Capítulo 1

Conjuntos. Aplicaciones. Relaciones de equivalencia. Relaciones de orden.

En esta práctica veremos cómo Maxima trae implementados algunos conceptos básicos sobre conjuntos, aplicaciones, relaciones de equivalencia y relaciones de orden.

Como ya sabemos, Maxima numera las sucesivas líneas de entrada como `(%i1)`, `(%i2)`, `(%i3)`, etcétera. En los guiones de prácticas de Maxima, cuando indiquemos la introducción de un comando, escribimos a la izquierda del mismo `(%ixx)`. Obviamente, el alumno sólo ha de escribir el comando en cuestión y no la cadena de caracteres `(%ixx)`. Por ejemplo, si escribimos

```
(%ixx) 1+1;
```

el alumno sólo ha de introducir 1+1 y pulsar la tecla Intro en el teclado numérico, ó bien pulsar al mismo tiempo las teclas Shift y Enter. El punto y coma final tampoco es necesario escribirlo. Lo añade Maxima.

1.1. Conjuntos

Lo primero es ver cómo introducir conjuntos. Tenemos, en principio, dos opciones:

- Enumerando los elementos separados por comas y encerrándolos entre llaves.

  ```
  (%ixx) A:{1,2,3,4,5,6,7,8,9};
  ```

  ```
  (%oxx) {1,2,3,4,5,6,7,8,9}
  ```

  De esta forma hemos creado un conjunto con nombre `A` y cuyos elementos son los números naturales del 1 al 9. Observe que para asignarle un valor a una variable, en Maxima se usa el símbolo "`:`" y no el símbolo "`:=`" más común en algunos lenguajes de programación.

  ```
  (%ixx) VACIO:{};
  ```

  ```
  (%oxx) {}
  ```

  Ahora hemos definido el conjunto vacio y le hemos llamado `VACIO`. Recuerde que Maxima a la hora de nombrar variables, distingue entre mayúsculas y minúsculas.

- Mediante la sentencia `set`.

  ```
  (%ixx) B:set(2,4,6,8,10,12,14,16);
  ```

  ```
  (%oxx) {2,4,6,8,10,12,14,16}
  ```

  Hemos creado un conjunto con nombre `B` cuyos elementos son los números naturales pares del 2 al 16.

Si escribimos elementos repetidos en la definición de un conjunto, Maxima deja sólo un elemento por cada grupo de repetidos, y además si los elementos son números, los escribe ordenados en orden creciente.

```
(%ixx) C:set(1,2,3,1,2,3,1,2,3,1,1,2);
(%oxx) {1,2,3}
```

Ahora vemos los comandos para calcular uniones, intersecciones, etc.

```
(%ixx) union(A,B);
(%oxx) {1,2,3,4,5,6,7,8,9,10,12,14,16}
(%ixx) intersection(A,B);
(%oxx) {2,4,6,8}
(%ixx) setdifference(A,B);
(%oxx) {1,3,5,7,9}
(%ixx) setdifference(B,A);
(%oxx) {10,12,14,16}
```

Con las últimas cuatro instrucciones hemos calculado $A \cup B$, $A \cap B$, $A \setminus B$ y $B \setminus A$.

También tenemos la posibilidad de preguntar algunas cuestiones, tales como si un elemento pertenece o no a un conjunto, si un conjunto es o no subconjunto de otro, etc.

```
(%ixx) elementp(5,A);
(%oxx) true
```

Le hemos preguntado si 5 es un elemento del conjunto `A` y su respuesta ha sido afirmativa devolviendo como salida `true`.

```
(%ixx) elementp(5,B);
(%oxx) false
```

Por tanto 5 no es un elemento del conjunto `B`.

Ahora le preguntamos si `A` es un subconjunto de `B`.

```
(%ixx) subsetp(A,B);
(%oxx) false
```

Otros ejemplos.

```
(%ixx) subsetp({2,6,10},B);
(%oxx) true
(%ixx) subsetp(VACIO,B);
(%oxx) true
(%ixx) elementp(5,VACIO);
(%oxx) false
```

También se puede preguntar a Maxima con el comando `is` si una expresión lógica se satisface ó no. Por ejemplo:

```
(%ixx) is(A=B);
(%oxx) false
```

Le hemos preguntado si los conjuntos `A` y `B` son iguales y la respuesta ha sido negativa. Veamos más ejemplos.

```
(%ixx) is(A=B or A=A);
(%oxx) true
```

Ahora le hemos preguntado si `A=B` ó `A=A`. Aunque lo primero es falso, como lo segundo es cierto, la disyunción de ambas afirmaciones es cierta y por eso devuelve `true`.

Para dos conjuntos cualesquiera A y B, se verifica que $(A \setminus B) \cap (B \setminus A) = \varnothing$. Comprobamos ésto para los conjuntos A y B que hemos definido previamente.

```
(%ixx) is(intersection(setdifference(A,B),setdifference(B,A))=VACIO);
(%oxx) true
```

El ejemplo siguiente ilustra la idea que ya hemos referido anteriormente, y es que el orden en el que escribimos los elementos de un conjunto es inmaterial.

```
(%ixx) is({1,2,3}={2,3,1});
(%oxx) true
```

Por supuesto el comando `is` también permite comparar expresiones aritméticas. Algunos ejemplos de ésto:

```
(%ixx) is(2<3);
(%oxx) true
(%ixx) is(1<3 and 2<=3);
(%oxx) true
```

El número de elementos de un conjunto finito se obtiene con el comando `cardinality`:

```
(%ixx) cardinality(B);
(%oxx) 8
(%ixx) cardinality(VACIO);
(%oxx) 0
```

Veamos cómo obtener el conjunto potencia ó conjunto de las partes de un conjunto dado.

```
(%ixx) powerset(C);
(%oxx) {{},{1},{1,2},{1,2,3},{1,3},{2},{2,3},{3}}
(%ixx) powerset(intersection(A,B));
(%oxx) {{},{2},{2,4},{2,4,6},{2,4,6,8},{2,4,8},{2,6},{2,6,8},{2,8},
{4},{4,6},{4,6,8},{4,8},{6},{6,8},{8}}
(%ixx) powerset(VACIO);
(%oxx) {{}}
```

Vamos a calcular, por ejemplo, el cardinal de $\mathcal{P}(A \cup B)$.

```
(%ixx) cardinality(powerset(union(A,B)));
(%oxx) 8192
```

que es 2^{13}.

Ejercicio 1. Utilice algunos comandos de Maxima para resolver los Ejercicios 1, 11(i) y 12 de la Relación de ejercicios del Tema 1.

A partir de lo visto, definimos una función que nos calcule la diferencia simétrica de dos conjuntos.

La diferencia simétrica de dos conjuntos A y B se define como $A \Delta B = (A \setminus B) \cup (B \setminus A)$, es decir, $A \Delta B$ es el conjunto formado por los elementos que están exactamente en uno de los dos conjuntos, A ó B. Introducimos lo siguiente:

```
(%ixx) dif_sim(X,Y):=union(setdifference(X,Y),setdifference(Y,X))$
```

Con ésto hemos definido una función llamada `dif_sim` que se aplica a dos argumentos designados por `X` e `Y`. Esta función devuelve el resultado de calcular $(\mathtt{X} \setminus \mathtt{Y}) \cup (\mathtt{Y} \setminus \mathtt{X})$.

Observe que ahora hemos usado el símbolo "`:=`" para definir a la función `dif_sim`. Recuerde, el símbolo "`:`" se usa para asignarle un valor a una variable, mientras que el símbolo "`:=`" se utiliza para definir una función.

Note también que hemos terminado la línea de definición de la función con el símbolo "$" en vez de con ";". Con ésto le estamos indicando a Maxima que no muestre en pantalla la salida correspondiente. Introduzca ahora el mismo comando pero acabando la línea con ";" para ver qué ocurre.

Ahora usamos la función que acabamos de definir.

```
(%ixx) dif_sim(A,B);
(%oxx) {1,3,5,7,9,10,12,14,16}
(%ixx) dif_sim(A,A);
(%oxx) {}
```

Ejercicio 2. Tal y como se dice en el Ejercicio 7 de la Relación de ejercicios del Tema 1, la diferencia simétrica también puede ser obtenida como $A\Delta B = (A \cup B) \setminus (A \cap B)$. Defina una función llamada `dif_sim2` que calcule la diferencia simétrica de dos conjuntos a partir de la expresión anterior, y compruebe con algunos ejemplos que las dos definiciones disponibles para la diferencia simétrica producen los mismos resultados.

Ejercicio 3. Escriba una función `subpro(X,Y)` que devuelva `true` si `X` es un subconjunto propio de `Y`, y `false` en caso contrario.

Si tenemos un conjunto X, podemos calcular el subconjunto formado por los elementos de X que satisfacen una determinada propiedad. Esto lo hacemos con la sentencia `subset` que tiene dos argumentos: el primero es el conjunto de partida, y el segundo una condición que deben cumplir los elementos para pertenecer al subconjunto.

```
(%ixx) f(x):=is(x>6)$
```

Con ésto hemos definido una función que se aplica a un objeto `x`, que suponemos es un número, y devuelve `true` si `x>6`, y `false` en caso contrario.

```
(%ixx) f(7);
(%oxx) true
(%ixx) f(5);
(%oxx) false
```

Ahora usamos `f` para obtener un subconjunto de `A`, el cual llamamos `D`.

```
(%ixx) A:{1,2,3,4,5,6,7,8,9}$
(%ixx) D:subset(A,f);
(%oxx) {7,8,9}
```

Observe que al usar en el comando `subset` la función `f` que hemos definido, no escribimos `f(x)`, sino el nombre de la función que es `f`.

Maxima tiene algunas funciones ya definidas y que en algunos ejemplos son útiles para especificar la condición, como por ejemplo `primep`, que nos dice si un número es o no primo, `oddp` o `evenp`, que nos dicen si un número es impar o par, respectivamente.

```
(%ixx) Primos:subset(A,primep);
(%oxx) {2,3,5,7}
```

Así, hemos obtenido los números primos que pertenecen a `A`. Recuerde que el número 1 por definición no es primo. Seguidamente calculamos los números impares pertenecientes a `A`.

```
(%ixx) Impares:subset(A,oddp);
(%oxx) {1,3,5,7,9}
```

Otra forma de obtener conjuntos es a partir de listas. Recordemos que una lista es una secuencia de objetos separados por comas y delimitados por corchetes. En una lista pueden haber objetos repetidos e importa el orden en el que éstos aparecen. Las listas se corresponden con lo que en teoría hemos definido como n-uplas.

```
(%ixx) lista:[1,2,3,4,2,5,1,2];
(%oxx) [1,2,3,4,2,5,1,2]
```

Así hemos definido una lista llamada `lista`.

La longitud de una lista se obtiene con el comando `length`.

```
(%ixx) length(lista);
(%oxx) 8
```

Veamos cómo podemos referirnos a los elementos de una lista.

```
(%ixx) lis:[78,-45,0,34];
(%oxx) [78,-45,0,34]
(%ixx) lis[1]; lis[2]; lis[3]; lis[4];
(%oxx) 78
(%oxx) -45
(%oxx) 0
(%oxx) 34
```

Podemos modificar algunos de sus valores.

```
(%ixx) lis[3]:11;
(%oxx) 11
(%ixx) lis[1]; lis[2]; lis[3]; lis[4];
(%oxx) 78
(%oxx) -45
(%oxx) 11
(%oxx) 34
```

Vea cómo el orden de los elementos ahora sí importa.

```
(%ixx) is([1,2,3]=[2,1,3]);
(%oxx) false
```

La función `setify` transforma una lista en un conjunto, mientras que la función `listify` transforma un conjunto en una lista.

```
(%ixx) setify([2,3,5,6,8,9,11,12]);
(%oxx) {2,3,5,6,8,9,11,12}
(%ixx) listify(A);
(%oxx) [1,2,3,4,5,6,7,8,9]
```

Vea lo que ocurre cuando pasamos a conjunto una lista con elementos repetidos.

```
(%ixx) lx:[1,2,2,3,3,3,4,4,4,4]$
(%ixx) setify(lx);
(%oxx) {1,2,3,4}
```

Como ejemplo, vamos a construir el conjunto de los números primos menores que 100. Para ésto, primero construímos una lista con los 100 primeros números, a continuación la pasamos a conjunto y por último la filtramos, quedándonos únicamente con los elementos que sean primos.

```
(%ixx) lx:makelist(i,i,1,100)$ D:setify(lx)$ P:subset(D,primep);
(%oxx) {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97}
```

Los dos primeros comandos que hemos introducido en la línea anterior acaban en $, pues no nos interesa que aparezcan todos los números del 1 al 100 en pantalla.

Con Maxima podemos construir el producto cartesiano de dos o más conjuntos.

```
(%ixx) A1:{a,b,c}$ A2:{3,5}$ cartesian_product(A1,A2);
(%oxx) {[a,3],[a,5],[b,3],[b,5],[c,3],[c,5]}
```

Con los tres comandos anteriores hemos definido los conjuntos A1 y A2, y hemos calculado el conjunto A1 $\times$ A2. Observe que el resultado es un conjunto de listas de longitud dos. Es decir, lo que en teoría representamos como pares ordenados, (x, y), en Maxima se representa como listas de longitud dos, [x,y]. Veamos otros ejemplos:

```
(%ixx) cartesian_product(A2,A1);
(%oxx) {[3,2],[3,b],[3,c],[5,2],[5,b],[5,c]}
(%ixx) cartesian_product({1,2,3},VACIO);
(%oxx) {}
(%ixx) D:{1,2}$ E:{3,5}$ cartesian_product(D,D,E);
(%oxx) {[1,1,3],[1,1,5],[1,2,3],[1,2,5],[2,1,3],[2,1,5],[2,2,3],[2,2,5]}
```

También es posible construir conjuntos con la instrucción makeset. Esta función tiene tres argumentos: una expresión, una lista de variables, y un conjunto de listas. Por ejemplo:

```
(%ixx) makeset(a+b,[a,b],cartesian_product(A,B));
(%oxx) {3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25}
```

Con ésto hemos calculado el conjunto formado por todos los elementos que pueden obtenerse sumando un elemento de A con uno de B. Al ser el resultado un conjunto, no hay elementos repetidos.

Ejercicio 4. Construya el conjunto formado por los números comprendidos entre 100 y 500 que son primos.

Ejercicio 5. Construya el conjunto formado por los números comprendidos entre 400 y 600, incluídos los dos extremos, que no son primos.

Ejercicio 6. Construya el conjunto formado por los números comprendidos entre 100 y 500, incluídos los dos extremos, que son suma de dos naturales al cuadrado. Por ejemplo 100, ya que $10^2 + 0^2 = 100$. También 146, ya que $5^2 + 11^2 = 146$.

Ejercicio 7. Construya el conjunto formado por los números comprendidos entre 100 y 700, incluídos los dos extremos, que al mismo tiempo son suma de dos naturales al cuadrado, y también son suma de dos naturales al cubo. Por ejemplo, $1 = 0^2 + 1^2 = 0^3 + 1^3,\ 72 = 6^2 + 6^2 = 2^3 + 4^3$.

1.2. Aplicaciones.

Cuando definimos una aplicación f en Maxima, no especificamos el dominio ni el codominio de f, sólo la expresión mediante la cual se calcula f.

Ya hemos visto en la sección anterior algunos ejemplos. Veamos algunos otros.

```
(%ixx) f(x):=3*x^2+8*x-1;
(%ixx) f(1);f(2);f(3);
(%oxx) 10
(%oxx) 27
(%oxx) 50
```

Podemos aplicar la función definida a una lista de números.

```
(%ixx) f([1,2,3]);
(%oxx) [10,27,50]
(%ixx) f([a,b,c,d]);
```

(%oxx) $[3a^2 + 8a - 1,\ 3b^2 + 8b - 1,\ 3c^2 + 8c - 1,\ 3d^2 + 8d - 1]$.

La composición de dos aplicaciones se escribe igual que hacemos en teoría.

```
(%ixx) f(x):=3*x^2;
(%ixx) g(x):=2*x-1;
(%ixx) g(f(x));
```

(%oxx) $6x^2 - 1$

```
(%ixx) f(g(x));
```

(%oxx) $3(2x-1)^2$

Si queremos que Maxima expanda las expresiones obtenidas, usamos el comando `expand`. Para ello escribimos `expand(%onumer)`, siendo `numer` el número de la salida que queremos expandir. Si escribimos `expand(%)`, estamos aplicando el comando a la última salida obtenida.

```
(%ixx) expand(%);
```

(%oxx) $12x^2 - 12x + 3$

Por tanto $3(2x-1)^2 = 12x^2 - 12x + 3$.

```
(%ixx) expand(f(g(x)));
```

(%oxx) $12x^2 - 12x + 3$

```
(%ixx) expand(g(f(x)));
```

(%oxx) $6x^2 - 1$

Por supuesto Maxima dispone de las funciones numéricas usuales como la exponencial, las funciones trigonométricas, etc, que el alumno podrá encontrar en la ayuda de Maxima.

Si tenemos una aplicación $f : S \to \mathbb{R} \times \mathbb{R}$, donde $S \subseteq \mathbb{R} \times \mathbb{R}$ y $f(x, y) = (2x + y, x - 3y)$, podemos definir f en Maxima de la forma siguiente:

```
(%ixx) f(x,y):=[2*x+y,x-3*y];
(%oxx) f(x,y):=[2*x+y,x-3*y]
(%ixx) f(1,1); f(0,0); f(-1,5);
(%oxx) [3,-2]
(%oxx) [0,0]
(%oxx) [3,-16]
```

Observe que f se aplica a dos números x e y, pero no a una lista de dos números. Si escribe lo siguiente, se obtiene un error precisamente por lo que estamos comentando.

```
(%ixx) f([1,1]);
```

Si queremos escribir f de modo que su argumento sea una lista formada por dos números, también es incorrecto introducir lo siguiente.

```
(%ixx) f([x,y]):=[2*x+y,x-3*y];
```

La forma de hacerlo es usar la referencia dentro de el argumento de entrada que se supone es una lista de longitud dos.

```
(%ixx) f(lis):=[2*lis[1]+lis[2],lis[1]-3*lis[2]];
(%ixx) f([1,1]); f([0,0]); f([-1,5]);
(%oxx) [3,-2]
(%oxx) [0,0]
(%oxx) [3,-16]
```

A continuación mostramos algunos ejemplos.

Ejemplo 1. Calcular la imagen de la aplicación $f : X \to \mathbb{Z}$, donde $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y $f(x) = x^2 - 10x + 21$.

```
(%ixx) X:{0,1,2,3,4,5,6,7,8,9};
```

```
(%ixx) f(x):=x^2-10*x+21;
```

Usamos el comando `map` que aplica la función `f` al dominio (conjunto) `X` y por tanto obtenemos la imagen de `f`.

```
(%ixx) map(f,X);
(%oxx) {-4,-3,0,5,12,21};
```

Ejemplo 2. Obtener la imagen de la aplicación $f : X \times X \to \mathbb{Z}$, donde $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y $f(x, y) = 2x - 3y$.

Primero construímos el dominio de f que llamamos `Dom`.

```
(%ixx) X:{0,1,2,3,4,5,6,7,8,9}$
(%ixx) Dom:cartesian_product(X,X)$
```

Observe que los dos comandos anteriores acaban con el símbolo \$.

Ahora definimos f y la aplicamos al dominio con lo que obtenemos la imagen de la aplicación.

```
(%ixx) f(lis):=2*lis[1]-3*lis[2];
(%ixx) map(f,Dom);
(%oxx) {-27,-25,-24,-23,-22,-21,-20,-19,-18,-17,-16,-15,-14,-13,-12,-11
(%oxx) ,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
(%oxx) ,16,18}
```

Hay otras formas de obtener el mismo resultado. Una de ellas consiste en usar el comando `makeset` que ya hemos usado en la sección anterior.

```
(%ixx) makeset(2*x-3*y,[x,y],Dom);
```

Ejemplo 3. Sean el conjunto $X = \{17, 19, 31, 33, 45, 57, 89\}$ y la aplicación $f : \mathcal{P}(X) \to \mathbb{N}$ tal que

$$f(A) = \sum_{a \in A} a$$

con $f(\varnothing) = 0$ por definición.

Es decir, f le hace corresponder a cada subconjunto A de X la suma de los elementos que pertenecen a A.

Primero definimos el dominio de f.

```
(%ixx) X:{17,19,31,33,45,57,89}$
(%ixx) Dom:powerset(X)$
```

Usaremos el comando `apply` que aplica un operador dado (en nuestro caso el operador suma) a los elementos de una lista para obtener un nuevo elemento (en nuestro ejemplo, la suma de todos los elementos que forman la lista). Por ejemplo:

```
(%ixx) apply("+",[5,11,13]);
(%oxx) 29
(%ixx) apply("+",[7]);
(%oxx) 7
(%ixx) apply("+",[]);
(%oxx) 0
```

La aplicación que nos interesa es la siguiente:

```
(%ixx) sumaelem(A):=apply("+",listify(A));
```

En la definición dada, suponemos que `A` es un conjunto, por lo que tenemos que transformalo previamente en una lista.

Probamos la función `sumaelem` con algunos ejemplos concretos.

```
(%ixx) sumaelem({31});
(%oxx) 31
(%ixx) sumaelem({31,33,45});
(%oxx) 109
(%ixx) sumaelem({});
(%oxx) 0
```

Finalmente obtenemos la imagen de la aplicación f.

```
(%ixx) map(sumaelem,Dom);
(%oxx) {0,17,19,31,33,36,45,48,50,52,57,62,64,67,69,74,76,78,81,83,88,89
(%oxx) ,90,93,95,97,100,102,105,106,107,108,109,112,114,119,120,121,122,124,125,
(%oxx) 126,128,133,134,135,137,138,139,140,141,145,146,150,151,152,153,154,156,
(%oxx) 157,158,163,165,166,167,169,170,171,172,177,179,182,183,184,185,186,189,
(%oxx) 191,194,196,198,201,202,203,208,210,213,215,217,222,224,227,229,234,239,
(%oxx) 241,243,246,255,258,260,272,274,291}
```

Ejercicio 8. Para cada una de las aplicaciones siguientes, obtenga el conjunto imagen.

1. $f : X \to \mathbb{Z}$, con $X = \{x \in \mathbb{Z} : -100 \leq x \leq 100\}$ y $f(x) = x^2 - 10x + 21$.
2. $f : X \times X \times X \to \mathbb{Z}$, donde $X = \{x \in \mathbb{Z} : 0 \leq x \leq 100\}$ y $f(x, y, z) = x^2 - xy + 3z$.
3. $f : \mathcal{P}(X) \to \mathbb{N}$, donde $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y $f(A)$ es el producto de todos los elementos pertenecientes a A. Por definición $f(\varnothing) = 1$.

Si tenemos una aplicación $f : A \to B$, con A y B conjuntos finitos, se verifica que:

1. f es inyectiva si y sólo si $|A| = |\mathrm{Imf}(f)|$.
2. f es sobreyectiva si y sólo si $|B| = |\mathrm{Imf}(f)|$.

Ejercicio 9. Para cada una de las aplicaciones siguientes, estudie si es inyectiva y si es sobreyectiva.

1. $f : X \to Y$ dada por $f(x) = x^2 - 5x + 6$ con $X = \{0, 1, 2, 3, 4, 5\}$ e $Y = \{0, 2, 4, 6, 8\}$.
2. $f : X \times X \times X \to Y$, donde $X = \{x \in \mathbb{Z} : 0 \leq x \leq 10\}$, $Y = \{x \in \mathbb{Z} : 0 \leq x \leq 930\}$ y $f(x, y, z) = x^2 + 8xy + 3z$.

Ejercicio 10. Utilice algunos comandos de Maxima para resolver los Ejercicios 14, 15, 16 y 25 de la Relación de ejercicios del Tema 1.

1.3. Relaciones de equivalencia.

En Maxima, dado un conjunto X podemos definir una relación de equivalencia en X. Para ello, utilizaremos el comando `equiv_classes`, que tiene dos argumentos. Un conjunto, y una expresión en dos variables que puede tomar los valores `true` (verdadero) ó `false` (falso).

Por ejemplo, vamos a construir el conjunto Z formado por todos los números enteros comprendidos entre -20 y 20, y vamos a definir sobre Z la relación binaria R siguiente: xRy si, y sólo si ambos números dan el mismo resto al dividir por 4. Recordemos que el resto de una división podemos calcularlo con la función `mod`.

```
(%ixx) z:makelist(i,i,-20,20)$ Z:setify(z)$
(%ixx) f(x,y):=is(mod(x,4)=mod(y,4))$
```

Ya tenemos definido el conjunto Z sobre el cual se define la relación de equivalencia R y una función auxiliar `f` que define precisamente a `R`. A continuación obtenemos el conjunto cociente y lo llamamos `Coc`.

```
(%ixx) Coc:equiv_classes(Z,f);
(%oxx) {{-20,-16,-12,-8,-4,0,4,8,12,16,20},{-19,-15,-11,-7,-3,1,5,9,13,17},{-18
,-14,-10,-6,-2,2,6,10,14,18},{-17,-13,-9,-5,-1,3,7,11,15,19}}
```

Observe que `Coc` es un conjunto que tiene cuatro elementos, cada uno de los cuales es una clase de equivalencia.

```
(%ixx) cardinality(Coc);
(%oxx) 4
```

Todos los números de la clase de equivalencia $\{-20, -16, -12, -8, -4, 0, 4, 8, 12, 16, 20\}$ dan resto 0 al ser divididos entre 4, todos los números de la clase de equivalencia $\{-19, -15, -11, -7, -3, 1, 5, 9, 13, 17\}$ dan resto 1 al ser divididos entre 4, etc.

Ejercicio 11. Defina en el conjunto Z visto en el ejemplo anterior, la relación de equivalencia R dada por: xRy si y sólo si el resto de dividir x^2 entre 4 es igual que el resto de dividir y^2 entre 4. Calcule el conjunto cociente, su cardinal y el cardinal de cada clase de equivalencia.

Ejercicio 12. Sea el conjunto $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. Definimos en $\mathcal{P}(X)$ la relación de equivalencia R tal que ARB si y sólo si la suma de los elmentos en A es igual a la suma de los elementos en B. Utilice los comandos apropiados de Maxima para calcular el conjunto cociente para R.

Ejercicio 13. Utilice algunos comandos de Maxima para resolver los Ejercicios 35 y 36 de la Relación de ejercicios del Tema 1.

1.4. Relaciones de orden.

Dado un conjunto X, sabemos que el conjunto de las partes de X, denotado por $\mathcal{P}(X)$, es un conjunto ordenado por inclusión.

Por ejemplo, si $S = \{A, B, C\} \subseteq \mathcal{P}(X)$, el supremo y el ínfimo de S son, respectivamente, $A \cup B \cup C$ y $A \cap B \cap C$, lo cual sabemos calcular con los comandos vistos en la Sección 1. Veamos un ejemplo concreto.

```
(%ixx) X:{0,1,2,3,4,5,6}$
(%ixx) S:{{4,5},{3,5},{2,4,5}}$
```

El supremo de S se obtiene con

```
(%ixx) union({4,5},{3,5},{2,4,5});
```

También podemos obtenerlo con el comando siguiente:

```
(%ixx) apply(union,listify(S));
```

Como el conjunto resultante no pertenece a S, inferimos que S carece de máximo. Aquí esta comprobación se hace de forma visual, pero si S fuera un conjunto muy grande, podríamos utilizar el comando `elementp` visto en la Sección 1.

El ínfimo de S se obtiene mediante

```
(%ixx) intersection({4,5},{3,5},{2,4,5});
```

Tal y como está pensando, también podemos obtenerlo con

```
(%ixx) apply(intersection,listify(S));
```

Ya que el conjunto resultante no pertenece a S, inferimos que S carece de mínimo.

Los elementos maximales de S son $\{3,5\},\{2,4,5\}$ y los elementos minimales de S son $\{3,5\},\{4,5\}$. Observe que en este ejemplo hay elementos que son simultáneamente maximales y minimales.

El conjunto de los minorantes de S se obtiene como el conjunto potencia del ínfimo de S.

```
(%ixx) powerset(intersection({4,5},{3,5},{2,4,5}));
```

El conjunto de los mayorantes de S es

$$\{A \cup \mathrm{Sup} : A \in \mathcal{P}(X \setminus \mathrm{Sup})\},$$

donde Sup es el supremo de S. En nuestro ejemplo ya sabemos que $\mathrm{Sup} = \{2,3,4,5\}$. Una forma de obtener el conjunto de los mayorantes de S es la siguiente.

```
(%ixx) Sup:union({4,5},{3,5},{2,4,5});
(%ixx) setify(makelist(union(A,Sup),A,listify(powerset(setdifference(X,Sup)))));
```

Nótese que resulta un conjunto de conjuntos, cada uno de los cuales contiene a los elementos 2, 3, 4 y 5.

Ejercicio 14. Dado el conjunto $X = \{0,1,2,3,4,5,6,7,8,9,10\}$, obtenga los elementos notables del subconjunto $S = \{\{1,2,6,8,9\},\{1,2,4,6,8,9\},\{2,6,7,8\}\}$ de $\mathcal{P}(X)$.

1.5. El orden de los objetos en Maxima

Los objetos de Maxima (símbolos, números y cadenas) se ordenan atendiendo al siguiente criterio: **Enteros** y **decimales en coma flotante** preceden a **constantes declaradas** (que representan al número π, e, etc), que a su vez preceden a las **cadenas de caracteres**, es decir, secuencias de caracteres encerradas entre dobles comillas. Por último, éstas preceden a nombres de variable que no tienen valores asignados.

Este orden queda reflejado en la función booleana `orderlessp`.

```
(%ixx) orderlessp(4,5);
(%ixx) orderlessp(4,4.0 );
(%ixx) orderlessp(4.0 ,4);
(%ixx) orderlessp(4,abc);
(%ixx) abc:1;
(%ixx) orderlessp(4,abc);
(%ixx) orderlessp(%pi,7);
```

El número π vale aproximadamente 3,14159265. Sin embargo la constante predeclarada en Maxima `%pi` que representa a dicho número, es considerada por el comando `orderlessp` mayor que la constante numérica 7.

```
(%ixx) is(3+%pi<7);
```

Como se aprecia, al usar `%pi` es una expresión aritmética, sí se toma su valor numérico.

```
(%ixx) orderlessp(400,"12");
(%ixx) orderlessp("tara","taza")
```

Para ordenar los elementos de una lista, Maxima dispone del comando `sort`.

Consideramos algunos ejemplos:

```
(%ixx) sort([%e,e,3,a,b,4,5,c,d,"xyz",abc,%pi,"xxyz",3.4,%phi,3.25]);
```

Para ver lo que valen aproximadamente las constantes predefinidas %e, %pi, %phi, %gamma, escribimos:

```
(%ixx) %e,numer;
(%ixx) %pi,numer;
```

```
(%ixx) %phi,numer;
(%ixx) %gamma,numer;
```

Para que tenga en cuenta el valor numérico de las constantes a la hora de ordenar una lista, escribiremos:

```
(%ixx) sort([%pi,3,4,%e,%gamma,%phi], "<");
```

1.6. Los órdenes producto cartesiano y lexicográfico

El comando que obtiene el producto cartesiano de dos o más conjuntos, genera los elementos del conjunto resultante ordenados lexicográficamente. Compruebe ésto viendo la salida del comando siguiente.

```
(%ixx) cartesian_product({1,2,3},{a,b},{4,5,6});
```

Hemos obtenido un conjunto de listas de longitud 3. Maxima asume que $1 < 2 < 3$, $a < b$ y $4 < 5 < 6$.

Es más, el comando `orderlessp` mencionado en la sección anterior, cuando se aplica a dos listas (no necesariamente de igual longitud) devuelve `true` sólo cuando la primera es estrictamente menor que la segunda en el orden lexicográfico. Introduzca los comandos siguientes y compruebe lo que acabamos de decir.

```
(%ixx) orderlessp([1,8,3],[4,5,6]);
(%ixx) orderlessp([1,8,3],[1,8,3]);
(%ixx) orderlessp([4,8,3],[4,5,6]);
(%ixx) orderlessp([1,8,3],[4,2,2,7,8]);
(%ixx) orderlessp([1,8,3,1,1,5],[4,2,2]);
```

El comando `reverse(L)` se aplica a una lista `L` y devuelve otra lista formada por los elementos de `L` escritos en orden inverso.

```
(%ixx) reverse([4,8,a,2,-3,0,b,s]);
(%ixx) reverse([1,2,3,2,1]);
```

Seguidamente implementamos un predicado o función lógica (es decir, aquella que devuelve el valor `true` ó `false`) que compara dos listas `li1` y `li2` pero mirándolas de derecha a izquierda. Por tanto este predicado implementa el orden lexicográfico inverso (o por la derecha):

```
(%ixx) lexinv(li1,li2):=orderlessp (reverse (li1), reverse (li2));
```

Lo probamos con algunos ejemplos:

```
(%ixx) lexinv([0,-3,4,7],[1,0,1,7]);
(%ixx) lexinv([0,-3,-4,7],[1,0,1,7]);
```

Si tenemos los conjuntos $A = \{1, 2, 3\}, B = \{a, b\}, C = \{8, 9\}$, obtenemos los elementos del conjunto $A \times B \times C$ ordenados según el orden lexicográfico inverso de la forma siguiente:

```
(%ixx) sort(listify(cartesian_product({1,2,3},{a,b},{8,9})),lexinv);
```

Ejercicio 15. Sean los conjuntos $A = \{1, 2, 3, 4, 5, 6, 7, 8\}$ y $H = A \times A \times A$.

1. Si consideramos H ordenado según el orden producto cartesiano, escriba algunas instrucciones en Maxima para obtener todos los elementos $\alpha \in H$ tales que $(2, 1, 3) \leq \alpha \leq (5, 6, 7)$.

2. Si consideramos H ordenado según el orden lexicográfico (por la izquierda), escriba algunas instrucciones en Maxima para obtener todos los elementos $\alpha \in H$ tales que $(2, 3, 3) \leq \alpha \leq (5, 1, 4)$.

(Sugerencia: En primer lugar hay que generar el conjunto H. No olvide usar el delimitador \$. Así tenemos un conjunto de listas de longitud tres. A continuación en cada apartado hay que escribir una función booleana que se aplique a una lista `L` y devuelva `true` si y sólo si `L` está comprendida entre las dos listas especificadas, es decir, entre $[2, 1, 3]$ y $[5, 6, 7]$ en el primer apartado, y entre $[2, 3, 3]$ y $[5, 1, 4]$ en el segundo. Por último usamos el comando `subset` que ya hemos visto en la Sección 1.)