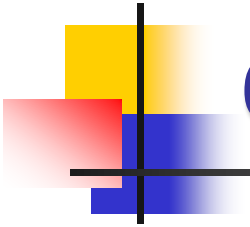
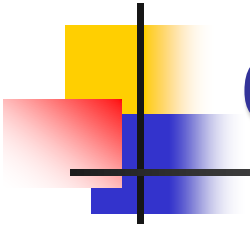


6. Operadores en PROLOG

1. Definición de operadores propios
2. Operadores predefinidos
   - Igualdad
   - Entrada y Salida básicos
   - Manejo de ficheros
   - Evaluación de expresiones aritméticas
   - Comparación de números
   - Introducción de nuevas cláusulas



6. Operadores en PROLOG

2. Operadores predefinidos (cont.)

- Cumplimiento y fracaso de objetivos
- Clasificación de términos
- Manejo de cláusulas como términos
- Construcción y acceso a componentes de estructuras
- Control de la re-evaluación
- Metavariables

1. Declaración de operadores (I)

- Utilización de un predicado predefinido:
  - ?- op(Precedencia, Tipo, Nombre).

```
?- op(500, yfx, por).
yes
?- X= a por b.
X= a por b
```

  - Precedencia $\in$ [1, 1200] y depende de implementaciones.
  - Tipo:
    - Aridad
    - Notación o posición
      - Infijo: xfy, yfx, xfx, yfy
        (f representa operador; x e y operandos)
      - Prefijo: fx, fy.
      - Sufijo: xf, yf

Declaración de operadores (II)

- Asociatividad:
  - "y" indica que el argumento puede contener operadores de igual o menor precedencia.
  - "x" obliga a que sea estrictamente menor.

  - Ejemplo: "+" declarado como yfx.

    "a+b+c": a+(b+c) ó (a+b)+c

    La **primera opción** queda eliminada porque el segundo argumento no puede contener un operador de precedencia igual.

  - Esto se conoce como asociatividad por la izquierda (yfx), por la derecha (xfy) o indistintamente (xfx); según esto, no tendría sentido (yfy).

2. Operadores predefinidos

- Ya hemos visto:
  - Pueden tener efectos laterales al satisfacerse, además de instanciar variables.
    - Ejemplo: corte
  - Pueden forzar a que los argumentos sean de un cierto tipo.
    - Ejemplo: < necesita dos argumentos numéricos

- Definiremos los predicados que *deberían* formar el núcleo del intérprete PROLOG, aunque habrá que revisarlos para cada versión

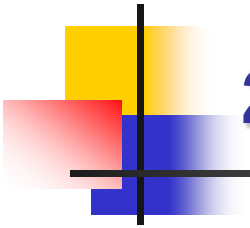
2.1. Igualdad

- X = Y.
  - Unificación: intenta hacer X e Y iguales mediante unificación.
- X \= Y.
  - Su opuesto: no unifican
- X == Y. Identidad
  - Más restrictivo que X = Y. Se satisface si X = Y, pero no necesariamente a la inversa.

```
?- X = Y.
?- X == X.
?- X = Y, X == Y.
?- append([A|B], C) == append(X, Y).
?- append([A|B], C) == append([A|B], C).
```

- X \== Y.
  - Su opuesto.

2.2. Entrada y Salida básicos

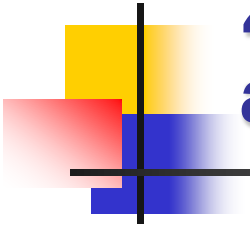


- Lectura
  - read(X),
  - get0(X),
  - get(X)
- Escritura
  - write(X),
  - display(X)
  - put(X)
- Formato:
  - nl
  - tab(X)
- skip(+Code)
  - skip(a).

2.3. Manejo de ficheros

- Lectura:
  - see(X),
  - seeing(X),
  - seen
- Escritura:
  - tell(X),
  - telling(X),
  - told

2.4. Evaluación de expresiones aritméticas



- X is Y.
  - Fuerza la evaluación de Y para unificar con X.
  - Los siguientes sólo se evalúan si están a la derecha de is:
- X + Y.
- X – Y.
- X * Y.
- X / Y.
- X mod Y.

2.5. Comparación de números

- X = Y.
- X \= Y.
- X < Y.
- X > Y.
- X >= Y.
- X =< Y.

2.6. Introducción de nuevas cláusulas

- consult(X).
  - Añade nuevas clausulas. Si ya están definidos predicados, añade clausulas al final.

  ?- consult(fichero).

  ?- consult(fichero1), consult(fichero2).

- reconsult(X).
  - Las clausulas leidas substituyen a las actuales.

  ?- reconsult(fichero3), reconsult(fichero4).

- Notación de lista:
  - *fichero* para consult.
  - *-fichero* para reconsult.

  ?- [fichero1, -'fichero3.pl', 'fichero2.pl', -'fichero4.pl']

2.7. Cumplimiento y fracaso de objetivos

- true
  - siempre se cumple

- fail
  - siempre fracasa
  - Utilidades:
    - ..., !, fail.

    si la ejecución llega hasta este punto, debemos dejar de intentar satisfacer el objetivo: fail lo hace fracasar y el corte anula la re-satisfacción.
    - ...,!, genera(X), muestra(X), fail.

    queremos recorrer todas las soluciones de un objetivo y, por ejemplo, mostrarlas.

2.8. Clasificación de términos

Permiten seleccionar ciertos tipos de términos para ser utilizados en las clausulas:

- var(X)
  - se satisface si en ese momento X es una variable no instanciada
  - permite saber si una variable ya tiene o no un valor, pero sin fijárselo como efecto lateral

  ?- var(X).

  yes

  ?- var(23).

  no

  ?- X = Y, Y = 23, var(X).

  ¿?
- nonvar(X)
  - comportamiento opuesto al anterior

Clasificación de términos (II)

- atom(X)
  - se cumple si X identifica en ese momento un átomo

  ?- atom(23).

  no

  ?- atom(libros).

  yes

  ?- atom(“esto es una cadena”).

  ¿?
- integer(X)
  - se satisface si X representa en ese momento a un entero
- atomic(X)
  - se cumple si en ese momento X es un entero o un átomo

  atomic(X):- atom(X).

  atomic(X):- integer(X).

2.9. Manejo de cláusulas como términos

PROLOG permite al programador examinar y modificar otros programas PROLOG:

- construir estructuras que representen cláusulas
- añadir clausulas
- retirar clausulas

- Para PROLOG una cláusula es una estructura

  le_gusta_a(juan, X) :- le_gusta_a(X, vino).

  - Una clausula es una estructura cuyo functor principal es ‘:-’ y cuyos argumentos son cabeza y cuerpo de la clausula.

    ‘:-’(le_gusta_a(juan, X), le_gusta_a(X, vino))

  - Si la cola tiene varios términos, deben considerarse unidos por el functor ‘,’.
    - abuelo(X, Z):- padres(X, Y), padres(Y,Z).
    - ‘:-’(abuelo(X, Z), ‘,’(padres(X,Y) padres(Y,Z)))

Manejo de cláusulas como términos (II)

- listing(A).
  - Se muestran por el canal de salida activo todas las cláusulas asociadas al átomo al que esté instanciado A.
  - El formato depende del intérprete.
  - Útil para descubrir errores.

  - Ejemplo:

```
?- [recorrer_fichero].
?- listing(mostrar_fichero).
mostrar_fichero :-
        write('Nombre de fichero: '),
        read(A),
        see(A),
        muestra_contenido,
        seen.
```

Manejo de cláusulas como términos (III)

- clause(X, Y).
  - Si se satisface hace coincidir X e Y con la cabeza y el cuerpo de una cláusula existente. Si no, falla.
  - Si no hay cláusulas asociadas, fracasa.
  - X debe tener información para identificar el predicado principal.
  - Se instancia a la primera cláusula y si se re-satisface, irán instanciándose a las siguientes cláusulas.
  - Para los hechos instancia el cuerpo a true.
  - Ejemplo:

```
añadir_sin_duplicar(X, L, L):- member(X, L), !.
añadir_sin_duplicar(X, L, [X|L]).
?- clause(añadir_sin_duplicar(X, U, V), Y).
U=V
Y = member(X,L),! ;

V = [X|U]
Y = true.
```

Manejo de cláusulas como términos (IV)

- asserta(X), assertz(X).
  - Añade una nueva clausula a la BC: *asserta()* lo hace al principio de la misma y *assertaz()* al final.
  - X debe estar suficientemente instanciada ser una cláusula válida.
  - Una vez satisfecho, no se deshace aunque haya reevaluación.
  - La única forma de eliminar entonces la cláusula es con:
- retract(X).
  - X debe estar suficientemente instanciada para identificar una cláusula válida.
  - Si se satisface, elimina la primera cláusula.
  - Si se reevalúa, eliminará las siguientes.

2.10. Construcción y acceso de componentes de estructuras

Se pueden escribir programas con cláusulas que hagan referencia a estructuras arbitrarias

- functor(T, F, N).
  - T es una estructura con functor F y aridad N
  - Si T está instanciada, fracasará si T no es un átomo o una estructura. Si no fracasa, nos dará el functor F y su aridad N
  - Si T no está instanciada, deben estarlo F y N. T devolverá las estructuras que verifiquen F y N.
  - Ejemplos:

```
?- functor(a+b, F, N).
F = +, N = 2
?- functor([a, b, c], F, N).
F = ., N = 2
?- functor([a, b, c], '.', 3).
No
```

Construcción y acceso de componentes de estructuras (II)

- arg(N, T, A).
  - N debe indicar una posición en los argumentos de T
  - Ambos deben estar instanciados
  - A será el argumento N-ésimo de T
- X =.. L
  - Llamado históricamente “univ”
  - L es la lista que consiste en el functor de X seguido de los argumentos de X
  - Permite:
    a) obtener todos los argumentos de una estructura
    b) construir una estructura a partir de una lista

Ejemplo:

?- algo(a, b, c) =.. L.

L = [algo, a, b, c]

Construcción y acceso de componentes de estructuras (III)

Ejemplo:

?- X =.. [f, a, b, c].

X = f(a, b, c).

- name(A, L)
  - Permite manejar átomos arbitrarios
  - Relaciona un átomo, A, con la lista de caracteres ASCII que lo compone, L

  - Ejemplos:

    ?- name(prueba, X).

    X = [112, 114, 117, 101, 98, 97]

    ?- name(prueba, “prueba”).

    yes

    ?- name(X, [112, 114, 117, 101, 98, 97]).

    X = prueba

2.11. Control de la reevaluación

- Predicado predefinido corte ó !
  - Detiene la búsqueda de nuevas soluciones
- repeat
  - Permite buscar soluciones alternativas.
  - Sería equivalente a tener definido:

```
repeat.
repeat :- repeat.
```

- Se usa en combinación con corte y fail

```
echo :- repeat, read(X), echo(X),!.
echo(X):- X=end_of_file, !.
echo(X):- write(X), nl, fail.
```

2.12. Metavariables

- call(X)
  - el objetivo call(X) se cumple si se satisface X como objetivo

  ?- padre(X).
  ?- call(padre(X)).
  ?-Meta=..[padre ,X], call(Meta).

- not(X)

  \+ padre(X).
  not(padre(X)).

- X, Y
  - conjunción de objetivos

- X ; Y
  - disyunción de objetivos