Lógica en la Informática

Josefina Sierra Santibáñez

2 de marzo de 2020

Resolución SLD

Ejemplo de **programa** PROLOG `familia.pl`.

```
padre(juan, pedro).
padre(maria, pedro).

hermano(pedro, vicente).
hermano(pedro, alberto).

tio(X,Y) :- padre(X,Z), hermano(Z,Y).
```

Ejemplo de **objetivo** para el programa `familia.pl`.

```
?- tio(U,V).
```

Prolog busca la primera cláusula por orden de aparición cuyo literal positivo (la **cabeza** de la cláusula) unifique con el objetivo. En este caso es la cláusula 5

```
tio(X,Y) :- padre(X,Z), hermano(Z,Y).
```

La resolución entre el objetivo y esta cláusula nos da un nuevo objetivo

```
padre(U,Z), hermano(Z,V).
```

En este nuevo objetivo procedemos de izquierda a derecha de la misma manera, unificando `padre(U,Z)` con la cláusula 1

```
padre(juan, pedro).
```

El objetivo `padre(U,Z)` también unifica con la cláusula 2

```
padre(maria, pedro).
```

Por tanto, la cláusula 2 se guarda en la *pila de backtracking* como alternativa para el objetivo `padre(U,Z), hermano(Z,V)`.

padre(U,Z), hermano(Z,V)	2. padre(maria, pedro)
Objetivos	**Alternativas para Bactracking**

Después de resolver el objetivo `padre(U,Z), hermano(Z,V)` con la cláusula `padre(juan, pedro)`, el nuevo objetivo es `hermano(pedro,V)`.

El objetivo `hermano(pedro,V)` unifica con las cláusulas 3 y 4.

```
3. hermano(pedro, vicente).
4. hermano(pedro, alberto).
```

La cláusula 3 `hermano(pedro, vicente)` se resuelve con el objetivo hermano(pedro,V) para obtener la primera respuesta

```
U = juan
V = vicente ?
```

y la cláusula 4 se almacena en la pila de backtracking como alternativa para el objetivo hermano(pedro,V).

hermano(pedro,V) padre(U,Z), hermano(Z,V)	4. hermano(pedro, alberto) 2. padre(maria, pedro)
Objetivos	**Alternativas para Bactracking**

hermano(pedro,V) padre(U,Z), hermano(Z,V)	4. hermano(pedro, alberto) 2. padre(maria, pedro)
Objetivos	**Alternativas para Bactracking**

Si ahora pedimos más respuestas (con el punto y coma), el interprete desempila el estado que hay en la cima de la pila de backtracking: hay que resolver el objetivo `hermano(pedro,V)` con la cláusula 4, Esto le permite escribir la segunda respuesta

```
U = juan
V = alberto ?
```

Puesto que en `familia.pl` no hay más alternativas para el objetivo `hermano(pedro,V)` debajo de la cláusula 4, la pila queda así.

padre(U,Z), hermano(Z,V)	2. padre(maria, pedro)
Objetivos	**Alternativas para Bactracking**

padre(U,Z), hermano(Z,V)	2. padre(maria, pedro)
Objetivos	**Alternativas para Bactracking**

Si pedimos más respuestas, el interprete desempila el estado que hay en la cima de la pila de backtracking: hay que resolver el objetivo `padre(U,Z), hermano(Z,V)` con la cláusula 2.
El resultado es el objetivo es `hermano(pedro,V)`, que unifica con

```
3. hermano(pedro, vicente).
4. hermano(pedro, alberto).
```

La cláusula 3 `hermano(pedro, vicente)` se resuelve con el objetivo `hermano(pedro,V)` para obtener la tercera respuesta,

```
U = maria
V = vicente ?
```

y la cláusula 4 se almacena en la pila de backtracking como alternativa para el objetivo `hermano(pedro,V)`.

hermano(pedro,V)	4. hermano(pedro, alberto)
Objetivos	**Alternativas para Bactracking**

hermano(pedro,V)	4. hermano(pedro, alberto)
Objetivos	**Alternativas para Bactracking**

Si pedimos más respuestas, el interprete desempila el estado que hay en la cima de la pila de backtracking: hay que resolver el objetivo `hermano(pedro,V)` con la cláusula 4. Esto permite obtener la cuarta respuesta.

```
U = maria
V = alberto ?
```

Puesto que en el programa `familia.pl` no hay más alternativas para el objetivo `hermano(pedro,V)` debajo de la cláusula 4, la pila queda vacía: no existen más respuestas.

Comportamiento del operador corte (!)

- Dado un **programa**

```
p(...) :- ...
...
p(...) :- q1(...),..., qn(...), !, r(...).
...
p(...) :- ...
```

- un **objetivo en curso** de la forma `p(...), A1, ..., Ak.`
- **un estado E** de la **pila de backtracking** antes de empezar a tratar el literal `p(...)` del objetivo en curso.

Si la ejecución llega a !, se eliminan de la pila de backtracking todas las alternativas para tratar `p(...)` y para tratar los objetivos q1(...),..., qn(...), es decir,

- la pila vuelve al estado **E**, y
- la ejecución continúa con el objetivo `r(...), A1, ..., Ak.`

Ejemplo de uso del operador corte !

```
fact(0,1) :- !.
fact(X,F) :- X1 is X-1, fact(X1,F1), F is X*F1.
```

Desde el punto de vista lógico un corte (!) como el que hay en la primera cláusula siempre se satisface.

Pero aquí además indica que si se llama a `fact(X,Y)` con la variable `X` instanciada en 0, entonces no hay que considerar la segunda cláusula como alternativa: no hay que empilar la segunda cláusula en la pila de backtracking como opción alternativa.

Si después de una llamada a `fact(X,F)`, se piden más respuestas (con el operador ;), el operador corte (!) de la primer cláusula evita que se hagan llamadas recursivas a `fact(X,F)` en las que `X` está instanciada con un entero negativo.

Observad que las llamadas a `fact(X,F)` con `X` instanciada con enteros negativos no terminan.

Este tipo de uso del corte se aplica en casos en los que verde existe una única solución.

```
p(1).
p(2).

q(a).
q(b).

r(3, 4, 5).
r(X, Y, Z) :- p(X), q(Y), !, s(Z).
r(5, 6, 7).

s(3).
s(4).

h(X, Y, Z) :- r(X, Y, Z).
h(a, b, c).

?- h(A,B,C), write([A,B,C]), nl, fail.}
[3,4,5]
[1,a,3]
[1,a,4]
[a,b,c]
false.
```

Se eliminan de la pila de backtracking las cláusulas:

p(2), q(b), porque los objetivos p(X), q(Y) están a la izquierda de **!**

r(5,6,7), porque está debajo de la cláusula para r/3 que contiene **!**