

Macros

Sample Macro: Dieder_groups.omf

General Remarks:

This macro generates the [Dihedral Group \$D_n\$](#) as a subgroup of the group of all permutations of (i.e. bijective mappings on) the set $\{0, 1, 2, \dots, n-1\}$. In this case, the group operation is the product of functions. As generators of D_n we choose the permutations $a := (n-1 \ n-2 \ \dots \ 2 \ 1 \ 0)$ of order 2 and $b := (1 \ 2 \ 3 \ \dots \ n-1 \ 0)$ of order n . D_n is a subgroup of the automorphism group on $\{0, 1, 2, \dots, n-1\}$. It is generated by the pairwise products of $\{a, b\}$. This process is repeated until no new elements are generated.

The macro contains several subroutines, i.e. lambda-macros, for calculations which are used several times. Several other lambda-macros are conceivable and could simplify and optimize this macro. For example, the calculation and adding of newmap (cf. [below](#)) could be put into a separate lambda-macro.

Implementation (Download plain source-code: [Dieder_groups.omf](#)):

```
setstarttime(d_t); % This initialises the variable d_t, which measures the time. If you
are not interested in measuring time, this step can be omitted. %
```

```
% This subroutine concatenates two operations given as arrays / lists / n-tuples. %
```

```
composemap:=lambda(m1,m2)
  mn:=emptyset;
  for i in m2 do
    additem(mn,item(m1,i)); % mn(i):=m1(m2(i)) %
  endfor;
  mn;
endlambda;
```

```
% This subroutine tests if two operations (given as lists) are identical. %
```

```
samemap:=lambda(m1,m2)
  result:=true;
  for j in card(m1) do
    if not(equal(item(m1,j),item(m2,j))) then result:=false else noop endif
  endfor;
  result
endlambda;
```

```
% This subroutine tests if the operation nmap is already contained in the list
listofmaps. %
```

```
alreadyinlist:=lambda(nmap)
  res:=false;
```

```

for m in listofmaps do
  if samemap(nmap, m) then res:=true else noop endif
endfor;
res
endlambda;

```

% This subroutine determines the index of the operation *nmap* in the list *listofmaps*.
%

```

indexofmap:=lambda(nmap)
  res:=(0,1);
  for m in card(listofmaps) do
    if samemap(nmap, item(listofmaps,m)) then res:=m else noop endif
  endfor;
  res
endlambda;

```

% This subroutine determines the name of operation *m* on by means of its index in the list *listofmaps*. %

```

nameofmap:=lambda(m)
  item(namesofmaps,indexofmap(m))
endlambda;

```

% Here begins the actual program. Input of integer *n* for D_n . %

```

string_n:=prompt("n for Dn");
if tonumber(n,string_n) then
% listofmaps is a list of the group elements taken as maps, i.e. lists (()) namesofmaps contains the group elements' names. %

```

```
listofmaps:=emptyset;
```

```
namesofmaps:=emptyset;
```

% *templist* is a temporary list; with its help the individual group elements are generated. %

```
templist:=emptyset;
```

% In *templist* the element *a* is created as the permutation (n-1 n-2 ... 2 1 0); it is then inserted in *listofmaps* and *listofnames*. %

```
for i in n do templist:=insertintolist(templist, 0, i) endfor;
```

```
additem(listofmaps, templist);
```

```
additem(namesofmaps, "a");
```

% In *templist* the element *b* is created as the permutation (1 2 3 ... n-1 0); it is then inserted in *listofmaps* and *listofnames*. %

```
templist:=emptyset;
```

```
for i in n do if >(i,0) then additem(templist, i) else noop endif endfor;
```

```
additem(templist, 0);
```

```
additem(listofmaps, templist);
```

```
additem(namesofmaps, "b");
```

% Now the pairwise products of those operations already in *listofmaps* are formed, until there are no more new operations created, i.e. until *weiter* after the for-loops

```

turns false. This is equivalent to the generation of the dihedral group as a
subgroup of corresponding () permutation group. weiter is a flag which checks
whether or not new elements are still created by the pairwise products. %
weiter:=true;
while weiter do
  weiter:=false;
  for map1 in listofmaps do
    for map2 in listofmaps do
      % newmap is the product of map1 and map2 as permutations. (return to intro) %
      newmap:=composemap(map1,map2);
      % If newmap is not yet in listofmaps, it's added. weiter is set to true. %
      if not(alreadyinlist(newmap)) then
        additem(listofmaps,newmap);
        additem(namesofmaps, concat(nameofmap(map1),nameofmap(map2)));
        weiter:=true
      else noop endif;
      % Now the same is done for the converse product, i.e. for the product of map2
and map1 as permutations. %
      newmap:=composemap(map2,map1);
      % If newmap is not yet in listofmaps, it's added. weiter is set to true. %
      if not(alreadyinlist(newmap)) then
        additem(listofmaps,newmap);
        additem(namesofmaps, concat(nameofmap(map2),nameofmap(map1)));
        weiter:=true
      else noop endif
    endfor;
  endfor;
endwhile;
% Output: %
writeln(listofmaps);
writeln(namesofmaps);
% Now the result is inserted as a structure into the current work environment. %
% Generate structure: %
dn:=addstructure(concat("D",string_n));
% Adds elements. The new elements are arranged along the window diagonal: %
for i in card(listofmaps) do addelement(dn, item(namesofmaps,i), *(+(i,1),20),
*(+(i,1),20)) endfor;
% Adds operation. After that, completes the table with the operations from
listofmaps. %
op:=addtooperations(dn,"*",2,1);
for i1 in card(listofmaps) do for i2 in card(listofmaps) do
  setvalue(op, [item(dn,i1),item(dn,i2)],
item(dn,indexofmap(composemap(item(listofmaps,i1), item(listofmaps,i2))))))
endfor endfor
else writeln("keine Zahl!"); endif;
% Finally, the time measurement. Measures how long the whole thing took to

```

calculate. %

```
writeln(timeasstring(d_t));
```

The computer program "Algebra Workbench" (AWB) was created by Markus Sprenger. The documentation found here is based on a 2005 master thesis by Christoph Röthlisberger. The translation and adaptation of the material was done by Cindy-Jane Armbruster.

This page was designed by [cja](#) in 2006. It was last updated on July 30, 2006.