Lenguajes Lógicos

Moisés García Villanueva

Diciembre de 2012

1 Lógica de Predicados (repaso breve)

1.1 Definiciones

En el cálculo proposicional se requiere tener enunciados que sean verdaderos o falsos. Una afirmación de la forma $x < 10$ o $z >= 10$, se dice que es una proposición en el momento que se le asigna un valor a las variables x y z, en ese momento es verdadera o falsa la afirmación..

Lo mismo sucede en nuestro lengua materna, al tener afirmaciones similares como las siguientes:

Ella es parte de la Facultad de Ingeniería y del equipo de fútbol.

El toma la combi para ir al trabajo y además va a estudiar.

Podemos observar que Ella, él, fútbol, trabajo, estudiar se pueden utilizar como variables y las afirmaciones anteriores pueden reescribirse como:

x es parte de la Facultad de Ingeniería y del equipo w

x toma la combi para ir al w y además va a z

1.1.1 Predicado

Es una afirmación que expresa una propiedad de un objeto o una relación entre objetos. Estas afirmaciones se hacen verdaderas o falsas cuando se reemplazan las variables (objetos) por valores específicos.

Ejemplos de predicados puede ser: $p(x, y) : x + y > 5$, en el cual se tienen dos variables.

1.2 Universo del discurso

Llamaremos de esta forma al conjunto al cual pertenecen los valores que puedan tomar las variables. Lo notaremos por U y lo nombraremos por conjunto universal o, simplemente, universo. Debe contener, al menos, un elemento. El conjunto de números enteros para el predicado $p(x, y) : x + y > 5$.

1.3 Predicados y Proposiciones

Si $p(x1, x2, ..., xn)$ es un predicado constante con n variables y asignamos los valores $c1, c2, ..., cn$ a cada una de ellas, el resultado es la proposición $p(c1, c2, ..., cn)$.

Para transformar un predicado en proposición, cada variable del predicado debe estar "ligada".

El predicado $p(x, y) : x + y > 5$ tiene las variables x e y libres, por que pueden tomar cualquier valor del Universo del discurso que se defina. En el momento que se dice $x = 2$ e $y = 4$, las variables son ligadas y el predicado se convierte en proposición, porque toma el valor de **verdadero**.

Ejemplo:

1. ¿Cuántas veces se imprime el valor de x en el siguiente programa?

```
x := 10
y := 1
Hacer mientras y <= 7
Comienzo
      z := 1
      Hacer mientras z y + 3
      Comienzo
            Si [(x > 8) o ((y > 5) y (z < 10))] entonces imprimir x
            z := z + 1
      Fin
      x:=x - 1
      y:=y + 1
Fin
```

2 Relaciones

Prolog es un lenguaje de programación lógica cuya primera versión fue desarrollada a principios de la década de 1970 por Colmerauer en la universidad de Marsella. Contrariamente a otros lenguajes de programación basados es estructuras de control y definición de funciones para calcular resultados, Prolog está orientado a la especificación de **relaciones** para responder consultas. En ese sentido Prolog es similar a un sistema de base de datos, aunque en el contexto de la inteligencia artificial se prefiere hablar de bases de conocimiento, enfatizando la complejidad estructural de los datos y de las deducciones que se pueden obtener de ellos.

Por ejemplo, para especificar la relación el `padre` de `X` es `Y`, se crea una base de conocimiento con hechos expresados mediante un predicado `padre(X,Y)` de la siguiente manera:

```
padre(juan,pedro).
padre(josé,pedro).
padre(maría,pedro).
padre(pedro,pablo).
padre(ana,alberto).
```

Podemos crear además la relación `madre` de la siguiente forma.

```
 madre(juan,ana).
madre(josé,ana).
madre(maría,ana).
madre(pedro,juanita).
madre(ana,julia).
```

Esto corresponde a definiciones por extensión (caso a caso) de la relación padre y madre.

3 Queries

Una vez creada la base de conocimiento (base de datos), procedemos ahora a realizar consultas de dicha información. Es el mecanismo para extraer conocimiento del programa. En Prolog todas las cláusulas terminan con el delimitador '.' . Las cláusulas de consulta se interpretan como ecuaciones lógicas y pueden incluir variables. Por ejemplo:

```
?- padre(maría,pablo).
No
?- padre(ana,alberto).
```

```
Yes
?- padre(maría,X).
X = pedro ;
No
```

Prolog genera soluciones a estas consultas o ecuaciones lógicas, dándoles valores a las variables (creando propocisiones).

Cuando la consulta no tiene variables, como en los dos primeros casos del ejemplo anterior, la respuesta es polar (Yes o No). Cuando hay variables, Prolog entrega la secuencia de soluciones. Para ver la lista de soluciones se debe presionar ';' (el último No indica que no hay más soluciones). Las variables son identificadores que empiezan siempre con una letra mayúscula. Los identificadores con letra inicial minúscula corresponden a nombres de predicados o átomos. La forma general de una consulta consiste en una secuencia de predicados que deben ser satisfechos conjuntamente en el orden especificado. Esto permite consultas complejas similares al join en bases de datos, por ejemplo para determinar el abuelo paterno de un individuo, aquí maría, se puede plantear la consulta:

```
?- padre(maría,X),padre(X,Y).
X = pedro
Y = pablo ;
No
```

Primero se obtiene el padre de maría en X y luego con ese valor el padre de X en Y, que corresponde al dato buscado. El "join" se logra compartiendo la variable X. Prolog resuelve consultas complejas encontrando una solución para el primer predicado y luego, con el valor obtenido para las variables, procede con el resto de la consulta. En el ejemplo anterior, el primer predicado `padre(maría,X)` produce como solución `X = pedro`. Al substituir el valor de `X` en el segundo predicado la consulta queda como `padre(pedro,Y)` lo que produce la solución `Y = pablo`. Además, Prolog intenta encontrar otras soluciones a la consulta haciendo *backtracking* (en particular cuando el usuario presiona `';'`). Esto consiste en buscar otra solución para el último predicado y, si ya se agotaron, volver al predicado anterior y así sucesivamente. En el ejemplo anterior, solo existe una solución para cada predicado. Sin embargo la consulta también habría podido plantearse en el orden inverso:

```
?- padre(X,Y),padre(maría,X).
X = pedro
Y = pablo ;
No
```

Se obtiene el mismo resultado pero en este caso el proceso es más complejo. La primera solución para el predicado `padre(X,Y)` es `X = juan, Y = pedro`, con esto se intenta resolver `padre(maría,juan)` lo que falla, *backtracking*, la segunda solución es `X = josé, Y = pedro` con lo que se intenta resolver `padre(maría,josé)`, falla, *backtracking*, la tercera solución es `X = maría, Y = pedro` con lo que se intenta resolver `padre(maría,maría)`, también falla, *backtracking*, la cuarta solución es `X = pedro, Y = pablo` con lo que se intenta resolver `padre(maría,pedro)`, funciona, se muestra el resultado. Si el usuario presiona `';'`, *backtracking*, la quinta solución para `padre(X,Y)` es `X = ana, Y = alberto` con lo que se intenta resolver `padre(maría,ana)`, falla y ya no hay más soluciones. Este proceso de búsqueda de la solución se puede explicitar intercalando un predicado para desplegar resultados intermedios como sigue:

```
?- padre(X,Y),print([X,Y,padre(maría,X)]),padre(maría,X).
[juan, pedro, padre(maría, juan)][josé, pedro, padre(maría, josé)][maría, pedro, padre(maría, maría

X = pedro
Y = pablo ;
[ana, alberto, padre(maría, ana)]
No
```

4 Reglas o programas

Prolog permite abstraer conceptos como la relación abuelo paterno definiendo **reglas** que precisan las condiciones en que se cumple la relación. Estas **reglas** constituyen una definición por comprensión que se pueden agregar a la base de conocimiento:

```
abuelo_paterno(X,Y) :- padre(X,Z),padre(Z,Y).
```

Una regla puede verse como una consulta empaquetada. El primer término corresponde a la relación que se está definiendo. La parte a la derecha de `':-'` indica bajo qué condiciones se cumple la relación definida, es decir, la consulta que se debe hacer. La definición termina con `'.'` . Un hecho corresponde a un caso particular de regla en que no hay condiciones en la parte derecha, lo que puede escribirse como:

```
padre(juan,pedro) :- true.
```

Las consultas relativas a relaciones definidas por comprensión mediante **reglas** se realizan de la misma manera que en el caso de definiciones por extensión con hechos. Por ejemplo:

```
?- abuelo_paterno(maría,X).
X = pablo ;
No
```

Decimos entonces que las REGLAS determinan interacciones lógicas del tipo: `si` ocurre `q, r, s y t` entonces `P`

5 Asignación vs. Instanciamiento

5.1 Igualdad y asignación

Disponemos de cuatro tipos de operadores de 'igualdad':

- Igualdad aritmética `[=:=]`. Comprueba la igualdad numérica de las expresiones argumento.

  ```
  igual1(X, Y) :- X =:= Y.
  ```

- Identidad `[==]`. Comprueba si los términos argumento son idénticos.

  ```
  igual2(X, Y) :- X == Y.
  ```

- Unificación `[ = ]`. Comprueba si los términos argumento son unificables (equiparables). Es equivalente a la asignación directa entre variables en un lenguaje procedimental. Da fallo si la unificación no es posible.

  ```
  igual3(X, Y) :- X = Y.
  ```

  Una definición equivalente sería:

  ```
  igual4(X,X).
  ```

- Asignación `[is]`. Evalúa la segunda expresión e intenta asignar el valor obtenido a la variable. No es conmutativo

  ```
  incremento(X,Y) :- Y is X+1.
  ```

  Una definición similar a la de igualdad sería:

  ```
  igual5(X, Y) :- X is Y.
  ```

6 Bibliografía

1. Introducción a Prolog, consultado el 5 de diciembre de 2012 en http://users.dcc.uchile.cl/ abassi/IA/Prolog.html
2. Prácticas de Prolog (IA I). en http://www.infor.uva.es/ calonso/IAI/PracticasProlog/practicas%20prolog.htm