

# Set Theory - An Introduction to Maple

Mth 355/399 Sept 28 2001 Maple 6

Bent E. Petersen

Filename: 355f2001\_set\_theory.mws

## The Worksheet

When you are using Maple in a windowing environment it is possible to move around on the worksheet by left-clicking the mouse. As a result, commands may end up being executed in a nonlinear order. This can cause some confusion, since there is no visual clue. One way to fix a mess is to have Maple re-execute the whole worksheet (look on the Edit menu). This works best if old expressions are cleaned up first, so it is a good idea to start each worksheet with the command restart; You do not need to do so of course ....

```
> restart;
```

Maple commands are executed by pressing the Enter key when the mouse cursor is in the line containing the commands.

Note each Maple command must be terminated by a colon or a semicolon (except help commands preceded by a question mark). You can spread the command over several lines by postponing the terminating colon or semicolon. You simply move to a new line by pressing Enter. Maple will chatter at you when you move to a new line in this manner if the previous command is unterminated. Ignore it, but keep in mind a command will not be executed before it is properly terminated.

The assignment operator in Maple is := (colon and equals sign juxtaposed).

You can also stack up several commands on one line by terminating them individually with colons or semicolons. The effect of the colon is to suppress output from the corresponding command, though the command is still carried out. All the commands on a line are executed when you press the Enter key.

Maple has two ditto operators, % and %%. The value of % is the previous evaluated expression, the value of %% is the one before that. Since the Worksheet commands may be executed in any order, the ditto operators can cause a lot of confusion. It is probably best to restrict them to the same line as the expressions they refer to. Here is a silly example, which also demonstrates the assignment operator.

Here's a useful fact: You can open a new command line below the current one by pressing Ctrl-J, or above the current line, by pressing Ctrl-K. This is pretty useful when you realize you omitted something at a certain step.

Functions in maple may be defined in several ways - the easiest is often the arrow notation. For example here is the Euler polynomial that produces so many primes

```
> p:=x->x^2+x+41;
```

$$p := x \rightarrow x^2 + x + 41$$

```
> for k from 1 to 40 do; k, ifactor(p(k)), isprime(p(k)); od;
```

```
1, (43), true  
2, (47), true  
3, (53), true  
4, (61), true  
5, (71), true  
6, (83), true  
7, (97), true  
8, (113), true  
9, (131), true  
10, (151), true  
11, (173), true  
12, (197), true  
13, (223), true  
14, (251), true  
15, (281), true  
16, (313), true  
17, (347), true  
18, (383), true  
19, (421), true  
20, (461), true  
21, (503), true  
22, (547), true  
23, (593), true  
24, (641), true  
25, (691), true  
26, (743), true  
27, (797), true  
28, (853), true  
29, (911), true  
30, (971), true  
31, (1033), true  
32, (1097), true
```

```

33, (1163), true
34, (1231), true
35, (1301), true
36, (1373), true
37, (1447), true
38, (1523), true
39, (1601), true
40, (41)2, false

```

Maple has a few set theory commands built in. Let's apply them to some simple problems modified from our text.

```

> U:={1,2,3,4,5,6,7,8,9,10,11,12}; # small universe
      U := {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
> A:={3,5,7,9}; B:={2,3,5,6,7}; C:={2,4,6,8};
      A := {3, 5, 7, 9}
      B := {2, 3, 5, 6, 7}
      C := {2, 4, 6, 8}
> A union B;
      {2, 3, 5, 6, 7, 9}
> B intersect C;
      {2, 6}
> B minus A;
      {2, 6}
> A minus B;
      {9}

```

We can define our own absolute complement function (relative to our selected universe):

```

> complement:=X->U minus X;
      complement := X → U minus X
> complement(A);
      {1, 2, 4, 6, 8, 10, 11, 12}
> complement(B);
      {1, 4, 8, 9, 10, 11, 12}
> complement(C);
      {1, 3, 5, 7, 9, 10, 11, 12}
> (A union B) minus C;
      {3, 5, 7, 9}

```

```

> complement(A union B) intersect complement(B union C);
      { 1, 10, 11, 12 }
> (A union C) minus (C minus A);
      { 3, 5, 7, 9 }

```

The symmetric difference measures by how much two sets differ. We can easily define our own symmetric difference function.

```

> symdiff:=(X,Y)->(X minus Y) union (Y minus X);
      symdiff := (X, Y) → (X minus Y) union (Y minus X)
> symdiff(A,B);
      { 2, 6, 9 }
> symdiff(A,C);
      { 2, 3, 4, 5, 6, 7, 8, 9 }
> symdiff(B,C);
      { 3, 4, 5, 7, 8 }

```

We can easily test membership

```

> member(3,A);
      true
> member(4,A);
      false

```

Here's a simple way to define a procedure `carp()` to compute the Cartesian product. This is a bit more complicated than the arrow notation for functions, but is much more flexible.

```

> carp:=proc(X,Y)
>   local Z,x,y;
>   Z:={};
>   for x in X do
>     for y in Y do
>       Z:=Z union {[x,y]};
>     od;
>   od;
>   return Z;
> end;
carp := proc(X, Y)
local Z, x, y;
      Z := { }; for x in X do for y in Y do Z := Z union { [x, y] } end do end do; return Z
end proc

```

```

> carp(A,A);
{[3, 3], [3, 5], [3, 7], [3, 9], [5, 3], [5, 5], [5, 7], [5, 9], [7, 3], [7, 5], [7, 7], [7, 9], [9, 3],
  [9, 5], [9, 7], [9, 9]}
> carp(A,B);
{[3, 3], [3, 5], [3, 7], [5, 3], [5, 5], [5, 7], [7, 3], [7, 5], [7, 7], [9, 3], [9, 5], [9, 7], [3, 2],
  [3, 6], [5, 2], [5, 6], [7, 2], [7, 6], [9, 2], [9, 6]}
> carp(A,C);
{[3, 2], [3, 6], [5, 2], [5, 6], [7, 2], [7, 6], [9, 2], [9, 6], [3, 4], [3, 8], [5, 4], [5, 8], [7, 4],
  [7, 8], [9, 4], [9, 8]}
> carp(A,B) minus carp(A,C);
{[3, 3], [3, 5], [3, 7], [5, 3], [5, 5], [5, 7], [7, 3], [7, 5], [7, 7], [9, 3], [9, 5], [9, 7]}

```

We can devise our own test for subsets

```

> subset:=proc(X,Y)
>   local x,s;
>   s:=true;
>   for x in X do
>     s:= s and member(x,Y);
>   od;
> end;
subset := proc(X, Y) local x, s; s := true; for x in X do s := s and member(x, Y) end do end proc
> subset(A,B);
false
> subset({9,5},A);
true

```

Here's a slicker way to check for a subset. Here evalb() means evaluate Boolean, that is, find the truth value of a statement.

```

> subset2:=(X,Y)->evalb((X minus Y)={});
subset2 := (X, Y) → evalb(X minus Y = { })
> subset2(A,B);
false
> subset2({9,5},A);
true

```

The power set may be computed by using the Maple function choose() from the the combinat package. To use it we have either to load the combinat package by issuing the command with(combinat) or we have to call the function by using its so called long name combinat[choose](). We will use the second method but we will rename the function to the more convenient powset().

```
[ > alias(powset=combinat[choose]);  
                                     powset  
[ > powset(A);  
  {{ }, {9}, {3, 5, 7, 9}, {3, 5, 7}, {5, 9}, {5, 7, 9}, {7, 9}, {3, 7, 9}, {3, 9}, {3, 5, 9}, {3},  
   {5}, {3, 5}, {7}, {3, 7}, {5, 7}}  
[ Try to be reasonable. If you decide to compute powset(carp(A,B)) you will have a long wait. This set  
  has 2^20 elements!  
[ > 2^20;  
                                     1048576  
[ It will take a while to list over 1 million elements.  
[ >
```