Groups and Rings - SF2729

Homework 1 (Rings)

Jim Holmström - 890503-7571

Exercise 1. Let $R \in M_2(\mathbb{Z}_2)$. Prove that R has exactly 9 divisors of 0. Prove that $R^* \cong S_3$.

Solution. An element $a \neq 0$ is a divisor of $0 \Leftrightarrow \exists b, c \neq 0 : ba = ac = 0$ Didn't find any nice things to use in this problem so I'm using the exhausted search technique, to avoid lots of hand calculations I made a program for this.

```
import operator
import copy
import itertools as itt
import math

#===================Helpers===========================================
def int2bits(i,n,zero_element=0,one_element=1):
    return list((zero_element,one_element)[i>>j & 1] for j in xrange(n-1,-1,-1))

def maplist(a,indices):
    return map(lambda i:operator.getitem(a,i),indices)

#==================Printers==========================================
def M2R2_printer(r):
    print r
    print
    return r

def M2R2_list_printer(rs,pre_r=None):
    """
    Fundamental flaw: doesnt wrap well
    pre_r is if you want to have an start M2R2 seperated from the others being first
    """
    firstline=""
```

```
    secondline=""
    if pre_r:
        firstline+=(str(pre_r.bits[0])+str(pre_r.bits[1]))
        firstline+=" | "
        secondline+=(str(pre_r.bits[2])+str(pre_r.bits[3]))
        secondline+=" | "

    for r in rs:
        firstline+=(str(r.bits[0])+str(r.bits[1])+ " ")
    for r in rs:
        secondline+=(str(r.bits[2])+str(r.bits[3])+" ")

    print firstline
    print secondline

def print_iso(iso):
    for x,y in iso.iteritems():
        print str(x),"=",str(y)
    print "-------------"

#==================Algebra definitions of the group/ring ==============
class perm_n:
    """
    NOTE 1-indexed
    """
    perm=[] # NOTE non cyclic representation
    def __init__(self,n,elem=1):
        self.n=n
        if isinstance(elem,list):
            self.perm=elem
        else:
            assert(1<=elem<=math.factorial(n))
            self.perm=list(itt.permutations(range(1,n+1)))[elem-1]

    def __call__(self,i):
        assert(0<i<=self.n)
        return self.perm[i-1]

    def __str__(self):
        return str(self.perm)

    def __mul__(self,other):
        assert(self.n==other.n)
        return perm_n(self.n,map(self,other.perm))
```

```
    def __eq__(self,other):
        assert(self.n==other.n)
        return all(map(operator.eq,self.perm,other.perm))

    def __ne__(self,other):
        return not operator.__eq__(self,b)

    def __hash__(self):
        return sum(map(lambda (a,k):a**k,zip(self.perm,range(1,len(self.perm)+1))))

class M2R2:
    def __init__(self,elem=0):
        """
        0->0 (important)
        one to one map (important)
        """
        if isinstance(elem,list):
            self.bits=elem
        else:
            self.bits=int2bits(elem,4)

    def __str__(self):
        return str(self.bits[0])+str(self.bits[1])+"\n"+str(self.bits[2])+str(self.bits[3]
);

    def __eq__(self,b):
        return all(map(operator.eq,self.bits,b.bits))
    def __ne__(self,b):
        return not operator.__eq__(self,b)

    def __add__(self,other):
        return M2R2(map(operator.xor,self.bits,b.bits))

    def __mul__(self,other):
        a=map(operator.and_,maplist(self.bits,[0,0,2,2]),maplist(other.bits,[0,1,0,1]))
        b=map(operator.and_,maplist(self.bits,[1,1,3,3]),maplist(other.bits,[2,3,2,3]))
        return M2R2(map(operator.xor,a,b))

    def __hash__(self):
        return sum(map(lambda (a,k):([2,3][a])**k,zip(self.bits,range(1,len(self.bits)+1
))))

#=======Setup==========================================================
```

```
R = map(lambda r:M2R2(r),range(16)) #enumerate all elements in M_2(Z_2)
Zero=M2R2(0)
Rstar=copy.copy(R)
Rstar.remove(Zero) #R\{0}

print "Commutative?",all(map(lambda (a,b):a*b==b*a,itt.product(R,repeat=2)))

print "e="
e=filter(lambda i: all(map(lambda b:i*b==b,R)),R) #e*b=b \forall b \in R
assert len(e)==1 #generalized to ensure the uniqueness of e
e=e[0]
print e

#======Assignment=====================================================
#TODO generalize and push to github
print "Divisors of zero"
# \exists b\neq 0 :ab=0
DOZ_left=filter(lambda a:any(map(lambda b:a*b==Zero,Rstar)),Rstar)
# \exists b\neq 0 :ba=0
DOZ_right=filter(lambda a:any(map(lambda b:b*a==Zero,Rstar)),Rstar)

#pickout the elements that are both left and right
DOZ=filter(lambda a:a in DOZ_left,DOZ_right)
M2R2_list_printer(DOZ)
```

Which returns the divisors:

```
Divisors of zero
00 00 00 01 01 10 10 11 11
01 10 11 00 01 00 10 00 11
```

And they are 9 in number □

To generate $U(M_2(Z_2))$ and find isomorphisms:

```
print "Group of units"
UM2R2=filter(lambda a:any(map(lambda b:a*b==b*a==e,R)),R) # \exists a:ab=ba=e
M2R2_list_printer(UM2R2)

N=3
Perms=map(lambda i:perm_n(N,i+1),range(math.factorial(N))) #in this case=S_3
```

```
#generate all possible isos
isos= map(lambda S:dict(zip(UM2R2,S)),itt.permutations(Perms,math.factorial(N)))

#filter out all isos that preserve the structure
valid_isos = filter(lambda iso: all( map(lambda (x,y):iso[x*y]==iso[x]*iso[y],
itt.product(UM2R2,repeat=2))),isos)

print "Valid isos"
map(print_iso,valid_isos)
```

Which returns the group of units:

```
Group of units
01 01 10 10 11 11
10 11 01 11 01 10
```

and all 6 possible isomorphisms (where the left side is a matrix and the right a non-cyclic notated permutations:

```
Valid isos
10
01 = (1, 2, 3)
11
01 = (3, 2, 1)
01
11 = (2, 3, 1)
10
11 = (2, 1, 3)
01
10 = (1, 3, 2)
11
10 = (3, 1, 2)
-------------
10
01 = (1, 2, 3)
11
01 = (2, 1, 3)
01
11 = (3, 1, 2)
```

```
10
11 = (3, 2, 1)
01
10 = (1, 3, 2)
11
10 = (2, 3, 1)
-------------
10
01 = (1, 2, 3)
11
01 = (1, 3, 2)
01
11 = (2, 3, 1)
10
11 = (3, 2, 1)
01
10 = (2, 1, 3)
11
10 = (3, 1, 2)
-------------
10
01 = (1, 2, 3)
11
01 = (3, 2, 1)
01
11 = (3, 1, 2)
10
11 = (1, 3, 2)
01
10 = (2, 1, 3)
11
10 = (2, 3, 1)
-------------
10
01 = (1, 2, 3)
11
01 = (2, 1, 3)
01
11 = (2, 3, 1)
10
11 = (1, 3, 2)
01
10 = (3, 2, 1)
11
```

```
10 = (3, 1, 2)
-------------
10
01 = (1, 2, 3)
11
01 = (1, 3, 2)
01
11 = (3, 1, 2)
10
11 = (2, 1, 3)
01
10 = (3, 2, 1)
11
10 = (2, 3, 1)
-------------
```

Seems to be in this case as long as the elements has the same order ($\phi(x^n) = \phi(x)^n = e$) they can be transformed in any way and still be a isomorphism. □

The code used can be downloaded from:
http://www.f.kth.se/~jimho/sf2729/m2r2_test.py

Exercise 2. $G = (\mathbb{Z}_{1026})^*$. **Prove that** $g^{18} = 1 \forall g \in G$

Solution. Didn't find any easy solution so did it the hard-way, to avoid hand calculations a made a script.

```
import operator
import copy
import itertools as itt
import string
import math

#=======Printers==========================
def print_listing(listing):
    line=""
    for g,has in listing.iteritems():
        line+=(str(g)+" | ")
        for h in has:
            line+=string.center(str(h),6)
        print line
        line=""

#=======Ring definition==================
class Zn:
```

```
    def __init__(self,n,i):
        """
        Initz Z_n with the element i
        """
        assert 0<=i<n
        self.n=n
        self.i=i

    def __str__(self):
        """
        You are on your own on tracking n, mostly one has the same n
        """
        return str(self.i)

    def __eq__(self,other):
        """
        Must be of the same Zn to be the same
        """
        return self.n==other.n and self.i==other.i
    def __ne__(self,other):
        return not operator.__eq__(self,other)

    def __add__(self,other):
        assert self.n==other.n
        return Zn(self.n,(self.i+other.i)%self.n)
    def __mul__(self,other):
        assert self.n==other.n #not defined else
        return Zn(self.n,(self.i*other.i)%self.n)
    def __pow__(self,m):
        """
        return g**n
        """
        return Zn(self.n,(self.i**m)%self.n)

    def __hash__(self):
        return self.i

#========Setup==================================
N=1026
m=18
Z=map(lambda i:Zn(N,i),range(N)) #Generate all elements
Zero=Zn(N,0) #Generate zero
One=Zn(N,1) #Generate one
```

```
#========Assignment============================
#generate U(Z_1026)
UZ=filter(lambda b:any(map(lambda g:g*b==b*g==One,Z)),Z) #\exists g:gb=bg=1

#is all g^18= \forall g \in U(Z_1026)
print "g^18=1 forall g?",all(map(lambda g:g**18==One,UZ))
```

Which returns true, showing by bruteforce that $g^{18} = 1 \forall g \in U(Z_{1026})$

The code used can be downloaded from:
http://www.f.kth.se/~jimho/sf2729/zn_test.py