

Beispielmakro: Dieder_groups.omf

Allgemeine Bemerkungen:

Die Diedergruppe D_n wird in diesem Makro als Untergruppe der Gruppe aller Permutationen (d.h. bijektiven Abbildungen auf) der Menge $\{0, 1, 2, \dots, n-1\}$ erzeugt. Gruppenoperation ist in diesem Fall die Verknüpfung von Abbildungen. Als erzeugende Elemente für D_n werden die Permutationen $a := (n-1\ n-2\ \dots\ 2\ 1\ 0)$ der Ordnung 2 und $b := (1\ 2\ 3\ \dots\ n-1\ 0)$ der Ordnung n gewählt. Aus diesen beiden Gruppenelementen wird D_n als die von $\{a,b\}$ erzeugte Untergruppe der Automorphismengruppe auf $\{0, 1, 2, \dots, n-1\}$ berechnet, indem ganz einfach solange paarweise verknüpft wird, bis nichts neues mehr erzeugt wird.

Das Makro enthält diverse "Subroutinen", d.h. lambda-makros für mehrfach verwendete Berechnungsschritte. Das Makro ist sicher nicht optimiert, so könnte man z.B. die Berechnung und das Hinzufügen von `newmap` ([siehe unten](#)) in ein weiteres lambda-makro packen, was eine weitere Vereinfachung bedeuten würde.

Die Implementation (Download: unformatierter Quelltext in [Dieder_groups.omf](#)):

```
setstarttime(d_t); % Die Variable d_t für die Zeitmessung wird initialisiert. Dieser Schritt kann auch weggelassen werden, wenn keine Zeitmessung angestrebt wird. %
```

```
% Dies ist eine Subroutine, die zwei als Arrays / Listen / n-Tupel gegebene Operationen verkettet. %
```

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

```
% Diese Subroutine testet, ob zwei als Listen gegebene Operationen identisch sind. %
```

```
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;
```

```
% Diese Subroutine testet, ob die Operation nmap bereits in der Liste listofmaps enthalten ist. %
```

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

```

endlambda;

% Diese Subroutine ermittelt den Index der Operation nmap in der Liste listofmaps. %
indexofmap:=lambda(nmap)
  res:=(o,i);
  for m in card(listofmaps) do
    if samemap(nmap, item(listofmaps,m)) then res:=m else noop endif
  endfor;
  res
endlambda;

% Diese Subroutine ermittelt den Namen der Operation m anhand des Indices von m in in der
Liste listofmaps. %
nameofmap:=lambda(m)
  item(namesofmaps,indexofmap(m))
endlambda;

% Hier beginnt das eigentliche "Hauptprogramm". Eingabe der Zahl n für  $D_n$ . %
string_n:=prompt("n for Dn");
if tonumber(n,string_n) then
  % listofmaps ist die Liste der Gruppenelemente als Abbildungen, d.h. Listen, während
namesofmaps die Bezeichnungen der Gruppenelemente enthält. %
listofmaps:=emptyset;
namesofmaps:=emptyset;
% templist ist eine temporäre Liste, mit der die einzelnen Gruppenelemente gebastelt werden.
%
templist:=emptyset;
% In templist wird das Element a als Permutation (n-1 n-2 ... 2 1 o) erzeugt und dieses
anschliessend in listofmaps und listofnames eingefügt. %
for i in n do templist:=insertintolist(templist, o, i) endfor;
additem(listofmaps, templist);
additem(namesofmaps, "a");
% In templist wird das Element b als Permutation (1 2 3 ... n-1 o) erzeugt und dieses anschliessend
in listofmaps und listofnames eingefügt. %
templist:=emptyset;
for i in n do if >(i,o) then additem(templist, i) else noop endif endfor;
additem(templist, o);
additem(listofmaps, templist);
additem(namesofmaps, "b");
% Nun werden die bereits in listofmaps vorhandenen Operationen paarweise verknüpft, und
dies solange, bis dadurch keine neue Operation mehr entsteht, d.h. weiter nach den for-
Schleifen false ist. Dies entspricht der Erzeugung der Diedergruppe als Unterppuppe der
entsprechenden Permutationsgruppe. Weiter ist ein Flag, mit dem überprüft wird, ob durch die
paarweise Verknüpfung noch neue Elemente erzeugt werden oder nicht. %
weiter:=true;
while weiter do
  weiter:=false;
  for map1 in listofmaps do
    for map2 in listofmaps do
      % newmap ist die Verknüpfung von map1 und map2 als Permutationen. (zurück zur intro)
      %
      newmap:=composemap(map1,map2);

```

```

    % Falls newmap noch nicht in listofmaps ist, wird es hinzugefügt, und weiter wird auf true
    gesetzt. %
    if not(alreadyinlist(newmap)) then
        additem(listofmaps,newmap);
        additem(namesofmaps, concat(nameofmap(map1),nameofmap(map2)));
        weiter:=true
    else noop endif;
    % Nun noch dasselbe für die umgekehrte Verknüpfung, d.h. die von map2 und map1 als
    Permutationen. %
    newmap:=composemap(map2,map1);
    % Falls newmap noch nicht in listofmaps ist, wird es hinzugefügt, und weiter wird auf true
    gesetzt.. %
    if not(alreadyinlist(newmap)) then
        additem(listofmaps,newmap);
        additem(namesofmaps, concat(nameofmap(map2),nameofmap(map1)));
        weiter:=true
    else noop endif
endfor;
endfor;
endwhile;
% Ausgabe: %
writeln(listofmaps);
writeln(namesofmaps);
% Nun wird das Resultat als Struktur in die aktuelle Arbeitsumgebung eingefügt. %
% Struktur erzeugen: %
dn:=addstructure(concat("D",string_n));
% Elemente hinzufügen, wobei die neuen Elemente einfach auf der Fensterdiagonale angeordnet
werden: %
for i in card(listofmaps) do addelement(dn, item(namesofmaps,i), *(+(i,i),20), *(+(i,i),20)) endfor;
% Operation hinzufügen und anschliessend Tabelle anhand der Operationen in listofmaps
ausfüllen. %
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;
% Zuletzt Zeitmessung, wie lange das ganze gedauert hat. %
writeln(timeasstring(d_t));

```