

# Tema4ApuntesTeoria.pdf



Isavb3

**Lenguajes, Tecnologías y Paradigmas de la Programación****2º Grado en Ingeniería Informática****Escuela Técnica Superior de Ingeniería Informática  
Universidad Politécnica de Valencia****Te regalamos****1/6**Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.**1****Abre tu Cuenta Online sin comisiones ni condiciones****2****Haz una compra igual o superior a 15€ con tu nueva tarjeta****3****BBVA te devuelve un máximo de 15€**

# Te regalamos



**1/6**

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

**1**

Abre tu Cuenta Online sin comisiones ni condiciones

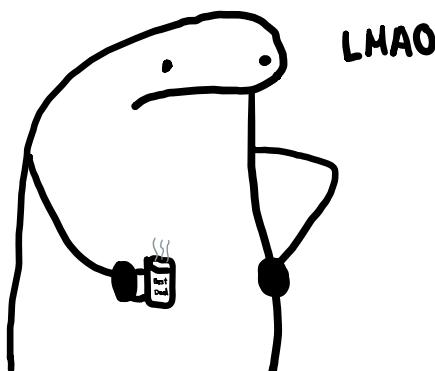
**2**

Haz una compra igual o superior a 15€ con tu nueva tarjeta

**3**

BBVA te devuelve un máximo de 15€

## LENGUAJES, TECNOLOGÍAS Y PARADIGMAS DE PROGRAMACIÓN



## TEMA 4 – PARADIGMA LÓGICO

### 1. INTRODUCCIÓN A LA PROGRAMACIÓN LÓGICA

#### 1.1 Uso de la lógica como lenguaje de programación

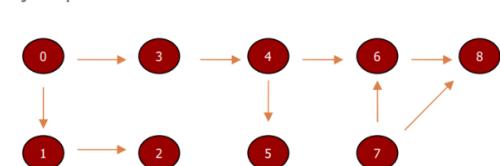
Escribir un programa lógico consiste en expresar una relación utilizando una notación lógica basada en la lógica de predicados (lógica de las cláusulas de Horn).

En el paradigma lógico → la computación como una **DEDUCCIÓN**

**PROGRAMA** → Representación del conocimiento sobre el problema mediante **ESCRIBIR SENTENCIAS LÓGICAS**

**EJECUCIÓN DEL PROGRAMA** → Serie de deducciones partiendo de una cierta consulta

Ejemplo:



Grafo  
 • nodos 0...8  
 • arcos  
 • conectados?

```

arcos(0,3).  arco(3,4).  arco(4,6).  arco(6,8).
arcos(0,1).  arco(1,2).  arco(4,5).
arcos(7,6).  arco(7,8).
conectado(X,Y) :- arco(X,Y).      recursivo
conectado(X,Y) :- arco(X,Z), conectado(Z,Y).
  
```

← Hechos

← Reglas

Información condicional

Se lee como 'si...'

**CONSULTA**  
 ¿Están conectados 0 y 8?  
 ?- conectado(0,8).  
 true

#### 1.2 Variables Lógicas

Son incógnitas a despejar de una ecuación cuyo valor queremos obtener a través de una serie de pasos de deducción.

Los hechos y reglas de nuestro programa tienen las variables cuantificadas universalmente

Ejemplo:

conectado(X, Y) :- arco(X, Z), conectado(Z, Y).



de manera implícita estamos diciendo que todas sus variables están cuantificadas universalmente

$\forall X, Y, Z (\text{conectado}(X, Y) :- \text{arco}(X, Z), \text{conectado}(Z, Y))$



todavía más intuitivo → para todo X e Y, X está conectado con Y si existe una Z tal que...

$\forall X, Y (\text{conectado}(X, Y) :- \exists Z (\text{arco}(X, Z), \text{conectado}(Z, Y)))$

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

# Te regalamos

15€

- 1 Abre tu Cuenta Online sin comisiones ni condiciones
- 2 Haz una compra igual o superior a 15€ con tu nueva tarjeta
- 3 BBVA te devuelve un máximo de 15€



### 1.3 Extracción de Respuestas

El uso de variables lógicas permite extraer respuestas, las **variables de las consultas** están cuantificadas existencialmente.

La ejecución realiza un proceso para averiguar valores de esas variables para los cuales la consulta es cierta.

?- conectado(X, Y)  
X=0  
X=1



Se lee → ¿Existen X Y tales que conectado(X,Y) es cierto con respecto al programa?

El **mecanismo** para hallar X Y es **constructivo**, si tiene éxito proporciona el valor de X Y

### 1.4 Invertibilidad

*Los argumentos de un predicado pueden ser tanto de entrada como de salida*

Ejemplo:    suma(x, y, z) //La suma de x+y=z  
suma(3, 4, z) ? z=7

Puedo usar esta misma relación de esta manera:

suma(x, 5, 10) ? x=5 //Aquí resto 10-5 para hallar x

Otro ejemplo:

//Comprueba si un elemento pertenece a una lista  
member(H,[H|L]).  
member(H,[H|L]) :- member(H,L).

Un elemento pertenece a una lista si está en cabeza o cuando no está en cabeza si recursivamente lo encuentra en el resto de la lista

Podemos realizar diferentes tipos de consulta:

1. Toda la información es de entrada, conocido: member(2, [1, 2])
2. 1º argumento NO conocido: member(X, [1, 2]) Devuelve valores de X tales que X pertenece a la lista.
3. Hacer llamadas: member(1, L) Devuelve listas en las que aparezca un 1.

True  
False

### 1.5 Indeterminismo

Una consulta puede tener varias respuestas, si contiene variables.

?- member(X, [1, 2, a]). {  
    x = 1  
    x = 2  
    x = a}

## **2. SINTAXIS DE LOS PROGRAMAS LÓGICOS**

## 2.1 Términos

Los datos de un programa lógico se denominan términos, pueden ser:

VARIABLES	CONSTANTES	ESTRUCTURAS
Comienzan por Mayúscula. Variable anónima: “_”	Comienzan por minúscula o comillas simples.	Forma: $f(t_1, \dots, t_n)$ $f \rightarrow$ nombre función $t \rightarrow$ términos
Ej.: X, Y, Area...	Numéricas: Enteros, reales...	f es un constructor de datos, comienza por minúscula
	Simbólicas: Identificador	Ej.: Hora(H, M, S)
	Ej.: 42, 3, lunes, ‘Pedro’	

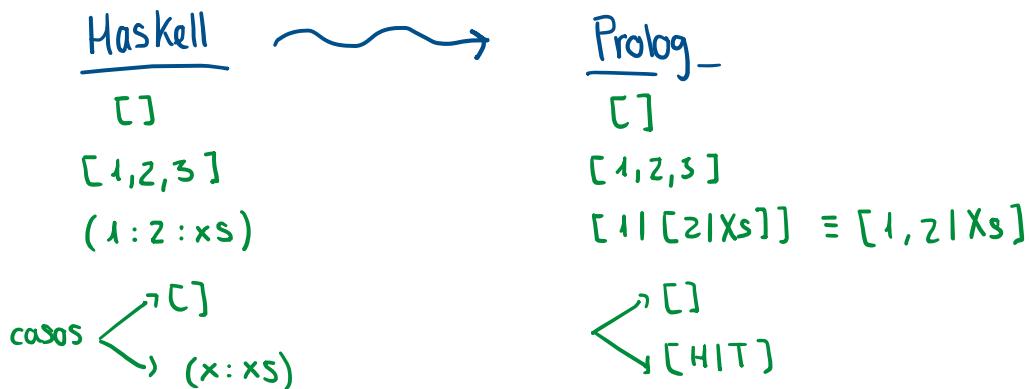
## 2.2 Listas (notación Prolog)

- Lista vacía → [ ]
  - Constructor de listas → [ | ] barra vertical

Ejemplo: [1|2|[ ]] equivale a [1,2]

Lista que empieza por 1, su resto es una lista que empieza por 2, cuyo resto es una lista vacía

[x|x<sub>s</sub>] ↑↑↑ Importante corchetes  
Mayus.



## 2.3 Átomos

Los **átomos** son las construcciones ejecutables →  $p(t_1, \dots, t_n)$  (Haskell: llamadas a función)

No hay anidamiento

~~$\rho(\rho(\dots))$~~

## terminos

1

↑  
símbolo de predicado → en minúscula

Se representa así → p/n → aridad, número de argumentos

Sirven para representar **propiedades o relaciones sobre** una serie de **datos**: arco(1,2)

# Te regalamos

15€

1 / 6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

**BBVA** está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

1

**Abre tu Cuenta  
Online  
sin comisiones  
ni condiciones**

2

**Haz una compra igual o superior a 15€ con tu nueva tarjeta**

3

**BBVA**  
te devuelve  
un **máximo de**  
**15%**

Isabel Vallés Bertomeu

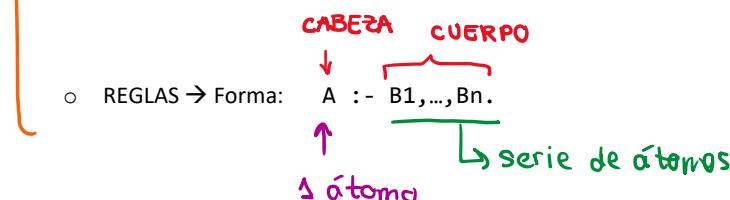
1TP

Tema 4

## 2.4 Programas Prolog

Es un conjunto de secuencias/declaraciones que pueden ser **hechos o reglas**.

- HECHOS → En general tiene la forma de ÁTOMO A.  
Ej.:  $\text{arco}(\theta, 1)$ .



Si  $B_1, \dots, B_n$  es cierto, entonces  $A$  es cierto

## 2.5 Objetivos

Es la llamada a ejecución de un programa lógico, se escribe como una cláusula sin cabeza

? - B1, ..., Bn

La ejecución **comienza con un objetivo**, el cual se irá reduciendo paso a paso hasta llegar a la **cláusula vacía “?-“**

## 2.6 De Haskell a Prolog

- ✓ En Haskell definíamos funciones
  - ✓ En Prolog procedimientos → tanto entradas como salidas son argumentos del predicado que definimos.

### 3 diferencias principales →

```
fibonacci(0) = 0                                     /*Haskell*/  
fibonacci(1) = 1  
fibonacci(n) | n>1 = fibonacci(n-1) + fibonacci(n-2)
```

```
fibonacci(0,0)                                /*Prolog*/  
fibonacci(1,1)  
fibonacci(N,M) :- N>1, N1 is N-1, N2 is N-2, fibonacci(N1,F1), M is F1+F2
```

## 1. De función a procedimiento

Lo que era **salida en Haskell**, en Prolog lo vemos como **argumentos**, el resultado es un parámetro más del procedimiento/predicado.

$$\text{fibonacci}(0) = 0 \rightarrow \text{fibonacci}(0, 0)$$

## 2. Las guardas de Haskell en Prolog son un átomo más

```
/*Haskell*/
fibonacci(n) | n>1 = fibonacci(n-1) + fibonacci(n-2)
```



```
fibonacci(N,M) :- N>1, N1 is N-1, N2 is N-2, fibonacci(N1,F1),
fibonacci(N2,F2), M is F1+F2
```

## 3. En Haskell podíamos anidar llamadas a función, en Prolog se hace paso a paso

```
/*Haskell*/
fibonacci(n) | n>1 = fibonacci(n-1) + fibonacci(n-2)
```

Aquí hay 3 niveles de anidamiento

```
/*Prolog*/
fibonacci(N,M) :- N>1, N1 is N-1, N2 is N-2, fibonacci(N1,F1),
fibonacci(N2,F2), M is F1+F2
```

1º Calcular  $n-1$  y  $n-2$

2º Llamadas a fibo.

3º Realizaremos la suma  $\rightarrow N$  parámetro de salida

### Longitud de una lista

Prolog → `length([],0).`  
`length([_|T],N) :- length(T,N1), N is N1+1.`

### Concatenación de listas

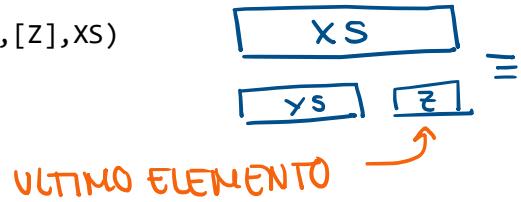
Prolog → `append([],Y,Y).`  
`append([X|R], Y, [Z]) :- append(R,Y,RY), Z = [X|RY].`

mejor → `append([],Y,Y).`  
`append([X|R], Y, [X|RY]) :- append(R,Y,RY).`

### Último elemento de una lista

Prolog → `last([X], X).`  
`last([X,Y|XS], Z) :- last([Y|XS],Z).`

con append → `last(XS, Z) :- append(YS,[Z],XS)`



## 3. EL MODELO DE COMPUTACIÓN DE LA PROGRAMACIÓN LÓGICA

Se basa en la regla de inferencia conocida como **Resolución**

- ✓ Para ejecutar la llamada a procedimiento **A** (átomo) tenemos dos casos →
  - Si la llamada a procedimiento **A** **unifica** (hace matching) con un hecho **A<sub>0</sub>**, finalizamos, la llamada a tenido éxito
  - Si **A** **unifica** con la cabeza de una regla **A<sub>0</sub> :- A<sub>1</sub>, ..., A<sub>n</sub>**, no hemos finalizado, reemplazamos el átomo por los del cuerpo de la regla, debemos proceder a chequear de la misma forma desde **A<sub>1</sub>** hasta **A<sub>n</sub>**.

### 3.1 Sustituciones. Composición de sustituciones

#### 3.2 Unificación (paso de parámetros bidireccional)

**Unificar** dos expresiones **A** y **B** (dos átomos, estructuras, constantes...) consiste en encontrar una sustitución "sigma" para sus variables que las haga idénticas.

**A sigma = B sigma** → **A** y **B** sean iguales aplicándoles la sustitución sigma

**f(X, b)** y **f(a, Y)** **unifican con la sustitución** {**X/a, Y/b**}

$$\begin{aligned} f(X, b)\{X/a, Y/b\} &= f(a, b) \quad \text{> iguales} \\ f(a, Y)\{X/a, Y/b\} &= f(a, b) \end{aligned}$$

Que casos podremos encontrar →

	Variable X	Constante c	Estructura $f(t_1, \dots, t_n)$
Variable $X'$	Sí $\{X/X'\}$ X toma el valor de $X'$ (o viceversa)	Sí $\{X'/c\}$ X toma el valor de c	Sí $\{X' / f(t_1, \dots, t_n)\}$ Siempre que no ocurra "occur check"
Constante $c'$		Sí solo si $c = c'$	NO
Estructura $f(t'_1, \dots, t'_m)$		NO	Sí solo si $f = f'$ , $n = m$ , $t_i$ unificará con $t'_i$ ; etc

**Occur check** → X y  $f(X)$  no unifican

Con al sustitución  $\{X/a\}$

$$\left\{ \begin{array}{l} X\{X/a\} = a \\ f(X)\{X/a\} = f(a) \end{array} \right. \neq$$

Una variable NUNCA unificará con una expresión que contenga dicha variable

### 3.3 Unificación de listas

Las listas deben **estar en el mismo formato** antes de unificar para que sea más sencillo.

Ejemplo:

- $[a, b]$  unifica con  $[X|R]$  usando  $\{X/a, R/[b]\}$        $\left[ \begin{array}{l} [a|[b]] \{X/a, R/[b]\} = [a|[b]] \\ [a, b] \text{ se puede escribir como } [a|[b]] \end{array} \right] =$
- $[a]$  unifica con  $[X|R]$  usando  $\{X/a, R/[]\}$        $\left[ \begin{array}{l} [a|[]] \{X/a, R/[]\} = [a|[]] \\ [a] \text{ se puede escribir como } [a|[]] \end{array} \right] =$

### 3.4 MGU (unificador más general)

Dadas dos expresiones que unifican, muchas veces hay infinitos unificadores, calculamos el **MGU**, la sustitución más práctica y que menos valores introduce en las dos expresiones

¿CÓMO SE CALCULA?

1. Dadas dos expresiones,  $t_1$  y  $t_2$  y una de ellas es una variable,  $t_1 = X \rightarrow$ 
  - $MGU = \{X/t_2\}$
  - Excepción: si  $t_1 = t_2 = X \quad MGU = \{\}$  Sustitución vacía (si son iguales  $t_1$  y  $t_2$ )
  - Excepción: si  $t_2$  NO es una variables y  $X$  aparece en  $t_2$  OCCUR CHECK

# Te regalamos

**15€**



**1/6**

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.

**1**  
Abre tu Cuenta Online sin comisiones ni condiciones

**2**  
Haz una compra igual o superior a 15€ con tu nueva tarjeta

**3**  
BBVA te devuelve un máximo de 15€

Isabel Vallés Bertomeu

LTP

Tema 4

2. Dadas dos expresiones de la forma  $p(t_1, \dots, t_n) \text{ y } q(s_1, \dots, s_m) \rightarrow$ 
  - a. Comprobamos que  $p = q$  y  $n = m$  (mismo símbolo de función y mismo número de argumentos) si no, **fallo**.
  - b. Recursivamente unificamos  $t_1$  y  $s_1$ ,  $t_2$  y  $s_2$ , ...,  $t_n$  y  $s_m$ .
  - c. Cada vez que unifiquemos obtendremos una sustitución  $\theta$  que aplicaremos al resto de términos que queden y al del MGU previo, antes de seguir con la unificación de  $t_2$  y  $s_2$ , ...,  $t_n$  y  $s_m$ .
  - d. Si alguna unificación falla, terminamos con **fallo**
  - e. Si llegamos al final sin fallo (las 2 expresiones serán ahora iguales), la unión de todos los  $\theta_i$  es el **MGU** de las expresiones

## EJEMPLO:

Calcular el MGU de  $p([X, c], X)$  y  $p([f(Y)|R], f(a))$

1. Listas en el mismo formato  $\rightarrow p([X|c], X)$   
 $p([f(Y)|R], f(a))$

2. Comprobamos que el que el predicador y el n.º de argumentos coincide (aridad)

$$p = p \text{ y } n = m = 2$$

3. Comenzamos a calcular unificadores de izquierda a derecha:

$$p([X|c], X) \quad \begin{matrix} 1^{\text{o}} \text{ Argumento } [X|c] \text{ y } [f(Y)|R] \text{ unifica con } \{X/f(Y), R/c\} \\ p([f(Y)|R], f(a)) \quad \text{la } X \text{ toma el valor } f(Y) \text{ y la } R \text{ el de } c \end{matrix}$$

Aplicamos la sustitución a todos los términos

$$p([f(Y)|c], f(Y))$$

$$p([f(Y)|c], f(a))$$

**2º Argumento**  $f(Y)$  y  $f(a)$  unifica con  $\{Y/a\}$

Aplicamos la sustitución a todos los términos y el MGU previo

$$p([f(a)|c], f(a)) \quad \{X/f(a), R/c\}$$

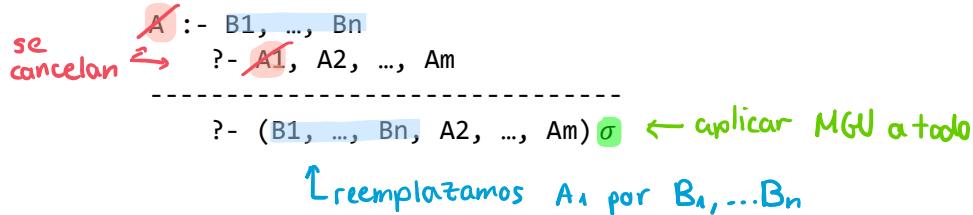
$$p([f(a)|c], f(a))$$

$$\text{MGU} = \{X/f(a), R/c, Y/a\}$$

### 3.5 Resolución - Modelo de ejecución de Prolog

Dado un programa lógico P y un objetivo ?-A<sub>1</sub>, ..., A<sub>m</sub> (el cual contiene la secuencia de átomo/funciones que queremos resolver)

Si el programa contiene una cláusula A :- B<sub>1</sub>, ..., B<sub>n</sub> y la cabeza **A** unifica con el átomo A<sub>1</sub> de la cláusula con un MGU σ, la regla de resolución genera el nuevo objetivo →



- Este procedimiento puede ser **indeterminista**, si el átomo A<sub>1</sub> unifica con la cabeza de varias cláusulas, tenemos **varias alternativas**, hay que explorarlas todas.
- Si A<sub>1</sub> unifica con un hecho, no se reemplaza por nada, solo se elimina.
- La aplicación sucesiva de esta regla genera un **árbol de búsqueda**.
- Una computación o derivación es una **secuencia encadenada de pasos de resolución** y se corresponde con cada una de **las ramas del árbol**.

### 3.6 Árbol de Búsqueda

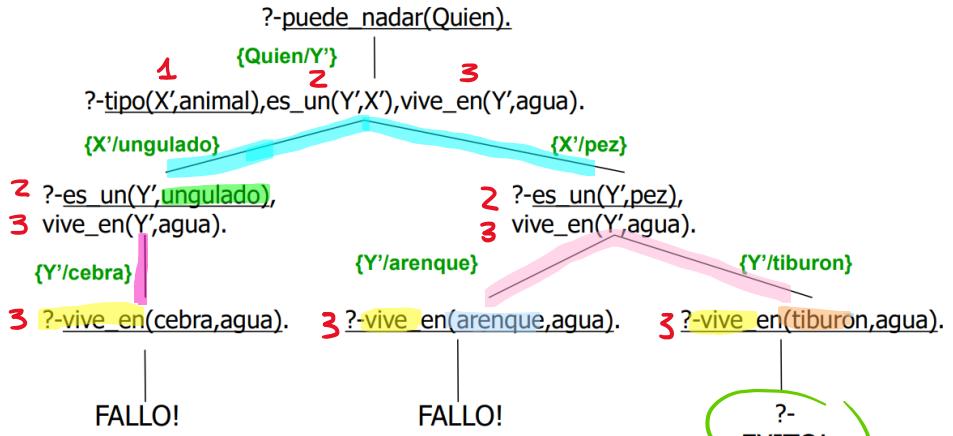
#### PROGRAMA

```

PROGRAMA
-----
tipo(ungulado, animal).
tipo(pez, animal).
es_un(cebra, ungado).
es_un(arenque, pez).
es_un(tiburon, pez).

vive_en(cebra, tierra).
vive_en(rana, tierra).
vive_en(rana, agua).
vive_en(tiburon, agua).

puede_nadar(Y) :-
  tipo(X, animal), es_un(Y, X),
  vive_en(Y, agua).
  
```



'Quien' por lo tanto toma el valor de tiburón para que haya éxito

### 3.7 Tipos de computación

- **Finita:** la computación termina en un número finito de pasos.
  - ✓ De fallo: ninguna cláusula unifica con el átomo seleccionado A1
  - ✓ De éxito: termina en la cláusula vacía (?-). También se les llama refutación.

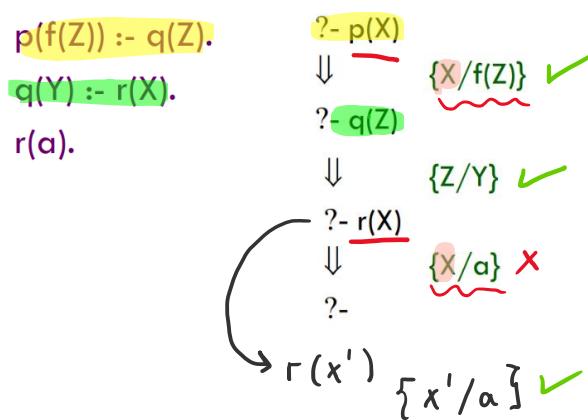
Cada rama de éxito produce una respuesta computada, se obtiene componiendo los MGUs de cada uno de los pasos de esa rama.
- **Infinita:** en cualquier objetivo de la secuencia, el átomo seleccionado A1 unifica con una cláusula del programa, bucle infinito.

### 3.8 Tipos de derivaciones

<u>INFINITA</u>	<u>DE FALLO</u>	<u>DE ÉXITO</u>
$\{p(f(X)) \leftarrow p(X).\}$ <i>En cada paso renombramos las variables (<math>x, x', x'' \dots</math>)</i> $\begin{aligned} &? \leftarrow p(X) \\ &\quad \Downarrow \{X/f(X')\} \\ &? \leftarrow p(X') \\ &\quad \Downarrow \{X'/f(X'')\} \\ &? \leftarrow p(X'') \\ &\quad \Downarrow \{X''/f(X''')\} \\ &? \leftarrow p(X''') \\ &\quad \Downarrow (\infty) \end{aligned}$	$\{p(0) \leftarrow q(X).\}$ $\begin{aligned} &? \leftarrow p(Z) \\ &\quad \Downarrow \{Z/0\} \\ &? \leftarrow q(X) \\ &\quad \Downarrow \text{falló} \end{aligned}$ <p><i>No unifica con la cabeza de ninguna cláusula</i></p>	$\{p(0) \leftarrow q(X).\}$ $\begin{aligned} &? \leftarrow p(Z) \\ &\quad \Downarrow \{Z/0\} \\ &? \leftarrow q(X) \\ &\quad \Downarrow \{X/1\} \end{aligned}$ <p><i>?- éxito!</i></p>

### 3.9 La importancia del renombramiento

Ejemplo:



Las variables de un procedimiento son LOCALES al procedimiento por lo que no puede coincidir con una variable GLOBAL que haya aparecido antes.

**SOLUCIÓN** → Renombrar las variables de las cláusulas

### 3.10 Búsqueda Predefinida

La regla de búsqueda determina:

1. El orden en que se ensayan las cláusulas del programa
2. La estrategia con que se recorre el árbol resultante.

Hay dos estrategias:

- ✓ **Profundidad:** se pierde la completitud del procedimiento de resolución cuando una rama es infinita y a la derecha hay posibles respuestas.
- ✓ **Anchura:** se recorre el árbol por niveles. Se mantiene la completitud, aunque es muy costosa.

PROLOG:

- 1) top-down → de arriba hacia abajo
- 2) búsqueda en **profundidad** con vuelta atrás (backtracking).

## 4. ALGUNAS CUESTIONES PRÁCTICAS

### 4.1 Aplicaciones de la PL

1. Verificación de software y hardware
2. Certificación de programas
3. Prototipado automático
4. Ingeniería del software automática (depuración automática, síntesis de programas a partir de especificaciones, transformación de programas,...)
5. Modelización en Sistemas de Información y Bases de Datos
6. Problemas de Aprendizaje
7. Robótica y Planificación
8. Sistemas Expertos
9. Tratamiento de Lenguaje Natural

Mucha suerte con el examen :). Sígueme en ig: [@isaaa.png](#)