Programación III
Presentación de Cátedra
Parte 2

Alejandro Sánchez
aljsanchez@gmail.com

Departamento de Informática
Universidad Nacional de San Luis

San Luis

Programación III
Presentación de Cátedra – Parte 2

Objetivos sencillos primero

- Orden de las cláusulas
  1. las más específicas
  2. las más generales (recursivas)
- Orden de los términos dentro de una cláusula
  1. los términos más específicos
  2. los términos más generales (recursivos)

Objetivos sencillos primero

- Orden de las cláusulas
  1. las más específicas
  2. las más generales (recursivas)
- Orden de los términos dentro de una cláusula
  1. los términos más específicos
  2. los términos más generales (recursivos)

```
antecesor1(X,Y):- padre_de(X, Y).
antecesor1(X,Y):- padre_de(X,Z),antecesor1(Z,Y).
```

- Funciona siempre
- Respeta las directivas
- Es eficiente

Objetivos sencillos primero

- Orden de las cláusulas
  1. las más específicas
  2. las más generales (recursivas)
- Orden de los términos dentro de una cláusula
  1. los términos más específicos
  2. los términos más generales (recursivos)

```
antecesor2(X,Y):- padre_de(X,Z),antecesor2(Z,Y).
antecesor2(X,Y):- padre_de(X,Y).
```

- Es menos eficiente

Objetivos sencillos primero

- Orden de las cláusulas
  1. las más específicas
  2. las más generales (recursivas)
- Orden de los términos dentro de una cláusula
  1. los términos más específicos
  2. los términos más generales (recursivos)

```
antecesor3(X,Y):- padre_de(X,Y).
antecesor3(X,Y):- antecesor3(Z,Y), padre_de(X, Z).
```

- Solo funciona en algunos casos

Objetivos sencillos primero

- Orden de las cláusulas
  1. las más específicas
  2. las más generales (recursivas)
- Orden de los términos dentro de una cláusula
  1. los términos más específicos
  2. los términos más generales (recursivos)

```
antecesor4(X,Y):- antecesor4(Z,Y), padre_de(X, Z).
antecesor4(X,Y):- padre_de(X,Y).
```

- No funciona: bucle infinito

Ejemplo: antecesor3

```
antecesor3(X,Y):- padre_de(X,Y).
antecesor3(X,Y):- antecesor3(Z,Y), padre_de(X, Z).

padre_de(carlos, fernando).
padre_de(antonio, maria).
padre_de(antonio, carlos).
```

- ?- antecesor3(antonio, carlos).

Ejemplo: antecesor3

```
antecesor3(X,Y):- padre_de(X,Y).
antecesor3(X,Y):- antecesor3(Z,Y), padre_de(X, Z).

padre_de(carlos, fernando).
padre_de(antonio, maria).
padre_de(antonio, carlos).
```

- ?- antecesor3(antonio, carlos).
  Verifica la relación
- ?- antecesor3(carlos, maria).

Ejemplo: antecesor3

```
antecesor3(X,Y):- padre_de(X,Y).
antecesor3(X,Y):- antecesor3(Z,Y), padre_de(X, Z).

padre_de(carlos, fernando).
padre_de(antonio, maria).
padre_de(antonio, carlos).
```

- ?- antecesor3(antonio, carlos).
  Verifica la relación
- ?- antecesor3(carlos, maria).
  Es falsa pero queda en ciclo infinito.
- ?- antecesor3(X, Y).

Ejemplo: antecesor3

```
antecesor3(X,Y):- padre_de(X,Y).
antecesor3(X,Y):- antecesor3(Z,Y), padre_de(X, Z).

padre_de(carlos, fernando).
padre_de(antonio, maria).
padre_de(antonio, carlos).
```

- ?- antecesor3(antonio, carlos).
  Verifica la relación
- ?- antecesor3(carlos, maria).
  Es falsa pero queda en ciclo infinito.
- ?- antecesor3(X, Y).
  Imprimer algún resultado, pero luego entra en ciclo infinito.

Acotar el espacio de búsqueda

```
amigos(pedro,antonio).
amigos(pedro,flora).
amigos(pedro,juan).
amigos(pedro,vicente).
amigos(luis,felipe).
amigos(luis,maria).
amigos(luis,vicente).
amigos(carlos,paloma).
amigos(carlos,lucia).
amigos(carlos,juan).
amigos(carlos,vicente).
amigos(fernando,eva).
amigos(fernando,pedro).
millonario(pedro).
millonario(antonio).
millonario(flora).
soltero(pedro).
soltero(flora).
soltero(eva).
soltero(luis).
```

```
amigos_interesantes(X,Z):-
 amigos(Y,X), amigos(Y,Z),
 millonario(Z), soltero(Z).
```

- Resolver objetivos menos frecuentes antes
- Se cambia orden de objetivos

Acotar el espacio de búsqueda

```
amigos(pedro,antonio).
amigos(pedro,flora).
amigos(pedro,juan).
amigos(pedro,vicente).
amigos(luis,felipe).
amigos(luis,maria).
amigos(luis,vicente).
amigos(carlos,paloma).
amigos(carlos,lucia).
amigos(carlos,juan).
amigos(carlos,vicente).
amigos(fernando,eva).
amigos(fernando,pedro).
millonario(pedro).
millonario(antonio).
millonario(flora).
soltero(pedro).
soltero(flora).
soltero(eva).
soltero(luis).
```

```
amigos_interesantes(X,Z):-
 amigos(Y,X), amigos(Y,Z),
 millonario(Z), soltero(Z).
```

- Resolver objetivos menos frecuentes antes
- Se cambia orden de objetivos

¿Cómo sería un mejor orden?

Acotar el espacio de búsqueda

```
amigos(pedro,antonio).
amigos(pedro,flora).
amigos(pedro,juan).
amigos(pedro,vicente).
amigos(luis,felipe).
amigos(luis,maria).
amigos(luis,vicente).
amigos(carlos,paloma).
amigos(carlos,lucia).
amigos(carlos,juan).
amigos(carlos,vicente).
amigos(fernando,eva).
amigos(fernando,pedro).
millonario(pedro).
millonario(antonio).
millonario(flora).
soltero(pedro).
soltero(flora).
soltero(eva).
soltero(luis).
```

```
amigos_interesantes(X,Z):-
 amigos(Y,X), amigos(Y,Z),
 millonario(Z), soltero(Z).
```

- Resolver objetivos menos frecuentes antes
- Se cambia orden de objetivos

¿Cómo sería un mejor orden?

```
amigos_interesantes(X,Z):-
 millonario(Z), soltero(Z),
 amigos(Y,X), amigos(Y,Z).
```

Programación III
Presentación de Cátedra – Parte 2

El operador corte: !

Dada una regla `C` de la forma $\texttt{A} :- \texttt{B}_1, \ldots, \texttt{B}_k, !, \texttt{B}_{k+2}, \ldots, \texttt{B}_n.$

- Si la consulta `G` unifica con `A` y $\texttt{B}_1, \ldots, \texttt{B}_k$ se satisfacen
  - Se fija esta regla para reducir `G`
  - A `G` se la denomina *fin/meta padre* del corte
  - Se ignoran reglas alternativas que unifiquen con `G`

  Poda todas las sentencias debajo de `C`
- Si $\texttt{B}_i$ con $\texttt{i} > \texttt{k}$ fracasan
  - La vuelta atrás solo puede hacerse hasta el corte
  - Las alternativas a $\texttt{B}_i$ con $\texttt{i} \leq \texttt{k}$ fueron podadas

  No afecta metas a su derecha
  Poda toda solución alternativa a metas a su izquierda
- Si el backtracking llega a !
  - Continúa con la elección previa a unificar `G` con `A`

El operador corte: ! : Resumen del efecto

El corte ! se satisface y
congela todas las elecciones hechas
desde que *su fin padre* se unificó
con la cabeza de la sentencia que le contiene.

mezcla(+L,+M,-N)

```
1 mezcla([X|Xs],[Y|Ys],[X|Zs]):-X<Y,mezcla(Xs,[Y|Ys],Zs).
2 mezcla([X|Xs],[Y|Ys],[X,Y|Zs]):-X=Y,mezcla(Xs,Ys,Zs).
3 mezcla([X|Xs],[Y|Ys],[Y|Zs]):-X>Y,mezcla([X|Xs],Ys,Zs).
4 mezcla(Xs,[ ],Xs).
5 mezcla([ ],Ys,Ys).
```

- Programa determinístico: solo una sentencia puede aplicar
- Agregar el corte permite expresar la naturaleza mutuamente exclusiva de los tests

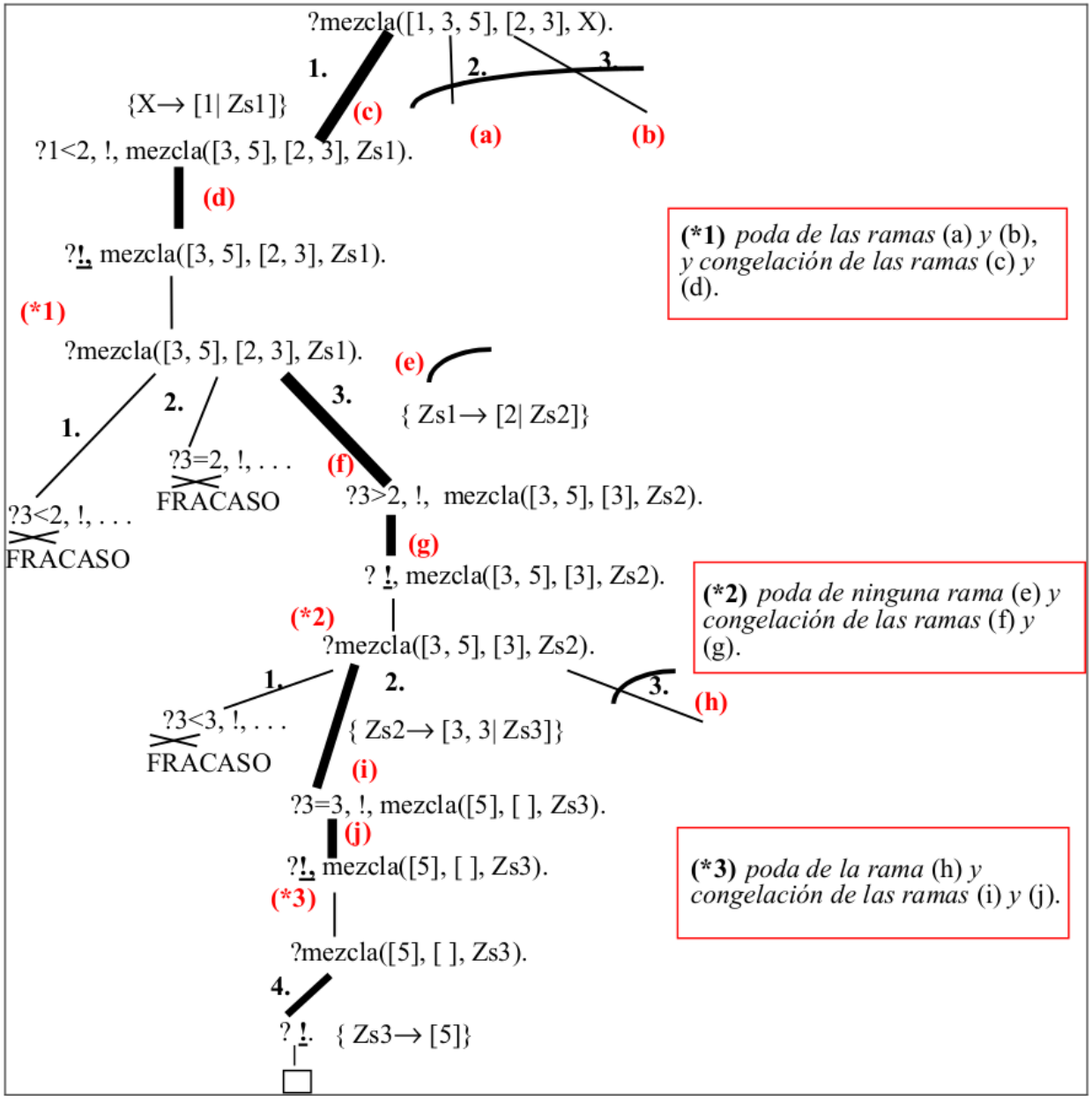
`mezcla(+L,+M,-N)` con corte

```
1 mezcla([X|Xs],[Y|Ys],[X|Zs]):-X<Y,!,mezcla(Xs,[Y|Ys],Zs).
2 mezcla([X|Xs],[Y|Ys],[X,Y|Zs]):-X=Y,!,mezcla(Xs,Ys,Zs).
3 mezcla([X|Xs],[Y|Ys],[Y|Zs]):-X>Y,!,mezcla([X|Xs],Ys,Zs).
4 mezcla(Xs,[ ],Xs):-!.
5 mezcla([ ],Ys,Ys).
```

- Luego del test en las tres primeras sentencias
- Como primer meta en la cuarta
  - Elimina solución redundante para

    `?- mezcla([],[],X).`

`mezcla(+L,+M,-N)` con corte: Árbol de búsqueda



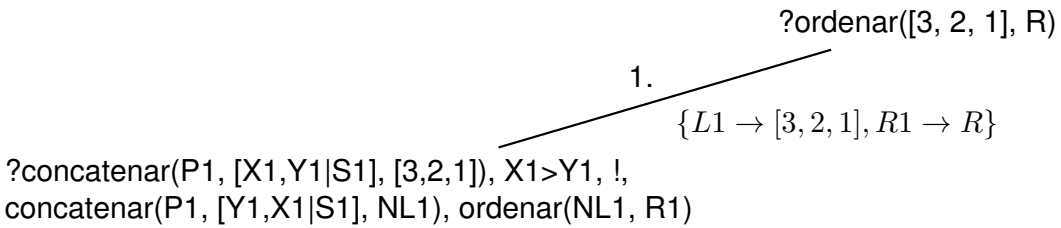
Programación III
Presentación de Cátedra – Parte 2

Corte verde: Caracterización

- No alteran el significado declarativo
- Sirven para expresar determinismo
  - La parte del cuerpo/o la cabeza que precede al corte excluye todos los demás casos

Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```

?ordenar([3, 2, 1], R)

1. $\{L1 \to [3,2,1], R1 \to R\}$

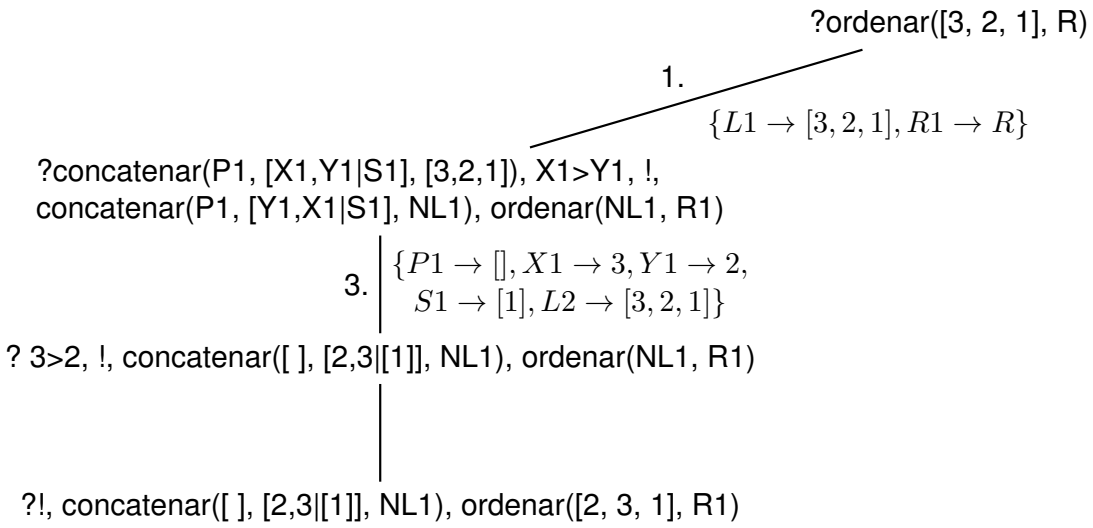
?concatenar(P1, [X1,Y1|S1], [3,2,1]), X1>Y1, !,
concatenar(P1, [Y1,X1|S1], NL1), ordenar(NL1, R1)

3. $\{P1 \to [], X1 \to 3, Y1 \to 2, S1 \to [1], L2 \to [3,2,1]\}$

? 3>2, !, concatenar([ ], [2,3|[1]], NL1), ordenar(NL1, R1)

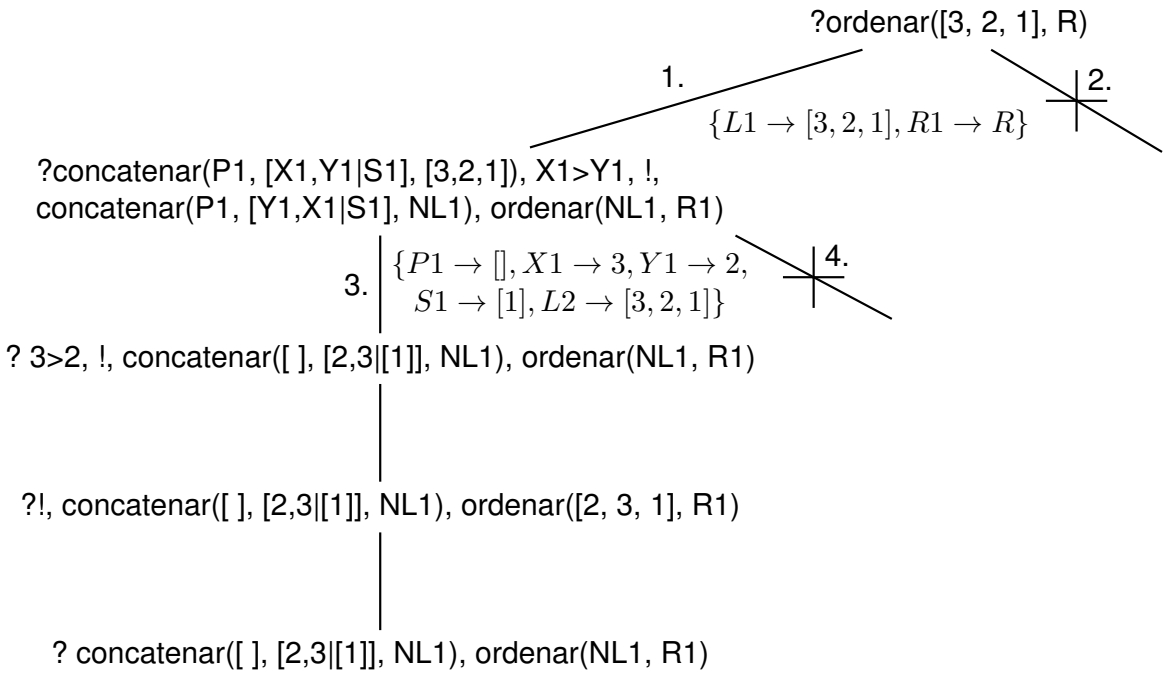
Corte verde: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L):-ordenada(L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



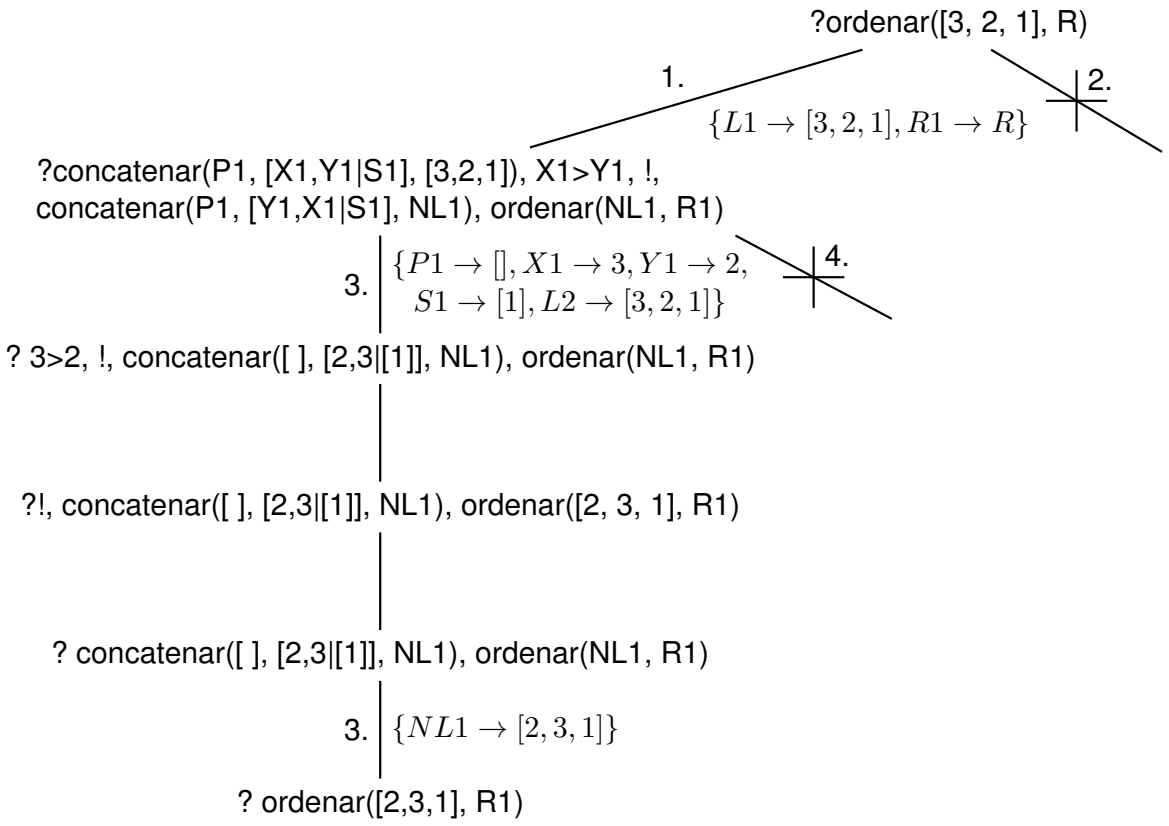
Corte verde: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L):-ordenada(L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



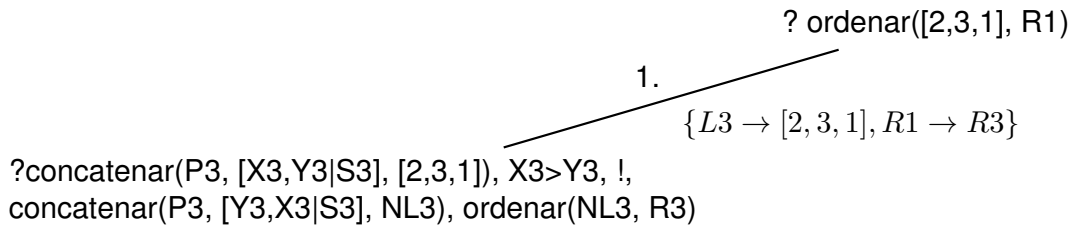
Corte verde: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L):-ordenada(L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```

? ordenar([2,3,1], R1)

1. $\{L3 \to [2,3,1], R1 \to R3\}$

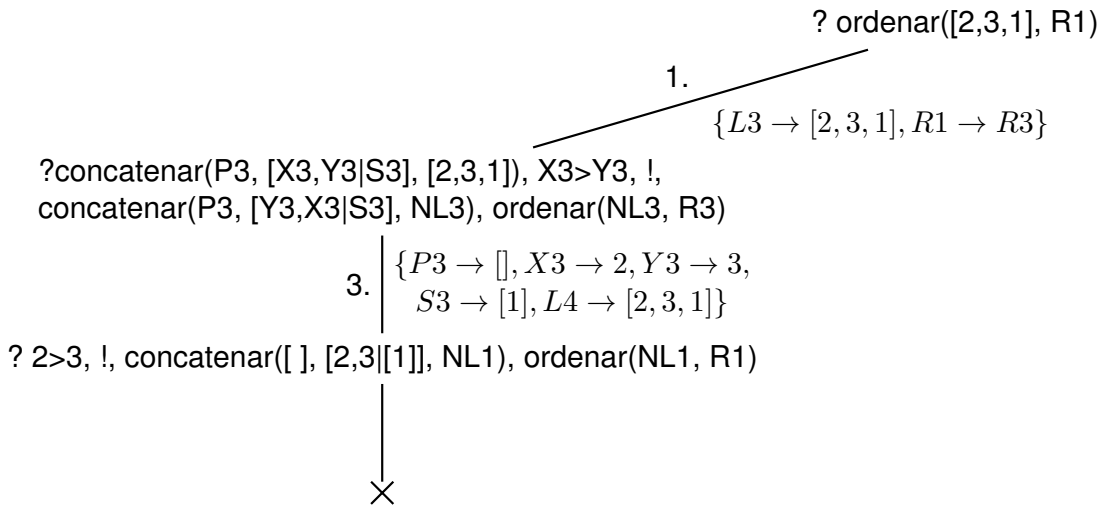
?concatenar(P3, [X3,Y3|S3], [2,3,1]), X3>Y3, !,
concatenar(P3, [Y3,X3|S3], NL3), ordenar(NL3, R3)

3. $\{P3 \to [], X3 \to 2, Y3 \to 3, S3 \to [1], L4 \to [2,3,1]\}$

? 2>3, !, concatenar([ ], [2,3|[1]], NL1), ordenar(NL1, R1)

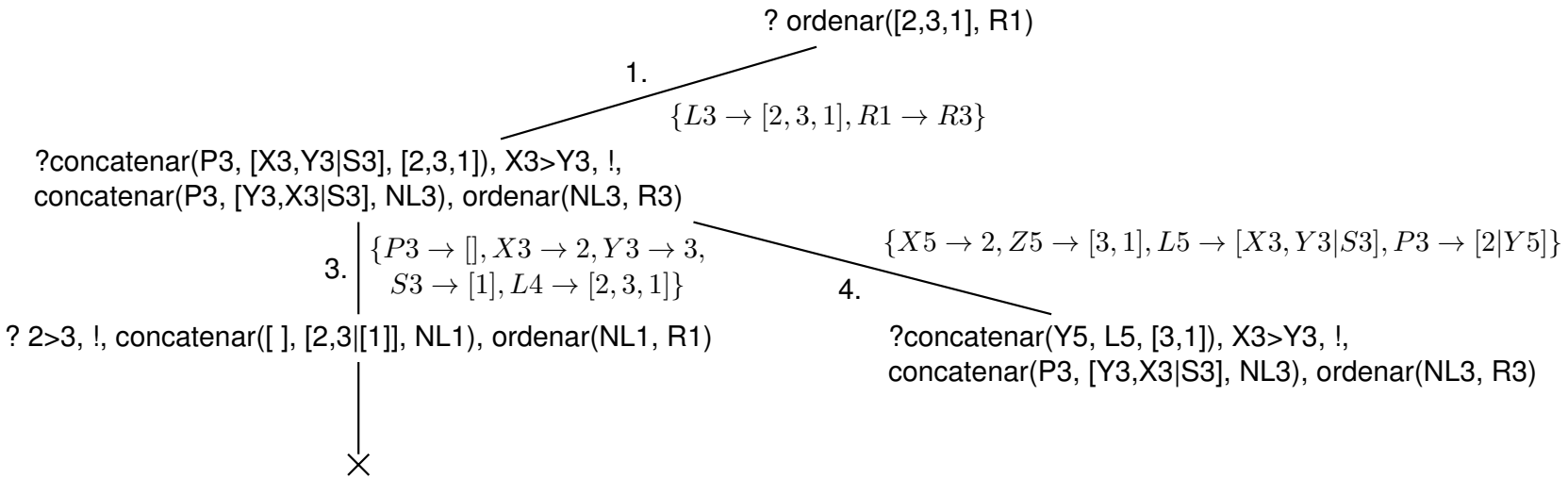
Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



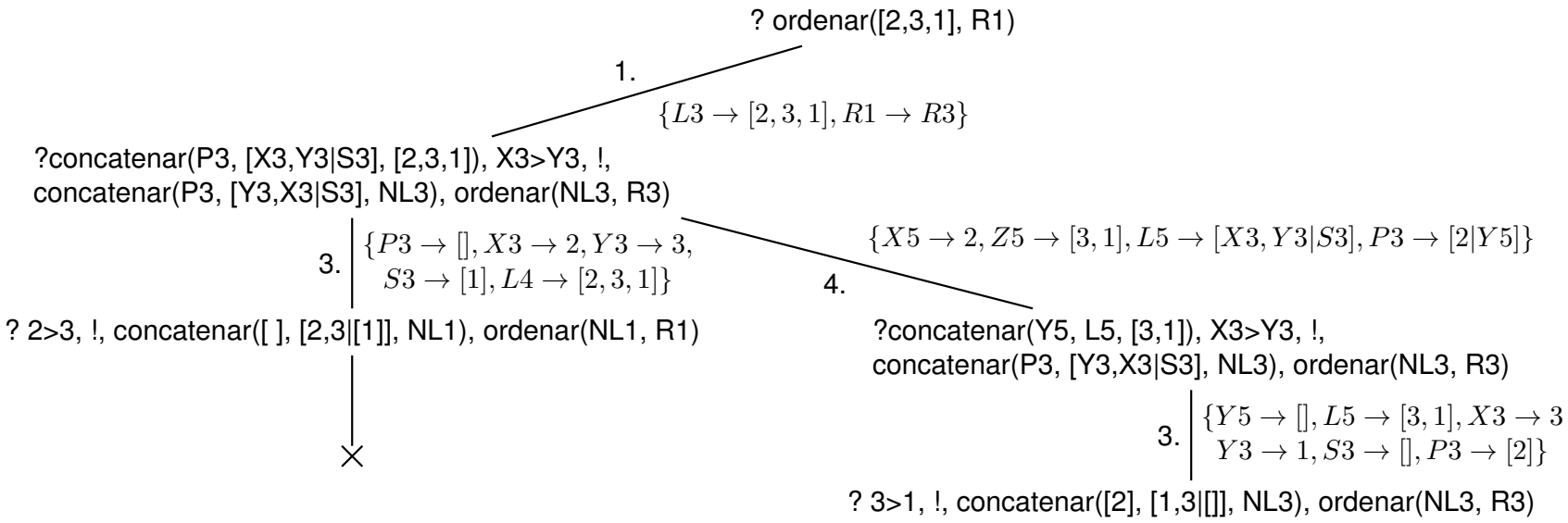
Corte verde: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L):-ordenada(L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



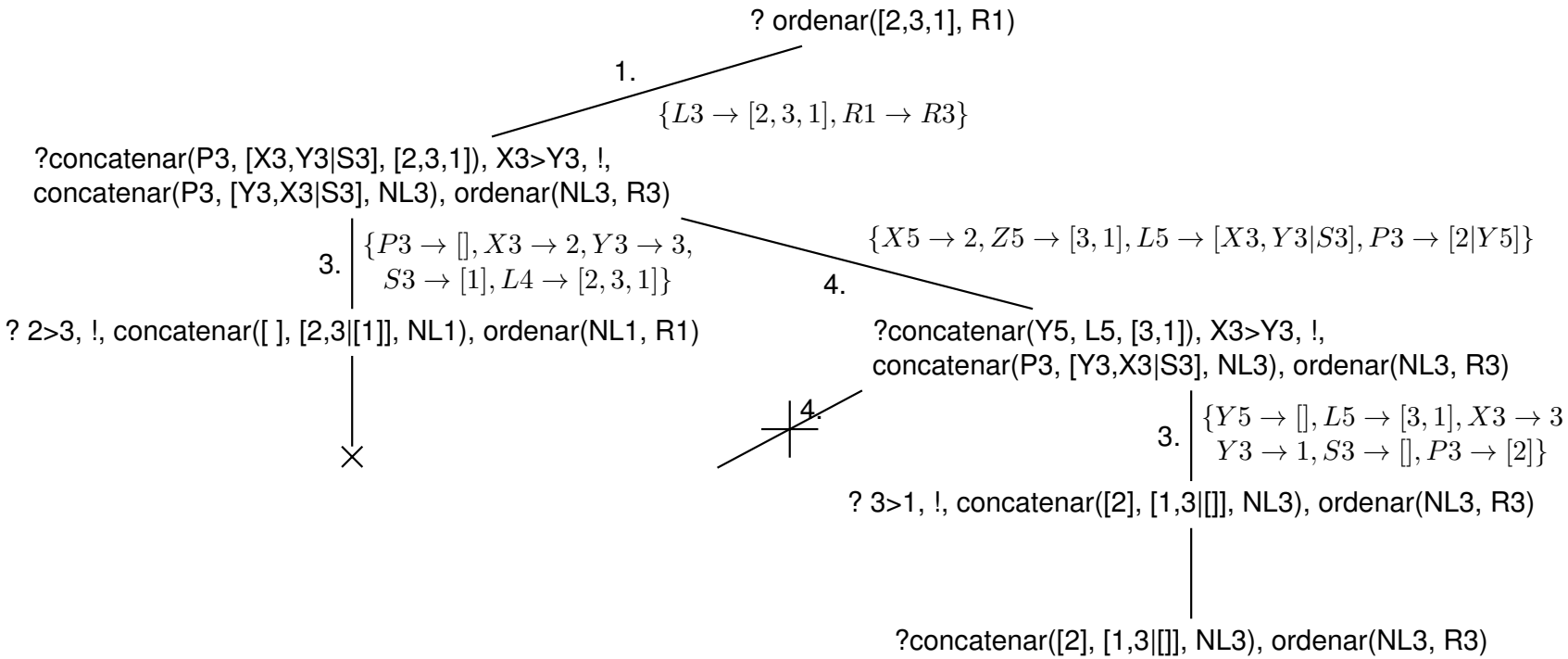
Corte verde: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L):-ordenada(L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



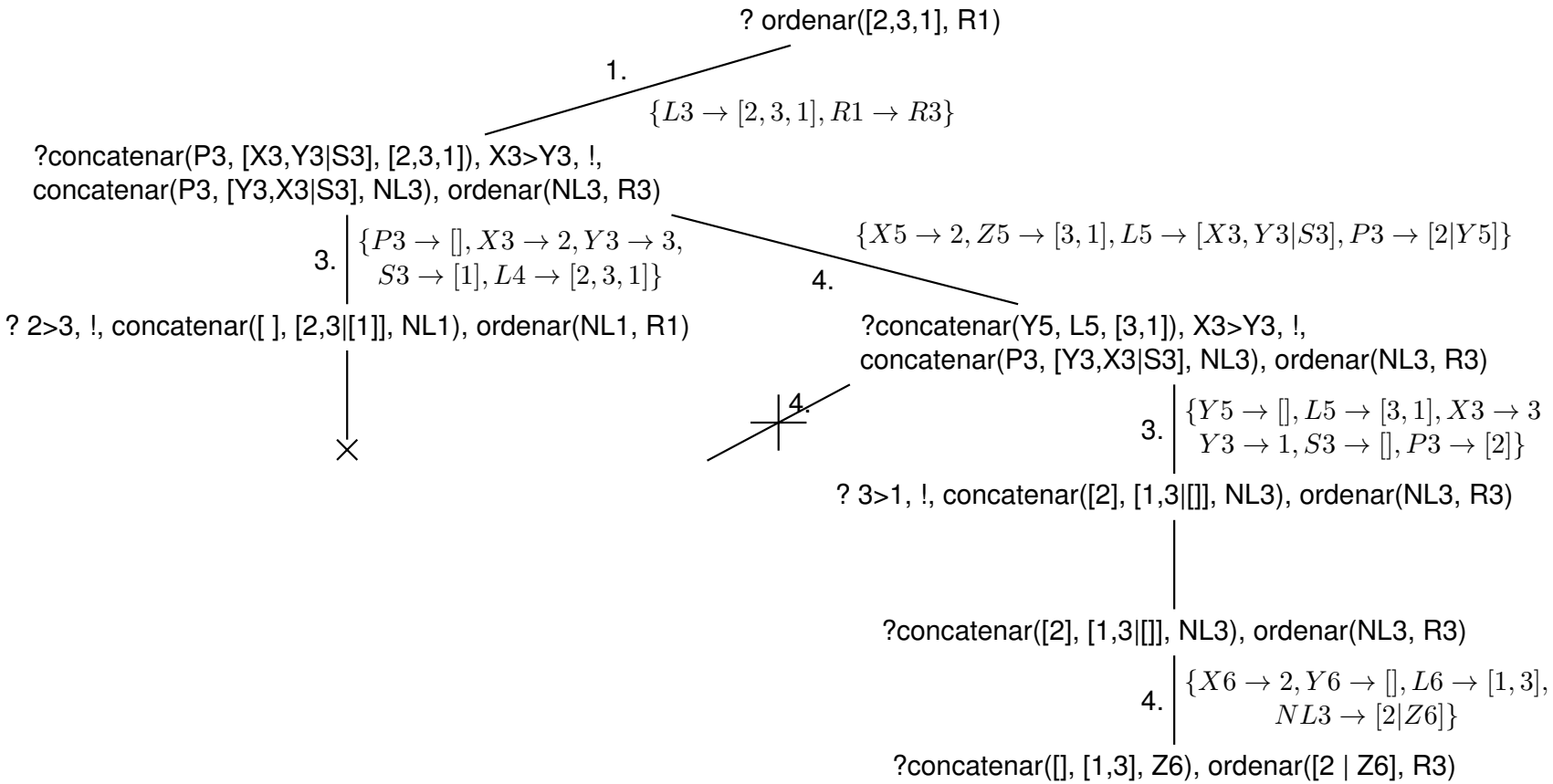
Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



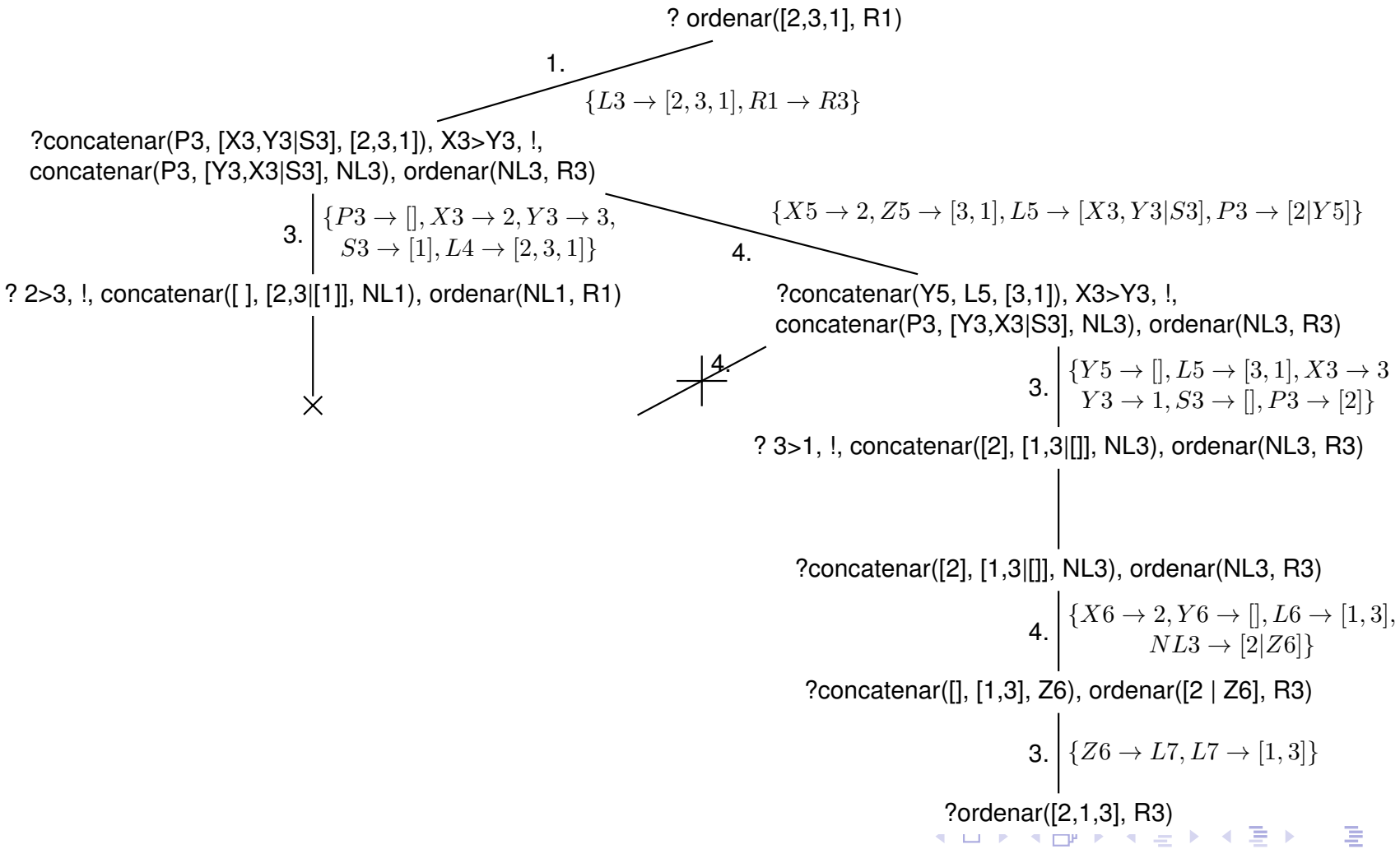
Corte verde: ordenar(+L, −N)

```
1  ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2  ordenar(L,L):-ordenada(L).
3  concatenar([ ],L,L).
4  concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



Corte verde: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L):-ordenada(L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```



Programación III
Presentación de Cátedra – Parte 2

Falla: `fail`

- Predicado que siempre produce fallo
- Útil para detectar casos explícitos que invalidan predicado

```
nand7(0,_,_,_,_,_,_).
nand7(_,0,_,_,_,_,_).
nand7(_,_,0,_,_,_,_).
nand7(_,_,_,0,_,_,_).
nand7(_,_,_,_,0,_,_).
nand7(_,_,_,_,_,0,_).
nand7(_,_,_,_,_,_,0).
```

- `nand` es falsa cuando todas sus entradas son ciertas

Falla: `fail`

- Predicado que siempre produce fallo
- Útil para detectar casos explícitos que invalidan predicado

```
nand7(0,_,_,_,_,_,_).
nand7(_,0,_,_,_,_,_).
nand7(_,_,0,_,_,_,_).
nand7(_,_,_,0,_,_,_).
nand7(_,_,_,_,0,_,_).
nand7(_,_,_,_,_,0,_).
nand7(_,_,_,_,_,_,0).
```

- `nand` es falsa cuando todas sus entradas son ciertas

¿Cómo especificar usando `fail` y !?

Falla: `fail`

- Predicado que siempre produce fallo
- Útil para detectar casos explícitos que invalidan predicado

```
nand7(0,_,_,_,_,_,_).
nand7(_,0,_,_,_,_,_).
nand7(_,_,0,_,_,_,_).
nand7(_,_,_,0,_,_,_).
nand7(_,_,_,_,0,_,_).
nand7(_,_,_,_,_,0,_).
nand7(_,_,_,_,_,_,0).
```

- `nand` es falsa cuando todas sus entradas son ciertas

¿Cómo especificar usando `fail` y !?

```
nand7(1,1,1,1,1,1,1):-!,fail.
nand7(_,_,_,_,_,_,_).
```

Programación III
Presentación de Cátedra – Parte 2

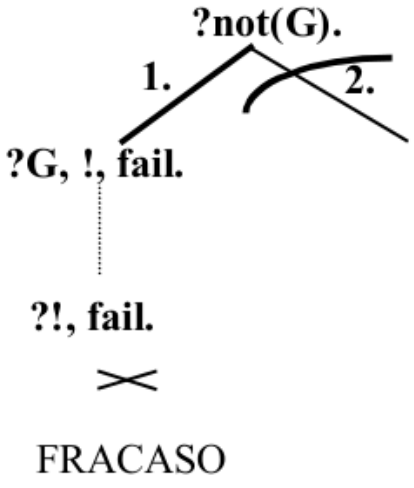
Negación

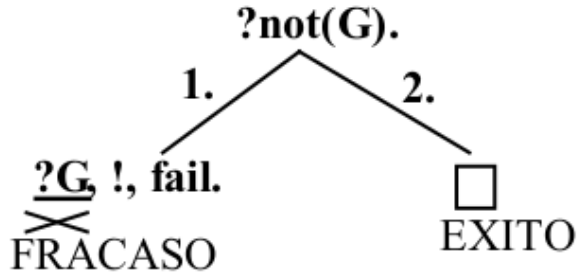
```
not(X):- X,!,fail.
not(X).
```

- Forma débil de Negación
- No es la negación lógica
- Implementado como negación por fallo
- `X` unificará solo sobre átomos
- Variables o átomos con variables será siempre `No`

Árbol de búsqueda

```
not(X):- X,!,fail.
not(X).
```





Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
?- millonario(X).
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
?- millonario(X).
X = flora  ;
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
?- millonario(X).
X = flora  ;
X = antonio
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

?- millonario(X).

X = flora ;

X = antonio

?- pobre(X).

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
?- millonario(X).
X = flora  ;
X = antonio
?- pobre(X).
No
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
persona_interesante(X):- not padre_de(X,_),millonario(X).
```

?- persona_interesante(X).

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
persona_interesante(X):- not padre_de(X,_),millonario(X).
```

```
?- persona_interesante(X).
No
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
persona_interesante(X):- not padre_de(X,_),millonario(X).
```

```
?- persona_interesante(X).
No
?- persona_interesante(flora).
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
persona_interesante(X):- not padre_de(X,_),millonario(X).
```

```
?- persona_interesante(X).
No
?- persona_interesante(flora).
Yes
```

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
persona_interesante(X):- millonario(X),not padre_de(X,_).
```

?- persona_interesante(X).

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
persona_interesante(X):- millonario(X),not padre_de(X,_).
```

?- persona_interesante(X).

X = flora

Ejemplos

```
saldo_cuenta(maria,1000).
saldo_cuenta(flora,3000000).
saldo_cuenta(antonio,2000000).
padre_de(antonio,maria).
millonario(X):- saldo_cuenta(X,Y), Y > 1000000.
pobre(X):- not millonario(X).
```

```
pudiente(X):- not pobre(X).
```

No equivale a millonario

Programación III
Presentación de Cátedra – Parte 2

Corte rojo: caracterización

- Afecta la semántica declarativa
- El orden de las sentencias se vuelve esencial
- Evita comprobación explícita que se conoce cierta asumiendo un modo de uso
- Otros modos de uso provocan incorrecciones
- Ejemplos: `nand7`

Corte rojo: `if – then – else`

```
a:- b, c.
a:- not(b), d.
```

Corte rojo: `if – then – else`

```
a:- b, c.
a:- not(b), d.
```

¿Cómo puedo usar corte para hacer más eficiente?

Corte rojo: `if – then – else`

```
a:- b, c.
a:- not(b), d.
```

¿Cómo puedo usar corte para hacer más eficiente?

```
a:- b,!,c.
a:- d.
```

Corte rojo: ordenar(+L, −N)

```
1 ordenar(L,R):-concatenar(P,[X,Y|S],L),X>Y,!,concatenar(P,[Y,X|S],NL),ordenar(NL,R).
2 ordenar(L,L).
3 concatenar([ ],L,L).
4 concatenar([X|Y],L,[X|Z]):-concatenar(Y, L, Z).
```

- Se suprime ordenada(L) de la sentencia 2
- Se asume que la sentencia 1 evalúa hasta ordenar la lista
- La sentencia 2 por sí sola dice que todo lista es ordenada
- El orden de las sentencias se volvió esencial

Programación III
Presentación de Cátedra – Parte 2

Mecanismo

- Generar de forma combinatoria soluciones posibles
- Comprobar si son correctas

Longitudes ordenadas de triángulo

- Dado un triángulo, ordenar sus lados por longitud

```
lado(a,5).
lado(b,3).
lado(c,4).
```

Longitudes ordenadas de triángulo

- Dado un triángulo, ordenar sus lados por longitud

```
lado(a,5).
lado(b,3).
lado(c,4).
```

- Solución determinista

```
ordena2(X,Y,X1,Y1):- X<Y, X1 is X, Y1 is Y.
ordena2(X,Y,X1,Y1):- X>=Y, X1 is Y, Y1 is X.
ordena3(X,Y,Z):- lado(a,X1),lado(b,Y1),lado(c,Z1),
  ordena2(X1,Y1,X2,Y2),
  ordena2(X2,Z1,X,Z2),
  ordena2(Y2,Z2,Y,Z).
```

Longitudes ordenadas de triángulo

- Dado un triángulo, ordenar sus lados por longitud

```
lado(a,5).
lado(b,3).
lado(c,4).
```

- Solución determinista

```
ordena2(X,Y,X1,Y1):- X<Y, X1 is X, Y1 is Y.
ordena2(X,Y,X1,Y1):- X>=Y, X1 is Y, Y1 is X.
ordena3(X,Y,Z):- lado(a,X1),lado(b,Y1),lado(c,Z1),
  ordena2(X1,Y1,X2,Y2),
  ordena2(X2,Z1,X,Z2),
  ordena2(Y2,Z2,Y,Z).
```

- Solución no determinista

```
ordena3(X,Y,Z):- lado(Xn,X),lado(Yn,Y),lado(Zn,Z),
  Xn\=Yn,Xn\=Zn,Yn\=Zn,X=<Y,Y=<Z.
```

Ordenar lista: ordenar(+X, −Y)

```
ordenar(X,Y):- permutar(X,Y),ordenada(Y),!.
```