Operador de corte y listas diferenciables..
Lógica Computacional 2017-2

Lourdes del Carmen González Huesca
Roberto Monroy Argumedo
Fernando A. Galicia Mendoza

Facultad de ciencias,UNAM

Miércoles,29 de marzo del 2017

Operador de corte

Recordemos que `Prolog` busca una solución a partir de árboles (de búsqueda) utilizando la técnica de backtracking.
Supongamos que tenemos una relación la cual sabemos que ciertas ramas obtenidas son falsas o cuyo resultado ya se había calculado.
Esto causa que se vuelva ineficiente un programa, para resolver esto `Prolog` cuenta con un operador que le indica al interprete que esos casos ya no deben ser analizados, tal operador es llamado *operador de corte*.
Cuya sintaxis es `!` y es un operador sin parámetros.

Ejemplo

Ejercicio: Definan una relación `minimo`(X, Y, Z) que es cierta syss Z es el mínimo de los números X e Y.

Ejemplo

Ejercicio: Definan una relación `minimo`(X, Y, Z) que es cierta syss Z es el mínimo de los números X e Y.Utilizando operador de corte.

```
minimoC(X,Y,X) :- X ≤ Y,!.
minimoC(X,Y,Y) :- X > Y,!.
```

Analicemos el caso de `minimo(5,7,X)` y `minimoC(5,7,X)`.

Ejemplo

Ejercicio: Definan una relación `minimo`(X, Y, Z) que es cierta syss Z es el mínimo de los números X e Y.Utilizando operador de corte.

```
minimoC(X,Y,X) :- X ≤ Y,!.
minimoC(X,Y,Y) :- X > Y,!.
```

Analicemos el caso de `minimo(5,7,X)` y `minimoC(5,7,X)`.
Ejercicio: Piensen e implementen 3 casos mas donde sería buena opción utilizar el operador de corte.

Ejemplo

Ejercicio: Definan una relación `minimo`(X, Y, Z) que es cierta syss Z es el mínimo de los números X e Y.Utilizando operador de corte.

```
minimoC(X,Y,X) :- X ≤ Y,!.
minimoC(X,Y,Y) :- X > Y,!.
```

Analicemos el caso de `minimo(5,7,X)` y `minimoC(5,7,X)`.
Ejercicio: Piensen e implementen 3 casos mas donde sería buena opción utilizar el operador de corte.Algunas relaciones podrían ser:

- Una que determine el signo de un número.
- Los operadores booleanos (conjunción y disyunción).

Un ejemplo interesante

Primero analicemos la siguiente relación:

```
mist(_,0,1).
mist(X,Y,Z) :- par(Y),D is Y div 2,mist(X,D,R),Z is R^2.
mist(X,Y,Z) :- D is Y-1,mist(X,D,R),Z is R*X.
```

Respondan las siguientes preguntas:

- ¿Qué está haciendo la función?
- ¿Cómo utilizarían el operador de corte para que no haga resoluciones innecesarias?

Listas diferenciables

Recordemos que una lista en `Prolog` se puede ver como `[X|XS]` donde `X` es la cabeza de la lista y `XS` la cola. Esta representación puede ser mas precisa por ejemplo si queremos analizar los dos primeros elementos esto se representa como `[X1,X2|XS]`.
Una notación cómoda en `Prolog` es el uso de listas diferenciables, si tenemos la lista `[1,2,3]` puede ser vista como `[1,2,3|X]-X`, es decir, la diferencia de la lista `[1,2,3]` con la lista `X`, de ahí el nombre de diferenciable.

Conveniente

La relación que determina si una lista es la concatenación de otras dos es de orden $O(n)$ con n la longitud de la segunda lista, esto puede ser mejorado utilizando la siguiente expresión:

```
concat2(X-Y,Y-Z,X-Z).
```

Con esto la complejidad de la concatenación se reduce a $O(1)$, ya que, literalmente se reduce a la unificación del término `Y` con el término `Y`.

Lista diferenciable $\neq$ Lista

Hay que tener cuidado en que una lista diferencial **no** es una lista, ya que, el operador - es realmente azúcar sintáctica de la relación `diff(L1,L2)`.
Sin embargo, podemos dar una representación para que sea más cómoda la visualización del resultado. . . ejercicio.
Por ejemplo la concatenación de `[1,2,3]` y `[4,5,6]` de el siguiente resultado:

```
X = [4, 5, 6],
Z = [1, 2, 3, 4, 5, 6].
```

¿Qué otros ejemplos se les ocurre utilizando listas diferenciables?

Lista diferenciable $\neq$ Lista

Hay que tener cuidado en que una lista diferencial **no** es una lista, ya que, el operador - es realmente azúcar sintáctica de la relación `diff(L1,L2)`.
Sin embargo, podemos dar una representación para que sea más cómoda la visualización del resultado. . . ejercicio.
Por ejemplo la concatenación de `[1,2,3]` y `[4,5,6]` de el siguiente resultado:

```
X = [4, 5, 6],
Z = [1, 2, 3, 4, 5, 6].
```

¿Qué otros ejemplos se les ocurre utilizando listas diferenciables?La función `concat` es su práctica. . . ni se molesten en mencionarla.

Doble recursión o acumulador

La relación `flatten` quedó como tarea moral, que un buen samaritano pase a implementarla.

Doble recursión o acumulador

La relación `flatten` quedó como tarea moral, que un buen samaritano pase a implementarla.
Recordemos que en programación funcional la recursión de cola (utilizando acumuladores) es mas eficiente que su versión en otro tipo de recursiones, ya que, el proceso se va almacenando en uno de los parámetros de la función y su acceso es constante.
Bien, en programación lógica tiene un conveniente similar, ya que, gracias al proceso de unificación almacenar el resultado es simplemente agregar al unificador mas general el acumulador utilizado, veamos el caso de flatten:

```
constant(X) :- atomic(X).

flatten(XS,YS) :- flatten(XS,[],YS).

flatten([X|XS],S,YS) :- list(X),flatten(X,[XS|S],YS).
flatten([X|XS],S,[X|YS]) :- constant(X),X \= [],
                            flatten(XS,S,YS).
flatten([],[X|S],YS) :- flatten(X,S,YS).
flatten([],[],[]).
list([_|_]).
```

Expliquen el código.

La mejor parte es que la versión de `flatten` utilizando listas diferenciables es más cómoda de escribir y también resulta mas eficiente que la anterior.

Relación call

La relación `call` es un metapredicado, en el sentido que no tiene un número de parámetros exactos. Está acotado de 2 parámetros en adelante. Como su nombre lo indica, `call(?Meta,+Arg1,...)` concatena `+Arg1,+Arg2,...` al argumento `?Meta` y **llama** al resultado.
Por ejemplo:

```
call(plus(1),2,X)
```

Lo anterior llama a `plus(1,2,X)`.
Hagamos unos cuantos ejemplos básicos.