PROGRAMACIÓN DECLARATIVA

PROGRAMACIÓN LÓGICA

Tema PL2: El lenguaje Prolog, aspectos básicos

3. Semántica

Grado en Ingeniería Informática

URJC

Ana Pradera

Contenido

INTRODUCCIÓN

- En lo que sigue se describe la *semántica operacional* de PROLOG, es decir, su mecanismo de cómputo.
- PROLOG implementa un refinamiento del sistema de Demostración Automática conocido como Sistema de Resolución SLD.
- Para comprender su funcionamiento es necesario conocer:
  - El problema de la Unificación y el algoritmo para resolverlo.
  - La Regla de Resolución, que utiliza el Algoritmo de Unificación.
  - Los Árboles de Resolución, que utilizan la Regla de Resolución.

UNIFICACIÓN

Expresión = término (constante, variable o compuesto) o predicado

El problema de la Unificación

Dadas dos expresiones, el problema de la Unificación consiste en determinar si dichas expresiones pueden resultar sintácticamente idénticas mediante la realización de sustituciones adecuadas en sus variables, en cuyo caso se dice que las expresiones son unificables.

Ejemplos

1. $p(f(X), a)$ y $p(f(g(b)), Y)$ son unificables: basta sustituir X por $g(b)$ e Y por a, obteniéndose la expresión $p(f(g(b)), a)$.
2. $p(f(X), a)$ y $p(f(g(b)), c)$ NO son unificables: nunca podrán resultar iguales, al ser a y c dos constantes distintas.

Sustituciones

- Una sustitución es una función que permite asociar términos a variables.
- Las sustituciones se pueden aplicar a expresiones (términos o predicados), dando lugar a nuevas expresiones denominadas *instancias* de las primeras.
- Las sustituciones se pueden componer entre sí dando lugar a nuevas sustituciones.

Definición (Sustitución)

Una sustitución es un conjunto finito de asociaciones entre variables y términos, denotado $\{X_1/t_1, X_2/t_2, \ldots, X_n/t_n\}$, donde:

- $X_1, \ldots, X_n$ son variables diferentes entre sí.
- $t_1, \ldots, t_n$ son términos y se verifica $t_i \neq X_i$ para todo $i \in \{1, \ldots, n\}$.

Notación

- Cuando $n = 0$ se obtiene la *sustitución vacía*, denotada ϵ o $\{\}$.
- El resto de sustituciones se denotan σ, τ, θ, etc.

Ejemplos

- $\sigma = \{X/a, Y/f(a,b), Z/g(Y)\}$ y $\tau = \{X/f(a,Z), Y/g(f(a,Y))\}$ son sustituciones.
- $\{\mathbf{X}/a, Y/f(a,b), \mathbf{X}/g(Y)\}$ y $\{X/f(a,Z), Y/g(f(a,Y)), \mathbf{Z}/\mathbf{Z}\}$ no son sustituciones.

Definición (Aplicación de una sustitución a una expresión)

Sea φ una expresión y sea $\sigma = \{X_1/t_1, X_2/t_2, \ldots, X_n/t_n\}$ una sustitución. La aplicación de σ a φ se define como la expresión resultante de sustituir en φ, de forma simultánea, cada aparición de la variable X_i, $1 \leq i \leq n$, por su término asociado en la sustitución σ, t_i. La expresión obtenida, denotada $\varphi\sigma$, se denomina una *instancia* de φ.

Ejemplos

- Si $\sigma = \{X/f(X,U), Y/X, Z/h(b,Y), U/a\}$ y $\varphi = g(X,Y,Z)$, entonces $\varphi\sigma = g(f(X,U), X, h(b,Y))$. ($\varphi\sigma \neq g(f(X,a), X, h(b,Y))$ y $\varphi\sigma \neq g(f(X,U), f(X,U), h(b,Y))$ puesto que *no* se encadenan las sustituciones, se hacen de forma simultánea).
- Si $\sigma = \{X/Y, Y/f(a)\}$ y $\varphi = p(X, f(Y), Z)$, entonces $\varphi\sigma = p(Y, f(f(a)), Z)$. ($\varphi\sigma \neq p(f(a), f(f(a)), Z)$ puesto que *no* se encadenan las sustituciones, se hacen de forma simultánea).

Definición (Composición de dos sustituciones)

Sean $\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$ y $\tau = \{Y_1/s_1, \ldots, Y_m/s_m\}$ dos sustituciones. La composición de σ con τ, denotada $\sigma\tau$, es la sustitución compuesta por:

1. todos los pares $X_i/(t_i\tau)$, $1 \leq i \leq n$, salvo aquellos tales que $t_i\tau = X_i$ ($t_i\tau$ es la ▸ aplicación de τ a t_i).
2. todos los pares Y_j/s_j, $1 \leq j \leq m$, salvo aquellos tales que $Y_j \in \{X_1, \ldots, X_n\}$.

Ejemplo

Sea $\sigma = \{X/g(U), Y/f(Z), Z/Y\}$ y $\tau = \{U/a, X/b, Y/Z, Z/g(X)\}$.

Entonces:

$$\begin{aligned}\sigma\tau &= \{X/g(a), Y/f(g(X)), \mathbf{Z/Z}, U/a, \mathbf{X/b}, \mathbf{Y/Z}, \mathbf{Z/g(X)}\} \\ &= \{X/g(a), Y/f(g(X)), U/a\}\end{aligned}$$

Ejercicios (Aplicación y composición de sustituciones)

1. *Dadas las sustituciones* $\sigma_1 = \{U/a, X/b, Y/Z, Z/g(X)\}$ *y* $\sigma_2 = \{X/g(U), Y/f(Z), Z/Y, V/U\}$*:*
   - *Calcule la composición* $\sigma_1\sigma_2$*.*
   - *Aplique la sustitución resultante a la expresión* $\varphi = p(X, f(Z), Y, V)$*, es decir, calcule* $\varphi\sigma_1\sigma_2$*.*

2. *Dadas las sustituciones* $\sigma_1 = \{X/Z, Z/g(Y, Y), V/Y, T/X\}$ *y* $\sigma_2 = \{Y/V, Z/g(V, V), T/a\}$*:*
   - *Calcule* $\sigma_1\sigma_2$*.*
   - *Calcule* $\sigma_2\sigma_1$*.*
   - *¿Existe alguna sustitución* θ *tal que* $\sigma_1\theta = \sigma_2$*?*
   - *¿Existe alguna sustitución* θ *tal que* $\sigma_2\theta = \sigma_1$*?*

Unificadores

Definición (Unificador)

Sean φ_1 y φ_2 dos expresiones. Si existe una sustitución σ tal que $\varphi_1\sigma = \varphi_2\sigma$, se dice que φ_1 y φ_2 son unificables y que la sustitución σ es un unificador de φ_1, φ_2.

Expresiones	¿Unifican?
a , b	No
a, $f(X)$	No
X , b	Sí
X , $g(a, Y)$	Sí

Expresiones	¿Unifican?
X , $f(X, a)$	No
$p(a)$, $p(f(Y))$	No
$g(a, b)$, $g(U, U)$	No
$p(X, f(Y), Z)$, $p(g(a), f(b), U)$	Sí

Observación

Cuando dos expresiones resultan ser unificables, es posible que la unificación se pueda conseguir mediante varios unificadores distintos.

Ejemplo

Por ejemplo, las expresiones $\varphi_1 = p(f(X), Y)$ y $\varphi_2 = p(f(a), Z)$ son unificables mediante los siguientes unificadores:

- Con $\sigma = \{X/a, Y/a, Z/a\}$, $\varphi_1\sigma = \varphi_2\sigma = p(f(a), a)$.
- Con $\sigma = \{X/a, Y/b, Z/b\}$, $\varphi_1\sigma = \varphi_2\sigma = p(f(a), b)$.
- Con $\sigma = \{X/a, Y/Z\}$, $\varphi_1\sigma = \varphi_2\sigma = p(f(a), Z)$.
- Con $\sigma = \{X/a, Z/Y\}$, $\varphi_1\sigma = \varphi_2\sigma = p(f(a), Y)$.
- ...

¿Cuál de ellos elegir? Uno que sea de máxima generalidad (si hay varios, cualquiera de ellos).

Definición (Unificador de máxima generalidad, u.m.g.)

Sean φ_1, φ_2 dos expresiones unificables y sea σ un unificador de $\{\varphi_1, \varphi_2\}$. Se dice que σ es un unificador de máxima generalidad (u.m.g.) si es *más general* que cualquier otro unificador de $\{\varphi_1, \varphi_2\}$, es decir, si para cualquier otro unificador τ resulta que existe una sustitución θ tal que $\tau = \sigma\theta$.

Ejemplo

Considere el ejemplo anterior:

- Tanto $\{X/a, Y/Z\}$ como $\{X/a, Z/Y\}$ resultan ser unificadores de máxima generalidad (intuitivamente: los más sencillos).
- Los unificadores $\{X/a, Y/a, Z/a\}$ y $\{X/a, Y/b, Z/b\}$ no puede ser u.m.g.'s puesto que no son más generales, por ejemplo, que $\{X/a, Y/Z\}$.

El Algoritmo de Unificación

- El problema de la unificación tiene por tanto un doble objetivo:
  1. Dado un par de expresiones, decidir si son o no unificables.
  2. En caso afirmativo, encontrar un unificador de máxima generalidad, u.m.g. (si hay varios, cualquiera de ellos).
- Existen distintos algoritmos capaces de resolver este problema. A continuación se presenta uno de los más comunes, basado en las ideas originales de Robinson de 1965.
- Se trata de un proceso iterativo que localiza e intenta resolver, mediante la aplicación de sustituciones adecuadas, las discordancias existentes entre las dos expresiones de entrada.
- El funcionamiento del algoritmo se ilustra a continuación con varios ejemplos, utilizando para ello una *tabla* en la que se resume la información relevante en cada uno de los pasos del algoritmo.

ALGORITMO DE UNIFICACIÓN

Entrada: Dos expresiones φ_1 y φ_2.

Salida:

1. Un valor booleano que indica si las expresiones son unificables.
2. En caso afirmativo, un u.m.g. para φ_1, φ_2.

Descripción:

PASO 1: *unificables* := cierto ; $\sigma := \{\}$

PASO 2: MIENTRAS *unificables* = cierto y $\varphi_1\sigma \neq \varphi_2\sigma$ HACER:

2.1. Encontrar el símbolo más a la izqda de $\varphi_1\sigma$ tal que el símbolo correspondiente de $\varphi_2\sigma$ sea distinto (en rojo en los ejemplos).

2.2. Sean s_1 y s_2 los subtérminos de $\varphi_1\sigma, \varphi_2\sigma$ que empiezan por los símbolos elegidos en el paso anterior (subrayados en los ejemplos):

(a) Si ninguno de los dos es una variable o uno es una variable que aparece en el otro (esto último se denomina test de ocurrencia):

unificables := falso.

(b) En otro caso:

- si s_1 es una variable: $\sigma := \sigma\{s_1/s_2\}$
- si s_1 no es una variable: $\sigma := \sigma\{s_2/s_1\}$
- actualizar las expresiones $\varphi_1\sigma$ y $\varphi_2\sigma$, para lo que basta aplicar $\{s_1/s_2\}$ (o $\{s_2/s_1\}$) a las expresiones de la iteración anterior.

(Nota: $\sigma\{s_1/s_2\}$ y $\sigma\{s_2/s_1\}$ son ▸ composiciones de sustituciones)

PASO 3:

DEVOLVER *unificables*.

SI *unificables* = cierto, σ es un u.m.g. de φ_1, φ_2.

Ejemplo

$\varphi_1\sigma$	$\varphi_2\sigma$	s_1	s_2	*unif.*	σ
$p(\underline{X}, X)$	$p(\underline{f(U)}, f(V))$	X	$f(U)$	cierto	$\{X/f(U)\}$
$p(f(U), f(\underline{U}))$	$p(f(U), f(\underline{V}))$	U	V	cierto	$\{X/f(V), U/V\}$
$p(f(V), f(V))$	$p(f(V), f(V))$	FIN			

El algoritmo termina en ese punto porque se llega a $\varphi_1\sigma = \varphi_2\sigma$, concluyendo que las dos expresiones de entrada son unificables con un u.m.g. dado por $\sigma = \{X/f(V), U/V\}$. Observe cómo para obtener el u.m.g. $\{X/f(V), U/V\}$ se ha realizado la *composición* de $\{X/f(U)\}$ con $\{U/V\}$. Otro posible u.m.g. sería $\{X/f(U), V/U\}$ si en la segunda línea se hubiese elegido $\{V/U\}$ en lugar de $\{U/V\}$.

Ejemplo

$\varphi_1\sigma$	$\varphi_2\sigma$	s_1	s_2	*unificables*	σ
$p(\underline{X}, f(Y))$	$p(\underline{Z}, a)$	X	Z	cierto	$\{X/Z\}$
$p(Z, \underline{f(Y)})$	$p(Z, \underline{a})$	$f(Y)$	a	falso	FIN

El Paso 2 termina en el punto **2.2. (a)**, puesto que ninguno de los dos subtérminos s_1 y s_2 es una variable, y concluye que las expresiones de entrada no son unificables.

Ejemplo

$\varphi_1\sigma$	$\varphi_2\sigma$	s_1	s_2	*unif.*	σ
$p(\underline{X}, f(X), X)$	$p(\underline{U}, W, W)$	X	U	cierto	$\{X/U\}$
$p(U, \underline{f(U)}, U)$	$p(U, \underline{W}, W)$	$f(U)$	W	cierto	$\{X/U, W/f(U)\}$
$p(U, f(U), \underline{U})$	$p(U, f(U), \underline{f(U)})$	U	$f(U)$	falso	FIN

El Paso 2 termina en el punto **2.2. (a)**, puesto que aunque el subtérmino s_1 es una variable, s_2 contiene a esa variable, por lo que falla el test de ocurrencia y se concluye que las expresiones de entrada no son unificables.

Ejercicios (Aplicación del Algoritmo de Unificación)

Para cada uno de los siguientes pares de expresiones, aplique el Algoritmo de Unificación para averiguar si las expresiones son unificables y, en caso afirmativo, facilitar un unificador de máxima generalidad (u.m.g.).

1. $p(W, X, f(g(Y)))$ *y* $p(Z, f(Z), f(U))$.
2. $p(a, X, f(g(Y)))$ *y* $p(Z, h(Z, U), f(U))$.
3. $q(h(X, a), X)$ *y* $q(Y, f(a, c))$.
4. $p(a, W, X, f(f(X)))$ *y* $p(Z, g(Y), g(Z), f(Y))$.
5. $q(X, g(Y), a)$ *y* $q(c, Z, Z)$.
6. $p(f(g(Y)), Y, Z)$ *y* $p(f(U), a, U)$.
7. $q(f(g(Z, Y)), Y, Z)$ *y* $q(f(U), b, U)$.

(los dos últimos ejercicios están resueltos en el vídeo Ejemplos Algoritmo de Unificación).

Unificación en PROLOG

PROLOG implementa el Algoritmo de Unificación anterior omitiendo el test de ocurrencia (occur-check) por razones de eficiencia. El lenguaje usa internamente este algoritmo (para aplicar la Regla de Resolución, ver siguiente apartado) pero también lo ofrece para uso del programador:

- El predicado predefinido = *(predicado de unificación)*, se usa en notación infija (`arg1 = arg2`) y:
  - Si las dos expresiones que se le pasan resultan ser, *omitiendo el test de ocurrencia*, unificables, devuelve cierto y realiza las sustituciones de variables indicadas por el u.m.g. calculado por el Algoritmo de Unificación.
  - En caso contrario falla (devuelve *false*).
- El predicado predefinido `\=` *(predicado de no unificación)* también se usa en notación infija (`arg1 \= arg2`) y devuelve cierto (*true*) si el predicado = falla, y falla (devuelve *false*) en caso contrario.

Ejemplo (Unificación en PROLOG)

```
?- f(X, g(X,c)) = f(h(U), Z).
        X = h(U),     Z = g(h(U), c).
?- f(X, g(X,c)) \= f(h(U), Z).
        false
?- p(f(X), g(Y)) = p(U, f(U)).
        false
?- p(f(X), g(Y)) \= p(U, f(U)).
        true
?- X = f(X).
        X = f(X)
% el predicado = no realiza el test de ocurrencia
?- unify_with_occurs_check(X, f(X)).
        false
% este predicado sí realiza el test de ocurrencia
```

Observación (Unificación de variables (semi)anónimas)

La variable anónima _ y las semianónimas (___ más todas las que empiezan por subrayado seguido de letra mayúscula) presentan las siguientes peculiaridades respecto a las variables normales:

	Normales	Semianónimas	Anónima
¿Tienen que unificar?	sí: $p(U, U) = p(a, b)$ da *false*	sí: $p(\_U, \_U) = p(a, b)$ da *false*	no: $p(\_, \_) = p(a, b)$ da *true*
¿Reportan valor?	sí: $p(U, b) = p(a, b)$ da $U = a$	no: $p(\_U, b) = p(a, b)$ da *true*	no: $p(\_, b) = p(a, b)$ da *true*

Ejercicios (El Algoritmo de Unificación de PROLOG)

- *Ejercicio nº 3 de la Práctica de PROLOG nº 1 (soluciones comentadas en Práctica de PROLOG nº 1 con soluciones).*

LA REGLA DE RESOLUCIÓN

Definición (Regla de Resolución (1/3))

La Regla de Resolución de PROLOG recibe como entrada:

- Una cláusula objetivo (consulta) $? - A_1, A_2, \ldots, A_n$, $n \geq 1$.
- Una cláusula de programa, que podrá ser un hecho $B.$ o bien una regla $B :- B_1, \ldots, B_m.$ con $m \geq 1$.

1. Si las cláusulas de entrada tienen variables en común, se deben hacer los renombramientos de variables oportunos (por convenio, en la cláusula de programa) para evitar esta colisión de variables.
2. Si el ▸ Algoritmo de Unificación determina que el *primer* subobjetivo de la cláusula objetivo, A_1, es unificable, mediante un cierto u.m.g. σ, con la cabeza B de la cláusula de programa, entonces la Regla de Resolución se puede aplicar y se llama cláusula resolvente a la cláusula objetivo obtenida como sigue:

Definición (Regla de Resolución (2/3))

- Si la cláusula de programa es un hecho $B.$, la cláusula resolvente es

$$? - A_2\sigma, \ldots, A_n\sigma$$

El subobjetivo A_1 desaparece porque se ha unificado con el hecho B mediante σ. Gráficamente:

$$? - \underline{A_1}, A_2, \ldots, A_n.$$

$$\underline{B}. \qquad \sigma = \{\ldots, \ldots\}$$

$$? - A_2\sigma, \ldots, A_n\sigma.$$

Definición (Regla de Resolución (3/3))

- Si la cláusula de programa es una regla $B :- B_1, \ldots, B_m.$, la cláusula resolvente es

$$? - \underbrace{B_1\sigma, \ldots, B_m\sigma}_{\text{cuerpo de la regla}}, \quad \underbrace{A_2\sigma, \ldots, A_n\sigma}_{\text{resto de subobjetivos}}$$

El subobjetivo A_1 se reemplaza por $B_1\sigma, \ldots, B_m\sigma$ puesto que se ha unificado con B mediante σ. Gráficamente:

$$? - \underline{A_1}, A_2, \ldots, A_n.$$

$$\underline{B} :- B_1, \ldots, B_m. \qquad \sigma = \{\ldots, \ldots\}$$

$$? - B_1\sigma, \ldots, B_m\sigma, A_2\sigma, \ldots, A_n\sigma.$$

Observación

En las cláusulas resolventes:

- $A_i\sigma/B_i\sigma$ son el resultado de ▸ aplicar la sustitución σ a los predicados A_i/B_i.
- Se debe respetar el orden en el que aparecen los predicados en la cláusula resolvente (primero los $B_i\sigma$, en el orden en el que aparecen en la regla, y luego los $A_i\sigma$, en el orden en el que estaban en la cláusula objetivo original).
- Al aplicar la Regla de Resolución entre un objetivo *unitario* "$? - A_1$." y un hecho "B." unificable con A_1, la cláusula resolvente obtenida es $?-$, que se conoce como la cláusula vacía.

Ejemplo (Regla de Resolución)

Dada la cláusula objetivo `?- amigo(abel,X), enemigo(X,Y).`

- La Regla de Resolución no es aplicable con el hecho `enemigo(abel, cain)` puesto que `amigo(abel, X)` y `enemigo(abel, cain)` no son unificables (¡ni siquiera tienen el mismo nombre de predicado!).
- Tampoco con la regla `amigo(abilio,X1):-amigo(abel,X1)`, puesto que `amigo(abel, X)` y `amigo(abilio, X1)` no son unificables (tienen dos constantes distintas como 1er argumento).
- Sí lo sería con `amigo(X1,Y1) :- enemigo(X1,Z1), enemigo(Z1,Y1)` (con renombramiento de variables debido a la coincidencia de variables entre las dos cláusulas): `amigo(abel, X)` y `amigo(X1, Y1)` sí son unificables, con u.m.g. $\sigma = \{X1/abel, X/Y1\}$, y la cláusula resolvente sería `?-enemigo(abel,Z1),enemigo(Z1,Y1),enemigo(Y1,Y).`

Ejemplo (Regla de Resolución)

¿Es posible aplicar la Regla de Resolución entre el objetivo `?-q(X,a),p(X),q(b,Y)` y la regla `q(f(X),Y) :- r(X),r(Y)`?

1. Como las dos cláusulas dadas tienen variables en común, se renombran las variables de la regla, que pasa a ser `q(f(X1), Y1) :- r(X1),r(Y1).`
2. la Regla de Resolución sí es aplicable entre las cláusulas dadas puesto que el Algoritmo de Unificación muestra que el primer predicado del objetivo, `q(X,a)`, es unificable con la cabeza de la regla, `q(f(X1), Y1)`, mediante u.m.g. $\sigma = \{X/f(X1), Y1/a\}$. La cláusula resolvente correspondiente es entonces:

$$\begin{aligned} & ?- \; r(X1)\sigma, r(Y1)\sigma, p(X)\sigma, q(b, Y)\sigma \\ = \; & ?- \; r(X1), r(a), p(f(X1)), q(b, Y). \end{aligned}$$

Ejercicios (Aplicación de la Regla de Resolución)

Para cada uno de los siguientes pares de cláusulas, razone si es posible aplicar la Regla de Resolución, y, en caso afirmativo, calcule la cláusula resolvente asociada:

1. *Objetivo* $?- r(Z,a), q(a,f(a))$. *y regla* $q(X,X) :- p(X), r(X,a)$.
2. *Objetivo* $?- q(a,f(a)), r(Z,a)$. *y regla* $q(X,X) :- p(X), r(X,a)$.
3. *Objetivo* $?- q(X,a), p(X), q(b,Y)$. *y hecho* $q(X,X)$.
4. *Objetivo* $?- r(f(X),X), q(X,Z), r(f(Z),f(f(a)))$. *y regla* $r(X,f(Z)) :- p(X,Z), q(Z,X)$.
5. *Objetivo* $?- p(X,b)$. *y hecho* $p(h(a,X),X)$.

EL ÁRBOL DE RESOLUCIÓN

Dados:

- Un *programa* (conjunto de fórmulas -cláusulas de Horn positivas- que pueden ser hechos o reglas),
- Y una *consulta* (fórmula denominada cláusula de Horn negativa),

el Árbol de Resolución asociado a ese programa y esa consulta

- Es una herramienta para *explorar y organizar todas las posibles formas en las que es posible resolver la consulta*, es decir, los posibles caminos (si los hay) para demostrar que la consulta es una consecuencia lógica del programa y, en su caso, *computar* las soluciones asociadas.
- Se construye aplicando reiteradamente la *Regla de Resolución* como se describe a continuación.

Construcción de un Árbol de Resolución (1/2)

El Árbol de Resolución correspondiente a un programa y una consulta es un árbol etiquetado y con raíz:

- Raíz del árbol: la consulta.
- Nodos: son cláusulas objetivo de la forma "`?- O1,..., Ok.`", con $k \geq 0$, que se expanden de la siguiente forma:
  - Si $k = 0$ (cláusula vacía), el nodo no se puede expandir, es una hoja del árbol denominada nodo éxito.
  - Si $k \neq 0$ pero no existe ninguna cláusula en el programa con la que se pueda aplicar la ▸ Regla de Resolución, el nodo tampoco se puede expandir, es una hoja del árbol denominada nodo fallo.
  - En cualquier otro caso, el nodo tendrá un nodo hijo por cada cláusula del programa con la que se pueda aplicar la ▸ Regla de Resolución, de forma que los hijos contienen las *cláusulas resolventes* obtenidas al aplicar la Regla de Resolución.

Construcción de un Árbol de Resolución (2/2)

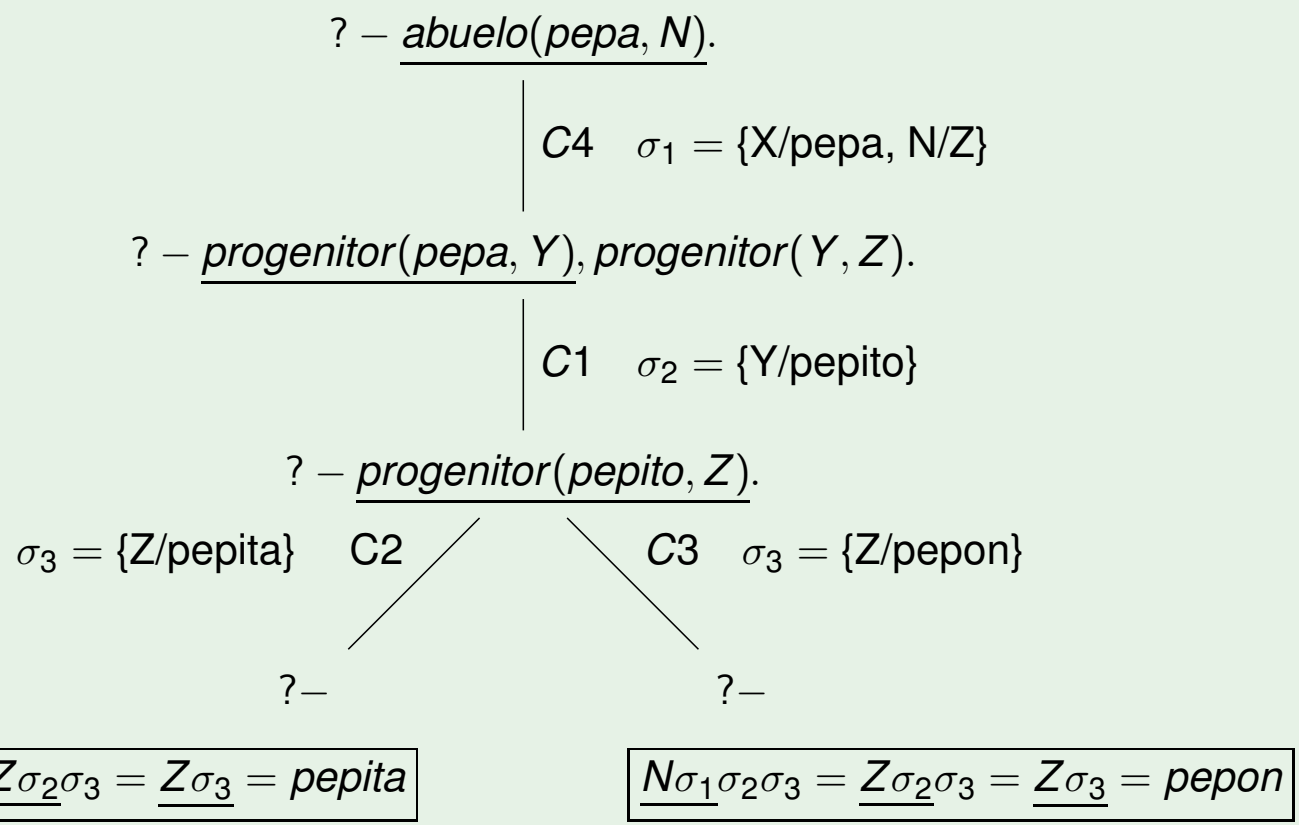
- Los hijos de los nodos que los tengan se colocan de izquierda a derecha eligiendo las cláusulas (hechos o reglas) del programa *de acuerdo con el orden en el que aparecen en él* (el hijo de más a la izquierda será la cláusula resolvente obtenida con la primera cláusula del programa con la que se pueda aplicar la Regla de Resolución, etc).
- Las aristas se etiquetan con la siguiente información:
  - El u.m.g. σ_j utilizado al aplicar la Regla de Resolución. Convenio: los u.m.g.'s se numeran en función del nivel del árbol (σ_1 en el primer nivel, etc).
  - Opcionalmente, la cláusula C_i (C_i^{ren} si se ha renombrado) del programa con la que se aplica la Regla de Resolución. Convenios: (1) la primera cláusula del programa es C_1, la segunda C_2, etc. (2) en caso necesario, las variables se renombran añadiéndoles un subíndice que indica el nivel del árbol (X_1 en el primer nivel, etc).

Interpretación de las hojas de un Árbol de Resolución

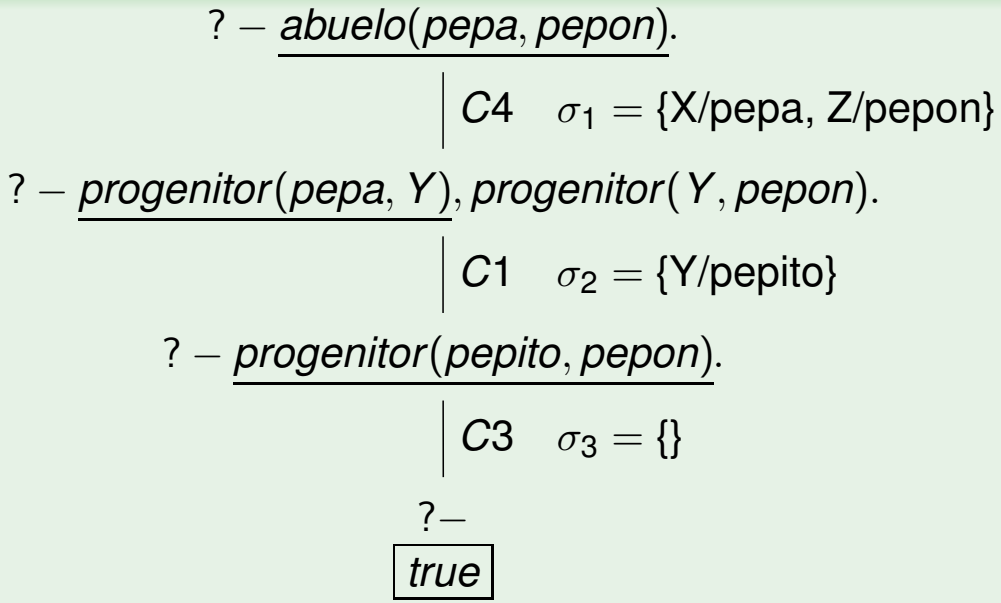
- Nodos éxito
  - Prueban que la consulta se ha podido realizar con éxito (se ha probado que es una consecuencia lógica del programa).
  - Cada nodo éxito proporciona una solución a la consulta:
    - Si la consulta no tiene variables: true.
    - En otro caso: la solución está compuesta por el valor $X\sigma_1 \ldots \sigma_r$ para cada variable X (salvo anónimas o semianónimas) que aparezca en la consulta, donde $\sigma_1, \ldots, \sigma_r$ son los u.m.g.'s asociados con cada una de las aristas de la rama, en orden, empezando desde la raíz del árbol. La forma más fácil de computar $X\sigma_1 \ldots \sigma_r$ es ▸ aplicar σ_1 a X, a continuación ▸ aplicar σ_2 a la expresión resultante de la aplicación anterior, etc.
- Nodos fallo
  - Son fracasos a la hora de intentar probar la consulta.
  - Su aparición no significa que la consulta no se pueda realizar con éxito, simplemente muestra que la rama elegida para intentarlo no lo permite.

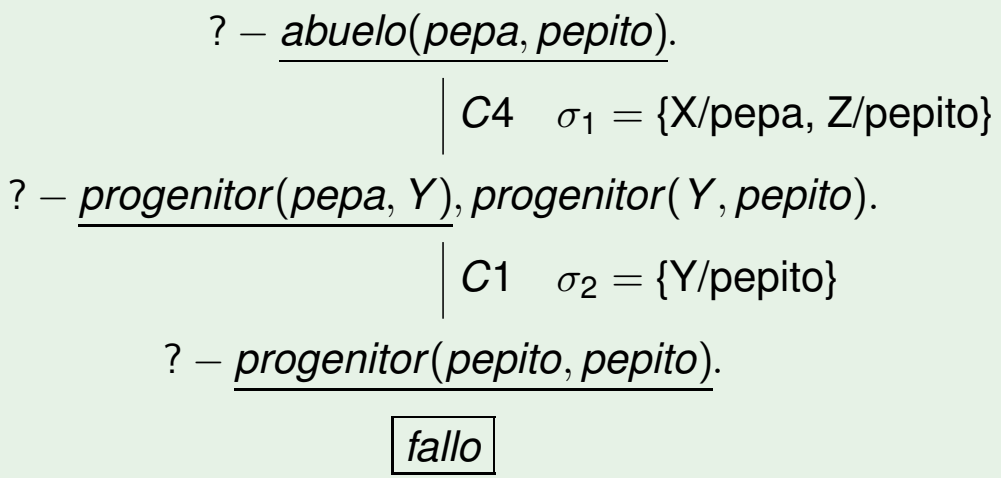
Ejemplo (Árboles de Resolución programa "Abuelas/os" 1/2)

```
progenitor(pepa, pepito).                          % C1
progenitor(pepito, pepita).                        % C2
progenitor(pepito, pepon).                         % C3
abuelo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).   % C4
```



Ejemplo (Árboles de Resolución programa “Abuelas/os” 2/2)



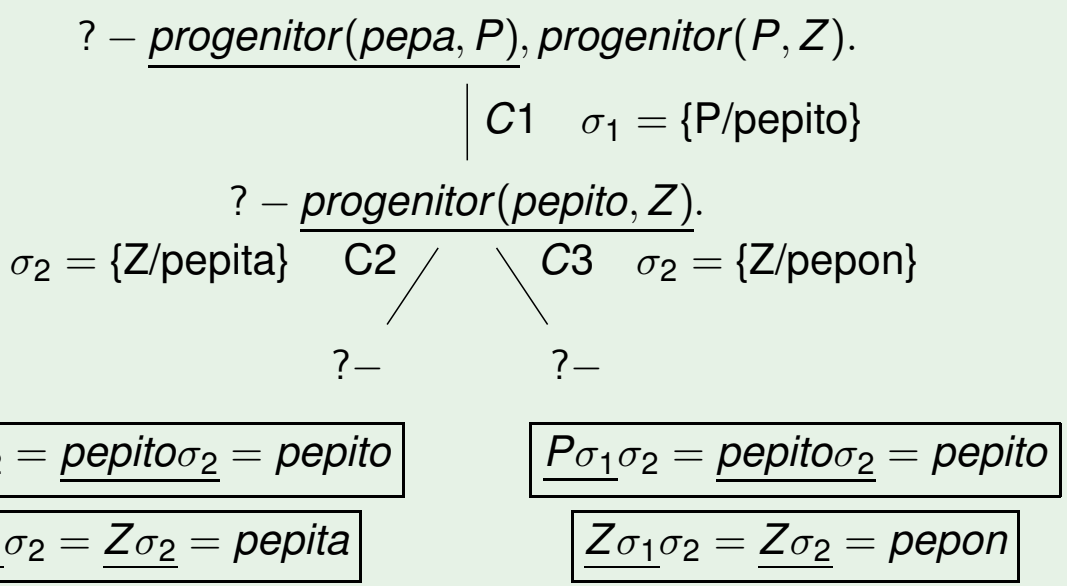


Observación (Árboles con variables anónimas/semianónimas)

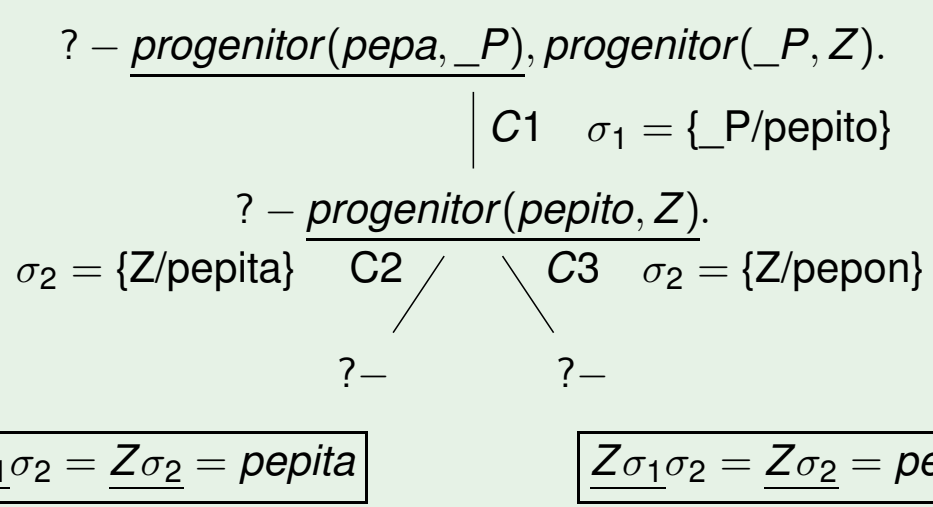
Al construir Árboles de Resolución en los que aparezcan variables anónimas o semianónimas se deben tener en cuenta las particularidades de este tipo de variables en la unificación. Compare los tres árboles a continuación.

Ejemplo (Árboles con variables anónimas/semianónimas (1/2))

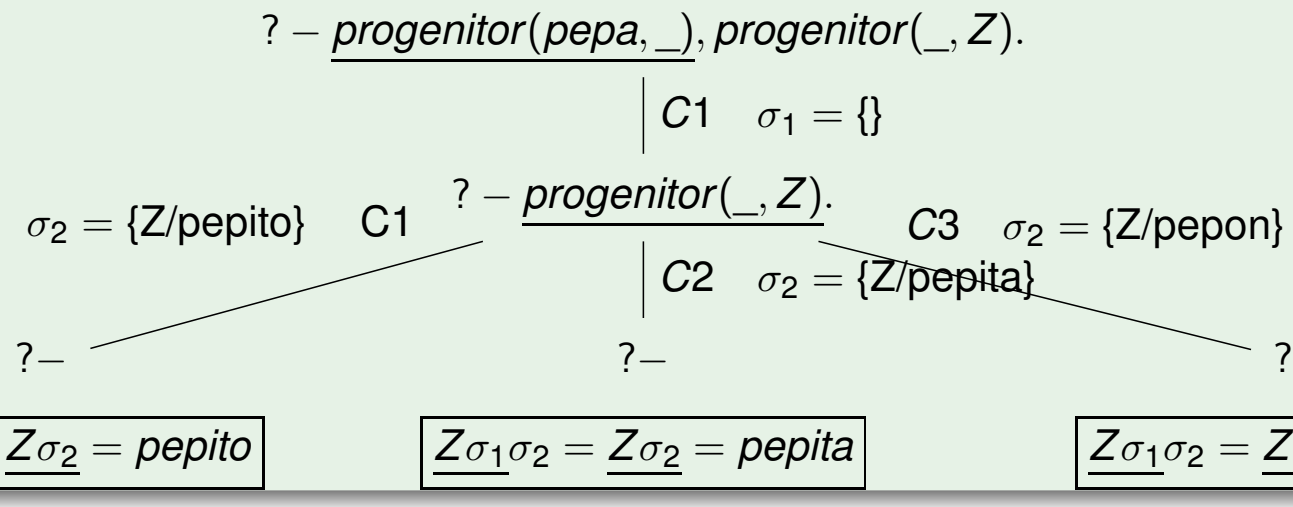
P = variable normal



Ejemplo (Árboles con variables anónimas/semianónimas (2/2))



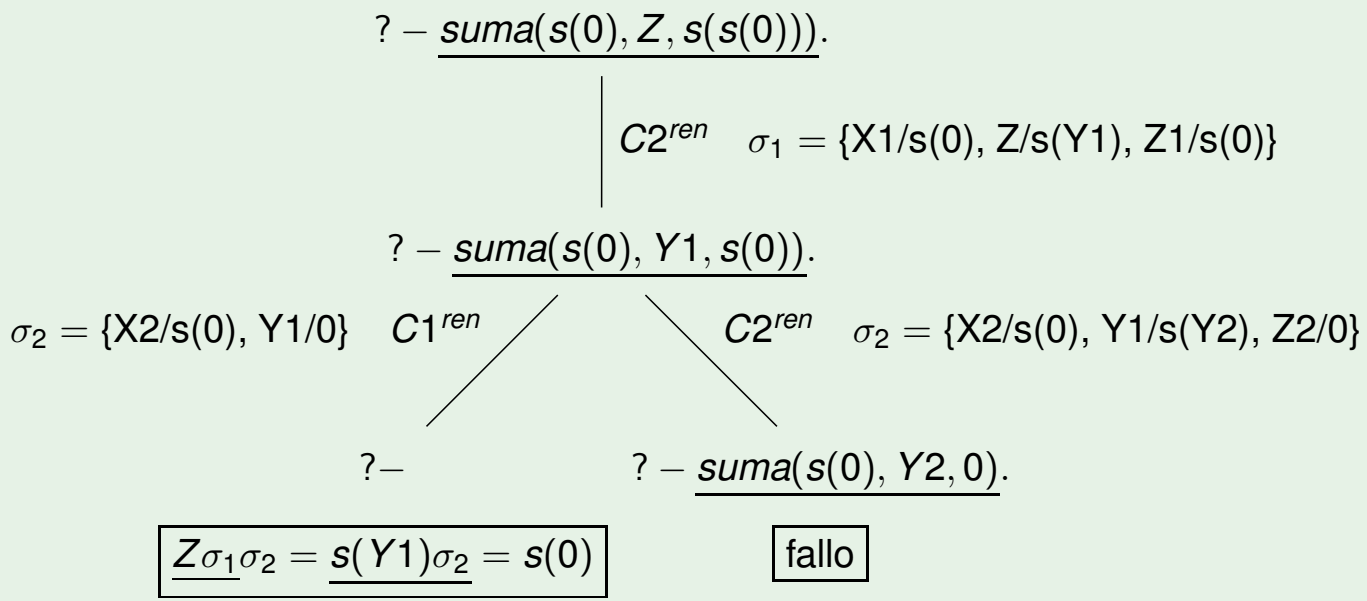
_P = v. semianónima



_ = v. anónima

Ejemplo (Árbol de Resolución programa "suma")

```
suma(X,0,X).                          % C1
suma(X,s(Y),s(Z)) :-  suma(X,Y,Z).    % C2
```



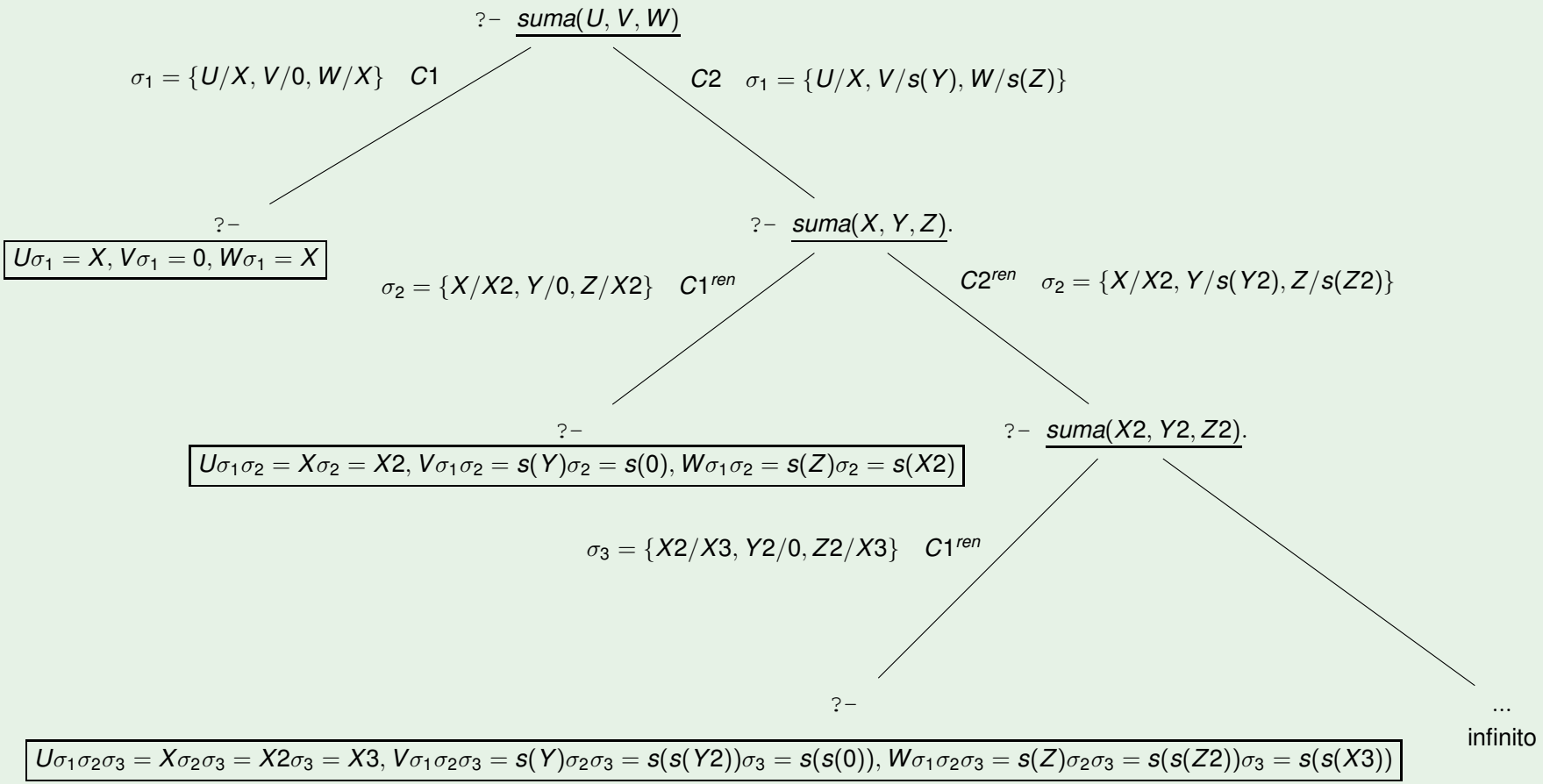
Observación (Árboles con ramas infinitas)

Los Árboles de Resolución también pueden tener ramas infinitas, que pueden ser de dos clases:

- Ramas con un número infinito de soluciones que van apareciendo sucesivamente (ver ejemplo a continuación con el programa de la suma), o bien
- Ramas sin soluciones que desarrollan hijos indefinidamente, hasta que acaban desbordando la memoria (ver ejemplo más adelante con la tercera versión del programa “ancestro”, `ancestro3`, rama de más a la derecha).

Ejemplo (Árbol de Resolución con infinitas soluciones)

```
suma(X,0,X).                              % C1
suma(X,s(Y),s(Z)) :-  suma(X,Y,Z).        % C2
```



Ejercicios (Construcción de Árboles de Resolución)

Construya los Árboles de Resolución asociados con los programas y consultas dados a continuación:

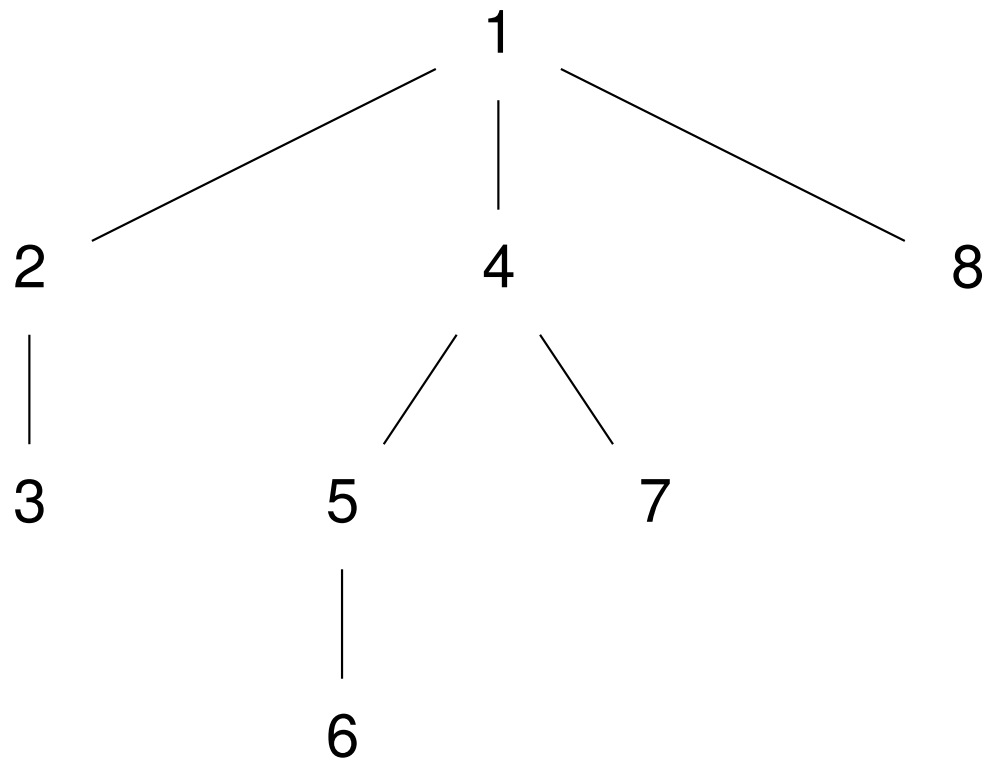
1. ▸ Programa Abuelos *y consulta (escríbala en LPO y en* PROLOG*) adecuada para averiguar quiénes son abuelas/os de Pepón.*
2. ▸ Programa Abuelos *y consultas (escríbalas en LPO y en* PROLOG*) adecuadas para averiguar si Pepito y Pepón son hermanos (tienen algún progenitor en común) y, en caso afirmativo, saber quiénes son esos progenitores comunes. ¿Y Pepita y Pepón?*
3. *Programa Amistades/Enemistades y consulta (escríbala en LPO y en* PROLOG*) adecuada para averiguar si Abel es amigo de alguien que, a su vez, sea enemigo de alguien, y, en caso afirmativo, saber quiénes serían.*
4. ▸ Programa Suma *y consulta* `?- suma(X,Y,s(s(0))).` *(escríbala en LPO y describa su objetivo en lenguaje natural).*

Mecanismo de cómputo de PROLOG

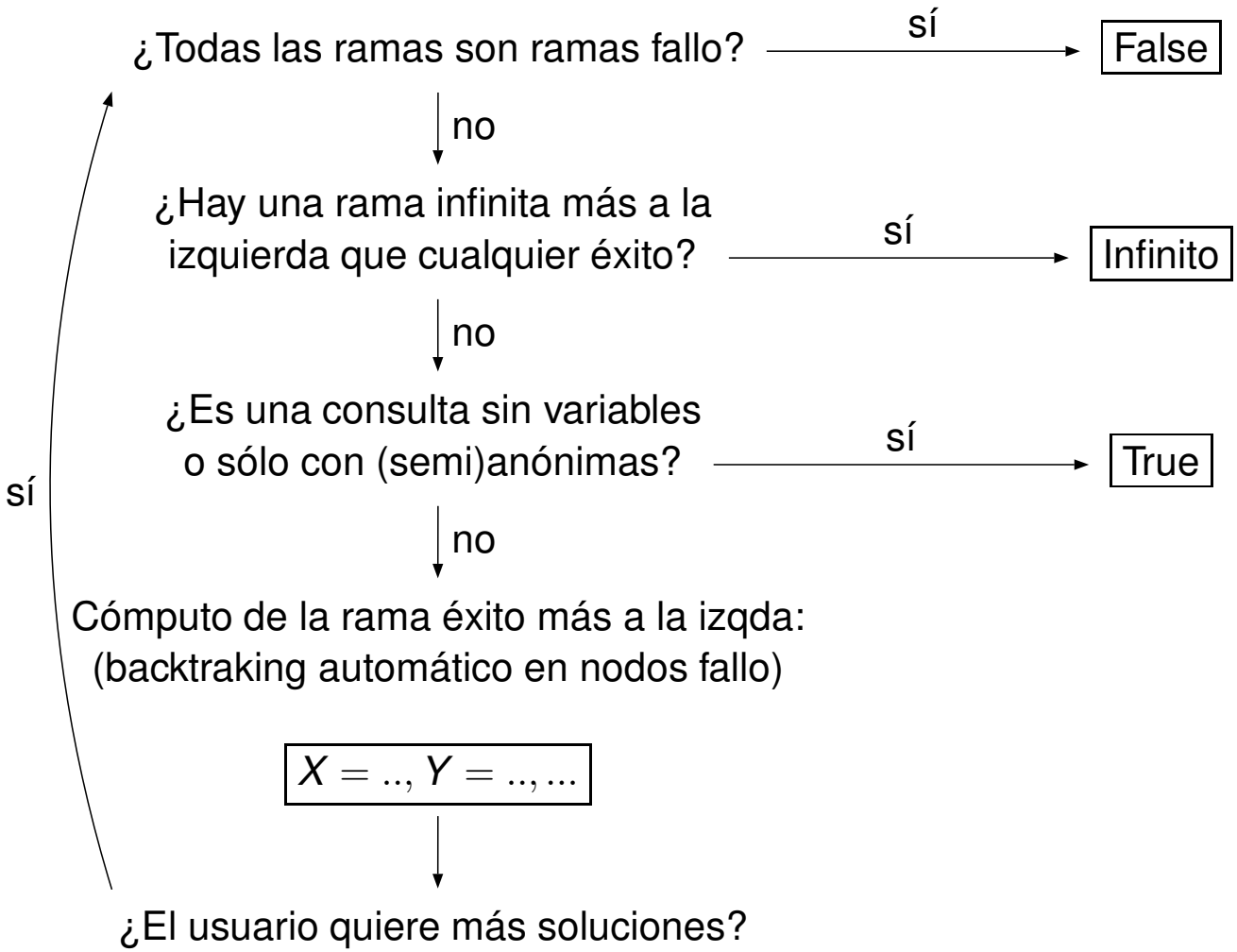
Cuando recibe una consulta relativa a un programa, PROLOG construye el Árbol de Resolución correspondiente de la siguiente forma:

- El árbol se construye en profundidad por la izquierda (ver dibujo en la página siguiente).
- El árbol se construye utilizando retroceso (backtracking) automático al llegar a un nodo fallo: cuando se encuentra con uno de estos nodos, PROLOG *no da cuenta del fallo*, simplemente retrocede para continuar la búsqueda (en profundidad) de posibles éxitos.
- El árbol no se construye necesariamente entero: en cada nodo *éxito* el sistema pregunta al usuario si quiere seguir o terminar la ejecución de la consulta.

Ejemplo de orden (en profundidad) con el que PROLOG construye los Árboles de Resolución:



Actitud de PROLOG al construir (en profundidad) un Árbol de Resolución



Ejemplo (Respuestas ofrecidas por PROLOG)

¿Qué respuestas ofrecería PROLOG, y en qué orden, al construir los Árboles de Resolución, completos, de los últimos ejemplos?

- En los ejemplos ▸ Árboles de Resolución programa Abuelas/os:
  - En el primer árbol: primera solución: `N = pepita`, segunda solución: `N = pepon` y a continuación `false` (este último indicando que no hay más soluciones).
  - Las respuestas para los otros dos árboles serían `true` y `false`.
- En el ejemplo ▸ Árbol de Resolución programa Suma:
  - Primera solución: `Z = s(0)`
  - `false` (indicando que no hay más soluciones -el nodo fallo no se reporta-)
- En el ejemplo ▸ Árbol de Resolución con infinitas soluciones:
  - Primera solución: `U = X, V = 0, W = X`
  - Segunda solución: `U = X2, V = s(0), W = s(X2)`
  - Tercera solución: `U = X3, V = s(s(0)), W = s(s(X3))`
  - Cuarta solución: ... (seguiría dando todas las posibles soluciones)

Ejercicios (Respuestas ofrecidas por PROLOG)

Para cada uno de los Árboles construidos en los ejercicios ▸ *Construcción de Árboles de Resolución*, *indique qué respuestas ofrecería* PROLOG, *y en qué orden lo haría.*

Observación (Influencia del orden en programas y reglas)

Tanto:

1. El orden de las cláusulas en el programa, como
2. El orden de los predicados en el cuerpo de una regla.

pueden influir en:

1. Qué soluciones se encuentran y en qué orden aparecen.
2. La terminación o no de las consultas que se realizan.

En particular, la búsqueda de soluciones no es completa: no asegura encontrar una solución a la consulta aunque la haya (esto ocurrirá siempre que el Árbol de Resolución tenga alguna rama infinita más a la izquierda que la primera rama éxito).

El predicado `ancestro`, cuya implementación se discute a continuación, se usa a menudo para ilustrar lo anterior.

Ejemplo (El predicado `ancestro`)

Dado `progenitor(X,Y)`, cierto si X es progenitor/a de Y, se define el predicado `ancestro(X,Y)`, cierto si X es un ancestro de Y, es decir, si X es progenitor/abuela-o/bisabuela-o/etc de Y.

- Caso base. Los progenitores son ancestros de sus hijas/os: $\forall X \forall Y(progenitor(X, Y) \rightarrow ancestro(X, Y))$. En Prolog:

```
ancestro(X, Y) :-
  progrenitor(X, Y).
```

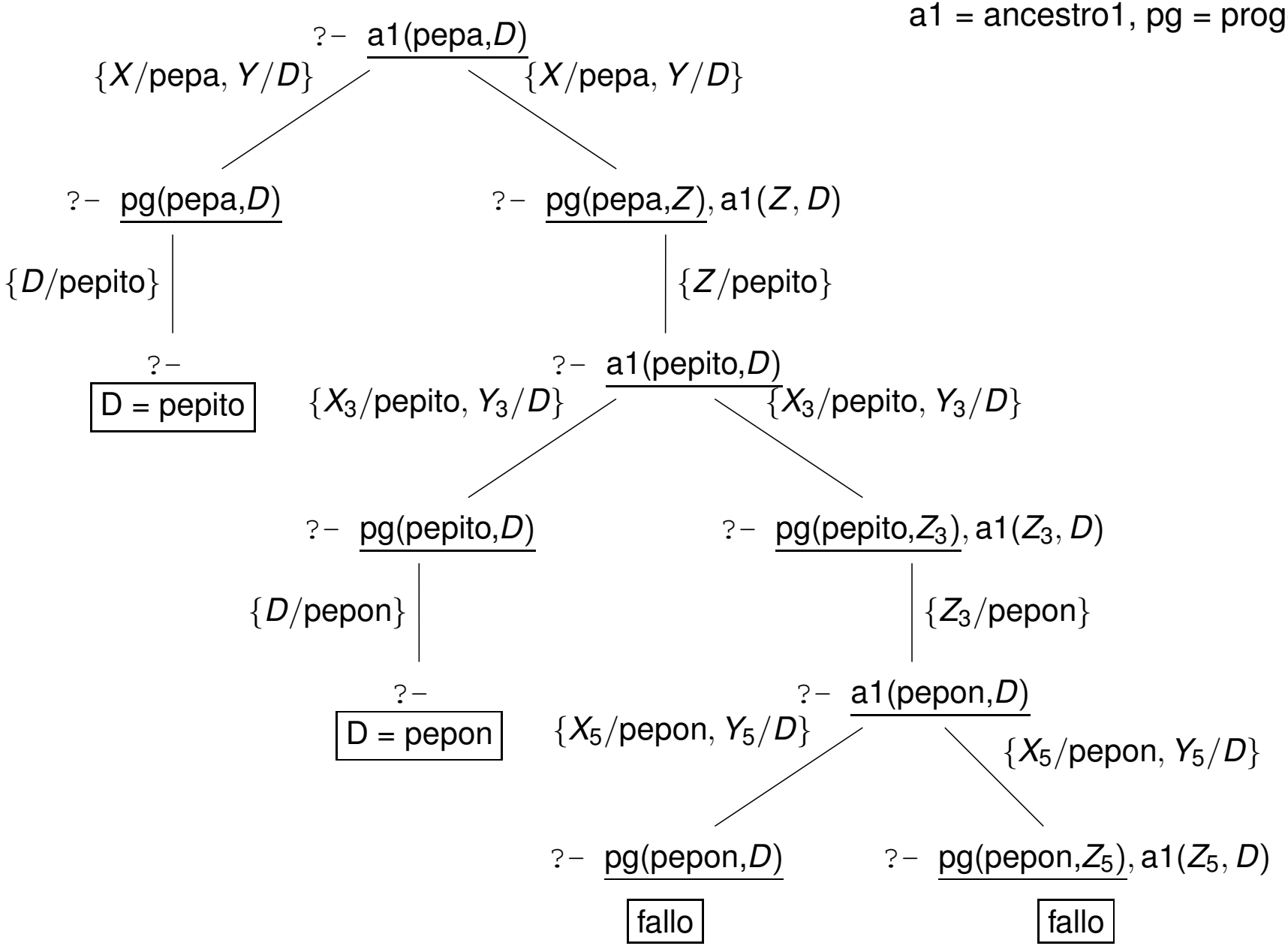
- Caso recursivo. Los progenitores son ancestros de todos aquellos de los que sus hijas/os lo son. $\forall X \forall Y \forall Z((progenitor(X, Z) \wedge ancestro(Z, Y)) \rightarrow ancestro(X, Y))$. En Prolog:

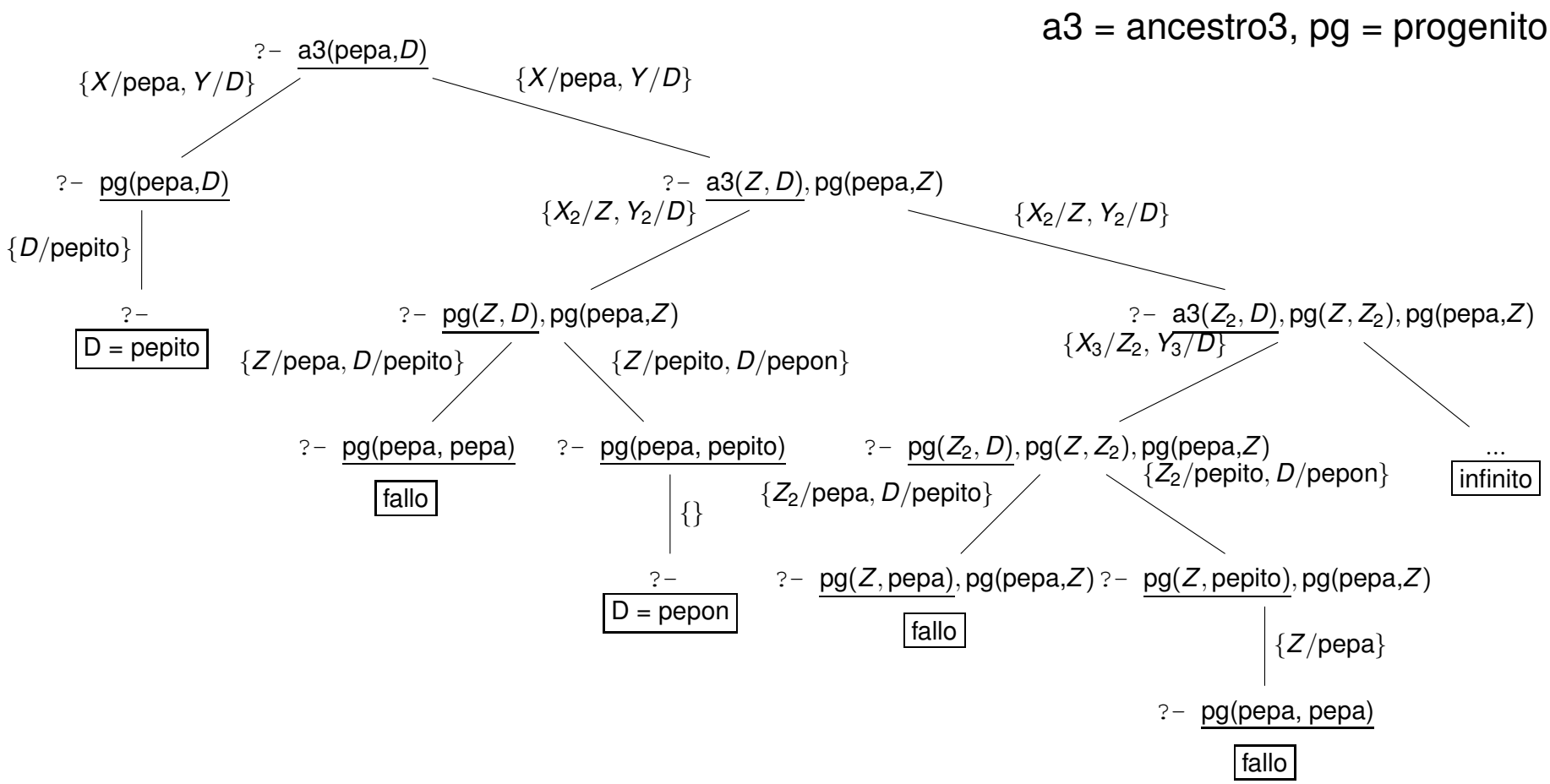
```
ancestro(X, Y) :-
  progrenitor(X, Z),
  ancestro(Z, Y).
```

Ejemplo (Influencia del orden en programas y reglas)

```
progenitor(pepa, pepito).
progenitor(pepito, pepon).
% CUATRO POSIBLES ORDENACIONES PARA "ancestro"
(todas equivalentes en Lógica, pero no en Prolog)

ancestro1(X,Y) :- progenitor(X,Y).
ancestro1(X,Y) :- progenitor(X,Z), ancestro1(Z,Y).

ancestro2(X,Y) :- progenitor(X,Z), ancestro2(Z,Y).
ancestro2(X,Y) :- progenitor(X,Y).

ancestro3(X,Y) :- progenitor(X,Y).
ancestro3(X,Y) :- ancestro3(Z,Y), progenitor(X,Z).

ancestro4(X,Y) :- ancestro4(Z,Y), progenitor(X,Z).
ancestro4(X,Y) :- progenitor(X,Y).
```

a1 = ancestro1, pg = progenitor





Los árboles de `ancestro1` y `ancestro3` son muy distintos entre sí. Los de `ancestro2` y `ancestro4` no son otra cosa que las *imágenes especulares* de los árboles de `ancestro1` y `ancestro3`, respectivamente (hágalos como ejercicio).

Respuestas ofrecidas: PROLOG construye los Árboles de Resolución correspondientes en profundidad, ofreciendo las soluciones de las ramas éxito según las va encontrando y sin reportar las ramas fallo (donde hace backtracking de forma automática):

```
?- ancestro1(pepa, D).
D = pepito ; D = pepon

?- ancestro2(pepa, D).
D = pepon ; D = pepito. % Soluciones intercambiadas

?- ancestro3(pepa, D).
D = pepito ; D = pepon ; % Encuentra las soluciones
ERROR: Out of local stack % rama infinita

?- ancestro4(pepa, D). % ¡NO encuentra soluciones!
ERROR: Out of local stack  % rama infinita
```

- No existe ninguna regla general que establezca el orden óptimo de las cláusulas en el programa ni el orden óptimo de los predicados en el cuerpo de las reglas (depende de cada caso).
- En general, sí es recomendable empezar por "lo más sencillo":
  1. Colocar primero los casos base (cláusulas que expresan las condiciones de parada de la recursividad).
  2. Evitar las reglas con recursión a la izquierda (reglas tales que el predicado del primer subobjetivo de su cuerpo coincide con el predicado de su cabeza), primando siempre que sea posible la recursión de cola o recursión final (recursión en último lugar).

Ejemplo

De las cuatro versiones del predicado `ancestro`, la única que cumple las dos recomendaciones anteriores es la primera, `ancestro1`.

Ejercicios (El orden importa + uso del depurador)

Ejercicio nº 4 de la Práctica de PROLOG nº 1.

Ejercicios (Mecanismo de cómputo de PROLOG)

1. *Considere el programa*

```
p(X,X).                          % C1
p(X,Z) :- q(X,Y), p(Y,Z).        % C2
q(a,b).                          % C3
```

   1. *Exprese en lenguaje natural y en LPO el significado de cada una de las cláusulas del programa.*
   2. *Construya el Árbol de Resolución correspondiente a la consulta* `?- p(X,b).` *¿Qué se pretende averiguar? ¿Qué respuestas ofrecería* PROLOG*, y en qué orden?*
   3. *¿Qué consecuencias tendría el intercambio de las dos primeras cláusulas del programa?*
   4. *¿Qué consecuencias tendría el intercambio de los dos predicados del cuerpo de la regla?*
   5. *¿Qué significado tiene la consulta* `?- p(X,b), q(X,b)` *? Construya el Árbol de Resolución necesario e indique qué respondería* PROLOG *ante esta consulta.*

BIBLIOGRAFÍA

- L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, Cambridge, Mass., second edition, 1994.
- W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, fifth edition, 2003.
- I. Bratko.*Prolog Programming for Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition, 2001.
- J. Lloyd. *Foundations of Logic Programming*, (Second Edition). Springer-Verlag, 1987.
- R. O'Keefe. *The Craft of Prolog*. The MIT Press, Cambridge, MA, 1990.
- U. Nilsson and J. Maluszynski. Logic, Programming and Prolog. John Wiley & Sons Ltd, 1996.
- SWI-Prolog, entorno de programación en Prolog de dominio público.
- comp.lang.prolog. Faq