Pontíficia Universidad Católica de Chile 6 de junio de 2017
Exploratorio de Computación. IIC-1005 Leonardo Olivares

Tarea Chica 3. Prolog

Parte 1. Explicación

Prolog es un lenguaje de programación logística, que se diferencia de otros lenguajes de programación por ser de tipo declarativo, es decir, se presentan reglas y hechos en lugar de instrucciones, y a partir de estas se obtienen demostraciones y relaciones respectivas.

En la primera parte de la tarea, se pedia definir el predicado *es_arbol(X),* de tal modo que retornara si *X,* era efectivamente un árbol binario. Para esto, se implemento el siguiente codigo en Prolog.

```
es_arbol(nil).
es_arbol(arbol(Valor,Izquierda,Derecha)) :- es_arbol(Izquierda) , es_arbol(Derecha).
```

Prolog funciona en gran parte de manera recursiva. Es por esto, que en la primera línea de código se estableció un “caso base”, en el cual se definió como una regla, que “nil” seria considerado un árbol binario. A partir de este caso base, se procedió a definir el “caso recursivo” . Para este se estableció que para confirmar que algun *X* es un árbol binario, es necesario confirmar si sus ramas (izquierda y derecha) son arboles binarios. Entonces, al ejecutar el código, con algún árbol, se verifica que cada rama del árbol principal tenga un árbol binario a su vez. Entonces si el código llegaba a verificar que siempre culminaban las ramas del arbol en (nil,nil) entonces, efectivamente era un aárbol binario.

Al ingresar la consulta en terminal *es_arbol(X)* prolog comenzaba a entregar posibles respuestas que cumplen con el predicado, es decir posibles arboles binarios. Se muestra el output de esta consulta a continuación.

```
?- es_arbol(X).
X = nil ;
X = arbol(_8156, nil, nil) ;
X = arbol(_8156, nil, arbol(_8164, nil, nil)) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, nil))) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, nil)
))) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, arbo
l(_8188, nil, nil))))) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, arbo
l(_8188, nil, arbol(_8196, nil, nil)))))) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, arbo
l(_8188, nil, arbol(_8196, nil, arbol(_8204, nil, nil)))))))) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, arbo
l(_8188, nil, arbol(_8196, nil, arbol(_8204, nil, arbol(_8212, nil, nil)))))))))
;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, arbo
l(_8188, nil, arbol(_8196, nil, arbol(_8204, nil, arbol(_8212, nil, arbol(_8220,
 nil, nil))))))))) ;
X = arbol(_8156, nil, arbol(_8164, nil, arbol(_8172, nil, arbol(_8180, nil, arbo
l(_8188, nil, arbol(_8196, nil, arbol(_8204, nil, arbol(_8212, nil, arbol(_8220,
 nil, arbol(..., ..., ...)))))))))) 
```

Se observa en el output que efectivamente se mostraban posibles arboles binarios, que cumplian con el predicado, sin embargo, cada nueva posible solución se extendía hacia una rama solamente, mientras que la rama izquierda principal se mantuvo igual. Esto ocurre por lo mencionado anteriormente. Prolog trabaja de manera recursiva, por lo que al realizar esta consultac parte del caso base y comienza a verificar que al añadir nuevas ramas cumpla con la definición de árbol binario. Utilizando la función *trace* de Prolog, se muestra cual es el proceso que prolog realiza.

```
[trace]  ?- es_arbol(X).
   Call: (8) es_arbol(_12036) ? creep
   Exit: (8) es_arbol(nil) ? creep
X = nil ;
   Redo: (8) es_arbol(_12036) ? creep
   Call: (9) es_arbol(_12238) ? creep
   Exit: (9) es_arbol(nil) ? creep
   Call: (9) es_arbol(_12240) ? creep
   Exit: (9) es_arbol(nil) ? creep
   Exit: (8) es_arbol(arbol(_12236, nil, nil)) ? creep
X = arbol(_12236, nil, nil) ;
   Redo: (9) es_arbol(_12240) ? creep
   Call: (10) es_arbol(_12246) ? creep
   Exit: (10) es_arbol(nil) ? creep
   Call: (10) es_arbol(_12248) ? creep
   Exit: (10) es_arbol(nil) ? creep
   Exit: (9) es_arbol(arbol(_12244, nil, nil)) ? creep
   Exit: (8) es_arbol(arbol(_12236, nil, arbol(_12244, nil, nil))) ? creep
X = arbol(_12236, nil, arbol(_12244, nil, nil))
```

Prolog inicia intentando con el árbol más simple, según las reglas establecidas, en este caso “nil”, y procede a verificar que efectivamente sea un árbol binario. Una vez confirmado, retorna “nil” indicando que sería una posible opcion. Sin embargo, al exigir una segunda respuesta, Prolog pasa a utilizar el caso recursivo para conseguir otra solución, entonces simula que no obtuvo resultado con el caso base y pasa al siguiente. Una vez que se encuentra en el recursivo, utiliza la definicion de arbol como *arbol(Valor,Izquierda, Derecha)*, y le otorga un valor cualquiera a *Valor*, mientras que vuelve a aplicar el caso base para ambas ramas del árbol. Luego, procede a verificar que sea un arbol binario y retorna la segunda solución. En el siguiente caso, se repite el proceso, debido a que retoma la solución anterior pero intenta cambiando la rama derecha por el caso recursivo (es decir *arbol(Valor,Izquierda, Derecha)*), y vuelve a confirmar que se trata de un árbol binario.

Como hay infinitas posibles soluciones, prolog se mantiene añadiendo nodos al mismo lado, y sigue obteniendo arboles que cumplen con el predicado. Es por esto que termina en un loop, y no es posible obtener todos los posibles arboles (es decir, incluyendo las ramas del lado izquierdo de los arboles), a menos que se altere el código para lograr este resultado.