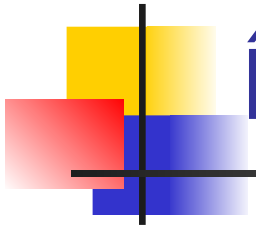
La reevaluación y el “corte”

UNIVERSIDAD DE VALLADOLID

Departamento de Informática



Índice:

1. Soluciones múltiples
2. El “corte”
3. Aplicaciones del “corte”
    - Confirmación de regla
    - Combinación corte-”fail”
    - Generación y comprobación

Soluciones múltiples (I)

- Generación de soluciones finitas.

  Ejemplo: cuaderno de baile

```
chico(juan).
chico(pedro).
chico(roberto).

chica(maria).
chica(ana).
chica(rosa).
chica(marta).

posible_pareja(X, Y):- chico(X), chica(Y).
```

Ejercicio: colocar el corte al principio, en medio y al final de la regla

```
?- posible_pareja(X, Y).
X = juan        Y = maria ;
X = juan        Y = ana ;
X = juan        Y = rosa ;
X = juan        Y = marta ;

X = pedro       Y = maria ;
X = pedro       Y = ana ;
X = pedro       Y = rosa ;
X = pedro       Y = marta ;

X = roberto     Y = maria ;
X = roberto     Y = ana ;
X = roberto     Y = rosa ;
X = roberto     Y = marta ;
```

Soluciones múltiples (II)

- Generación de soluciones infinitas.

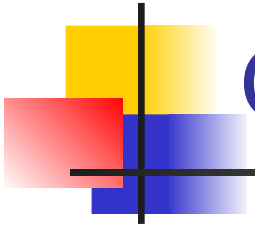
  Ejemplo: números naturales (recurrencia)

```
es_entero(0).
es_entero(X) :- es_entero(Y), X is Y+1.
```

```
?- es_entero(X)
```

```
X=0;
X=1;
X=2; etc.
```

- **Ejercicio**: hacer la pregunta es_entero(3). Colocar el corte en todos los posibles lugares y ver en dónde respondería bien a lo anterior. Comprobar si sigue generando la secuencia de



Corte (I)

- Ejemplo: Una biblioteca.
  - Libros existentes.
  - Libros prestados y a quién.
  - Fecha de devolución del préstamo.
  - Servicios básicos (accesibles a cualquiera):
    - Biblioteca de referencias o mostrador de consulta
  - Servicios adicionales (regla):
    - Préstamo normal o interbiblioteca.
  - Regla: no permitir servicios adicionales a personas con libros pendientes de devolución fuera de plazo.

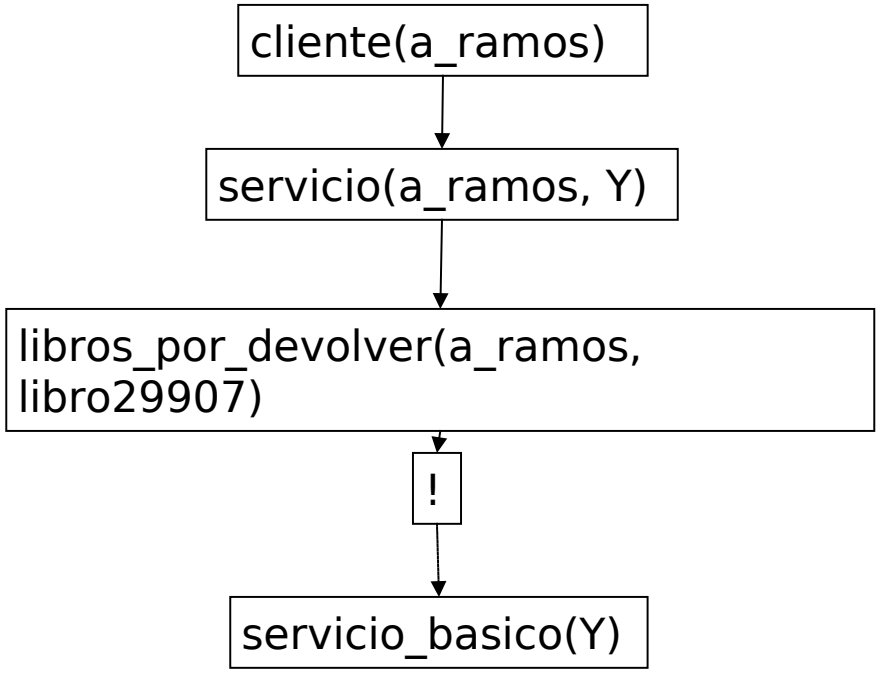
Corte (II)

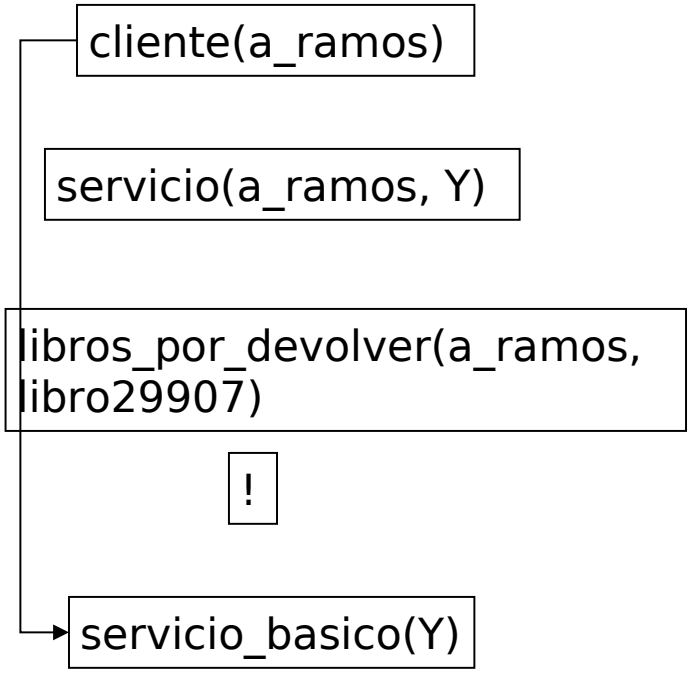
```
libros_por_devolver(c_perez, libro10089).
libros_por_devolver(a_ramos, libro29907).
.
.
.
cliente(a_ramos).
cliente(c_perez).
cliente(p_gonzalez).
.
.
```

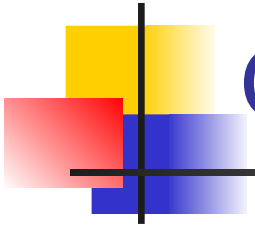
```
servicio(Pers, Serv) : -
        cliente(Pers),
        libros_por_devolver(Pers, Libro),
        !,
        servicio_basico(Serv).
servicio(Pers, Serv) :-
        servico_general(Serv).
servicio_basico(referencia).
servicio_basico(consulta).

servicio_adicional(prestamo).
servicio_adicional(pres_inter_biblio).

servicio_general(X) :- servicio_basico(X).
servicio_general(X) :-
servicio_adicional(X).
```

Corte (III)

?- cliente(X), servicio(X, Y).



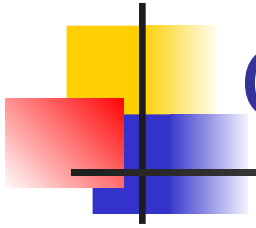




Corte (IV)

- Formalmente, el corte es un objetivo “!”, que siempre se satisface.
- Fuerza a no evaluar más objetivos a la izquierda dentro de la cláusula.
- Las variables se instancian al valor antes del “corte”.

Corte (V)



- Ahorro de tiempo no satisfaciendo objetivos que no contribuyen a solución alguna.
- Menor consumo de recursos.
- Herramienta para un correcto funcionamiento.
- Puede restar eficiencia ante preguntas no planteadas en el diseño del programa:

```
?-servicio(X, referencia).
?-cliente(X), servicio(X, referencia)
```

Corte: código más eficiente

- Diferencia absoluta de dos números:

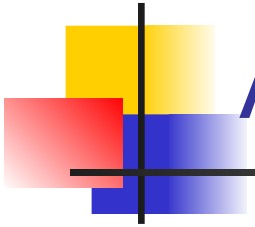
```
dif_abs(X,Y,Z) :- X>=Y, Z is X-Y.
dif_abs(X,Y,Z) :- X<Y, Z is Y-X.
```



```
dif_abs(X,Y,Z) :- X>=Y, !, Z is X-Y.
dif_abs(X,Y,Z) :- X<Y, Z is Y-X.
```

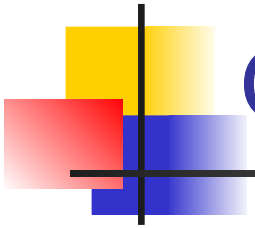


```
dif_abs(X,Y,Z) :- X>=Y, !, Z is X-Y.
dif_abs(X,Y,Z) :- Z is Y-X.
```



Aplicaciones del corte

- Decir a PROLOG: “si has llegado aquí, es que has escogido la regla adecuada para este objetivo”.
- Forzar el fracaso de un objetivo: justo lo opuesto del punto anterior.
- Finalizar la generación de soluciones alternativas mediante reevaluaciones: “si has llegado aquí, es que has encontrado la solución única y no hay razón para continuar”



Confirmación de regla

- Ejemplo: sumar los N primeros naturales

```
sumara(1, 1) :- !.
sumara(N, X) :- N1 is N-1, sumara(N1, Res), X is Res+N.

?- sumara(7, X).
```

X=28 (7+6+5+4+3+2+1)

```
sumara(N, N) :- N=<1, !.
sumara(N, X) :- N1 is N-1, sumara(N1, Res), X is Res+N.
```

Comprobar cómo funciona con números inferiores a uno.

Generación y comprobación (I)

- **Ejercicio**: plantear una relación dividir

```
divide(N1, N2, Resultado):-
        es_entero(Resultado),
        Producto1 is Resultado*N2,
        Producto2 is (Resultado+1)*N2,
        Producto1 =< N1,
        Producto2>N1, !.

es_entero(0).
es_entero(N) :- es_entero(Y), N is Y+1.
```

```
?- divide(100, 25, X).
X=4;
No

?- divide(100,3, X).
X=33;
No

?- divide(100, 5, 50)
<bloqueo>
```

- Generador -> relación *es_entero*
- Comprobador -> relación *divide*

Combinación corte-”fail” (I)

- “fail” es un predicado predefinido en PROLOG.
- Siempre produce un fracaso en la satisfacción del objetivo.
- Desencadena proceso de reevaluación.

```
carnet_uva(X):- matriculado(X),
fail.
matriculado(juan).
matriculado(pedro).
matriculado(maria).
matriculado(ana).
```

```
?- carnet_uva(X).

No
```

Combinación corte-”fail” (II)

- A Elena le gustan los animales, salvo las serpientes

```
animal(snoopy).
animal(lamia).
serpiente(lamia).
gusta(elena, X):- serpiente(X), !, fail.
gusta(elena, X):- animal(X).
```

```
?- gusta(elena,lamia).
no
?- gusta(elena,snoopy).
Yes
?-gusta(elena,X).
no
```

Combinación corte-”fail” (II)

- Se puede usar para simular funciones lógicas:

```
animal(snoopy).
animal(lamia).
serpiente(lamia).
gusta(elena, X):- animal(X), not(serpiente(X)).

?- gusta(elena,lamia).
no
?- gusta(elena,snoopy).
Yes
?-gusta(elena,X).
X = snoopy;

No
```