Prolog (un poco más) Avanzado

Lógica para Ciencias de la Computación

Primer Cuatrimestre de 2008

– Material Adicional –

Representación de Expresiones Aritméticas

- Las expresiones aritméticas presentan una estructuración recursiva:
  - E ::= E + E | E – E | (E)
- Prolog permite representar expresiones aritméticas mediante términos. Por ejemplo, podríamos adoptar la siguiente representación:
  - E ::= suma(E, E) | resta(E, E) | (E)

Ejemplo

Dada una expresión aritmética (codificada de acuerdo a la representación adoptada en la trasp. anterior), hallar su valuación asociada:

```
eval(suma(E1, E2), V) :-
        eval(E1, V1), eval(E2, V2), V is V1 + V2.

eval(resta(E1, E2), V) :-
        eval(E1, V1), eval(E2, V2), V is V1 - V2.

eval((E), V) :- eval(E,V).

eval(N, N) :- integer(N).
```

¡Esto trae problemas!

```
?- eval(resta(suma(2, 2), 1), Rsta).
Rsta = 3

?- eval([1,2,3], Rsta).
...
```

Representación Nativa

- Prolog permite representar expresiones aritméticas de manera aún más intuitiva:
  - Uso de nombres como +, -, *, etc. y notación infija:

```
+(1,*(2,3))
1+ *(2,3)
+(1,2*3)
(1+(2*3))
```

¡Todas estas instancias denotan el mismo término!

Representación Nativa

- Prolog permite representar expresiones aritméticas de manera aún más intuitiva:
  - Permite la especificación de expresiones sin parentizar, interpretándolas de acuerdo a la precedencia y asociatividad predefinidas asociadas a cada oprador:

    1+2*3 la interpreta como 1+(2*3)

    (Prolog asocia al * mayor precedencia que al +)

    10-5-2 la interpreta como (10-5)-2

    (Prolog establece que el - tiene asociativiada a izquierda)

Ejemplo

Dada una expresión aritmética en notación convencional, hallar su valuación asociada:

```
eval(E1 + E2, V) :-
        eval(E1, V1), eval(E2, V2), V is V1 + V2.

eval(E1 - E2, V) :-
        eval(E1, V1), eval(E2, V2), V is V1 - V2.

eval(N, N) :- integer(N).

?- eval((2 + 2) – 1, Rsta).
Rsta = 3
```

Operadores Definidos por el Programador

- El predicado predefinido op/3 permite definir nuevos operadores, o bien redefinir los existentes.
- La sintaxis es op(Prio, Clase, Nombre)
  - Prio es un entero denotando la prioridad del operador.
  - Clase especifica la clase de operador que se está definiendo (ver la tabla).
  - Nombre es la cadena de caracteres asociada al operador en cuestión.

Tipos de Operadores

Clase	Tipo	Asociatividad
fx	Prefijo	Ninguna
fy	Prefijo	A Derecha
xfy	Infijo	A Derecha
xfx	Infijo	Ninguna
yfx	Infijo	A Izquierda
xf	Posfijo	Ninguna
yf	Posfijo	A Izquierda

Multiplicidad de Soluciones

- Prolog suele ofrecer muchas formas de encarar la resolución de un problema.
- Por caso, para el problema *"determinar si una lista es una sublista de otra lista"*, se puede adoptar una resolución recursiva:
  - `sublista(Xs, Ys) :- prefijo(Xs, Ys).`
  - `sublista(Xs, [Y|Ys]) :- sublista(Xs, Ys).`

Multiplicidad de Soluciones

- O bien, caracterizarlo como sufijo de alguno de los posibles prefijos:
  - ```
    sublista(Xs, AsXsBs) :-
        prefijo(AsXs, AsXsBs),
        sufijo(Xs, AsXs).
    ```
- O bien, como un prefijo de alguno de los posibles sufijos:
  - ```
    sublista(Xs, AsXsBs) :-
        sufijo(XsBs, AsXsBs),
        prefijo(Xs, XsBs).
    ```

Multiplicidad de Soluciones

- O bien, como prefijo de un sufijo, pero usando `concat/3` de forma astuta:
  - ```
    sublist(Xs, AsXsBs) :-
            concat(As, XsBs, AsXsBs),
            concat(Xs, Bs, XsBs).
    ```
- O bien, como sufijo de un prefijo, usando `concat/3` de la misma forma:
  - ...
    ```

Predicados de 2do Orden

- Invocar un predicado Prolog suelen brindar exactamente una de las posibles soluciones.
- Existen predicados predefinidos que permiten acceder a todas las posibles soluciones.
- El predicado `findall/3` genera una lista con todas las posibles soluciones de un predicado.
- Este predicado nunca falla, ya que si no hay soluciones, retorna una lista vacía.

Ejemplo

```
menor(1,2).     menor(2,3).     menor(3,4).
menor(1,3).     menor(2,4).
menor(1,4).

?- findall(X, menor(X, 3), Rsta).
Rsta = [1, 2]
?- findall(X, menor(X, 1), Rsta).
Rsta = []
?- findall(sol(X), menor(X, 3), Rsta).
Rsta = [sol(1), sol(2)]
```

Ejemplo

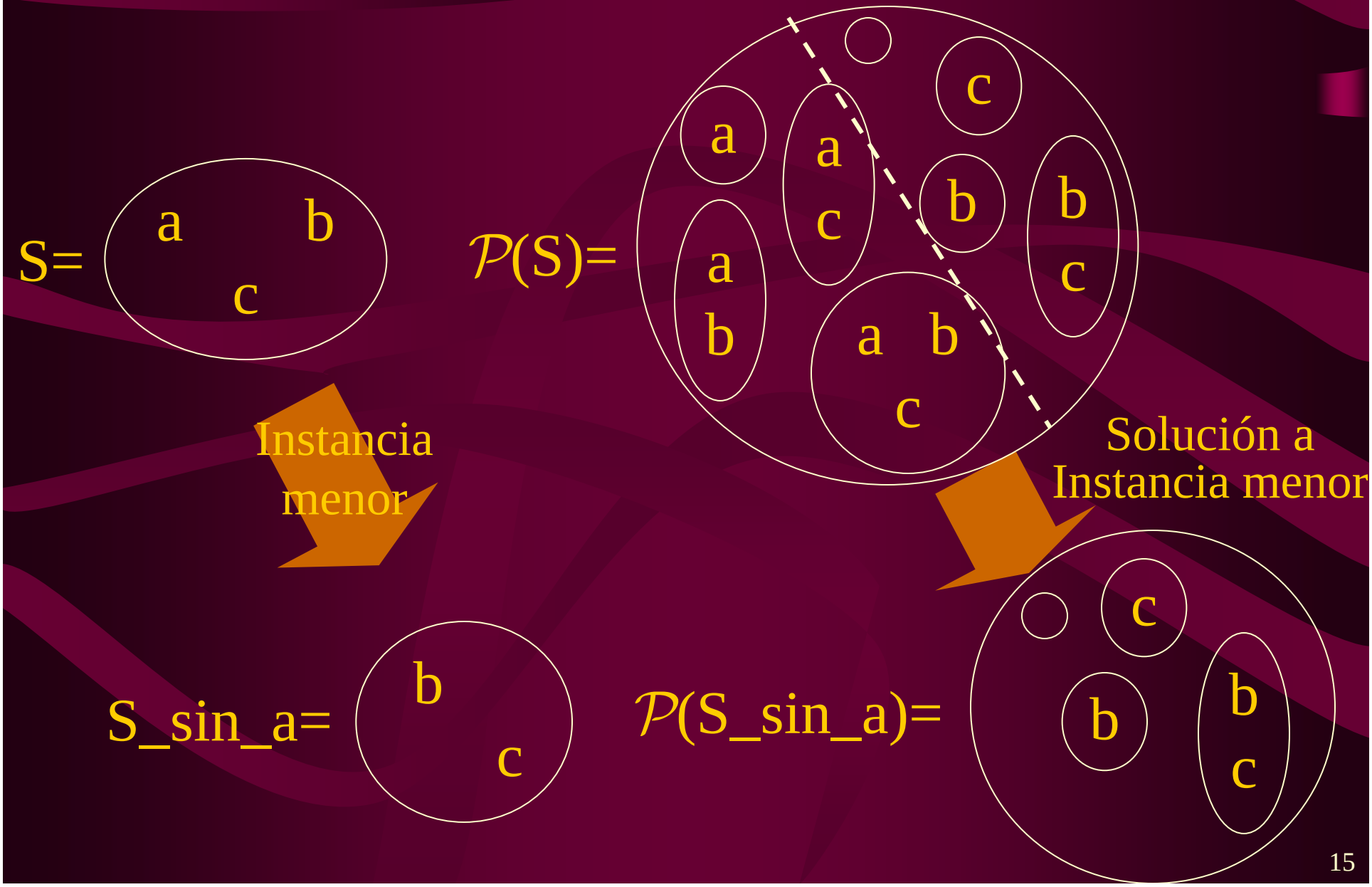
A partir de un conjunto, obtener el conjunto de sus partes.

- Recordar: partes de un conjunto S es el conjunto de todos los subconjuntos de S.
- Implementaremos un predicado partes/2 que establezca la relación entre un conjunto y su correspondiente conjunto de partes:

```
partes([], [[]]).
```

CB

Buscando un planteo recursivo



Ejemplo

A partir de un conjunto, obtener el conjunto de sus partes:

```
partes([], [[]]).
partes([X|Xs], P) :-
        partes(Xs, PXs),
        agregar(X, PXs, PXsConX),
        concat(PXsConX, PXs, P).

agregar(E, [], []).
agregar(E, [C|Cs], [[E|C]|ECs]) :-
        agregar(E, Cs, ECs).
```

esto hace a la vez de unión

Ejemplo

A partir de un conjunto, obtener el conjunto de sus partes:

```
partes(Xs, Ys) :-
        findall(Zs, subconj(Zs, Xs), Ys).

subconj([],[]).
subconj([X|Xs], [X|Ys]) :- subconj(Xs, Ys).
subconj(Xs, [Y|Ys]) :- subconj(Xs, Ys).
```

Predicados Extra-lógicos

- Prolog cuenta con predicados predefinidos cuyo semántica escapa a la lógica subyacente.
- El predicado write/1 escribe por pantalla lo que reciba como argumento.

```
?-
write(123).
123
yes
```

```
?- write('Hola mundo!').
Hola mundo!
yes
```

- Para finalizar una línea, se puede usar el predicado nl/0.

```
?- write('Hola '), write('mundo').
Hola mundo
yes
```

```
?- write('Hola '), nl, write('mundo').
Hola
mundo
yes
```

Simulando a findall/3

- El predicado predefinido fail/0 siempre falla.
- Puede ser utilizado para “simular” al predicado findall/3:
  - ?- subconj(SC, [1,2,3]), write(SC), nl, fail.
  - []
  - [1]
  - [2]
  - ...
- Más adelante encontraremos otros usos para este predicado.

Predicados Dinámicos

- Un predicados Prolog se especifica usualmente de forma estática, a partir de un archivo fuente.
- No obstante, los predicados Prolog también puede ser modificados en tiempo de ejecución.
- El predicado predefinido `assert/1` establece que el predicado recibido como argumento se verifica.
- El predicado predefinido `retract/1` establece que el predicado recibido como argumento ha dejado de verificarse.

Ejemplo

```
?- consult(prog).
```

prog.pl

```
:- dynamic b/0.
:- dynamic d/0.
a :- b.
c :- d.
d.
```

Ejemplo

```
?- consult(prog).
yes
?- a.
no
?- assert(b).
yes
?- a.
yes
```

Intérprete

```
a :- b.
c :- d.
d.
b.
```

dinámicos:
b/0, d/0.

Ejemplo

```
?- consult(prog).
```

prog.pl

```
:- dynamic b/0.
:- dynamic d/0.
a :- b.
c :- d.
d.
```

Ejemplo

```
?- consult(prog).
yes
?- c.
yes
?- retract(d).
yes
?- c.
no
```