Tema 6: De la Resolución a la programación lógica: Prolog

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Lógica Informática
(Tecnologías Informáticas)

Contenido

De Cláusulas a Reglas Prolog: lógica (I)

- Cláusulas de Horn: a lo sumo tiene un literal positivo:
  - **Hechos**: $\{p\}$, $\{q\}$, ...
  - **Reglas**: $\{\neg p, \neg q, r\}$ (equivalente a $p \wedge q \rightarrow r$. También se escribe $r \leftarrow p \wedge q$)
  - **Preguntas**: $\{\neg r\}$, $\{\neg p, \neg r\}$, ...
- Prolog (del francés, PROgrammation en LOGique).
- En los **programas Prolog sólo se usan los dos primeros tipos**;

```
p.
q.
r :- p,q.
```

De Cláusulas a Reglas Prolog: lógica (II)

- El tercero sirve **para preguntar** lo que hay que *deducir*; `:-r.`
- Un predicado en un programa Prolog está **definido** si está en la parte izquierda de una regla o es un hecho.
- Idea: las cláusulas que tienen a la izquierda un predicado P funcionan como una *definición*.
- Un programa Prolog **siempre es consistente** (basta asignar 1 al literal positivo).
- Si no se añaden preguntas (cláusulas negativas) no se deduce nada (no se *ejecuta*, pues no existe refutación; es consistente).

Recordatorio e ideas

- En cláusulas de Horn: resolución positiva, negativa, unitaria y por entradas **son completas**

  (realmente, son iguales)
- Si nos **preguntamos** si r es consecuencia lógica usamos:

  $\{\{p\},\{q\},\{\neg p,\neg q,r\}\} \models \{r\}$

  $\Updownarrow$

  $\{\{p\},\{q\},\{\neg p,\neg q,r\},\{\neg r\}\}$ inconsistente
- **En Prolog**:
  - El programa sería:

    ```
    p.
    q.
    r :- p,q.
    ```
  - y la pregunta *se niega* para buscar la incosistencia:

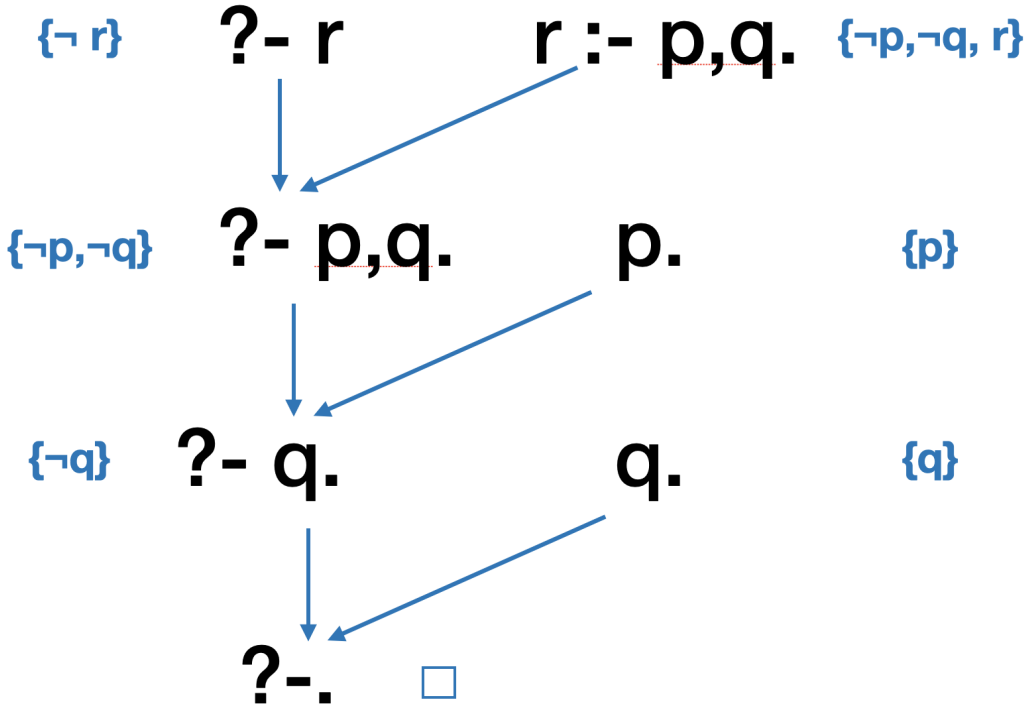
    ```
    :- r.
    ```

Ejecución como deducción

- La pregunta es **la negación que se añade al programa para buscar la refutación**:

  `?- r.`

- Una refutación descrita en sintaxis Prolog y clausal:



- **Cuestión**: ¿Cómo encontrar una refutación (si existe)?

Deducción Prolog en LP: Ejemplo

Consideremos una base de conocimiento sobre clasificación de animales y hechos sobre características de un animal:

- Regla 1: Si un animal es ungulado y tiene rayas negras, entonces es una cebra.
- Regla 2: Si un animal rumia y es mamífero, entonces es ungulado.
- Regla 3: Si un animal es mamífero y tiene pezuñas, entonces es ungulado.
- Hecho 1: El animal tiene es mamífero.
- Hecho 2: El animal tiene pezuñas.
- Hecho 3: El animal tiene rayas negras.

Demostrar a partir de la base de conocimientos que **el animal es una cebra**.

Representación en PROLOG

```
es_cebra    :- es_ungulado, tiene_rayas_negras.  % Regla 1
es_ungulado :- rumia, es_mamifero.               % Regla 2
es_ungulado :- es_mamifero, tiene_pezunas.       % Regla 3
es_mamifero.                                     % Hecho 1
tiene_pezunas.                                   % Hecho 2
tiene_rayas_negras.                              % Hecho 3
```

- ¿Cuáles de los predicados están definidos y cuáles no?
- Si no está definido, Prolog tiene un problema con él porque no puede usarlo en las computaciones (ya lo veremos).

Una demostración *hacia atrás*

Una forma de demostrarlo es **razonando hacia atrás**:
El problema inicial consiste en demostrar que

el animal es una cebra

Por la Regla 1, el problema se reduce a demostrar que

el animal es ungulado y tiene rayas negras

Por la Regla 3, el problema se reduce a demostrar que

el animal es mamífero, tiene pezuñas y rayas negras

Por el Hecho 1, el problema se reduce a demostrar que

el animal tiene pezuñas y tiene rayas negras

Por el Hecho 2, el problema se reduce a demostrar que

el animal tiene rayas negras

Que es cierto por el Hecho 3.

Sesión de Prolog

La base de conocimiento se carga en la sesión Prolog escribiendo el nombre entre corchetes y terminado en un punto. La pregunta se plantea escribiendo el átomo y un punto.

```
> pl
Welcome to SWI-Prolog (Version 5.0.3)
Copyright (c) 1990-2002 University of Amsterdam.

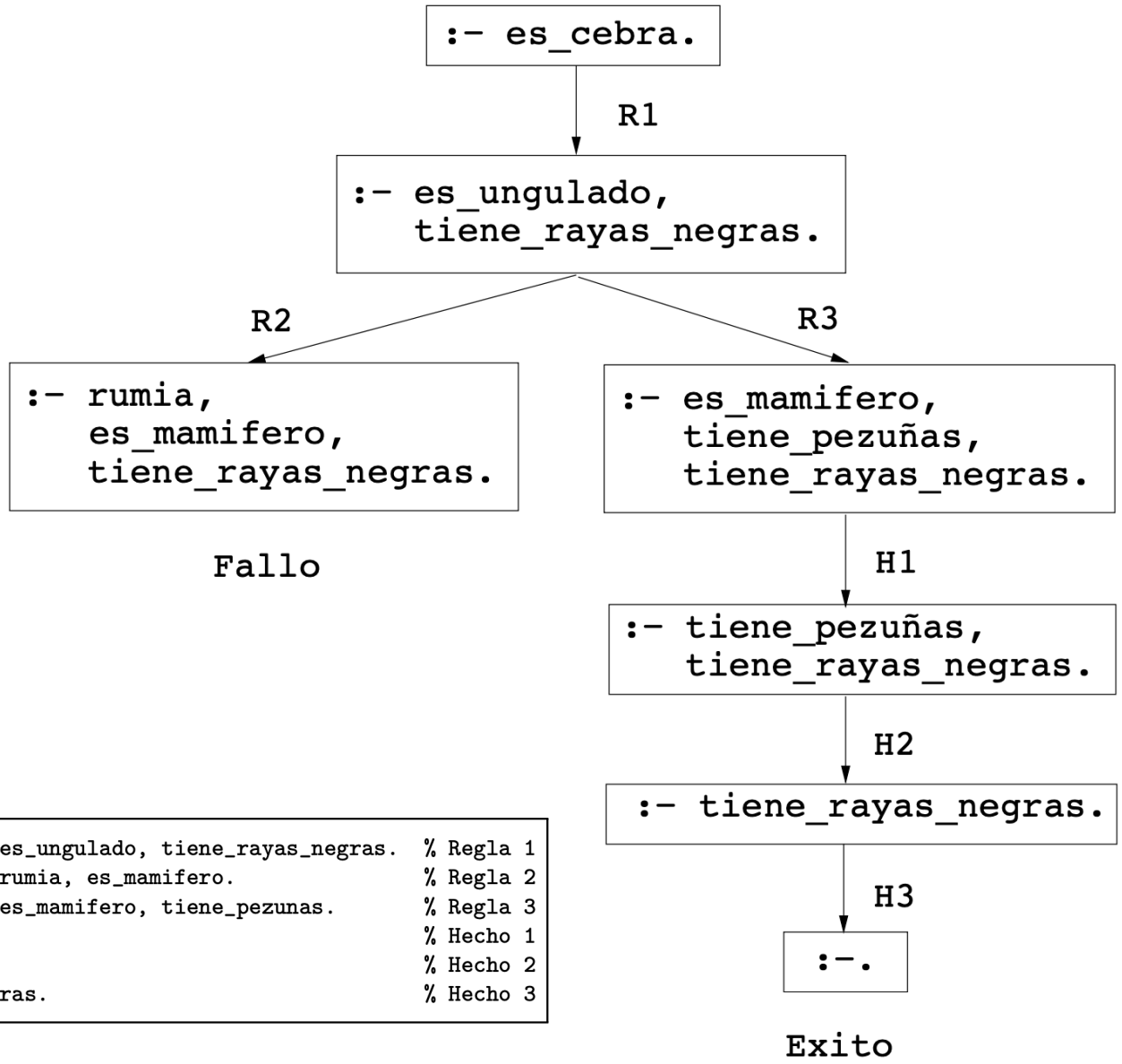
?- [animales].        %Carga el programa
Yes

?- es_cebra.
Yes
```

La respuesta "Yes" significa que **ha demostrado** que el animal es una cebra.

Árbol de deducción del programa

Veamos cómo obtiene la demostración mediante el **árbol de deducción**:




```
es_cebra     :- es_ungulado, tiene_rayas_negras.   % Regla 1
es_ungulado  :- rumia, es_mamifero.                % Regla 2
es_ungulado  :- es_mamifero, tiene_pezunas.        % Regla 3
es_mamifero.                                       % Hecho 1
tiene_pezunas.                                     % Hecho 2
tiene_rayas_negras.                                % Hecho 3
```

¿Cómo se construye?

Búsqueda en profundidad en un espacio de estados (cada estado es una pila de problemas por resolver). En nuestro ejemplo:

- el estado inicial consta de un único problema **" es_cebra"**.
- Buscamos cláusula cuya cabeza coincida con el primer problema de la pila, encontrando sólo la regla 1. **Sustituimos el problema por el cuerpo de la regla**, dando lugar a la pila **" es_ungulado, tiene_rayas_negras"**
- Para el primer problema tenemos 2 reglas cuyas cabezas coinciden (R2,R3).
- Consideramos en primer lugar la regla 2, produciendo la pila de problemas **"rumia, es_mamífero, tiene_rayas_negras"**
- El primer problema no coincide con la cabeza de ninguna cláusula: **fallo** y se reconsidera la elección anterior.
- Consideramos ahora la cláusula 3, produciendo la pila **" es_mamífero, tiene_pezuñas, tiene_rayas_negras"**
- Cada uno de los problemas restantes coincide con uno de los hechos, con lo que **obtenemos una solución del problema.**

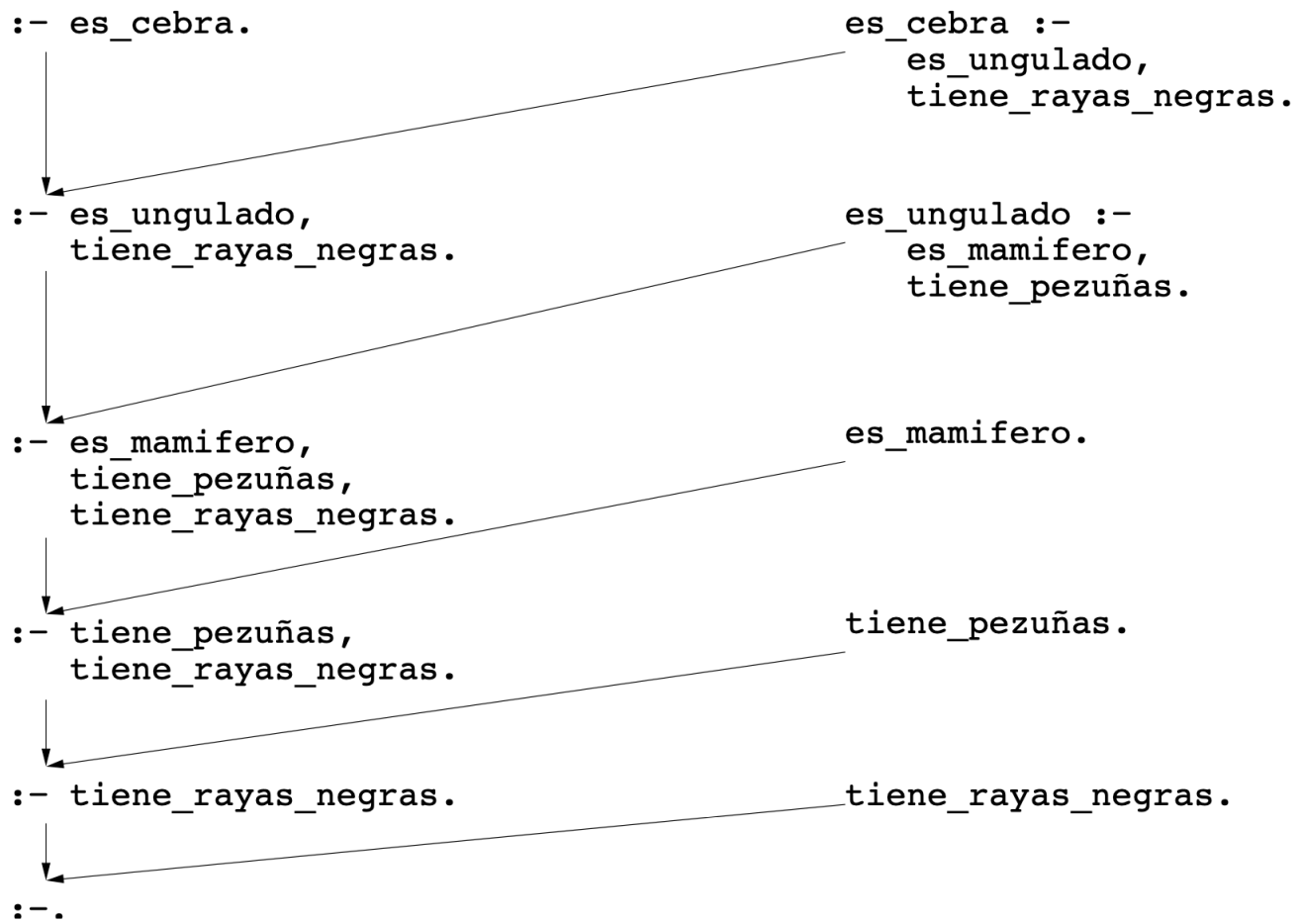
Interpretación de **Fallo**

- Sea S el conjunto de cláusulas correspondientes al programa Π, y P la pregunta.
- **Fallo**$\equiv$**No demostrable**; es decir, $S \not\models \neg P$.
- **Fallo**$\not\equiv$**Falso**; es decir, **no necesariamente** $S \models P$.
- Objetivo del programador: que **Fallo** sea equivalente en mi programa a **Falso**, con la idea:
- *Si unos datos no satisfacen las reglas que definen un predicado, entonces no lo satisface*.
- Así nos aseguraríamos el objetivo ideal de que

  $S \models p(a1, \ldots, an)$ **si y sólo sí** Π **responde sí a la pregunta** `?- p(a1,...,an)`

Extracción de la demostración

- Hay una rama de fallo (hoja no vacía y primer átomo no coincide con la cabeza de ninguna regla) y rama de éxito (su hoja es vacía). Con la rama éxito se extrae la demostración (**resolución SLD**):



Como demostración...

1	"tiene_rayas_negras"	Hecho 3
2	"tiene_pezuñas"	Hecho 2
3	" es_mamífero"	Hecho 1
4	" es_ungulado"	Regla 3 y líneas 3 y 2
5	" es_cebra"	Regla 1 y líneas 4 y 1

Véamoslo usando Prolog (en una implementación online)
https://swish.swi-prolog.org/
Los predicados no definidos los lleva a error, en el programa anterior hay que añadir la línea:

```
:- dynamic(rumia/0).
```

Importante: En la versión 5.0 de SWI Prolog, para que no provoque error en los predicados no definidos, hay que añadirle la línea

```
:- set_prolog_flag(unknown,warning).
```

Lógica relacional

Lógica relacional: variables, cuantificadores, constantes y predicados

- **Ejemplo**: Representemos la siguiente información:
  - Hechos: 6 y 12 son divisibles por 2 y por 3.
  - Hecho: 4 es divisible por 2.
  - Regla: Los números divisibles por 2 y por 3 son divisibles por 6.
- Elegimos por ejemplo el lenguaje:
  - Constantes: "2", "3", "6" y "12"
  - Predicado binario: "divide(X,Y)" (X divide a Y)
- Para representarlo en Prolog:
  - Los hechos: cuatro cláusulas unitarias
  - Regla: *para todo X: si X es divisible por 2 y X es divisible por 3, entonces X es divisible por 6*, $\forall X[\text{divide}(2, X) \wedge \text{divide}(3, X) \rightarrow \text{divide}(6, X)]$
- En Prolog (**teniendo en cuenta que en Prolog las variables comienzan por mayúscula**):

```
divide(6,X) :- divide(2,X), divide(3,X).
```

Programa Prolog del ejemplo

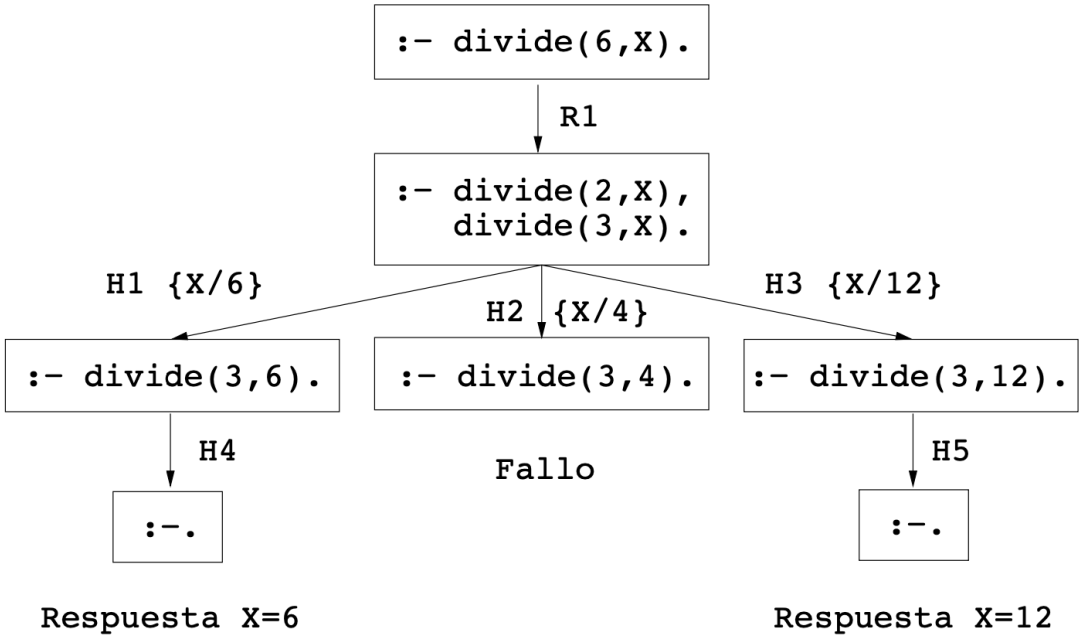
```
divide(2,6).                                  % Hecho 1
divide(2,4).                                  % Hecho 2
divide(2,12).                                 % Hecho 3
divide(3,6).                                  % Hecho 4
divide(3,12).                                 % Hecho 5
divide(6,X) :- divide(2,X), divide(3,X). % Regla 1
```

Ejecutando...

```
>?- divide(6,X).
X = 6 ;
X = 12 ;
No
```

Después de obtener la primera respuesta se determina otra pulsando punto y coma. Cuando se intenta buscar otra responde `No`. Significa que **no hay más respuestas**.

Árbol de deducción



- **No se exige** que el primer objetivo sea igual que la cabeza de una cláusula, sino que sea **unificable**

  Ejemplo: en el segundo paso, `divide(2,X)` **se unifica con H1**, `divide(2,6)` mediante la sustitución $\{X/6\}$
- Componiendo las sustituciones usadas en una rama de éxito se obtiene la respuesta.

Deducción Prolog en lógica funcional

Lógica funcional, en Lógica de Primer Orden

- Trabajamos en el LPO con constante `0` y un símbolo de función unitaria `s(.)` (el cero y la función sucesor).

  $$0, s(0), s(s(0)), \ldots$$

- Elegimos `suma(X,Y,Z)` para representar: "Z" es la suma de los números naturales "X" e "Y".
- En LPO se *define* la suma de números naturales con:

```
0 + Y = Y
s(X) + Y = s(X+Y)
```

- es decir,

  $(\forall Y)[\text{suma}(0, Y, Y)]$
  $(\forall X, Y, Z)[\text{suma}(X, Y, Z) \rightarrow \text{suma}(s(X), Y, s(Z))]$

- en Prolog:

```
suma(0,Y,Y).                          % R1
suma(s(X),Y,s(Z)) :- suma(X,Y,Z).     % R2
```

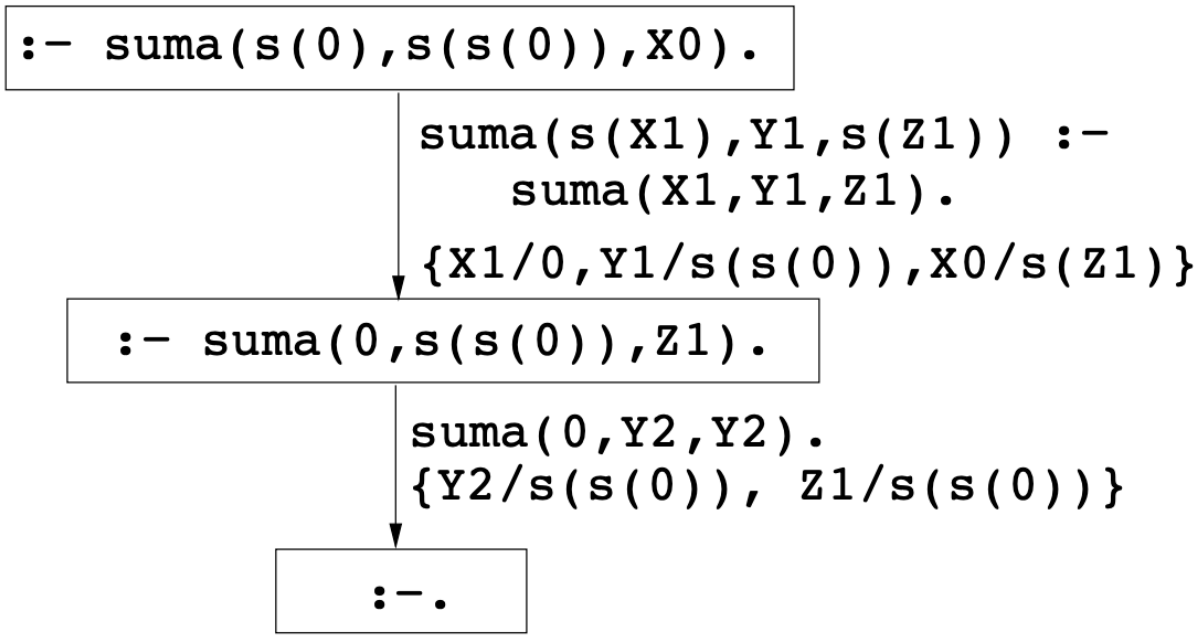
Ejecución del programa

- ¿Cuánto es 1+2?
- Representados con `s(0)` y `s(s(0))`, respectivamente
- En Prolog:

```
> ?- suma(s(0),s(s(0)),X).
> X = s(s(s(0)))
> Yes
```

Ejecución del programa

- Árbol de deducción (recuerda: variables separadas):



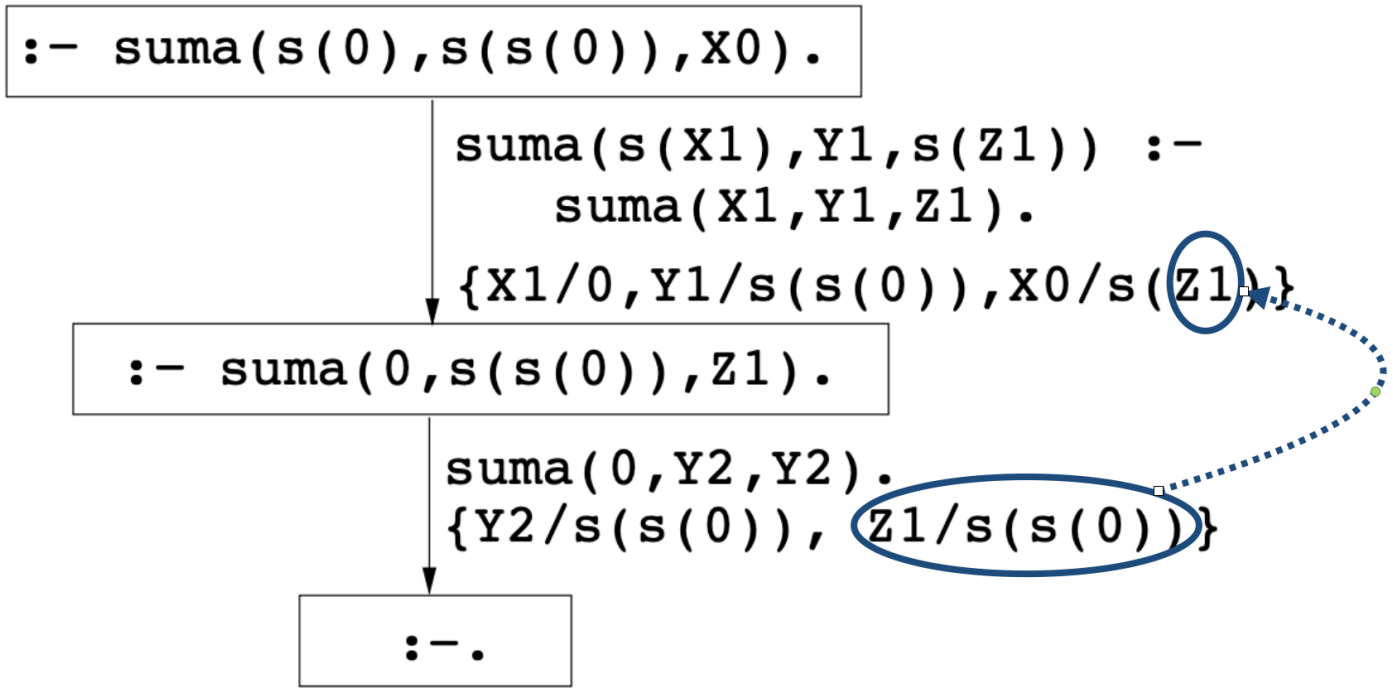
Resp.: X = X0 = s(Z1) = s(s(s(0)))

```
suma(0,Y,Y).                          % R1
suma(s(X),Y,s(Z)) :- suma(X,Y,Z).     % R2
```

Explicación

- El nodo inicial sólo tiene un sucesor con R2, porque es la cabeza de la única regla con la que unifica:
  - El átomo `suma(s(0),s(s(0)),X0)` **no es unificable** con `suma(0,Y1,Y1)`
  - **Es unificable** con "suma(s(X1),Y1,s(Z1)" mediante la sustitución "X1/0, Y1/s(s(0)), X0/s(Z1)"
- Lo mismo sucede con el segundo nodo.
- Finalmente, la respuesta se calcula **componiendo las sustituciones realizadas en la rama de éxito** a las variables iniciales:
  - "X" se sustituye inicialmente por "X0",
  - en el primer paso se sustituye "X0" por "s(Z1)"
  - en el segundo se sustituye "Z1" por "s(s(0))"
- **Resultado**: "X" es "s(s(s(0)))".

Obtención de respuesta



- Es la composición:

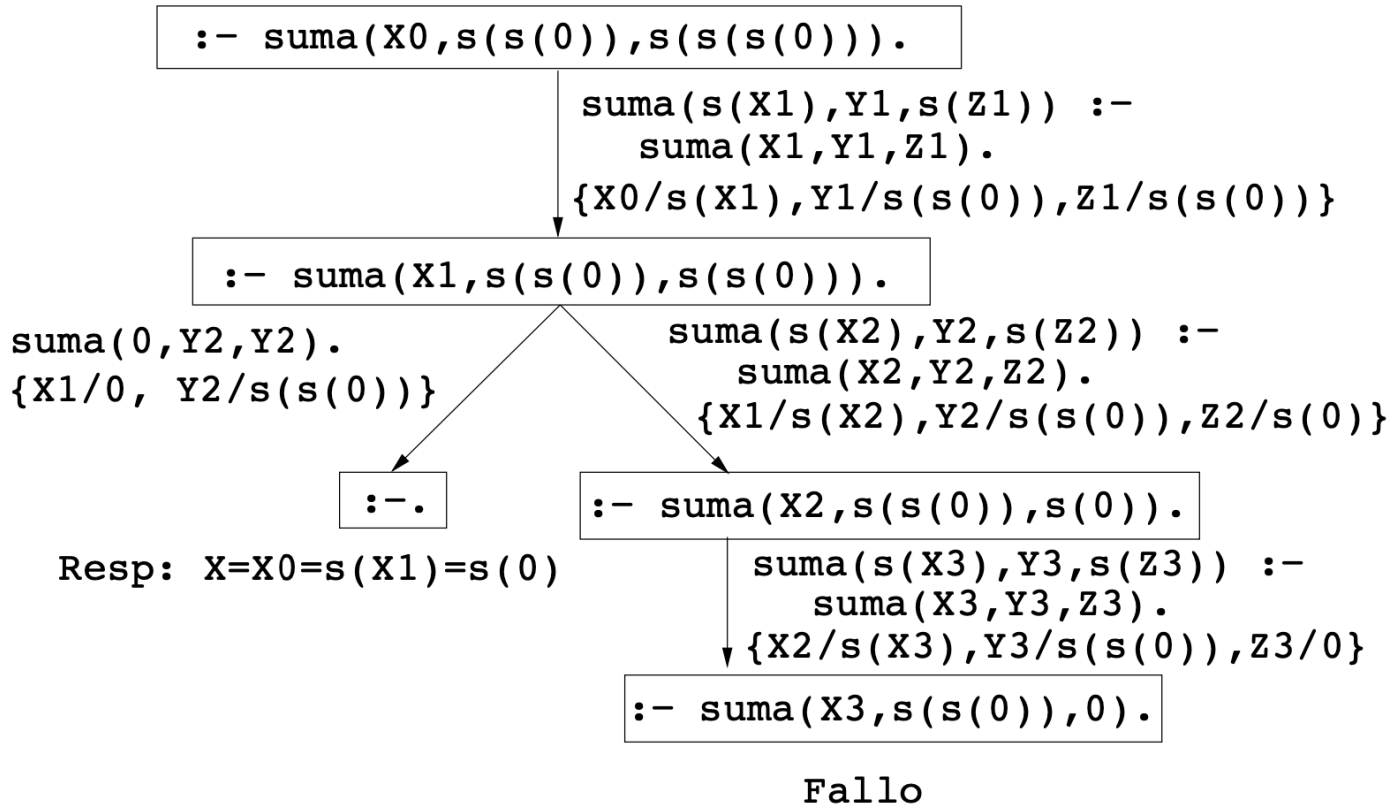
$$\{\mathtt{X1}/0,\ \mathtt{Y1}/\mathtt{s(s(0))},\ \mathtt{X0}/\mathtt{S(Z1)}\} \circ \{\mathtt{Y2}/\mathtt{s(s(0))},\ \mathtt{Z1}/\mathtt{s(s(0))}\} =$$
$$= \{X1/0,\ Y1/s(s(0)),\ \mathbf{X0/s(s(s(0)))},\ Y2/s(s(0)),\ Z1/s(s(0))\}$$

¿Cómo calculamos la resta?

- ¿Cómo calcular $3 - 2$? ("s(s(s(0)))" menos "s(s(0))")?
- **No es necesario hacer un nuevo programa**, pues

$$X = a - b \leftrightarrow X + a = b$$

- ```
  ?- suma(X,s(s(0)),s(s(s(0)))).
  X = s(0) ;
  No
  ```


  ```

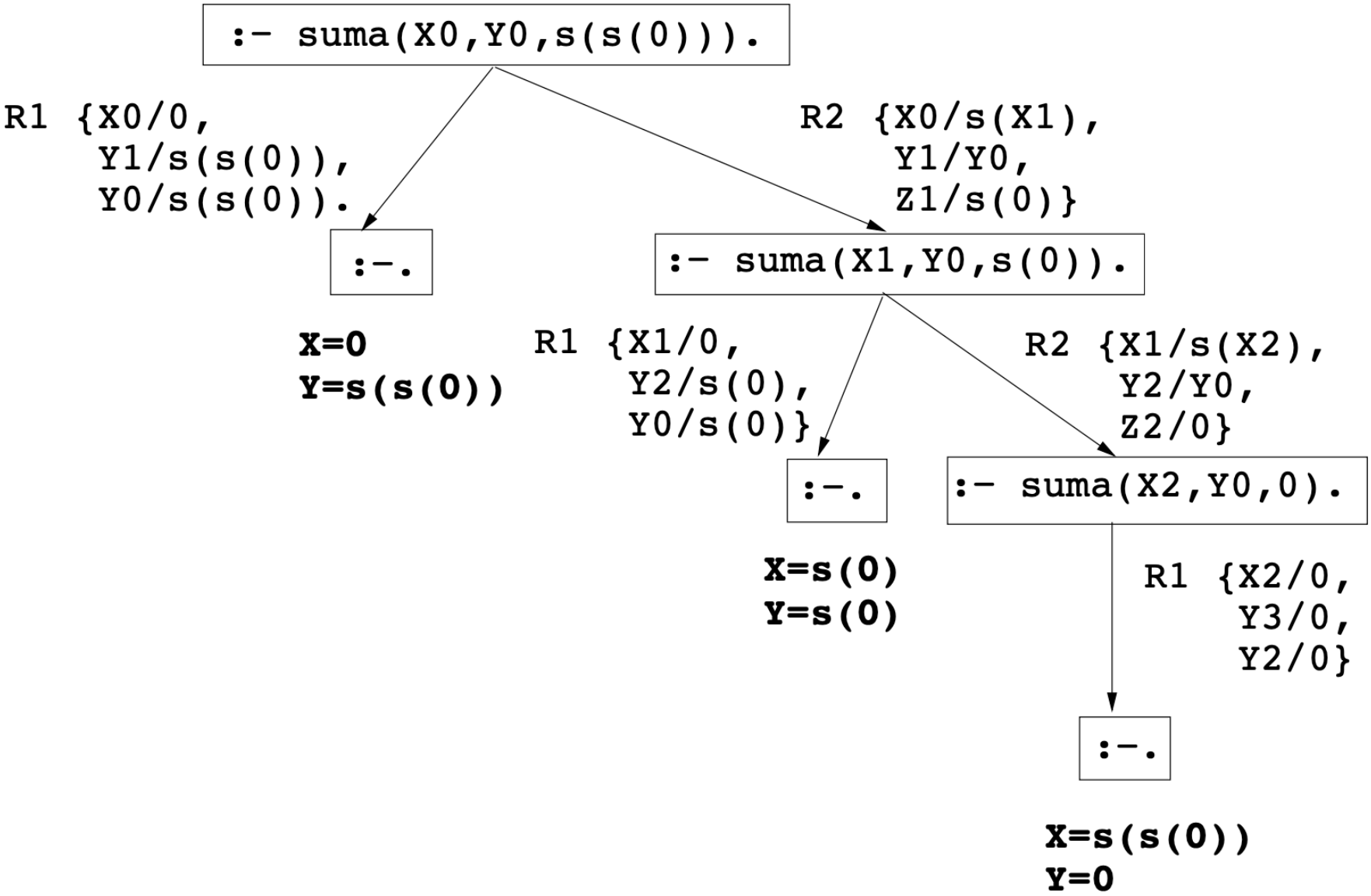
¿De cuántas formas podemos descomponer un número?

- Problema: descomponer el número 2 en suma de dos números naturales; es decir resolver la ecuación "X + Y = 2"
- **No hace falta otro programa**:

```
?- suma(X,Y,s(s(0))).
X = 0
Y = s(s(0)) ;
X = s(0)
Y = s(0) ;
X = s(s(0))
Y = 0 ;
No
```

- Es decir, son las tres soluciones:
  $2 = 0 + 2 = 1 + 1 = 2 + 0$

Árbol de deducción



Resolución de ecuaciones

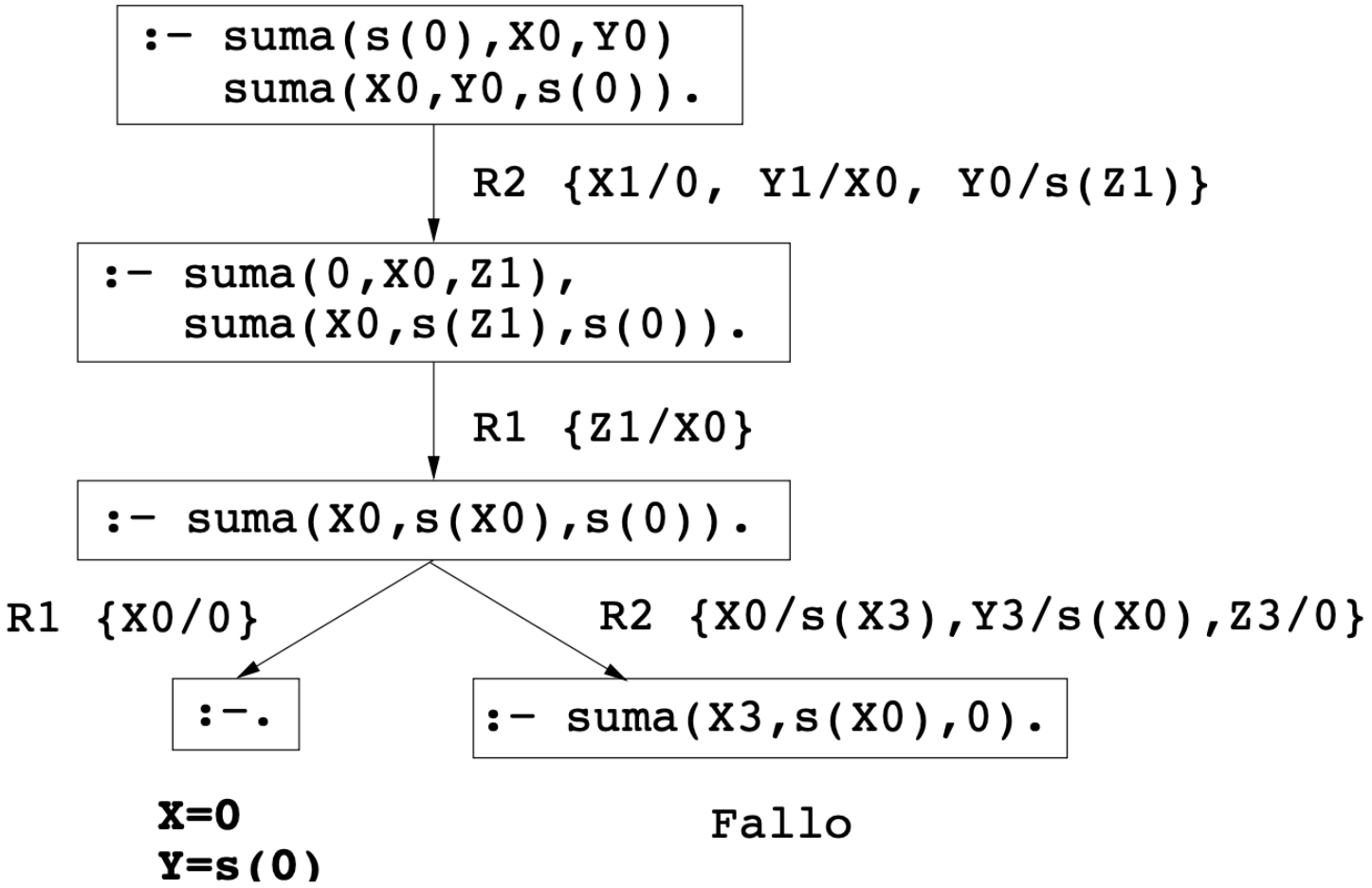
- Deseamos resolver el sistema:

$$1 + X = Y$$
$$X + Y = 1$$

- Que se podría hacer simplemente con el mismo programa, **preguntando las dos cuestiones** (es decir, una cláusula del tipo $\{\neg p, \neg q\}$).

```
?- suma(s(0),X,Y), suma(X,Y,s(0)).
X = 0
Y = s(0) ;
No
```

Árbol de deducción



Se observa que el unificador del primer objetivo y la cabeza de la cláusula se le aplica a los restantes objetivos: en el primer paso se ha sustituido la variable "Y0" del segundo objetivo por "s(Z1)".

Referencias

- J. Antonio Alonso Jiménez, Joaquín Borrego Díaz Deducción automática (Vol. 1: Construcción lógica de sistemas lógicos) `https://idus.us.es/handle/11441/75303`
- SWI Prolog online: `https://swish.swi-prolog.org/example/kb.pl`
- Quien quiera profundizar:
  - I. Bratko. Prolog Programming for Artificial Intelligence. Addison–Wesley, 1986.
  - W. Clocksin and C. Mellish. Programming in Prolog. Springer–Verlag, 4 edition, 1994.
  - E. S. L. Sterling. L'art de Prolog. Masson, 1990 http://cliplab.org/~logalg/doc/The_Art_of_Prolog.pdf
  - T. V. Le. Techniques of Prolog Programming (with implementation of logical negation and quantified goals). John Wiley, 1993