

# *Prolog*

## *Predicados Extra-Lógicos y Otros*

Conceptos de Inteligencia Artificial  
Sistemas Inteligentes Artificiales

Departamento de Ciencias e Ingeniería de la Computación  
Universidad Nacional del Sur

# *Operador de Corte: Cut (!)*

- 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 el espacio de búsqueda**.

Permite al programador ejercer **control** sobre el **backtracking**

# Cut (!) - Efectos

1. El cut **poda soluciones alternativas** correspondientes a la cláusula que lo contiene (poda alternativas **por debajo**).

$p:- q, !, r.$

$p:- s, t.$

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

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

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

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

# Cut (!) - Ejemplo

{a}

?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# Cut (!) - Ejemplo

{a}

?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# Cut (!) - Ejemplo

{a}

?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

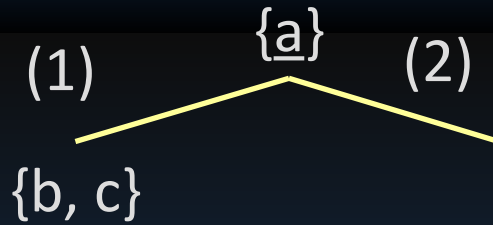
# Cut (!) - Ejemplo

(1)  $\{a\}$   
    └───┐  
      {b, c}

?- a.

(1) a:- b, c.  
(2) a:- d.  
(3) b:- e, !, f.  
(4) b:- g.  
(5) e:- h.  
(6) e:- i.  
(7) e:- g.  
(8) i.  
(9) d.  
(10) g.

# Cut (!) - Ejemplo

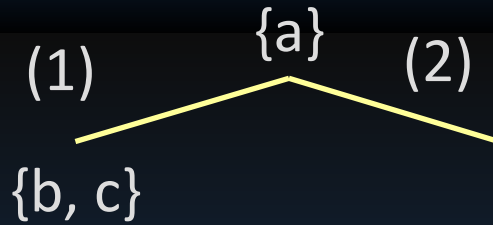


?- a.

- (1) a:- b, c.
- (2) a:- d. ←
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.



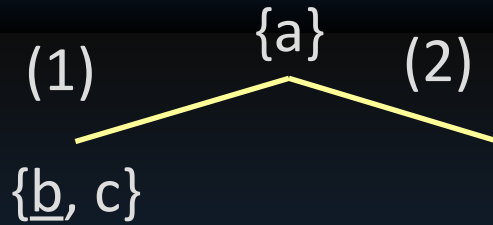
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

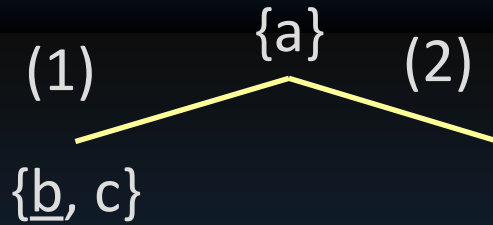
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

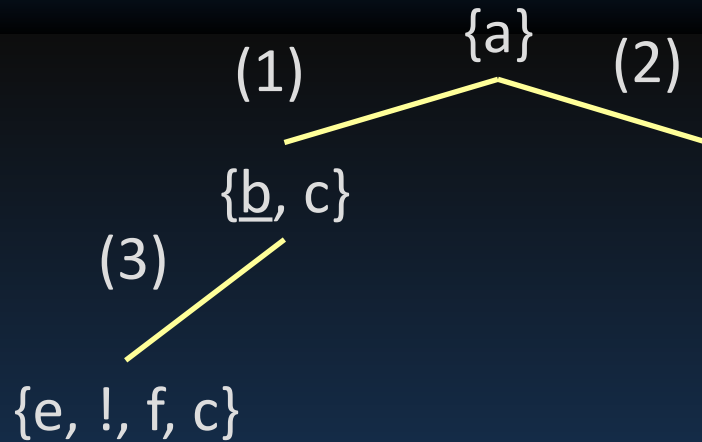
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

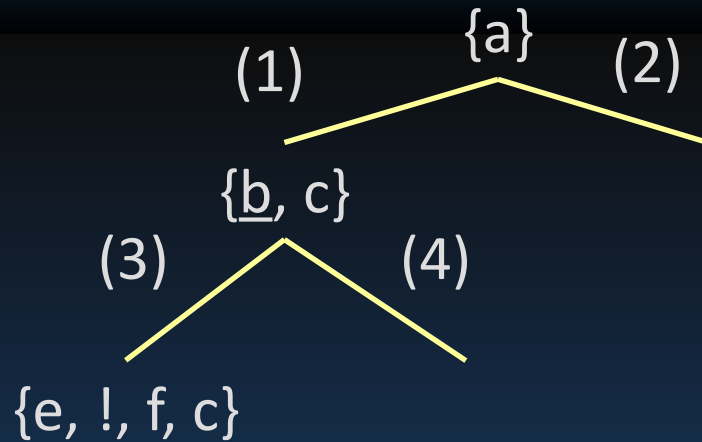
# Cut (!) - Ejemplo




?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

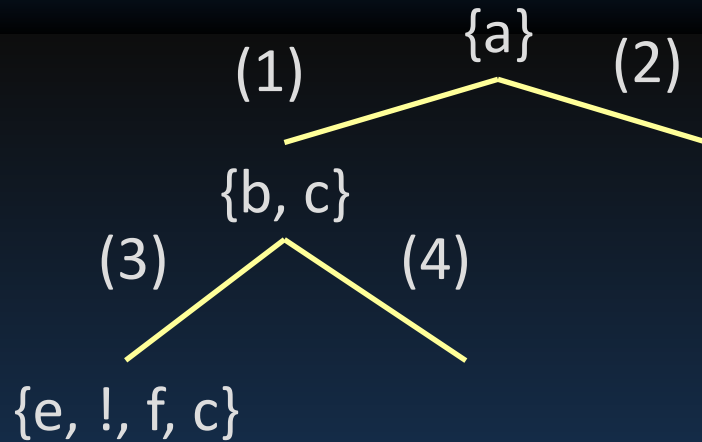
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g. 
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

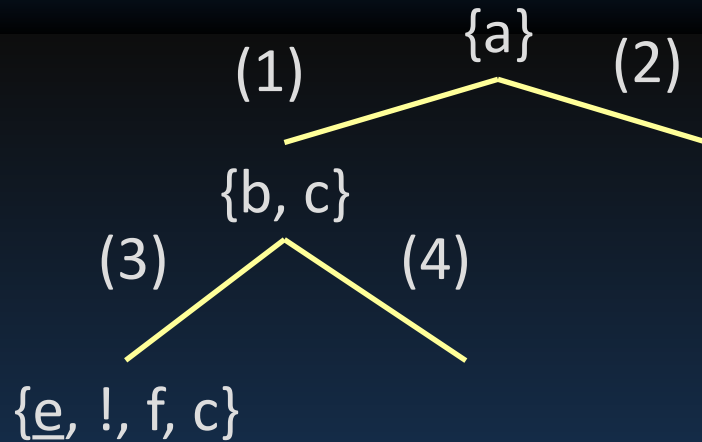
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

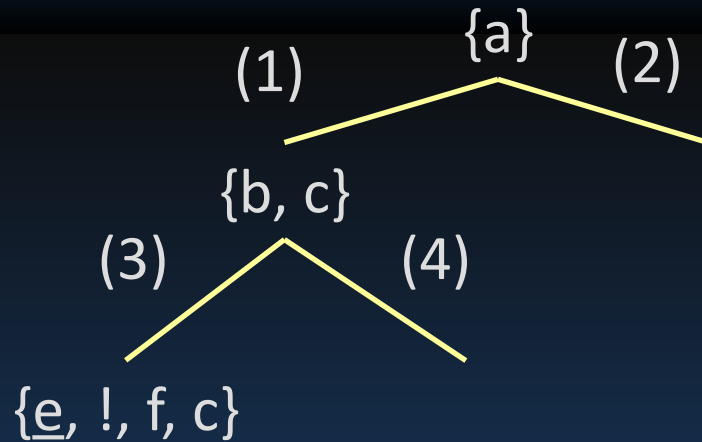
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# Cut (!) - Ejemplo

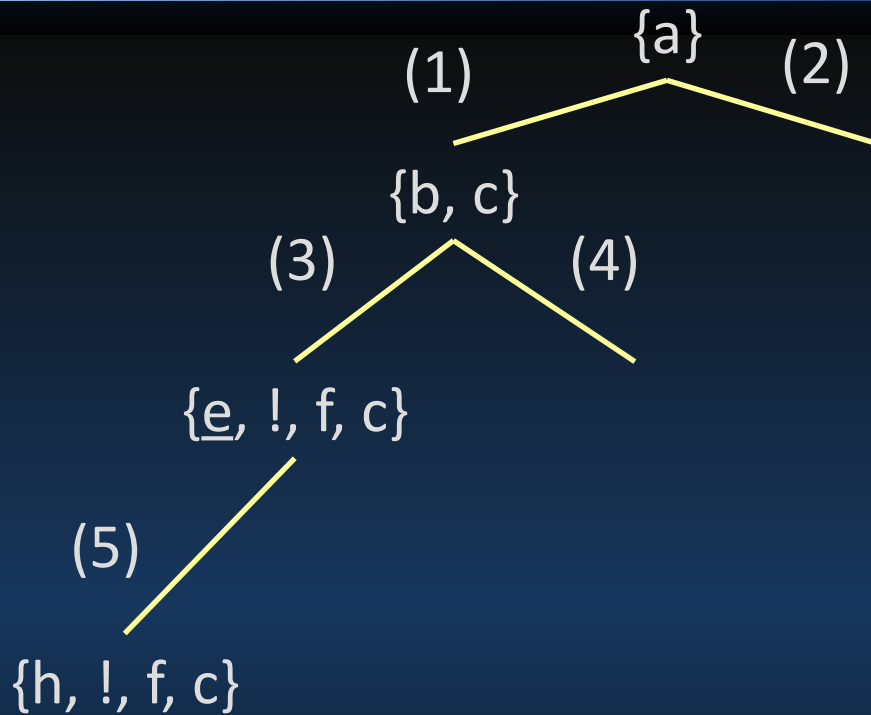


?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.



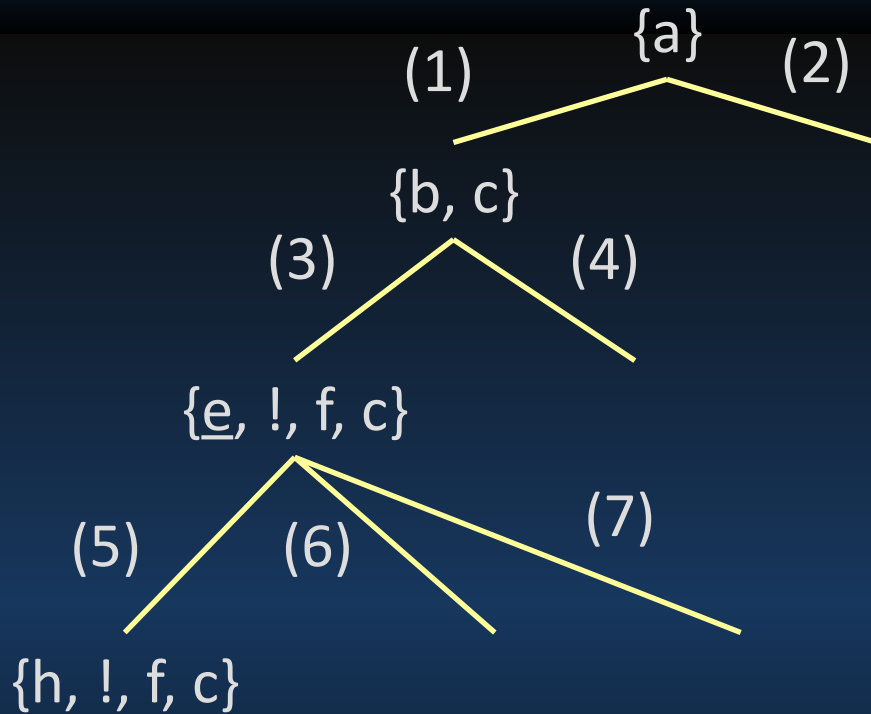
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

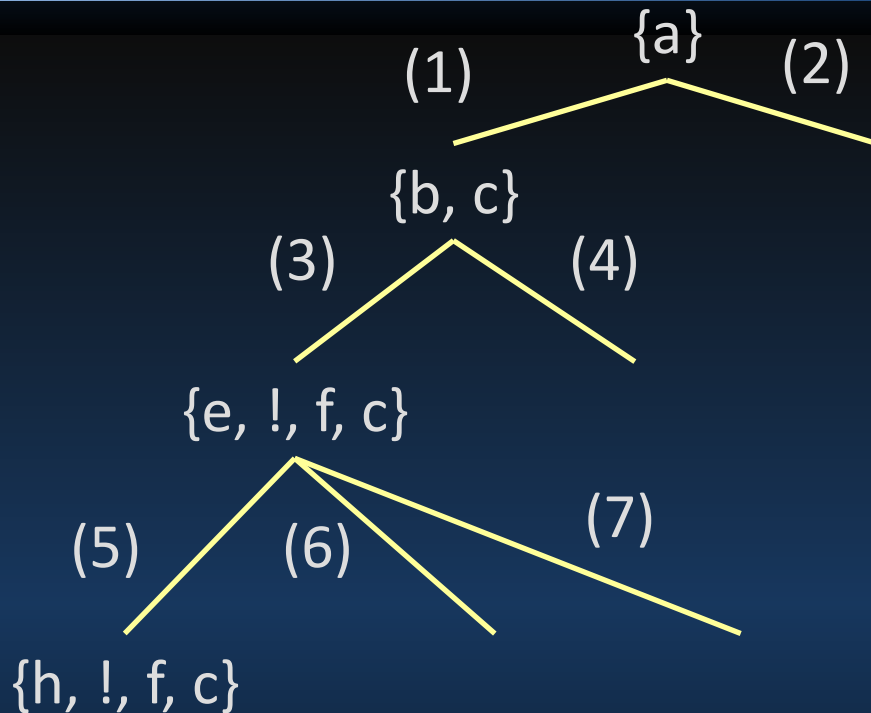
# Cut (!) - Ejemplo



**?- a.**

- (1)  $a:- b, c.$
- (2)  $a:- d.$
- (3)  $b:- e, !, f.$
- (4)  $b:- g.$
- (5)  $e:- h.$
- (6)  $e:- i.$  ←
- (7)  $e:- g.$  ←
- (8)  $i.$
- (9)  $d.$
- (10)  $g.$

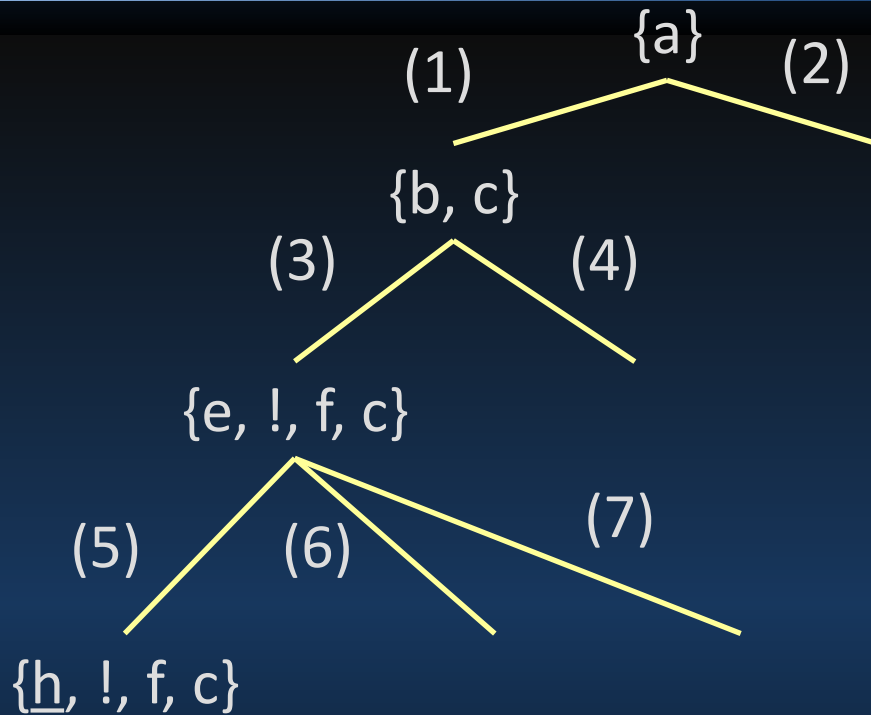
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

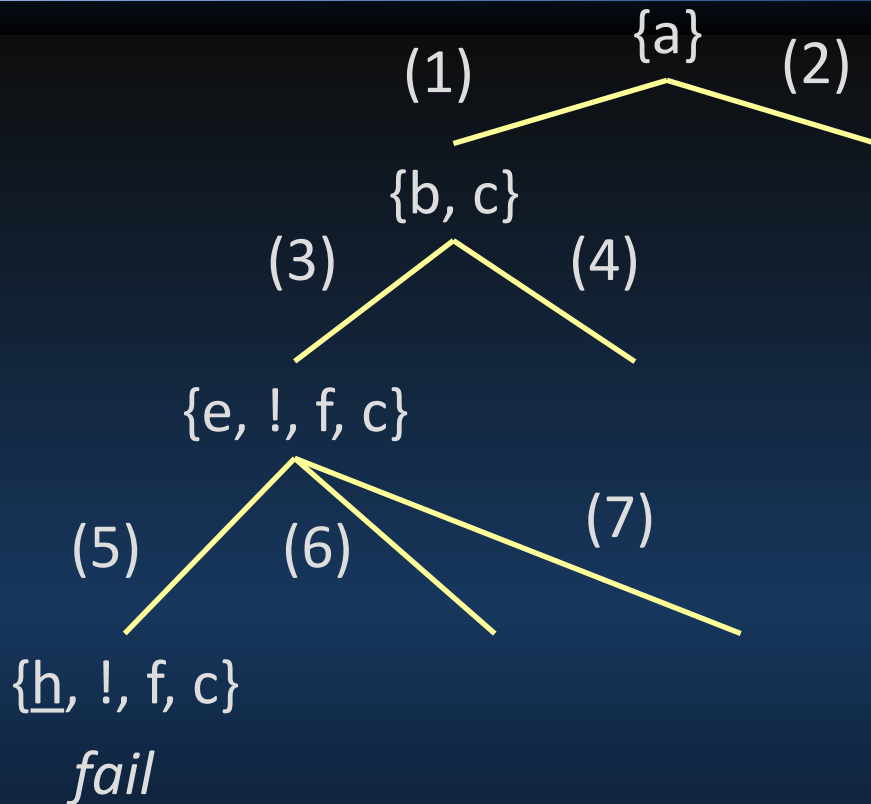
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

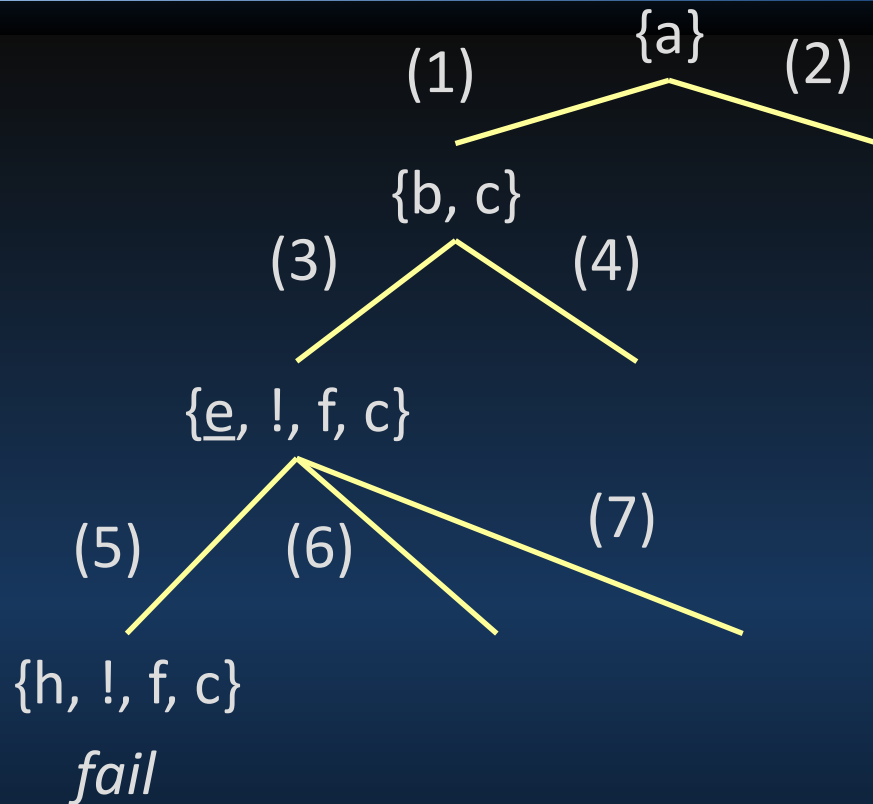
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

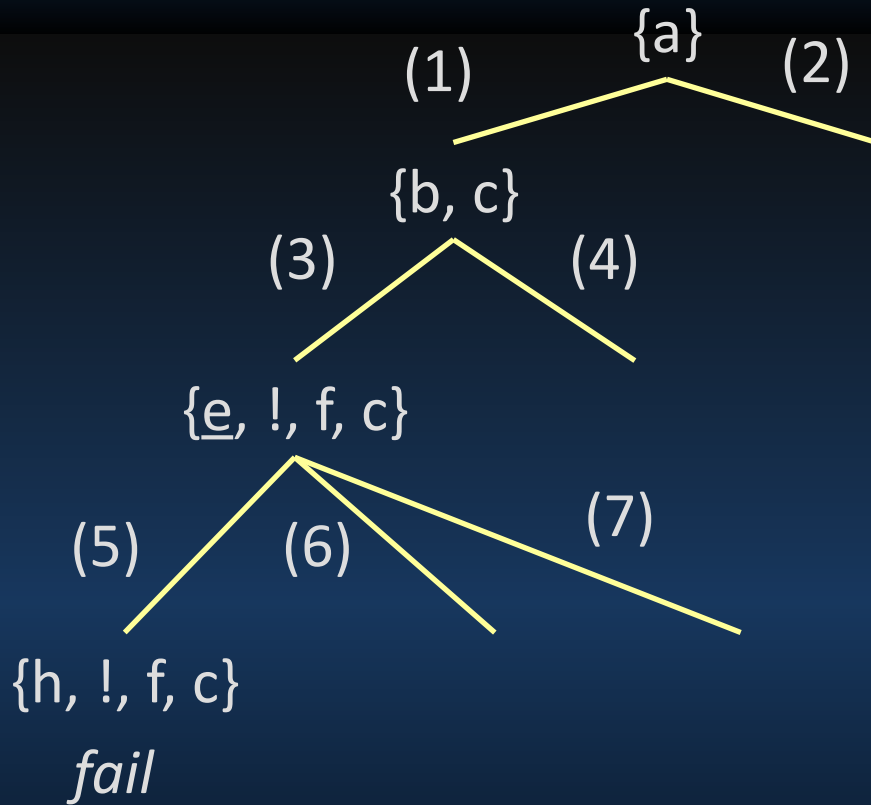
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

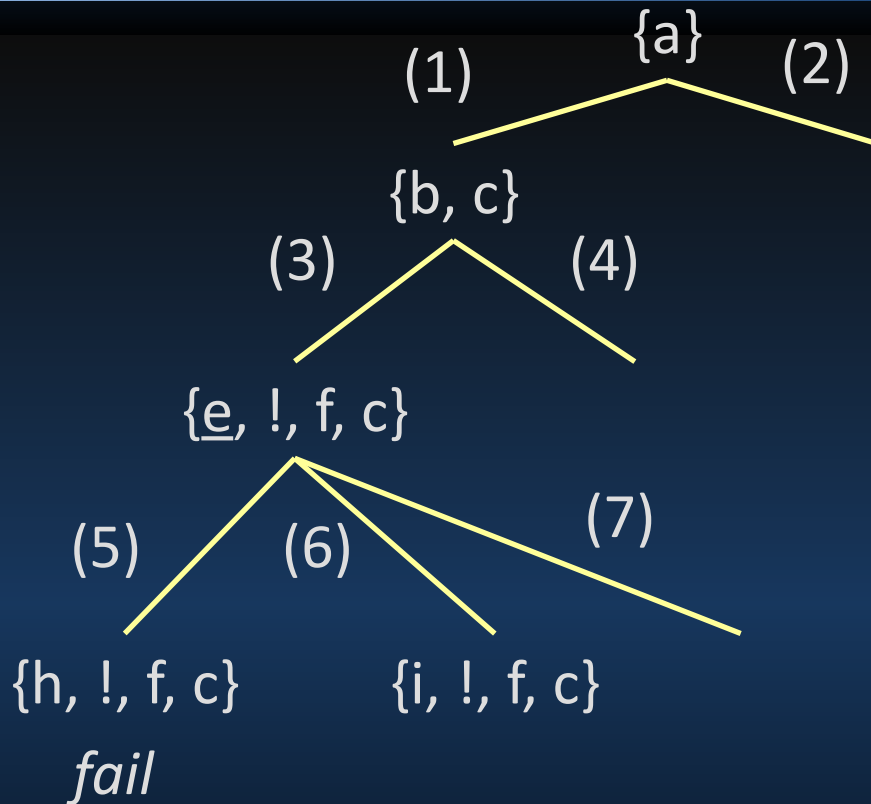
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# Cut (!) - Ejemplo

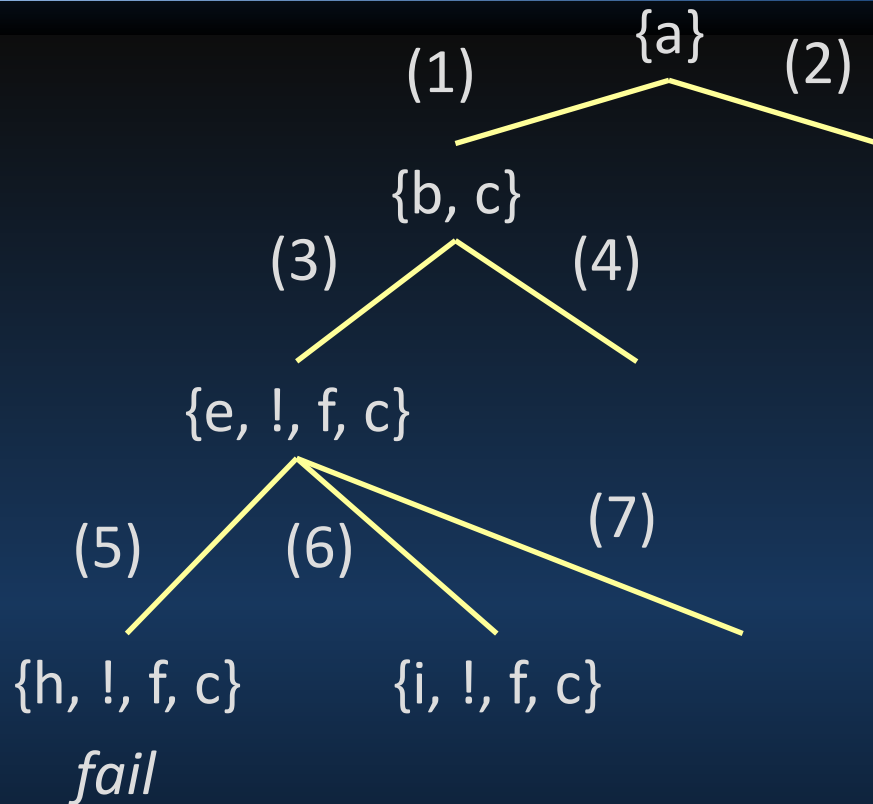


?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.



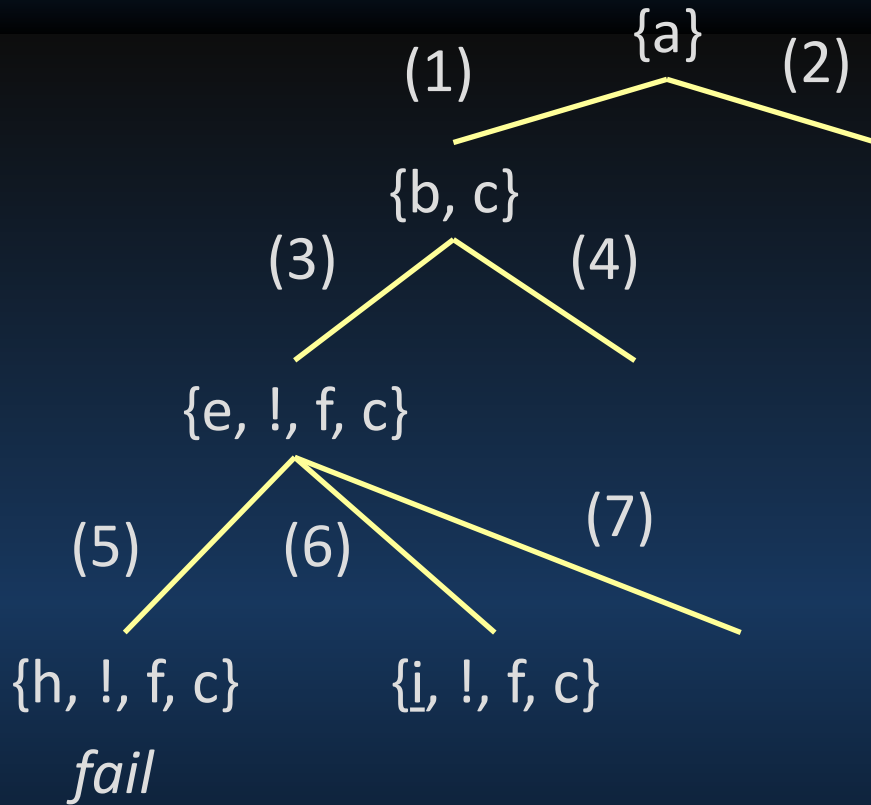
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

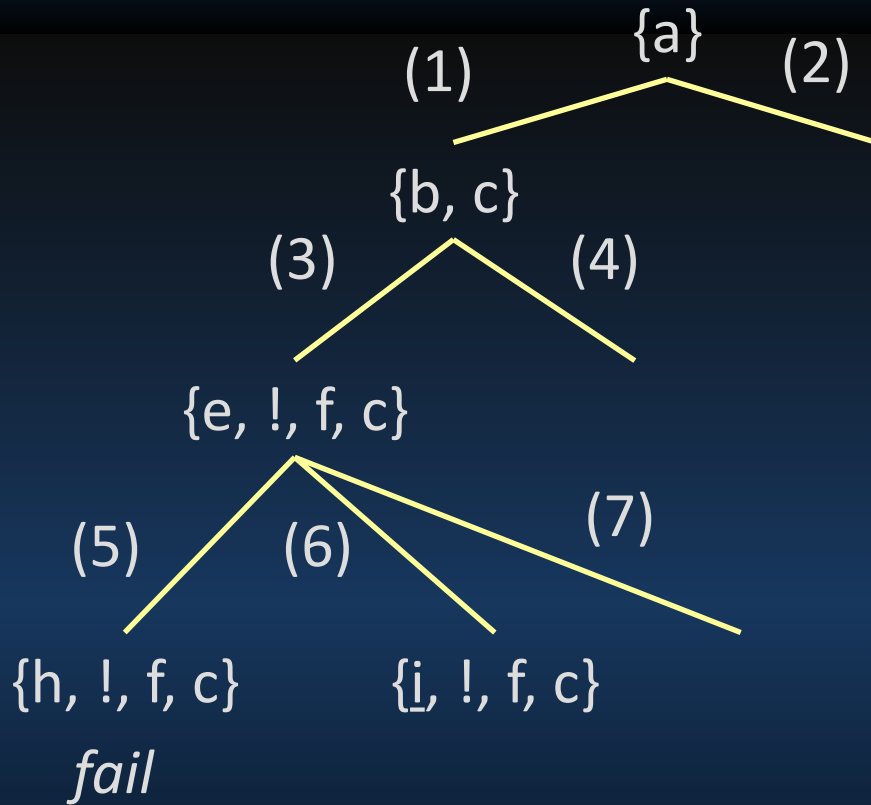
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

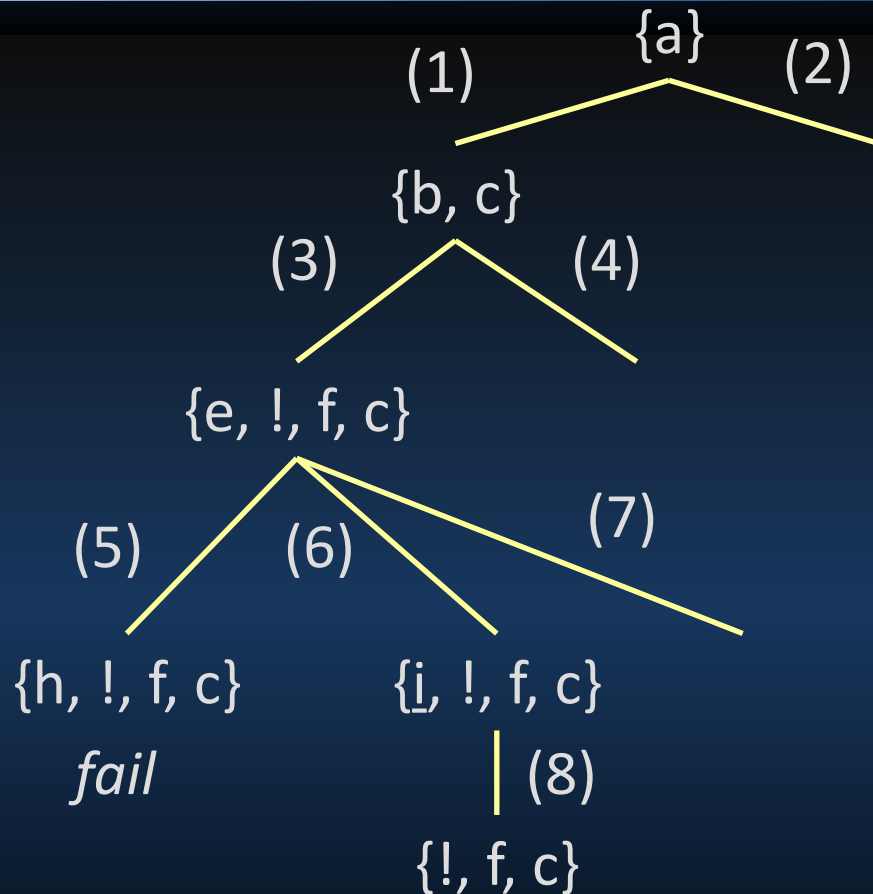
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

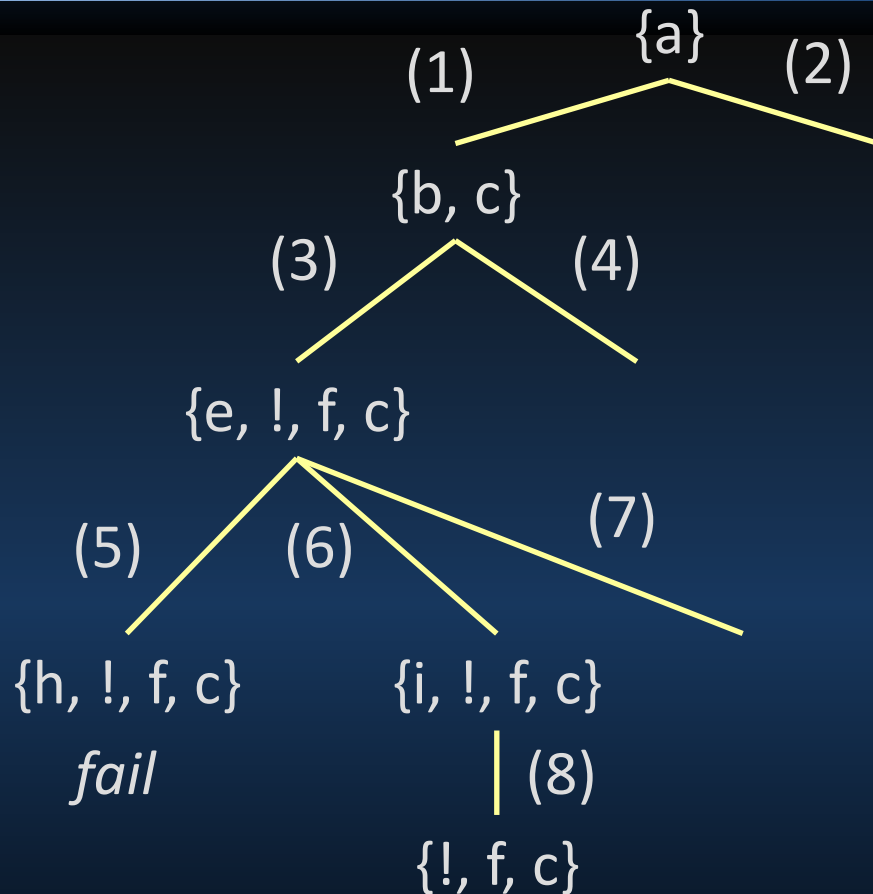
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

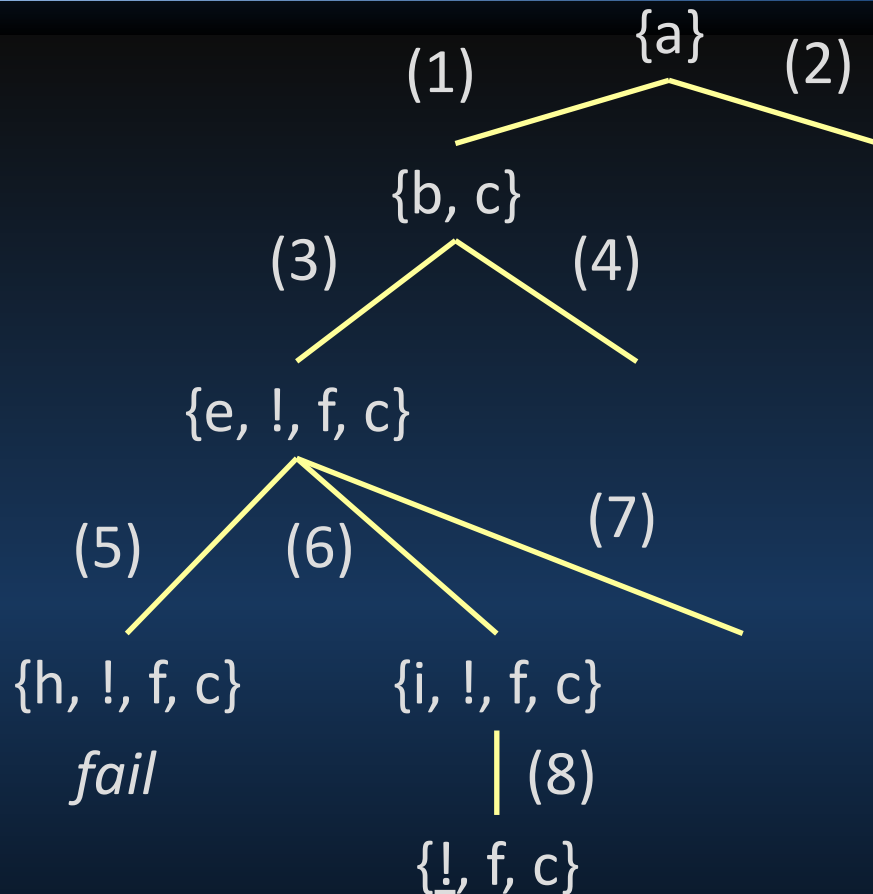
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

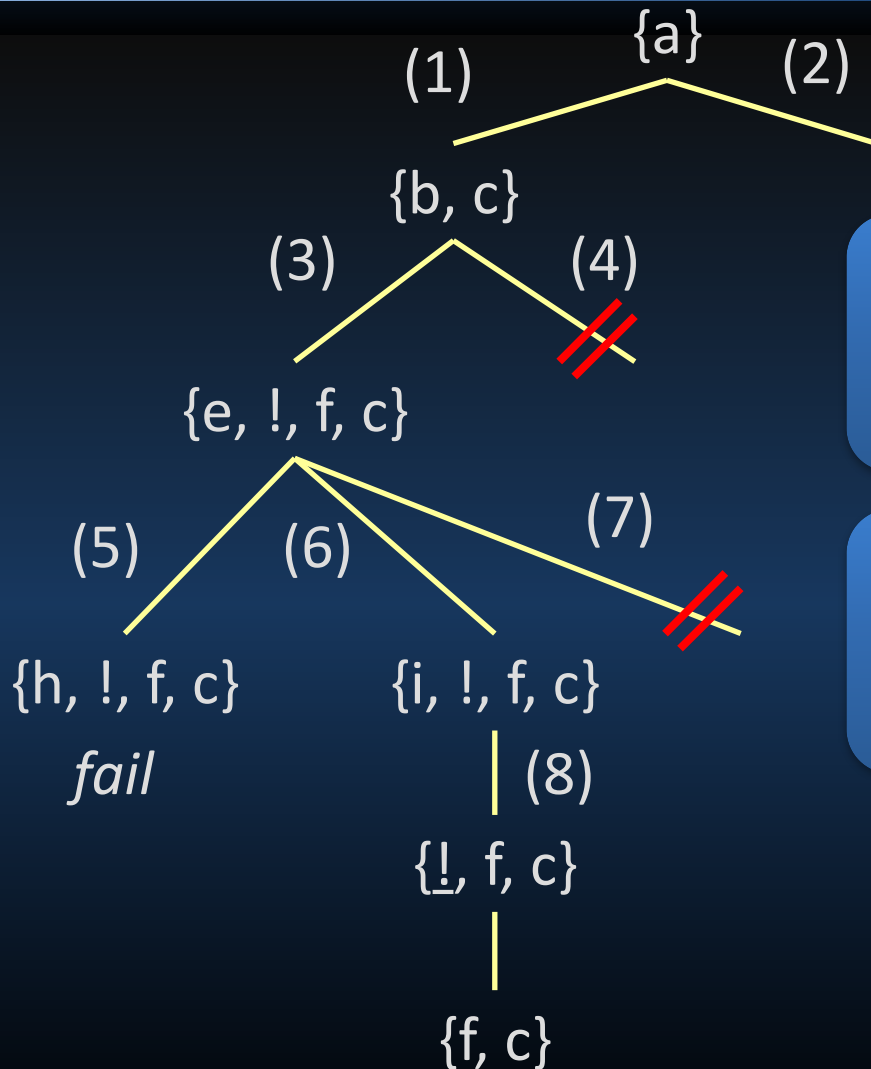
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# Cut (!) - Ejemplo



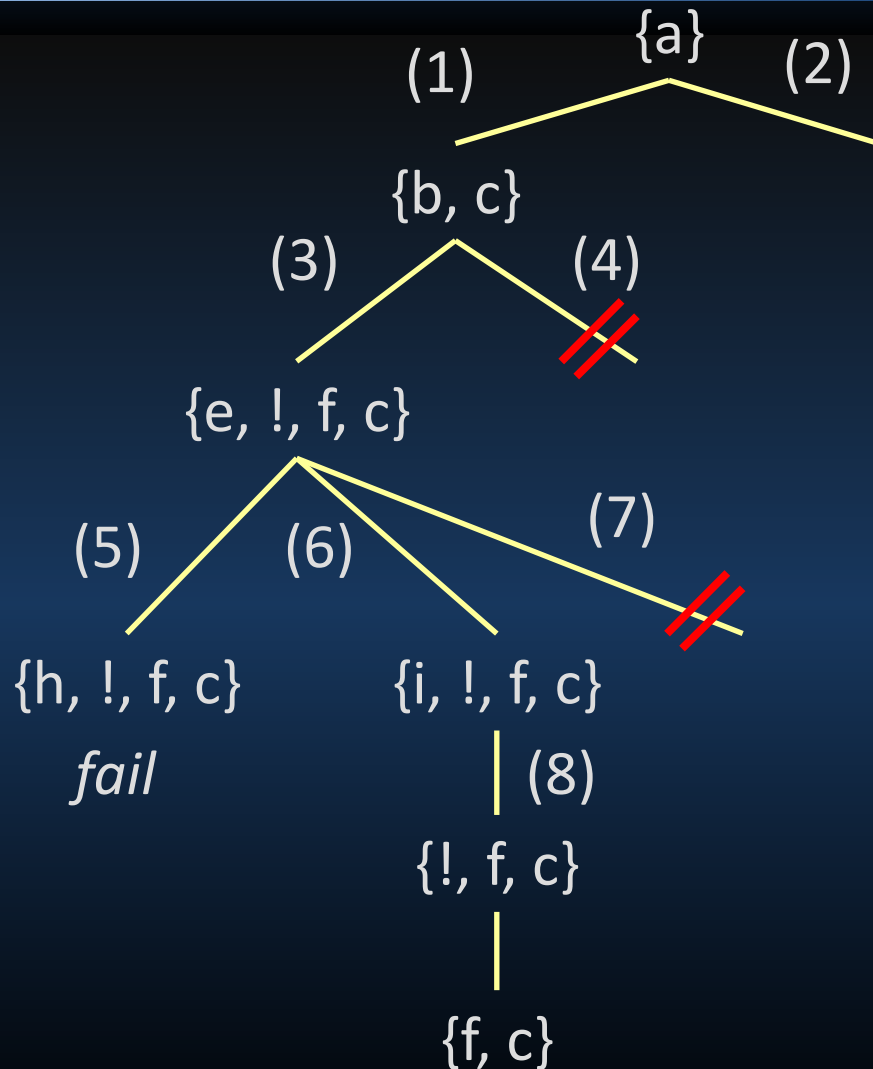
?- a.

Alternativas no exploradas de la cláusula que contiene al cut

Alternativas no exploradas de la meta conjuntiva a la izquierda del cut

(1) a:- b, c, !.  
(2) a:- d.  
(3) b:- e, !, f.  
(4) b:- g.  
(5) e:- h.  
(6) e:- i.  
(7) e:- g.  
(8) i.  
(9) d.  
(10) g.

# Cut (!) - Ejemplo

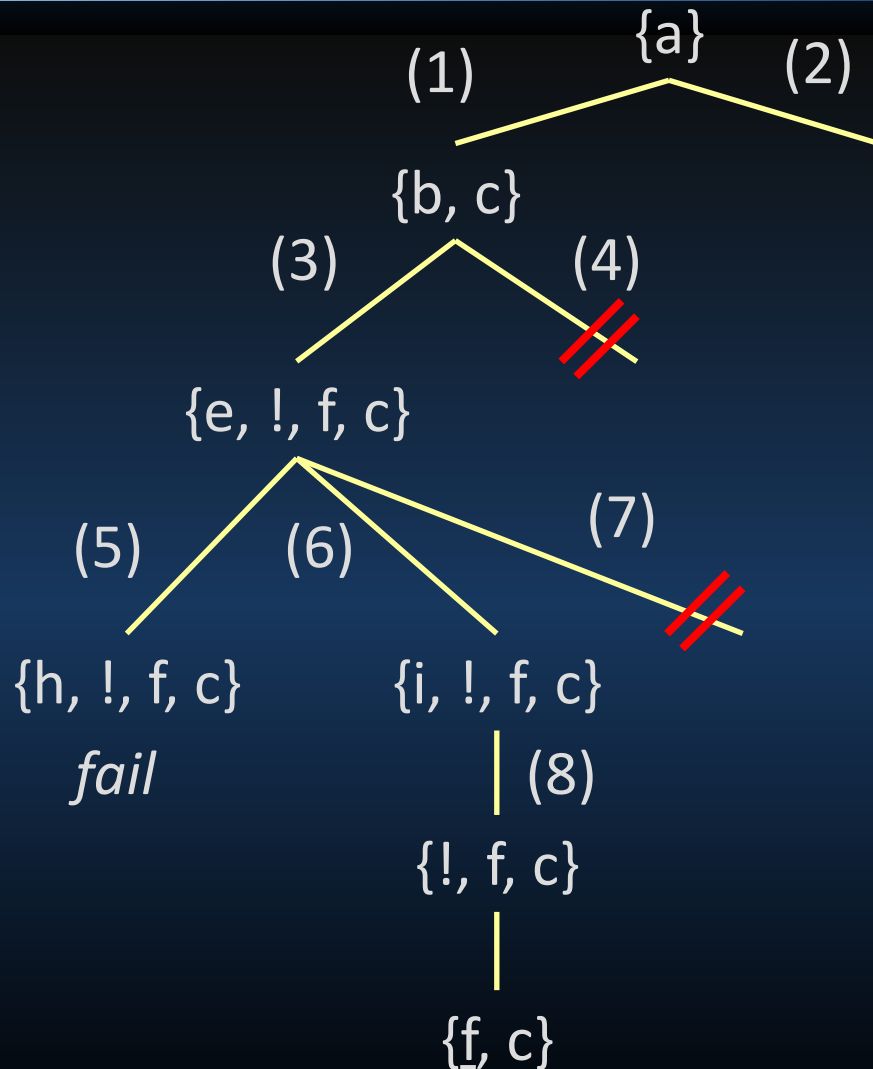


?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.



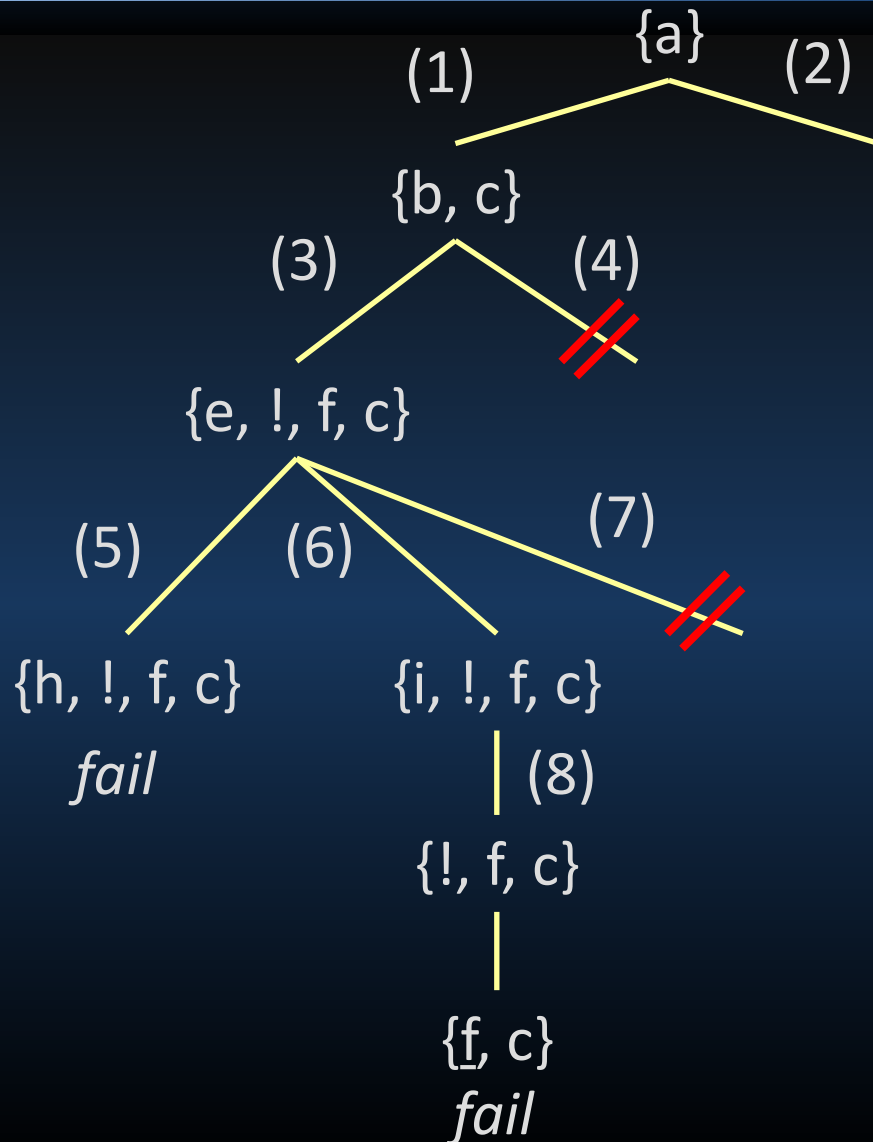
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

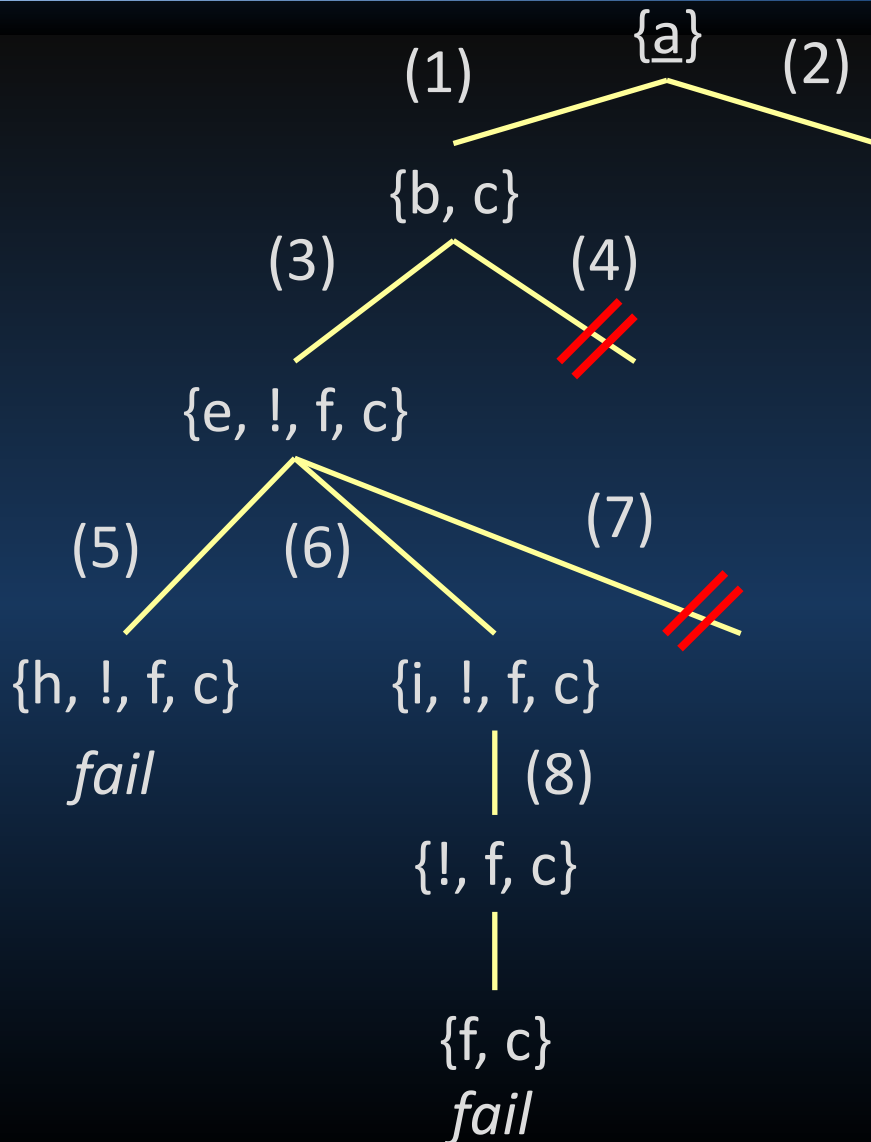
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

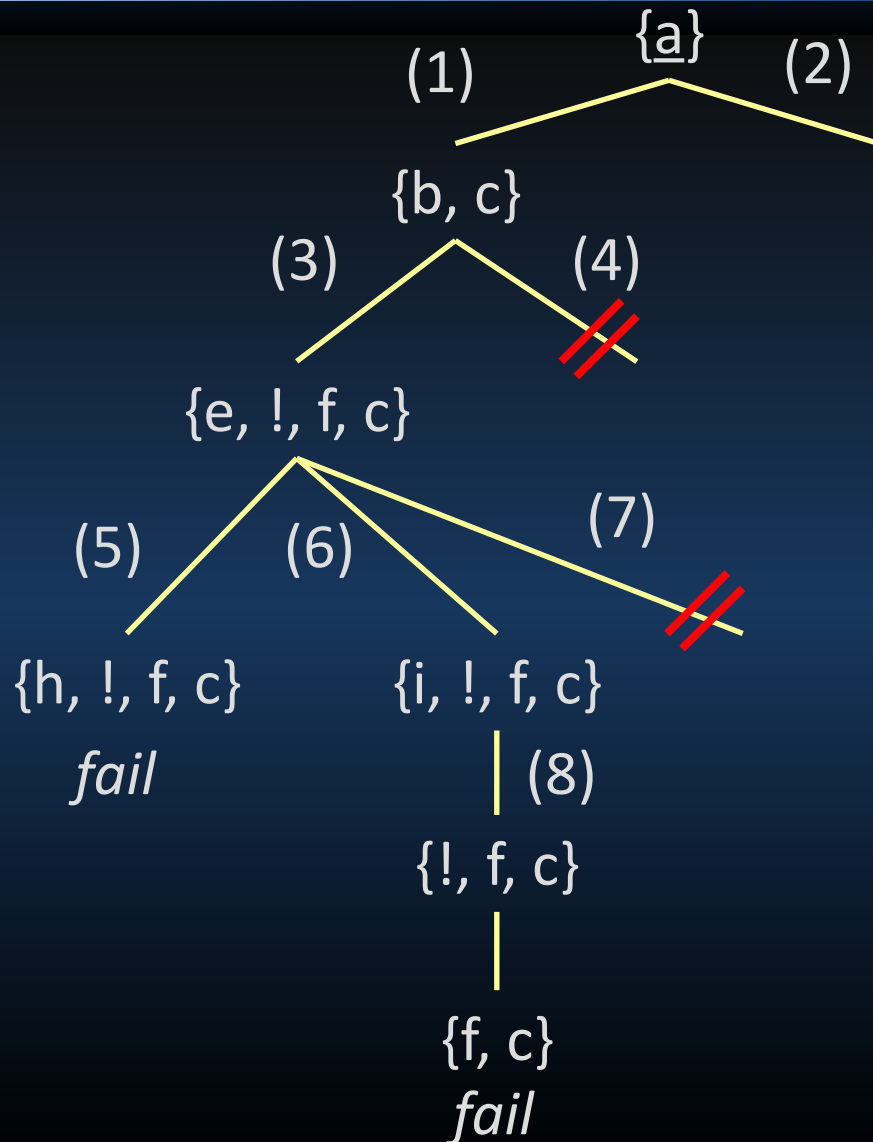
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

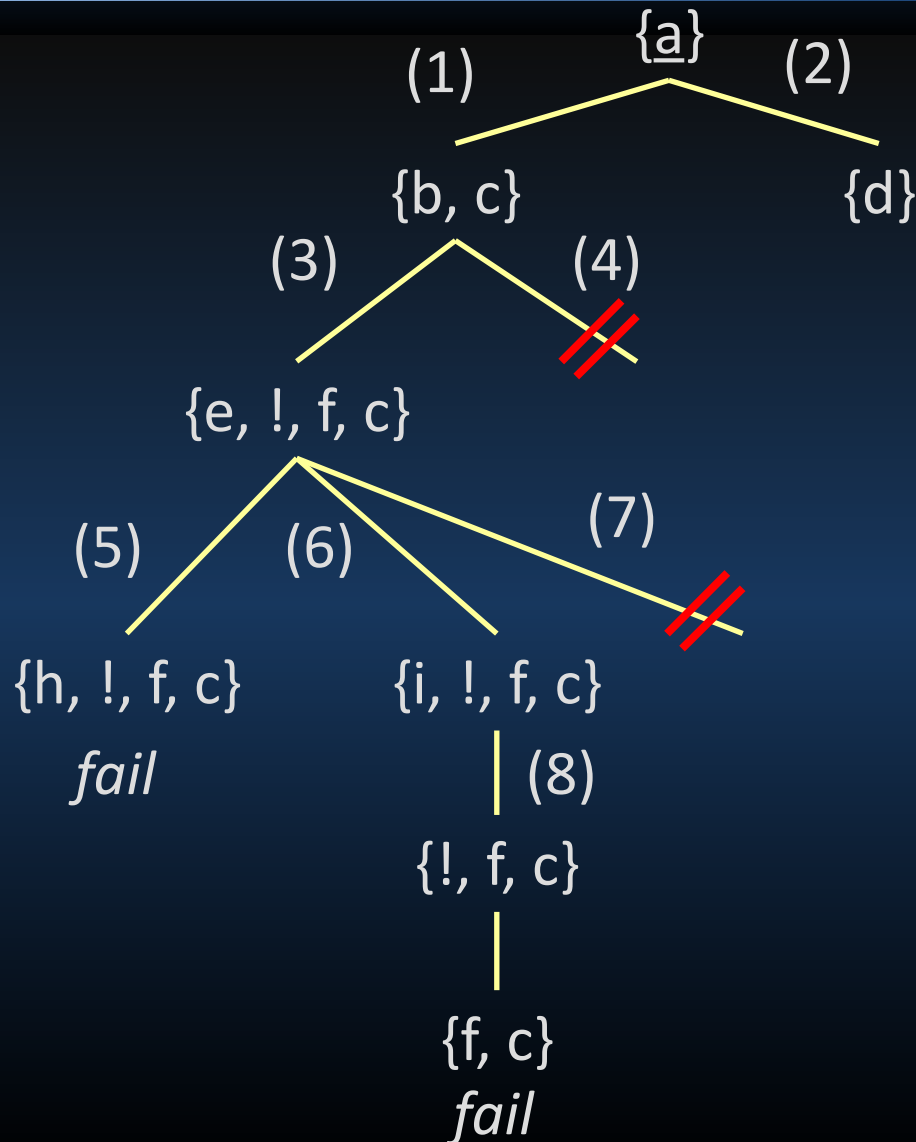
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

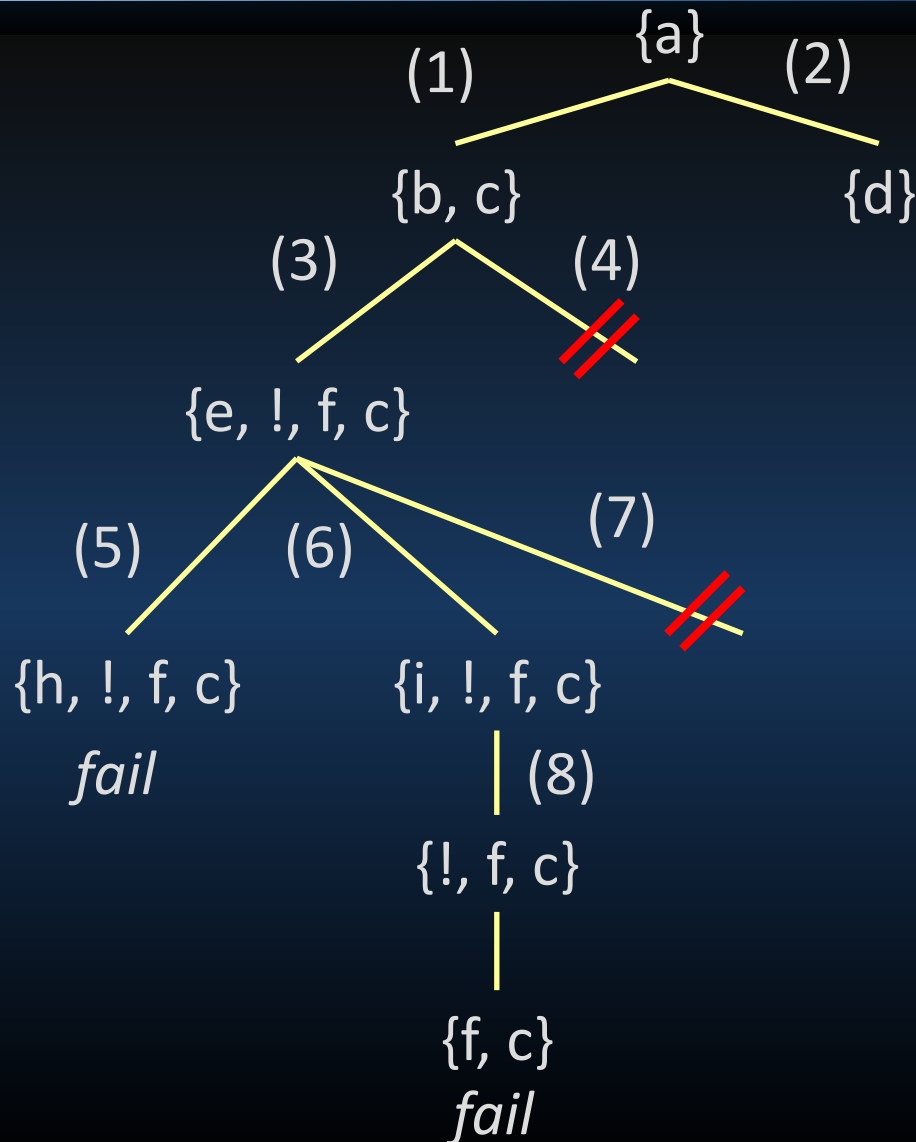
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

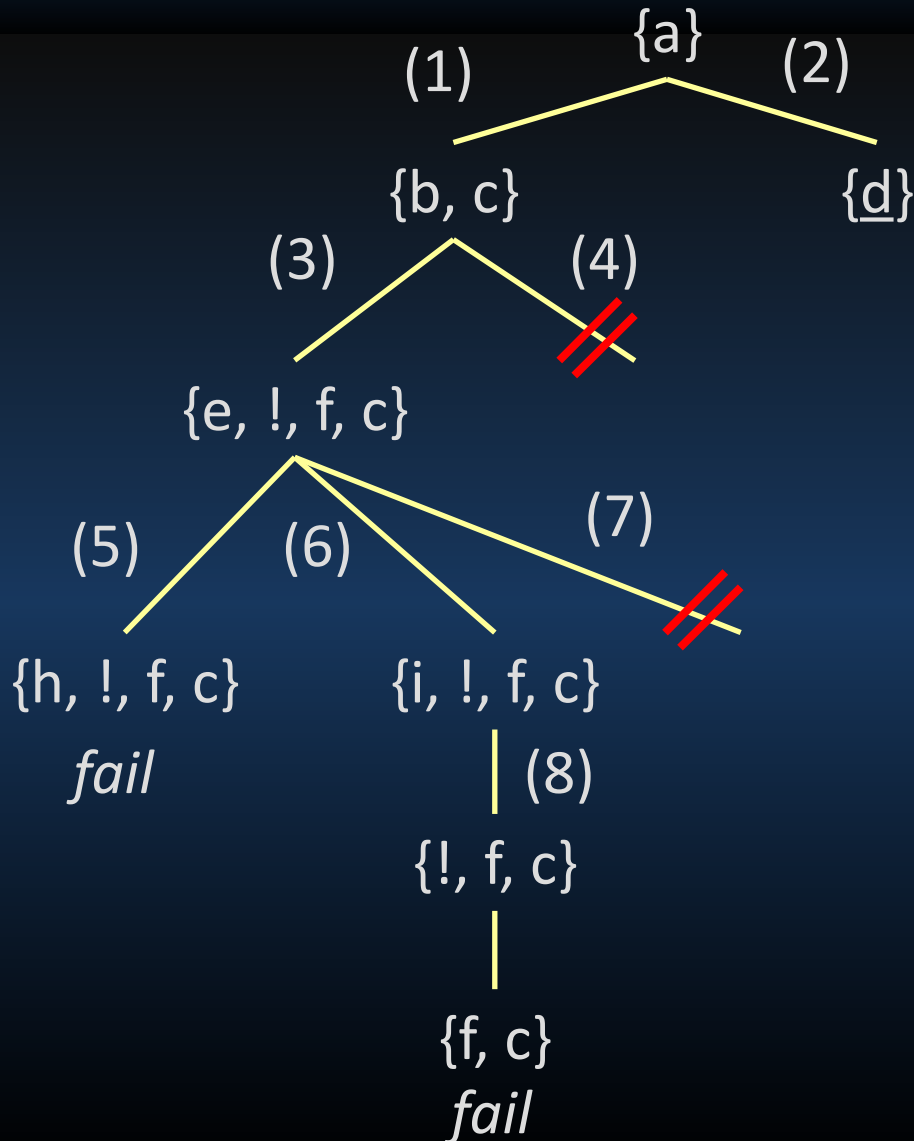
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

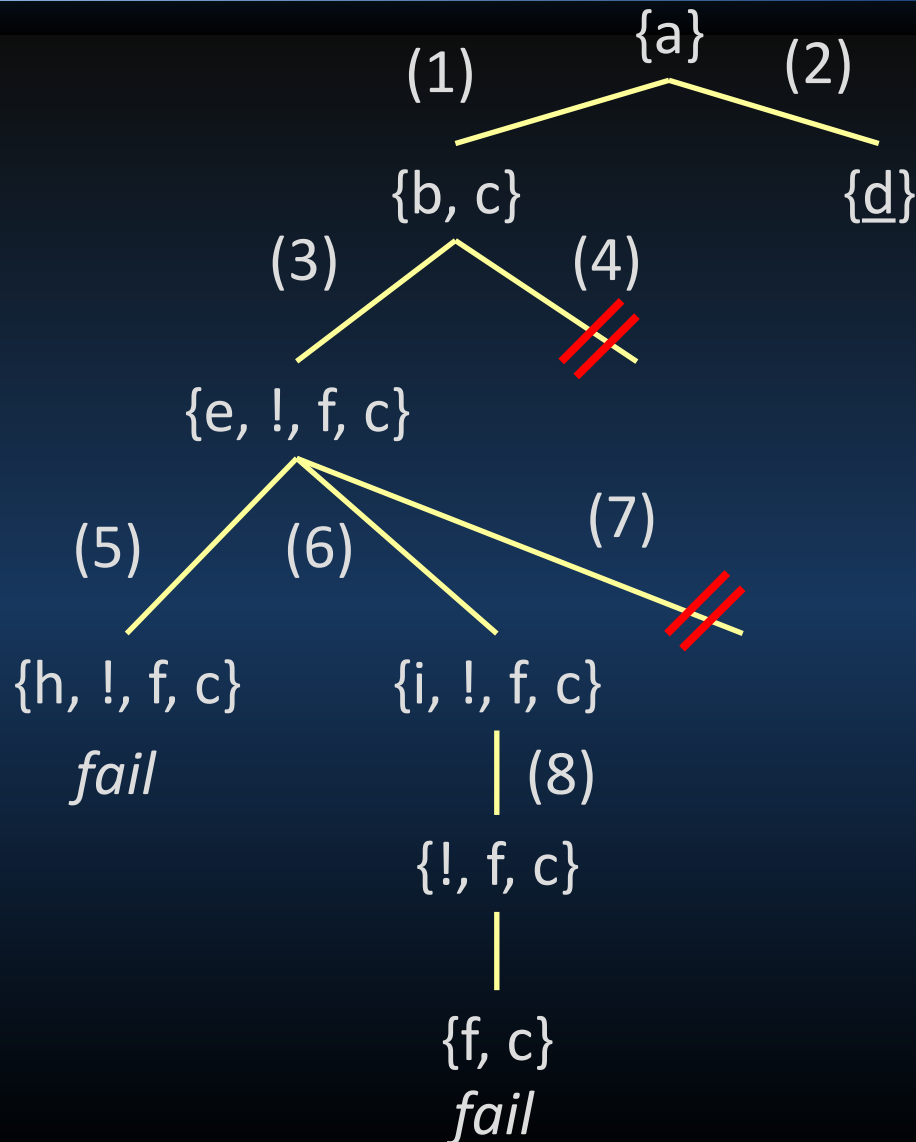
# Cut (!) - Ejemplo



?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# Cut (!) - Ejemplo

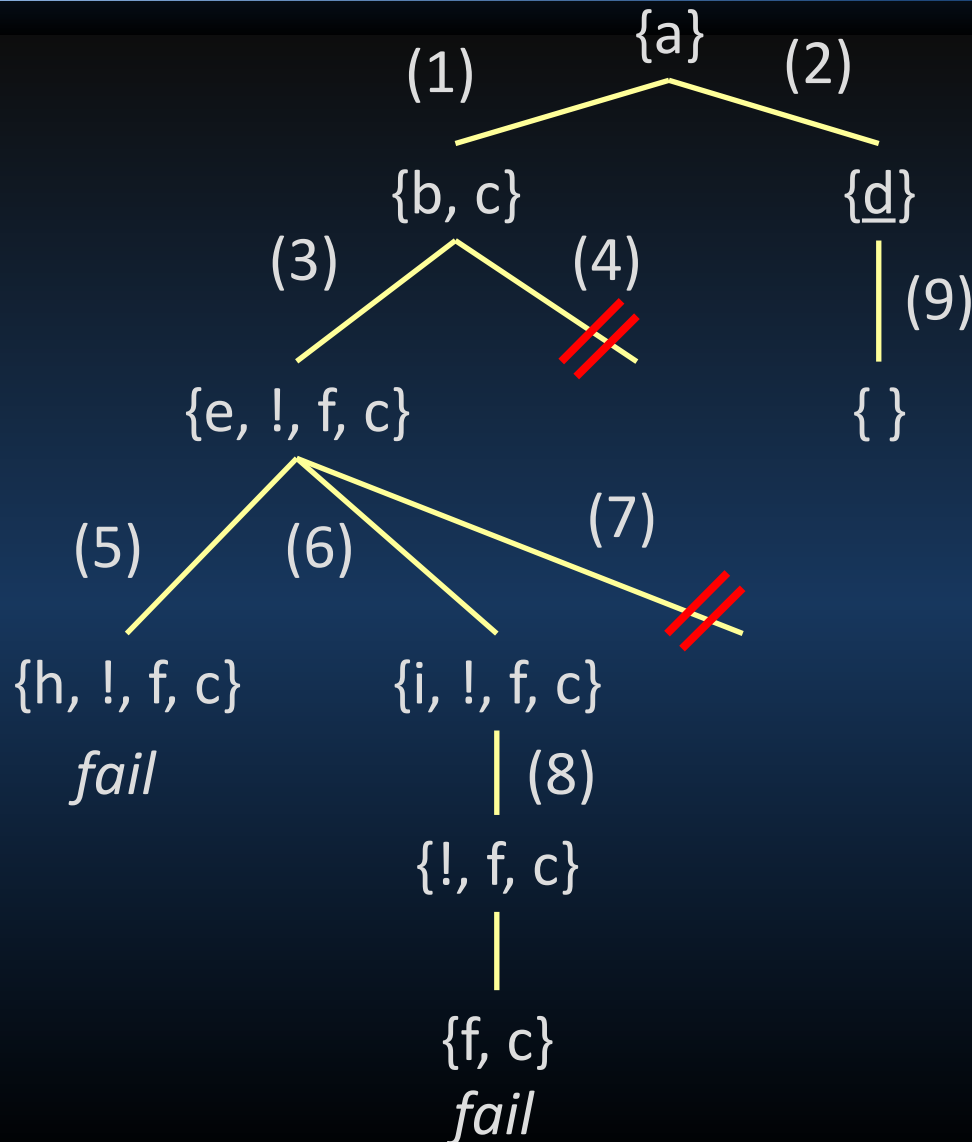


?- a.

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.



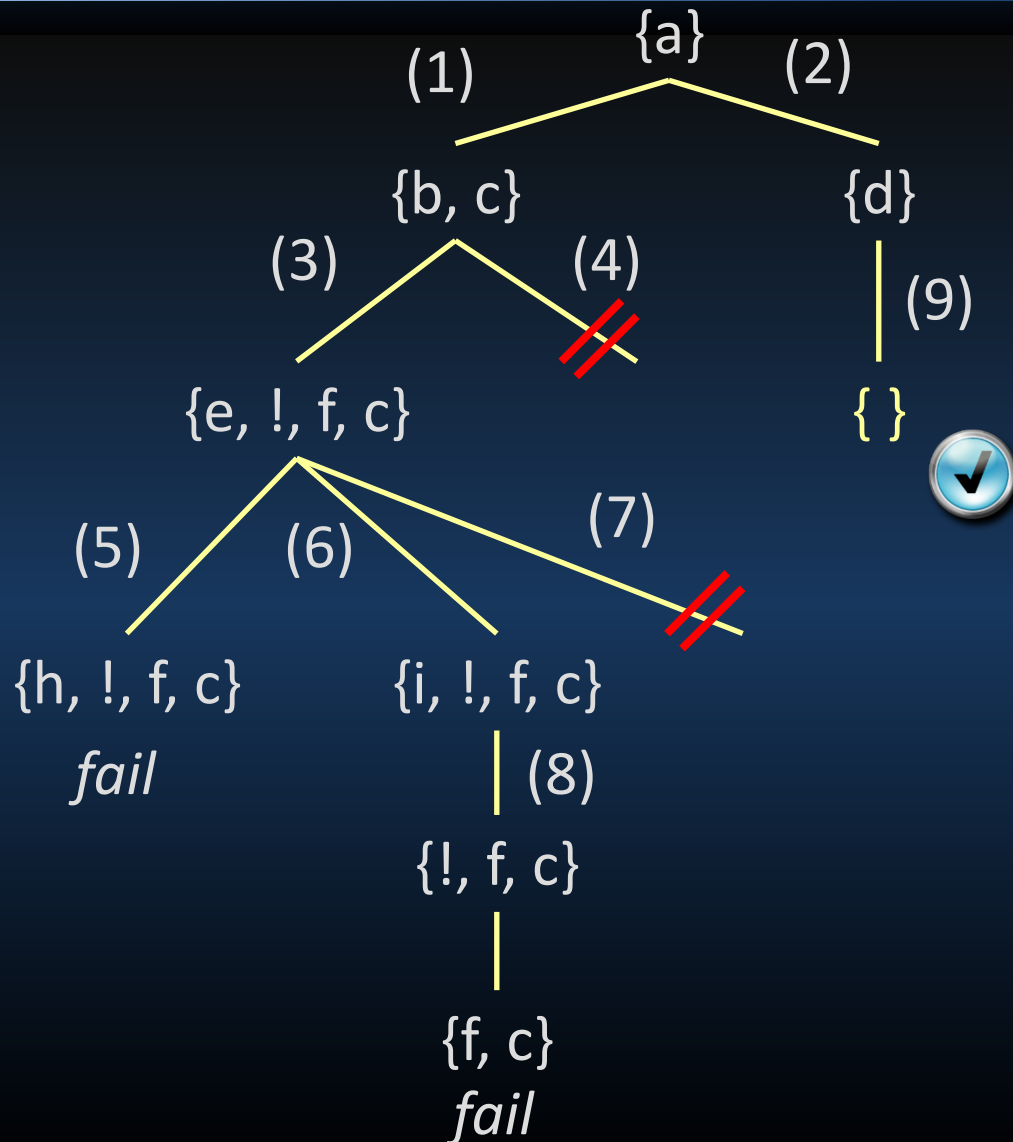
# Cut (!) - Ejemplo



?- a.

(1) a:- b, c.  
(2) a:- d.  
(3) b:- e, !, f.  
(4) b:- g.  
(5) e:- h.  
(6) e:- i.  
(7) e:- g.  
(8) i.  
(9) d.  
(10) g.

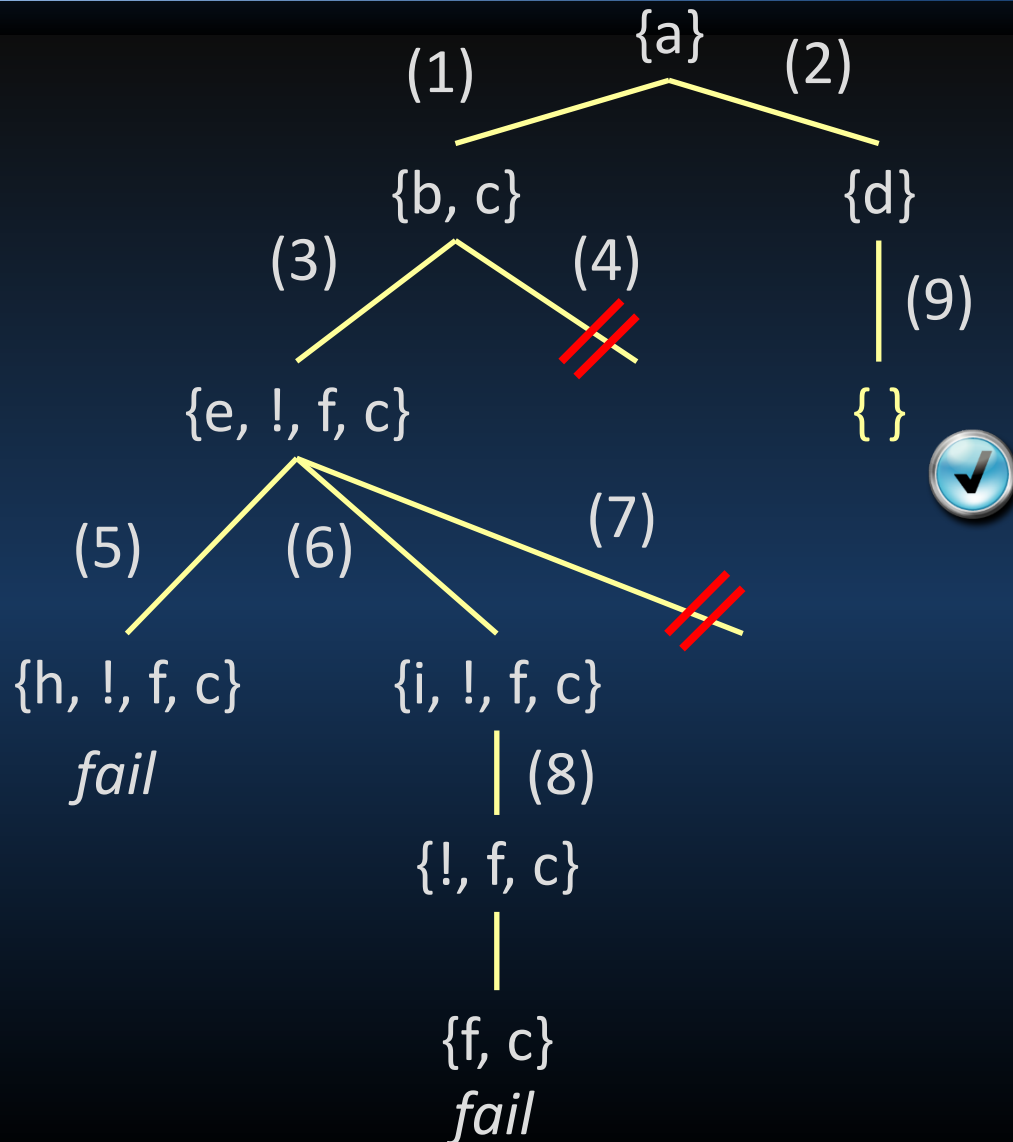
# Cut (!) - Ejemplo



?- a.

(1) a:- b, c.  
(2) a:- d.  
(3) b:- e, !, f.  
(4) b:- g.  
(5) e:- h.  
(6) e:- i.  
(7) e:- g.  
(8) i.  
(9) d.  
(10) g.

# Cut (!) - Ejemplo



?- a.  
true

- (1) a:- b, c.
- (2) a:- d.
- (3) b:- e, !, f.
- (4) b:- g.
- (5) e:- h.
- (6) e:- i.
- (7) e:- g.
- (8) i.
- (9) d.
- (10) g.

# *Cut (!) - Control de Backtracking*

- Consideremos el programa Prolog que se indica a continuación:

```
p(a).  
p(b).  
p(c).
```

- ¿Qué sucede al efectuar la siguiente consulta?

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

# *Cut (!) - Control de Backtracking*

- Consideremos el programa Prolog que se indica a continuación:

```
p(a).  
p(b).  
p(c).
```

- ¿Qué sucede al efectuar la siguiente consulta?

```
?- p(X), !.  
X=a
```

# *Cut (!) - Control de Backtracking*

- Consideremos el programa Prolog que se indica a continuación:

```
p(a).  
p(b).  
p(c).
```

- ¿Qué sucede al efectuar la siguiente consulta?

```
?- p(X), !.  
X=a ;  
false
```

Al utilizar ';' el intérprete de Prolog intenta buscar **respuestas alternativas**, pero fueron **podadas como efecto del cut (!)**

# *Cut (!) - Control de Backtracking*

- Dado el siguiente programa:

```
r(a).    s(d).  
r(b).    s(e).  
r(c).    s(f).
```

- Consideremos la resolución de la siguiente consulta:

```
?- r(X), !, s(Y).
```

# *Cut (!) - Control de Backtracking*

- Dado el siguiente programa:

```
r(a).    s(d).  
r(b).    s(e).  
r(c).    s(f).
```

- Consideremos la resolución de la siguiente consulta:

```
?- r(X), !, s(Y).  
X=a  Y=d
```



# Cut (!) - Control de Backtracking

- Dado el siguiente programa:

```
r(a).    s(d).  
r(b).    s(e).  
r(c).    s(f).
```

- Consideremos la resolución de la siguiente consulta:

```
?- r(X), !, s(Y).  
X=a  Y=d ;  
X=a  Y=e
```

No se encuentran **respuestas alternativas** para **r(X)**, pero si para **s(Y)**

El **cut (!)** poda **soluciones alternativas** de la meta conjuntiva **a su izquierda**, y **admite soluciones alternativas** de la meta conjuntiva **a su derecha**

# *Cut (!) - Control de Backtracking*

- Dado el siguiente programa:

```
r(a).    s(d).  
r(b).    s(e).  
r(c).    s(f).
```

- Consideremos la resolución de la siguiente consulta:

```
?- r(X), !, s(Y).  
X=a  Y=d ;  
X=a  Y=e ;  
X=a  Y=f
```

# *Cut (!) - Control de Backtracking*

- Dado el siguiente programa:

```
r(a).    s(d).  
r(b).    s(e).  
r(c).    s(f).
```

- Consideremos la resolución de la siguiente consulta:

```
?- r(X), !, s(Y).  
X=a  Y=d  ;  
X=a  Y=e  ;  
X=a  Y=f  ;  
false
```

# *Cut (!) - Aplicaciones*

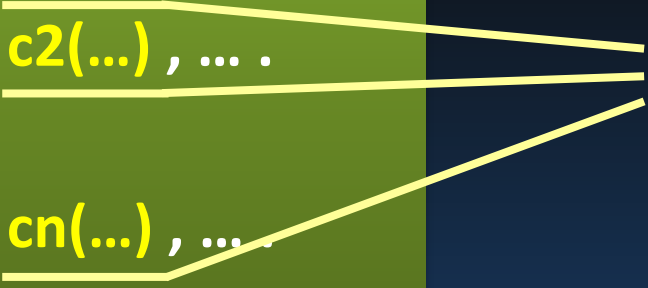
- Entre los **principales uso del cut (!)** se destaca su aplicación para **mejorar la eficiencia** mediante la poda del espacio de búsqueda contemplado en el proceso de derivación, admitiendo, en algunos casos, **omitir la especificación de condiciones asociadas a predicados**.
- El **cut (!)** también permite **implementar la negación por falla** (not/1).

# Cut (!) - Aplicaciones

- Consideremos un predicado definido de la siguiente forma:

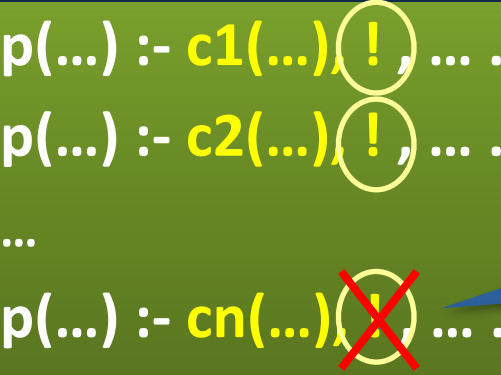
```
p(...) :- c1(...), ... .  
p(...) :- c2(...), ... .  
...  
p(...) :- cn(...), ... .
```

Condiciones



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

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



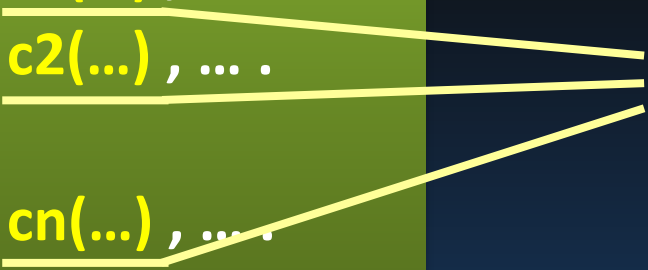
Este último cut no es necesario ya que no hay más alternativas por debajo

# Cut (!) - Aplicaciones: Omitir Condiciones

- Consideremos nuevamente el siguiente predicado:


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

Condiciones



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

```
p(...) :- c1(...), (!), ... .  
p(...) :- c2(...), (!), ... .  
...  
p(...) :- cn-1(...), (!), ... .  
p(...) :- cn(...), ... .
```



Cuidado! Ahora estos cuts son esenciales para la correctitud del programa

# Cut (!) - Aplicaciones: Ejemplo

- Consideremos el siguiente predicado para calcular la diferencia absoluta entre dos números 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 añadiendo un cut:

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

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

- 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.
```

Cuidado! Ahora este cut es esencial para la correctitud del programa

# *Cut (!) - Aplicaciones: Negación por Falla*

- La **negación por falla** (predicado **not/1**) se define en términos de los predicados **!/0** y **fail/0**.
- El único efecto del predicado **fail/0** es hacer fallar la consulta.

```
not(Goal):- Goal, !, fail.  
not(_Goal).
```

Si la meta **Goal** tiene éxito, entonces se produce la poda por efecto del cut (!) y **not(Goal)** falla. En caso contrario, si la meta **Goal** falla, el cut no tiene efecto y **not(Goal)** tiene éxito



# *Cut (!) - Implicancias*

- El operador de corte (!) debe ser **empleado con cuidado y analizando las consecuencias** en cada situación.
- Si se emplea mal puede causar **comportamiento poco predecible** en algunos casos.
- Ventajas y Desventajas:
  - (+) **Implementaciones más eficientes** en algunos casos.
  - (+) **Elimina soluciones repetidas** si se hace un uso cuidadoso del mismo.
  - (-) **Oscurece la semántica** de los programas.

# *Predicados Dinámicos*

- Un **programa** escrito en Prolog usualmente se especifica de forma **estática**, a partir de un **archivo fuente**.
- SWI-Prolog ofrece una familia de predicados que permiten **agregar y quitar cláusulas** del programa **dinámicamente**. Es decir, permiten modificar dinámicamente la definición de predicados.

Estos elementos le aportan una gran flexibilidad al lenguaje

# *Predicados Dinámicos – assertz/1*

- El predicado **assertz(+Term)** recibe como argumento un término y lo agrega como una nueva cláusula del programa.  
**Term** se agrega como **última cláusula** del predicado correspondiente.

# Predicados Dinámicos – assertz/1

El predicado **dynamic/1** informa al intérprete que la **definición de los predicados** indicados a continuación **puede cambiar durante la ejecución**

prog1.pl

```
:- dynamic b/0, d/0.
```

```
a :- b.
```

```
c :- d.
```

```
d.
```

# *Predicados Dinámicos – assertz/1*

?- a.

false

?- assertz(b).

true

?- a.

true

Intérprete

a :- b.

c :- d.

d.

b.

Dinámicos:

b/0 , d/0

# Predicados Dinámicos – assertz/1

- Como todo predicado **assertz(+Term)** puede ser parte del cuerpo de una regla

?- a.  
true

prog2.pl

```
:- dynamic b/0, c/0.
```

```
a :- assertz(b:-c), b.
```

```
c :- d.
```

```
d.
```

CUIDADO! El backtracking **no deshace** el **efecto colateral** provocado por el assert.

# *Predicados Dinámicos – retract/1*

- El predicado **retract(+Term)** quita del programa la primera cláusula que unifique con **Term**. La consulta falla si no existe tal cláusula.

Observación: El argumento de entrada **no** puede ser una **variable**.

# *Predicados Dinámicos – retract/1*

prog.pl

```
:- dynamic b/0, c/0.
```

```
a :- b.
```

```
c :- d.
```

```
d.
```



# Predicados Dinámicos – retract/1

?- c.

true

?- retract(c:-d).

true

?- c.

false

Intérprete

a :- b.

~~c :- d.~~

d.

Dinámicos:

b/0 , c/0

# *Predicados Dinámicos – Otros*

- Otros predicados de la misma familia:
  - **asserta(+Term)**: similar a assertz/1.  
El término **Term** se agrega como **primera cláusula** del predicado correspondiente.
  - **retractall(+Head)**: quita del programa todas las cláusulas cuya cabeza unifique con **Head**.

# Otros Predicados – Findall/3

## findall(+Term,:Goal,-ListOfTerms)

- Retorna en **ListOfTerms** la lista de todas las posibles instanciaciones de **Term** obtenidas al considerar todas las soluciones alternativas de la meta **Goal**.
- Este predicado **nunca falla**, ya que si no hay soluciones retorna en **ListOfTerms** la **lista vacía**.

**Findall/3** es un **predicado de segundo orden** (recibe predicados como argumento), ya que **Goal** corresponde a una **meta** posiblemente conjuntiva. Es decir, la resolución del **findall/3** implica la resolución de la **consulta especificada en Goal**

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall(X, es_par(X), L).
```

```
?- findall(X, (es_par(X), X<5), L).
```

```
?- findall(X, (es_par(X), X>=27), L).
```

Recordar: Goal puede ser  
una meta conjuntiva

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall(X, es_par(X), L).
```

```
L = [2, 4, 6, 8, 10]
```

```
?- findall(X, (es_par(X), X<5), L).
```

```
?- findall(X, (es_par(X), X>=27), L).
```

Recordar: Goal puede ser  
una meta conjuntiva

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall(X, es_par(X), L).  
L = [2, 4, 6, 8, 10]  
?- findall(X, (es_par(X), X<5), L).  
L = [2, 4]  
?- findall(X, (es_par(X), X>=27), L).
```

Recordar: Goal puede ser  
una meta conjuntiva

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall(X, es_par(X), L).  
L = [2, 4, 6, 8, 10]  
?- findall(X, (es_par(X), X<5), L).  
L = [2, 4]  
?- findall(X, (es_par(X), X>=27), L).  
L = [ ]
```

Recordar: Goal puede ser  
una meta conjuntiva

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall([X,Y], (es_par(X), not(es_par(Y)), X < Y), L).
```

```
?- findall([X,Y], (es_par(X), num(Y), not(es_par(Y)), X < Y), L).
```



# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall([X,Y], (es_par(X), not(es_par(Y)), X < Y), L).  
L = []
```

```
?- findall([X,Y], (es_par(X), num(Y), not(es_par(Y)), X < Y), L).
```

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall([X,Y], (es_par(X), not(es_par(Y)), X < Y), L).
```

L = []

No hay [X,Y] que satisfaga la meta conjuntiva porque al momento de evaluar `not(es_par(Y))`, Y no está instanciada!  
Por lo tanto, `es_par(Y)` tiene éxito y `not(es_par(Y))` falla

```
?- findall([X,Y], (es_par(X), num(Y), not(es_par(Y)), X < Y), L).
```

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall([X,Y], (es_par(X), not(es_par(Y)), X < Y), L).
```

```
L = []
```

Se instancia Y previamente haciendo uso de num/1 para lograr el **comportamiento esperado**: obtener en L los pares [X,Y] tales que X es par, Y es impar, y X<Y

```
?- findall([X,Y], (es_par(X), num(Y), not(es_par(Y)), X < Y), L).
```

# Otros Predicados – Findall/3

- Consideremos el siguiente programa:

```
num(1).num(2).          es_par(X):- num(X), 0 is X mod 2.  
num(3).num(4).  
num(5).num(6).  
num(7).num(8).  
num(9).num(10).
```

- ¿Cuál es la respuesta a las siguientes consultas?

```
?- findall([X,Y], (es_par(X), not(es_par(Y)), X < Y), L).
```

```
L = []
```

Se instancia Y previamente haciendo uso de num/1 para lograr el **comportamiento esperado**: obtener en L los pares [X,Y] tales que X es par, Y es impar, y X<Y

```
?- findall([X,Y], (es_par(X), num(Y), not(es_par(Y)), X < Y), L).
```

```
L = [[2, 3], [2, 5], [2, 7], [2,9], [4,5], [4,7], [4,9], [6,7], [6,9],[8,9]]
```

# Otros Predicados – Forall/2

## forall(:Cond, :Action)

- Tiene éxito si **para todas las soluciones** alternativas de **Cond** (antecedente) se verifica **Action** (consecuente).

## bagof(+Term, :Goal, -Bag)

- Comportamiento similar al predicado findall/3

## setof(+Term, :Goal, -Set)

- Similar al bagof/3, pero ordena la lista resultante y elimina elementos duplicados.

Probar Ejemplos 😊

# *Predicados de Entrada/Salida*

- El predicado **read/1** recibe un término como argumento de entrada. Solicita el ingreso de un término por la entrada estándar (teclado) e intenta unificarlo con el argumento de entrada.
- El predicado **readln/1** recibe un término como argumento de entrada. Solicita el ingreso de una secuencia de elementos (separados por espacios en blanco) finalizada en <ENTER>, los cuales son convertidos en una lista de átomos y números. Finalmente intenta unificar el argumento de entrada con dicha lista.

# *Predicados de Entrada/Salida*

- El predicado **write/1** recibe un término como argumento de entrada y lo muestra por la salida estándar (pantalla).
- El predicado **nl/0** realiza un salto de línea en la salida estándar.
- El predicado **writeln/1** combina las funcionalidades de los predicados **write/1** y **nl/0**.

Todos estos predicados pueden ser utilizados en el cuerpo de otros predicados