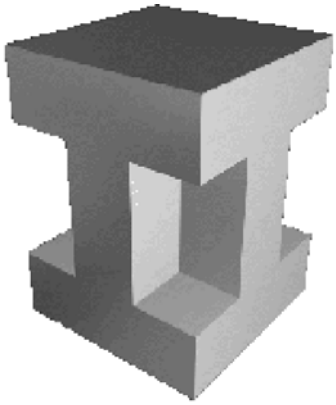
INTRODUCCIÓN BÁSICA AL PROLOG



Ingeniería Informática

VNIVERSITAT DE VALÈNCIA

Departamento de Informática

PRÓLOGO

Esta introducción al Prolog está pensada para las prácticas de la asignatura de Matemática Discreta de la Universitat de València, no es por tanto autocontenida ni completa. Para comprender correctamente lo aquí explicado es conveniente leer los apuntes de teoría de la asignatura.

Para cualquier comentario o error en el texto ponerse en contacto con el autor: Fernando Barber Miralles (Fernando.Barber@uv.es).

1. INTRODUCCIÓN

El Prolog es un lenguaje de programación lógica (de ahí su nombre). Esto significa que está basado en la lógica de predicados, en concreto en un subconjunto de esta lógica denominado cláusulas de Horn.

En Prolog, un programa es un conjunto de hechos y reglas que representan el problema que se pretende resolver. Ante una determinada pregunta sobre el problema, el Prolog utilizará estos hechos y reglas para intentar demostrar la veracidad o falsedad de la pregunta que se le ha planteado.

La demostración de la pregunta se basa en dos principios:

- **La unificación**: Es el algoritmo que se encarga de resolver las igualdades lógicas.
- **El principio de resolución**: Es el algoritmo que, a partir de la negación de la pregunta y los hechos y reglas del programa, intenta llegar a un absurdo para demostrar que la pregunta es cierta.

2. PROGRAMAS, PREDICADOS

Un *programa* Prolog está formado por *predicados*. Cada predicado está definido unívocamente por su *nombre* y su *aridad*. La aridad es el número de argumentos (o parámetros) de un predicado.

Ej.

```
humano(pepe).
humano(juan).
```

Para referenciar este predicado se utiliza únicamente su nombre y aridad: humano/1.

Cada predicado en Prolog puede estar definido por una o más *cláusulas*. En nuestro ejemplo, humano/1 está definido por dos cláusulas.

Las cláusulas a su vez pueden ser de dos tipos: *hechos* y *reglas*.

Los hechos son afirmaciones que consideramos ciertas en nuestro programa.

Las reglas son implicaciones lógicas, que pueden tener varios antecedentes pero un único consecuente. Este tipo de reglas se denominan cláusulas de Horn.

Ej.

humano(X) → mortal(X).

que utilizando la sintaxis de Prolog se escribe:

```
mortal(X):-
        humano(X).
```

Los hechos se pueden considerar casos particulares de reglas en donde no hay ningún antecedente.

3. TÉRMINOS DE PROLOG

Los argumentos en una cláusula pueden ser cualquier *término* de Prolog. Los términos de Prolog son los siguientes:

Átomos: Los átomos hacen las funciones de identificadores en Prolog. Un átomo puede ser cualquier combinación de caracteres alfanuméricos, pero si no empieza por una letra minúscula o si contiene caracteres que no sean letras, números o el carácter subrayado '_', debe encerrarse entre comillas simples.

Ej.

Válidos	No Válidos
unatomo	un atomo
otro_atomo	_otro_atomo
'Atomo'	Atomo
'2+3'	2+3
'un atomo'	

Enteros: Números enteros. El rango es dependiente de la implementación.

Reales: Números reales. El rango también es dependiente de la implementación.

Ej:

12.3 2.5e-4

Estructuras (funciones): Se utilizan para agrupar un número constante de términos. Son las funciones de lógica de predicados.

Listas: Se utilizan para agrupar un número variable de términos.

Variables: Una variable puede ser instanciada a cualquier otro término de Prolog, incluida otra variable. No son equivalentes a las variables en los lenguajes procedurales, sino a las variables lógicas o matemáticas. El nombre de una variable ha de comenzar por una mayúscula o subrayado y puede contener cualquier combinación de letras, números y el carácter subrayado.

Ej:

Suma X2 _32 _

El carácter subrayado es un tipo de variable especial que se denomina *variable anónima*. Ésta se utiliza para escribir variables sin necesidad de darles un nombre. Cada aparición del carácter subrayado representa una variable distinta.

4. UNIFICACIÓN

La unificación, como ya se ha comentado, es el algoritmo que se encarga de resolver las igualdades lógicas.

Aunque la unificación en Prolog sustituye a la asignación de los lenguajes procedurales, no hay que confundirla con esta, son operaciones totalmente distintas. La operación de unificación se representa mediante el operador =.

Ejercicios:

```
X = pepe.
3 = X.
X = Y.
X = 3, X = 5.
f(3, 2) = f(X, 2).
f(X, p(a) ) = f(p(Y), Y).
X = 3 + 2.
5 = 3 + 2.
X + 3 = 2 + Y.
f(X, f(X) ) = f(Y, Y).
```

5. COMPARACIÓN DE TÉRMINOS

Para comparar términos existen varios operadores. De estos operadores, los más importantes son el igual (==) y el distinto (\==). Para éstos hay que tener en cuenta que dos términos son iguales solamente si son exactamente el mismo término (mismo valor y mismo tipo).

Ej.

5 == 5	Yes	
5 == '5'	No	Estamos comparando un entero con un átomo.
5 == 2 + 3	No	Estamos comparando un entero con una estructura.
5 == X	No	Estamos comparando un entero con una variable.

6. RESOLUCIÓN DE PROGRAMAS

En Prolog, todo lo que se introduce en el "prompt" del sistema Prolog se consideran preguntas. Para escribir un programa es necesario escribirlo en un fichero de texto y posteriormente cargarlo o compilarlo en Prolog. Para compilar un programa hay que escribir el nombre del fichero encerrado entre corchetes. Si el nombre tiene la extensión por defecto para el Prolog (en SWI es .pl) no hace falta incluirla. El nombre ha de ser un átomo, por tanto si contiene algún carácter especial deberá ir encerrado entre comillas simples.

Ej.

Para cargar el fichero prueba.pl que está en el directorio /prolog

[prueba]	ó	['prueba.pl']	Si estamos en el mismo directorio.
['/prolog/prueba.pl']			Si estamos en otro directorio.

Cuando se compila un fichero, todos los predicados que tiene (tanto hechos como reglas) pasan a ser axiomas del sistema, es decir, se consideran ciertos. A partir de ese momento se podrán hacer preguntas sobre esos predicados.

Ej.

```
humano(pepe).
humano(juan).

mortal(X):-
      humano(X).
```

Después de cargar este fichero podremos hacer preguntas como las siguientes:

1 ?- humano(pepe). Yes
2 ?- mortal(pepe). Yes
3 ?- mortal(javi). No
4 ?- mortal(X).
X = pepe more? (;)
X = juan

(NOTA: '1 ?-' es el prompt del SWI-Prolog)
Si encuentra la pregunta como un hecho, la respuesta es directamente que **sí** (pregunta 1 del ejemplo anterior). Si encuentra la pregunta como la cabeza (o consecuencia) de una regla, toma cada uno de los predicados del cuerpo de la regla como nuevas preguntas y sólo si todas son ciertas, la respuesta será **sí** (pregunta 2). Si no encuentra ningún hecho o regla que haga cierta la pregunta, la respuesta será **no** (pregunta 3).

Es importante tener en cuenta que para comprobar si la pregunta coincide con un hecho o con la cabeza de una regla se aplica unificación. Esto significa que si en la pregunta hay alguna variable, ésta unificará con los términos del hecho o regla, es decir, será igual a los valores que hacen cierta la pregunta. Esto se utiliza para extraer resultados (pregunta 4). Si existe más de un resultado, habrá que pulsar ";" para verlos.

Ejercicio:
Sea el siguiente programa Prolog:

```
padre(juan, pepe).
padre(pepe, javi).
padre(pepe, jose).
padre(antonio, david).
```

Completarlo con varias reglas que definan las relaciones hijo/2, abuelo/2, hermano/2, primo/2, etc. Por ejemplo, para hijo/2 será:

```
hijo(X, Y):-
      padre(Y, X).
```

7. ESTRUCTURAS DE CONTROL

En Prolog, a diferencia de lenguajes procedurales como Pascal o C, no existen estructuras de control para bucles. Éstos se implementan mediante predicados recursivos. En cambio existen estructuras de control nuevas que no existen en otros lenguajes y que describiremos a continuación.

8.1. AND

El "AND" o Y lógico se representa en Prolog mediante la coma ','. Este operador ya se ha visto en detalle en todos los ejemplos utilizados hasta ahora.

8.2. OR

El "OR" o O lógico se puede realizar en Prolog de dos formas distintas: mediante varias cláusulas para un mismo predicado o mediante el operador ';'.

Mediante varias cláusulas se consigue poniendo cada una de las opciones en una cláusula distinta del predicado. Entre las distintas cláusulas de un mismo predicado se puede considerar que existe un "OR". Esto de hecho ya se ha utilizado por ejemplo en el factorial (se define mediante dos cláusulas) o en el predicado padre/2.

El operador ';' se utiliza igual que el operador ','. Únicamente hay que prestar especial atención a la prioridad de los operadores, ya que ';' tiene menor prioridad que ',', lo que significa que normalmente habrá que encerrar entre paréntesis la operación. A este respecto hay que recordar que el "OR" se considera una suma lógica y el "AND" una multiplicación lógica.
Ej.
Escribir un predicado numero/1 que sea cierto para los números enteros o reales y falso en caso contrario.

Varias cláusulas

```
numero(X):-
      integer(X).
numero(X):-
      real(X).
```

Operador ';'

```
numero(X):-
      integer(X);
      real(X).
```

Normalmente se utiliza la disyunción mediante cláusulas puesto que resulta más claro.

8.3. NOT

La negación se realiza mediante el predicado not. El predicado not/1 antes de la llamada a un predicado *P* cambia su valor de verdad, es decir, si el predicado *P* tiene éxito, not(*P*) fallará y si el predicado *P* falla, not(*P*) tendrá éxito.
Ej.

```
      no_entero(X):-
            not(integer(X)).
```

Sin embargo hay que tener la precaución de aplicar la negación únicamente en llamadas a predicados donde todas las variables existentes estén ya instanciadas, ya que si no el comportamiento no es el esperado.
Ej.

```
      padre(pepe, juan).

      huerfano(X):-
            not(padre(Z, X)).
```

Para la pregunta huerfano(pepe) funcionará bien y dirá que sí, pero si preguntamos quién es huérfano con huerfano(X) dirá que no, es decir, que no hay ningún huérfano, lo cual no es correcto.

8.4. Fallo, Cierto

El predicado de fallo, *fail*, se utiliza para obligar al Prolog a dar un fallo.

El predicado cierto, *true*, se utiliza como instrucción nula, es decir, cuando se tiene que escribir una instrucción pero no se quiere que haga nada.

Un ejemplo típico es para imitar las instrucciones *if-then*:

```
cero(X):-
      (X == 0, write('Cero');
      true

      ).
```

9. ARITMÉTICA

Como hemos visto, en la unificación no se evalúan expresiones. Para ello existe un operador especial 'is' que antes de realizar la unificación evalúa la parte derecha como si se tratase de una expresión aritmética.

X is 3 + 2 es equivalente a X = evaluar(3 + 2)

Para que esta evaluación se pueda llevar a cabo es necesario que la parte derecha no contenga ninguna variable sin instanciar, en caso contrario el sistema dará error. En el apéndice 1 hay una tabla con las operaciones y funciones más comunes en Prolog.

Ej.:	Resultado
X is 3 + 2.	X = 5 Yes
5 is 3 + 2	Yes
X is 3 + Y	Error. No se puede evaluar.
X is ln(exp(2))	X = 2.0 Yes

9.1 Comparación aritmética

Existen varios operadores para comparar expresiones aritméticas. Estos operadores evalúan sus dos operandos antes de realizar la comparación. En el apéndice 1 se pueden ver los distintos operadores existentes.

5 =:= 2 + 3. es equivalente a evaluar(5) =:= evaluar(2 + 3)

10. LISTAS

La única estructura de datos predefinida de Prolog son las listas. Sirven para agrupar un número indeterminado de términos. En Prolog se escriben de la siguiente forma:

[1, 2, 3]

Los términos son de cualquier tipo, y se pueden mezclar dentro de una misma lista:

[1, pepe, 6.3, [10, 0], A]

El átomo '[]' se utiliza para indicar la lista vacía.

La única operación permitida sobre listas es separar la lista en su primer elemento y el resto de la lista. Al primer elemento de la lista se le denomina *cabeza* y al resto de la lista *cola*. El operador que se utiliza para separar la lista es la barra vertical: '|'.

[1| [2]] representa [1, 2]

Para separar una lista determinada en su cabeza y su cola, se utiliza unificación.

Ej. Si la lista L es [1, 2, 3]
L = [Cab| Cola]. Da como resultado:
Cab = 1
Cola = [2, 3]

Ej. Predicado para mostrar todos los elementos de una lista:

```
mostrar([]).
mostrar([Cab| Cola]):-
      writeln(Cab),
      mostrar(Cola).
```

APÉNDICE 1: PREDICADOS MÁS COMUNES DE PROLOG

Estructuras de control

,	Es un "and" entre objetivos. Si una cláusula tiene una secuencia de objetivos separados por "," todos los objetivos deben cumplirse para que la cláusula sea verdad.
;	Es un "or" entre objetivos. Todos los objetivos de un sólo lado de la disyunción deben ser ciertos para que la disyunción sea cierta.
!	"Cut" o "corte" es un objetivo que siempre se cumple. El corte limita la vuelta atrás ("backtracking"), de forma que si en la vuelta atrás se encuentra un corte, entonces el predicado entero que contiene el corte falla, no sólo la cláusula.
not(P)	El término P es interpretado como un objetivo. Si P tiene éxito, entonces not P falla y si P falla, not P tiene éxito.
true	Es un objetivo que siempre tiene éxito.
fail	Es un objetivo que siempre falla. Útil para provocar backtracking.
repeat	Este objetivo siempre es cierto y siempre deja un punto de backtracking.

Operadores y predicados aritméticos

Operador	Explicación	Operador	Explicación
+	Suma	-	Resta
*	Producto	/	División
//	División entera	^	Exponenciación
mod	Resto de la división	abs()	Valor absoluto
sin()	Seno	cos()	Coseno
tan()	Tangente	asin()	Arco seno
acos()	Arco coseno	atan()	Arco tangente
ln()	Logaritmo neperiano	exp()	Potencia neperiana
sqrt()	Raíz cuadrada	fix()	Trunca a entero

Operadores aritméticos de evaluación

Operador	Explicación	Operador	Explicación
>	Mayor que	<	Menor que
>=	Mayor o igual que	=<	Menor igual que
=:=	Igual que	=\=	Distinto de

Clasificación de términos

Predicado	Explicación	Predicado	Explicación
atom()	Es un átomo	integer()	Es un entero
real()	Es un número en coma flotante	number()	Es un entero o un número en punto flotante
string()	Es una cadena	atomic()	Es un dato de tipo atómico. Es decir, átomo o número.
var()	Es una variable libre	nonvar()	No es una variable libre

Unificación de términos

Operador	Explicación	Operador	Explicación
=	Unifica ambos términos	\=	Cierto si ambos términos no pueden unificar

Comparación de términos

Operador	Explicación	Operador	Explicación
==	Términos equivalentes	\==	Términos no equivalentes
@<	Menor que	@>	Mayor que
@=<	Menor o igual que	@>=	Mayor o igual que

Entrada/Salida estándar

Predicado	Explicación
read()	Lee un término de la entrada estándar. El término debe acabar en un punto.
write()	Escribe un término en la salida estándar.
writeq()	Escribe un término en la salida estándar de forma que pueda ser leído por read().
get()	Lee un carácter (código ASCII) de la entrada estándar.
put()	Escribe un carácter (código ASCII) en la salida estándar.
nl	Escribe un salto de línea en la salida estándar.