Programación en PROLOG(1)

Inteligencia Artificial
2º cuatrimestre de 2009

Departamento de Ciencias e Ingeniería de la Computación

Universidad Nacional del Sur

Prolog

- Es el representante más conocido del paradigma lógico.
- Prolog permite:
  - Definir/especificar relaciones entre objetos.
  - Verificar si ciertos objetos están relacionados entre sí.
- A continuación veremos cómo...

Sintaxis Informal

nombres de relaciones (también llamadas predicados) y objetos.

secuencias de caracteres comenzando con **minúscula**.

- Prolog cuenta con variables. Las variables denotan objetos sin especificar.

nombres de variables

secuencias de caracteres comenzando con **mayúscula**.

Sintaxis Informal

- Existen 3 tipos construcciones en PROLOG (llamadas cláusulas) :
  - **Hechos**
  - **Reglas**

  Permiten definir relaciones entre objetos.
  - **Consultas**

  Permiten verificar si ciertos objetos están relacionados entre sí

Hechos

- permiten establecer que una determinada tupla de objetos están relacionados bajo una relación en particular.

- Sintaxis:

$$\mathbf{r(obj_1, obj_2,\ldots,obj_n).}$$

"la tupla (obj1, obj2,...,objn) pertenece a la relación r"

Hechos: ejemplos

amigo(homero, barny).

suma(X, 0, X).

socio_club(juan, liniers).

socio_club_liniers(juan).

ateniense(socrates).

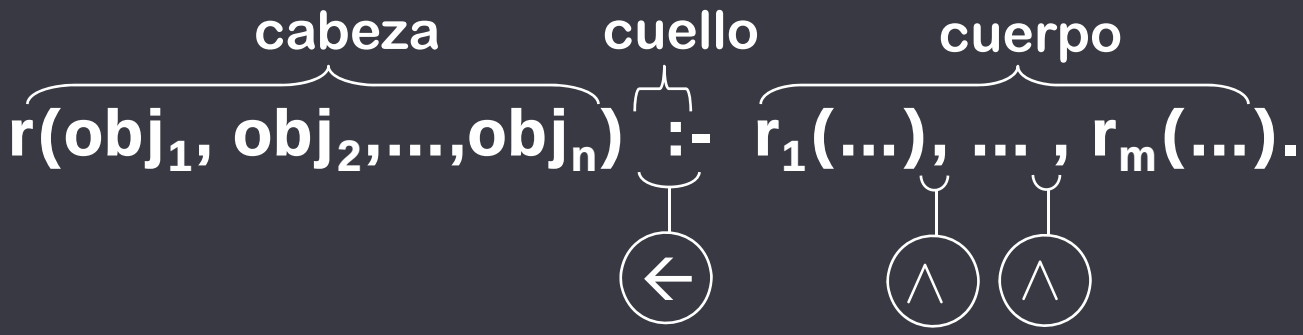
socrates_es_ateniense.

relación de aridad 0: proposición

Reglas

- Permiten establecer que una determinada tupla de objetos están relacionados bajo una relación en particular, pero en términos de otras relaciones.

- Sintaxis:



Reglas: ejemplos

```
mentiroso(X):- ateniense(X).

gustos_en_comun(X,Y):- gusta(X, Algo),
                       gusta(Y, Algo).

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

Programas Prolog

- Un programa **Prolog** es un conjunto de hechos y reglas.

Ej:

```
progenitor(homero, bart).
progenitor(homero, lisa).
progenitor(abraham, homero).
abuelo(X,Y):- progenitor(X,Z), progenitor(Z,Y).
```

definición del predicado progenitor/2

definición del predicado abuelo/2

Consultas

- permiten verificar si una determinada tupla de objetos están relacionados bajo una relación en partircular.

- Sintaxis:

?- r(obj$_1$, obj$_2$,...,obj$_n$).

- Ej:

?- abuelo(abraham, lisa)

?- abuelo(X, bart) ¿bart tiene abuelo?

Consultas y Respuestas

```
progenitor(homero, bart).

progenitor(homero, lisa).

progenitor(abraham, homero).

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

```
?- progenitor(homero, lisa).
yes

?- progenitor(homero, abraham).
no

?- progenitor(marge, lisa).
no
```

```
?- abuelo(abraham, bart).
yes

?- abuelo(abraham, X).
X = bart;
X = lisa;
no
```

¿abraham tiene algún nieto?

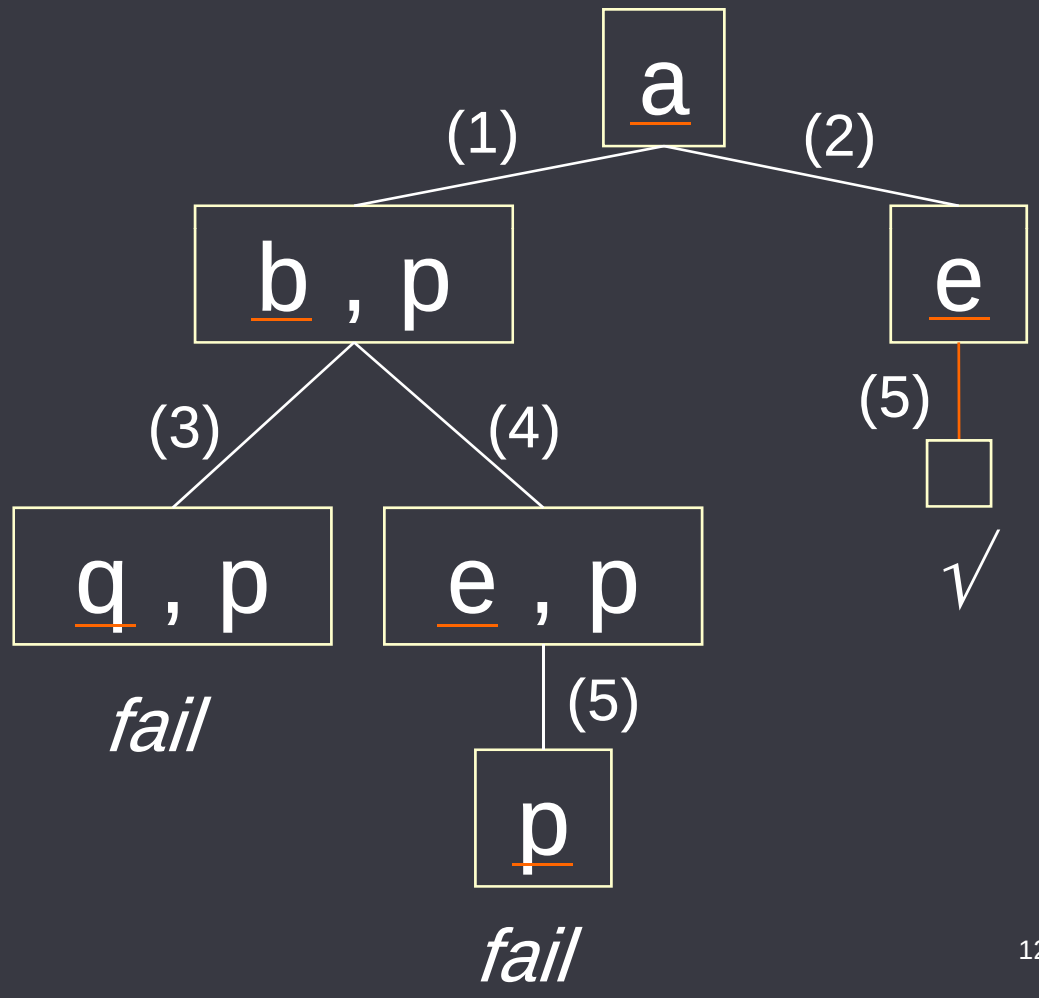
Resolución SLD y Backtracking

Resolución de la meta “? – a.”

1) a : – b, p.
2) a : – e.
3) b : – q.
4) b : – e.
5) e.

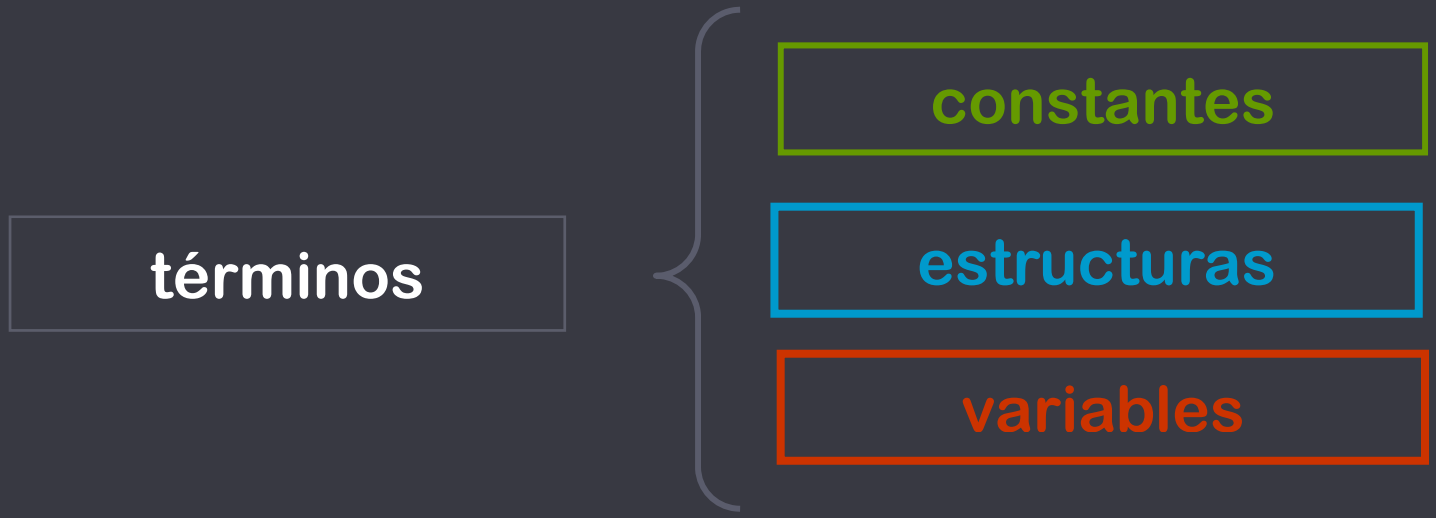
? – a.

yes

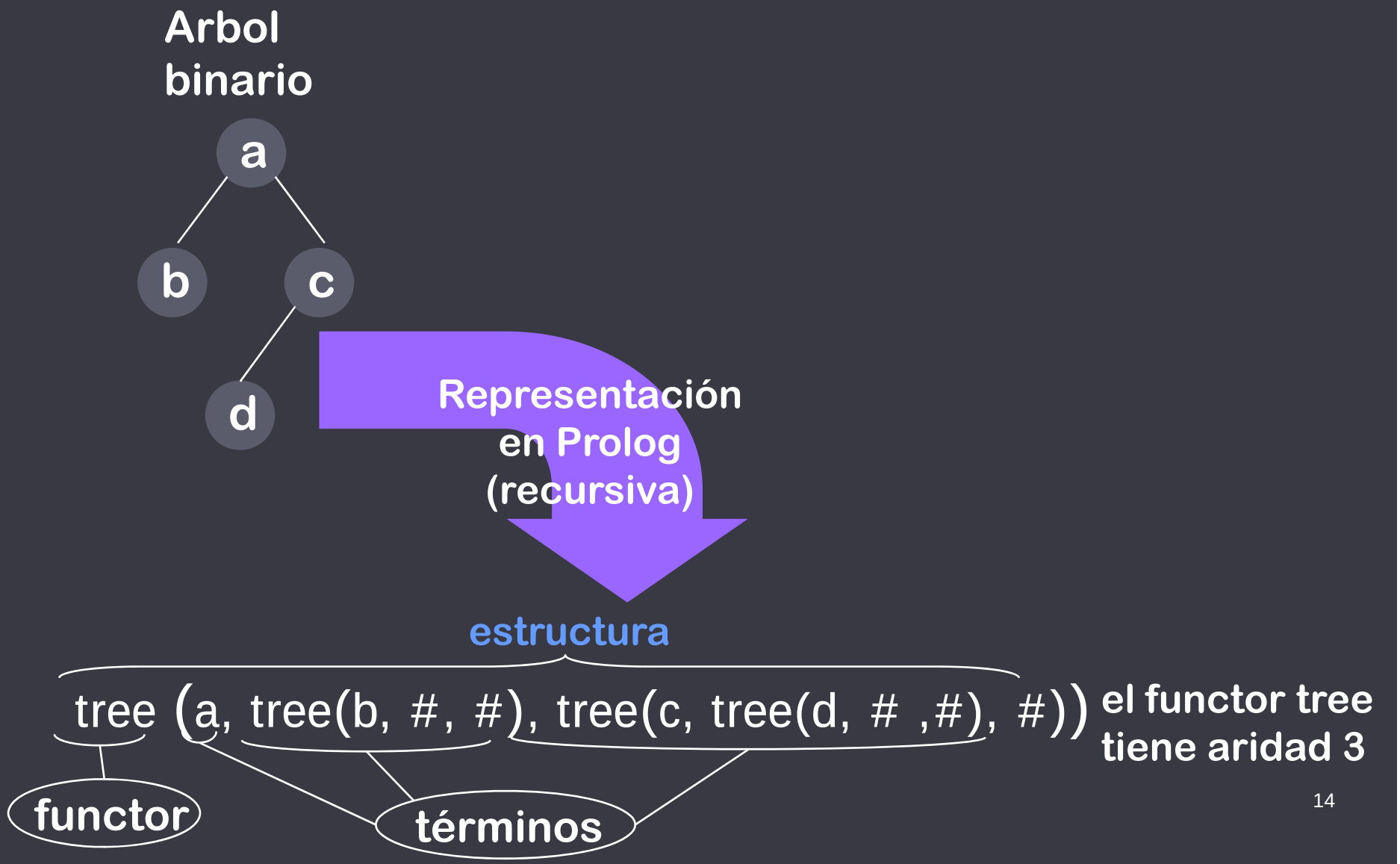


(Tipos de) Términos

- Hasta ahora solo utilizamos 2 tipos de términos ...



Estructuras: Rep. de Arboles Binarios



Estructuras: Rep. de Arboles Binarios

- A continuación se define el predicado arbolBin/1 que usa la estructura para árboles binarios.

```
% arbolBin(+Termino)
% Recibe como argumento un término y determina
% si éste corresponde a un árbol binario válido.

arbolBin(#).
arbolBin(tree(R,Hi,Hd)):- arbolBin(Hi),
                          arbolBin(Hd).
```

```
?- arbolBin(tree(a,tree(b,#,#),#)).
yes
?- arbolBin(tree(a, a, #)).
no
```

Estructuras: Rep. de Arboles Binarios

- Otro predicado que manipula árboles binarios.

```
% altura(+Bin, -Alt)
% Recibe como argumento un árbol binario Bin
% y retorna su altura Alt.

altura(#, -1).
altura(tree(R,Hi,Hd), A):- altura(Hi, AHi),
                           altura(Hd, AHd),
                           max(AHi, AHd, MA),
                           suma(MA, 1, A).
```

```
?- altura(tree(a,tree(b,#,#),#), A).
A = 1
```

Estructuras: Rep. de Registros

Nombre	Bart
Apellido	Simpson
Dirección	**Calle**: Av. Siempre Viva **Nro**: 742

Registro de datos personales

en Prolog

estructura

datos_pers (nom(bart), ap(simpson), dir(calle(av_s_viva), nro(742)))

functores

Estructuras: Listas (rep. predefinida)

- Las listas también son estructuras, de aridad 2, solo cambia la notación.
- En lugar de: <functor_de_lista>(A,B) se usa [A|B] para representar la lista con cabeza A y cola B.
- Ej: la lista formada por 1, 2 y 3:

[1 | [2 | [3 | []]]]

que PROLOG permite notar abreviadamente de la siguiente forma:

[1, 2 , 3]

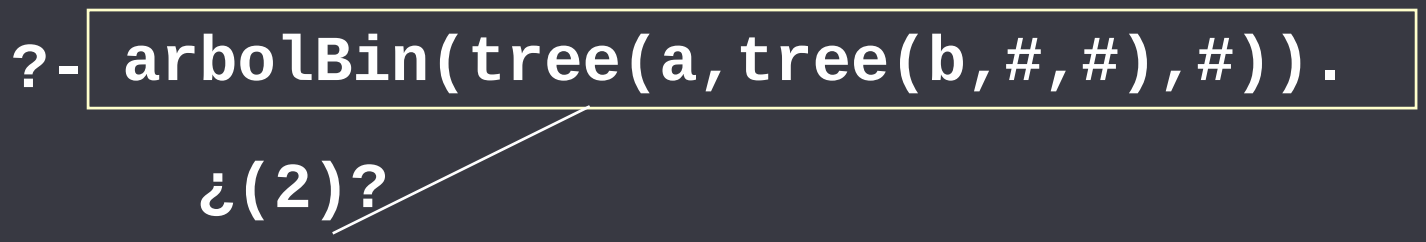
Listas: Ejemplo

- A continuación se define el predicado veces/3 que maipula listas.

```
% veces(+X, +L, -C)
% determina la cantidad C de apariciones de
% un dado elemento X en una dada lista L.

veces(X,[],0).
veces(X,[X|L],C):-
          veces(X,L,C1),
          suma(C1,1,C).
veces(X,[Y|L],C):-
          veces(X,L,C).
```

Resolución SLD y Unificación




```
(1) arbolBin(#).
(2) arbolBin(tree(R,Hi,Hd)):- arbolBin(Hi),
                              arbolBin(Hd).
```

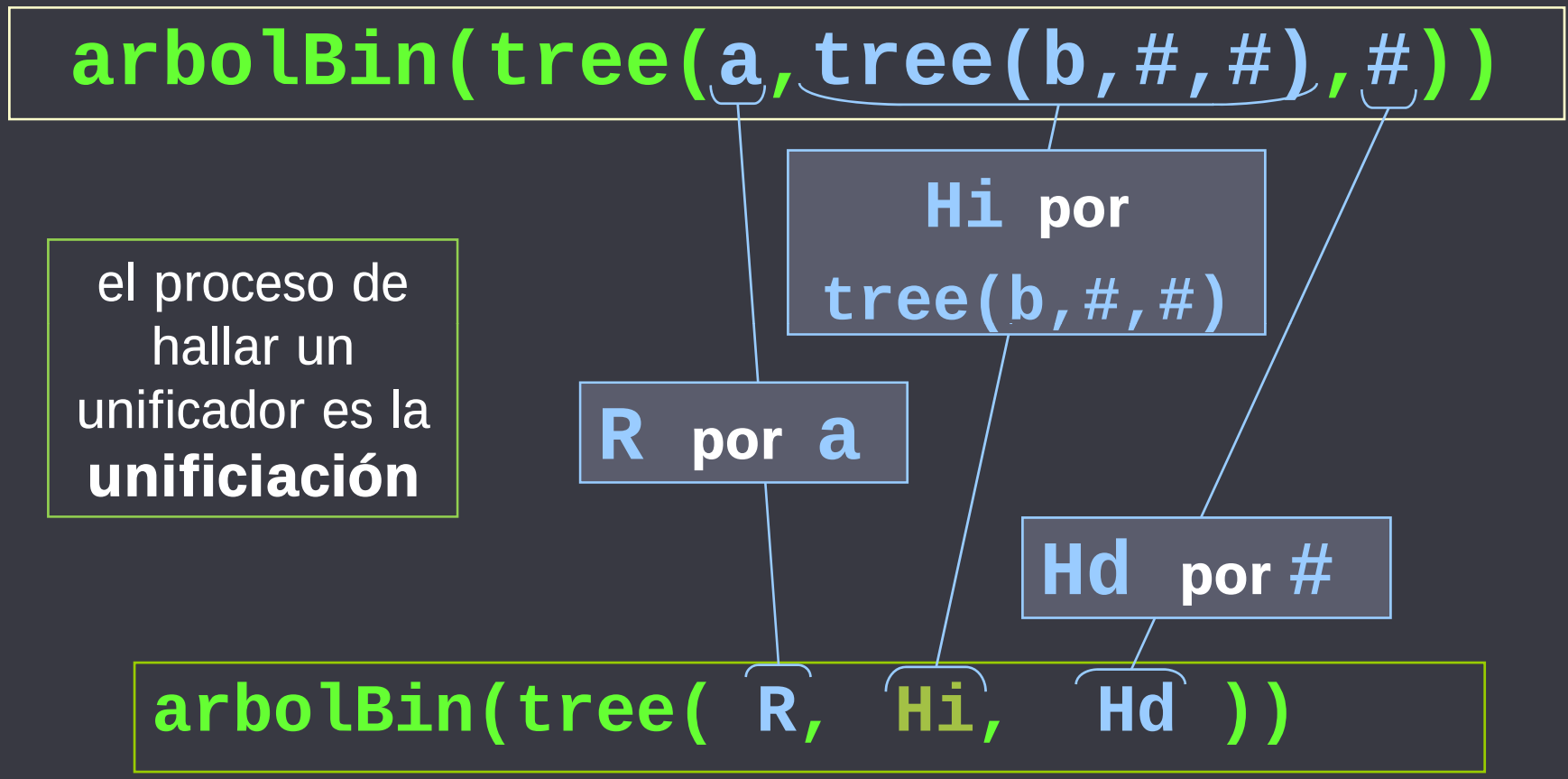
Unificación

```
arbolBin(tree(a,tree(b,#,#),#))
```

¿Existe alguna sustitución de variables que los haga iguales?

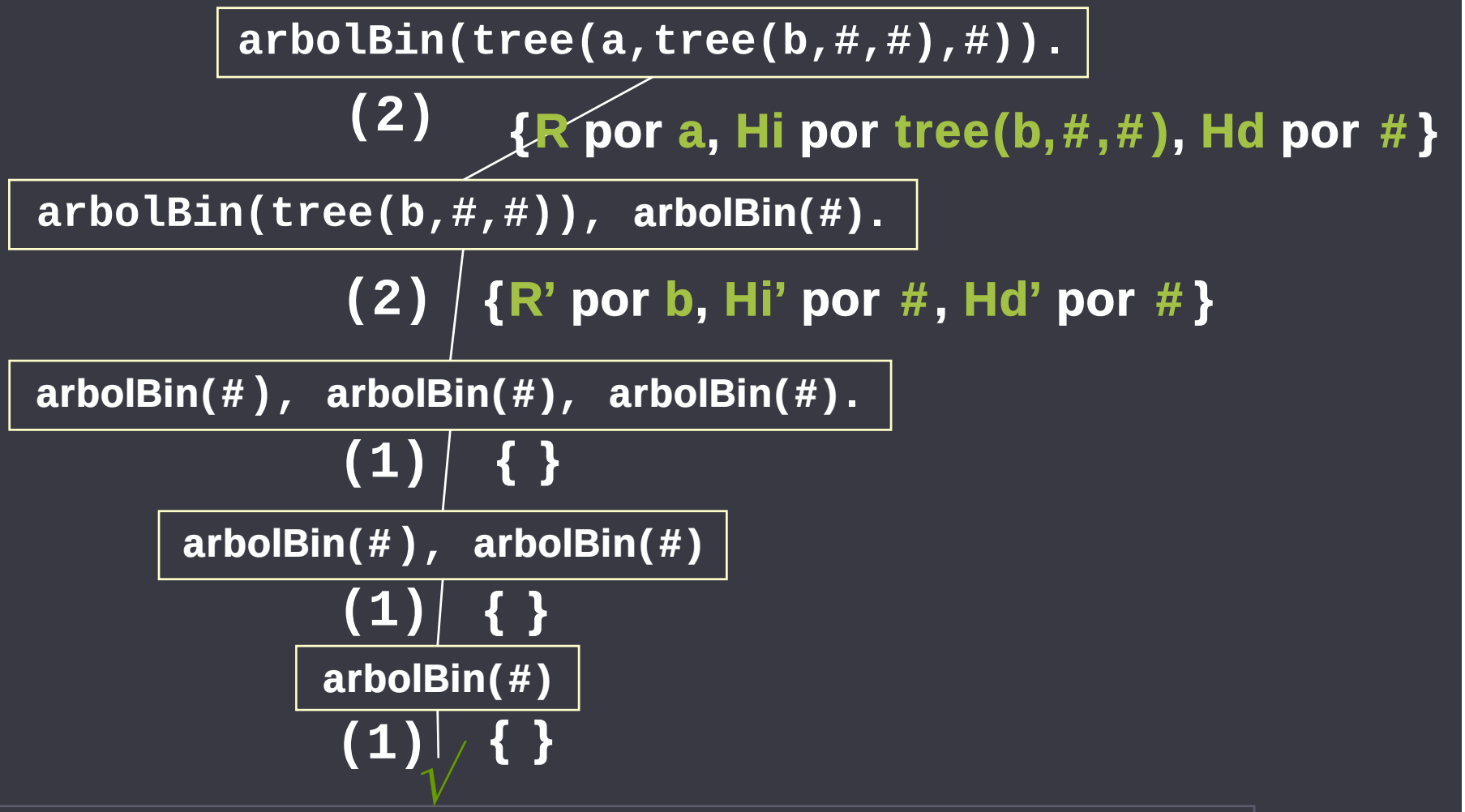
```
arbolBin(tree( R, Hi, Hd ))
```

Unificación



{ R por a, Hi por tree(b,#,#), Hd por # } es una sustitución unificadora, o simplemente unificador, para estos dos átomos.

Unificación




```
(1) arbolBin(#).
(2) arbolBin(tree(R,Hi,Hd)):- arbolBin(Hi),
                              arbolBin(Hd).
```

Predicado de Unificación (=/2)

```
?- X = a.
X = a
yes
```

```
?- 5 = 2+3.
no
```

```
?- X*Y = (2+3)*8.
X = (2+3)
Y = 8
yes
```

Predicado de Unificación (=/2)

```
?- W = f(X), X = a.
W = f(a)
X = a
yes
```

```
?- W = a, W = b.
no
```

Una variable INSTANCIADA ya no se "*desinstancia*"

Predicado “no unifica” (\=/2)

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

```
?- a \= a.
no
```

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

```
?- X \= a.
no
```

Predicado is/2

- La meta “T is E” es exitosa si T unifica con el resultado de la evaluación de la expresión numérica E.

T is E ≡ **T = *res_eval_E***

```
?- 5 is 2+3.
yes
```

```
?- R is 5*8.
R = 40
yes
```

```
?- R = 10, R is 2+3.
no
```

Predicado is/2

- La meta “T is E” es exitosa si T unifica con el resultado de la evaluación de la expresión numérica E.

T is E ≡ **T = *res_eval_E***

```
?- 2+3 is 2+3.
no
```

```
?- R is 2+X.
ERROR
```

```
?- W is [1,2,3]
ERROR
```

Chequeo de variable “singleton”

- Una variable singleton es una variable que aparece sólo una vez en una cláusula.
- En ocasiones, la presencia de una variable singleton manifiesta un error cometido por el programador.

```
concat(Lista, [], List).
concat([X|SubL1], L2,  SubL1_L2):-
      concat(SubL1, L2, SubL1_L2).

miembro(X,[X|L]).

miembro(X,[Y|L]):- miembro(X,L).
```

ERROR

ERROR

OK

OK

Singleton Variable Warning

- La presencia de variables singleton es usualmente una mani-festación de dos clases de errores:
  - olvidó hacer algo que debía hacer con esa variable.
  - cometió un error de nombre al intentar referirse a una variable introducida previamente (ej., error de tipeo).

```
concat(Lista, [], List).
```

error de nombre

```
concat([X|SubL1], L2,  SubL1_L2):-
    concat(SubL1, L2, SubL1_L2).
```

olvidó hacer algo

- Al cargar un archivo conteniendo un programa PROLOG, el SWI muestra un WARNING para avisar al programador de la presencia de variables singleton.

Variables anónimas

- Cuando desea introducirse una variable singleton intencionalmente, como es el caso del predicado miembro/2, deberíamos utilizar variables anónimas.

```
miembro(X,[X|_L]).

miembro(X,[_Y|L]):- miembro(X,L).
```

- De esta forma, estamos indicando al SWI que dichas variables son intencionalmente NO usadas, y así no recibiremos advertencias por ellas.

FIN