Prolog

- Unificación
- Unificación en Prolog
- Búsqueda de pruebas

Objetivos

- Revisar **unificación** de términos
- Discutir la **unificación** en Prolog
  - Mostrar como la unificación en Prolog difiere de la unificación standard
- Explicar al estrategia por la cual Prolog trata de deducir información nueva a partir de información conocida

Unificación

- En ejemplos previos dijimos que Prolog **unifica**

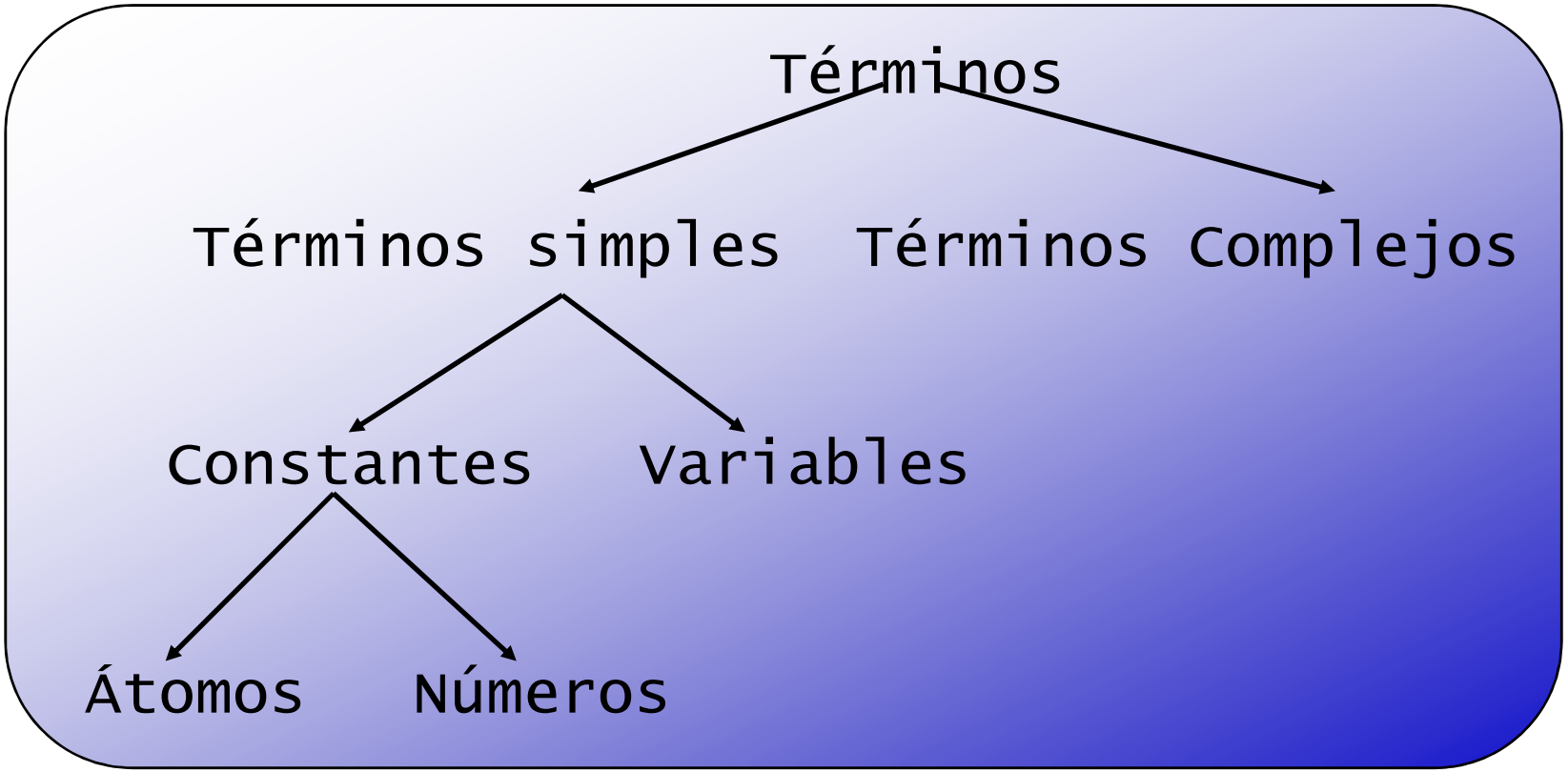
  mujer(X)

  con

  mujer(mia)

  instanciando la variable **X** con el átomo **mia**.

Términos Prolog



Unificación

- Definición (tal como se siguió en los ejemplos)
  - Dos términos unifican si :
    - Son el mismo término (son idénticos)
    - Contienen variables que pueden ser **instanciadas uniformemente** con términos de modo que obtenemos al final términos idénticos

Unificación

- Esto significa que:
  - **mia** y **mia** unifican
  - **42** y **42** unifican
  - **mujer(mia)** y **mujer(mia)** unifican
- Esto significa también que:
  - **vicente** y **mia** no unifican
  - **mujer(mia)** y **mujer(julia)** no unifican

Unificación

- Qué pasa con:
  - **mia** y **X**

Unificación

- Qué pasa con:
  - **mia** y **X**
  - **mujer(Z)** y **mujer(mia)**

Unificación

- ¿Qué pasa con:
  - **mia** y **X**
  - **mujer(Z)** y **mujer(mia)**
  - **quiere(mia,X)** y **quiere(X,vicente)**

Instanciaciones

- Cuando Prolog unifica 2 términos realiza todas las instanciaciones necesarias
- La unificación es un poderoso mecanismo de programación

Definición revisada 1/3

1. Si T_1 y T_2 son constantes, entonces T_1 y T_2 unifican si son idénticos.

Definición revisada 2/3

- Si T_1 y T_2 son constantes, entonces T_1 y T_2 unifican si son idénticos.
- Si T_1 es una variable y T_2 es cualquier tipo de término , entonces T_1 y T_2 unifican, y T_1 se instancia en T_2. (y vice versa)

Definición revisada 3/3

1. Si T_1 y T_2 son constantes, entonces T_1 y T_2 unifican si son idénticos.
2. Si T_1 es una variable y T_2 es cualquier tipo de término , entonces T_1 y T_2 unifican, y T_1 se instancia en T_2. (y vice versa)
3. Si T_1 y T_2 son términos complejos, unifican si:
   a) Tienen el mismo functor y aridad
   b) Todos los argumentos unifican
   c) Las instanciaciones de variables son compatibles.

unificación Prolog =/2

?- mia = mia.

yes

?-

= es un predicado especial (predicado del sistema) de aridad 2 (=/2)

unificación Prolog =/2


```
?- mia = mia.
yes
?- mia = vicente.
no
?-
```

unificación Prolog =/2


```
?- mia = X.
X=mia
yes
?-
```

unificación Prolog =/2

?- X=mia, X=vincent.

Unificación

?- X=mia, X=vicente.

no

?-

Por qué?

Luego de satisfacer el 1er objetivo,
Prolog instanció X con **mia**, de modo que no puede ya unificar con **vincent**.
El segundo objetivo falla.

Ejemplo con términos complejos

?- k(s(g),Y) = k(X,t(k)).

Ejemplo con términos complejos


```
?- k(s(g),Y) = k(X,t(k)).
X=s(g)
Y=t(k)
yes
?-
```

Ejemplo con términos complejos

?- k(s(g),t(k)) = k(X,t(Y)).

Ejemplo con términos complejos


```
?- k(s(g),t(k)) = k(X,t(Y)).
X=s(g)
Y=k
yes
?-
```

Otro ejemplo

?- quiere(X,X) = quiere(marcelo,mia).

Prolog y la unificación

- Prolog no usa un algoritmo standard de unificación
- Considere la siguiente consulta:

  ?- padre(X) = X.

- Unifican?

Términos infinitos

?- padre(X) = X.
X=padre(padre … (X) …..))

En SWI
?- X=padre(X).
X = padre(X).

2 ?- X=padre(X),write(X).
padre(**)
X = padre(X).

Occurs Check

- Un algoritmo standard de unificación incluye un test de ocurrencia (*occurs check*)
- Si tiene que unificar una variable con un término, chequea que la variable **no** ocurra en el término
- En Prolog:

```
?- unify_with_occurs_check(padre(X), X).
no
```

Programando con Unificación

vertical(linea(Punto1,Punto2)) ← la línea definida por Punto1 y Punto2 es vertical

horizontal(linea(Punto1,Punto2)) ← la línea definida por Punto1 y Punto2 es horizontal

Programando con Unificación

vertical(linea(Punto1,Punto2)) ← la línea definida por Punto1 y Punto2 es vertical

horizontal(linea(Punto1,Punto2)) ← la línea definida por Punto1 y Punto2 es horizontal

```
vertical( linea(punto (X,Y),
                punto(X,Z))).
```

Programando con Unificación

vertical(linea(Punto1,Punto2)) ← la línea definida por Punto1 y Punto2 es vertical

horizontal(linea(Punto1,Punto2)) ← la línea definida por Punto1 y Punto2 es horizontal

```
vertical( linea(punto(X,Y),
                punto(X,Z))).

horizontal( linea(punto(X,Y),
                  punto(Z,Y))).
```

Programando con Unificación


```
vertical( linea(punto(X,Y),
                punto(X,Z))).

horizontal( linea(punto(X,Y),
                  punto(Z,Y))).
```

```
?-
```

Programando con Unificación


```
vertical( linea(punto(X,Y),
                punto(X,Z))).

horizontal( linea(punto(X,Y),
                  punto(Z,Y))).
```

```
?- vertical(linea(punto(1,1),punto(1,3))).
yes
?-
```

Programando con Unificación


```
vertical( linea(punto(X,Y),
                punto(X,Z))).

horizontal( linea(punto(X,Y),
                  punto(Z,Y))).
```

```
?- vertical(linea(punto(1,1),punto(1,3))).
yes
?- vertical(linea(punto(1,1),punto(3,2))).
no
?-
```

Programando con Unificación


```
vertical( linea(punto(X,Y),
                punto(X,Z))).

horizontal( linea(punto(X,Y),
                  punto(Z,Y))).
```

```
?- horizontal(linea(punto(1,1),punto(1,Y))).
Y = 1;
no
?-
```

Programando con Unificación


```
vertical( linea(punto(X,Y),
                punto(X,Z))).

horizontal( linea(punto(X,Y),
                  punto(Z,Y))).
```

```
?- horizontal(linea(punto(2,3),Punto)).
Punto = punto(_554,3);
no
?-
```

Búsqueda de pruebas

- Prolog tiene un procedimiento específico para buscar respuestas a una consulta.
- Sigue un orden predefinido en:
  - Siguiente predicado en la consulta a satisfacer
  - Cláusula en el programa a utilizar

Ejemplo

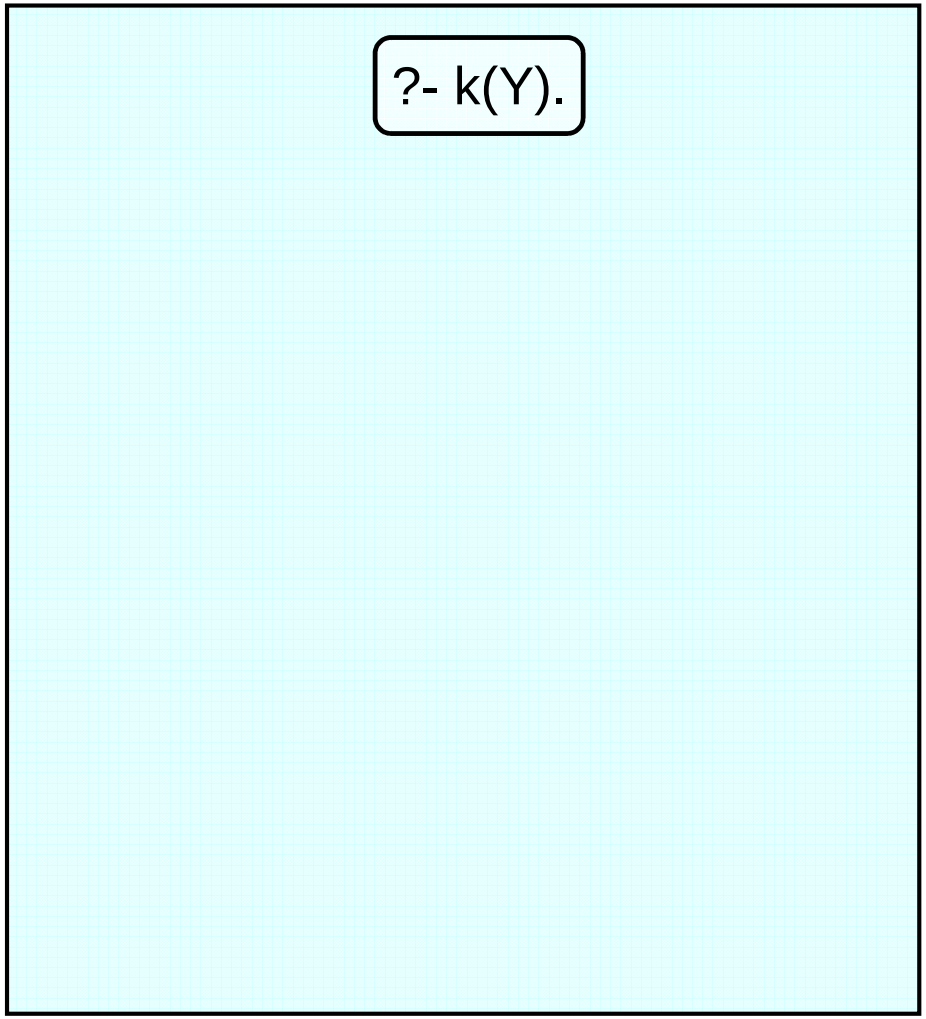

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
```

Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

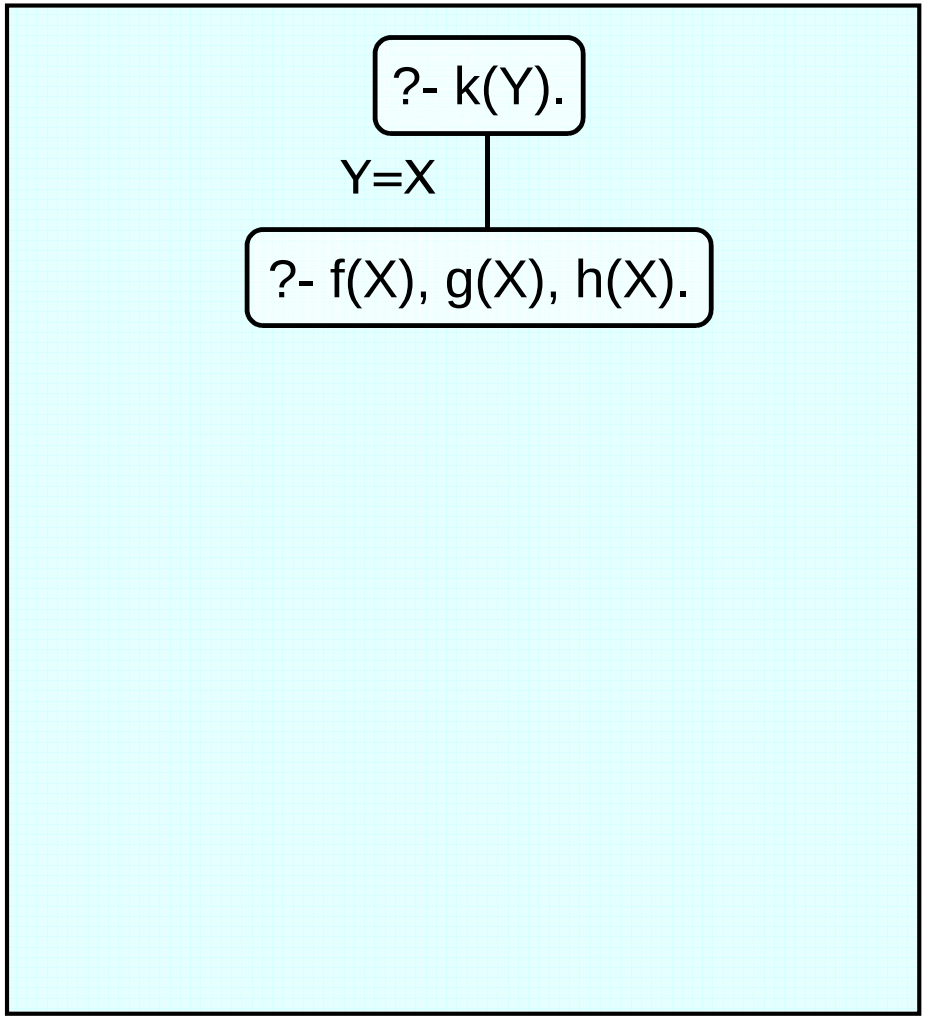
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

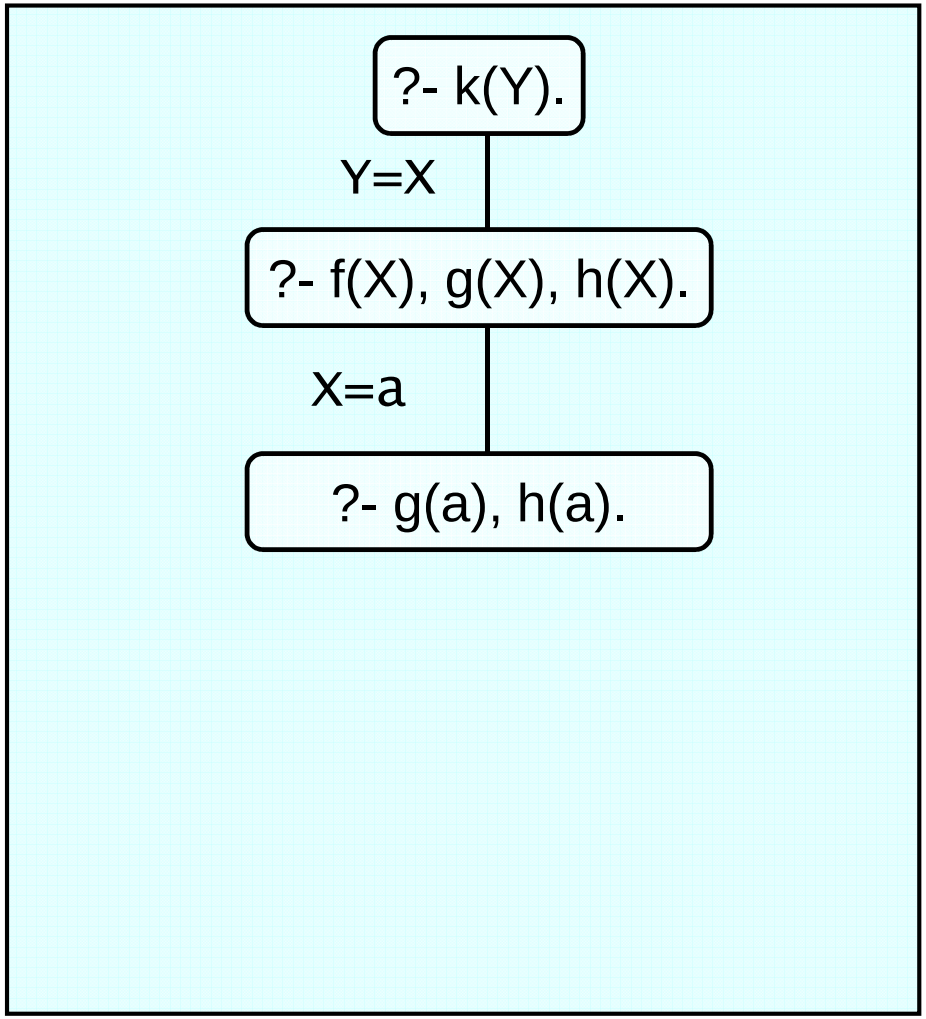
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

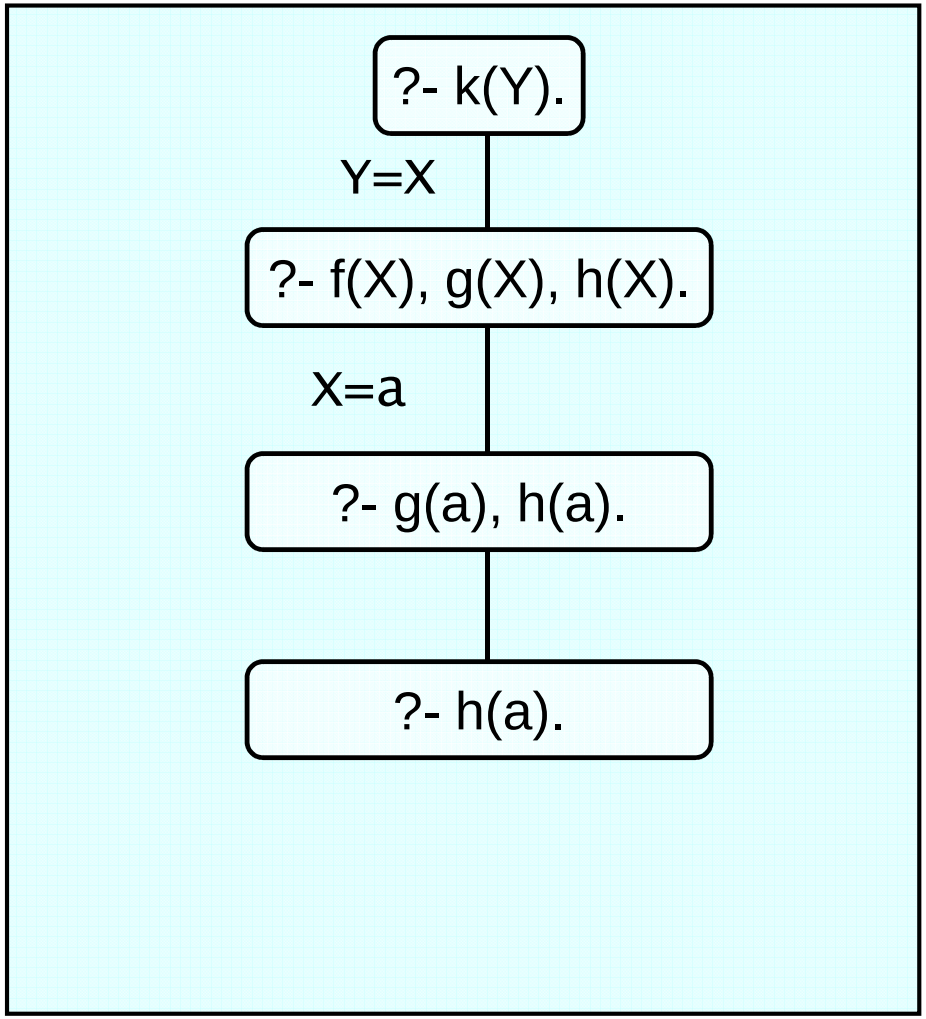
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

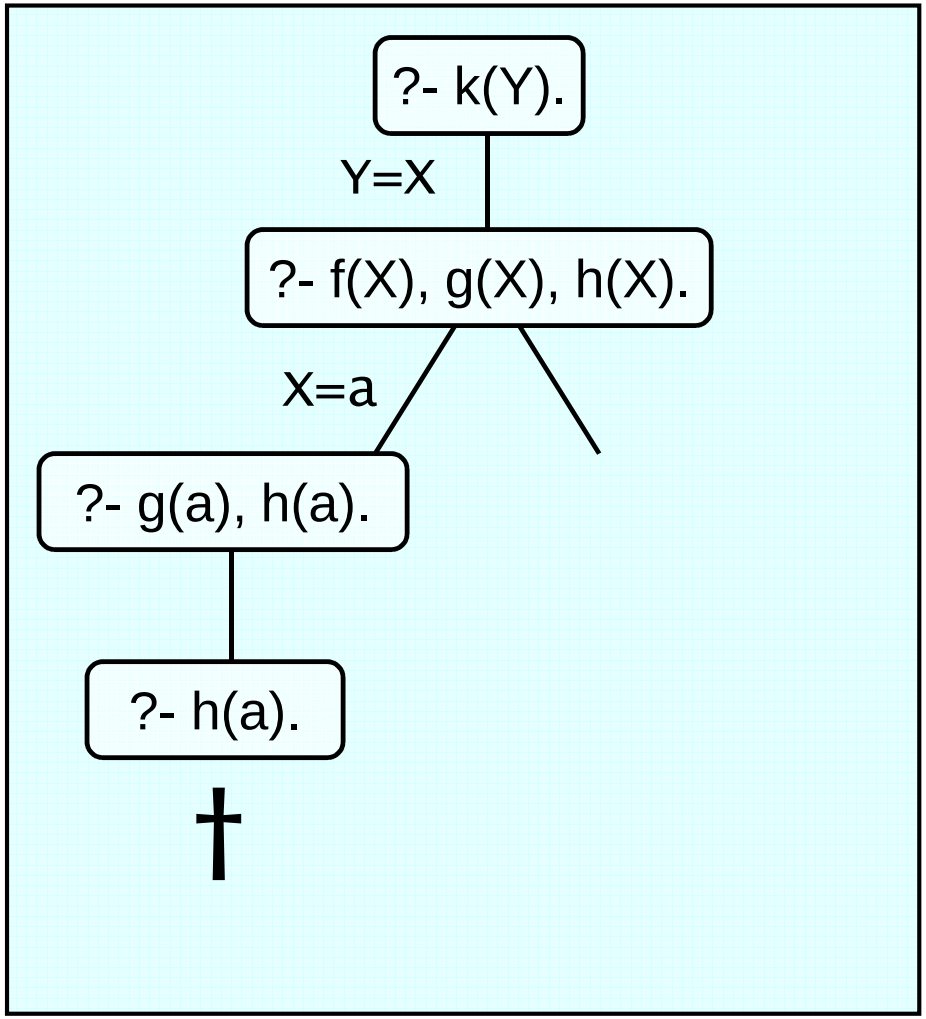
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

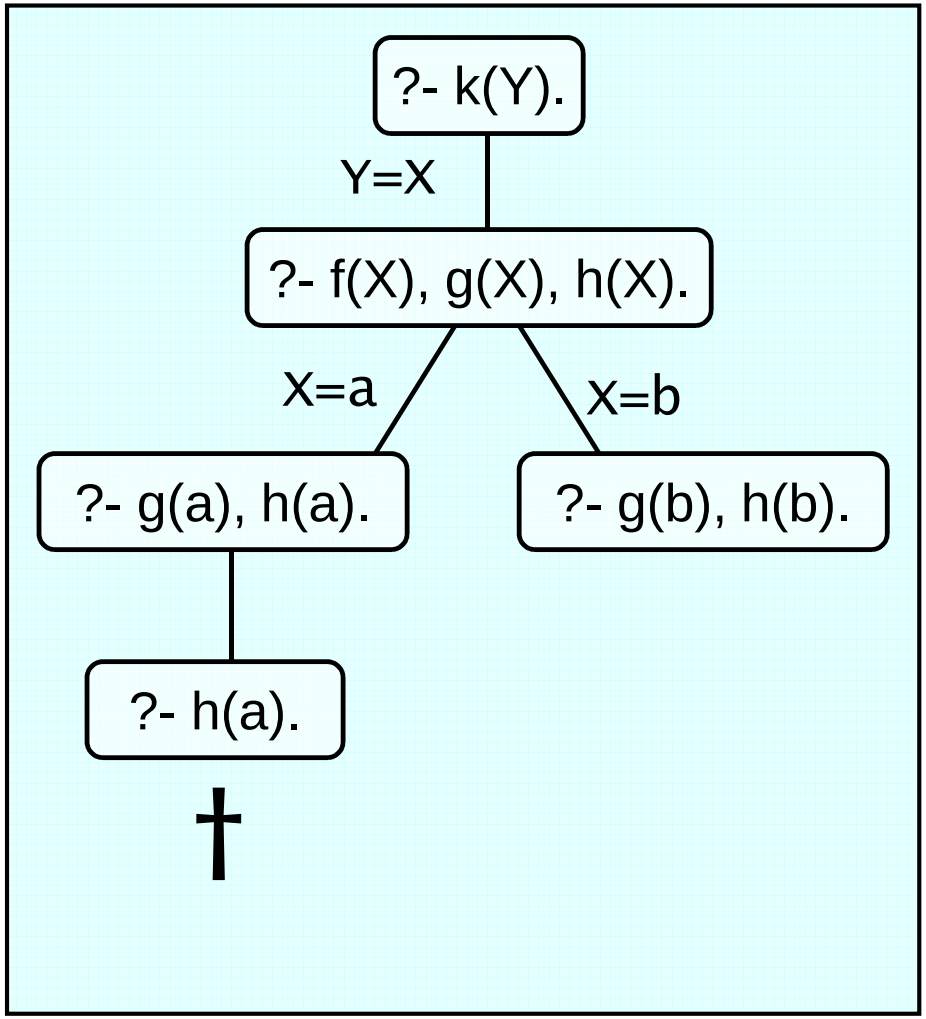
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

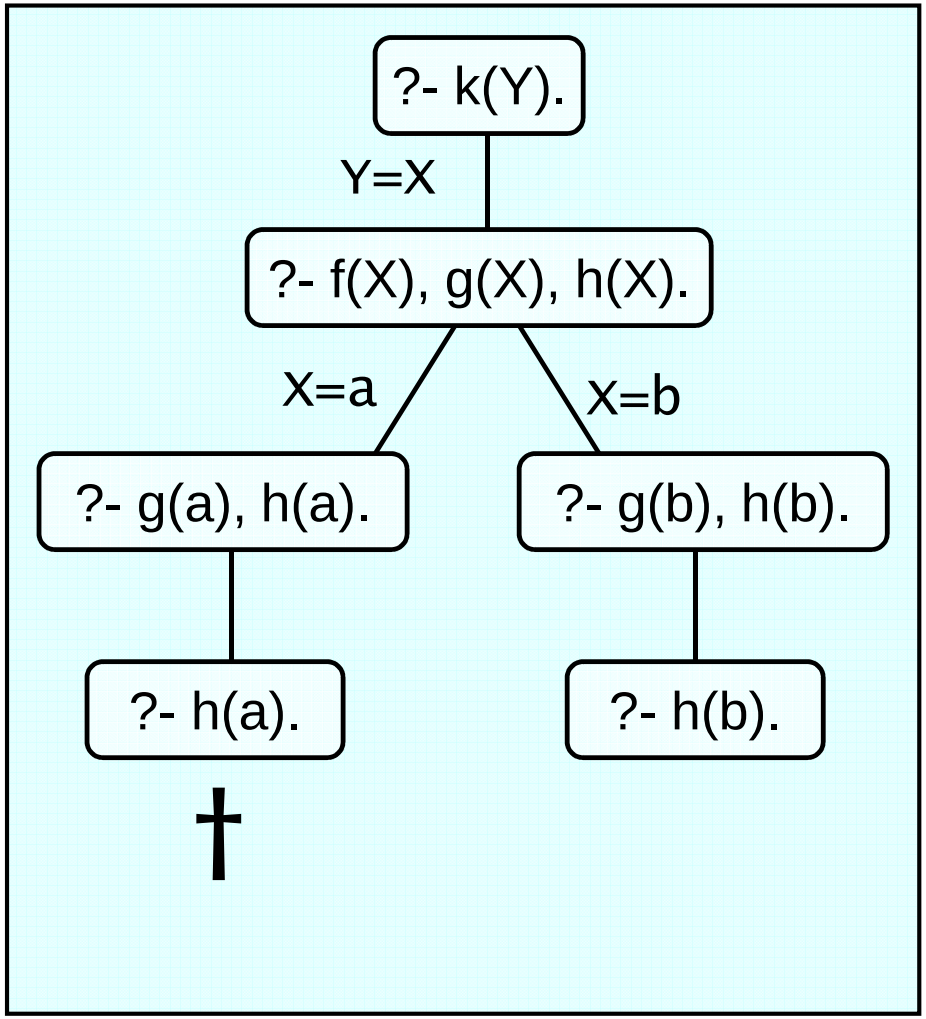
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

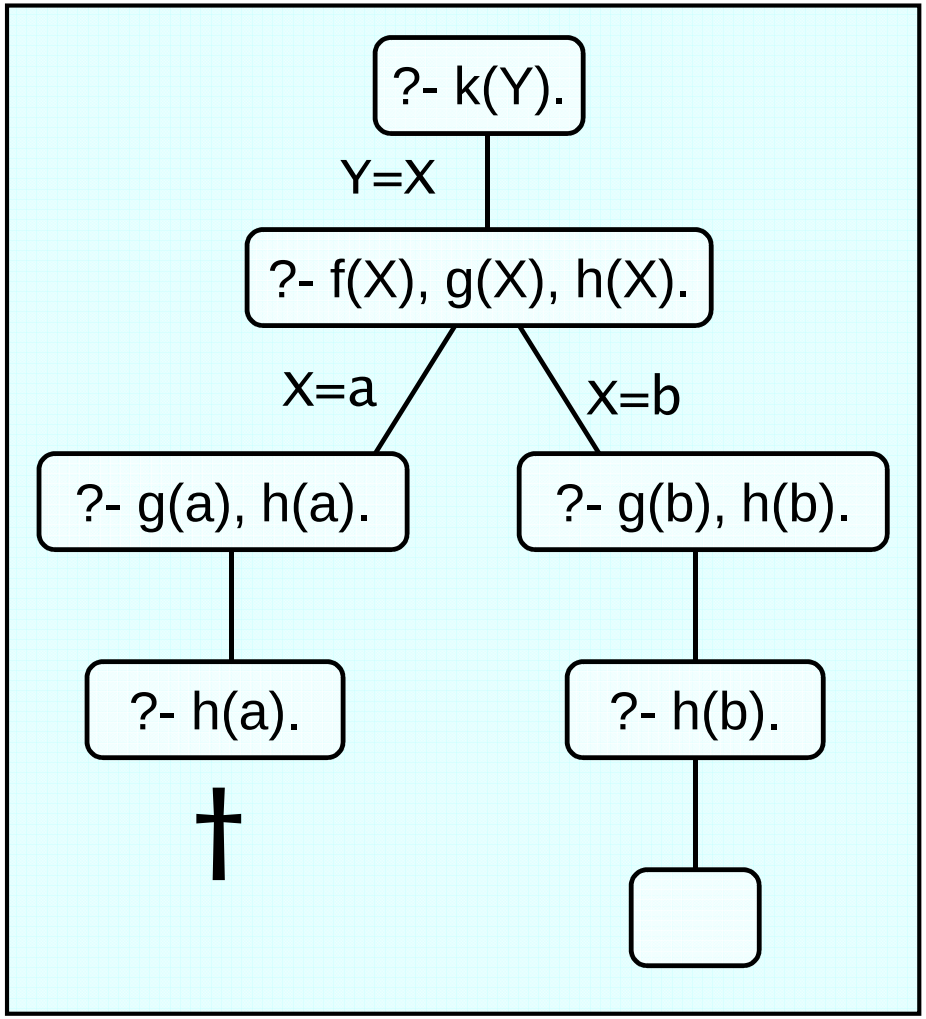
```
?- k(Y).
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

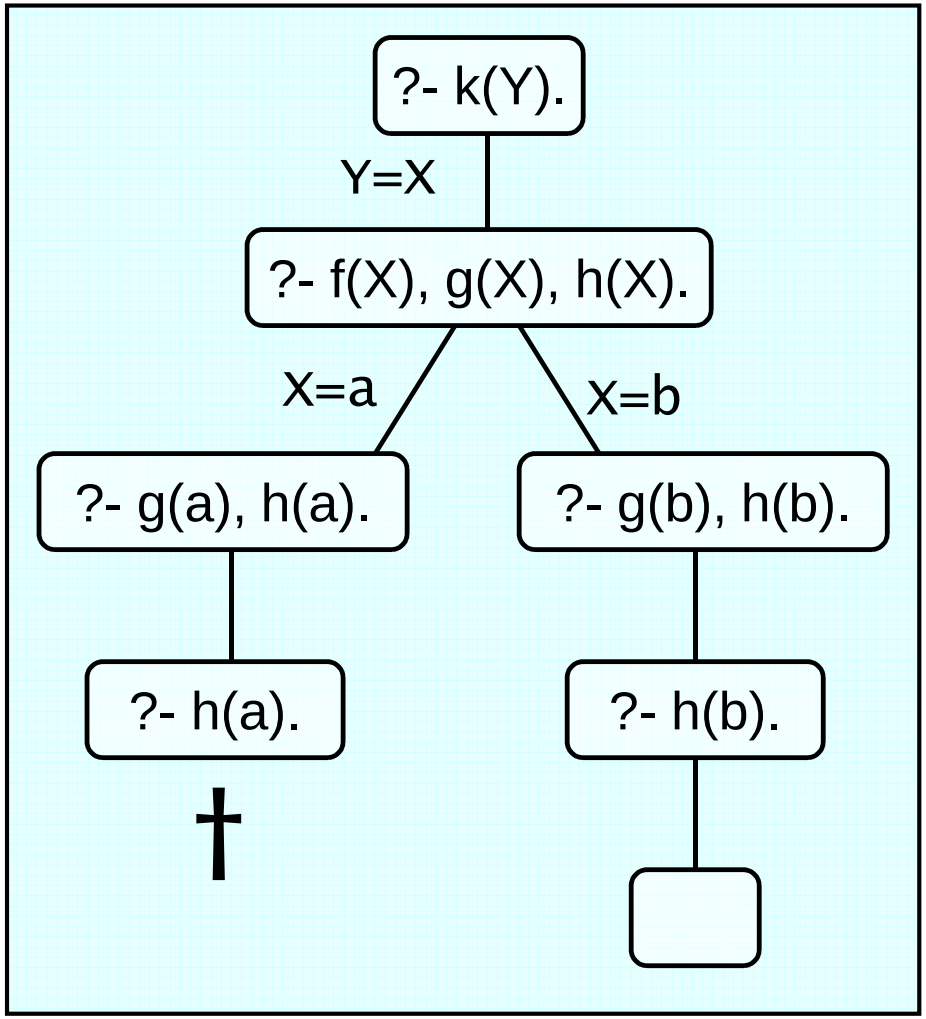
```
?- k(Y).
Y=b
```



Ejemplo: árbol de búsqueda


```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
Y=b;
no
?-
```



Resumen

- Se ha:
  - Definido la unificación
  - Visto la diferencia entre unificación standard y de Prolog
  - Introducido árboles de búsqueda

Próxima

- Recursión en Prolog
  - Definiciones recursivas en Prolog
  - Problemas con significado declarativo y procedimiento de prueba de Prolog