por el contrario, si la introducción del corte altera el significado declarativo de un programa se dice que el corte es **rojo**.
b) Dado que la introducción del corte puede alterar el significado declarativo de un programa debe usarse con cuidado. La regla es, evitar el corte siempre que sea posible.

Ejercicio 23.

Definir un predicado **add(X, L, L1)** que añada un elemento X, a la lista L para dar L1, sin incurrir en repeticiones de elementos. Esto puede hacerse mediante la siguiente regla

> Si X es miembro de la lista L entonces L1 = L,
> si no L1 es igual a la lista [X | L];

para cuya implementación es necesario el empleo del corte. Comprobar que omitiendo el corte o alguna construcción derivada del corte (e. g. el predicado **not**) sería posible la adición de elementos repetidos. Por ejemplo, llame a la nueva versión sin corte "add2" y compruebe que

```
?- add2(a, [a, b, c], L).
L = [a, b, c];
L = [a,a, b, c]
```

Así pues, en la solución de este ejercicio es vital el empleo del corte para obtener el significado esperado para nuestro predicado y no como un mero instrumento para aumentar la eficiencia.

Ejercicio 24.
Queremos definir un predicado **diferente(X, Y)** que sea verdadero cuando X e Y no unifican. Esto puede decirse en PROLOG teniendo en cuenta que

> Si X e Y unifican entonces **diferente(X, Y)** falla,
> si no **diferente(X, Y)** tiene éxito.

Definir este predicado empleando a) el corte, **fail** y **true** (llame a esta versión "diferente_a"); b) empleando únicamente **not** (llame a esta versión "diferente_b").

Ejercicio 25.
El predicado predefinido **repeat** no es imprescindible en la programación con el lenguaje PROLOG. a) Para confirmar el anterior aserto, escribe un pequeño programa llamado cubo que solicite un número por teclado y calcule su cubo. El programa debe operar como a continuación se indica

```
?- cubo.
Siguiente número 5.
El cubo de 5 es 125
Siguiente número 3.
El cubo de 3 es 27
Siguiente número stop.
yes
```

b) Dar una versión empleando **repeat** (llame a esta versión "cubo_2").

Ejercicio 26.
Definir un predicado “cont(Fichero, Carac, Palab, Lineas)” que tome como argumento el nombre de un fichero de texto y cuente los caracteres, las palabras y las lineas contenidas en ese fichero. Como efecto lateral, el predicado cont/4 deberá imprimir el número de caracteres, palabras y líneas, procurando dar un formato agradable a la salida por pantalla.

LABORATORIO 7 (Manipulación de términos y orden superior)

Para medir lo aprendido durante la Práctica 7, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab7.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab7.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab7.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 27.

Definir un predicado, ficha(H), que actuando sobre un hecho almacenado en una base de datos lo visualice en forma de ficha. Por ejemplo, si se lanzase el objetivo

```
?- ficha(libro(tol85,
            autor('Tolkien', 'J.R.R.' ),
            titulo('El Señor de los Anillos'),
            editorial(minotauro),
            prestado(jul101, fecha(28, noviembre, 2004)))).
```

se mostraría:

```
libro:
     tol85,
     * autor:
          Tolkien,
          J.R.R.,
     * titulo:
          El Señor de los Anillos,
     * editorial:
          minotauro,
     * prestado:
          jul101,
          ** fecha:
               28,
               noviembre,
               2004
```

[Observación: Escriba 8 caracteres para el sangrado de cada elemento.]

Ejercicio 28.

a) Defina un predicado **unific_a(X,Y)** que tenga éxito cuando las expresiones X e Y puedan ser unificadas. Esto es, queremos que se comporte como el predicado predefinido en Prolog para la unificación de expresiones. b) Amplíe el programa anterior para que el predicado **unific_a/2** compruebe la ocurrencia de variables. Denomine al nuevo predicado **unific_b(X,Y)**.

Ejercicio 29.

a) Defina un predicado **reduce(List,Func,Base,Result)** que toma como argumentos una lista, **List**,

el nombre de una función de aridad 2, **Func**, (representada por un predicado de aridad 3) y un valor base, **Base**, para dar un resultado **Result**, que reduce la lista a un valor. Supongamos que el predicado **reduce** toma como argumentos de entrada, por ejemplo, una lista [e1, e2, e3], una función f (de aridad 2) y un valor inicial (base) b, el objeto es producir como resultado el valor f(e1, f(e2, f(e3, b))).
(**AYUDA**: utilice el predicado predefinido **apply/2** que aplica "funciones" sobre sus argumentos de forma eficiente. Por ejemplo, si hemos definido la función **sum(X,Y,S)**, el objetivo **apply(sum, [2,3,S])** construye el átomo **sum(2,3,S)** y lo lanza evaluándolo, con lo que se obtiene **S=5**.)

b) Haciendo uso del predicado **reduce/4** y de la "función" **sum(X,Y,S)**, que toma los números **X** e **Y**, dando su suma **S=X+Y** como resultado, definir el predicado **sumList(List, Suma)** que suma los elementos de una lista de números.

c) Haciendo uso del predicado **reduce/4** y de la "función" **mult(X,Y,P)**, que multiplica los números **X** e **Y** dando su producto **P**, definir el predicado **multList(List, Prod)** que multiplica los elementos de una lista de números.

LABORATORIO 8 (Definición de funciones en el lenguaje Haskell y funciones de orden superior)

Para medir lo aprendido durante la Práctica 7, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab8.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab8.hs con el código Haskell de todos los programas solicitados.
Apellidos_Lab8.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 30.

Definir las siguientes funciones sin usar las funciones predefinidas **ord** y **char**:
a) El predicado **digit** que aplicado a un carácter nos dice si este es un dígito.
b) Los predicados **uppercase** y **lowercase** que aplicados a un carácter nos dice si es una letra mayúscula o minúscula. (nota: entienda por letra los caracteres [a .. z] y [A .. Z]).
c) El predicado **letter** que aplicado a un carácter nos dice si es una letra.
d) La función **charofdigit** que aplicada a un entero entre 0 .. 9 lo convierte en carácter.

Ejercicio 31.

Como ejemplo de diseño de funciones recursivas, definir las siguientes funciones:

a) (**among x y**); La función among es un predicado. x es un objeto de tipo T, y es una lista formada por objetos de tipo T. among devuelve true
si x se encuentra como elemento de y, false en cualquier otro caso.
ej.: (among a [a,b,c]) == (among a [c,d,e,a]) = true; (among a [b,c,d]) == false.

b) **stringcopy : Int -> String -> String**, que concatene un string un número entero de veces. Emplee parámetros formales para su definición.

c) En una segunda versión de la última función, denominada **strcop** utilizar ajuste de patrones para su definición.

Estas funciones están definidas para números positivos, por lo que es parcial sobre el tipo de los enteros: Tratar los casos de indefinición utilizando la expresión **if** y la función **error**.

Ejercicio 32.

Una carta puede caracterizarse por su valor y su palo. Utilizando la declaración data

definir los tipos de datos: **Valor**, **Palo** y **Carta**. Definir las funciones: **palode**: Carta -> Palo; y **valorde**: Carta -> Valor; . Y los predicados: **espalo**: Palo -> Carta -> Bool; y **esvalor**: Valor -> Carta -> Bool; .

Ejercicio 33.

Un saco es como un conjunto, excepto que un elemento puede ocurrir mas de una vez. Un saco puede representarse mediante una lista de pares que consisten en un elemento y el número de veces que aparece el elemento en el saco. Escribe una definición polimórfica del tipo saco, junto con las funciones: **add**, que añade un elemento a un saco; **remove**, que se le proporciona un elemento y devuelve el elemento del saco eliminado junto con el saco al que se le ha quitado ese elemento; **isempty**, que devuelve true si un saco esta vacío.

Ejercicio 34.

a) Definir las funciones **map** y **++** (concatenar dos listas) en términos de la función reduce. Llamad a las nuevas funciones **mapR** y **appR** respectivamente.

b) Comprobar que la función among puede definirse en términos de las funciones lambda y reduce de la siguiente forma:

```
reduce (\next -> (\isthea -> (if next == x then True else isthea)))  False (y:ys)
```

La lambda expresión que se pasa como argumento funcional no deja rastro en el interprete. ¿Que tendríamos que hacer si quisiéramos evaluarla?. ¿Explicar como actúa esta función?.

Ejercicio 35.

Definir una función **uncurry** que tome una función de curry de tipo **alpha -> beta -> gamma** y la convierta en una función de tipo **(alpha, beta) -> gamma**, sin currificar.

LABORATORIO 9 (Representación del conocimiento y resolución de problemas mediante búsqueda)

(SE IMPARTIRÁ EN HORAS DE TEORÍA: LOS EJERCICIOS SERVIRAN DE EJEMPLO PARA ILUSTRAR LAS TÉCNICAS DE REPRESENTACIÓN DEL CONOCIMIENTO Y RESOLUCIÓN DE PROBLEMAS. EL PROFESOR DARÁ LA SOLUCIÓN DETALLADA DE ESOS EJERCICIOS)

Para medir lo aprendido durante la Práctica 8, los alumnos deberán enviar los programas y soluciones a las preguntas que a continuación se indican. El envío se realizará en un archivo ZIP de nombre:

gN_Apellidos_Lab8.ZIP

donde N es el número del equipo de trabajo. El archivo ZIP contendrá los ficheros:
Apellidos_Lab8.pl con el código Prolog de todos los programas solicitados.
Apellidos_Lab8.txt con los datos personales de la persona que hace el envío y el enunciado y solución de aquéllas preguntas que (no pudiéndose responder vía programa) se formulen en cada uno de los ejercicios del laboratorio.

En todos los casos, "**Apellidos**" son los apellidos del alumno que hace el envío.

Ejercicio 30.

Consideremos la base de datos del Ejercicio 7, que estructuraba como un hecho la información sobre una familia y represente el conocimiento almacenado, utilizando ahora relaciones binarias. (Ayuda: Emplear el concepto de red semántica puede facilitar esta tarea.)

Tomando como base la nueva representación, responder a las siguientes preguntas planteando objetivos:
a) Encontrar los nombres y apellidos de las mujeres casadas que tienen tres o más hijos;
b) Encontrar los nombres de las familias que no tienen hijos;
c) Encontrar los nombres de las familias en las que la mujer trabaja pero el marido no.
d) Encontrar los nombres de todas las personas en la base de datos;
e) Encontrar los hijos nacidos después de 1980;
f) Encontrar todas la mujeres empleadas;
g) Encontrar los nombres de las personas desempleadas nacidas antes de 1960;
h) Encontrar las personas nacidas después de 1950 cuyo salario esté comprendido entre las 800 y 1300 euros.

[**Observación.**
Convertir los objetivos planteados en los anteriores subapartados en los siguientes predicados: ej30a(Nombre, Apellidos); ej30b(Apellidos_Padre, Apellidos_Madre); ej30c(Apellidos_Padre, Apellidos_Madre); ej30d(Nombre, Apellidos); ej30e(Nombre_Hijo, Apellidos_Hijo), ej30f(Nombre_Mujer, Apellidos_Mujer), ej30g(Nombre, Apellidos), y ej30h(Nombre, Apellidos). Introducir estos predicados en el fichero **g_N_Lab9.pl** .]

Ejercicio 31. (Juego del 24)

Definir un predicado, **game24**(Lista4Digitos, Expresion), que resuelva el "Juego del 24". El Juego del 24 es un entretenimiento matemático del que tuve noticia en una reunión del Portland Extreme Programming User Group (http://xpdx.org), celebrada el 5 de junio de

2001: Está inspirado en un juego comercializado para desarrollar habilidades matemáticas entre los estudiantes de educación básica. El juego procede de la siguiente manera: dados cuatro dígitos entre el 1 y el 9, encontrar una expresión aritmética que se evalúe a 24 y en la que cada dígito ocurra exactamente una vez. Por ejemplo, una solución para la lista de dígitos [2,3,6,8] es: (2+8)*3-6. Existen otras 25 soluciones más para este caso (incluyendo las soluciones conmutativa y asociativamente equivalentes.

Ejercicio 32. (Misioneros y caníbales)

Tres misioneros y tres caníbales se encuentran en una orilla de un río. A todos ellos les gustaría pasar a la otra orilla. Los misioneros no se fían de los caníbales. Por ello, los misioneros han planificado el viaje de forma que el número de misioneros en cada orilla del río nunca sea menor que el número de caníbales en esa misma orilla. Sólo disponen de una lancha de dos plazas. ¿Cómo podrían atravesar el río sin que los misioneros corran peligro de ser devorados por los caníbales? Resuelva el problema usando una estrategia de búsqueda. Para ello defina un predicado **getPlan**(Plan), que muestre una lista con las acciones o movimientos necesarios para llevar a la práctica el plan.

[**Observación**.
Para representar las acciones o movimientos, emplee términos como “trans(Orilla, Misioneros, Canibales)”. Más concretamente, el término “trans(der, M, C)” se interpretará como:

> trasladar a la orilla derecha (desde la orilla izquierda) M misioneros y C caníbales.

El término “trans(izq, M, C)” se interpretará como:

> trasladar a la orilla izquierda (desde la orilla derecha) M misioneros y C caníbales.

La solución al problema (es decir, el plan) se compondrá de una lista de estos términos. El plan se podrá obtener en orden inverso, esto es, la primera de las acciones a realizar será el último elemento de la lista.]