GAP ile Grup Cebirlerin Lie Cebirleri

Ahmet Faruk ASLAN

YÜKSEK LİSANS TEZİ
Matematik Anabilim Dalı
Ocak 2010

Lie Algebras of Group Algebras with GAP

Ahmet Faruk ASLAN

MASTER DISSERTATION
Department of Mathematics
January 2010

GAP ile Grup Cebirlerin Lie Cebirleri

Ahmet Faruk ASLAN

Eskişehir Osmangazi Üniversitesi

Fen Bilimleri Enstitüsü

Lisans üstü Yönetmeliği Uyarınca

Matematik Anabilim Dalı

Cebir ve Sayılar Teorisi Bilim Dalında

YÜKSEK LİSANS TEZİ

Olarak Hazırlanmıştır

Danışman: Prof. Dr. Zekeriya ARVASİ

Ocak 2010

ONAY

Matematik Anabilim Dalı yüksek lisans öğrencisi Ahmet Faruk ASLAN' ın YÜKSEK LİSANS TEZİ olarak hazırladığı "**GAP ile Grup Cebirlerin Lie Cebirleri**" başlıklı bu çalışma, jürimizce lisans üstü yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Danışman : Prof. Dr. Zekeriya ARVASİ

İkinci Danışman : –

Yüksek Lisans Tez Savunma Jürisi:

Üye : Prof. Dr. Zekeriya ARVASİ

Üye : Prof. Dr. Mahmut KOÇAK

Üye : Yrd. Doç. Dr. Ummahan EGE ARSLAN

Üye : Yrd. Doç. Dr. Enver Önder USLU

Üye : Yrd. Doç. Dr. Alper ODABAŞ

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Prof. Dr. Nimetullah BURNAK

Enstitü Müdürü

GAP ile Grup Cebirlerin Lie Cebirleri

Ahmet Faruk ASLAN

ÖZET

Cebirin bir bilgisayar uygulaması olan GAP, yeni matematiksel yapıların bilgisayar ortamına aktarılmasında kullanılan güçlü bir programlama dilidir. GAP oldukça gelişmiş, anlaşılması kolay ve serbest kullanıma açık bir dile sahiptir. Özellikle grup teorisinde güçlü olan GAP birçok işletim sisteminde kullanılabilir.

Tezin birinci bölümünde GAP programlama dili ve GAP fonksiyon yazımı üzerinde durulmuştur. İkinci bölümde cebir, grup cebir ve Lie cebiri kavramları temel özellikleriyle verilerek GAP fonksiyonları için teorik altyapı edinilmiştir.

Bovdi, Konovalov, Rossmanith ve Schneider tarafından yazılan ve Lie cebirlerin birçok özelliğini grup cebirler aracılığıyla bilgisayar ortamına aktarımı olan LAGUNA, GAP ortak paketi üçüncü bölümde incelenmiştir.

Whitehead tarafından tanımlanan çaprazlanmış modüller önemli bir cebirsel sistemdir. Çaprazlanmış modül, bazı fiziksel problem çözümlerinde iki boyutlu cebirsel yapı olarak düşünülebilir. Dördüncü bölümde Lie cebirlerin çaprazlanmış modülleri incelenmiş ve tez kapsamında yazılan GAP programı fonksiyonları verilmiştir.

Anahtar Kelimeler: GAP, Grup Cebir, Lie Cebirler, Çaprazlanmış modüller.

Lie Algebras of Group Algebras with GAP

Ahmet Faruk ASLAN

SUMMARY

The powerful computer algebra system GAP provides a high level programming language with several advantages for coding of new mathematical structures. It has a highly developed, easy to understand programming language, incorporated. It is especially powerful for group theory.

In the first section of the thesis GAP programming language and GAP function writing has been asserted. In the second section of the thesis by given with basic properties of algebra, group algebra and Lie algebra concepts, a theoric basis has been gained for GAP functions.

The LAGUNA, GAP share package, a transferring of most of the properties of Lie algebras via group algebras into computer environment and written by Bovdi, Konovalov, Rossmanith and Scheider, has been analyzed in the third section.

Crossed modules defined by Whitehead are significant algebra system and can be considered as two-dimensional algebraic structure in solving some physical problems.

In the forth section of the thesis crossed modules of Lie algebras have been analyzed and given the GAP programme functions which were written as a part of the thesis.

Keywords: GAP, Group algebra, Lie algebras, Crossed Modules of Lie algebras.

TEŞEKKÜR

Beni bu çalışmaya sevkeden ve yöneten, çalışma boyunca değerli yardımlarını esirgemeyen,

Sayın hocalarım,

Prof. Dr. Zekeriya ARVASİ *ve* **Yrd. Doç Dr. Alper ODABAŞ'**a

desteklerini hep yanımda hissettiğim aileme sonsuz saygı ve teşekkürlerimi sunarım.

Bu tez çalışması TUBİTAK-TBAG 107T542 nolu araştırma projesi tarafından desteklenmiştir.

İÇİNDEKİLER

BÖLÜM 0

ÖNSÖZ

0.1 Giriş

GAP (Group, Algorithm and Programming) cebirin bilgisayar aracılığı ile hesaplanması için geliştirilen bir bilgisayar paket programıdır. GAP programı açık kaynak kodludur ve kullanıcıları tarafından geliştirilmektedir. Bu amaçla yazılan ortak paketler, yazılan kullanım kılavuzu ile birlikte St. Andrews deki GAP merkezine gönderilir. Buradaki hakemlerin değerlendirmesi sonucunda uygun bulunan paketler program kütüphanesine eklenerek dünya üzerindeki tüm kullanıcılara internet üzerinden ücretsiz olarak dağıtılır ve yazılan kullanım kılavuzu uluslararası kitapta bir bölüm olarak sunulur.

GAP kullanıcısının `RequirePackage(\program-kompleksi")` komutu girmesiyle ortak paket içerisindeki fonksiyonlar çalışır hale getirilir.

Çaprazlanmış modül kavramı, (Whitehead, 1949) tarafından tanımlanmıştır. Whitehead, özellikle relatif homotopi gruplarının cebirsel yapıları üzerine yaptığı çalışmasında çaprazlanmış modüllere yer vermiştir. O zamandan itibaren çaprazlanmış modül kavramı diğer alanlarda da önemli bir yer tutmuştur. Bu konuda yapılan önemli çalışmalardan bazıları (Brown, 1982,1984) , (Brown ve Higgins, 1981) , ve (Brown ve Huebschmann, 1981) dir.

GAP programı kullanılarak gruplar üzerinde çaprazlanmış modüllerin bazı özellikleri (Brown ve Wensley, 1995,1996,2003) ile (Alp ve Wensley, 2000) çalışmalarında incelenmiştir. Asosyatif ve değişmeli cebirler üzerinde çaprazlanmış modül kavramı, farklı bir adla (Lichtenbaum, Schlessinger, 1967) ve (Gerstenhaber, 1966) çalışmalarında karşımıza çıkar. (Porter, 1986) çalışmasında değişmeli cebirler üzerinde çaprazlanmış modül kavramını tanımlamıştır. Bununla birlikte, (Arvasi ve Porter, 1996) çalışmalarında değişmeli cebirler için çaprazlanmış modüllerle ilgili birçok önemli sonuçlar elde etmişlerdir. (Odabaş, 2009) çalışmasında cebirler üzerinde çaprazlanmış modülleri ve Cat^1-cebirleri bilgisayar ortamına aktarmıştır.

(Kassel ve Loday, 1982) çalışmasında Lie cebirleri üzerinde çaprazlanmış modül kavramını tanımlamıştır. Bu tez çalışmasında Lie cebirleri üzerinde tanımlanan çaprazlanmış modüller GAP vasıtası ile bilgisayar ortamına aktarılacaktır. (Cayley, 1854) çalışmasında grup cebir

kavramını tanımlamıştır. Grup cebir kavramı, GAP programının gruplar için oluşturulmuş zengin kütüphanesini cebirler için kullanılabilir hale getirmiştir.

Tez kapsamında ilk olarak GAP programlama dili tanıtlacaktır. Daha sonra Lie cebirleri ve grup cebirler hakkında bilgi verilldikten sonra bir GAP ortak paketi olan LAGUNA tüm özellikleriyle birlikte incelenecektir. 4. Bölümde LAGUNA paketinin de yardımıyla Lie cebirleri üzerindeki çaprazlanmış modülleri oluşturmak için tez kapsamında yazılan derlemeler tanıtılacaktır. Yazılan bu derlemeler (Odabaş, 2009) da yazılan derlemelerle birleştirilerek bir ortak paket oluşturulmuştur. GAP için tez çalışması süresince hazırlanan bu ortak pakete XMODALG ismi verilmiştir. GAP programı içerisinde bu ortak paketin çağırılması aşağıdaki komut ile yapılacaktır.

GAP

```
gap> RequirePackage("xmodalg");
                                 |||/
           .-._________          (o o)              _________.-.
       ----/ _)________)  +-oooO--(_)----------+  (________(_/ ----
         (    ()___)      |   XMod Of Algebra  |    (___()     )
              ()__)       |                    |     (__()
       ----___()_)        |     Ver. 1.000     |      (_()___/----
                          +------------Ooo-----+
true
```

BÖLÜM 1

GAP Programlama Dili

1.1 Giriş

GAP programı, Pascal ve C programlama dilleri ile yazılmış bir paket programdır. Pascal ve C kodları kullanılarak, sadece GAP programı içerisinde kullanılabilen ve yapı olarak Pascal a benzeyen yeni bir programla dili oluşturulmuştur. Diğer programlama dilleri gibi değişkenlere, sabitlere, denetim deyimlerine, aritmetik operatörlere ve döngü deyimlerine sahip olan bu programlama dilleri GAP programı içersinde kullanılabilir ve yorumlanabilir hale getirilmiştir.

Bu bölüm içerisinde GAP programında kullanılacak semboller, anahtar kelimeler, değişken isimleri, deyimler ve açıklamaların nasıl yapılacağı, değişken ve değişken atama yöntemleri, karşılaştırma operatörleri ($<, >, <=, >=, <>, =$), aritmetik operatörler, mantıksal operatörler çeşitli örneklerle detaylı bir şekilde sunulmuştur. Ayrıca bu bölüm içerisinde GAP programının `if`, `while`, `repeat`, `for`, `break`, `continue`, `print` denetim deyimleri de çeşitli basit örneklerle sunulmuştur. Programlama dili hakkında ayrıntılı bilgilere (GAP, 2008) den de bakılabilir. Burada bulunan derlemelerin bir bölümü (Odabaş, 2002) den alınmıştır.

1.2 Semboller

GAP programlama dili içerisinde kullanılan özel tanımlı karakter ve semboller vardır. Aşağıdaki semboller, aritmetik işlemler, karşılaştırmalar gibi birçok işlevi gerçekleştirebilmek için kullanılırlar.

GAP

```
+ -
* / ^~
= <> < <= >= ![
:= . .. -> , ; !{
[ ] { } ( ) :
```

1.3 Anahtar Kelimeler

GAP programlama dili içerisinde, kullanılacak fonksiyonlar için ayrılmış özel kelimeler vardır. Aşağıda listesi verilen anahtar kelimeler, değişken ismi olarak kullanılamazlar.

GAP
```
and     do       elif   else    end     fi
for     function if     in      local   mod
not     od       or     repeat  return  then
until   while    quit   QUIT    break   rec
continue
```

Kullanıcı bu anahtar kelimeleri değişken ismi olarak kullanırsa bir hata mesajı ile karşılaşacaktır.

GAP
```
gap> local:=2;
Syntax error: expression expected
local:=2;
  ^
```

GAP programı büyük harf küçük harf duyarlılığına sahiptir. Hemen hemen tüm anahtar kelimelerin küçük harfler kullanılarak yazıldığına dikkat edilmelidir. Örneğin `return` anahtar kelimesindeki harflerden herhangi birisi büyük harfle yazılırsa bu kelime anahtar kelime olmaktan çıkar. `Return` , `rEturn` , `reTurn` , `retUrn` , `retuRn` , `returN` kelimeleri anahtar kelime değildirler ve değişken ismi olarak kullanılabilirler. Anahtar kelimeler arasında boşluk kullanılmamalıdır. `el if` şeklindeki bir yazılım `elif` anahtar kelimesinin yaptığı işlevi yerine getirmeyecektir.

Yukarıdaki anahtar kelimelere ek olarak aşağıdaki kelimeler de GAP programlama dili tarafından anahtar kelime olarak ayrılmıştır ve değişken ismi olarak kullanılamazlar.

GAP
```
false true  IsBound  Unbind TryNextMethod
Info Assert fail     SaveWorkspace
```

1.4 Değişken İsimleri

Bir değişkene isim atamak için değişken isimleri kullanılır. Değişken isimleri tanımlanırken anahtar kelimelerden farklı olmak üzere büyük harfler, küçük harfler, rakamlar ve alt çizgi (_)

işareti kullanılabilir. GAP programında değişken isimleri tanımlanırken harfler kullanılacaksa mutlaka İngilizce karakterler kullanılmalıdır. GAP programında büyük küçük harf duyarlılığı olduğundan aynı kelimenin büyük harflerle veya küçük harflerle yazılması farklı değişken isimlerine karşılık gelecektir. Aşağıda verilen değişken isimleri farklı değişkenlere karşılık gelirler:

GAP

```
Birim1   birim1   biRim1   BIRIM1
Birim_1  birim_1  biRim_1\ BIRIM_1
1_Birim  _birim   _biRim   1_BIRIM
_1Birim  _1birim  _1biRim  _1BIRIM
```

Değişken isimleri tanımlanırken herhangi bir karakter uzunluğu sınırlaması olmamasına rağmen GAP programı için yalnızca ilk 1023 karakter anlamlıdır. GAP programı için ilk 1023 karaktere kadar aynı olan iki değişken isimleri 1023 ten sonraki karakterler farklı olsa dahi aynı değişkeni gösterecektir.

1.5 Deyimler ve Açıklama

GAP programında herhangi bir komutta sağ taraftaki kısım deyim olarak adlandırılır. Deyimler değer yapılarını ifade etmekte kullanılırlar. Genel olarak deyimler, bir değer olmayıp bir veya birkaç tane değerin aritmetik operatörlerle veya fonksiyonlarla bir işlem gerçekleştirmesini sağlayan yapılardır. Örneğin 3+5 bir deyim, bu deyimin sonucu olan 8 bir değerdir. GAP programında # işareti bir açıklama yapmak için kullanılır.

GAP

```
gap> 3+5; # bilinen toplama islemi
8
gap> 3*5; # bilinen carpma islemi
15
```

1.6 Değişkenler

Diğer programlama dillerinde olduğu gibi GAP programında da değişkenler, bir değere karşılık gelmesi için kullanılırlar. Her bir değişken ismine bir tanımlayıcı (identifier) denir. GAP programında değişken tanımlama işlemi PASCAL programlama dilindeki gibidir. Bir tanımlayıcı (identifier) genel olarak `tanımlayıcı_ismi:=değer;` biçiminde oluşturulur. Bir ifade tanımlayıcı ismi olarak tanımlanmamışsa bu değeri deyim olarak kullanmak hata mesajına

sebep olacaktır. Daha önce oluşturulmuş bir tanımlayıcı için yeni bir değer girildiğinde, tanımlayıcı artık yeni değeri kullanmaya başlayacaktır.

GAP

```
gap> x:=3;
3
gap> x;
3
gap> y;
Variable: 'y' must have a value
gap> x:=7;
7
gap> x;
7
```

GAP programında tanımlayıcı ismi olarak anahtar kelimeler kullanılamaz. Anahtar kelimelerin dışında GAP programı içerisinde kullanılmak istenilen tanımlayıcı isimleri, GAP programı kütüphanesinde bir fonksiyon olarak tanımlanmışsa bu isimler de kullanılamaz, GAP programı bu değerin salt okunur olduğunu belirtip hata verecektir.

GAP programında `IsReadOnlyGlobal()` komutu, değişken ismi olarak kullanmak istenilen karakterlerin salt okunur olup olmadığını yani GAP programı içerisinde bir fonksiyon ismi olarak kullanılıp kullanılmadığını test eder. `()` işaretleri arasına salt okunur olup olmadığı test edilmek istenen karakterler `\"` işaretleri arasında yazılır. GAP programı `true` veya `false` yanıtı verecektir. Bu fonksiyonun sonucu olarak `true` yanıtı verilirse, değişkenin salt okunur olduğu anlaşılacaktır. Böylece bu değişkenden farklı bir değişken kullanılması gerekir.

GAP

```
gap> IsReadOnlyGlobal("Group");
true
gap> Group:=2;
Variable: 'Group' is read only
not in any function
Entering break read-eval-print loop ...
you can 'quit;' to quit to outer loop, or
you can 'return;' after making it writable to continue
brk>
```

Yukarıdaki örnekte salt okunur olup olmadığı sorgulanan Group karakteri, grup cebirsel yapılarını oluşturmak için yazılmış bir fonksiyon olup, tanımlayıcı ismi olarak kullanılamaz.

GAP programında `MakeReadOnlyGlobal()` komutu tanımlayıcı ismini salt okunur hale getirir. Bir karakter değişken ismi olarak atandıktan sonra bu komut kullanılırsa, artık bu değişken ismine yeni bir değer atanamaz ve bu değer sürekli sabit kalır. Burada dikkat edilecek olan durum GAP programı kapatılıp yeniden açıldığında bu komutla salt okunur hale getirilen karakterler bu özelliklerini kaybederler. `()` işaretleri arasına, salt okunur hale getirilmek istenen karakterler `\"` işaretleri arasında yazılır. Salt okunur hale getirilmek istenilen karakterler zaten salt okunur özellikte ise GAP programı hata verecektir.

GAP

```
gap> pi:=22/7;
22/7
gap> MakeReadOnlyGlobal("pi");
gap> pi:=4;
Variable: 'pi' is read only
not in any function
Entering break read-eval-print loop ...
you can 'quit;' to quit to outer loop, or
you can 'return;' after making it writable to continue
brk> quit;
gap> pi;
22/7
```

GAP programında fonksiyon olarak kullanılan karakterler veya `MakeReadOnlyGlobal()` komutu kullanılarak salt okunur hale getirilen karakterler, tanımlayıcı ismi olarak kullanılmak istenirse `MakeReadWriteGlobal()` komutu ile `()` işaretleri arasına `\"` işaretleri arasında yazılan karakterler, yazılıp okunabilir hale getirilir.Yazılabilir hale getirilmek istenen karakterler zaten yazılabilir özellikte ise GAP programı hata verecektir.

GAP

```
gap> IsReadOnlyGlobal("pi");
true
gap> MakeReadWriteGlobal("pi");
gap> pi:=4;
4
gap> IsReadOnlyGlobal("pi");
false
gap> MakeReadWriteGlobal("Group");
gap> Group:=2;
2
```

GAP programında bir tanımlayıcı isminin hangi değere karşılık geldiğini öğrenmek için `tanımlayıcı_ismi;` komutunu kullanmak yeterlidir. `ValueGlobal()` komutu da yine bir tanımlayıcı isminin karşılık geldiği değeri öğrenmek için kullanılabilir.

GAP
```
gap> pi;
4
gap> ValueGlobal("pi");
4
```

GAP programında herhangi bir değişkenin bir değer tutup tutmadığı `IsBoundGlobal()` komutu kullanılarak test edilebilir.

GAP
```
gap> IsBoundGlobal("pi");
true
gap> IsBoundGlobal("k");
false
```

GAP programında `UnbindGlobal()` komutu daha önce tanımlayıcı olarak kullanılan bir değeri, tanımlayıcı olmaktan çıkarıp tekrar boş karakter haline dönüştürür. `()` işaretleri içine `\"` işaretleri arasında tekrar boş hale getirilmek istenilen karakterler yazılır.

GAP
```
gap> UnbindGlobal(``pi'');
gap> pi;
Variable: 'a' must have a value
```

GAP programında bir tanımlayıcı isminin genel olarak `tanımlayıcı_ismi:=değer;` biçiminde oluşturulduğundan bahsedilmişti. `BindGlobal()` komutu da bir tanımlayıcı ismi oluşturmak için kullanılabilir. `()` işaretleri arasına , `\"` işaretleri arasında tanımlayıcı ismini oluşturacak karakter ve , işareti kullanılarak bu karakterin tutacağı değer yazılmalıdır. Bu komut kullanılarak oluşturulan tanımlayıcı isimleri otomatik olarak salt okunur hale gelecektir. Tüm çalışma boyunca sabit kalması istenilen değerler için bu komut kullanılabilir.

GAP
```
gap> Birim:=0;
0
gap> BindGlobal("Birim",1);
#W BIND_GLOBAL: variable 'D' already has a value
gap> Birim;
1
gap> IsReadOnlyGlobal("Birim");
true
```

`NamesSystemGVars()` komutu, GAP programı çalıştırılmaya başlandığında GAP programı kütüphanesi tarafından oluşturulan tüm değişken isimlerini listelemek için kullanılır.

`NamesUserGVars()` komutu, GAP programı çalıştırılmaya başlandığında, kullanıcı tarafından oluşturulan tüm değişken isimlerini listelemek için kullanılır .

GAP

```
gap> NamesUserGVars();
[ "A", "Birim", "KK", "x" ]
```

1.7 Program Denetim Deyimleri

1.7.1 If

Bu komut bir koşula bağlı olan işlemleri gerçekleştirmek için kullanılır. `If` denetim deyiminin genel kullanım şekli aşağıdaki gibidir .

GAP

```
if kosul1 then
deyim1;
elif kosul2 then
deyim2;
else
deyim3;
fi;
```

GAP programı koşullara bağlı işlemleri gerçekleştirmeye en üstte yer alan `if` denetim deyiminden başlar. Buradaki `kosul1` olumlu ise `deyim1` de yer alan ifadeler gerçekleştirilir ve program akışı `fi` komutundan itibaren devam eder. Aksi halde, yani `kosul1` olumlu değil ise, program `elif` komutunun bulunduğu satıra gelir ve `kosul2` bölümünde yer alan kıyaslamanın doğruluğunu test eder. `kosul2` olumlu ise `deyim2` de yer alan ifadeler gerçekleştirilir ve program akışı `fi` komutundan itibaren devam eder. `if` ve `elif` komutlarındaki `kosul1` ve `kosul2` bölümlerinde yer alan kıyaslamalar doğru değil ise, program else komutunun bulunduğu satıra gelir ve `deyim3` de yer alan ifadeler gerçekleştirilir. `if` deyimi kullanılırken `elif` ve `else` komutlarını kullanmak zorunluluğu yoktur. `elif` komutu bir `if` deyimi içersinde istenilen çoklukta kullanılabilir.

GAP

```
gap> x:=-13;
-13
```

```
gap> if 0<x then
> sgnx:=1;
> elif x<0 then
> sgnx:=-1;
> else
> sgnx:=0;
> fi;
gap> sgnx;
-1
```

1.7.2 While

Belirli bir koşulla birlikte bir döngü oluşturmak için kullanılan bir deyimdir. `while` denetim deyiminin genel kullanım şekli aşağıdaki gibidir .

GAP

```
while kosul1 do
deyim1;
od;
```

Koşulların gerçekleştirilmesine, en üstte yer alan `while` deyiminden başlanır. Buradaki `kosul1` olumlu ise `deyim1` de yer alan ifadeler gerçekleştirilir. Daha sonra program `od;` komutuna geldiğinde tekrar `while` deyiminin olduğu satıra gelir ve `kosul1` in doğruluğunu test eder. `kosul1` deki kıyaslama yanlış oluncaya kadar bu döngü devam eder. `kosul1` olumsuz olduğunda program `od;` satırından sonraki ilk satırdan işlemlerine devam eder.

GAP

```
gap> para:=10;
10
gap> yil:=0;
0
gap> while para<50 do
> para:=para+(para*(30/100));
> yil:=yil+1;
> od;
gap> yil;
7
gap> para;
62748517/1000000
```

Yukarıdaki program, 10 liranın yıllık %30, faizle 50 lirayı geçtiği yada eşit olduğu ilk yılı ve paranın kaç lira olduğunu hesaplamak için kullanılabilir. `para<50` koşulu sağlanıncaya

kadar `while` deyimi tekrarlanmaya devam edecektir. `para>=50` değeri sağlandıktan sonra koşul sağlanamayacak ve döngüden çıkılacaktır. GAP programında `while` deyimi kullanılırken dikkat edilmesi gereken önemli bir konu, koşulda sınanan değerin döngü içersinde değiştirilme gerektiğidir. Aksi halde sonsuz döngüye girilir ki bu GAP programının yanıt veremez hale gelmesi demektir. Yukarıdaki örnekte `while` içerisinde `para` değişkeninin değeri değiştirilmeseydi `para<50` koşulu sürekli olarak sağlanacağından program sonsuz döngüye girerdi.

1.7.3 Repeat

Belirli bir koşulla birlikte bir döngü oluşturmak için kullanılan başka bir deyimdir. `repeat` denetim deyiminin genel kullanım şekli aşağıdaki gibidir .

GAP

```
repeat
deyim1;
until kosul1;
```

`repeat` denetim deyiminde hiçbir koşula bakılmaksızın `deyim1` de yer alan ifadeler gerçekleştirilir. Daha sonra `until` komutunun bulunduğu satırda yer alan `kosul1` test edilir. Buradaki `kosul1` sağlanıncaya kadar program `repeat` satırının olduğu yere gidecek ve `deyim1` de yer alan ifadeler tekrar gerçekleştirilecektir. `kosul1` sağlandığında program `until` komutunun bulunduğu satırdan bir sonraki satıra geçerek işlemlerine devam edecektir.

`repeat` denetim deyimi ile `while` denetim deyimi yapı ve kullanım şekli olarak birbirine çok benzemektedir. `repeat` ve `while` arasındaki belirgin fark, `while` deyiminde işlemler yapılmadan önce koşulun sağlanması gerektiği, `repeat` deyiminde ise koşula bakılmaksızın işlemlerin gerçekleştirip daha sonra koşula bakılmasıdır. `repeat` ve `while` deyimleri arasındaki önemli bir diğer fark ise, `while` ile oluşturulan döngünden koşul olumsuz olduğunda, `repeat` ile oluşturulan döngüden ise koşul sağlandığında çıkılır.

GAP

```
gap> para:=10;
10
gap> yil:=0;
0
gap> repeat
> para:=para+(para*(30/100));
> yil:=yil+1;
```

```
> until para>=50;
gap> yil;
7
gap> para;
62748517/1000000
```

Yukarıdaki program, `repeat` deyimi ile 10 liranın yıllık %30 faizle 50 lirayı geçtiği yada eşit olduğu ilk yılı ve paranın kaç lira olduğunu hesaplamak için kullanılabilir. `para>=50` koşulu sağlanıncaya kadar `repeat` deyimi tekrarlanmaya devam eder. `para>=50` değeri sağlandıktan sonra koşul sağlanacak ve döngüden çıkılacaktır. `while` deyiminde olduğu gibi `repeat` deyimi kullanılırken de koşul içersindeki değer döngü içersinde değiştirilmez ise program sonsuz döngüye girebilir. Yukarıdaki örnekte `repeat` içerisinde `para` değişkeninin değeri değiştirilmeseydi `para>=50` koşulu hiçbir zaman sağlanamayacak ve program sonsuz döngüye girecekti.

GAP

```
gap> x:=3;
3
gap> y:=5;
5
gap> repeat
> y:=y+1;
> x:=y^2;
> until x>0;
gap> x;
36
gap> y;
6
```

Yukarıdaki örnekte, `x>0` durumu sağlandığı halde, döngü bir kez çalışmış `x` ve `y` değişkenlerinin karşılık geldiği değerler değişmiştir.

1.7.4 For

Belirli bir koşulla birlikte bir döngü oluşturmak için kullanılan başka bir deyimdir. `for` denetim deyiminin en önemli özelliği, bir döngü sayacına sahip olmasıdır. `for` denetim deyiminin genel kullanım şekli aşağıdaki gibidir .

GAP

```
for i in [1..n] do
deyim1;
> od;
```

`for` denetim deyiminde hiçbir koşula bakılmaksızın `deyim1` de yer alan ifadeler gerçekleştirilir. Burada verilen `[1..n]` gösterimi, ilk elemanı `1` olan ve birer artışla son elamanı `n` ye ulaşan bir listeyi ifade etmektedir. `[..]` ifadesi, ilk elemandan son elemana doğru birer birer artışı göstermektedir. `i` değişkenin ilk değeri verilen listenin ilk elemanıdır. Program `deyim1` de yer alan ifadeleri gerçekleştirip `od;` komutunun bulunduğu satıra geldiğinde tekrar `for` deyiminin olduğu satıra gelir ve `i` değeri verilen listenin ikinci elemanın değerini alır. Döngü, verilen liste elemanlarının sayısınca çalışacaktır. Döngü, son kez çalıştığında yani `i` değeri, verilen listenin son elemanın değerini alarak `od;` komutunun bulunduğu satıra geldiğinde herhangi bir koşul sınaması yapmadan bu satırdan sonraki satırdan işleme devam edecektir.

GAP

```
gap> faktoriyel:=1;
1
gap> for i in [1..6] do
> faktoriyel:=faktoriyel*i;
> od;
gap> faktoriyel;
720
```

Yukarıdaki örnekte `for` deyimi ile oluşturulan döngü altı kez çalışacaktır. Örnekte `i` değeri `[1..6]` listesinin tüm elemanlarının değerini bir kez alacaktır. (Döngü 1. kez dönerken $i = 1$, 2. kez dönerken $i = 2$, 6. kez dönerken $i = 6$ gibi.) faktoriyel değişkeni ile birlikte döngü altı kez çalıştıktan sonra $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$ değeri hesaplanacaktır.

GAP programında kullanılacak listelere bir tanımlayıcı ismi verilebilmektedir. GAP programında liste kullanmak konusunda sonraki bölümlerde ayrıntılı bilgi verilecektir. Burada `for` denetim deyimi açıklanırken kullanılacak bir listenin uzunluğunun `Length(liste)` komutu ile belirlendiğini belirtmek gerekir. `for` denetim deyimi ile birlikte tanımlayıcı ismi verilen listeler de kullanabilir.

GAP

```
gap> liste:=[1,3,5,7,9,15];
[ 1, 3, 5, 7, 9, 15 ]
gap> for k in liste do
> Print(k, " ");
> od; Print("\n");
1 3 5 7 9 15
```

Yukarıdaki örnekte `liste` tanımlayıcı ismi ile oluşturulan liste elemanları sırası ile `k` değerini almışlardır. Sonraki bölümde ayrıntılı olarak bahsedilecek olan `Print()` komutunun şu an için bir tanımlayıcı isminin tuttuğu değeri ekrana yazdırdığını bilmek yeterlidir.

`for` denetim deyimi, listeler dışında kümeler veya gruplarla birlikte de kullanılabilir. GAP programında grup cebirsel yapılarından 9. bölümde bahsedilecektir. `Order()` komutu grubun bir elemanının mertebesini bulmak için kullanılmıştır.

GAP

```
G:=Group((1,2,3,4,5),(6,7,8,9,10),(1,2));
Group([ (1,2,3,4,5), (6,7,8,9,10), (1,2) ])
gap> eleman:=0;
0
gap> elmertop:=0;
0
gap> for i in G do
> eleman:=eleman+1;
> elmertop:=elmertop+Order(i);
> od;
gap> eleman;
600
gap> elmertop;
7971
```

Yukarıdaki örnekte bir G grubu oluşturulmuştur. Bu G grubunun `for` denetim deyimi içerisinde kullanılması, `i` değişkeninin grubun birinci elemanından başlayarak son elemana kadar değerler alması anlamına gelir. Döngü sona erdiğinde `eleman` değişkeni G grubunun eleman sayısını (yani grubunun mertebesini) ve `elmertop` değişkeni, G grubunun elemanlarının mertebelerinin toplamını bulmuş olacaktır.

`for` denetim deyiminin hiçbir koşula bakılmaksızın liste elemanlarının sayısınca çalıştırıldığından bahsedilmişti. GAP programı kullanılırken `i` değerinin birer birer artması yerine farklı adımlarla artması veya azalması istenilen programlar yazılması gerekebilir. `while` denetim deyimi ihtiyaç halinde `for` denetim deyimine benzer şekilde kullanılabilir.

GAP

```
dongu_listesi := liste;
i:= 1;
while i<= Length(dongu_listesi) do
deyim1;
i:=i+1;
od;
```

GAP programı kullanılırken sonsuz döngüye düşmemek için `i` değişkeninin döngü içerisinde değiştirilmesi gerektiği unutulmamalıdır.

GAP
```
gap> Liste:=[1,2,3,4,5,6,7,8,9,10,12,14,15];
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15 ]
gap> i:=2;
2
gap> while i<=Length(Liste) do
> Print(i," ");
> i:=i+3;
> od; Print("\n");
2 5 8 11
```

Program denetim deyimleri, birbirleri içerisinde kullanılarak iç içe döngüler oluşturulabilir.

1.7.5 Break

Denetim deyimi ile oluşturulmuş bir döngüden çıkmak için kullanılan deyimdir. Döngü içersinde `break` deyimi ile karşılaşıldığı zaman hiçbir koşul dikkate alınmadan döngü sonlanır ve program akışı döngüden sonraki ilk satırdan itibaren devam eder. İç içe döngüler içersinde `break` deyimi kullanıldığında ise en içteki döngüden çıkılır .

GAP
```
gap> G:=Group((1,2,3,4,5),(8,9));
Group([ (1,2,3,4,5), (8,9) ])
gap> for i in G do
> if Order(i)=2 then
> break;
> fi;
> od;
gap> i;
(8,9)
gap> Order(i);
2
gap>
```

Yukarıdaki örnekte bir G grubu oluşturulmuş ve `if` deyimi ile G grubunun elemanlarının mertebelerinin 2 ye eşit olup olmadığı koşulu incelenmiştir. G grubunda, mertebesi 2 ye eşit olan ilk elemana sıra geldiğinde `break` deyimi işlenecek ve döngüden çıkılacaktır. Döngüden çıkıldığı anda `i` değişkeni, mertebesi 2 olan ilk elemanı tutuyor olacaktır.

Genellikle `if` deyimi ile birlikte kullanılan `break` deyiminin, mutlaka bir döngü içerisinde kullanılması gerekir.

1.7.6 Continue

Denetim deyimi ile oluşturulmuş bir döngüden, içerisinde belli özellikteki çevrim işlemleri gerçekleştirmeden sonraki çevrime geçmek için kullanılan deyimdir. Döngü içersinde `continue` deyimi ile karşılaşıldığı zaman ondan sonra gelen deyimler atlanır ve döngü bir sonraki çevrime girer. Bir bakıma `break` deyiminin tersi gibi düşünülebilir. `break` döngüyü sonlandırma görevini `continue` ise bir sonraki çevrime geçirme görevini yürütür .

GAP

```
gap> G:=Group((1,3,2),(1,3));
Group([ (1,3,2), (1,3) ])
gap> for k in G do
> if Order(k)=4 then
> continue;
> fi;
> Print(k,"\n");
> od;
()
(1,2,3)
(1,3,2)
(2,3)
(1,2)
(1,3)
gap> Order(());
1
gap> Order((1,2,3));
3
gap> Order((1,3,2));
3
gap> Order((2,3));
2
gap> Order((1,2));
2
gap> Order((1,3));
2
```

Yukarıdaki örnekte bir G grubu oluşturulmuş ve `if` deyimi ile G grubunun elemanlarının mertebelerinin 4 e eşit olup olmadığı koşulu incelenmiştir. G grubunda mertebesi 4 e eşit olan ilk elemana sıra geldiğinde `continue` deyimi işlenecek ve döngü içerisinde yer alan `Print()` komutu, G nin mertebesi 4 olan elemanı için işletilmeyecek ve döngü bir sonraki eleman için

devam edecektir. Döngüden çıkıldığında grubunun mertebeleri 4 den farklı olan elemanları yazdırılmış olacaktır.

`continue` deyimi de `break` deyiminde olduğu gibi genellikle `if` deyimi ile birlikte ve mutlaka bir döngü içerisinde kullanılmalıdır.

1.7.7 Print

Bir veya daha fazla tanımlayıcı isminin tuttuğu değeri ekrana yazdırmak için kullanılır. `Print` komutu genel olarak döngüler içerisinde kullanılır. Birden fazla tanımlayıcı ismi tek bir komut içerisinde virgülle birbirinden ayrılarak kullanılabilir .

GAP

```
gap> liste1:=[1,2,3];
 [ 1, 2, 3 ]
gap> Print(liste1);
[ 1, 2, 3 ]gap> liste2:=[1,2,3,4];
[ 1, 2, 3, 4 ]
gap> Print(liste2);
[ 1, 2, 3, 4 ]gap>
```

Yukarıdaki örnekte görüldüğü gibi `Print` komutu yalın olarak kullanıldığında `gap>` ifadesi bir alt satırdan değil de yazdırılan tanımlayıcı ismi değerinden sonra gelecektir. GAP programında bu gibi durumlar için kullanılabilecek özel parametreler vardır.

GAP

```
\n
```

Bu karakter GAP programında, `\"` işaretleri arasında, imlecin bir satır aşağıya atılması için kullanılır. Bu karakter bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır.

GAP

```
 gap> Print(liste1,liste2);
[ 1, 2, 3 ][ 1, 2, 3, 4 ]gap> Print(liste1,"\n",liste2,"\n");
[ 1, 2, 3 ]
[ 1, 2, 3, 4 ]
```

GAP

```
\"
```

Bu karakter GAP programında, \" işaretleri arasında, ekrana " işaretinin çıkarılması için kullanılır. Bu karakter bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır.

GAP

```
gap> Print("\"",liste1,"\"",liste2,"\"","\n");
"[ 1, 2, 3 ]"[ 1, 2, 3, 4 ]"
```

GAP

```
\'
```

Bu karakter GAP programında, \" işaretleri arasında, ekrana ′ işaretinin çıkarılmasını için kullanılır. Bu karakter bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır.

GAP

```
gap> Print("\'",liste1,"\'",liste2,"\'","\n");
'[ 1, 2, 3 ]'[ 1, 2, 3, 4 ]'
```

GAP

```
\\
```

Bu karakter GAP programında, \" işaretleri arasında, ekrana \ işaretinin çıkarılması için kullanılır. Bu karakter bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır.

GAP

```
gap> Print("\\",liste1,"\\",liste2,"\\","\n");
\[ 1, 2, 3 ]\[ 1, 2, 3, 4 ]\
```

GAP

```
\b
```

Bu karakter GAP programında, \" işaretleri arasında, ekrana yazdırılan ifadeye backspace (geriye boşluk) işlemini gerçekleştirmek için kullanılır. Bu karakter bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır. Bu karakter kullanıldığı yerde kendisinden önceki ilk karakteri siler.

GAP

```
gap> Print(liste1,"\b",liste2,"\n");
[ 1, 2, 3 [ 1, 2, 3, 4 ]
```

GAP programında `Print` komutunu kullanarak tanımlayıcı isminin, tuttuğu değerden başka yazılmak istenilen herhangi bir açıklama da ekrana yazdırılabilir. Açıklama GAP programında, `\"` işaretleri arasında kullanılır. Açıklamalar bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır. Tanımlayıcı isimlerinin tuttuğu değerler arasına boşluk bırakmak için de içerisi boş `\"` işaretleri kullanılabilir.

GAP

```
gap> Print(liste1,"\n Yukarida liste1 degerini yazdirdik\n");
[ 1, 2, 3 ]
 Yukarida liste1 degerini yazdirdik
gap> Print("   ",liste1,"   ","\n Yukarida liste1 degerini yazdirdik\n");
   [ 1, 2, 3 ]
   Yukarida liste1 degerini yazdirdik
```

GAP

```
\t
```

Bu karakter GAP programında, `\"` işaretleri arasında, kendisinden önce ve sonra gelen karakterler arasında bir tab (sekiz kez boşluk tuşuna basmak) boşluk bırakmak için kullanılır. Bu karakter bir tanımlayıcı ismi ile birlikte kullanılıyorsa araya virgül koyulmalıdır.

GAP

```
gap> Print("\t",liste1,"\n","        ",liste1,"\n");
  [ 1, 2, 3 ]
  [ 1, 2, 3 ]
```

GAP programında `Print` komutu genel olarak döngü içerisinde kullanılır. Bu komut içerisinde ki değişkenlere temel aritmetik işlemler de yaptırılabilir.

GAP

```
gap> liste:=[1,2,3,4,5,6,7];
[ 1, 2, 3, 4, 5, 6, 7 ]
gap> for i in liste do
> Print(i," ",2*i," ",i^2,"\n");
> od;
1 2 1
2 4 4
3 6 9
4 8 16
5 10 25
6 12 36
7 14 49
```

1.8 GAP Programında Fonksiyon Kullanımı

GAP programında özel fonksiyonlar da oluşturulabilir. Fonksiyonlar oluşturulurken üç farklı yöntem kullanılabilir. Birinci yöntemde fonksiyon oluşturabilmek için $->$ simgesi kullanılır. Örnek olarak, yazılan sayıların karesini hesaplayan bir fonksiyon ;

GAP

```
gap> kuvvet4:=x -> x^2;
function( x ) ... end
gap>
```

biçiminde yazılabilir. Yukarıdaki komutlar yardımıyla yazılan fonksiyon aşağıdaki gibi kullanılır.

GAP

```
gap> kuvvet4(2);
4
gap> kuvvet4(4);
16
```

GAP programında fonksiyon oluşturmak için kullanılan ikinci yöntem ise `function` komutudur .

GAP

```
gap> fonk:=function(x)
> local y;
> y:=x^2+2*x+1;
> return y;
> end;
function( x ) ... end
```

Yukarıdaki kodlar $y = f(x) = x^2 + 2x + 1$ şeklindeki bir fonksiyonu hesaplamak için yazılmış kodlardır. Buradaki `function(x)` komutu yazılacak fonksiyonun temel parametresinin x olacağını göstermektedir. Fonksiyon tanımlanırken kullanılacak diğer değişkenler yani yerel değişkenler de `local` komutundan sonra belirtilmelidir. `return` komutu ise fonksiyon içerisinde tanımlanan işlemler gerçekleştirildikten sonra geriye bir değer gönderilmesini sağlar.

GAP

```
gap> fonk(10);
121
```

```
gap> fonk(1);
4
gap> fonk(2);
9
```

GAP programı kullanırken bir kez yazılmış olan bir fonksiyon, daha sonra tekrar kullanılabilmektedir. Bu işlemi gerçekleştirmek için yazılan fonksiyonu bir şekilde kaydetmek gerekir. `LogTo` komutunda GAP programı kullanırken yapılan işlemlerin bir kaydının tutulduğundan daha önce bahsedilmişti, ancak bu komut yazılan fonksiyonu tekrar çağırmak için yeterli değildir. `InputLogTo` komutu oluşturulan fonksiyonların tekrar yüklenmesine olanak sağlar.

GAP

```
gap> InputLogTo("fonk");
gap> fonk:=function(x)
> local y;
> y:=x^2+2*x+1;
> return y;
> end;
function( x ) ... end
gap> InputLogTo();
```

Yukarıdaki `InputLogTo("fonk");` komutu kendisinden sonraki komutların "fonk" adlı bir dosya içerisinde saklanılması sağlar. Bu dosya GAP programının kurulduğu dizinin içerisine oluşturulacaktır. GAP programından çıkıp oluşturulan "fonk" adlı dosya bir metin editörü ile açılırsa bu dosyanın içeriği aşağıdaki gibi olacaktır.

GAP

```
fonk:=function(x)
local y;
y:=x^2+2*x+1;
return y;
end;
InputLogTo();
```

Bu dosyada `InputLogTo();` satırı silinir ve dosya tekrar kaydedilirse daha sonra bu yazılan fonksiyon ne zaman kullanılmak istenirse `Read("DosyaAdı");` komutu kullanılarak dosya GAP programına okutulabilir .

GAP

```
gap> Read("fonk");
gap> fonk(1);
4
gap> fonk(3);
16
gap> fonk(1234567);
1524158146623
```

GAP programında fonksiyon oluşturmanın üçüncü ve en kullanışlı yöntemi ise direkt olarak metin editörleri kullanarak `gi` dosyaları oluşturmaktır. `gi` dosyaları, içerisinde birden fazla fonksiyon barındırabilen ve doğrudan GAP programı tarafından okunabilen dosyalardır. `gi` dosyaları oluşturulduktan sonra GAP programı `Read("fonk.gi")` şeklindeki bir komutla karşılaştığında `fonk.gi` dosyasının içerisinde yer alan her fonksiyonun kullanılabilir duruma geldiğini anlar.

Aşağıda kodları verilen fonksiyon bir metin editörü içine yazılarak `kuvvet.gi` adıyla kaydedilebilir. Fonksiyon verilen bir x sayısı için x^2+2x+1 değerini hesaplayacaktır.

GAP

```
kuv:=function(x)
local y;
y:=x^2+2*x+1;
return y;
end;
```

Şimdi, var olan `kuvvet.gi` dosyasını `Read()` komutu kullanarak GAP programına okutup daha sonra bu dosya içerisindeki fonksiyon kullanılabilir.

GAP

```
gap> Read("kuvvet.gi");
gap> kuv(2);
9
gap> kuv(10);
121
```

`Read()` komutu bir dosyanın GAP programında okutulup çalışmaya hazır hale getirilmesi için kullanılır. `()` işaretleri arasına dosyanın adı ve varsa uzantısı yazılmalıdır. `ReadAsFunction()` komutu da yine GAP programına bir dosya okutmak için kullanılabilir. `ReadAsFunction()` komutunun, kullanılacak olan dosyadaki fonksiyon için bir fonksiyon adı

verilmesine, `function` ve `end` komutlarının bulunmasına ihtiyacı yoktur. Bunun anlamı metin içersindeki ifadeler `local` satırı ile başlar.

Aşağıda verilen program kodları bir metin editörü içinde oluşturulabilir ve bu dosya `carp.gi` adıyla kaydedilebilir.

GAP

```
local y;
y:=12;
return y*100;
```

Oluşturulmuş olan `carp.gi` dosyası `ReadAsFunction` komutu kullanılarak GAP programına okutulabilir ve aynı anda fonksiyon olarak kullanılabilir.

GAP

```
gap> ReadAsFunction("carp.gi")();
1200
```

Aşağıda kodları verilen fonksiyon bir metin editörü içinde oluşturulabilir ve bu dosya `tekcift.gi` adıyla kaydedilebilir. Yazılan fonksiyon bir sayının tek mi yoksa çift mi olduğunu inceleyecektir.

GAP

```
Tekcift:=function(x)
local k;
k:=(-1)^x;
if x=0 then
Print(x," sayisi tek veya cift ozellige sahip degildir. \n");
elif k=1 then
Print(x," sayisi cift sayidir. \n");
else
Print(x," sayisi tek sayidir. \n");
fi;
end;
```

Oluşturulmuş olan `tekcift.gi` dosyası `Read()` komutu kullanılarak GAP programına okutulabilir ve daha sonra bu dosya içerisindeki `Tekcift` isimli fonksiyon kullanılabilir.

GAP

```
gap> Read("tekcift.gi");
gap> Tekcift(46);
46 sayisi cift sayidir.
```

```
gap> Tekcift(23);
23 sayisi tek sayidir.
gap> Tekcift(0);
0 sayisi tek veya cift ozellige sahip degildir.
```

GAP programı kullanarak yarıçapı verilen bir dairenin alanını hesaplayan bir fonksiyon oluşturulabilir. Metin editörü kullanarak `daire.gi` adı altında bir dosya oluşturulabilir. Yazdılan fonksiyon πr^2 değerini hesaplayıp ekrana yazdıracaktır.

GAP

```
daire:=function(r)
local pi,alan;
pi:=22/7;
alan:=pi*r^2;
return alan;
end;
```

Yukarıdaki kodlar yazılarak oluşturulan `daire.gi` dosya GAP programına okutulabilir ve içerisinde yer alan `daire` isimli fonksiyonu kullanılarak herhangi bir r değeri için daire alanı hesaplanabilir.

GAP

```
gap> Read("daire.gi");
gap> daire(2);
88/7
gap> daire(112);
39424
```

GAP programı kullanarak yarıçapı verilen bir silindirin alanını hesaplayan fonksiyon yazılabilir. Metin editörü kullanarak `silindir.gi` adı altında bir dosya oluşturulabilir. Yazılan fonksiyon $2\pi r(r+h)$ değerini hesaplayıp ekrana yazdıracaktır.

GAP

```
AlanSilindir := function(arg)
local narg, alan, dikkat, hata, pi, r, h;
narg:= Length(arg);
dikkat:= " Dikkat: yar\i cap ve yukseklik olmak uzere iki deger girilmelidir. \n";
hata:= " Hata: Yar\i cap ve yukseklik degerleri pozitif olmal\i d\i r. \n";
if ((narg < 2) or (narg >= 3)) then
Print(dikkat);
return false;
fi;
```

```
r:=arg[1];
h:=arg[2];
pi:=22/7;
alan:=(2*pi*r)*(r+h);
if (arg[1]<=0) or (arg[2]<=0) then
Print(hata);
return false;
fi;
return alan;
end;
```

Fonksiyon oluşturma esnasında kullanılan `narg` değişkeni, fonksiyon içerisinde listesi verilen elemanların uzunluğunu bulur. Oluşturulan fonksiyon için elemanların uzunluğu iki olmalıdır çünkü bir silindirin alanını bulmak için yarıçap ve yükseklik değerlerine ihtiyaç vardır. GAP programı kullanıcısı `AlanSilindir` fonksiyonunu kullanırken ikiden fazla veya eksik değer girerse yazılan fonksiyon uygun bir hata mesajı verecektir. Bir silindirin alanı hesaplanırken yarıçap ve yükseklik değerleri sıfır veya sıfırdan küçük bir sayı olamayacağından fonksiyon oluşturulurken girilen değerlerin pozitif sayı olup olmadığını denetleyen ve uygun mesajla ekrana yazdıran bir komut da hata değişkeni ile eklenebilir.

Yukarıdaki kodlar yazılarak oluşturulan `silindir.gi` adlı dosya GAP programına okutulabilir ve içerisinde yer alan `AlanSilindir` isimli fonksiyon kullanılarak herhangi bir r ve h değerleri için silindirin alanı hesaplanabilir.

GAP

```
gap> Read("silindir.gi");
gap> AlanSilindir(13,40);
30316/7
gap> AlanSilindir(7,21);
1232
gap> AlanSilindir(3,6,6);
 Dikkat: yar\i cap ve yukseklik olmak uzere iki deger girilmelidir.
false
gap> AlanSilindir(3,-2);
 Hata: Yar\i cap ve yukseklik degerleri pozitif olmal\i d\i r.
false
```

Bir sayının pozitif tam bölenlerinin toplamı, sayının iki katı oluyorsa bu sayıya mükemmel sayı denilmektedir. Örneğin 6 sayısı mükemmel bir sayıdır çünkü $1+2+3+6=12=2\times 6$ dır. GAP programı kullanarak verilen bir sayının mükemmel bir sayı olup olmadığını hesaplayan bir fonksiyon oluşturulabilir. Metin editörü kullanılarak `mukemmel.gi` adı altında bir dosya oluşturulabilir.

GAP

```
Mukemmel:=function(n)
local mu,i,top,kat;
top:=0;
kat:=2*n;
for i in [1..n] do
mu:=(n mod i);
if (mu=0) then
top:=top+i;
fi;
od;
if (kat=top) then
Print(" ",n ," sayisi mukemmel sayidir.\n");
else
Print(" ",n ," sayisi mukemmel sayi degildir.\n");
fi;
end;
```

Yukarıdaki kodlar yazılarak oluşturulan `mukemmel.gi` dosyası GAP programına okutulabilir ve içerisinde yer alan `Mukemmel` isimli fonksiyon kullanılarak herhangi bir n sayısının mükemmel sayı olup olmadığı incelenebilir.

GAP

```
gap> Read("mukemmel.gi");
gap> Mukemmel(6);
 6 sayisi mukemmel sayidir.
gap> Mukemmel(12);
 12 sayisi mukemmel sayi degildir.
gap> Mukemmel(10001224);
 10001224 sayisi mukemmel sayi degildir.
```

Birden büyük olmak üzere kendisinden ve birden başka tam böleni olmayan sayılara asal sayılar denir. GAP programı kullanılarak verilen bir n sayısından küçük olan kaç tane asal sayı olduğunu hesaplayan bir fonksiyon oluşturulabilir. (Örnek olarak 10 sayısından küçük 4 tane asal sayı olup bunlar 2,3,5 ve 7 dir) Metin editörü kullanarak `kacasal.gi` adı altında bir dosya oluşturulabilir.

GAP

```
Kacasal:=function(n)
local i,j,kactane,asalmi;
kactane:=0;
i:=n-1;
while i>1 do
asalmi:=1;
j:=2;
```

```
while j<i do
if (i mod j=0) then
asalmi:=0;
fi;
j:=j+1;
od;
if (asalmi=1) then
kactane:=kactane+1;
fi;
i:=i-1;
od;
return kactane;
end;
```

Yukarıdaki kodlar yazılarak oluşturulan `kacasal.gi` dosyası GAP programına okutulabilir ve içerisinde yer alan `Kacasal` isimli fonksiyon kullanılarak herhangi bir n sayısından küçük kaç tane asal sayı olduğu hesaplanabilir.

GAP

```
gap> Read("kacasal.gi");
gap> Kacasal(2);
0
gap> Kacasal(3);
1
gap> Kacasal(10);
4
gap> Kacasal(100);
```

BÖLÜM 2

GRUP CEBİRLERİ VE LİE CEBİRLERİ

2.1 Giriş

GAP programı genel anlamda cebirlerin bilgisayar ortamında incelenmesine yardımcı olmakla beraber gruplar için tasarlanmış olduğundan cebirler için verilen destek henüz yeterli seviyelere ulaşamamıştır. Bu bakımdam Lie cebir yapısını bilgisayar ortamında geniş bir biçimde inceleyebilmek için grupların bu özelliğinden faydalanmalıyız. Herhangi bir grup ve halka yardımıyla elde edilen grup cebir (genel olarak grup halka) kavramı bu amacımız için kullanışlı bir araçtır. Bu şekilde GAP ın gruplar üzerindeki zengin veritabanını Lie cebirler içinde daha incelenebilir hale getirmeyi hedefliyoruz. Grup cebir yapısı herhangi bir gruptan cebir elde etmenin yanı sıra özellikle temsil teorisi için kullanışlı bir araçtır. Bu bölümde cebir yapısı, grup cebirleri ve Lie cebirleri genel özellikleriyle birlikte tanıtalacaktır.

2.2 Cebirler

A ve B birimli değişmeli halka ve $f : A \longrightarrow B$ halka homomorfizmi olsun. $a \in A, b \in B$ için

$$\begin{array}{ccc} A \times B & \longrightarrow & B \\ (a,b) & \longmapsto & a \cdot b = f(a)b \end{array}$$

işlemi tanımlayalım. Bu çarpımla birlikte B de bir A-modül yapısı oluşturur. Böylece B halka yapısı ile birlikte aynı zamanda modül yapısına da sahiptir.

Tanım 2.1 B halka yapısı aynı zamanda A-modül yapısına sahip ise B ye A-cebir denir.

Tanım 2.2 R değişmeli halka, A, B ve G birer R-modül olmak üzere $f : A \times B \to G$ fonksiyonu

$$\begin{array}{ll} i) & f(a+a',b) = f(a,b) + f(a',b) \\ ii) & f(a,b+b') = f(a,b) + f(a,b') \\ ii) & rf(a,b) = f(ra,b) = f(a,rb) \end{array}$$

şartlarını sağlıyor ise f ye R-bilineer denir.

Uyarı 2.3 1) A değişmeli fakat B değişmeli olmasın. Her $a \in A$ ve $b_1, b_2 \in B$ için

$$a(b_1 b_2) = (ab_1)b_2 = b_1(ab_2)$$

ise B ye A-cebir denir.

2) A değişmeli halka ve B bir A-modül olsun. Eğer

$$\begin{array}{rccc} f: & B\times B & \longrightarrow & B \\ & (b_1,b_2) & \longmapsto & \langle b_1,b_2\rangle \end{array}$$

bilineer fonksiyonu ise B bir A-cebirdir. *f bilineer* ise $b_1 \longmapsto \langle b_1,b_2\rangle$ ve $b_2 \longmapsto \langle b_1,b_2\rangle$ homomorfizmleri

$$\begin{array}{rl} f(b_1+b_1') & = \langle b_1+b_1',b_2\rangle = \langle b_1,b_2\rangle + \langle b_1',b_2\rangle \\ & = f(b_1)+f(b_1') \\ f(b_1b_1') & = \langle b_1b_1',b_2\rangle = \langle b_1,b_2\rangle\,\langle b_1',b_2\rangle \end{array}$$

$$\begin{array}{rl} f(b_2+b_2') & = \langle b_1,b_2+b_2'\rangle = \langle b_1,b_2\rangle + \langle b_1,b_2'\rangle \\ & = f(b_2)+f(b_2') \end{array}$$

şartlarını sağlar.

Cebir Örnekleri

1) Her B halkası toplamsal Abelyan gruptur. Buradan

$$\begin{array}{ccc} \mathbb{Z}\times B & \longrightarrow & B \\ (n,b) & \longmapsto & n\cdot b = \underbrace{b+\ldots+b}_{n \text{ tane}} \end{array}$$

işlemiyle B bir $\mathbb{Z}$-modüldür. Dolayısıyla her halka bir $\mathbb{Z}$-cebirdir.

2) Her B halkası aynı zamanda B-modül olduğundan bir B-cebirdir.

3) A birimli değişmeli halka olsun.

$$\begin{array}{rccc} f: & A & \hookrightarrow & A[X] \\ & a & \longmapsto & a=a+0x+\ldots \end{array}$$

fonksiyonu bir homomorfizmdir. Böylece $A[X]$ bir A-cebirdir.

4) V F-vektör uzayı olsun.

$$Hom_F(V,V) = \{f \mid f: V \longrightarrow V \text{ lineer dönüşüm}\}$$

bir F-cebirdir.

$$\begin{array}{rcl} F\times Hom(V,V) & \longrightarrow & Hom(V,V) \\ (\alpha,f) & \longmapsto & \alpha\cdot f: \begin{array}{rcl} V & \longrightarrow & V \\ v & \longmapsto & (\alpha\cdot f)(v)=f(\alpha v) \end{array} \end{array}$$

şeklinde tanımlanan işlemle $Hom(V,V)$ bir F-cebirdir.

5) A birimli değişmeli halka ve $Mat_{n\times n}(A)$ $n\times n$ tipindeki matrisler halkası verilsin. Bu durumda $Mat_{n\times n}(A)$ bir A-cebirdir.

$$\begin{array}{ccc} A\times Mat_{n\times n}(A) & \longrightarrow & Mat_{n\times n}(A) \\ (a,(a_{ij})) & \longmapsto & a\cdot(a_{ij})=(aa_{ij}) \end{array}$$

olmak üzere genel olarak B bir A-cebir ise $Mat_{n\times n}(B)$ bir A-cebirdir.

2.3 Grup Cebirleri

$R[G]$ ile gösterilen grup halka yapısı bir halkadır. Grup halka kavramı ilk olarak Cayley (1854) tarafından tanımlanmıştır. Bu yapıda R bir halka G ise bir gruptur. Bir grup halkanın elemanları G grubunun elemanlarının sonlu lineer kombinasyonları ile R nin elemanlarının katsayı olarak kullanılmasıyla oluşur.

Her $R[G]$ grup halkası için $R\leq R[G]$ olduğundan $R[G]$ bir R-modüldür. Eğer R bir cisim ise (değişmeli halka), grup halka yapısı *grup cebir* olarak adlandırılır. $R=\mathbb{Z}$ alınırsa $\mathbb{Z}[G]$, $\mathbb{Z}$-cebirine tam grup halka (integral group ring) adı verilir. Grup cebirler hakkında ayrıntılı bilgi için (Connel, 1963) ve (Passman, 1977) çalışmalarına bakılabilir.

Tanım 2.4 K bir cisim, $(G,*)$ bir grup olsun. Her $i\in I$ için $a_i\in K$ ve $g_i\in G$ olmak üzere, her elemanı

$$a_1g_1+a_2g_2+\cdots+a_ng_n$$

formunda olan, G nin elemanlarının sonlu lineer kombinasyonları ile K nın elemanlarını katsayı kabul edilmesinden oluşan $K[G]$ kümesini göz önüne alalım.

$K[G]$ nin herhangi elemanı genellikle

$$\sum_{g\in G} a_g g$$

biçiminde gösterilir. Aşağıdaki işlemlerle birlikte $K[G]$, K üzerinde bir cebirdir. Bu cebire *grup cebir* denir.

Toplama:

$$\sum_{g\in G} a_g g+\sum_{g\in G} b_g g=\sum_{g\in G}(a_g+b_g)g$$

Skalerle çarpma :

$$a\sum_{g\in G} a_g g = \sum_{g\in G} (aa_g)g$$

Çarpma:

$$\left(\sum_{g\in G} a_g g\right)\left(\sum_{h\in G} b_h h\right) = \sum_{g,h\in G} (a_g b_h)g*h$$

$|G| = n$ ve $|K| = m$ olmak üzere $|K[G]| = m^n$ dir.

Örnek 2.1 $G = C_3 = \langle g\rangle$ 3. mertebeden devirli grup olsun. $z_1, z_2, z_3 \in \mathbb{C}$ olmak üzere $\mathbb{C}[G]$ grup cebirinin herhangi bir elemanı

$$r = z_1 + z_2 g + z_3 g^2$$

şeklinde yazılır. $s \in \mathbb{C}[G]$ başka bir eleman olmak üzere

$$s = w_1 + w_2 g + w_3 g^2$$

elemanların toplamı

$$r + s = z_1 + w_1 + (z_2 + w_2)g + (z_3 + w_3)g^2$$

ve çarpımı

$$rs = z_1 w_1 + z_2 w_3 + z_3 w_2 + (z_1 w_2 + z_2 w_1 + z_3 w_3)g + (z_1 w_3 + z_3 w_1 + z_2 w_2)g^2$$

biçimindedir.

Örnek 2.2 $G = C_3 = \langle g\rangle$ 3. mertebeden devirli grup ve $K = \mathbb{Z}_2$ olmak üzere $K[G]$ grup cebirinin elemanları

$$\mathbb{Z}_2[C_3] = \left\{0, 1, g, g^2, 1+g, 1+g^2, g+g^2, 1+g+g^2\right\}$$

şeklindedir.

Önerme 2.5 a) K cisim ve G Abelyen ise $K[G]$ grup cebiri de değişmelidir.

b) H, G nin bir alt grubu ise $K[H]$ da $K[G]$ nin bir alt grubudur. Benzer şekilde S, K nin bir alt halkası ise $S[G]$ de $K[G]$ nin bir alt halkasıdır.

$f : G \to H$ herhangi bir grup homomorfizmi olmak üzere $K[f] : K[G] \to K[H]$ grup cebir homomorfizmi

$$\sum_{g\in G} a_g g \longmapsto \sum_{g\in G} a_g f(g)$$

şeklinde tanımlanabilir. $f' : H \to L$ başka bir grup homomorfizmi ise $K[ff'] = K[f]K[f']$ dir. Grup cebir tanımı ve grup cebir homomorfizm yardımıyla aşağıdaki önerme verilebilir.

Önerme 2.6 Herhangi bir grup alındığında, herzaman bir K-cebir

$$K[.]: \ \mathbf{Gr} \ \rightarrow \ \mathbf{K\text{-}Alg}$$

funktoru ile elde edilir.

$K[.]$ grup cebir funktorunun aksine herhangi bir cebirden bir grup elde edilebilir. Cebirdeki çarpma unutularak toplamsal abelyen grup elde edilir, bu da unutulabilir (forgetful)

$$\mathbf{K\text{-}Alg} \rightarrow \mathbf{Ab}$$

funktorunu verir. Ayrıca bilindiği üzere cebirdeki çarpmaya göre tersi bulunabilen elemanların oluşturduğu küme bir altgruptur. Bu gruba cebirin terslenebilen elemanları grubu denir, bu da

$$U(.): \ \mathbf{K\text{-}Alg} \ \rightarrow \mathbf{Gr}$$

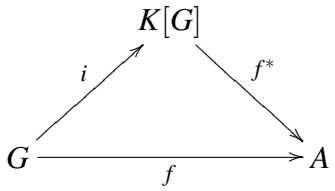
funktorunu verir. Genel olarak komutatif olmayan cebirlerin terslenebilen elemanları grubu abelyen olmak zorunda değildir. Buradan, ispatı (Barker, 2003) tarafından yapılan aşağıdaki önermeyi verebiliriz.

Önerme 2.7 $K[.]$: **Gr** $\rightarrow$ **K-Alg** grup cebir funktoru $U(.)$: **K-Alg** $\rightarrow$ **Gr** funktorunun sol ekidir. Böylece G bir grup ve A bir K-cebir olmak üzere

$$\mathbf{Gr}(G, U(A)) \quad \cong \quad \mathbf{K\text{-}Alg}(K[G], A)$$

izomorfizmi vardır.

Önerme 2.8 (Evrensellik Özelliği) G bir grup ve K bir cisim olsun. A, $K \subset A$ biçiminde herhangi bir halka ve $f : G \rightarrow A$ grup homomorfizmi olsun. Bu durumda $i : G \rightarrow K[G]$ içine dönüşüm olmak üzere



diyagramı değişmeli olacak şekilde (yani $f^* \circ i = f$) birtek

$$\begin{array}{rccc} f^*: & K[G] & \longrightarrow & A \\ & \sum\limits_{g \in G} a_g g & \longmapsto & \sum\limits_{g \in G} a_g f(g) \end{array}$$

homomorfizmi vardır.

$A = K[H]$ alınırsa $f^* : K[G] \rightarrow K[H]$ birtek grup cebir homomorfizmi vardır. Ayrıca;

ii f birebir ise f^* birebirdir.

iii f örten ise f^* da örtendir.

Tanım 2.9 $\varepsilon : K[G] \to K$

$$\varepsilon\left(\sum_{g\in G} a_g g\right) = \sum_{g\in G} a_g$$

homomorfizmine agümentasyon homomorfizmi denir. Bu homomorfizmin çekirdeğine ise agümentasyon ideali denir ve $\Delta(G)$ ile gösterilir.

Grup cebirlerin temel özelliklerinden birisi de sağ-sol simetri özelliğidir. Herhangi bir gruptaki her elemanın tersi varolduğundan herhangi $R[G]$ grup halka üzerinde aşağıdaki gibi homomorfizm tanımlanabilir.

Önerme 2.10 R değişmeli halka ve G bir grup olsun.

$$\left(\sum_{g\in G} a_g g\right) \longmapsto \sum_{g\in G} a_g g^{-1}$$

şeklinde tanımlanan $* : R[G] \to R[G]$ fonksiyonu aşağıdaki özellikleri sağlayan bir homomorfizmdir.

i $*(\alpha+\beta) = *(\alpha)+*(\beta)$,

ii $*(\alpha\beta) = *(\beta)*(\alpha)$,

iii $*(*(\alpha)) = \alpha$.

Her $g,h \in G$ için $(gh)^{-1} = h^{-1}g^{-1}$ olduğundan bu özelliklerin sağlandığı kolaylıkla gösterilebilir ve $*$ bir antiotomorfizmdir.

2.4 Lie Cebirleri

Tanım 2.11 F bir cisim ve L, F vektör uzayı olsun.

$$\begin{array}{ccc} L\times L & \longrightarrow & L \\ (x,y) & \longmapsto & [x,y] \end{array}$$

bilineer fonksiyonu

L1) Her $x \in L$ için $[x,x] = 0$

L2) Her $x,y,z \in L$ için $[x,[y,z]] + [y,[z,x]] + [z,[x,y]] = 0$

şartlarını sağlıyorsa L ye bir Lie cebir denir.

Lie cebiri hakkında ayrıntılı bilgi için (Erdmann ve Wildon, 2006) çalışmasına bakılabilir.

Genellikle $[x,y]$ Lie braketine x ve y nin komutatörü, ayrıca L2'ye de Jacobi özdeşliği denir.

$[\ ,\ ]$ Lie braket ikili işlemi,

$$\begin{aligned} 0 = [x+y, x+y] &= [x,x] + [x,y] + [y,x] + [y,y] \\ &= [x,y] + [y,x] \end{aligned}$$

ile bilineerdir. Ayrıca L1 durumu,

L1′) Her $x,y \in L$ için $[x,y] = -[y,x]$ şeklinde de ifade edilebilir. F cisminin karakteristiği 2 değil ise L1′ de $x = y$ alınarak L1 yerine L1′ kullanılır.

Örnek 2.3 M, A üzerinde bir cebir olsun. $[,] : M \times M \to M$ fonksiyonu

$$[x,y] = xy - yx$$

şeklinde tanımlansın. $[,]$ fonksiyonun iki lineer olduğunu gösterelim.

i)

$$[x,x] = xx - xx = 0$$

ii)

$$\begin{aligned} [x,[y,z]] + [y,[z,x]] + [z,[x,y]] &= [x,yz-zy] + [y,zx-xz] + [z,xy-yx] \\ &= x(yz-zy) - (yz-zy)x + (xy-yx)z \\ &\quad -(zx-xz)y + z(xy-yx) - (xy-yx)z \\ &= 0 \end{aligned}$$

olur. Böylece M, $[,]$ ile birlikte Lie cebiridir.

Örnek 2.4 $F = \mathbb{R}$ olsun. $(x,y) \longmapsto x \wedge y$ vektör çarpımı ile $\mathbb{R}^3$ de bir Lie Cebiri oluşturulabilir. $x = (x_1,x_2,x_3)$ ve $y = (y_1,y_2,y_3)$ olmak üzere

$$x \wedge y = (x_2y_3 - x_3y_2, x_3y_1 - x_1y_3, x_1y_2 - x_2y_1)$$

şeklindedir.

Tanım 2.12 M bir Lie cebiri olsun. Her $x,y \in M$ için $[x,y]=0$ oluyorsa M ye abelyen Lie cebiri denir. (Amoya, 1974)

Örnek 2.5 V,F üzerine sonlu-boyutlu vektör uzayı olsun. V den V ye bütün lineer fonksiyonların kümesi $GL(V)$ olarak yazılır. Bu da yine F üzerine bir vektör uzayıdır. $[\ ,\ ]$ Lie braket işlemi, her $x,y \in GL(V)$ için, $\circ$ işlemi fonksiyonların bileşke işlemi olmak üzere

$$[x,y] := x \circ y - y \circ x$$

şeklindedir. Böylelikle $GL(V)$ bir Lie cebirdir. Ayrıca $GL(V)$ ye *genel lineer cebir* denir.

Tanım 2.13 $K \subseteq L$ olmak üzere her $x,y \in K$ için $[x,y] \in K$ ise K ya L F-vektör uzayının (veya L Lie cebirinin) alt cebiri denir.

Tanım 2.14 L Lie cebiri için I,L nin alt uzayı olmak üzere her $x \in L$ ve $y \in I$ için $[x,y] \in I$ ise I ya L nin bir ideali denir. Ayrıca ideal bir alt cebirdir. Diğer taraftan alt cebir bir ideal olmak zorunda değildir. L1$'$ şartında $[x,y] = -[y,x]$ olmasıyla sağ ve sol ideallerin ayrı ayrı ifade edilmesine gerek yoktur. L Lie cebiri kendisinin bir idealidir. Ayrıca $\{0\}$ da yine L nin bir idealidir. Bu ideallere L nin aşikar idealleri denir. İdealler için önemli bir örnek de, çoğunlukla aşikar olmayan

$$Z(L) := \{x \in L : [x,y] = 0, \text{her } y \in L \text{ için}\}$$

şeklinde tanımlanan L nin merkezidir. $L = Z(L)$ olması için gerek ve yeter koşulun L nin Abelyan olması gerektiğini biliyoruz.

Örnek 2.6 I ve J,L Lie cebirinin idealleri olsun. I ve J den yeni idealler oluşturmanın birçok yolu vardır. Öncelikle $I \cap J$ nin L nin bir ideali olduğunu gösterelim. Öncelikle $I \cap J$ nin L nin altuzayı olduğunu biliyoruz. Bu durumda $x \in L$ ve $\in y \in I \cap J$ ise $[x,y] \in I \cap J$ dir.

Tanım 2.15 L,F cismi üzerinde $(x_1,...,x_n)$ tabanı ile bir Lie cebiri ise $[x_i,x_j]$ çarpımı tarafından $[\ ,\ ]$ işlemi $a_{ij}^k \in F$ skaleri $[x_i,x_j] = \sum_{k=1}^{n} a_{ij}^k x_k$ şeklinde tanımlanır. $a_{ij}^k, (x_1,...,x_n)$ tabanı ile L nin oluşturulmuş sabitleridir. a_{ij}^k, L nin tabanının seçimine bağlıdır. Farklı tabanlar genelde farklı oluşturulmuş sabitleri verir.

Örnek 2.7 $I+J = \{x+y : x \in I, y \in J\}$ olmak üzere $I+J, L$ nin idealidir. Böylece ideallerin çarpımını tanımlayabiliriz. $[I,J] := Span\{[x,y] : x \in I, y \in J\}$ olsun. Öncelikle, tanımı gereği $[I,J], L$ nin altuzayıdır. İkinci olarak $x \in I, y \in J$ ve $u \in L$ olmak üzere Jacobi birimliğinden $[u,[x,y]] = [x,[u,y]] + [[u,x],y]$ dir. Burada $[u,y] \in J, J$ nin bir idealidir, böylece $[x,[u,y]] \in [I,J]$, benzer şekilde $[[u,x],y] \in [I,J]$ dir. Dolayısıyla toplamları da $[I,J]$ dedir.

Örnek 2.8 $[I,J]$ nin genel elemanı t olmak üzere, $x \in I, y \in J$ ile $[x,y]$ braketlerinin lineer kombinasyonu ile $t = \sum c_i[x_i,y_j]$ şeklindedir. Burada c_i ler birer skaler ve $x_i \in I, y_j \in J$ dir. Böylece $u \in L$ için

$$[u,t] = [u, \sum c_i[x_i,y_j]] = \sum c_i[u,[x_i,y_j]]$$

şeklinde olup $[u,[x_i,y_j]] \in [I,J]$ dir. O halde $[u,t] \in [I,J]$ olup $[I,J], L$ nin bir idealidir.

Tanım 2.16 F üzerinde L_1 ve L_2 iki Lie cebiri olsun. $\varphi : L_1 \longrightarrow L_2$ lineer fonksiyonu her $x,y \in L_1$ için

$$\varphi([x,y]) = [\varphi(x), \varphi(y)]$$

ise φ ye bir homomorfizm denir. Burada eşitliğin sol tarafındaki L_1 in, sağ tarafındaki ise L_2 nin braket işlemidir. φ bire bir ve örten ise φ bir izomorfizmdir.

Örnek 2.9 L bir Lie cebiri olsun. $x,y \in L$ için

$$\begin{array}{rccl} \mathrm{Ad}: & L & \longrightarrow & \mathrm{GL}(L) \\ & x & \longmapsto & ((\mathrm{Ad})(x))(y) = [x,y] \end{array}$$

fonksiyonu bir Lie cebir homomorfizmidir. Bu homomorfizme Adjoint homomorfizmi denir. Ad nin çekirdeği L nin merkezidir.

Tanım 2.17 I, L Lie cebirinin bir ideali ise I, L nin bir altuzayıdır, böylece $z \in L$ için $z+I = \{z+x : x \in I\}$ olmak üzere bölüm vektör uzayı

$$L/I = \{z+I : z \in L\}$$

şeklindedir. L/I kümesi

$$[w+I, z+I] = [w,z]+I$$

işlemiyle birlikte bir Lie cebiridir. Bu Lie cebirine L/I bölüm cebiri adı verilir.

Önerme 2.18 Lie cebirleri için izomorfizm teoremleri,

a) $\varphi : L_1 \longrightarrow L_2$ Lie cebirlerin bir homomorfizmi olsun. Böylece $\ker\varphi, L_1$ in bir ideali ve $\mathrm{Im}\,\varphi, L_2$ nin bir altcebiridir ve $L_1/\ker\varphi \cong \mathrm{Im}\,\varphi$ dir.

b) I ve J bir Lie cebirinin idealleri ise $(I+J)/J \cong I/(I \cap J)$ dir.

c) I ve J bir L Lie cebirinin idealleri ve $I \subseteq J$ olmak üzere $J/I, L/I$ nın bir ideali olmak üzere $(L/I)/(J/I) \cong L/J$ dir.

Tanım 2.19 L Lie cebirinin düşük merkezli serileri $k \geq 2$ için $L^1 = L'$ ve $L^k = [L, L^{k-1}]$ şeklindedir. Böylece $L \supseteq L^1 \supseteq L^2 \supseteq ...$ ideallerin çarpımı yine bir ideal olduğu gibi L^k da L nin bir idealidir. Bu durumda L/L^{k+1} in merkezlerinden oluşan L^k/L^{k+1} ya merkez seriler denir.

Tanım 2.20 Bazı $m \geq 1$ için $L^m = 0$ ise L Lie cebirine nilpotent denir.

Yardımcı Teorem 2.21 L bir Lie cebir olsun

a) L nilpotent ise L nin her Lie altcebiri nilpotenttir.

b) $L/Z(L)$ nilpotent ise L de nilpotenttir.

BÖLÜM 3

LAGUNA Ortak Paketi

3.1 Giriş

LAGUNA Victor Bovdi, Alexander Konovalov, Richard Rossmanith, Csaba Schneider tarafından 2009 da yazılmış grup cebirlerin ve Lie cebirlerin özelliklerinin bilgisayar ortamına aktarılmasına olanak sağlayan bir GAP ortak paketidir. LAGUNA ortak paketi yardımıyla herhangi bir R halkası ve bir grup yardımıyla bir sol R-modül oluşturulabilir. Buradaki R halkası yerine F cismi alınırsa oluşan yapı bir cisim olacaktır. Örneğin 32. mertebeden Dihedral grup, 2. mertebeden Galois cismi ve $\mathbb{Z}_4$ halkası kullanılarak $K[G]$ ve $R[G]$ grup halkaları oluşturulabilir. Bu bölümde LAGUNA ortak paketi incelenmiştir.

GAP

```
gap> G:=DihedralGroup(32);
<pc group of size 32 with 5 generators>
gap> K:=GaloisField(2);
GF(2)
gap> KG:=GroupRing(K,G);
<algebra-with-one over GF(2), with 5 generators>
gap> R:=Integers mod 4;
(Integers mod 4)
gap> RG:=GroupRing(R,G);
<free left module over (Integers mod 4), and ring-with-one, with 5 generators>
```

Herhangi bir grup halkası verildiğinde, bu halkayı oluşturan grup ve halkayı bulmak için `LeftActionDomain` ve `UnderlyingGroup` denetim deyimleri kullanılabilir.

GAP

```
gap> UnderlyingGroup(KG);
<pc group of size 32 with 5 generators>
gap> LeftActingDomain(KG);
GF(2)
gap> UnderlyingRing(RG);
(Integers mod 4)
gap> UnderlyingField(KG);
GF(2)
```

Grup halkayı oluşturan halkanın cisim olup olmaması durumuna göre `LeftActionDomain` denetim deyimi yerine `UnderlyingField` veya `UnderlyingRing` denetim deyimleri de aynı

sonucu verir.

3.2 Grup Cebiri İçin Genel Fonksiyonlar

◊ IsGroupAlgebra(KG) (özellik)

GroupRing denetim deyimi ile oluştuşturulmuş cebirsel yapının bir cebir olup olmadığını denetlemek için IsGroupAlgebra denetim deyimi kullanılır. Yapı bir cebir ise GAP programı true yanıtı verir.

GAP

```
gap> IsGroupAlgebra(KG);
true
gap> IsAlgebra(KG);
true
gap> IsGroupAlgebra(RG);
false
gap> IsLeftModule(RG);
true
```

◊ IsFModularGroupAlgebra(KG) (özellik)

K cisminin karakteristiği, G nin bazı elemanlarının mertebesini bölüyor ise $K[G]$ grup cebiri *modüler* olarak adlandırılır. GAP programında IsFModularGroupAlgebra denetim deyimi grup cebirlerin modüler olup olmadığını kontrol eder.

GAP

```
gap> IsFModularGroupAlgebra(GroupRing(GF(3),SymmetricGroup(6)));
true
gap> IsFModularGroupAlgebra(GroupRing(GF(3),CyclicGroup(7)));
false
gap> Characteristic(GF(3));
3
gap> List(CyclicGroup(7),Order);
[ 1, 7, 7, 7, 7, 7, 7 ]
```

◊ IsPModularGroupAlgebra(KG) (özellik)

Bir grubun birimden farklı her elemanının mertebesi bir p asal sayısının kuvveti ise bu gruba p-grup denir. Bu özellikten dolayı p-gruplar nilpotent özellik gösterirler. K karakteristiği

p olan cisim ve G de aynı p asal sayısı için p-grup ise $K[G]$ grup cebirine p-modüler denir. GAP programında `IsPModularGroupAlgebra` denetim deyimi bir $K[G]$ grup cebirinin p-modüler olup olmadığını denetler.

GAP

```
gap> List(G,Order);
[ 1, 2, 4, 4, 8, 8, 8, 8, 16, 16, 16, 16, 16, 16, 16, 16, 2, 2, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ]
gap> IsPGroup(G);
true
gap> IsNilpotent(G);
true
gap> PrimePGroup(G);
2
gap> Characteristic(K);
2
gap> IsPModularGroupAlgebra(KG);
true
gap> IsPModularGroupAlgebra( GroupRing( GF( 2 ), SymmetricGroup( 6 ) ) );
false
```

3.3 Grup Cebir Elemanları

◊ `Support(x)` (nitelik)

Bu denetim deyimi, $\alpha_i \in R$ ve $g_i \in G$ olmak üzere $R[G]$ grup halkasının herhangi $x = \alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ elemanındaki $g_i \in G$ lerin listesini verir.

GAP

```
gap> KG:=GroupRing(GF(3),CyclicGroup(8));
<algebra-with-one over GF(3), with 3 generators>
gap> eL:=Elements(KG);;
gap> Size(KG);
6561
gap> x:=eL[6001];
(Z(3)^0)*f2+(Z(3)^0)*f1*f2+(Z(3)^0)*f1*f3+(Z(3)^0)*f2*f3+(Z(3))*f1*f2*f3
gap> Support(x);
[ f2, f1*f2, f1*f3, f2*f3, f1*f2*f3 ]
```

◊ `CoefficientsBySupport(x)` (nitelik)

Bu denetim deyimi `Support` denetim deyimi ile benzer olarak $x \in R[G]$ elemanında kullanılan $\alpha_i \in R$ leri listeler.

GAP
```
gap> CoefficientsBySupport(x);
[ Z(3)^0, Z(3)^0, Z(3)^0, Z(3)^0, Z(3) ]
```

◊ TraceOfMagmaRingElement(x) (nitelik)

Bu denetim deyimi $R[G]$ nin $x = \alpha_1 \cdot 1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ şeklindeki elemanında G'nin birim elemanının katsayısı olan $\alpha_1 \in R$ yi verir.

GAP
```
gap> TraceOfMagmaRingElement(x);
0*Z(3)
```

◊ Length(x) (nitelik)

Bu denetim deyimi, bir x elamanını oluşturan lineer toplamdaki elemanların sayısını verir. Açıkca bu sayı G nin eleman sayısını geçemez.

GAP
```
gap> Length(x);
5
```

◊ Augmentation(x) (nitelik)

Bu denetim deyimi $x = \alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ şeklindeki grup halka elemanının $\alpha_1 + \alpha_2 \cdots + \alpha_k$ katsayılarının toplamını verir.

GAP
```
gap> Augmentation(x);
0*Z(3)
```

◊ Involution(x,f) (operasyon)

◊ Involution(x) (operasyon)

$R[G]$ bir grup halkası, f^2 birim fonksiyon olacak şekilde $f : G \to G$ bir fonksiyon olsun. İnvolüsyon fonksiyonu $R[G]$ nin $x = \alpha_1 \cdot g_1 + \alpha_2 \cdot g_2 + \cdots + \alpha_k \cdot g_k$ elemanını $\alpha_1 \cdot f(g_1) + \alpha_2 \cdot f(g_2) + \cdots + \alpha_k \cdot f(g_k)$ elemanına dönüştürür. GAP programında Involution denetim deyimi

bu işlemi yapmak için kullanılır. Bu opersayon bir veya iki parametreli olarak kullanılabilir. İki parametreli kullanımda, kullanıcı f fonksiyonunu kendisi belirlerken, tek parametreli kullanımda $x \mapsto x^{-1}$ klasik involüsyonu f fonksiyonu olarak alınır.

GAP
```
gap> Involution(x);
(Z(3))*f1+(Z(3)^0)*f2+(Z(3)^0)*f1*f2+(Z(3)^0)*f1*f3+(Z(3)^0)*f2*f3
```

◊ IsSymmetric(x) (nitelik)

$R[G]$ grup halkasının bir x elemanı, klasik involüsyon işlemi altında aynı kalıyorsa simetrik olarak adlandırılır. GAP programında `IsSymmetric` denetim deyimi verilen elemanın simetrik olup olmadığını denetler.

GAP
```
gap> IsSymmetric(x);
false
gap> IsSymmetric(x*Involution(x));
true
```

◊ IsUnit(x) (metod)

◊ InverseOp(x) (metod)

$R[G]$ grup halkasının bir x elemanının tersinin olup olmadığını hesaplamak için `IsUnit` denetim deyimi, varolan tersi hesaplamak için `InverseOp` denetim deyimi kullanılır. $x\hat{}-1$ ifadesi de tersi hesaplamak için kullanılabilir.

GAP
```
gap> IsUnit(x);
false
gap> x^-1;
fail
gap> y:=eL[1001];
(Z(3)^0)*<identity> of ...+(Z(3))*f1+(Z(3))*f2+(Z(3)^0)*f3+(Z(3)^0)*f1*f2+(
Z(3))*f2*f3+(Z(3)^0)*f1*f2*f3
gap> IsUnit(y);
true
gap> y^-1;
(Z(3)^0)*<identity> of ...+(Z(3)^0)*f1+(Z(3))*f2+(Z(3)^0)*f3+(Z(3)^0)*f1*f3+(
Z(3))*f2*f3+(Z(3))*f1*f2*f3
```

3.4 Grup Cebir İşlemleri

◊ AugmentationHomomorphism(RG) (nitelik)

Bu operasyon $R[G] \to R$ biçiminde, tanım 2.9 da verilen agümantasyon homomorfizmini oluşturmak için kullanılır. $R[G]$ nin herhangi bir elemanının bu homomorfizm altındaki görüntsünü bulmak için `Augmentation` denetim deyimi daha hızlı sonuç verir.

GAP

```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
gap> f:=AugmentationHomomorphism(KG);
[ (Z(3)^0)*f1, (Z(3)^0)*f2, (Z(3)^0)*f3 ] -> [ Z(3)^0, Z(3)^0, Z(3)^0 ]
gap> IsSurjective(f);
true
gap> Augmentation(x)=Image(f,x);
true
gap> Augmentation(x+y);
Z(3)
```

◊ AugmentationIdeal(KG) (nitelik)

Tanım 2.9 da verilen agümentasyon ideali, agümentasyon homomorfizminin çekirdeğinden oluşur. Başka bir deyişle `Augmentation` denetim deyimi R nin sıfırını olan $R[G]$ nin elemanları kümesini verir.

GAP

```
gap> A:=AugmentationIdeal(KG);
<two-sided ideal in <algebra-with-one of dimension 8 over GF(3)>,
  (3 generators)>
gap> IsIdeal(KG,A);
true
gap> eA:=Elements(A);;
gap> Image(f,eA[8]);
0*Z(3)
gap> A=Kernel(f);
true
```

◊ AugmentationIdealPowerSeries(KG) (nitelik)

Bu denetim deyimi, $K[G]$ bir p-modüler grup cebiri olmak üzere agümentasyon ideal yardımı ile $K[G]$ nin ideallerinin listesini verir. `AugmentationIdealPowerSeries(KG)[i]` denetim deyimi $K[G]$ agümentasyon idealinin i. kuvvetini verir.

GAP
```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> AugmentationIdealPowerSeries( KG );
[ <algebra of dimension 15 over GF(2)>, <algebra of dimension 13 over GF(2)>,
  <algebra of dimension 11 over GF(2)>, <algebra of dimension 9 over GF(2)>,
  <algebra of dimension 7 over GF(2)>, <algebra of dimension 5 over GF(2)>,
  <algebra of dimension 3 over GF(2)>, <algebra of dimension 1 over GF(2)>,
  <algebra over GF(2)> ]
gap> Length(last);
9
```

◊ AugmentationIdealNilpotencyIndex(KG) (nitelik)

$K[G]$ bir p-modüler grup cebiri ve A agümentasyon ideali olmak üzere $A^n = 0$ olacak şekildeki en küçük n sayısı bu denetim deyimi ile hesaplanır.

GAP
```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> AugmentationIdealNilpotencyIndex( KG );
9
```

◊ AugmentationIdealOfDerivedSubgroupNilpotencyIndex(KG) (nitelik)

$K[G]$ p-modüler grup cebiri ve $\overline{G}$, G nin komutatör altgrubu olmak üzere $K\left[\overline{G}\right]$ grup cebirinin agümentasyon idealinin nilpotentlik indeksi bu denetim deyimi ile bulunur.

GAP
```
gap> KG:=GroupRing(GF(2),DihedralGroup(32));
<algebra-with-one over GF(2), with 5 generators>
gap> AugmentationIdealOfDerivedSubgroupNilpotencyIndex(KG);
8
gap> D:=DerivedSubgroup(UnderlyingGroup(KG));
Group([ f3, f4, f5 ])
gap> KD:=GroupRing(GF(2),D);
<algebra-with-one over GF(2), with 3 generators>
gap> AugmentationIdealNilpotencyIndex(KD);
8
```

◊ LeftIdealBySubgroup(KG, H) (operasyon)

◊ RightIdealBySubgroup(KG, H) (operasyon)

◊ TwoSidedIdalBySubgroup(KG, H) (operasyon)

◊ LeftIdealBySubgroup(KG, H) (operasyon)

Bu denetim deyimleri grup halkanın ideallerini oluşturmak için kullanılırlar. $R[G]$ bir grup halka ve H, G nin bir altgrubu olsun. Böylece $R[G]$ nin elemanlarından oluşan $J_l(H)$,

$$\sum_{h\in H} x_h(h-1)$$

formunda, $h \in H$ elemanları tarafından üretilen bir sol idealdir. Sağ ideal olan $J_r(H)$ benzer şekilde tanımlanır. $J_l(H)$ nin bir ideal olması için gerek ve yeter şart $H \trianglelefteq G$ olmasıdır. H nın normal olmayan bir alt grubunu kullanarak ideal oluşturmaya çalışmak GAP programında hata mesajına sebep olacaktır.

GAP

```
gap> KG := GroupRing( GF(2), DihedralGroup(16) );
<algebra-with-one over GF(2), with 4 generators>
gap> G := DihedralGroup(16);
<pc group of size 16 with 4 generators>
gap> KG := GroupRing( GF(2), G );
<algebra-with-one over GF(2), with 4 generators>
gap> D := DerivedSubgroup( G );
Group([ f3, f4 ])
gap> LeftIdealBySubgroup( KG, D );
<two-sided ideal in <algebra-with-one over GF(2), with 4 generators>,
  (dimension 12)>
gap> H := Subgroup( G, [ GeneratorsOfGroup(G)[1] ]);
Group([ f1 ])
gap> IsNormal( G, H );
false
gap> LeftIdealBySubgroup( KG, H );
<left ideal in <algebra-with-one over GF(2), with 4 generators>,
  (dimension 8)>
```

◊ NormalizedUnitGroup(KG) (nitelik)

Bu denetim deyimi, grup cebir elemanları tarafından üretilen grubu verir. Herhangi bir halkanın tersinir olan elemanlarının çarpımsal bir grup oluşturduğu biliniyor. $K[G]$, p-modüler grup cebirinin tersinir olan elemanlarının oluşturduğu grup $T(K[G])$ olsun. α, $K[G]$ nin agümentasyon homomorfizmi ve $u \in T(K[G])$ olmak üzere $\alpha(u) = 1_K$ özelliğini sağlayan u elemanına normalleştirilmiş tersinir eleman denir. $K[G]$ nin tüm normalleştirilmiş tersinir elemanları kümesi $T(K[G])$ nin bir alt grubudur. $V(K[G])$ ile gösterilen bu alt gruba grup cebirinin normalleştirilmiş tersinir elemanları grubu denir. K^*, K nın çarpımsal grubu olmak üzere

$$T(K[G]) = K^* \times V(K[G])$$

eşitliği sağlanır.

GAP
```
gap> KG := GroupRing( GF( 2 ), DihedralGroup( 16 ) );
<algebra-with-one over GF(2), with 4 generators>
gap> V := NormalizedUnitGroup( KG );
<group of size 32768 with 15 generators>
gap> u := GeneratorsOfGroup( V )[4];
(Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f1*f2
```

◊ Units(KG) (nitelik)

Bu denetim deyimi bir grup cebirinin tersinir olan elemanlarının oluşturduğu $T(K[G])$ grubunu oluşturur. GAP programı bu grubu oluşturmak için

$$T(K[G]) = K^* \times V(K[G])$$

direk çarpımını kullanır.

GAP
```
gap> T := Units( KG );
#I  LAGUNA package: Computing the unit group ...
<group of size 32768 with 15 generators>
gap> GeneratorsOfGroup( U )[5];
(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f2*f3
gap> IsSubgroup(T,V);
true
gap> FH := GroupRing( GF(3), SmallGroup(27,3) );
<algebra-with-one over GF(3), with 3 generators>
gap> T := Units( FH );
#I  LAGUNA package: Computing the unit group ...
<group of size 5083731656658 with 27 generators>
gap> x := GeneratorsOfGroup( T )[1];
Tuple( [ Z(3), (Z(3)^0)*<identity> of ... ] )
gap> x in FH;
false
gap> x[1] * x[2] in FH;
true
```

◊ UnderlyingGroupRing(U) (nitelik)

Bu denetim deyimi Units veya NormalizedUnitGroup denetim deyimleri kullanılarak elde edilmiş U grubu için kullanılan p-modüler grup cebirini verir.

GAP
```
gap>A:= UnderlyingGroupRing( V );
<algebra-with-one over GF(2), with 4 generators>
gap>B:= UnderlyingGroupRing( T );
<algebra-with-one over GF(2), with 4 generators>
gap> KG=A; KG=B;
true
true
```

◊ GroupBases(KG) (nitelik)

$KB = KG$ ve K cismi üzerine B' nin elemanları lineer bağımsız ise KG grup cebirinin normalleştirilmiş tersinir elemanlar grubunun altgrubu olan B' ye grup taban denir. KG, p-modüler grup cebiri ise GroupBases denetim deyimi KG grup cebirinin normalleştirilmiş tersinir elemanlar grubunda, grup tabanlarının denklik sınıflarının temsilinin bir listesini verir.

GAP
```
gap> D8 := DihedralGroup( 8 );
<pc group of size 8 with 3 generators>
gap> K := GF(2);
GF(2)
gap> KD8 := GroupRing( GF( 2 ), D8 );
<algebra-with-one over GF(2), with 3 generators>
gap> gb := GroupBases( KD8 );;
gap> Length( gb );
32
gap> gb[1];
[ (Z(2)^0)*<identity> of ..., (Z(2)^0)*f3,
  (Z(2)^0)*f1*f2+(Z(2)^0)*f2*f3+(Z(2)^0)*f1*f2*f3,
  (Z(2)^0)*f2+(Z(2)^0)*f1*f2+(Z(2)^0)*f1*f2*f3,
  (Z(2)^0)*<identity> of ...+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f2*f3+(Z(2)^
    0)*f1*f2*f3, (Z(2)^0)*f2+(Z(2)^0)*f1*f3+(Z(2)^0)*f2*f3,
  (Z(2)^0)*<identity> of ...+(Z(2)^0)*f2+(Z(2)^0)*f3+(Z(2)^0)*f1*f2+(Z(2)^
    0)*f2*f3, (Z(2)^0)*f1+(Z(2)^0)*f2+(Z(2)^0)*f2*f3 ]
gap> Length( last );
8
```

3.5 Bir Grup Cebirinin Lie Cebiri

◊ LieAlgebraByDomain(A) (metod)

Bu denetim deyimi $[a,b] = ab - ba$ braket işlemiyle birlikte bir Lie cebiri oluşturmak için kullanılır. LAGUNA ortak paketinde LieAlgebraByDomain yerine direkt olarak LieAlgebra denetim deyimi de kullanılabilir.

GAP
```
gap> G:=SymmetricGroup(6);
Sym( [ 1 .. 6 ] )
gap> KG:=GroupRing(GF(8),G);
<algebra-with-one over GF(2^3), with 2 generators>
gap> L:=LieAlgebraByDomain(KG);
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2^3)>
```

◊ IsLieAlgebraByAssociativeAlgebra(L) (kategori)

Bu denetim deyimi ile Lie cebirin, bir asosyetif cebir kullanılarak oluşturulup oluşturulmadığı test edilir.

GAP
```
gap> IsLieAlgebraByAssociativeAlgebra(L);
true
gap> m:=[[0,1,2],[0,0,3],[0,0,0,0]];
[ [ 0, 1, 2 ], [ 0, 0, 3 ], [ 0, 0, 0, 0 ] ]
gap> A:=AlgebraWithOne(Rationals,[m]);
<algebra-with-one over Rationals, with 1 generators>
gap> IsLieAlgebraByAssociativeAlgebra(A);
false
gap> One(A);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
```

◊ UnderlyingAssociativeAlgebra(L) (nitelik)

Bu denetim deyimi ile Lie cebiri oluşturulurken kullanılan asosyetif cebiri elde edilir.

GAP
```
gap> UnderlyingAssociativeAlgebra(L);
<algebra-with-one over GF(2^3), with 2 generators>
gap> KG=last;
true
gap> UnderlyingAssociativeAlgebra(A);
Error, no method found! For debugging hints type ?
```

◊ NaturalBijectionToLieAlgebra(A) (nitelik)

Bu denetim deyimi $f : A \to L$ şeklinde birebir örten fonksiyonu oluşturur. Bu fonksiyon, cebir izomorfizmi olmamasına rağmen bir vektör uzayı izomorfizmidir.

GAP
```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> t := NaturalBijectionToLieAlgebra( FG );
MappingByFunction( <algebra-with-one over GF(2), with
2 generators>, <Lie algebra over GF(
2)>, <Operation "LieObject">, function( y ) ... end )
```

◊ NaturalBijectionToAssociativeAlgebra(L) (nitelik)

Bu denetim deyimi yukarıda tanımlanan f fonksiyonunun $f^{-1} : L \to A$ biçiminde ters fonksiyonu verir.

GAP
```
gap> G := SymmetricGroup(3); FG := GroupRing( GF( 2 ), G );
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> s := NaturalBijectionToAssociativeAlgebra( L );
MappingByFunction( <Lie algebra over GF(2)>, <algebra-with-one over GF(
2), with 2 generators>, function( y ) ... end, <Operation "LieObject"> )
gap> InverseGeneralMapping( s ) = NaturalBijectionToLieAlgebra( FG );
true
```

◊ IsLieAlgebraOfGroupRing(L) (özellik)

Bu denetim deyimi L Lie cebirinin oluşturulduğu asosyetif cebirin bir grup halkası olup olmadığını denetler.

GAP
```
gap> IsLieAlgebraOfGroupRing( L );
true
```

◊ UnderlyingGroup(L) (nitelik)

$R[G]$ grup halkası tarafından oluşturulmuş L Lie cebirinden G grubunu elde etmek için bu denetim deyimi kullanılır.

GAP
```
gap> F := GF( 2 ); G := SymmetricGroup( 3 ); FG := GroupRing( F, G );
GF(2)
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> UnderlyingGroup( L );
Sym( [ 1 .. 3 ] )
gap> LeftActingDomain( L );
GF(2)
```

◊ LieCentre(L) (metod)

Bu denetim deyimi bir $K[G]$ grup cebiri üzerinde tanımlanmış Lie cebirinin merkezini verir. Bir Lie cebirinin merkezi G grubunun merkezi ve K cismi tarafından üretilen grup, cebirin Lie cebiridir. L Lie cebirinin merkezi olan C bir ideal oluşturur.

◊ LieDerivedSubalgebra(L) (metod)

Bu denetim deyimi ile bir Lie cebirinin alt cebiri elde edilir. $K[G]$ grup cebiri yardımıyla elde edilen L Lie cebirinin $K\left[\overline{G}\right]$ grup cebiri yardımıyla elde edilen alt Lie cebiri de elde edilmiş olur.

GAP
```
gap> G := SmallGroup( 256, 400 ); FG := GroupRing( GF( 2 ), G );
<pc group of size 256 with 8 generators>
<algebra-with-one over GF(2), with 8 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> C := LieCentre( L );
<Lie algebra of dimension 28 over GF(2)>
gap> D := LieDerivedSubalgebra( L );
#I  LAGUNA package: Computing the Lie derived subalgebra ...
<Lie algebra of dimension 228 over GF(2)>
gap> l := Dimension( L ); c := Dimension( C ); d := Dimension( D );
256
28
228
gap> c + d = l;
true  # This is always the case for Lie algebras of group algebras!
```

◊ IsLieAbelian(L) (metod)

Bu denetim deyimi, A asosyatif cebiri üzerinde tanımlı Lie cebirinin değişmeli olup olmadığını test eder. L Lie cebirinin değişmeli olması için gerek ve yeter şart A nın değişmeli olmasıdır. GAP programı için `IsAbelyan` denetim deyimi Lie cebiri için doğru sonuç vermez.

GAP

```
gap> G := SymmetricGroup( 3 ); FG := GroupRing( GF( 2 ), G);
Sym( [ 1 .. 3 ] )
<algebra-with-one over GF(2), with 2 generators>
gap> L := LieAlgebra( FG );
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra over GF(2)>
gap> IsAbelian( G );
false
gap> IsAbelian( L );
true
gap> IsLieAbelian( L );
false
```

BÖLÜM 4

Çaprazlanmış Modül Kavramı

4.1 Giriş

Çaprazlanmış modül kavramı Whitehead (1949) tarafından tanımlanmıştır. Alp ve Wensley (2000) de GAP aracılığıyla gruplar üzerinde çaprazlanmış modülleri bilgisayar ortamına aktarmıştır. XMOD adını verdikleri ortak paket, Cayley teoreminin de yardımıyla simetrik gruplar üzerinde çalışarak sonlu mertebeli grubun çaprazlanmış modülünü oluşturur ve diğer tüm özelliklerini incelenebilir hale getirir.

Değişmeli cebirler üzerinde çaprazlanmış modüller Porter (1986) tarafından, GAP kullanılarak bilgisayar ortamına aktarılması ise Odabaş (2009) tarafından yapılmıştır.

Kassel ve Loday (1982) çalışmasında Lie cebirler üzerinde çaprazlanmış modülleri tanımlamıştır. Bu bölümde Lie cebirleri üzerinde çaprazlanmış modülleri ve genel özellikleri verilmiştir. Bu bölümde ayrıca Lie cebirleri üzerinde çaprazlanmış modülleri GAP programına aktarmak için yazdığımız operasyonlar tanıtılacaktır.

Birinci bölümde tanıtılan GAP programlama dili, LAGUNA paketinin verdiği destek ve cebirler üzerinde çaprazlanmış modüller oluşturmak için kullanılan yöntemlerin geliştirilerek Lie cebirleri üzerinde çaprazlanmış modül oluşturmak için kullanılır.

Tanım 4.1 M ve S iki Lie $\mathbf{K}$ -cebirler olmak üzere M üzerinde S nin Lie etkisi, aşağıdaki aksiyomları sağlayan

$$\begin{array}{ccc} S \times M & \longrightarrow & M \\ (s,m) & \longmapsto & s \cdot m \end{array}$$

dönüşümdür. Her $k \in \mathbf{K}$, $m, m' \in M$ ve $s, s' \in S$ için

i. $k(s \cdot m) = (ks) \cdot m = s \cdot (km)$

ii. $s \cdot (m + m') = s \cdot m + s \cdot m'$

iii. $(s + s') \cdot m = s \cdot m + s' \cdot m$

iv. $[s, s'] \cdot m = s(s' \cdot m) - s'(s \cdot m)$

v. $\quad s\cdot[m,m'] = [s\cdot m, m'] + [m, s\cdot m']$

Tanım 4.2 M ve S iki Lie **K** -cebir olsun.

$$\mu \;:\; M \longrightarrow S$$

bir Lie **K** -cebir morfizmi ve

$$\begin{array}{ccc} S\times M & \longrightarrow & M \\ (s,m) & \longmapsto & s\cdot m \end{array}$$

S nin M üzerine Lie etkisi ile birlikte her $m,m'\in M$ ve $s,s'\in S$ için

ÇM1) $\mu(s\cdot m) = [s,\mu(m)]$

ÇM2) $\mu(m)\cdot m' = [m,m']$

şartları sağlanıyor ise (M,S,μ) üçlüsüne Lie çaprazlanmış modül denir. Sadece (ÇM1) aksiyomunu sağlayan (M,S,μ) üçlüsüne Lie ön çaprazlanmış modül denir. (ÇM2) özelliğine de Peiffer özdeşiği denir.

Örnek: R bir Lie cebir ve I,R nin bir ideali olsun.

$$\begin{array}{rccc} \partial: & I & \longrightarrow & R \\ & i & \longmapsto & i \end{array}$$

içine dönüşümünü ele alalım. R nin I üzerine etkisi

$$\begin{array}{ccc} R\times I & \longrightarrow & I \\ (r,i) & \longmapsto & ri = [r,i] \end{array}$$

şeklinde Lie çarpım işlemi olarak verilsin. Bu durumda çaprazlanmış modül aksiyomlarının sağlandığını gösterelim.

ÇM1) $\partial(r\cdot i) = \partial[r,i] = [r,i] = [r,\partial i]$

ÇM2) $\partial(r)\cdot r' = rr' = [r,r']$

olduğundan (I,R,∂) bir çaprazlanmış modül yapısı oluşturur.

Örnek 4.1 M herhangi bir S -bimodül olsun.

$$\begin{array}{ccc} M\times M & \longrightarrow & M \\ (m_1,m_2) & \longmapsto & [m_1,m_2] = 0 \end{array}$$

çarpımı tanımlanırsa, M bir Lie S -cebir yapısı oluşturur. Bu durumda

$$\begin{array}{rccc} 0: & M & \longrightarrow & R \\ & x & \longmapsto & 0(x) = 0 \end{array}$$

şeklinde verilen sıfır morfizminin

$$\begin{array}{ccc} S \times M & \longrightarrow & M \\ (s,m) & \longmapsto & s \cdot m = sm \end{array}$$

etkisi ile bir çaprazlanmış modül yapısı oluşturduğunu gösterelim.

ÇM1) $0(r \cdot m) = 0[r,m] = 0 = [r,0m]$

ÇM2) $\partial(m) \cdot m' = 0 \cdot m' = 0 = [m,m']$

olduğundan $(M,S,0)$ bir çaprazlanmış modül yapısı oluşturur.

Örnek 4.2 M bir Lie S -cebir ve

$$\pi_2 \ : \ M \longrightarrow S$$

ikinci izdüşüm fonksiyonu bir Lie S -cebir morfizmidir. S nin $S \ltimes M$ üzerine Lie etkisi $s' \in S$ ve $(s,m) \in S \ltimes M$ için

$$s'(m,s) = (s' \cdot m, [s',s])$$

şeklinde tanımlansın. Bu durumda

$$\begin{array}{rccc} \pi_2 : & M \times G & \longrightarrow & G \\ & (m,g) & \longmapsto & g \end{array}$$

ve

$$\begin{array}{ccc} G \times (M \times G) & \longrightarrow & M \times G \\ (g',(m,g)) & \longmapsto & g' \cdot (m,g) = (g' \cdot m, [g',g]) \end{array}$$

olmak üzere

ÇM1)

$$\begin{aligned} \pi_2(g' \cdot (m,g)) &= \pi_2(g' \cdot m, [g',g]) \\ &= [g',g] \\ &= [g', \pi_2((m,g))] \end{aligned}$$

olup $(S \ltimes M, S, \pi_2)$ bir ön çaprazlanmış modüldür.

Diğer taraftan

ÇM2)

$$\begin{aligned}\pi_2((m\cdot g'))\cdot(m,g) &= g'\cdot(m,g)\\ &= (g'\cdot m,[g',g])\end{aligned} \quad .....\text{I}$$

ve

$$[(m',g'),(m,g)] = ([m',m],[g',g]).....\text{II}$$

olduğundan I=II olması için

$$(g'\cdot m,[g',g]) = ([m',m],[g',g]) \Longrightarrow g'\cdot m = [m',m]$$

olması gerekirdi. Fakat bu mümkün olmadığından genellikle $(S\ltimes M,S,\pi_2)$ bir çaprazlanmış modül değildir.

Şimdi iki çaprazlanmış modül arasındaki morfizm kavramını tanımlayalım.

Tanım 4.3 (M,S,μ) ve (M',S',μ') iki Lie çaprazlanmış modül olsun.

$$f(s\cdot m) = \phi(s)\cdot f(m)$$

ve

$$\begin{array}{ccc} M & \xrightarrow{\ f\ } & M' \\ {\scriptstyle\mu}\downarrow & & \downarrow{\scriptstyle\mu'} \\ G & \xrightarrow[\ \phi\ ]{} & G' \end{array}$$

diyagramı değişmeli, yani

$$\mu' f(m) = \phi\mu(m)$$

olacak şekilde $f : M \to M', \phi : S \longrightarrow S'$ Lie $\mathbf{K}$ -cebir morfizmleri varsa

$$(f,\phi) : (M,S,\mu) \longrightarrow (M',S',\mu')$$

morfizmine çaprazlanmış modüller arasındaki morfizm denir. Eğer f ve ϕ örtense (f,ϕ) ye örten, f ve ϕ bire bir ise (f,ϕ) ye bire bir denir. f,ϕ izomorfizma ise, yani f ve ϕ bire bir örtense

$$(f,\phi) : (M,S,\mu) \longrightarrow (M',S',\mu')$$

morfizmine izomorfizma denir. Bu durumda

$$(f,\phi)^{-1} = (f^{-1},\phi^{-1}) : (M',S',\mu') \longrightarrow (M,S,\mu)$$

bir çaprazlanmış modül morfizmasıdır ve $(f,\phi)^{-1}(f,\phi) = (Id,Id) = (f,\phi)(f,\phi)^{-1}$ dir. Böylece çaprazlanmış modüllerin kategorisi oluşturulur ve bu kategori $LXMod(k)$ ile gösterilir. Özel olarak, $S = S'$ ve ϕ birim dönüşüm ise, f bir Lie S -cebir morfizması olduğundan

$$f(s \cdot m) = sf(m)$$

olur ve diyagram değişmeli, yani

$$\mu' f(m) = \mu(m)$$

olduğundan, f bir çaprazlanmış Lie S -modül morfizmasıdır. S üzerinde iki çaprazlanmış modülün bileşkesi bir çaprazlanmış Lie S -modül morfizmi olduğundan $LXMod(k)$ nın bir alt kategorisi elde edilir ve bu kategori $LXMod(k)/S$ ile gösterilir. (Casas, 1990)

Örnek 4.3 $(f,Id) : (M,S,\mu) \longrightarrow (M',S,\mu')$ bir çaprazlanmış modül homomorfizması ise (M,M',f) bir çaprazlanmış modüldür. Burada M' nün M ye etkisi μ' yardımıyla her $m' \in M'$ ve $m \in M$ için

$$\begin{array}{ccc} M \times M' & \longrightarrow & M \\ (m',m) & \longmapsto & m' \cdot m = \mu'(m') \cdot m \end{array}$$

olarak verilir. Şimdi (M,M',f) üçlüsünün çaprazlanmış modül olduğunu gösterelim

ÇM1)

$$\begin{aligned} f(m' \cdot m) &= f(\mu'(m') \cdot m) \\ &= \mu'(m') \cdot f(m) \\ &= [m', f(m)] \end{aligned}$$

ÇM2)

$$\begin{aligned} f(m) \cdot m_1 &= \mu'(f(m)) \cdot m_1 \\ &= \mu(m) \cdot m_1 \\ &= [m, m_1] \end{aligned}$$

olur. Ayrıca birim morfizması

$$(Id_M, Id_S) : (M,S,\mu) \longrightarrow (M,S,\mu)$$

şeklindedir.

4.2 Çaprazlanmış Alt Modüller

Genel olarak bir matematiksel yapının alt yapıları, ilgili alanda önemli yer turar. Bir grubun (normal) altgrubu, bir cebirin alt cebiri, ideali, bir topolojik uzayın alt uzayı gibi. Benzer olarak bir çaprazlanmış modülün alt çaprazlanmış modülü ve ideali gibi kavramlar da çaprazlanmış modüllerle ilgili çalışmalarda önem kazanır. N.M.Shammu (Shammu, 1992) tezinde $LXMod(k)/S$ kategorisinde bu kavramlara yer vermiştir.

Tanım 4.4 (M,S,μ) bir çaprazlanmış modül olsun. M', M nin ve S',S nin bir alt Lie cebiri olmak üzere

$$\mu' : \mu \mid_M : M' \longrightarrow S'$$

μ nin M' ye kısıtlanmışı ve S' nün M' ne etkisi S nin M üzerine etkisinin kısıtlanmışı olmak üzere (M',S',μ') çaprazlanmış modülüne (M,S,μ) çaprazlanmış modülünün alt çaprazlanmış modülü denir ve $(M',S',\mu') \leq (M,S,\mu)$ şeklinde gösterilir. (Casas, 1990)

Örnek 4.4 A,S Lie cebirinin bir alt Lie cebiri olsun. Bu durumda (A,A,Id_A), $(0,A,i)$, (S,S,Id_S) ve $(0,S,i)$ birer çaprazlanmış modüldür. Ayrıca (A,A,Id_A), (S,S,Id_S) nin bir alt çaprazlanmış modülü ve $(0,A,i)$ de $(0,S,i)$ nin alt çaprazlanmış modülüdür. (Casas, 1990)

Örnek 4.5 I,S Lie cebirinin herhangi bir ideali olmak üzere (I,S,i), (S,S,Id) çaprazlanmış modülünün alt çaprazlanmış modülünü oluşturur. (Casas, 1990)

Örnek 4.6 A ve B,S nin ideali $B \subseteq A$ olmak üzere (B,A,i), (A,S,i) çaprazlanmış modülünün alt çaprazlanmış modülü oluşturur. (Casas, 1990)

Örnek 4.7 A, S -modül, B,A içinde S -alt modül olmak üzere $(B,S,0)$, $(A,S,0)$ çaprazlanmış modülünün alt çaprazlanmış modülünü oluşturur. (Casas, 1990)

4.3 Çaprazlanmış İdeal

Tanım 4.5 (M,S,μ) çaprazlanmış modülünün (M',S',μ') alt çaprazlanmış modülü olmak üzere

i. S',S cebirinin bir idealidir, yani $S' \trianglelefteq S$

ii. Her $s \in S$ ve $m' \in M'$ için $s \cdot m' \in M'$

iii. Her $s' \in S'$ ve $m \in M$ için $s' \cdot m \in M$

şartları sağlanıyorsa (M',S',μ') alt çaprazlanmış modülüne (M,S,μ) çaprazlanmış modülünün ideali denir ve $(M',S',\mu') \trianglelefteq (M,S,\mu)$ şeklinde gösterilir. Eğer $(M',S',\mu') \trianglelefteq (M,S,\mu)$ ise her $m \in M$ ve $m' \in M'$ için

$$[m,m'] = \mu(m) \cdot m' \in M'$$

olduğundan M',M nin bir idealidir. (Casas, 1990)

Örnek 4.8 I, S Lie cebirinin bir ideali olmak üzere (I,I,Id), (S,S,Id) nin ve $(0,I,i)$ de $(0,S,i)$ nin birer çaprazlanmış idealidir. (Casas, 1990)

Önerme 4.6 $(M',S',\mu') \leq (M'',S'',\mu'') \leq (M,S,\mu)$ şeklinde alt çaprazlanmış modüller için, (M',S',μ'), (M,S,μ) nün ideali ise (M',S',μ'), (M'',S'',μ'') nün idealidir. (Casas, 1990)

Proof. i) $s \in S$, $s' \in S'$, $s'' \in S''$, $m \in M$ ve $m' \in M'$ için, $(M',S',\mu') \trianglelefteq (M,S,\mu)$ olduğundan $S' \trianglelefteq S$ olur ve $S' \leq S'' \leq S$ olduğundan $[s',s''] \in S'$ olup $S' \trianglelefteq S''$ bulunur.

ii) $(M',S',\mu') \trianglelefteq (M,S,\mu)$ olduğundan $s \cdot m' \in M'$ olur ve $S'' \trianglelefteq S$ olduğundan $s'' \cdot m' \in M'$ sağlanır.

iii) $s' \cdot m \in M$ ve $M' \leq M'' \leq M$ olduğundan $s \cdot m'' \in M'$ olur. Böylece $(M',S',\mu') \trianglelefteq (M'',S'',\mu'')$ elde edilir. (Casas, 1990) ■

4.4 Çaprazlanmış Modüllerin Çekirdeği ve Görüntüsü

Tanım 4.7 $(f,\phi) : (M,S,\mu) \longrightarrow (M',S',\mu')$ bir çaprazlanmış modül morfizmi olmak üzere

$$\mu : Kerf \longrightarrow Ker\phi$$

çaprazlanmış modülüne (f,ϕ) morfizminin çekirdeği denir ve $Ker(f,\phi)$ ile gösterilir. (Casas, 1990)

Yardımcı Teorem 4.8 $Ker(f,\phi) = (Ker\ f, Ker\ \phi, \mu)$ çaprazlanmış modülü, (M,S,μ) nin bir idealidir. (Casas, 1990)

Proof. $s \in S$, $s' \in Ker\ \phi$ için

$$\phi([s,s_1]) = [\phi(s),\phi(s_1)] = [\phi(s),0] = 0$$

olduğundan $[s,s_1] \in Ker\ \phi$ olur. Böylece $Ker\ \phi$, S nin idealidir. Ayrıca $s \in S$, $m_1 \in Ker\ f$ için

$$f(s \cdot m_1) = \phi(s) \cdot f(m_1) = \phi(s) \cdot 0 = 0$$

olduğundan $s \cdot m_1 \in Ker\ f$ olur. $s_1 \in Ker\ \phi$, $m \in M$ için

$$f(s_1 \cdot m) = \phi(s_1) \cdot f(m) = 0 \cdot f(m) = 0$$

olduğundan $s_1 \cdot m \in Ker\ f$ elde edilir. (Casas, 1990) ■

Tanım 4.9 $(f,\phi) : (M,S,\mu) \longrightarrow (M',S',\mu')$ bir çaprazlanmış modül morfizmi olmak üzere

$$\mu' : \mathrm{Im}\, f \longrightarrow \mathrm{Im}\,\phi$$

çaprazlanmış modülüne (f,ϕ) morfizminin görüntüsü denir ve $\mathrm{Im}(f,\phi)$ ile gösterilir. (Casas, 1990)

Yardımcı Teorem 4.10 $(f,\phi) : (M,S,\mu) \longrightarrow (M',S',\mu')$ bir morfizm olsun. Bu durumda $\mathrm{Im}(f,\phi)$, (M',S',μ') nün bir alt çaprazlanmış modülüdür. (Casas, 1990)

Tanım 4.11 (M,S,μ) çaprazlanmış modül olmak üzere S yardımıyla M nin sabit elemanlarının oluşturmuş olduğu küme

$$M^S = \{m \in M \mid hers \in S \text{ için } s \cdot m = 0\}$$

şeklinde tanımlanır ve M^S, $Z(M)$ (M nin merkezi) nin idealidir. (Casas, 1990)

Tanım 4.12 S bir Lie $\mathbf{K}$ -cebir ve $d : S \longrightarrow M$,$\mathbf{K}$ -lineer dönüşümler olsun. Her $s,s' \in S$ için,

$$d[s,s'] = sd(s') - s'd(s)$$

özelliği sağlanıyorsa d ye bir derivasyon denir. S den M ye bütün derivasyonların kümesi $Der(S)$ ile gösterilir. (Casas, 1990)

Tanım 4.13 (M,S,μ) çaprazlanmış modül olmak üzere

i) Eğer S nin M üzerine faithful etkisi varsa, yani $st_S(M) = 0$ ise (M,S,μ) ye bağlı (faithful) çaprazlanmış modül denir.

ii) μ bire bir, yani $Ker\mu = 0$ ise (M,S,μ) ye asferikal çaprazlanmış modül denir. μ örten, yani $Co\ker\mu = 0$ ise (M,S,μ) ye basit bağlantılı çaprazlanmış modül denir. (Casas, 1990)

4.5 GAP Uygulaması

Lie cebirleri üzerindeki çaprazlanmış modüllerin tüm operasyonlar tez kapsamında yazılan `XModLieAlg` operasyonuna bağlanmıştır. Bir Lie cebiri çaprazlanmış modül oluşturmak için eldeki bilgileri bir veri haline getirecek olan `XModAlgLieObj` objesi oluşturulmuş ve diğer tüm fonksiyonlar bu objeye bağlanmıştır.

Kaynak Kodu

```
#############################################################################
###
##M  XModLieAlgObj( <bdy>, <act> ) . .  make pre-crossed module of Lie Algebras
###
InstallMethod( XModLieAlgObj, "for homomorphism and action", true,
    [ IsAlgebraHomomorphism, IsLieAlgebraAction ], 0,
function( bdy, act )

    local  A,B,filter,fam,PM,AB,BB;
    fam := FamilyObj( [ bdy, act ] );
    filter := IsPreXModLieAlgObj;
    B:= Source(bdy);
    A:= Range(bdy);
    AB:=Source(act);
    BB:=Range(act);
    if not ( B = BB ) then
        return false;
    fi;
    PM := Objectify( NewType( fam, filter ), rec() );
    SetSource( PM, B );
    SetRange( PM, A );
    SetBoundary( PM, bdy );
    SetXModLieAlgAction( PM, act );
    SetIs2dLieAlgObject( PM, true );
    return PM;
end );
```

Oluşturulan objenin Lie (ön)çaprazlanmış modül olup olmadığını denetlemek için aşağıdaki fonksiyonlar yazılmıştır.

◊ `IsPreXModLieAlg( L )` (operasyon)

◊ `IsXModLieAlg( L )` (operasyon)

`GroupRing` derlemesi ile oluştuşturulmuş cebirsel yapının bir cebir olup olmadığını denetlemek için `IsGroupAlgebra` derlemesi kullanılır. Yapı bir cebir ise derleme `true` yanıtını verir.

◊ `XModLieAlg( KG )` (özellik)

Bu operasyon yardımıyla Lie cebir çaprazlanmış modül oluşturmak için verilen tüm fonksiyonlar birarada tutularak, kullanıcının daha az operasyonla işlem yapmasına olanak sağlar.

◊ Source(LC) (nitelik)

◊ Range(LC) (nitelik)

◊ Tail(LC) (nitelik)

◊ Head(LC) (nitelik)

◊ Kernel(LC) (nitelik)

◊ Boundary(LC) (nitelik)

GAP

```
gap> KG:=GroupRing(GF(2),DihedralGroup(8));
<algebra-with-one over GF(2), with 3 generators>
gap> L:=LieAlgebra(KG);
#I  LAGUNA package: Constructing Lie algebra ...
<Lie algebra of dimension 8 over GF(2)>
gap> D:=LieDerivedSubalgebra(L);
<Lie algebra of dimension 3 over GF(2)>
gap> IsIdeal(L,D);
true
gap> LX:=XModLieAlg(L,D);
[Algebra( GF(2), [ LieObject( (Z(2)^0)*f1+(Z(2)^0)*f1*f3 ),
  LieObject( (Z(2)^0)*f2+(Z(2)^0)*f2*f3 ), LieObject( (Z(2)^0)*f1*f2+(Z(2)^
    0)*f1*f2*f3 ) ] )->Algebra( GF(2),
[ LieObject( (Z(2)^0)*<identity> of ... ), LieObject( (Z(2)^0)*f1 ),
  LieObject( (Z(2)^0)*f2 ), LieObject( (Z(2)^0)*f3 ),
  LieObject( (Z(2)^0)*f1*f2 ), LieObject( (Z(2)^0)*f1*f3 ),
  LieObject( (Z(2)^0)*f2*f3 ), LieObject( (Z(2)^0)*f1*f2*f3 ) ] )]
gap> IsXModLieAlg(LX);
true
gap> Display(LX);
Crossed module [..->..] :-
: Source lie algebra has generators:
  [ LieObject( (Z(2)^0)*f1+(Z(2)^0)*f1*f3 ),
  LieObject( (Z(2)^0)*f2+(Z(2)^0)*f2*f3 ), LieObject( (Z(2)^0)*f1*f2+(Z(2)^
    0)*f1*f2*f3 ) ]
: Range lie algebra has generators:
  [ LieObject( (Z(2)^0)*<identity> of ... ), LieObject( (Z(2)^0)*f1 ),
  LieObject( (Z(2)^0)*f2 ), LieObject( (Z(2)^0)*f3 ),
  LieObject( (Z(2)^0)*f1*f2 ), LieObject( (Z(2)^0)*f1*f3 ),
  LieObject( (Z(2)^0)*f2*f3 ), LieObject( (Z(2)^0)*f1*f2*f3 ) ]
: Boundary homomorphism maps source generators to:
  [ LieObject( (Z(2)^0)*f1+(Z(2)^0)*f1*f3 ),
  LieObject( (Z(2)^0)*f2+(Z(2)^0)*f2*f3 ), LieObject( (Z(2)^0)*f1*f2+(Z(2)^
    0)*f1*f2*f3 ) ]
gap> Size(LX);
[ 8, 256 ]
gap> Boundary(LX);
MappingByFunction( <Lie algebra of dimension 3 over GF(
2)>, <Lie algebra of dimension 8 over GF(2)>, function( i ) ... end )
```

Aşağıda yazılan operasyonların hepsi grup cebirler üzerinde Lie cebiri oluşturmak için kullanılırlar.

◊ `PreCat1LieAlgByTailHeadEmbedding(t,h,e)` (operasyon)

◊ `PreCat1LieAlgByEndomorphisms(t,h)` (operasyon)

◊ `PreCat1LieAlgObj(LC)` (operasyon)

◊ `IsIdentityCat1LieAlg(LC)` (operasyon)

◊ `IsCat1LieAlg(LC)` (operasyon)

◊ `IsPreCat1LieAlg(LC)` (operasyon)

KAYNAKLAR DİZİNİ

Alp, M. and Wensley, 2000, Crossed Modules and Cat^1-groups in GAP, version 2.1 Manual for the XMOD share package.

Amoya, R. , Stewart, I., 1974, Infinite-dimensional lie algebras, Nordhoff.

Arvasi, Z., Porter T., 1996, Simplicial and Crossed Resolutions of Commutative Algebras, Journal of Algebras,181, 426-448.

Arvasi, Z., Ege, U., 2003, Annihilators, Multipliers and Crossed Modules, Applied Categorical Structures, Vol.11, 487-506.

Aytekin, A. , 2005, Lie Cebirlerin Çaprazlanmış Modülleri, Yüksek Lisans Tezi, Dumlupınar Üniversitesi.

Barker, M., 2003, Representations of Crossed Modules and Cat^1-Groups, Ph.D. Thesis, University of Wales, Bangor.

Bovdi, V., Konovalov, A., Rossmanith, R. , SchneiderBarker, C., 2009, LAGUNA : Lie AlGebras and UNits of group Algebras, GAP Share Package.

Brown, R., Wensley, Ch. D., 1995, Christopher. On finite induced crossed modules, and the homotopy 2-type of mapping cones, Theory Appl. Categ. 1 , No. 3, 54-70.

Brown, R., Wensley, Ch. D., 1996, Computing Crossed Modules Induced by an Inclusion of a Normal Subgroup, with Applications to Homotopy 2-types, Theory Appl. Categ., 2, 1, 3-16.

Brown, R., Wensley, Ch. D., 2003, Computation and homotopical applications of induced crossed modules, . J. Symbolic Comput. 35 (2003), no. 1, 59-72.

Casas, J.M., 1990, Invariantes de Módulos Cruzados en Álgebras de Lie, Ph.D.Thesis, University of Santiago.

Cayley, A., 1854, On the theory of groups as depending on the symbolic equation $\theta^n = 1$, Phil. Mag. 7, 40-47.

Connel, I., 1963, On the group ring, Can. J. Math. 15, 650-685.

Ellis, G.J., 1988, Higher Dimensional Crossed Modules of Algebras, J.Pure Appl. Algebra 52 , 277-282.

Ege, U., 2002, Çarpım Cebiri ve Çaprazlanmış Modüller, Doktora Tezi, Osmangazi Universitesi.

Erdmann, K. , Wildon, M. J. , Introduction to Lie Algebras, 2006, Oxford University, UK.

GAP:, GAP - Groups, Algortihms, and Programming, Version 4, 2008, Lehrstuhl D für Mathematik, RWTH Aachen Germany and School of Mathematical and Computational Sciences, U. St. Andrews, Scotland.

Gerstenhaber, M., 1966, On the Deformation of Rings and Algebras, Annals of Math. Soc., 84, 1-19.

Kassel, C., Loday, J.L. 1982, Extensions centrales d'algèbres de Lie, Ann. Inst. Fourier (Grenoble) 33, 119-142.

Bovdi, V., Konovalov, A., Rossmanith, R., Schneider, C., 2003, LAGUNA - Lie AlGebras and UNits of group Algebras, GAP share package.

Milies, C. P., Sehgal, S. K, 2002, An Introduction to Group Rings Kluwer Academic Publishers.

Norrie, K.J., 1987, Crossed Modules and Analogues of Group Teorems, Ph.D.Thesis, King's College.

Odabaş, A. , GAP (Grup, Algoritma, Programlama) ile Cebirler ve Sayılar Teorisi, 2004, Yüksek Lisans Tezi, Dumlupınar Üniversitesi.

Odabaş, A. , GAP (Grup, Algoritma, Programlama) ile Cebirler Üzerinde Çaprazlanmış Modüller, 2009, Doktora Tezi, Eskişehir Osmangazi Üniversitesi.

Passman, D. S. , 1977, The Algebraic Structure Of Group Rings, Wiley, New York.

Porter, T., 1978, Some Categorical Results in the Category of Crossed Modules in Commutative Algebra, J. Algebra, 109, 415-429.

Porter, T., 1986, Homology of Commutative Algebras and an Invariant of Simis and Vasconceles, J. Algebra, 99, 458-465.

Shammu, N.M., 1992, Algebraic and an Categorical Structure of Category of Crossed Modules of Algebras, Ph.D. Thesis,U.C.N.W.

Whitehead, J. H. C., 1949, Combinatorial Homotopy I, Bull. Amer. Math. Soc., 55, 453-496.

ÖZGEÇMİŞ

Ahmet Faruk ASLAN 6 Ocak 1984 arihinde İstanbul'da doğmuştur. Lisans öğrenimine 2002 yılında Eskişehir Osmangazi Üniversitesi Fen Edebiyat Fakültesi Matematik Bölümünde başlamış ve 2006 yılında mezun olmuştur. İş hayatına 2007 yılında Eskişehir Osmangazi Üniversitesi, Fen Edebiyat Fakültesi Matematik Bölümünde araştırma görevlisi olarak başlamıştır ve halen aynı bölümde akademik çalışmalarına devam etmektedir.

Eskişehir Osmangazi
Üniversitesi Fen-Edebiyat
Fakültesi Matematik ve
Bilgisayar Bilimleri Bölümü
ESKİŞEHİR