Metaintérpretes

Lógica para Ciencias de la Computación

Primer Cuatrimestre de 2008

– Material Adicional –

¿Qué es un Metaintérprete?

- Denominaremos metaintérprete a un intérprete para un lenguaje de programación que esté escrito en ese mismo lenguaje.

Ejemplos de Metaintérpretes

- Un intérprete para *Java* escrito en *Java*.
  - 100.000 líneas de código o más…
- Un intérprete para *Pascal* escrito en *Pascal*.
  - 20.000 líneas de código o más…
- Un intérprete para *Prolog* escrito en *Prolog.*
  - Exactamente 3 líneas de código!!!

Hechos como Reglas

- Recordar que al definir el intérprete abstracto para Prolog, consideramos a los hechos, por uniformidad, como reglas con cuerpo vacío.
- En realidad, Prolog también interpreta/considera a los hechos como un caso particular de las reglas.
- Concretamente, admite dos notaciones alternativas (equivalentes) para los hechos:
  - la tradicional: `r(obj1,…,objn).`
  - como una regla: `r(obj1,…,objn):- true.`

  donde `true/0` es un predicado predefinido que siempre tiene éxito.
- Por ejemplo, `p(a).` y `p(a):-true.` denotan el mismo hecho (verificarlo en Prolog).
- En definitiva, Prolog trata uniformemente a los hechos como reglas cuyo cuerpo siempre se satisface.

Conjunción de Metas

- En Prolog, la *coma* (“,”) se utiliza para separar los predicados que componen el cuerpo de una regla, y su significado es el de un *and* o conjunción. Ejemplo:
  - `a :- b,c,d,e.`
- También se utiliza para denotar una conjunción de (sub)metas (o meta conjuntiva) en una consulta. Ejemplo:
  - `?- p,q,r.`
- Prolog interpreta a los principales simbolos del lenguaje (“`,`”, “`:-`”, “`;`”, “`?-`”) como operadores, definiendo precedencia y asociativodad para cada uno de ellos.
- En particular, Prolog interpreta a la “`,`” como un operador con asociatividad a derecha.
- De esta forma, la conjunción de predicados `p,q,r,s` se interpretará como `(p,(q,(r,s)))`.

Código del Metaintérprete

```
1. solve(true).

2. solve( (A, B) ) :-
        solve(A), solve(B).

3. solve(A) :-
        clause(A, B), solve(B).
```

Análisis del Código

1. `solve(true).`

   `true` siempre se debe satisfacer. En particular, constituye el caso base del planteo recursivo.

2. `solve( (A, B) ) :-`
   `solve(A), solve(B).`

   La meta conjuntiva `(A, B)` se satisface sólo si se satisface tanto `A` como `B`.

Análisis del Código

3. solve(A) :-
   clause(A, B), solve(B).

   Si la meta *A* es la cabeza de una regla con cuerpo B, entonces satisfacer *A* es equivalente a satisfacer (recursivamente) a todas las metas del cuerpo B.

   clause/2 es un predicado predefinido que determina si en el programa Prolog actual existe alguna regla con la cabeza y el cuerpo suministrados como argumentos.

Uso del predicado Clause/2

```
p(X) :- q(X).
q(a).

p(b) :- r(b), s(b).
```

```
?- clause( p(a), Cuerpo).
Cuerpo = q(a)
?- clause( p(b), Cuerpo).
Cuerpo = q(b);
Cuerpo = r(b), s(b)
?- clause( q(a), Cuerpo).
Cuerpo = true
```

Aplicaciones

- Monitorear el progreso de la resolución de una consulta.
- Extender al lenguaje mediante nuevos predicados predefinidos.
- Modificar la semántica de alguna de las partes del lenguaje.
- No es gratis: se paga un costo en términos de eficiencia.

Ejemplo Concreto

Definir un metaintérprete que cuente la cantidad de hechos visitados a lo largo de la resolución de una cierta consulta:

```
solve(true, 1).

solve( (A, B), CHAB ) :-
        solve(A, CHA), solve(B, CHB),
        CHAB is CHA + CHB.

solve(A, CH) :-
        clause(A, B), solve(B, CH).
```