



Tecnologías de Programación

Paradigma Lógico – ProLog

Corte



El Corte

- El corte es un predicado predefinido utilizado para prevenir el backtracking.
- Extiende el poder de expresividad de Prolog.
- Permite definir un tipo de negación: La negación como fallo.



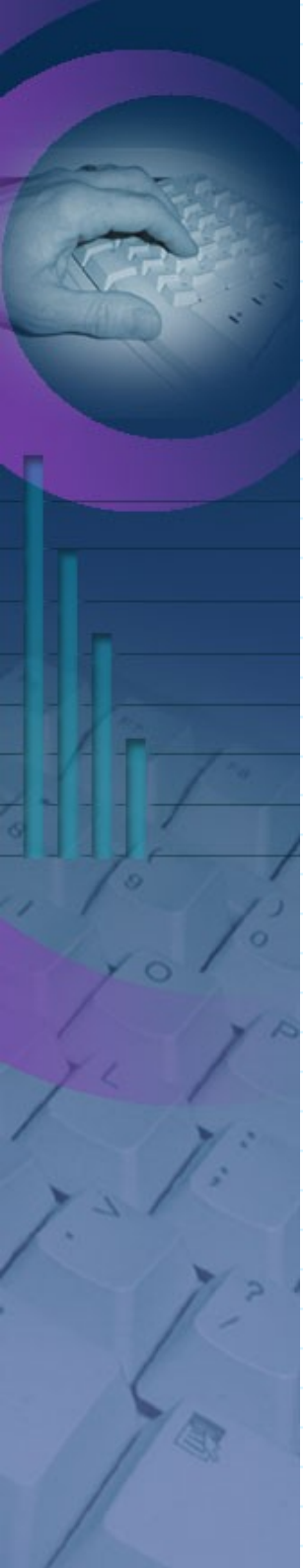
Ejemplo de Uso

- El backtraking automático es útil pero puede resultar ineficiente.
- Hay situaciones en las cuales quisiéramos controlar el proceso de backtraking, o incluso impedirlo.



Ejemplo de Uso

- Supongamos el siguiente ejemplo:
 - R1: si $X < 3$ entonces $Y = 0$
 - R2: si $3 \leq X$ y $X < 6$ entonces $Y = 2$
 - R3: si $6 \leq X$ entonces $Y = 4$
- En Prolog:
 - $f(X, 0) :- X < 3.$
 - $f(X, 2) :- X \geq 3, X < 6.$
 - $f(X, 4) :- X \geq 6.$



Ejemplo de Uso

- Supongamos la siguiente pregunta:
 - $f(1, Y), 2 < Y$.
- La pregunta falla, pero antes debe recorrer todas las posibles soluciones con las que se encuentra en principio, aún sabiendo que las soluciones son incompatibles
- Usando el predicado de corte podemos asegurarnos de impedir el backtracking automático en cualquier momento.



El Predicado Corte “!”

- El corte es un predicado predefinido cuya sintaxis es el signo “!”.
- Siempre se evalúa verdadero, es decir que su evaluación nunca falla.
- Supongamos:
 - $a :- b, !, c.$
 - $a :- d.$
- si “b” no es cierto se pasará a evaluar la segunda instancia del predicado, si “d” es cierto, entonces se demuestra que “a” es cierto.
- Si “b” es cierto se evalúa el corte, lo cual produce:
 - No se podrá hacer backtraking para tomar otras posibles alternativas en el predicado “b”.
 - No se podrá considerar otras posibles definiciones del predicado “a”.



El Predicado Corte “!”

- Usemos el corte para el ejemplo anterior:
 - $f(X, 0):- X < 3, !.$
 - $f(X, 2):- 3 \leq X, X < 6, !.$
 - $f(X, 4):- 6 \leq X.$
- Otra forma de usarlo sería:
 - $f(X, 0):- X < 3, !.$
 - $f(X, 2):- X < 6, !.$
 - $f(X, 4).$
- Que diferencia se observa entre la primera forma y la segunda?
 - El corte en el primer caso no cambia el sentido declarativo en el programa y por ende se conoce como “corte verde”, en el segundo caso si se da éste cambio y se lo conoce como “corte rojo”



Ejercicio 1 (25 min.)

Dado las siguientes declaraciones:

- $f(X, 0):- X < 3.$
- $f(X, 2):- 3 \leq X, X < 6.$
- $f(X, 4):- 6 \leq X.$

- 1) Construir el árbol de deducción correspondiente a la pregunta:
 - $f(1, Y), 2 < Y$
- 2) Definir una relación $f1(X, Y)$ a partir de la definición $f(X, Y)$ introduciendo un corte al final de las dos primeras cláusulas.
- 3) Optimice las cláusulas de $f1$ para realizar la menor cantidad de comparaciones posibles (debe quedar una por definición del predicado).
- 4) Construir el árbol de deducción de la última solución encontrada (en el punto 3) con la consulta del ejercicio 1.



Uso del Corte

- Tres usos más comunes:
 - No buscar soluciones en predicados alternativos.
 - Combinación de corte y fallo.
 - Solo es necesaria la primera opción.



Combinación de Corte y Fallo

- Se utiliza esta combinación para indicar que ante una determinada situación, no se debe proseguir con la búsqueda de alternativas.
- La definición del predicado predefinido “not” es como sigue:
 - `not(P) :- call(P), !, fail.`
 - `not(P).`



Ejemplo de Corte y Fallo

- Supongamos un predicado para calcular la dosis diaria de un medicamento, en base a la edad y peso de un paciente. El medicamento está contraindicado para personas mayores de 80 años y menores de 12:
 - `dosis(_, Edad, _)` :- Edad \leq 12, !, fail.
 - `dosis(_, Edad, _)` :- Edad \geq 80, !, fail.
 - `dosis(Peso, Edad, Dosis)` :-
 - Edad $<$ 25, !,
Dosis is $\text{Peso} / 20$.
 - `dosis(Peso, Edad, Dosis)` :-
 - Edad $<$ 50, !,
Dosis is $\text{Peso} / 15$.
 - `dosis(Peso, Edad, Dosis)` :-
 - Dosis is $\text{Peso} / 12$.



No Buscar más Opciones

- La situación sería: “Si has llegado hasta aquí has encontrado la solución y no es necesario seguir evaluando”.
- La situación planteada se da cuando queremos finalizar una secuencia de generación y prueba.



Ejemplo

- Supongamos un predicado para calcular la longitud de una lista:
 - `contar([], 1).`
 - `contar([_ | C], N) :- contar(C, N1), N is N1 + 1.`
- Si consultáramos “`contar([1,2,3], X).`” la respuesta sería:
 - `X = 3;`
 - `false.` ¿por que? ¿como lo evitamos?
- Podríamos usar el corte:
 - `contar([], 1) :- !.`
 - `contar([_ | C], N) :- contar(C, N1), N is N1 + 1.`

Ejercicios 2 (40 min.)

Cree un programa que encuentre la salida dado el plano de una casa:

conectado/2: establece como hecho la existencia de una puerta entre dos habitaciones o entre una habitación y la salida

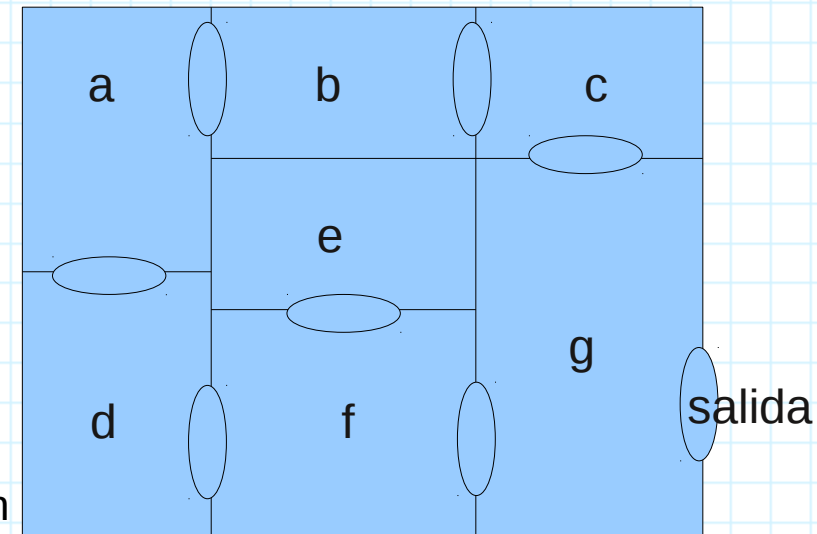
pasa/2: establece que se puede pasar desde una habitación X a otra Y

pertenece/2: determina si un elemento pertenece a una lista

concatenar/3: concatena las dos primeras listas en una tercera

invertir/2: invierte una lista

salida/2: el primer argumento es la habitación donde se está, y el segundo unifica con el camino a seguir hasta la salida





Problemas con el Corte

- Los cortes deben ser usados con cuidado ya que pueden afectar el significado declarativo de un programa.
- Los cortes que afectan el significado declarativo se denominan “Cortes Rojos”.
- Los cortes que no afectan el significado declarativo se denominan “Cortes Verdes”.



Problemas con el Corte

- Supongamos el siguiente ejemplo:
 - $p \text{ :- } a, b.$
 - $p \text{ :- } c.$ $p \leftarrow (a \wedge b) \vee c$
- Si agregamos el siguiente corte “ $p \text{ :- } a, !, b.$ ”, el significado declarativo se ve alterado, y la disposición de los predicados influye.
 - $p \text{ :- } a, !, b.$
 - $p \text{ :- } c.$ $p \leftarrow (a \wedge b) \vee (\neg a \wedge c)$
- Si cambiamos el orden obtenemos
 - $p \text{ :- } c.$
 - $p \text{ :- } a, !, b.$ $p \leftarrow c \vee (a \wedge b)$



Problemas con la Negación

- Supongamos las siguientes definiciones:
 - bueno(elCastillo).
 - bueno(asadolandia).
 - lejos(elCastillo).
 - cerca(Restaurante) :- not(lejos(Restaurante)).
- Si quisiéramos evaluar las siguientes conjunciones “bueno(X), cerca(X).” el resultado sería:
 - X = asadolandia.
- Pero si evaluáramos “cerca(X), bueno(X).” el resultado sería:
 - false.



Problemas con al Negación

- La negación utilizada en PROLOG no corresponde a la negación de la lógica clásica.
- Cuando Prolog contesta afirmativamente una negación, no quiere decir necesariamente que la expresión negada sea falsa, sino que no se ha podido comprobar que sea verdadera.
- Esto se da debido a que Prolog no intenta probar la negación directamente, sino que prueba el predicado a negar, el cual falla por falta de pruebas. Luego asume que la negación se satisface.
- Este razonamiento es válido ya que asumimos que el dominio del problema es un “mundo cerrado”. En éstos dominios, solo es cierto todo aquello que está declarado como hecho o se puede inferir a partir de estos, y todo lo que no está declarado y no puede ser inferido, se entiende como falso.



Ejercicios 2 (25 min.)

La relación “es_miembro/2” se define de la siguiente manera:

- $\text{es_miembro}(X, [X \mid _]) :- !.$
- $\text{es_miembro}(X, [_ \mid L]) :- \text{es_miembro}(X, L).$

1) Construir los árboles de resolución correspondientes a las siguientes preguntas:

1) $\text{es_miembro}(X, [a, b, c]), X = a.$

2) $\text{es_miembro}(X, [a, b, c]), X = b.$

3) $X = b, \text{es_miembro}(X, [a, b, c]).$