Programación Lógica

Tecnología de Programación

Adolfo Muñoz – Juan Magallón

Grado en Ingeniería Informática

Universidad Zaragoza 1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza 1542

2023-01-20

Programación Lógica

Relaciones

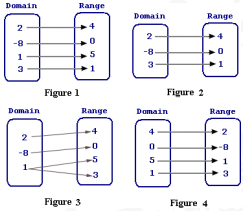
Tanto los lenguajes imperativos como los lenguajes funcionales se pueden analizar desde un punto de vista abstracto como la implementación de una **aplicación** entre los datos de entrada y los datos de salida.

La programación lógica se basa en la noción de que un programa implementa una **relación**, en vez de una aplicación.

Relaciones

La diferencia fundamental entre una **aplicación** y una **relación** es:

- Aplicación: correspondencia *muchos-a-uno*.
- Relación: correspondencia *muchos-a-muchos*.



De manera informal, las relaciones no tienen sentido de dirección ni prejuicio alguno acerca de qué se calcula a partir de qué (son más generales)

Cálculo de predicados y prueba de teoremas

Con el inicio de la era de los computadores (años 50) apareció la preocupación por la automatización del proceso de prueba de teoremas. Quizás el resultado mas esperanzador fue el descubrimiento del principio de resolución de Alan Robinson (1965).

Resolución es una regla de inferencia que permite deducir proposiciones que pueden calcularse a partir de otras proposiciones dadas.

Programación Lógica

La Programación Lógica trabaja con hechos y proposiciones.

Ejemplo:

```
1  parent(john,peter).
2  parent(john,ann).
3  parent(mary,ann).
4
5  brother(X,Y) :- parent(P,X) , parent(P,Y).
```

Dada una aplicación `f` (programación imperativa/funcional) se puede responder a:

- dado `(x)`, determinar `f(x)`

Dada una relación `r` (programación lógica) se puede responder a:

- dados `(a)` y `(b)`, determinar si `r(a,b)` es `verdad`
- dado `(a)`, encontrar todos los valores `(y)` tales que `r(a,y)` es `verdad`
- dado `(b)`, encontrar todos los valores `(x)` tales que `r(x,b)` es `verdad`
- encontrar todos los valores `(x)` e `(y)` tales que `r(x,y)` es `verdad`

Ningún lenguaje de programación lógico puede implementar totalmente la *lógica matemática*.

Ejemplo: el último teorema de fermat:
$fermat(n)$ si y sólo si existen $a \in \mathbb{N}$, $b \in \mathbb{N}$ y $c \in \mathbb{N}$ tales que $a^n + b^n = c^n$.

Prolog

Alain Colmerauer y Phillippe Roussel del Grupo de IA de la Universidad de Marseille y Robert Kowalski del Departamento de IA de la Universidad de Edimburgo trabajaron en colaboración para implementar una forma restringida de resolución del cálculo de predicados, que dió lugar a la primera versión de un intérprete de Prolog en 1972.

Desde entonces dicho lenguaje a progresado de forma independiente dando lugar a unas cuantas versiones diferentes entre si.

Por ejemplo:

http://www.gprolog.org/
http://www.swi-prolog.org/

Un programa en Prolog maneja identificadores simbólicos:

- constantes: empiezan con minúscula
- variables: empiezan con mayúscula
- nombres de predicados: empiezan con minúscula

A las constantes y variables se les llama en general términos.

- Las variables en Prolog son como las de los lenguajes funcionales, algo de lo que desconocemos su valor, y que una vez fijado no cambia.
- Existe un nombre de variable anónima, el símbolo '_', que no toma ningún valor.

Un programa en Prolog está compuesto por **cláusulas**.

Una cláusula representa una relación o regla:

```
1  brother(X,Y) :- parent(P,X) , parent(P,Y).
```

Una cláusula puede representar un regla que siempre se cumple:

```
1  parent(john,peter) :- True.
```

por lo que se le llama **hecho** y se puede abreviar como:

```
1  parent(john,peter).
```

Con los elementos anteriores podemos definir los **datos** y **relaciones** de nuestro programa.

- Mediante **hechos** definimos nuestra base de datos y con las cláusulas nuestras funciones para combinarlas.
- A partir de él podemos intentar responder distintas preguntas, a las que se llama **objetivos**.

Las preguntas más simples se reducen a evaluaciones de la base de datos

```
?- parent(mary,ann).
yes
?- parent(mary,john).
no
```

Pero la ventaja es que podemos hacer preguntas no tan sencillas:

```
?- parent(X,peter).
X = john

?- parent(X,ann).
X = john
X = mary
```

```
?- parent(john,Y).
Y = peter
Y = ann

?- parent(john,_).
true
```

Considérense estos hechos:

```
estrella(sol).
estrella(sirio).
estrella(alpheratz).
estrella(betelgeuse).
estrella(aldebaran).
```

Se pueden resolver las siguientes preguntas:

```
?- estrella(sol).
yes
?- estrella(jupiter).
no
```

Definimos ahora una serie de hechos binarios (*orbita*):

```
1  orbita(mercurio, sol).
2  orbita(venus, sol).
3  orbita(tierra, sol).
4  orbita(marte, sol).
5  orbita(luna, tierra).
6  orbita(phobos, marte).
7  orbita(deimos, marte).
```

Se pueden resolver las siguientes preguntas:

```
?- orbita(marte,sol).
yes
?- orbita(luna, sol).
no
?- orbita(phobos, B).
B=marte
?- orbita(B, marte).
B=phobos
B=deimos
?- orbita(B, venus).
no
```

Definimos ahora una nueva cláusula en base a los hechos:

```
planeta(B) :- estrella(E) , orbita(B,E).
```

Se pueden resolver las siguientes preguntas:

```
?- planeta(marte).
yes
?- planeta(P).
P=mercurio
P=venus
P=tierra
P=marte
```

Definimos ahora una nueva cláusula en base a los hechos:

```
satelite(B) :- planeta(P) , orbita(B,P).
```

Se pueden resolver las siguientes preguntas:

```
?- satelite(phobos).
yes
?- satelite(S).
S=luna
S=phobos
S=deimos
```

Definimos ahora una nueva cláusula recursiva:

```
1  solar(sol).
2  solar(B) :- orbita(B,S), solar(S)
```

Se pueden resolver las siguientes preguntas:

```
1  ?- solar(sol).
2  yes
3  ?- solar(sirio).
4  no
5  ?- solar(venus).
6  yes
7  ?- solar(luna).
8  yes
```

El operador '=' en Prolog se comporta como un predicado más. El sistema busca valores que hagan cierto el resultado:

```
?- X = a.
X = a
yes
?- X = a, X = b.
no
?- X = a, Y = X.
X = a
Y = a
yes
```

La estructura de datos básica en Prolog son las listas. El predicado de igualdad funciona:

```
?- [X,Y,c,d] = [a,b,c,d].
X = a
Y = b

?- [X,Y,c,d] = [a,b,c,e].
no
```

```
?- [H|T] = [a,b,c,d].
H = a
T = [b,c,d]

?- [X|_] = [a,b,c,d].
X = a
```

Podemos definir operaciones con listas:

```
1  lstapp(L1,L2,R) :-
2      L1 = [],
3      R  = L2.
4
5  lstapp(L1,L2,R) :-
6      L1 = [H|T],
7      lstapp(T,L2,Temp),
8      R  = [H|Temp].
```

Que nos permiten buscar distintos objetivos:

```
?- lstapp([a,b],[c,d],R).
R = [a,b,c,d]

?- lstapp([a,b],X,[a,b,c,d]).
X = [c,d]
```

Incluso algunos irresolubles:

```
?- lstapp(X,Y,[a,b,c,d]).

X = []
Y = [a,b,c,d] ? a

X = [a]
Y = [b,c,d]

X = [a,b]
Y = [c,d]

X = [a,b,c]
Y = [d]

X = [a,b,c,d]
^CY = []
Prolog interruption (h for help) ?
```

Recursividad:

```
1  longitud([],0).
2  longitud([_|T],N) :- longitud(T,N0), N is N0 + 1.
3
4  ?- longitud([a,b,c],L).
5  L = 3
6
7  ?- longitud([a,b,c],4).
8  no
```

Ejemplos

```
1  factorial(0,1).
2
3  factorial(N,F) :-
4     N>0,
5     N1 is N-1,
6     factorial(N1,F1),
7     F is N * F1.
```

```
1  ?- factorial(4,N).
2  N = 24
```

```
1  factorial(0,1).
2
3  factorial(N,F) :-
4     N>0,
5     N1 is N-1,
6     factorial(N1,F1),
7     F is N * F1.
```

```
1  ?- factorial(4,N).
2  N = 24
```

```
1  ?- factorial(X,24).
2  ERROR: >/2: Arguments are not sufficiently instantiated
```

Programación Lógica

Tecnología de Programación

Adolfo Muñoz – Juan Magallón

Grado en Ingeniería Informática

Universidad Zaragoza 1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza 1542

2023-01-20