

**CORTES Y NEGACIONES**

# CORTES

Backtracking es un rasgo característico de Prolog.

Pero el backtracking puede conducir a la ineficiencia:

- Prolog puede perder tiempo y memoria explorando posibilidades que no llevan a ninguna parte.
- Sería bueno tener algo de control.

El predicado de corte (!) ofrece una forma de controlar el backtracking.

El corte es un predicado, por lo que podemos agregarlo al cuerpo de reglas:

- Ejemplo: `p(X):- b(X), c(X), !, d(X), e(X).`
- Cortar es un objetivo que siempre tiene éxito
- El corte compromete a Prolog con las elecciones que se hicieron desde que se llamó al objetivo principal

# EXPLICACIONES

$p(X) \text{ :- } a(X).$

$p(X) \text{ :- } b(X), c(X), d(X), e(X).$

$p(X) \text{ :- } f(X).$

$a(1).$

$b(1), b(2).$

$c(1), c(2).$

$d(2).$

$e(2).$

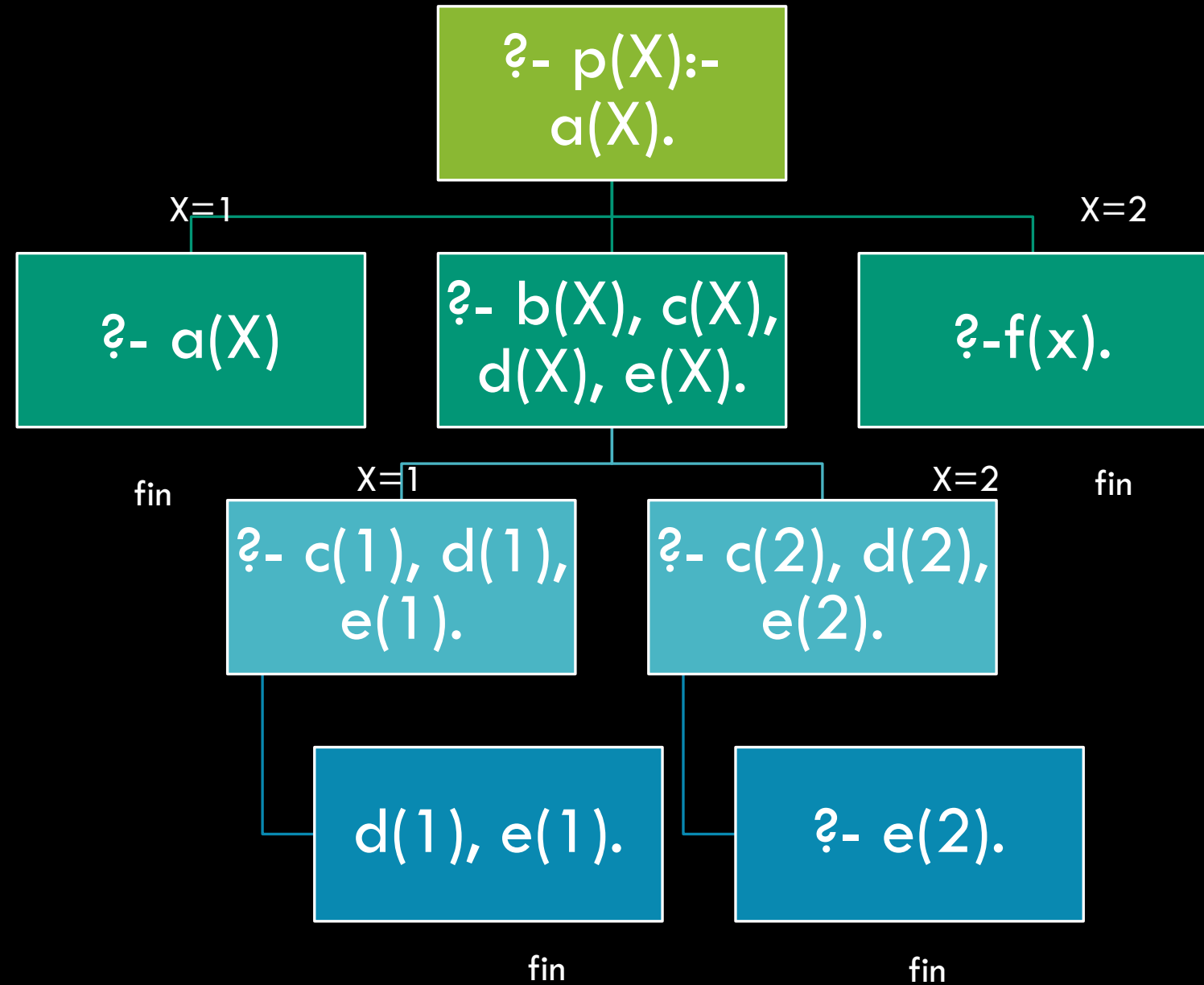
$f(3).$

$? \text{ :- } p(X).$

$X=1$

$X=2$

$=3$



# SUMANDO !

Supongamos que insertamos un corte en la segunda cláusula:

- $p(X) \text{:- } b(X), c(X), \text{!, } d(X), e(X).$

Si ahora planteamos la misma consulta obtendremos la siguiente respuesta:

- $? \text{- } p(X).$
- $X=1;$
- False

# EXPLICACIONES

$p(X) \text{ :- } a(X).$

$p(X) \text{ :- } b(X), c(X), d(X), e(X).$

$p(X) \text{ :- } f(X).$

$a(1).$

$b(1). b(2).$

$c(1). c(2).$

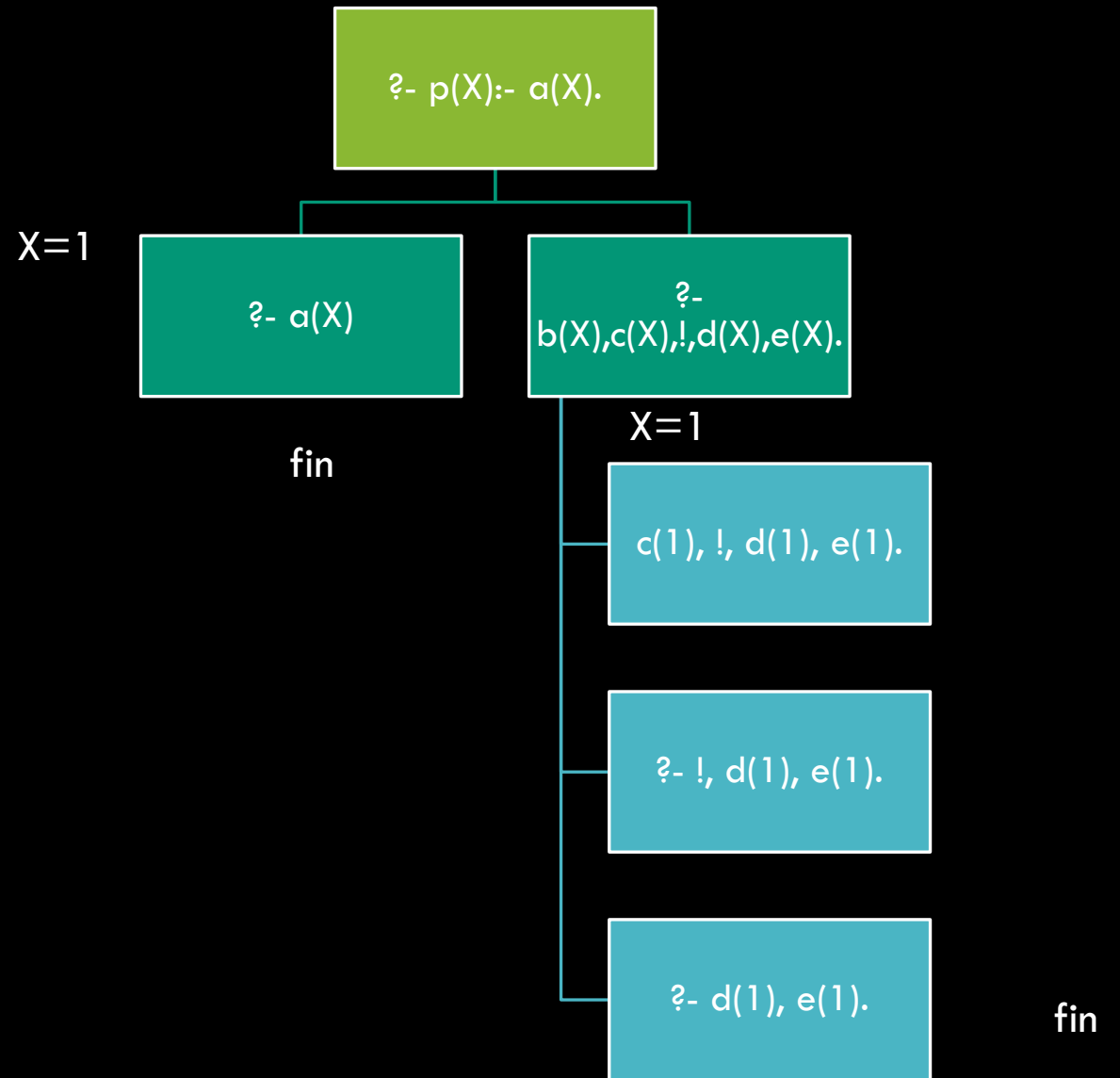
$d(2).$

$e(2).$

$f(3).$

$?- p(X).$

$X=1$



# ¿QUÉ ES LO QUE HACE ! ?

El ! solo nos compromete a tomar decisiones ya que el objetivo principal se unificó con el.

Por ejemplo, en una regla de la forma

- $q:- p_1, \dots, p_m, !, r_1, \dots, r_n.$

Cuando llegamos al corte nos comprometemos:

- a esta cláusula particular de  $q$
- a las elecciones hechas por  $p_1, \dots, p_m$
- NO a las elecciones hechas por  $r_1, \dots, r_n$

# USÁNDOLO

Considere el siguiente predicado  $\text{max}/3$  que tiene éxito si el tercer argumento es el máximo de los dos primeros.

$\text{max}(X,Y,Y):- X \leq Y.$

$\text{max}(X,Y,X):- X > Y.$

?-  $\text{max}(2,3,3).$

true

?-  $\text{max}(7,3,7).$

true

# USÁNDOLO

$\text{max}(X, Y, Y) :- X \leq Y, !.$

$\text{max}(X, Y, X) :- X > Y.$

Si  $X \leq Y$  tiene éxito, el corte nos compromete a esta elección, y la segunda cláusula de  $\text{max}/3$  no se considera.

Si  $X \leq Y$  falla, Prolog pasa a la segunda cláusula.



# CORTES PERMITIDOS O VERDES

Los cortes que no cambian el significado de un predicado se denominan cortes verdes.

El corte en  $\max/3$  es un ejemplo de un corte verde:

- el nuevo código da exactamente las mismas respuestas que la versión anterior,
- pero es más eficiente

# CORTES PERMITIDOS O VERDES

Los cortes que no cambian el significado de un predicado se denominan cortes verdes.

El corte en `max/3` es un ejemplo de un corte verde:

- el nuevo código da exactamente las mismas respuestas que la versión anterior,
- pero es más eficiente.

Y ahora:

- `max(X,Y,Z):- X =< Y, !,.`
- `max(X,Y,X).`
- `?- max(200,300,X).`
  - `X=300`
  - `?- max(400,300,X).`
  - `X=400`
  - `?- max(200,300,200).`
  - `True`

# CORTES NO PERMITIDOS O ROJOS

Los cortes que cambian el significado de un predicado se denominan cortes rojos.

Por ejemplo:

```
max(X,Y,Z):- X =< Y, !, Y=Z. % corte rojo:
```

```
max(X,Y,X).
```

- `?- max(200,300,200).`
- `false`

Si sacamos el corte, no obtenemos un programa equivalente.

Programas que contienen cortes rojos

- No son totalmente declarativos
- Puede ser difícil de leer
- Puede conducir a errores de programación sutiles

# FAIL

Como su nombre indica, este es un objetivo que fallará inmediatamente cuando Prolog intenta probarlo.

Eso puede no sonar demasiado útil.

Pero recuerde: cuando Prolog falla, **intenta retroceder**.

# PROBEMOS

legusta(paola,X):- macNifica(X), !, fail.

legusta(paola,X):- burger(X).

burger(X):- macNifica(X).

burger(X):- bigMac(X).

burger(X):- macFiesta(X).

macNifica(b).

bigMac(a).

bigMac(c).

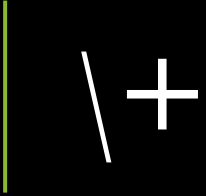
macFiesta(d).

?- legusta(paola,c).

True

?- legusta(paola,d).

True



`\+`

Debido a que la negación como fracaso se usa tan a menudo, no hay necesidad de definirla.

En Prolog estándar, el operador de prefijo `\+` significa negación como fracaso

Así podríamos definir la preferencias de la siguiente manera.

# PROBEMOS

legusta(paola,X):- burger(X), \+  
macNifica(X).

burger(X):- macNifica(X).

burger(X):- bigMac(X).

burger(X):- macFiesta(X).

macNifica(b).

bigMac(a).

bigMac(c).

macFiesta(d).

?- legusta(paola,X).

X=a;

X=c;

X=d;

# EJERCITEMOS

Supongamos que tenemos la siguiente base de datos:

p(1).  
p(2) :- !.  
p(3).

Escriba todas las respuestas de Prolog a las siguientes consultas:

?- p(X).  
?- p(X),p(Y).  
?- p(X),!,p(Y).

Primero, explique lo que hace el siguiente programa:

- class(Numero,positivo) :- Numero > 0.
- class(0,cero).
- class(Numero,negativo) :- Numero < 0.

En segundo lugar, mejóralo añadiendo cortes verdes.



# EJERCITEMOS

Sin usar `!`, escribe un predicado `split/3` que divida una lista de enteros en dos listas: una que contenga los positivos (y cero), la otra que contenga los negativos.

Por ejemplo: `split([3,4,-5,-1,0,4,-9],P,N)`

Debera devolver

$P = [3,4,0,4]$   
 $N = [-5,-1,-9].$

Luego mejore este programa, sin cambiar su significado, con la ayuda del `!`.

# EJERCITEMOS

Recordemos que en recursividad te dimos la siguiente base de conocimientos:

```
directTrain(saarbruecken,dudweiler).
directTrain(forbach,saarbruecken).
directTrain(freyming,forbach).
directTrain(stAvold,freyming).
directTrain(fahlquemont,stAvold).
directTrain(metz,fahlquemont).
directTrain(nancy,metz).
```

Le pedimos que escribiera un predicado recursivo `travelFromTo` que nos dijera cuándo podíamos viajar en tren entre dos ciudades.

Ahora, es plausible suponer que siempre que sea posible tomar un tren directo de A a B, también es posible tomar un tren directo de B a A.

Agregue esta información a la base de datos. Luego escribe una ruta predicada/3 que te dé una lista de las ciudades que se visitan tomando el tren de una ciudad a otra. Por ejemplo:

```
?- route(forbach,metz,Route).
Route = [forbach,freyming,stAvold,fahlquemont,metz]
```

# EJERCITEMOS

La definición de celos.

$\text{celos}(X,Y):- \text{ama-a}(X,Z), \text{ ama-a}(Y,Z).$

En un mundo donde tanto Vincent como Marsellus aman a Mia, Vincent estará celoso de Marsellus y Marsellus de Vincent.

Pero Marsellus también estará celoso de sí mismo, al igual que Vincent. Revise la definición de celos de Prolog de tal manera que las personas no puedan estar celosas de sí mismas.