LENGUAJE DE PROGRAMACIÓN LÓGICA

PROLOG

UNSL

ÍNDICE

ÍNDICE

PROGRAMACIÓN LÓGICA vs PROGRAMACIÓN ALGORÍTMICA

Un algoritmo que resuelve un problema tiene dos componentes:

- Lógica o declarativa: especifica el conocimiento útil para la resolución del problema.
- Control: determina la forma en que ese conocimiento puede utilizarse para resolverlo.

EJEMPLO: El factorial de un número se define como

$$Fact(n) = \begin{cases} 1 & \text{cuando } n = 0 \\ n * Fact(n-1) & \text{cuando } n > 0 \end{cases}$$

Definición que constituye el componente lógico de cualquier algoritmo de cálculo del factorial.

PROGRAMACIÓN LÓGICA

EJEMPLO Algoritmos para realizar el cómputo del factorial

Entrada	n
$Paso_1$	$Fact = 1$
$Paso_2$	$If\ n = 0\ then\ return\ Fact$
$Paso_3$	$Fact = Fact * n,\ n = n - 1,\ go\ to\ Paso_2$

Entrada	n
$Paso_1$	$Fact = 1,\ n_1 = 0$
$Paso_2$	$If\ n_1 = n\ then\ return\ Fact$
$Paso_3$	$n_1 = n_1 + 1,\ Fact = Fact * n_1,\ go\ to\ Paso_2$

Estos algoritmos constituyen dos diferentes componentes de control para el mismo componente lógico.

PROGRAMACIÓN LÓGICA

⌘ PROGRAMACIÓN ALGORÍTMICA

El énfasis está en diseñar algoritmos que resuelvan problemas, de modo que al ejecutarlos se calcule la solución.

⌘ PROGRAMACIÓN LÓGICA

El énfasis está en el problema que se pretende resolver.

- El programador especifica el componente lógico de los algoritmos, el cual consiste del conjunto de condiciones que la solución debe satisfacer.
- El intérprete, por medio de sus mecanismos de inferencia, deduce las soluciones a partir del conjunto de condiciones de la solución.

PROGRAMACIÓN LÓGICA

$$Fact(n) = \begin{cases} 1 & \text{cuando } n = 0 \\ n * Fact(n-1) & \text{cuando } n > 0 \end{cases}$$

EJEMPLO Condiciones que satisfacen el factorial , en lenguaje de lógica de predicados

$$(Fact(0,1) \land (\forall n(\forall n1(\forall m(\forall m1(((>(n,0) \land =(n1,-(n,1)) \land (Fact(n1,m1) \land =(m,^\star(n,m1)))) \rightarrow Fact(n,m)))))))$$

Interpretación:

Fact(x,y): x es el factorial de y (predicado)

>(x,y): x es mayor que y (predicado)

=(x,y): x es igual a y (predicado)

-(x,y): resta entre x e y (función)

*(x,y): producto entre x e y (función)

ÍNDICE

PROGRAMACIÓN LÓGICA: PROLOG

⌘ PROLOG

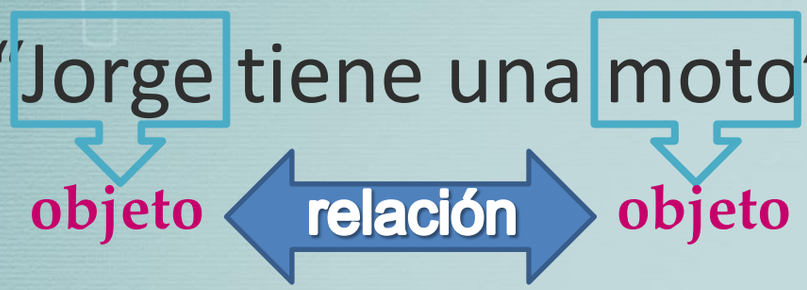
- Es un lenguaje de programación para ordenadores que se basa en el lenguaje de la **Lógica de Predicados (primer orden)**.

- Se utiliza para resolver problemas en los que entran en juego **objetos** y **relaciones** entre ellos.

PROGRAMACIÓN LÓGICA: PROLOG

- Se utiliza para resolver problemas en los que entran en juego **objetos** y **relaciones** entre ellos.

Ej: "Jorge tiene una moto"



¿Tiene Jorge una moto? ⇨ Indagamos acerca de una relación

También usamos **reglas** para describir relaciones:

Ej: "Dos personas son hermanas si ambas son mujeres y tienen los mismos padres"

PROGRAMACIÓN LÓGICA: PROLOG

- Prolog utiliza programación lógica: se especifica **qué** se tiene que hacer (programación declarativa), y no **cómo** se debe hacer (programación imperativa).

- Además incluye predicados predefinidos:

• meta-lógicos (var, nonvar, ==, …),
• extra-lógicos (write,get,…) y
• para expresar información de control de cómo realizar una tarea (el corte,…)

ÍNDICE

PROLOG Y LÓGICA DE PREDICADO

La Lógica de Predicado (lógica de primer orden) analiza:

- **fórmulas atómicas**: frases sencillas del lenguaje que consisten en Términos y Predicados.

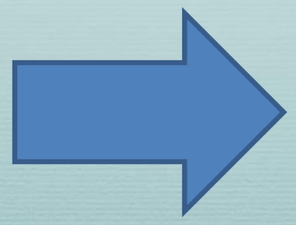
Términos ⇒ objetos que intervienen

Predicados ⇒ propiedades o relaciones entre objetos.

- **fórmulas moleculares** (formulas bien formadas): fórmulas atómicas combinadas mediante conectivas lógicas que dan lugar a fórmulas más complejas.

PREDICADOS EN PROLOG

- Se utilizan para:
  - expresar <u>propiedades</u> de los objetos (predicados monádicos).
  - Expresar <u>relaciones</u> entre los objetos (predicados poliádicos).

PREDICADOS  **HECHOS**

Predicados en Prolog: HECHOS

`simbolo_de_predicado(arg1,arg2,…,argn).`

- Los nombres de todos los objetos y relaciones deben comenzar con minúscula.
- Primeo se escribe la relación o propiedad (predicado).
- Los objetos se escriben separados por comas y encerrados entre paréntesis (argumentos).
- Al final del hecho debe ir un punto (“.”)

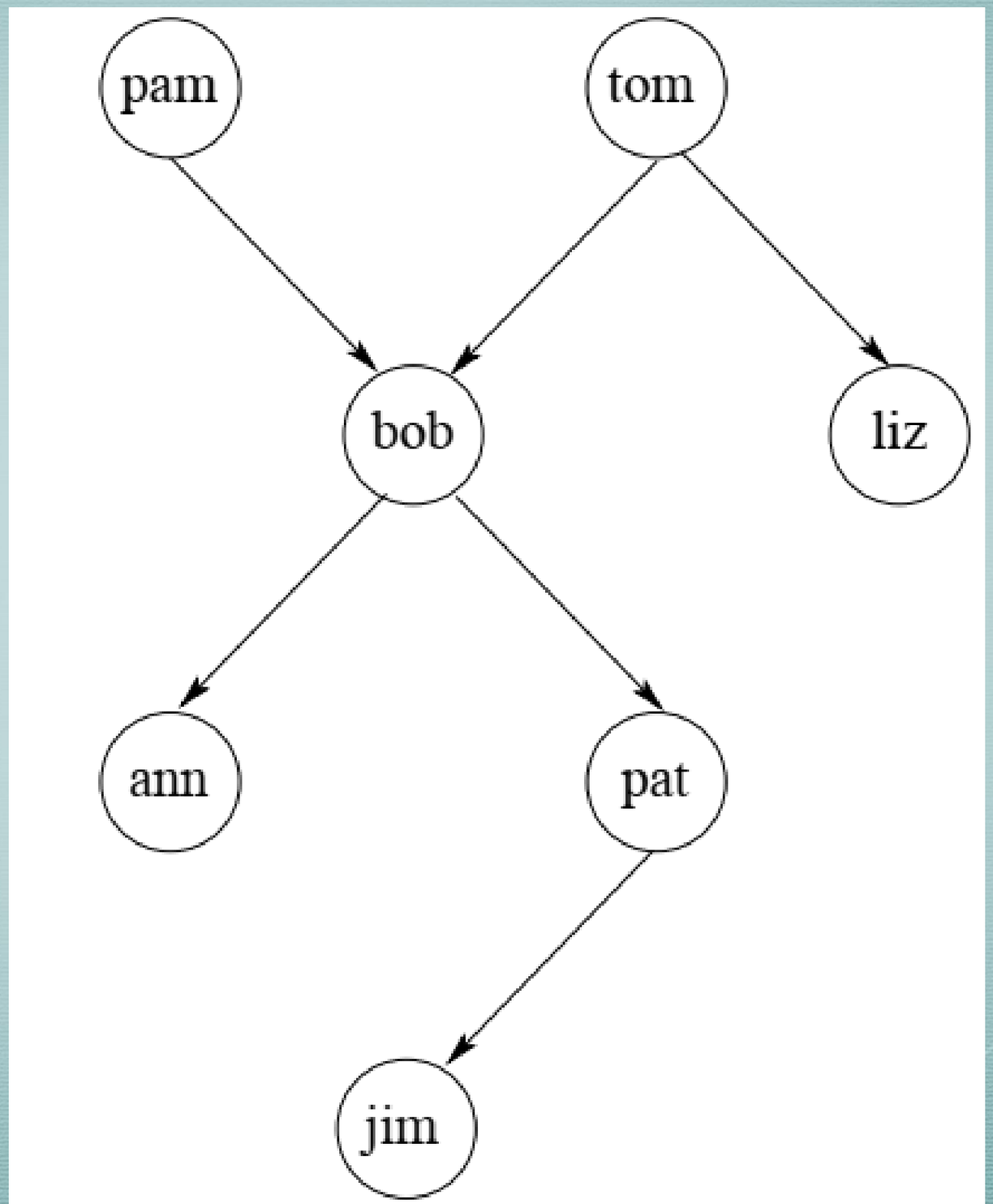
EJEMPLO

puedo expresar “Tom es progenitor de Bob” con el siguiente hecho

`progenitor(tom,bob).`

Predicados en Prolog: HECHOS

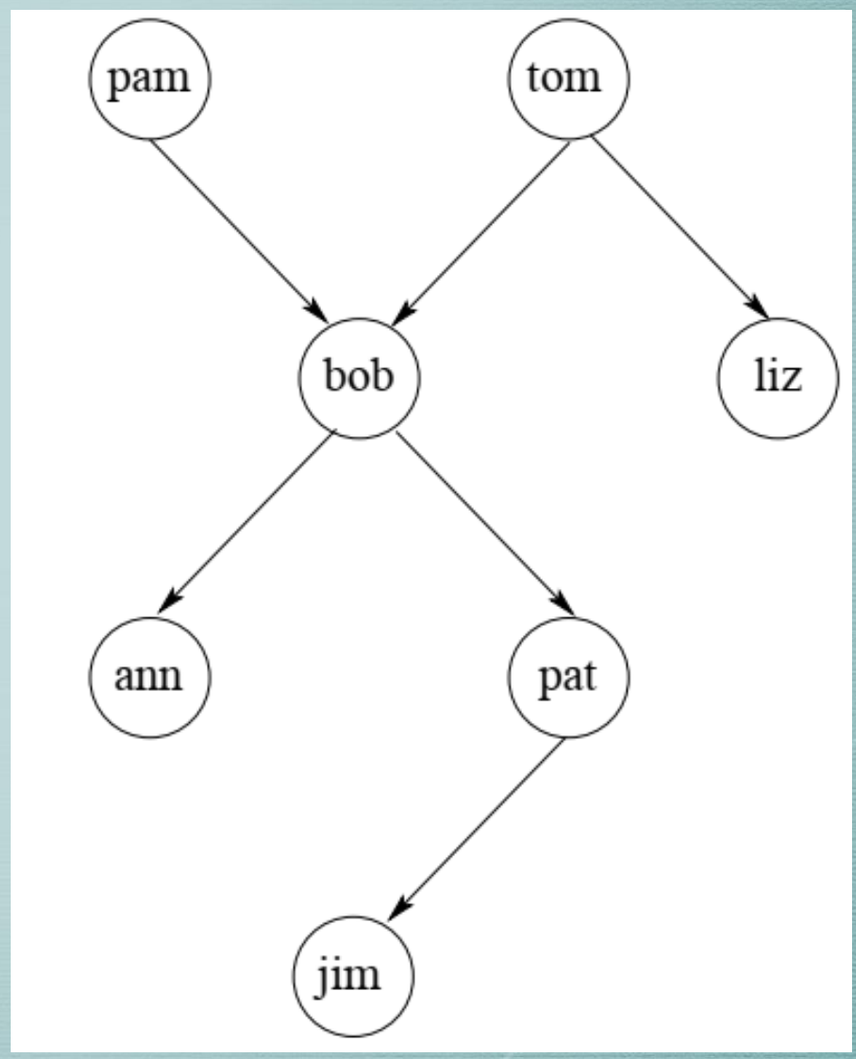
EJEMPLO:



Predicados en Prolog: HECHOS

EJEMPLO: (predicados **monádicos**-propiedades-)

```
mujer(pam).
mujer(liz).
mujer(pat).
mujer(ann).
hombre(tom).
hombre(bob).
hombre(jim).
```

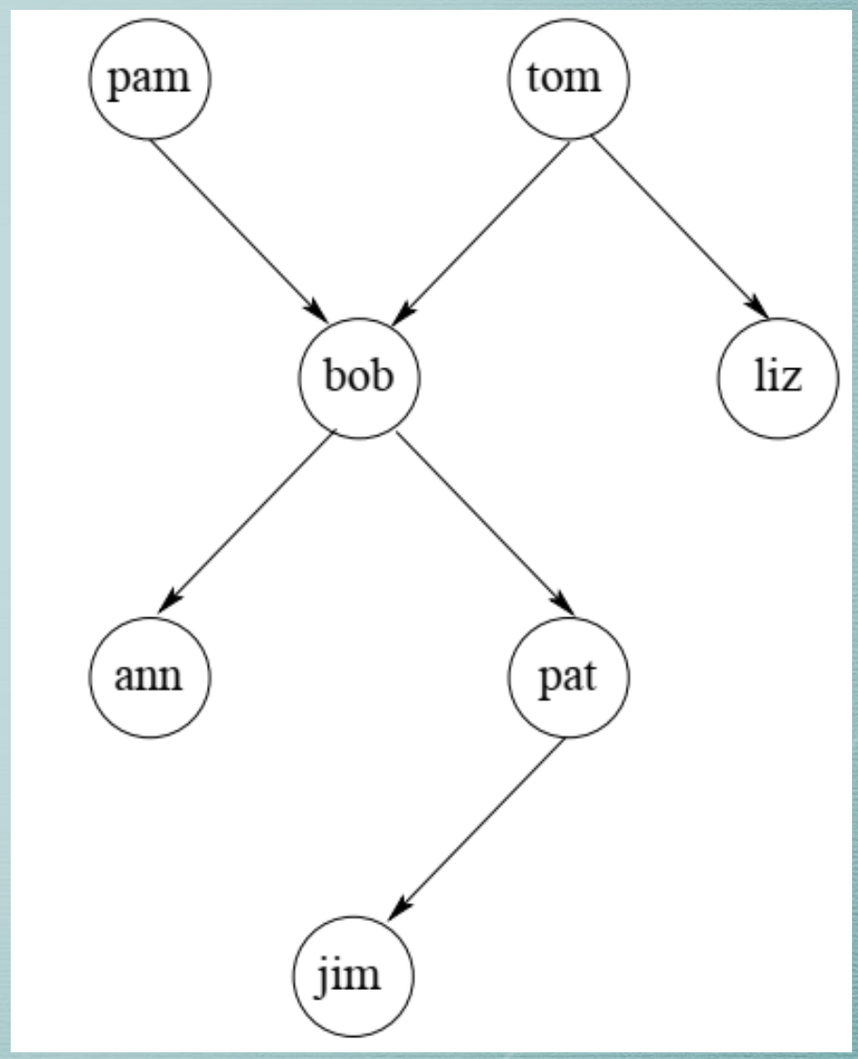


Base de conocimientos (programa) comunicado al sistema Prolog

Predicados en Prolog: HECHOS

EJEMPLO: (predicados **poliádicos**-relaciones-)

```
progenitor(pam,bob).
progenitor(tom,bob).
progenitor(tom,liz).
progenitor(bob,ann).
progenitor(bob,pat).
progenitor(pat,jim).
```



Base de conocimientos (programa) comunicado al sistema Prolog

Predicados en Prolog: HECHOS

EJEMPLO

- ¿Bob es progenitor de Pat?

```
?-progenitor(bob,pat).
yes
?-progenitor(liz,pat).
no
?-progenitor(tom,ben).
no
```

Predicados en Prolog: HECHOS
EJEMPLO

- ¿Quién es el progenitor de Liz?

```
?-progenitor(X,liz).

X=tom
```

- ¿Quiénes son los hijos o hijas de Bob?

```
?-progenitor(bob,X).

X=ann;

X=pat;

no
```

Predicados en Prolog: HECHOS

EJEMPLO

- ¿Quién es progenitor y de quienes?

```
?-progenitor(X,Y).

X=pam

Y=bob;

X=tom

Y=bob;

...
```

Predicados en Prolog: HECHOS

EJEMPLO

- ¿Quién es abuelo o abuela de Jim?

progenitor(y,jim) ∧ *progenitor(x,y)*

```
?-progenitor(Y,jim),progenitor(X,Y).
X=bob
Y=pat
```

Predicados en Prolog: HECHOS

EJERCICIO:

- Plantear las consultas que permitan saber:

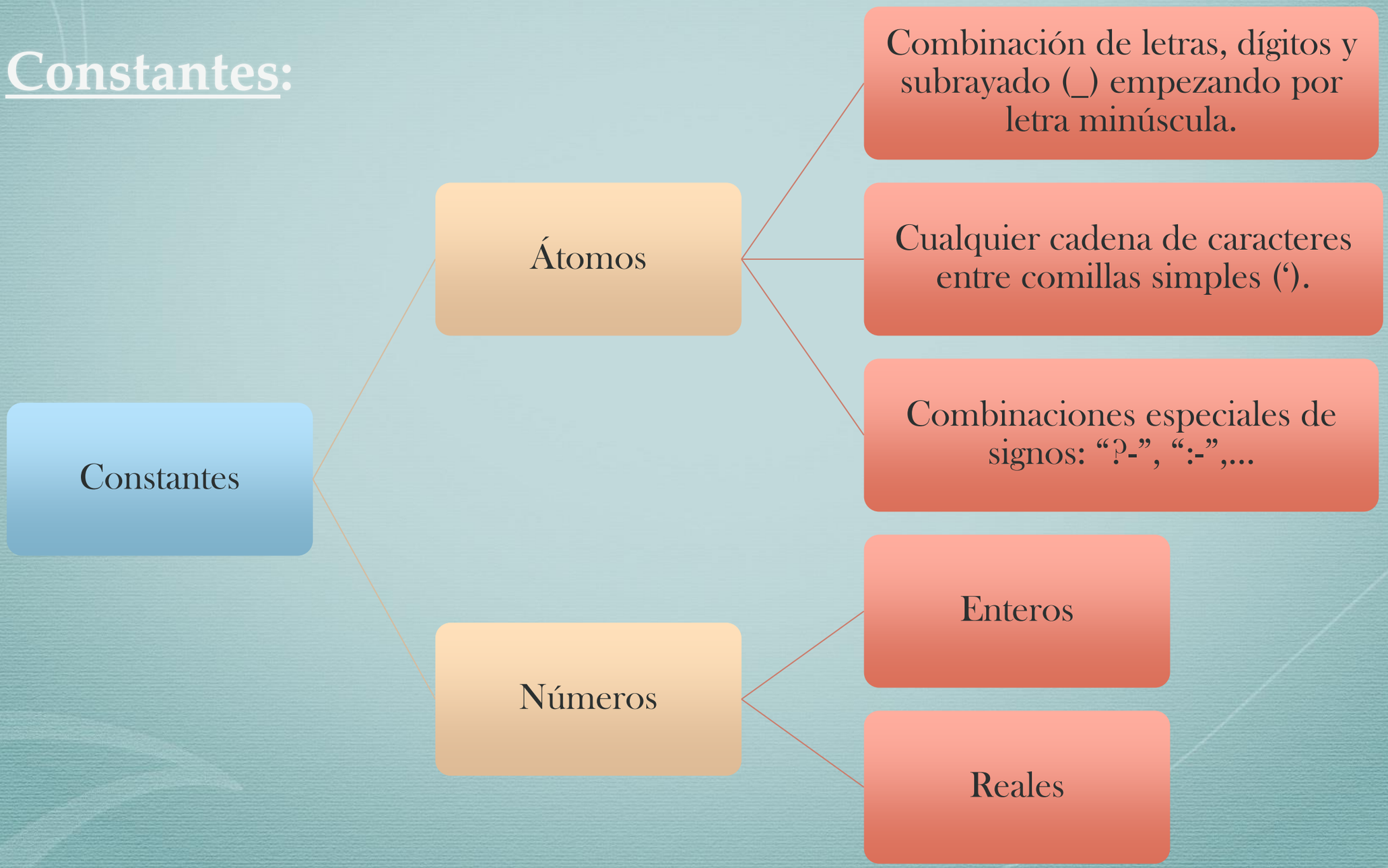
1. ¿Quiénes son los nietos de Tom?
2. ¿Ann y Pat tienen algún progenitor en común?

ÍNDICE

Términos en Prolog

- Pueden ser constantes o variables.

⌘ **Constantes:**



Términos en Prolog: Constantes

- Átomos:

1. Cadenas de letras,dígitos, y "_" comenzando con letra minúscula.
2. Cadenas de caracteres especiales ==>, ><:>, .:.>
3. Cadenas de caracteres encerrados entre comillas simples. Ej: 'Ana'

- Números:

1. Enteros: 1, 1313, -97
2. Reales: 3.14, -0.035

Constantes: Números y aritmética en Prolog

EJEMPLO:

Supongamos la siguiente consulta:

```
?- X=1+2.
```

```
X=1+2
```

El resultado no es X=3 como uno esperaría.

El operador "=" trata de igualar dos términos pero no fuerza la valuación de las operaciones.

Para lograr dicho efecto se utiliza el operador "is":

```
?- X is 1+2.
```

```
X=3
```

Términos en Prolog: Constantes

- Los operadores **no** hacen que se efectúe ningún tipo de operación aritmética.
- El predicado de evaluación es el operador infijo "**is**".

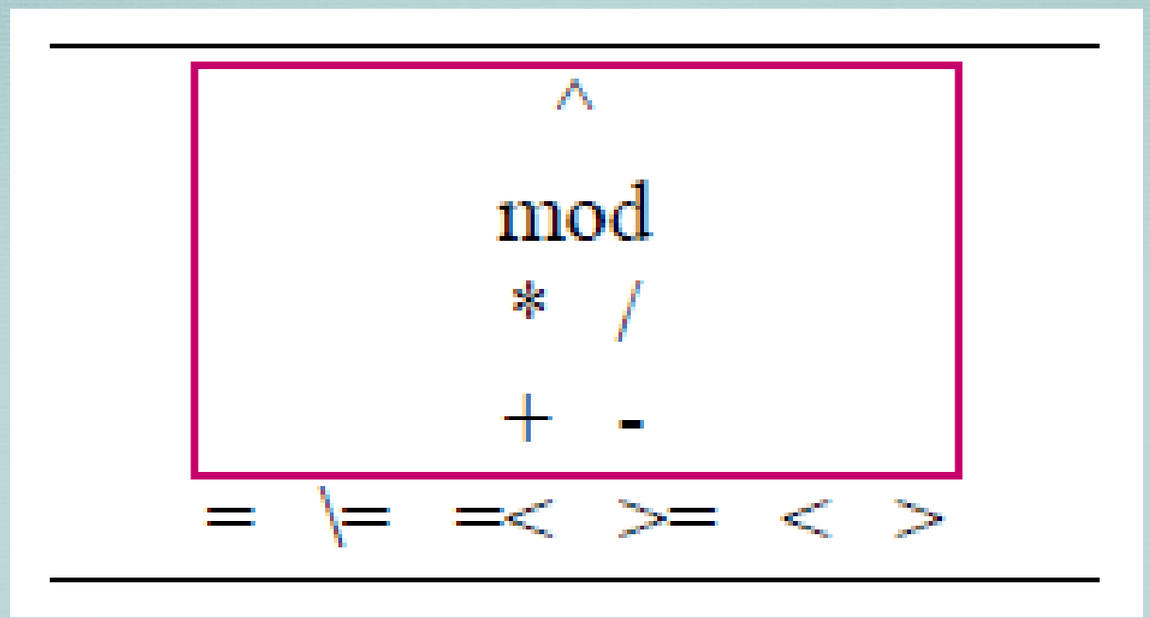
Número **is** Expresión

Puede ser una variable o una constante numérica.

Término que se interpreta como una expresión aritmética con todo sus valores instanciados.

Términos en Prolog: Constantes

"is" también permite evaluar otras operaciones :

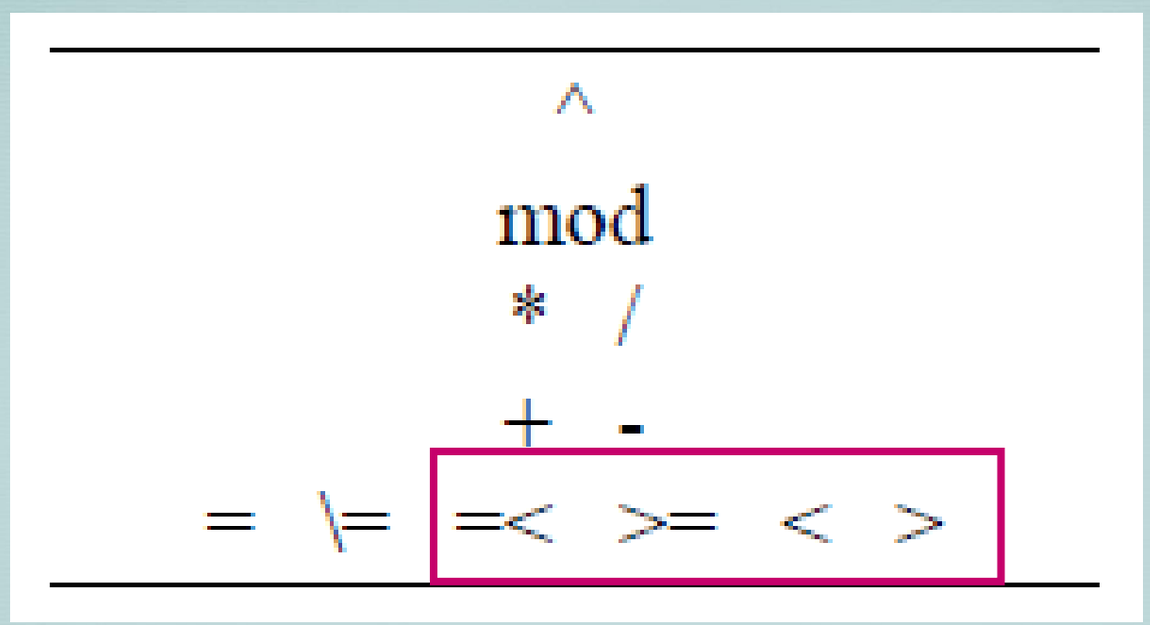


Ejemplo:

```
?- X is 5/2, Y is 2**3, Z is 5 mod 2.
X=2.5
Y=8
Z=1
```

Términos en Prolog: Constantes

Otros operadores que comparan valores numéricos suelen forzar la evaluación automática de los términos:



Ejemplo:

```
?- 3*4 > 10.
yes
```

"\=" sirve para controlar si dos términos son distintos

Términos en Prolog: Variables

- **Variables:** se utilizan para representar objetos cualquiera del Universo u objetos desconocidos en ese momento (incógnitas del problema).

  - Empiezan siempre con letra mayúscula o con el signo de subrayado ("_").

    Ej: `X` `_indice` `Sumando`

  - Se puede utilizar la variable anónima ("_") cuando trabajamos con objetos cuya identidad no nos interesa.

    Ej: `abuelo(jorge,_).`

Términos en Prolog: Variables

- Una variable está instanciada cuando existe un objeto determinado representado por ella. Y está no instanciada cuando todavía no se sabe lo que representa.
- El alcance de una variable está delimitado por la cláusula que la contiene (un hecho, una regla o una consulta).
- Prolog no utiliza símbolos de cuantificación para las variables, pero sí están implícitamente. En general, todas las variables que aparecen están cuantificadas universalmente.

Términos en Prolog: Variables

Cuantificadores

- Las variables que aparecen en los hechos están cuantificadas universalmente:

EJEMPLO:

```
gusta(jorge,X).
```

∀x gusta(jorge,x)

A Jorge le gusta cualquier cosa.

Términos en Prolog: Variables

Cuantificadores

- Las variables que aparecen en la cabeza de las reglas están cuantificadas universalmente. Las variables que aparecen en el cuerpo de la regla, pero no en la cabeza, están cuantificadas existencialmente.

EJEMPLO:

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

$(\forall x(\forall y(\exists z\ (padre(x,z) \wedge padre(z,y)) \rightarrow abuelo(x,y))))$

Para toda pareja de personas, una será el abuelo de otra si existe alguna persona del cual el primero es padre y a su vez es padre del segundo.

Términos en Prolog: Variables

Cuantificadores

- Las variables que aparecen en las preguntas están cuantificadas existencialmente.

EJEMPLO:

```
?-progenitor(tom,X).
```

$(\exists x\ progenitor(tom,x))$

Pregunta: ¿existe alguien de quien Tom es progenitor?

ÍNDICE

Conectivas lógicas

- Fórmulas moleculares: fórmulas atómicas combinadas mediante conectivos.

- **Conjunción**: se representa poniendo "," entre los objetivos. Y consiste en objetivos separados que Prolog debe satisfacer, uno después del otro.

```
X , Y
```

Conectivas lógicas

- Fórmulas moleculares: fórmulas atómicas combinadas mediante conectivos.

  - **Disyunción**: se representa poniendo ";" entre los objetivos. Tendrá éxito si se cumple alguno de los objetivos que la componen.

```
X ; Y
```

  - También se puede representar mediante un conjunto de sentencias alternativas, es decir, poniendo cada miembro de la disyunción en una cláusula aparte.

```
X.
Y.
```

Conectivas lógicas

- Fórmulas moleculares: fórmulas atómicas combinadas mediante conectivos.

  - **Negación**: no puede ser representada explícitamente. Se representa implícitamente por la falta de aserción: "no", y tendrá éxito si el objetivo X fracasa. Se lo representa con el predicado predefinido **`not`** o con "`\+`".

```
not(X)
```

```
\+ X
```

Conectivas lógicas

- Fórmulas moleculares: fórmulas atómicas combinadas mediante conectivos.

  - **Implicación o condicional**: Sirve para significar que un hecho depende de un grupo de otros hechos ("si… entonces…"). Se utiliza el símbolo ":-" para representar lo que se llama una REGLA.

```
cabeza_de_la_regla:- cuerpo_de_la_regla
```

  - La cabeza describe el hecho que se intenta definir y el cuerpo describe los objetivos que deben satisfacer para que la cabeza sea cierta.

Conectivas lógicas

- Fórmulas moleculares: fórmulas atómicas combinadas mediante conectivos.

- **Implicación o condicional**:

C:- O1, O2, …, On

Puede ser leída declarativamente como:

"La demostración de la cláusula C se sigue de la demostración de los objetivos O1,O2,…On".

O procedimentalmente como:

"Para ejecutar el procedimiento C, se deben llamar para su ejecución los objetivos O1,O2,…On".

Conectivas lógicas

- Fórmulas moleculares: fórmulas atómicas combinadas mediante conectivos.

  - **Implicación o condicional**:

```
cabeza_de_la_regla:- cuerpo_de_la_regla
```

También puede verse como una implicación lógica "al revés":

```
cuerpo_de_la_regla -> cabeza_de_la_regla
```

Nota: un mismo nombre de variable representa el mismo objeto siempre que aparece en la regla.

Relaciones en Prolog: REGLAS

EJEMPLO:

```
%programa comunicado a Prolog
   mujer(pam).
   mujer(liz).
   mujer(pat).
   mujer(ann).
   hombre(tom).
   hombre(bob).
   hombre(jim).
```

```
progenitor(pam,bob).
progenitor(tom,bob).
progenitor(tom,liz).
progenitor(bob,ann).
progenitor(bob,pat).
progenitor(pat,jim).
```

Relaciones en Prolog: REGLAS

EJEMPLO

¿Cómo definiríamos la relación descendiente?

- **Alternativa 1**: igual que progenitor pero con sus argumentos invertidos.

```
Descendiente(bob,pam).

Descendiente(bob,tom).

...
```

Relaciones en Prolog: REGLAS

EJEMPLO

¿Cómo definiríamos la relación descendiente?

- **Alternativa 2**: aprovechar que se cumple la siguiente relación.

Para todo x e y, si x es progenitor de y entonces y es descendiente de x

$$(\forall x (\forall y (progenitor(x,y) \rightarrow descendiente(y,x))))$$

```
descendiente(Y,X):-progenitor(X,Y).
```

conclusión/cabeza **si** condición/cuerpo

Relaciones en Prolog: REGLAS

EJEMPLO

⌘ ¿Qué sucede si realizamos la siguiente consulta?

```
?-descendiente(liz,tom).
```

La variables X e Y se instancian a X=tom, Y=liz

Luego de la instanciación hemos obtenido un caso especial de la regla general:

```
descendiente(liz,tom):-progenitor(tom,liz).

?-descendiente(liz,tom).

yes
```

Relaciones en Prolog: REGLAS

EJERCICIO:

- Determinar cuáles son las relaciones familiares que determinan las siguientes reglas:

1. `ma(X,Y):-progenitor(X,Y),mujer(X).`
2. `ab(X,Z):-progenitor(X,Y),progenitor(Y,Z).`

Relaciones en Prolog: REGLAS

EJERCICIO:

⌘ Definir las siguientes relaciones en el programa Prolog:

```
padre(X,Y).          X es padre de Y

madre(X,Y).          X es madre de Y

esmadre(X).          X es madre

espadre(X).          X es padre

eshijo(X).           X es hijo
```

Relaciones en Prolog: REGLAS

EJERCICIO:

⌘ Definir las siguientes relaciones al programa Prolog:

```
hija(X,Y).          X es hija de Y

hijo(X,Y).          X es tío de Y

sobrino(X,Y).       X es sobrino de Y

prima(X,Y).         X es prima de Y

abuel(X,Y).         X es abuelo o abuela de Y
```

Relaciones en Prolog: REGLAS

EJEMPLO:

⌘ Representar la siguiente inferencia:

Todos los hombres son mortales,

Dado que Sócrates es un hombre,

Podemos concluir que Sócrates es mortal.

Se puede modelar como:

$$((\forall x\,(hombre(x) \rightarrow mortal(x)))\wedge hombre(socrates)) \rightarrow mortal(socrates)$$

Relaciones en Prolog: REGLAS

EJEMPLO:

⌘ Representar la siguiente inferencia:

Todos los hombres son mortales.

Dado que Sócrates es un hombre,

podemos concluir que Sócrates es mortal.

Esquema de consulta:

$(\forall\, x\,(hombre(x) \rightarrow mortal(x)))$ (1)

$hombre(socrates)$ (2)

} oraciones

$mortal(socrates)$?

} consulta

Relaciones en Prolog: REGLAS

EJEMPLO:

- En Prolog:

```
hombre(socrates).

mortal(X):-hombre(X).

?-mortal(socrates).

yes
```

Relaciones en Prolog: REGLAS RECURSIVAS

EJEMPLO

¿Cómo definiríamos la relación predecesor?

- **Alternativa 1**: definirla en función de la relación progenitor.

```
predecesor(X,Z):-       progenitor(X,Z).

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

predecesor(X,Z):-       progenitor(X,Y1),
                        progenitor(Y1,Y2),
                        progenitor(Y2,Z).

...
```

Relaciones en Prolog: REGLAS RECURSIVAS

EJEMPLO

¿Cómo definiríamos la relación predecesor?

⌘ **Alternativa 2**: definirla en forma recursiva.

```
predecesor(X,Z):- progenitor(X,Z).

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

Relaciones en Prolog: REGLAS RECURSIVAS

EJERCICIO

- Especificar una consulta que determine las personas de las cuales Pam es predecesor (o sea los sucesores de Pam), y descubrir cuáles serán las respuestas de Prolog.

ÍNDICE

Estructura de un programa

⌘ Llamaremos cláusulas de un predicado tanto a los hechos como a las reglas.

⌘ Una colección de cláusulas forma una Base de Conocimientos (BC).

⌘ La programación en Prolog consiste en :

- declarar algunos **HECHOS** sobre los objetos y sus relaciones.
- definir algunas **REGLAS** sobre los objetos y sus relaciones.
- hacer **PREGUNTAS** sobre los objetos y relaciones.

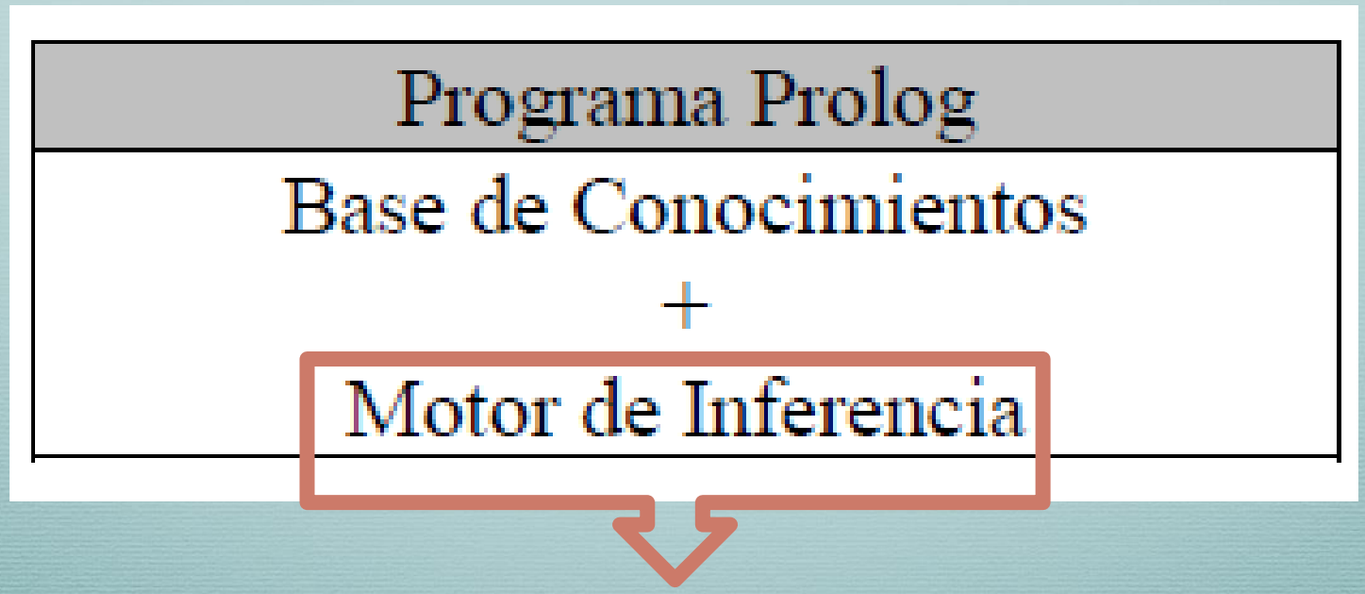
Estructura de un programa

⌘ **Programa Prolog**: conjunto de afirmaciones (hechos y reglas) representando los conocimientos que poseemos de un determinado dominio.

PREMISAS

⌘ **Ejecución del programa**: demostración de un Teorema en este Universo, es decir, demostración de que una conclusión se deduce de las premisas (afirmaciones previas).

CONSECUENCIA



Estructura de un programa: Preguntas

Prolog responde:

yes → Ha podido demostrarlo — true

no → No lo ha podido demostrar — false

No implica que sea "falso"

Estructura de un programa: Preguntas

EJEMPLO:

⌘ Dada la siguiente definición de la relación **hermana**:

```
hermana(X,Y):- mujer(X),
               progenitor(Z,X),
               progenitor(Z,Y).
```

Estructura de un programa: Preguntas

Ejemplo:

```
?-hermana(liz,bob).
yes
?-hermana(bob,X).
no
?-hermana(ann,X).
X=pat
More (y/n)? n
yes
```

Estructura de un programa: Preguntas

Ejercicio:

1. ¿Qué sucedería si a la pregunta de si queremos más soluciones le contestamos que sí?

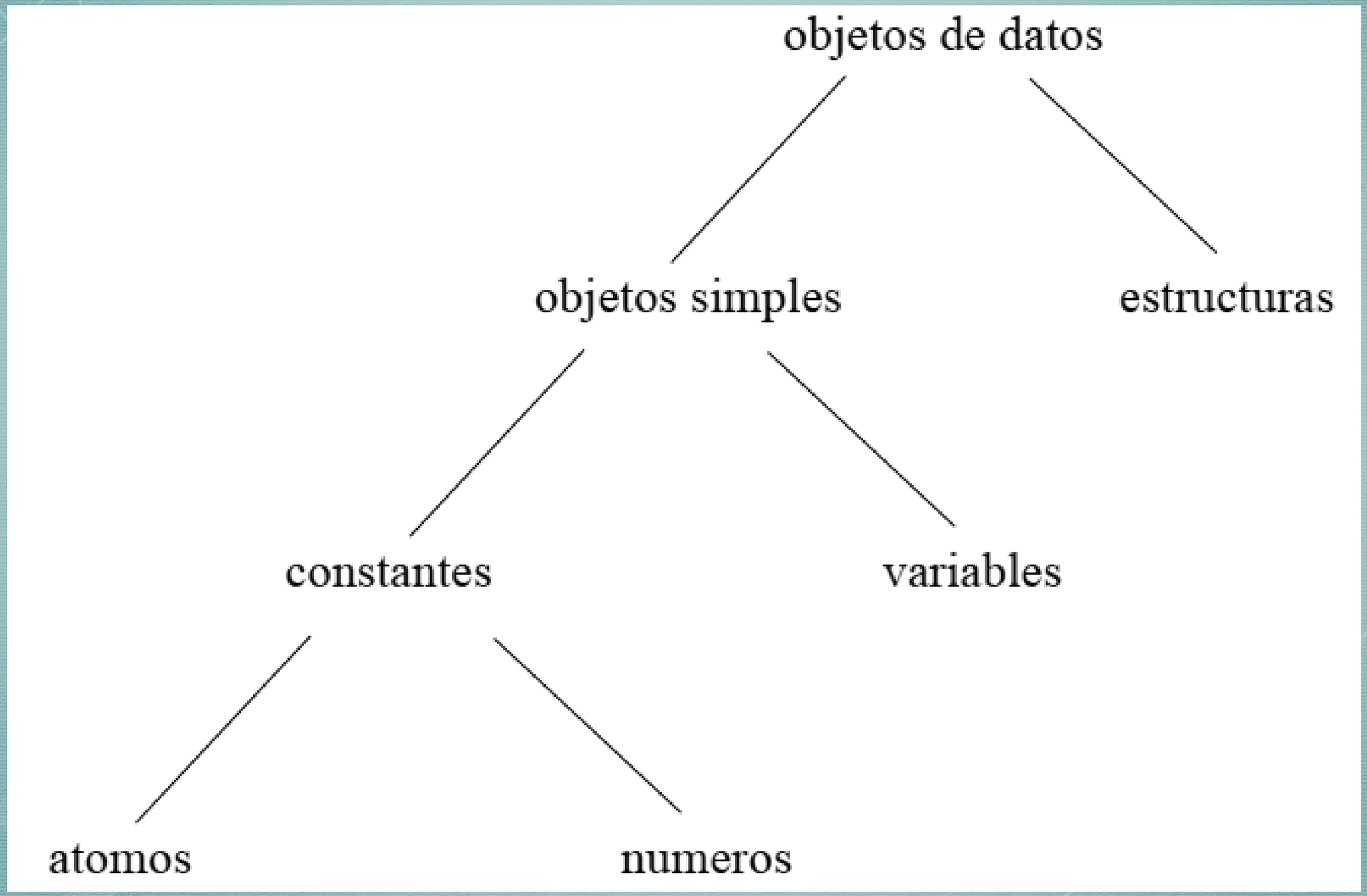
```
?-hermana(ann,X).

X=pat
More (y/n)? y
X=ann
More (y/n)? y
No
```

2. ¿Cómo lo solucionarías?

ÍNDICE

Objetos de Datos



Objetos de Datos: Variables anónimas

EJEMPLO:

```
tiene_un_hijo(X):-progenitor(X,Y).
```

innecesaria

```
tiene_un_hijo(X):-progenitor(X,_).
```

En las consultas, los valores que toman **no** son mostrados.

```
?-progenitor(X,_).

X=pam;

X=tom;

X=tom;

X=bob;

...
```

Objetos de datos: Estructuras

Una estructura es un único objeto que se compone de una colección de otros objetos, llamados componentes.

nombre(comp_1,comp_2,…,comp_n)

Están formadas por:

1. Un functor.

2. Las componentes (o argumentos) separados por comas y encerradas entre paréntesis.

Las componentes de una estructura pueden ser objetos simples (átomos, números o variables) o bien otras estructuras).

Objetos de datos: Estructuras

Ejemplo:

```
fecha(9,mayo,2024)

aula(27,capacidad(100),horario(15,18))

materia(logica_para_computacion,docente(veronica,ludueña),2).
```

Sintaxis parecida a la sintaxis de relaciones (predicados).

ATENCIÓN: Las estructuras (al igual que otros tipos de objetos) se utilizan en los argumentos de las relaciones.

```
correlativa(materia(programacion2),materia(logica)).
```

Hecho/relación

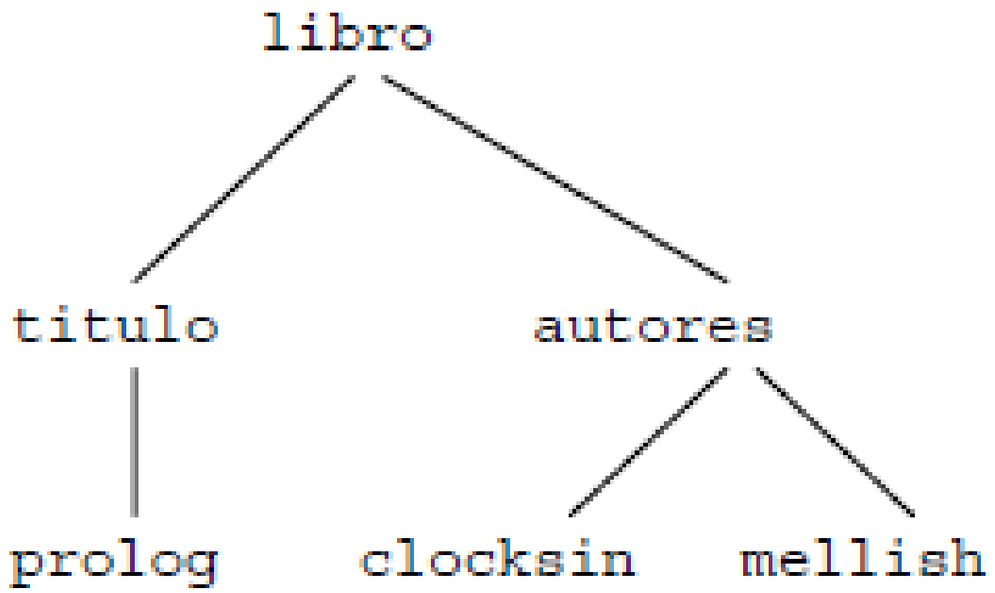
Estructuras de datos

Objetos de Datos: Árboles

La estructuras se pueden representar como árboles:

1. Raíz del árbol ➔functor
2. Descendientes de la raíz ➔ componentes

Ejemplo: libro(titulo(prolog), autores(clocksin, mellish))



Objetos de Datos: Árboles

EJERCICIO:

Dibuje los objetos representados por las siguientes estructuras y dé las representaciones de árbol correspondiente:

```
punto(1,1)

punto(2,3)

linea(punto(1,1),punto(2,3))

triangulo(punto(4,2),punto(6,4),punto(7,1))
```

Objetos de Datos: Listas

- Son secuencias ordenada de elementos de cualquier longitud.
- Los elementos de una lista pueden ser cualquier término (constantes, variables, estructuras) u otras listas.

EJEMPLO:

```
[ana,tenis,tom,sky]
```

Objetos de Datos: Listas

⌘ Una lista puede definirse recursivamente como:

- Una lista **vacia []**,
- Una estructura con dos componentes:

1. El primer ítem, llamado **cabeza** de la lista;
2. El resto de la lista, llamado la **cola.**

EJEMPLO:

`[ana,tenis,tom,sky]`

`ana` es la cabeza

`[tenis,tom,sky]` es la cola

Objetos de Datos: Listas

- Se puede diferenciar la cabeza de la cola de una lista, con la barra vertical (“|”):

```
[ Cabeza | Cola ]
```

- Esto permite referenciar toda la cola de una lista como un objeto simple.

EJEMPLO:

```
-?[1,2,3]=[C|T].

C=1

T=[2,3]
```

Objetos de Datos: Listas

EJEMPLO:

```
-?[a,b,c]=[a|[b|[c]]].

yes
```

```
-?[ana,pepe,tom,lucas]=[A|[pepe|B]].

A=ana

B=[tom,lucas]
```

Objetos de Datos: Listas

⌘ Con la barra vertical también se pueden listar cualquier número de elementos seguidos por una "|" y la lista de elementos restantes.

EJEMPLO:

```
-?[a,b,c,d]=[a,b|[c,d]].
yes
```

Objetos de Datos: Listas

EJEMPLO:

```
-? [1,2,3,4,5]=[A,B,C|T].

A=1

B=2

C=3

T=[4,5]


-?[a|[b,c]]=[a,b|[c]].

yes
```

ÍNDICE

Estructuras de control: Recursión

⌘ Vimos como podíamos definir reglas recursivas como en el caso de `predecesor(X,Z)`.

EJEMPLO:

```
predecesor(X,Z):- progenitor(X,Z).

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

Punto de parada

⌘ El cuerpo de la cláusula se llama a sí mismo.

Estructuras de control: Recursión

⌘ En una llamada recursiva al menos uno de los argumentos crece o decrece, para así poder unificar en un momento dado con la cláusula de la condición de parada.

EJEMPLO:

Procedimiento que calcula la cantidad de elementos de una lista:

- Condición de parada: lista vacía.

```
longitud([],0).
```

- Evolución de los parámetros: lista con un elemento menos.

```
longitud([C|Y],N):-longitud(Y,M),N is M+1.
```

Estructuras de control: Recursión

- En la recursión encontramos dos partes:

  - La primera parte en la que descendemos y construimos el árbol hasta encontrar el valor que unifica con la condición de parada.

  - Una segunda parte en la que ascendemos por el árbol asignando valores a las variables que teníamos pendientes en las sucesivas llamadas.

Estructuras de control: Recursión

EJEMPLO:

```
?-longitud([a,[b,c],d],N).
```

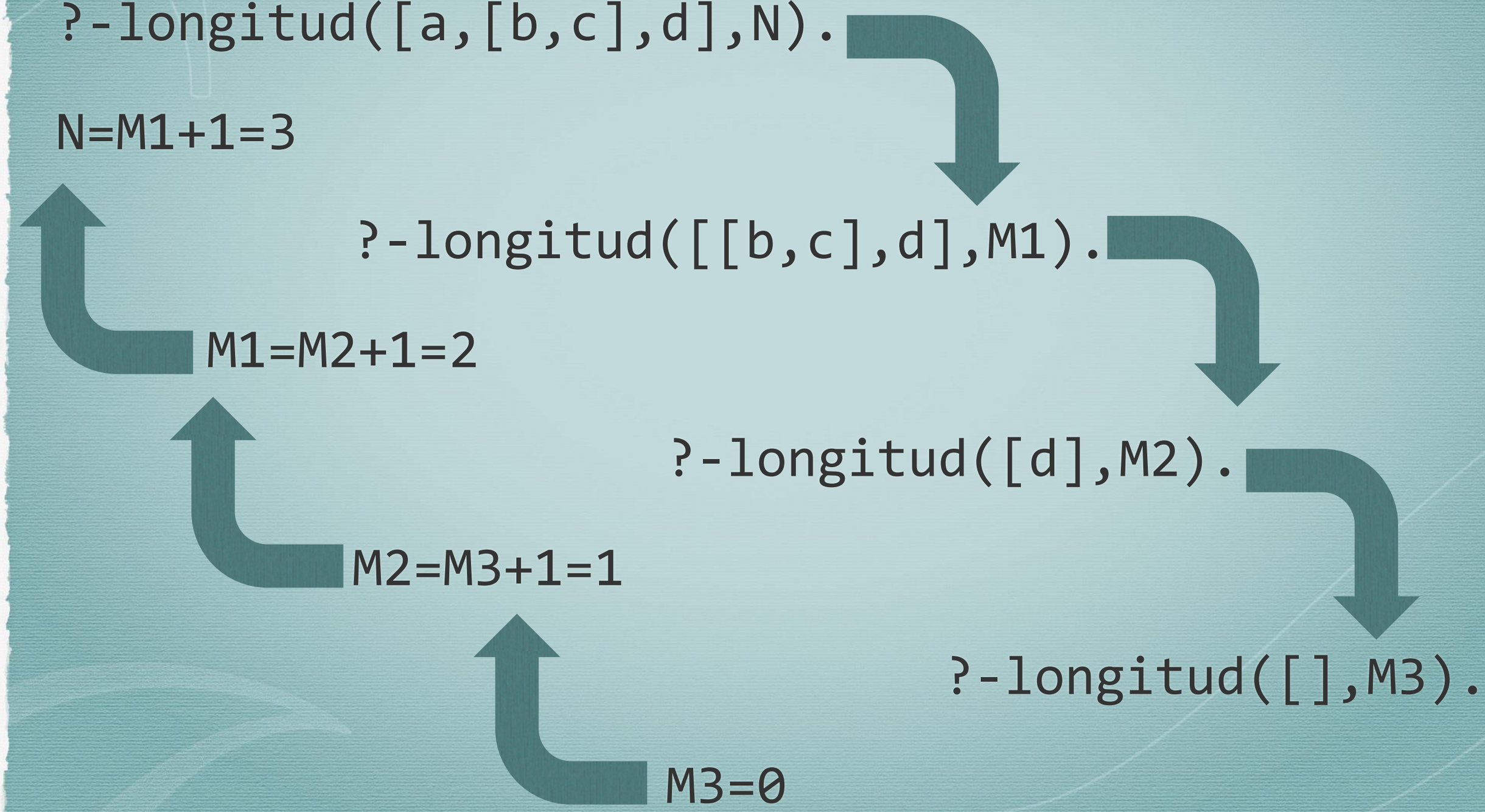
```
?-longitud([[b,c],d],M1).
```

```
?-longitud([d],M2).
```

```
?-longitud([],M3).
```

Estructuras de control: Recursión

EJEMPLO:



Estructuras de control: Recursión

⌘ Debemos tener cuidado para no escribir **recursiones circulares** y evitar **recursiones a izquierda**, que causarían que Prolog no terminara nunca.

EJEMPLO:

- Recursión circular: entrada en un bucle que no terminaría nunca.

```
padre(X,Y):-hijo(Y,X).

hijo(A,B):-padre(B,A).
```

- Recursión a izquierda: cuando una regla llama a un objetivo que es esencialmente equivalente al objetivo original.

```
persona(X):-persona(Y), madre(X,Y).

persona(adan).
```

Estructuras de control: Recursión

EJERCICIO:

Defina en forma recursiva los predicados que permitan:

1. Dado un número N, determinar el factorial de N.
2. Dado un número N determinar si N es par.

Estructuras de Control: Unificación

- Los distintos tipos de objetos➔términos.
- Informalmente, la operación de unificación (**matching**) determina si dos términos coinciden o "hacen juego".
- Dos términos hacen matching si:

1. Los términos son idénticos, o
2. Las variables en ambos términos pueden ser instanciadas a objetos de tal manera que después de la sustitución de las variables por estos objetos, los términos se tornan idénticos.

Estructuras de Control: Unificación

- Los resultados de la operación de matching son:

1. Fracaso: los términos no coinciden.
2. Éxito: los términos coinciden y se instancian las variables en ambos términos o valores que los hacen idénticos.

EJEMPLO:

`fecha(D,M,2024)` y `fecha(D1,mayo,Y1)`

Hacen matching con la siguiente instanciación:

1. **D** es instanciada con **D1**,
2. **M** es instanciada con **mayo**,
3. **Y1** es instanciada con **2024**.

Estructuras de Control: Unificación

- Los resultados de la operación de matching son:

1. Fracaso: los términos no coinciden.
2. Éxito: los términos coinciden y se instancian las variables en ambos términos o valores que los hacen idénticos.

EJEMPLO:

`fecha(D,M,2024)` y `fecha(D1,mayo,Y1)`

En formato Prolog:

```
D=D1
M=mayo
Y1=2024
```

Estructuras de Control: Unificación

- ¿Cuándo se realiza el proceso de matching?

1. Al utilizar el operador "=" (o su inverso "\=").
2. Cuando se intenta hacer coincidir los argumentos de un objetivo a cumplir con los de un hecho del programa o la cabeza de una regla.

EJEMPLO:

```
?-fecha(D,M,2024)=fecha(D1,mayo,Y1)
```

En formato Prolog:

```
D=D1
M=mayo
Y1=2024
```

Esta instanciación no es única

Estructuras de Control: Unificación

- Otras instanciaciones válidas:

EJEMPLO:

```
?-fecha(D,M,2024)=fecha(D1,mayo,Y1)

D=1

D1=1

M=mayo

Y1=2024
```

Estructuras de Control: Unificación

- Otras instanciaciones válidas:

EJEMPLO:

```
?-fecha(D,M,2024)=fecha(D1,mayo,Y1)

D=pirulo

D1=pirulo

M=mayo

Y1=2024
```

Prolog siempre muestra la instanciación más general. Deja el mayor grado de libertad posible para realizar otras operaciones de matching.

Estructuras de Control: Unificación

EJEMPLO:

```
?-fecha(D,M,2024)=fecha(D1,mayo,Y1),
  fecha(D,M,2024)=fecha(15,M,Y).
```

Para el primer objetivo Prolog internamente instancia las variable así:

```
D=D1, M=mayo, Y1=2024
```

Estructuras de Control: Unificación

EJEMPLO:

```
?-fecha(D,M,2024)=fecha(D1,mayo,Y1),
  fecha(D,M,2024)=fecha(15,M,Y).
```

Y para el segundo, la instancia se vuelve más específica respondiendo:

```
D=15
D1=15
M=mayo
Y1=2024
Y=2024
```

Estructuras de Control: Unificación

EJEMPLO:

```
%programa comunicado al sistema Prolog
nacimiento(pepe,fecha(16,11,1964)).
nacimiento(paula,fecha(22,11,1988)).
nacimiento(monica,fecha(4,4,1963)).


?-nacimiento(paula,fecha(22,X,Y)).
X=11
Y=1988
```

Estructuras de Control: Reglas de matching

⌘ Reglas generales para determinar si dos términos S y T hacen matching:

1. Si S y T son constantes, S y T hacen matching sólo si son el mismo objeto.

2. Si S es una variable y T es cualquier cosa, los términos hacen matching y S es instanciada a T. A su vez, T es una variable es instanciada a S.

3. Si S y T son estructuras, estas hacen matching si:

a) S y T tienen el mismo functor principal.

b) Las componentes correspondientes hacen matching.

La instanciación resultante es determinada por el matching de cada una de las componentes.

Estructuras de Control: Unificación

EJEMPLO:

```
?-triangulo(punto(1,1),A,punto(2,3))=triangulo(X,punto(4,Y),punto(2,Z)).
```

Estructuras de Control: Unificación

EJEMPLO:

```
?-triangulo(punto(1,1),A,punto(2,3))=triangulo(X,punto(4,Y),punto(2,Z)).
```

Internamente:

X=punto(1,1)

Estructuras de Control: Unificación

EJEMPLO:

```
?-triangulo(punto(1,1),A,punto(2,3))=triangulo(X,punto(4,Y),punto(2,Z)).
```

Internamente:

X=punto(1,1) A=punto(4,Y)

Estructuras de Control: Unificación

EJEMPLO:

```
?-triangulo(punto(1,1),A,punto(2,3))=triangulo(X,punto(4,Y),punto(2,Z)).
```

Internamente:

X=punto(1,1) A=punto(4,Y) Z=3

Estructuras de Control: Unificación

EJEMPLO:

```
?-triangulo(punto(1,1),A,punto(2,3))=triangulo(X,punto(4,Y),punto(2,Z)).
```

Internamente:

X=punto(1,1) A=punto(4,Y) Z=3

X=punto(1,1)

A=punto(4,Y)

Z=3

Estructuras de Control: Reevaluación

⌘ La reevaluación ("backtracking") se trata de volver a mirar lo que se ha hecho e intentar resatisfacer los objetivos buscando una forma alternativa de hacerlo.

⌘ Si se quiere obtener más de una respuesta a una consulta dada, puede iniciarse una reevaluación pulsando la tecla "y" cuando Prolog acaba de dar una solución y pregunta "More(y/n)?", con lo que se pone en marcha el proceso de **generación de soluciones múltiples.**

Estructuras de Control: Reevaluación

EJEMPLO:

```
%programa comunicado a Prolog

animal(mono).

animal(gallina).

animal(araña).

animal(mosca).

animal(cocodrilo).

gusta(mono,banana).

gusta(araña,mosca).

gusta(alumno,lógica).

gusta(araña,hormiga).

gusta(cocodrilo,X):-animal(X).

gusta(mosca,espejo).

regalo(X,Y):-animal(X),gusta(X,Y).
```

```
?-regalo(X,Y).
```

Estructuras de Control: Reevaluación

EJEMPLO:

```
?-regalo(X,Y).

X=mono, Y=banana

More (y/n)? y

X=araña, Y=mosca

More (y/n)? y

X=araña, Y=hormiga

More (y/n)? y

X=mosca, Y=espejo

More (y/n)? y

X=cocodrilo, Y=mono

More(y/n)? y

X= cocodrilo, Y=mosca

More (y/n)? n
```

```
%programa comunicado a Prolog

animal(mono).

animal(gallina).

animal(araña).

animal(mosca).

animal(cocodrilo).


gusta(mono,banana).

gusta(araña,mosca).

gusta(alumno,lógica).

gusta(araña,hormiga).

gusta(cocodrilo,X):-animal(X).

gusta(mosca,espejo).

regalo(X,Y):-animal(X),gusta(X,Y).
```

Gracias por su atención!!

DUDAS?