# El Operador de Corte y la Negación por Falla

***Lógica para Ciencias de la Computación***

*Primer Cuatrimestre de 2008*

– Material Adicional –

# Motivaciones

- Prolog es un lenguaje de programación altamente declarativo.
- Para alcanzar esta declaratividad, en ocasiones se paga un alto costo computacional.
- Los diseñadores del lenguaje incorporaron un operador de corte que permite podar los espacios de búsqueda.

# Definición formal

- El operador de corte (cut) es un predicado predefinido, notado !.
- Operacionalmete se define como sigue:

*La meta '!' siempre tiene éxito y provoca el descarte de todas las (ramas) alternativas que quedaron pendientes de ser exploradas desde el instante en que se utilizó para resolver la regla conteniendo dicho '!'.*

# Implicancias de la Definición

1. El cut poda alternativas correspondientes a claúsulas por debajo de él.

```
p :- q, !, r.
p :- s, t.
```

2. El cut poda las soluciones alternativas de la meta conjuntiva a la izquierda del mismo.

```
p :- q, r, !, s, t.
```

3. El cut permite soluciones alternativas en la meta conjuntiva a la derecha del mismo.

```
p :- q, r, !, s, t.
```

# Ejemplo

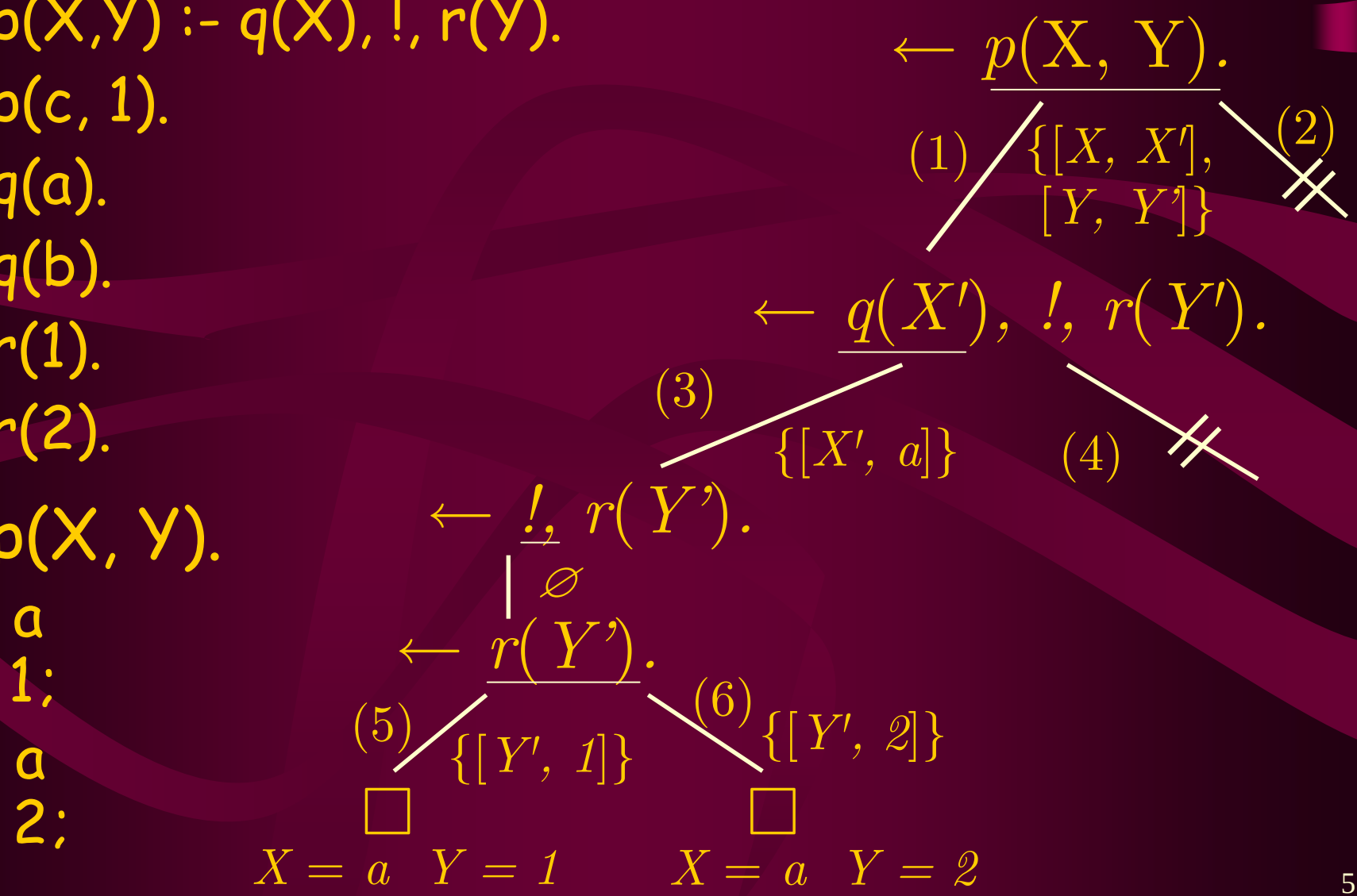
1. p(X,Y) :- q(X), !, r(Y).
2. p(c, 1).
3. q(a).
4. q(b).
5. r(1).
6. r(2).

?- p(X, Y).

X = a
Y = 1;

X = a
Y = 2;

no



# Cuts Rojos y Verdes

- Como se puede apreciar en los ejemplos anteriores, el operador de corte puede podar soluciones de la consulta realizada.

- **Terminología:**
  - Denominaremos cuts rojos a los que poden nuevas soluciones.
  - Denominaremos cuts verdes a los demás, es decir a aquellos que no poden soluciones, o que a lo sumo podan soluciones repetidas.

# Principales Aplicaciones

- Entre sus principales aplicaciones se encuentran:
  - Expresar determinismo.
  - Omitir condiciones.
  - Implementar una forma de negación diferente a la clásica, denominada negación por falla.

condiciones excluyentes

*Negación clásica ≠ Negación por falla*

# Principales Aplicaciones

- Consideremos un predicado definido de la siguiente forma:

```
p(...):- c1(...), ....
p(...):- c2(...), ....
...
p(...):- cn(...), ....
```

condiciones

- Si las condiciones son mutuamente excluyentes, podemos mejorar la eficiencia colocando un cut luego de cada condición:

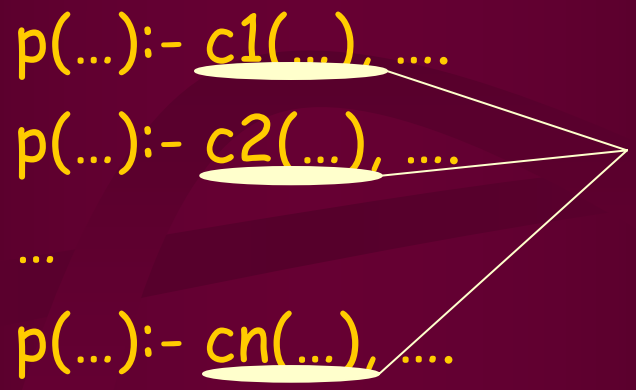
```
p(...):- c1(...), !, ....
p(...):- c2(...), !, ....
...
p(...):- cn(...), !, ....
```

verdes

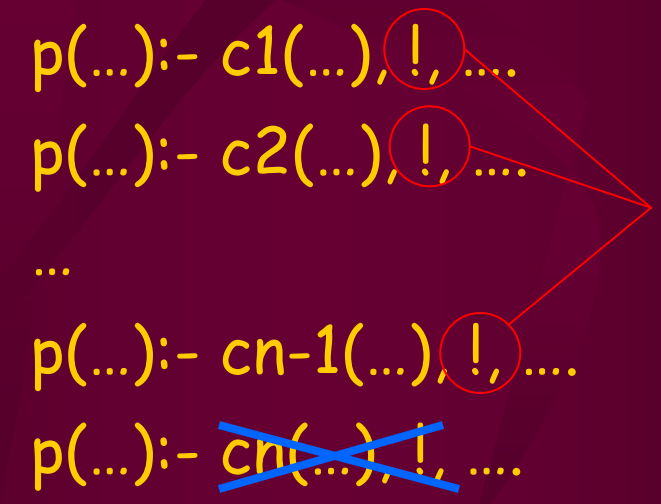
Este último cut no es necesario

# Principales Aplicaciones

- Consideremos un predicado definido de la siguiente forma:




- Si además las condiciones son exhaustivas, podemos omitir la última de ellas al agregar los cuts.



# Ejemplo

- Dados dos números naturales X e Y, hallar la diferencia absoluta entre X e Y:

```
dif_abs(X, Y, Z):- X >=Y, Z is X-Y.
dif_abs(X, Y, Z):- X < Y, Z is Y-X.
```

- Como las condiciones son mutuamente excluyentes, podemos mejorar la eficiencia utilizando cuts:

```
dif_abs(X, Y, Z):- X >=Y, !, Z is X-Y.
dif_abs(X, Y, Z):- X < Y, Z is Y-X.
```

verde

- Como además las condiciones son exhaustivas, directamente podemos omitir la última de ellas.

```
dif_abs(X, Y, Z):- X >=Y, !, Z is X-Y.
dif_abs(X, Y, Z):-          Z is Y-X.
```

rojo

Ahora este cut resulta fundamental para la correctitud del programa.

# Negación por Falla

- La negación por falla se define en términos del operador de corte:

```
naf(X) :- X, !, fail.
naf(X).
```

Observación: **naf/1** es un predicado de segundo orden.

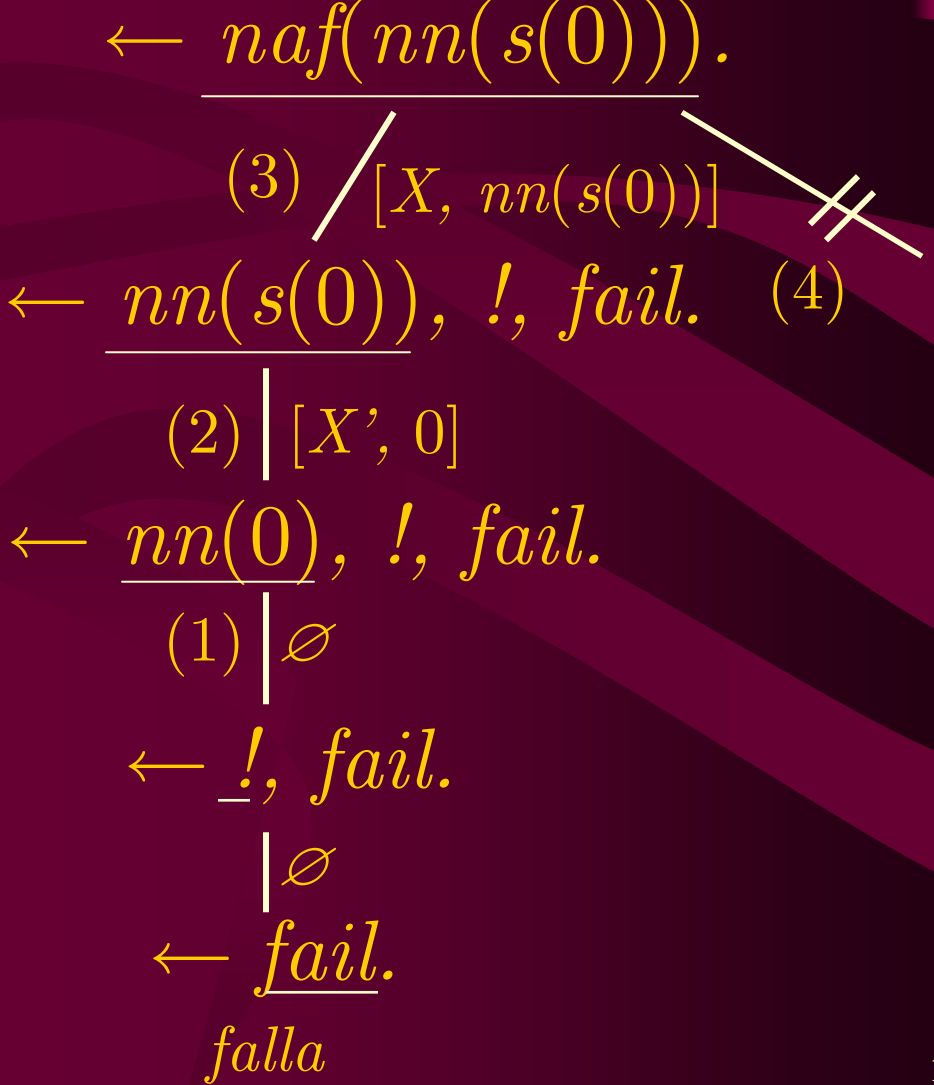
- **Comportamiento de naf/1:** Si la meta X se satisface, se dispara la poda y naf(X) falla. Caso contrario, si la meta X falla, el cut no se ejecuta y naf(X) se satisface.

# Ejemplo

1. nn(0).
2. nn(s(X)) :- nn(X).

3. naf(X) :- X, !, fail.
4. naf(X).

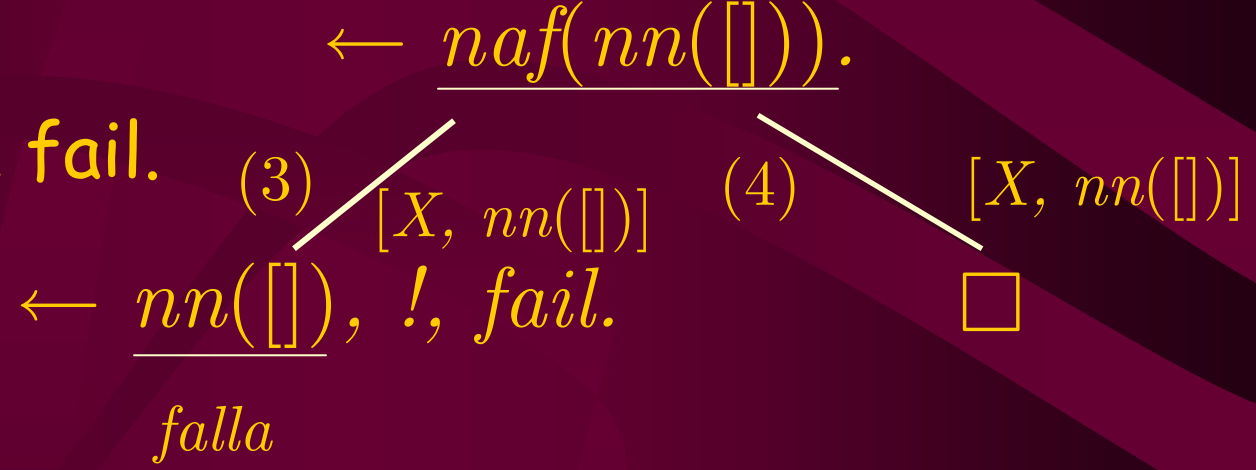
?- naf(nn(s(0))).
no



# Ejemplo

1. nn(0).
2. nn(s(X)) :- nn(X).

3. naf(X) :- X, !, fail.
4. naf(X).




?- naf(nn([])).
yes