Facultad de Ingeniería
Universidad Nacional de Jujuy

Desarrollo Sistemático de Programas

Unidad 5

Programación Lógica

Ing. Carlos A. Afranllie

Introducción

Programación Lógica

• La programación lógica es un paradigma de los lenguajes de programación en el cual los programas se consideran como una serie de aserciones lógicas.

• De esta forma, el conocimiento se representa mediante reglas, tratándose de sistemas *declarativos*.

• Una representación declarativa es aquella en la que el conocimiento está especificado, pero en la que la manera en que dicho conocimiento debe ser usado no viene dado.

• El más popular de los sistemas de programación lógica es el PROLOG.

Introducción

El lenguaje PROLOG

• El lenguaje de programación PROLOG ("**PRO**grammation en **LOG**ique") fue creado por Alain Colmerauer y sus colaboradores alrededor de 1970 en la Universidad de Marseille-Aix, si bien uno de los principales protagonistas de su desarrollo y promoción fue Robert Kowalski de la Universidad de Edimburgh.

• Las investigaciones de Kowalski proporcionaron el marco teórico, mientras que los trabajos de Colmerauer dieron origen al actual lenguaje de programación, construyendo el primer interprete PROLOG. David Warren, de la Universidad de Edimburgh, desarrolló el primer compilador de PROLOG (WAM – "Warren Abstract Machine").

La programación lógica en PROLOG

• Un programa escrito en PROLOG puro, es un conjunto de cláusulas de Horn (instrucciones ejecutables de PROLOG).

• Las cláusulas de Horn tienen el siguiente aspecto:

hija (A, B) <– mujer (A), padre (B, A).

que podría leerse:

A es hija de B si A es mujer y B es padre de A

• Sin embargo, PROLOG, como lenguaje de programación moderno, incorpora más cosas, como instrucciones de Entrada/Salida, etc.

La programación lógica en PROLOG

• Una cláusula de Horn puede ser ó bien una conjunción de hechos positivos ó una implicación con un único consecuente (un único termino a la derecha).

• La negación no tiene representación en PROLOG, y se asocia con la falta de una afirmación (negación por fallo), según el modelo de *suposición de un mundo cerrado;* solo es cierto lo que aparece en la base de conocimiento ó bien se deriva de esta.

La programación lógica en PROLOG

• En PROLOG todas las variables están implícitamente cuantificadas universalmente.

• En PROLOG existe un símbolo explicito para la conjunción "y" (,), pero no existe uno para la disyunción "o", que se expresa como una lista de sentencias alternativas.

• En PROLOG, las implicaciones p --> q se escriben al revés q :- p, ya que el interprete siempre trabaja hacia atrás sobre un objetivo, como se verá más adelante.

Predicados

• Se utilizan para expresar propiedades de los objetos, *predicados monádicos.*

• Para expresar relaciones entre ellos existen los *predicados poliádicos*. En PROLOG los llamaremos hechos.

Predicados

- Los nombres de todos los objetos y relaciones deben comenzar con una letra minúscula.

- Primero se escribe la relación o propiedad: *predicado*

- Los objetos se escriben separándolos mediante comas y encerrados entre paréntesis: *argumentos*.

- Al final del hecho debe ir un punto (".").

simbolo_de_predicado(arg1,arg2,...,argn).

Predicados

/* Predicados monádicos: PROPIEDADES */

```
/* mujer(Per) <- Per es una mujer    */
mujer(clara).
mujer(chela).

/* hombre(Per) <- Per es un hombre   */
hombre(jorge).
hombre(felix).
hombre(borja).

/* moreno(Per) <- Per tiene el pelo de color oscuro */
moreno(jorge).
```

Predicados

/* Predicados poliádicos: RELACIONES */

```
/* tiene(Per,Obj) <- Per posee el objeto Obj */
tiene(jorge,moto).

/* le_gusta_a(X,Y) <- a X le gusta Y */
le_gusta_a(clara,jorge).
le_gusta_a(jorge,clara).
le_gusta_a(jorge,informatica).
le_gusta_a(clara,informatica).
```

Predicados

/* Predicados poliádicos: RELACIONES */

/* es_padre_de(Padre,Hijo-a) <- Padre es el padre de Hijo-a */
es_padre_de(felix,borja).
es_padre_de(felix,clara).

/* es_madre_de(Madre,Hijo-a) <- Madre es la madre de Hijo-a */
es_madre_de(chela,borja).
es_madre_de(chela, clara).

/* regala(Per1,Obj,Per2) <- Per1 regala Obj a Per2 */
regala(jorge, flores, clara).

Términos

• Los términos pueden ser *constantes* o *variables*, y suponemos definido un dominio no vacío en el cual toman valores.

• Las constantes se utilizan para dar nombre a objetos concretos del dominio, representan individuos conocidos de nuestro Universo. Además, las constantes atómicas de PROLOG también se utilizan para representar propiedades y relaciones entre los objetos del dominio.

• Las variables se utilizan para representar objetos cualesquiera del Universo u objetos desconocidos en ese momento, es decir, son las incógnitas del problema.

Términos - Constantes

Hay dos clases de constantes:

Átomos: existen tres clases de constantes atómicas:

- Cadenas de letras, dígitos y subrayado (_) empezando por letra minúscula.
- Cualquier cadena de caracteres encerrada entre comillas simples (').
- Combinaciones especiales de signos: "?-", ":-", ...

Números: se utilizan para representar números de forma que se puedan realizar operaciones aritméticas. Dependen del computador y la implementación.

Términos - Constantes

Números:

• *Enteros*: en la implementación de PROLOG-2 puede utilizarse cualquier entero que el intervalo

[-223,223-1]=[-8.388.608,8.388.607].

• *Reales*: decimales en coma flotante, consistentes en al menos un dígito, opcionalmente un punto decimal y más dígitos, opcionalmente *E*, un «+» o «-» y más dígitos.

Términos - Constantes

átomos válidos	átomos no válidos
f	2mesas
vacio	Vacio
juan_perez	juan-perez
'Juan Perez'	_juan
a352	352a

n° válidos	nº no válidos
-123	123-
1.23	.2
1.2E3	1.
1.2E+3	1.2e3
1.2E-3	1.2+3

Términos - Variables

• Se diferencian de los átomos en que ***empiezan siempre con una letra mayúscula*** o con el signo de subrayado (_).

• Así, deberemos ir con cuidado ya que cualquier identificador que empiece por mayúscula, será tomado por PROLOG como una variable.

• Para trabajar con objetos desconocidos cuya identidad no nos interesa, podemos utilizar la *variable anónima* (_).

Términos - Variables

Ejemplos de variables:

X
Sumando
Primer_factor
_indice
_ (variable anónima)

Una variable está *instanciada* cuando existe un objeto determinado representado por ella. Y está *no instanciada* cuando todavía no se sabe lo que representa la variable. PROLOG no soporta asignación destructiva de variables, es decir, cuando una variable es instanciada su contenido no puede cambiar.

Conectivas Lógicas

• Puede que nos interese trabajar con sentencias más complejas, fórmulas moleculares, que constarán de fórmulas atómicas combinadas mediante conectivas.

• Las conectivas que se utilizan en la Lógica de Primer Orden son:

- *conjunción*
- *disyunción*
- *negación*
- *implicación*.

Conectivas Lógicas

La conjunción, “y”, la representaremos poniendo una coma entre los objetivos “,” y consiste en *objetivos* separados que PROLOG debe satisfacer, uno después de otro:

X , Y

Cuando se le da a PROLOG una secuencia de objetivos separados por comas, intentará satisfacerlos por orden, buscando objetivos coincidentes en la Base de Datos. Para que se satisfaga la secuencia se tendrán que satisfacer todos los objetivos.

Conectivas Lógicas

La disyunción, “o”, tendrá éxito si se cumple alguno de los objetivos que la componen. Se utiliza un punto y coma “;” colocado entre los objetivos:

X ; Y

La disyunción lógica también la podemos representar mediante un conjunto de sentencias alternativas, es decir, poniendo cada miembro de la disyunción en una cláusula aparte.

Conectivas Lógicas

La negación lógica no puede ser representada explícitamente en PROLOG, sino que se representa implícitamente por la falta de aserción: “no”, tendrá éxito si el objetivo *X* fracasa.

No es una verdadera negación, en el sentido de la Lógica, sino una negación “por fallo”. Se representa con el predicado predefinido *not* o con \+:

not(X)
\+ X

Reglas (implicación)

Las reglas se utilizan en PROLOG para significar que un hecho depende de uno ó más hechos.

Son la representación de las implicaciones lógicas del tipo:

$$p \rightarrow q \text{ (p implica q).}$$

Reglas (implicación)

- Una regla consiste en una cabeza y un cuerpo, unidos por el signo “:-”

- La cabeza está formada por un único hecho.

- El cuerpo puede ser uno ó más hechos (conjunción de hechos), separados por una coma (“,”), que actúa como el "y" lógico.

- Las reglas finalizan con un punto (“.”).

Reglas (implicación)

• La implicación o condicional, sirve para significar que un hecho depende de un grupo de otros hechos.

• En castellano solemos utilizar las palabras “si ... entonces ...”.

• En PROLOG se usa el símbolo “:-” para representar lo que llamamos una regla:

cabeza_de_la_regla :- cuerpo_de_la_regla.

Reglas (implicación)

La cabeza describe el hecho que se intenta definir; el cuerpo describe los objetivos que deben satisfacerse para que la cabeza sea cierta. Así, la regla:

C :- O1, O2, ..., On.

puede ser leída declarativamente como:

"La demostración de la cláusula *C* se sigue de la demostración de los objetivos *O1, O2, ..., On.*"

El ámbito de las variables

Cuando en una regla aparece una variable, el *ámbito* de esa variable es únicamente esa regla. Supongamos las siguientes reglas:

```
hermana_de(X, Y):-mujer(X),padres(X, M, P),padres(Y, M, P).
puede_robar(X, P) :- ladron(X), le_gusta_a(X, P), valioso(P).
```

Aunque en ambas aparece la variable X (y la variable P), no tiene nada que ver la X de la regla (1) con la de la regla (2), y por lo tanto, la instanciación de la X en (1) no implica la instanciación en (2).
Sin embargo todas las X de **una misma regla** sí que se instanciarán con el mismo valor.

Operadores

• Son predicados predefinidos en PROLOG para las operaciones matemáticas básicas.

• Su sintaxis depende de la posición que ocupen, pudiendo ser infijos ó prefijos. Por ejemplo el operador suma ("+"), podemos encontrarlo en forma prefija '+(2,5)' ó bien infija, '2 + 5'.

También disponemos de predicados de igualdad y desigualdad.

X = Y	igual	X \= Y	distinto
X < Y	menor	X > Y	mayor
X =< Y	menor ó igual	X >= Y	mayor ó igual

El operador “is”

Es un operador infijo, que en su parte derecha lleva un término que se interpreta como una expresión aritmética, contrastándose con el término de su izquierda.

Por ejemplo, la expresión '6 is 4+3.' es falsa. Por otra parte, si la expresión es 'X is 4+3.', el resultado será la instanciación de X:

X = 7

Una regla PROLOG puede ser esta:

```
densidad(X,Y) :- poblacion(X,P), area(X,A), Y is P/A.
```

Programando en Prolog

Con los datos que conocemos, ya podemos construir un programa en PROLOG.

Necesitaremos un editor de textos para escribir los hechos y reglas que lo componen.

Un ejemplo sencillo de programa PROLOG es el siguiente:

```
quiere_a(maria,enrique).
quiere_a(juan,jorge).
quiere_a(maria,susana).
quiere_a(maria,ana).
quiere_a(susana,pablo).
quiere_a(ana,jorge).
```

Programando en Prolog

```
hombre(juan).
hombre(pablo).
hombre(jorge).
hombre(enrique).
mujer(maria).
mujer(susana).
mujer(ana).
teme_a(susana,pablo).
teme_a(jorge,enrique).
teme_a(maria,pablo).
```

Programando en Prolog

```
/*  Esta linea es un comentario  */
quiere_pero_teme_a(X,Y) :- quiere_a(X,Y), teme_a(X,Y).
querido_por(X,Y) :- quiere_a(Y,X).
puede_casarse_con(X,Y) :- quiere_a(X,Y), hombre(X),
mujer(Y).
puede_casarse_con(X,Y) :- quiere_a(X,Y), mujer(X),
hombre(Y).
```

Una vez creado, lo salvaremos para su posterior consulta desde el interprete PROLOG. Un programa PROLOG tiene como extensión por defecto '.PRO'. Le daremos el nombre 'relacion.pro'.

Comando básico “consult”

consult: El predicado *consult* está pensado para leer y compilar un programa PROLOG ó bien para las situaciones en las que se precise añadir las clausulas existentes en un determinado fichero a las que ya están almacenadas y compiladas en la base de datos.

Su sintaxis puede ser una de las siguientes:

```
consult(fichero).
consult('fichero.ext').
consult('c:\ia\prolog\fichero').
```

Resolución de objetivos

Ya hemos creado un programa PROLOG [relacion.pro] y lo hemos compilado en nuestro interprete PROLOG [consult(relacion)].

A partir de este momento podemos interrogar la base de datos, mediante consultas.

Una consulta tiene la misma forma que un hecho. Consideremos la pregunta:

```
?- quiere_a(susana,pablo).
```

Resolución de objetivos

PROLOG buscará por toda la base de datos hechos que *coincidan* con el anterior.

Dos hechos coinciden si sus predicados son iguales, y cada uno de sus correspondientes argumentos lo son entre sí.

Si PROLOG encuentra un hecho que coincida con la pregunta, responderá *yes*.

En caso contrario responderá *no.*

Resolución de objetivos

Además, una pregunta puede contener variables. En este caso PROLOG buscara por toda la base de hechos aquellos objetos que pueden ser representado por la variable. Por ejemplo:

```
?- quiere_a(maria, Alguien).
```

[NOTA: Alguien es un nombre perfectamente válido de variable, puesto que empieza por una letra mayúscula.]

Resolución de objetivos

El resultado de la consulta es:

Alguien = enrique
More (Y/N):

El hecho 'quiere_a(maria,enrique).' coincide con la pregunta al instanciar la variable Alguien con el objeto 'enrique'.
Por lo tanto es una respuesta válida, pero no la única. Por eso se nos pregunta si queremos obtener más respuestas.

Resolución de objetivos

En caso afirmativo, obtendríamos:

```
Alguien = susana
More (Y/N):y
Alguien = ana
More (Y/N):y
No.
```

Cuando Prolog no tenga más respuestas, nos dirá que No; esa negativa no significa que lo que preguntemos sea falso, sino que no lo conoce o no puede demostrarlo porque nuestro programa no cuenta con el conocimiento suficiente. Esto se denomina negación por falla.

El mecanismo de control en Prolog

El mecanismo empleado por PROLOG para satisfacer las cuestiones que se le plantean está compuesto por:

- *Razonamiento hacia atrás* (backward)
- *Búsqueda en profundidad* (depth first)
- *Vuelta atrás* ó *reevaluación* (backtracking).

El mecanismo de control en Prolog

Razonamiento hacia atrás: Partiendo de un objetivo a probar, busca las aserciones que pueden probar el objetivo. Si en un punto caben varios caminos, se recorren en el orden que aparecen en el programa, esto es, de arriba a abajo y de izquierda a derecha.

Reevaluación: Si en un momento dado una variable se instancia con determinado valor con el fin de alcanzar una solución, y se llega a un camino no satisfactorio, el mecanismo de control retrocede al punto en el cual se instanció la variable, la desinstancia y si es posible, busca otra instanciación que supondrá un nuevo camino de búsqueda.

El mecanismo de control en Prolog

Se puede ilustrar esta estrategia sobre el ejemplo anterior. Supongamos la pregunta:

?- puede_casarse_con(maria,X).

PROLOG recorre la base de datos en busca de un hecho que coincida con la cuestión planteada. Lo que haya es la regla

puede_casarse_con(X,Y) :- quiere_a(X,Y), hombre(X), mujer(Y)., produciéndose una coincidencia con la cabeza de la misma, y una instanciación de la variable X de la regla con el objeto 'maria'.

El mecanismo de control en Prolog

Tendremos por lo tanto:
puede_casarse_con(maria,Y) :- quiere_a(maria,Y), hombre(maria), mujer(Y).

A continuación, se busca una instanciación de la variable Y que haga cierta la regla, es decir, que verifique los hechos del cuerpo de la misma. La nueva meta será:

(2) quiere_a(maria,Y).

De nuevo PROLOG recorre la base de datos. En este caso encuentra un hecho que coincide con el objetivo:
quiere_a(maria,enrique).

El mecanismo de control en Prolog

Instanciando la variable Y con el objeto 'enrique'. Siguiendo el orden dado por la regla (1), quedan por probar dos hechos una vez instanciada la variable Y:

hombre(maria), mujer(enrique).

Se recorre de nuevo la base de datos, no hallando en este caso ninguna coincidencia con el hecho 'hombre(maria)'.

Por lo tanto, PROLOG recurre a la vuelta atrás, desistanciando el valor de la variable Y, y retrocediendo con el fin de encontrar una nueva instanciación de la misma que verifique el hecho (2).

El mecanismo de control en Prolog

Un nuevo recorrido de la base de hechos da como resultado la coincidencia con:

quiere_a(maria,susana).

Se repite el proceso anterior. La variable Y se instancia con el objeto 'susana' y se intentan probar los hechos restantes:

hombre(maria), mujer(susana).

De nuevo se produce un fallo que provoca la desinstanciación de la variable Y, así como una vuelta atrás en busca de nuevos hechos que coincidan con (2).

El mecanismo de control en Prolog

Una nueva reevaluación da como resultado la instanciación de Y con el objeto 'ana' (la última posible), y un nuevo fallo en el hecho 'hombre(maria)'.

Una vez comprobadas sin éxito todas las posibles instanciaciones del hecho (2), PROLOG da por imposible la regla (1), se produce de nuevo la vuelta atrás y una nueva búsqueda en la base de datos que tiene como resultado la coincidencia con la regla:

(3) puede_casarse_con(maria,Y) :- quiere_a(maria,Y), mujer(maria), hombre(Y).

El mecanismo de control en Prolog

Se repite todo el proceso anterior, buscando nuevas instanciaciones de la variable Y que verifiquen el cuerpo de la regla. La primera coincidencia corresponde al hecho

quiere_a(maria,enrique).

que provoca la instanciación de la variable Y con el objeto 'enrique'. PROLOG tratará de probar ahora el resto del cuerpo de la regla con las instanciaciones actuales:

mujer(maria), hombre(enrique).

El mecanismo de control en Prolog

Un recorrido de la base de datos, da un resultado positivo en ambos hechos, quedando probado en su totalidad el cuerpo de la regla (3) y por lo tanto su cabeza, que no es m s que una de las soluciones al objetivo inicial.

X = enrique
More (Y/N): y

De esta forma se generarán el resto de las soluciones, si las hubiera.

Operador especial “corte”

El operador "corte": se representa por el símbolo "!" nos da un cierto control sobre el mecanismo de deducción del PROLOG.

Su función es la de controlar el proceso de reevaluación, limitándolo a los hechos que nos interesen. Supongamos la siguiente regla:

```
regla :- hecho1, hecho2, !, hecho3, hecho4, hecho5
```

Operador especial “corte”

PROLOG efectuará reevaluaciones entre los hechos 1, 2 sin ningún problema, hasta que se satisface el hecho2.

En ese momento se alcanza el hecho3, pudiendo haber a continuación reevaluaciones de los hechos 3, 4 y 5.

Sin embargo, si el hecho3 fracasa, no se intentara de ninguna forma reevaluar el hecho2.

Operador especial “corte”

Una interpretación práctica del significado del corte en una regla puede ser que "si has llegado hasta aquí es que has encontrado la única solución a este problema y no hay razón para seguir buscando alternativas".

Aunque se suele emplear como una herramienta para la optimización de programas, en muchos casos marca la diferencia entre un programa que funciona y otro que no lo hace.

Estructuras de datos: Listas

- Es una secuencia ordenada de elementos que puede tener cualquier longitud.

- Ordenada significa que el orden de cada elemento es significativo.

- Un elemento puede ser cualquier término e incluso otra lista.

- Se representa como una serie de elementos separados por comas y encerrados entre corchetes.

- Para procesar una lista, la dividimos en dos partes: la cabeza y la cola.

Estructuras de datos: Listas

Por ejemplo:

Lista	Cabeza	Cola
[a,b,c,d]	a	[b,c,d]
[a]	a	[ ] (lista vacía)
[ ]	no tiene	no tiene
[[a,b],c]	[a,b]	[c]
[a,[b,c]]	a	[[b,c]]
[a,b,[c,d]]	a	[b,[c,d]]

Estructuras de datos: Listas

Para dividir una lista, utilizamos el símbolo "|". Una expresión con la forma [X | Y] instanciará X a la cabeza de una lista e Y a la cola. Por ejemplo:

```
p([1,2,3]).
p([el,gato,estaba,[en,la,alfombra]]).
?- p([X|Y]).
X = 1,
Y = [2,3]
More (Y/N):y
X = el,
Y = [gato,estaba,[en,la,alfombra]]
```

Entrada / Salida

write: Su sintaxis es:

write('Hello world.').

Las comillas simples encierran constantes, mientras que todo lo que se encuentra entre comillas dobles es tratado como una lista.

También podemos mostrar el valor de una variable, siempre que esté instanciada:

write(X).

Entrada / Salida

nl: El predicado nl fuerza un retorno de carro en la salida. Por ejemplo:

write('linea 1'), nl, write('linea 2').

tiene como resultado:
linea 1
linea 2

Entrada / Salida

read: Lee un valor del teclado. La lectura del comando read no finaliza hasta que se introduce un punto ".". Su sintaxis es:

read(X). (Instancia la variable X con el valor leído del teclado)

read(ejemplo).

Se evalúa como cierta siempre que lo tecleado coincida con la constante entre paréntesis (en este caso 'ejemplo').