

PROLOG

PROgramación en LÓGica

Lógica para Ciencias de la Computación

Primer Cuatrimestre de 2008

– Material Adicional –

Repaso

- ❖ En el paradigma de la programación en lógica sólo se debe **describir** el problema considerado.
- ❖ La solución será encontrada al **interpretar** el programa lógico asociado.
- ❖ Un programa lógico describe un conjunto de **relaciones entre objetos**.
- ❖ Se compone de **hechos y reglas**.
- ❖ Las consultas permiten **explorar las relaciones** definidas por un programa lógico.

Sintaxis Formal

programa ::= cláusula . | cláusula . programa
cláusula ::= predicado | predicado :- predicados
predicados ::= predicado | predicado , predicados
predicado ::= nombre | nombre (términos)
términos ::= término | término , términos
término ::= nombre | variable | lista | estructura
estructura ::= nombre (términos)
lista ::= [términos] | [términos | lista] | []

padre_de(homero, bart).

Programa

hermanos(X,Y):- padre_de(P, X), padre_de(P, Y).

Estructuras y Listas

- ❖ Las estructuras se asemejan a los predicados en su forma, pero no en su rol:
 - ➡ Las estructuras son **objetos**, y por lo tanto aparecen como argumentos a predicados.
- ❖ Las estructuras **pueden** contener estructuras.
- ❖ Las **listas** son un tipo especial de estructura.

Estructuras

```
cliente (homero, simpson, av_s_viva, 742).
```



```
cliente (nom(homero, simpson),  
         calle(av_s_viva), nro(742) ).
```



```
cliente (nom(homero, simpson),  
         dir(calle(av_s_viva), nro(742) ))
```

Sustitución

Def.: Una **sustitución** es un conjunto finito de pares ordenados $[X_n, T_n]$ donde X_n es una variable y T_n es un término, tales que:

- ➡ para cada $i \neq j$, se verifica que $X_i \neq X_j$, y
- ➡ para todo valor posible de i, j , se debe cumplir que X_i no aparezca en T_j .

Terminología: Aplicar una sustitución consiste en efectuar los reemplazos correspondientes.

Ejemplos

$$\begin{array}{ll}\theta_1 = \{ [X/a], [Y/b] \} & s(X) \theta_1 \\ \theta_2 = \{ [X/a], [X/b] \} & = s(a) \\ \theta_3 = \{ [X/Z], [Y/a] \} & p(Y) \theta_1 \\ \theta_4 = \{ [a/b], [b/a] \} & = p(b) \\ \theta_5 = \{ [X/a], [Y/X] \} & p(s(X)) \theta_1 \\ & = p(s(a))\end{array}$$

¡Sólo θ_1 y θ_3 respetan la definición anterior!

Instanciación

Def.: Sean A y B sendos términos. Diremos que A es una **instancia** de B si, y sólo si, existe una sustitución θ tal que $A = B\theta$.

$$\begin{array}{ccc} s(X) & \theta = \{[X/a]\} & \\ \downarrow & & \\ s(a) & & \end{array}$$

$s(a) = s(X)\theta$

$$\begin{array}{cc} t(X) & t(Y) \\ \downarrow & \downarrow \\ t(Y) & t(X) \end{array}$$

Instanciación

Resumiendo, sean A y B sendos términos:

- ➡ A puede ser instancia de B , pero B no ser instancia de A .
- ➡ A puede ser instancia de B y B instancia de A .
- ➡ B puede ser instancia de A , pero A no ser instancia de B .
- ➡ A puede no ser instancia de B , ni B de A .

A	B
$s(a)$	$s(X)$
$s(U)$	$s(V)$
Idem al primero	
$s(a)$	$s(b)$
$f(X,b)$	$f(a, Y)$

Notemos que si A es una instancia de B , B es **más general** que A .

Intérprete Abstracto

- La ejecución de cualquier programa Prolog puede ser simulada mediante un intérprete abstracto, donde:
 - ➡ ENTRADA: Un programa lógico P y una consulta fija G .
 - ➡ SALIDA: **YES** si G es satisfecha por P ; **NO** en caso contrario.
- **Terminología**: Una consulta fija es una consulta sin variables.

¿Qué respondería PROLOG?

1. $p(a) :- q(a), r(b).$
2. $q(a).$
3. $r(b).$

?- $p(a).$

yes

- ¿Como nos damos cuenta que $p(a)$ es satisfecha por el programa?
- Para el siguiente algoritmo tener presente que un hecho puede verse como una regla con cuerpo **vacío**.

Intérprete Abstracto

1. Inicializar **RESOLVENTES** en G ;
2. Mientras **RESOLVENTES** no esté vacío, hacer lo siguiente:
 1. Elegir una meta A de **RESOLVENTES**;
 2. Elegir una instancia fija $A' \leftarrow B_1, B_2, \dots, B_n$ de alguna regla presente en P , tal que A y A' sean idénticos (en caso de que no haya regla alguna en estas condiciones, salir del mientras);
 3. Reemplazar a A por $B_1, B_2, \dots, B_n$ en **RESOLVENTES**;
3. Si **RESOLVENTES** está vacío, retornar **YES**; caso contrario, retornar **NO**;

Trazas

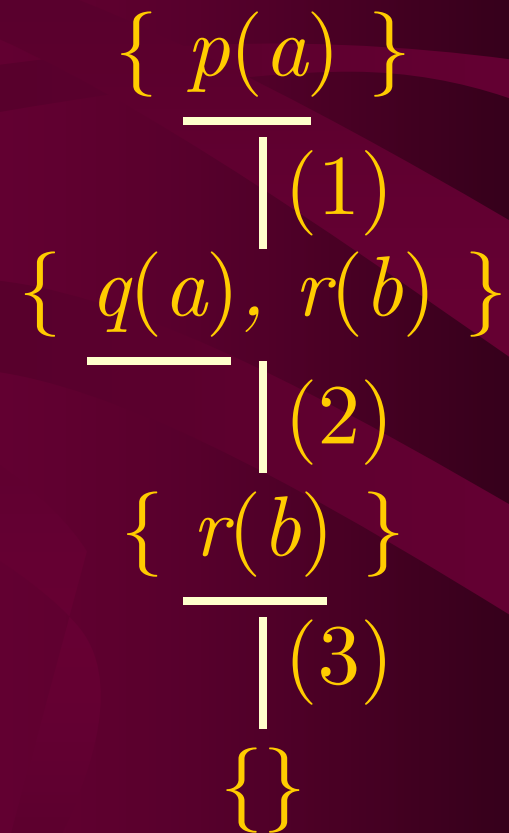
1. $p(a) :- q(a), r(b).$

2. $q(a).$

3. $r(b).$

?- $p(a).$

yes



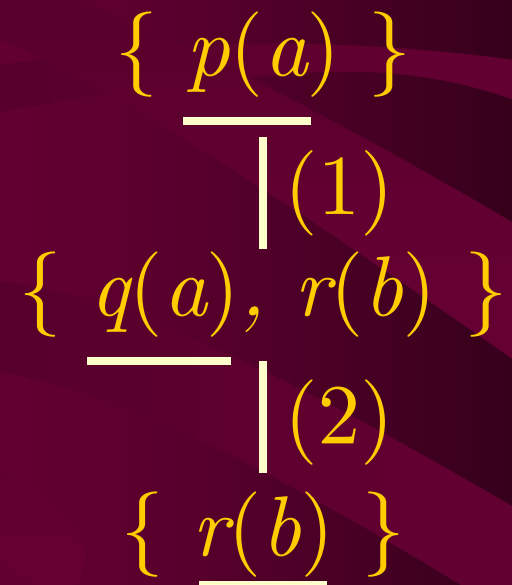
Trazas

1. $p(a) \text{ :- } q(a), r(b).$

2. $q(a).$

?- $p(a).$

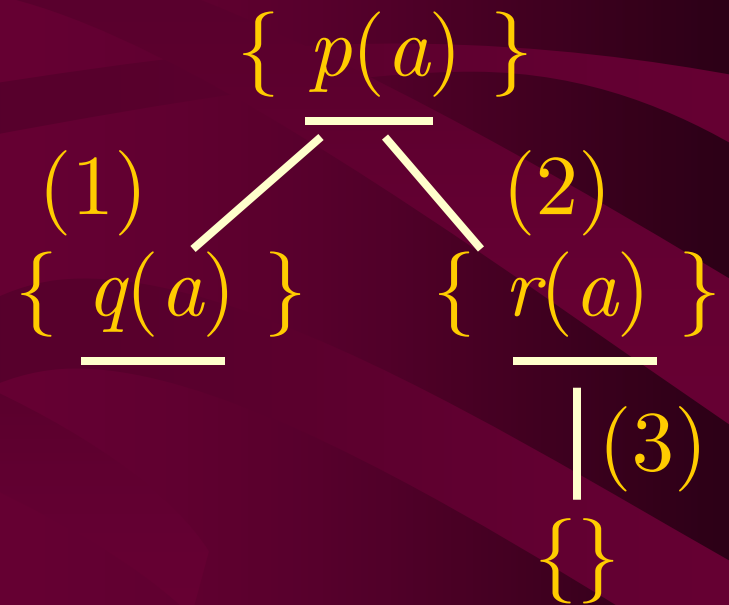
no



Trazas

1. $p(a) :- q(a).$
2. $p(a) :- r(a).$
3. $r(a).$

?- $p(a).$
yes



predicado $\neq/2$

- El predicado predefinido $\neq/2$ denota la relación “son iguales”.

Ejemplos

```
?-  $\neq(a, a)$ .
```

yes

```
?-  $\neq(a, b)$ .
```

no

- Para comodidad del programador, Prolog permite emplear notación infija para $\neq/2$.

```
?-  $a = a$ .
```

yes

```
?-  $a = b$ .
```

no

predicado =/2

- El predicado predefinido `=/2` denota la relación “son iguales”.
- ¿Pero cual es el significado de la siguiente consulta y qué responde Prolog frente a ella?

`?- f(X) = f(a) .`

`X = a`

¿ $f(X)$ y $f(a)$ son iguales?

Si, cuando $X = a$.

predicado =/2: ejemplos

?- f(Y) = f(g(a)).

Y = g(a)

?- f(a) = f(g(X)).

no

?- h(X,b) = h(a,Y).

X = a

Y = b

?- h(X,X) = h(a,b).

no

?- X = Y.

PROBARLO Y PREGUNTAR

predicado \=/2

- Tiene éxito cuando sus argumentos NO pueden hacerse iguales.
- Es decir, devuelve **yes** cada vez que **=/2** devuelve **no** y viceversa.

```
?- a \= a.
```

no

```
?- a \= b.
```

yes

```
?- X \= a.
```

no

- Observar que solo responde yes o no!!!

Expresiones numéricas y predicado is/2

- En Prolog, las expresiones numéricas se representan mediante *estructuras*.
- A continuación se muestra una BNF describiendo la sintáxis de las expresiones:

$E ::= \text{Numero} \mid +(E_1, E_2) \mid -(E_1, E_2) \mid *(E_1, E_2) \mid \dots$

$+(2, 3)$

$*(5, +(3, 2))$

$-(*(5, 8), 10)$

Para comodidad del programador, Prolog admite el uso de notación infija.

$2+3$

$5*(3+2)$

$5*8-10$

predicado is/2

- Sean T y E términos, donde E representa una expresión numérica. La consulta

?- T is E

permite verificar si **T** es igual al resultado de evaluar **E**

T is E

es
equivalente a

T = *res_eval_E*

predicado is/2: ejemplos

?- 5 is 2+3.

yes

?- 8 is 2+3.

no

?- R is 5*8.

R = 40

yes

?- W is [1,2,3]

ERROR

?- R is 2+X.

ERROR



FIN