QUANTUM THEORY PROJECT
FOR RESEARCH IN ATOMIC, MOLECULAR, AND SOLID STATE
CHEMISTRY AND PHYSICS
UNIVERSITY OF FLORIDA, GAINESVILLE, FLORIDA

GROUP ANALYSIS PROGRAMS

By

Robert Yates*

PROGRAM NOTE # 5

1 July 1963

*Student, Johns Hopkins University

ABSTRACT

A complete series of LISP functions is described, whereby the commonly recognized properties of finite groups may be analyzed. The analysis includes the determination of the subgroups of the group, its normal subgroups, the H-classes, cosets and double cosets, commutator subgroup, center, and other specialized subgroups. Factor groups, direct and semidirect products can be constructed and analyzed in terms of their constituents. The information required for the analysis is a definition of the group multiplication, either in terms of a group table prepared in a certain form or a LISP function yielding the product of any two group elements.

ACKNOWLEDGMENTS

The LISP functions herein described have gradually evolved over a period of years. The first tentative functions were constructed by Tony Conrad in the summer of 1960 at the Institute for Physical Chemistry at the University of Copenhagen. A SOS program for the IBM 7090 was prepared by Joe Schlessinger in the spring of 1961 which calculated the subgroups, classes and normal subgroups given the group table of groups of orders less than 40. On the basis of this experience a systematic effort to use LISP to analyze all the generally recognized aspects of the structure of finite groups was commenced in the summer of 1962 at a Summer Institute directed by Harold V. McIntosh at the University of Florida. We are grateful to Dean R. B. Mautz and Professor Harry Sisler for their support and encouragement of this Institute and to the Computer Center of the University of Florida for making available the time to verify the functions. Some additional adjustments were made while a guest of the Yale Computer Center in the spring of 1963. We are greatly indebted to Lowell Hawkinson and to Dr. Morris S. Davis, the director of the center for their hospitality. The final preparation of this report has been made at the University of Florida, where the efforts of Professor Billy S. Thomas have been invaluable in testing and applying the functions, as well as contributing to the style of their final form. It is a pleasure to acknowledge the hospitality, interest, and support extended by Dr. Darwin Smith on behalf of the Quantum Theory Project, Professor S. S. Ballard, head

of the Physics Department, Professor H. S. Sisler, head of the Chemistry Department, and Professor J. E. Maxfield, director of the University Computer Center. The programs have been written in MBLISP, which was developed at MARTIN BALTIMORE with the support and interest of Mr. Welcome W. Bender, the director of RIAS, and the wholehearted cooperation of the Digital Computations Section, of which we particularly acknowledge the assistance of Solis James, the director, the operating staff of the IBM 7090, who consistently expedited the checking and running of the programs, and the programming staff who willingly gave advice on the Baltimore system.

Robert Yates

Gainesville, 1 July 1963

GROUP ANALYSIS PROGRAMS

The analysis of the properties of a finite group is a challenging task which can conveniently be performed by symbolic manipulation programs written for an electronic computer. The determination of these properties, such as the enumeration of the subgroups, the calculation of the classes or of the cosets of a subgroup, can be done with a knowledge of the group multiplication, usually gained from the consultation of a table of possible products. Since the calculations required are not numerical in nature, but rather logical, involving the searching of lists, comparison of symbols, and so on, the operations can be performed by a programming language such as LISP, especially designed for this purpose.

By a group we mean a set G, together with a multiplication satisfying the following axioms:

1. The set is closed under the multiplication.
2. The multiplication is associative.
3. There exists an identity element.
4. Each element has an inverse.

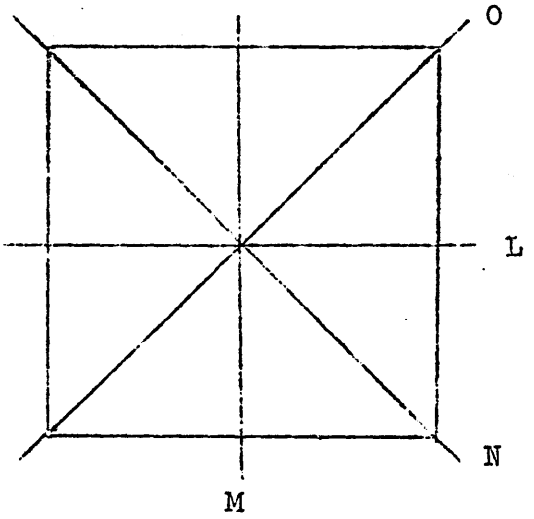
The properties of a finite group are thoroughly discussed in many textbooks on modern algebra to which the reader may refer for the definitions and concepts used in this paper.

Since the structure of an abstract group is determined solely by its multiplication, for purposes of calculation we have only to define this multiplication, representing the group elements by any convenient symbols. For finite groups this can be done in a table, usually of the following form:

	e a b ... q ... n
e	
a	
b	
⋮	
p	. . . . pq . . . .
⋮	
n	

Here the group elements are listed across the top commencing with the identity and in the same order down the left hand side. The element in the p^{th} row and q^{th} column corresponds to the product pq; we shall generally denote this as the standard form of a group table.

One example of a relatively simple finite group to which we shall refer throughout this paper is the group of symmetries of a square, D^4, which is the dihedral group of degree four. This group and its multiplication table are given below.



E = identity

A = 90° counterclockwise rotation

A2 = 180° counterclockwise rotation

A3 = 270° counterclockwise rotation

R = reflection through axis L

RA = reflection through axis N

RA2 = reflection through axis M

RA3 = reflection through axis O

By the product of two symmetries, ab, we mean the symmetry obtained by applying b first, and then a. The group table would then be:

	E	A	A2	A3	R	RA	RA2	RA3
E	E	A	A2	A3	R	RA	RA2	RA3
A	A	A2	A3	E	RA3	R	RA	RA2
A2	A2	A3	E	A	RA2	RA3	R	RA
A3	A3	E	A	A2	RA	RA2	RA3	R
R	R	RA	RA2	RA3	E	A	A2	A3
RA	RA	RA2	RA3	R	A3	E	A	A2
RA2	RA2	RA3	R	RA	A2	A3	E	A
RA3	RA3	R	RA	RA2	A	A2	A3	E

To represent such a function in LISP, it is essential to have a rapid and convenient access to the product qp for any group elements; that is, an access to the p^{th} row and q^{th} element in that row. We find that such a representation is most easily accomplished by the use of alternating lists. By an alternating list we mean a list L = (A1 B1 A2 B2 ... AN BN) of even length, serving as a dictionary. The odd elements AI serve as references, and the even elements BI are their equivalents. To use such a list, a LISP function (ASSOC X L) is defined to search the odd elements A1, A2, ... in turn until the desired element AI is found; its successor BI is then taken as the value.

The definition of ASSOC is as follows:

```
(ASSOC (LAMBDA (X L) (IF (EQ X (CAR L)) (CADR L) (ASSOC X (CDDR L)))))
```

This technique can be employed twice to represent a group table; one uses the alternating lists to pair the rows of the table with the corresponding row headings, while the rows are represented by a list in which the column headings alternate with the products themselves.

Thus if the group G has the elements (A1 A2 ... AN), the group table representation can then be retained as a LISP function of no variables:

```
(GROUPNAME (LAMBDA () (QUOTE
    (A1 (A1 A1A1 A2 A1A2 ... AN A1AN)
     A2 (A1 A2A1 A2 A2A2 ... AN A2AN)
     . . .
     AN (A1 ANA1 A2 ANA2 ... AN ANAN)
```

where the symbol AIAJ is understood to mean the product of AI and AJ. In this format the group D4 would be represented as follows:

```
(D4 (LAMBDA ()(QUOTE
    (E (E E A A A2 A2 A3 A3 R R RA RA RA2 RA2 RA3 RA3)
     A (E A A A2 A2 A3 A3 E R RA3 RA R RA2 RA RA3 RA2)
    A2 (E A2 A A3 A2 E A3 A R RA2 RA RA3 RA2 R RA3 RA)
    A3 (E A3 A E A2 A A3 A2 R RA RA RA2 RA2 RA3 RA3 R)
     R (E R A RA A2 RA2 A3 RA3 R E RA A RA2 A2 RA3 A3)
    RA (E RA A RA2 A2 RA3 A3 R R A3 RA # RA2 A RA3 A2)
   RA2 (E RA2 A RA3 A2 R A3 RA R A2 RA A3 RA2 E RA3 A)
   RA3 (E RA3 A R A2 RA A3 RA2 R A RA A2 RA2 A3 RA3 E)))))
```

We may then define a function (GP A B) to calculate the product A B of two group elements:

```
(GP (LAMBDA (A B) (ASSOC B (ASSOC A T))))
```

The free variable T in the above expression refers to the name of the particular group table, and is not bound until a higher level, usually not until the function is actually executed.

To find the inverse of a group element AI from this table format, we need the element in the AI^{th} row which when multipled by A gives the unit element. In our alternating list structure, L = (A1 B1 ...), where BJ = AIAJ. We then need to search the BJ's for the identity and select the corresponding AJ; this is done by the function (ASSOC* X L): (ASSOC* (LAMBDA (X L) (IF (EQ X (CADR X)) (CAR X) (ASSOC* X (CDDR L))))). Letting the identity element be represented by a specific function, (UNIT), our definition of the inverse of an element X, (A-1 X) would be as follows:

```
(A-1 (LAMBDA (X) (ASSOC* (UNIT (ASSOC X T))))
```

Finally, in the calculation of various group properties, we often need to have a list of the group elements, to which end we define a function, (GELEMENTS), to provide such a list. It may be extracted from the group table as follows:

```
(GELEMENTS (LAMBDA () (GELEMENTS* T)))
(GELEMENTS* (LAMBDA (L)
    (IF (NULL L) L (CONS (CAR L) (GELEMENTS* (CDDR L))))))
```

To enable the program to be as flexible as possible, we must allow it to accept all types of groups--where the multiplication may not be given in a single table, but perhaps in a combination of tables, or even as a rule. We therefore choose to base the entire program on a foundation of the four functions GP, A-1, UNIT, and GELEMENTS. For each means of describing a group we require that only these four functions be redefined for the particular purpose.

This idea becomes especially useful when we desire to represent the direct or semi-direct product of two groups in LISP. All we essentially need to do is define a multiplication for ordered pairs (A B), and the same program may be used intact for this new group.

From the central function GP, various other types of group products can be defined. The function (MULTIGP A1 A2 ... AN) will calculate the product of an arbitrary number of group elements, not being restricted to simply two. The functions (LGP X L) and (RGP X L) calculate respectively the left and right translations xL and Lx, of the complex L by the element x, while the function (COMPLEXGP L M) calculates the complex product of the subsets L and M. (The definitions and some examples of these functions may be found in the appendix.)

One can analyze the structure of a group according to three main classifications; its substructure, factor structure, and product structure.

The first of these, the substructure, deals mainly with the study of the subgroups of a group and the internal group structure within the group itself. The problems then deal with such calculations as listing all of the subgroups of a group, or of computing particular subgroups such as the normal subgroups, the commutator subgroup, the center of the group, and perhaps the normalizer and stabilizers of certain elements and complexes.

One of the primary functions necessary in this area is one to calculate the hull of a complex, C, which is the smallest subgroup

containing the complex, denoted by ((C)). Since a necessary and sufficient condition that a complex C of a finite group be a subgroup is that it be closed under multiplication, we have that

$$((C)) = C \Leftrightarrow C^2 \subset C \ .$$

An explicit way to calculate the hull would be to square the complex, testing to see whether or not the result was the same set. If not, we adjoin all of the new elements so obtained, and again compute all possible products. Since the group is finite, and the supply of additional elements available for adjunction is limited, we must eventually arrive at the desired subgroup.

The calculation involved in this "squaring" of the complex can be materially simplified by eliminating some of the redundant products. If C is the complex, and if C' is the new set of elements obtained upon squaring C, then when we again square the resulting complex $C \cup C'$, there is no need to again recalculate the products in C^2. Consequently,

$$(C \cup C')^2 = (C \cup C') \cup [(C'C \cup CC' \cup C^2].$$

The LISP function (HULL C), which calculates the hull of the complex C, works on exactly this principle. For an example we calculate the hull of the elements (RA2 R), yielding the subgroup (E A2 RA2 R).

```
(APPLY (LAMBDA (X) ((LAMBDA (T) (HULL X)) (D4))) ((RA2 R)))
(A2 E RA2 R)
```

With the use of this function we can proceed to a calculation of all the subgroups of a group.

Every subgroup is a union of certain of its cyclic subgroups. This

can be seen by considering the hull of one of the elements of the subgroup, that is, the cyclic subgroup generated by this element. If this hull yields the entire subgroup, then the subgroup itself was cyclic and we are through. Otherwise,pick an element not in this hull and proceed in the same fashion. Since every subgroup is so generated, we can first calculate all of the cyclic subgroups of the group, and consider only their distinct unions. By testing in each case as to whether or not a particular union actually forms a subgroup, we may in this fashion obtain all the possible subgroups. The LISP function (SUBGROUPS) is based on this idea; the subgroups of D^4 are computed as an example:

```
(APPLY (LAMBDA () ((LAMBDA (T) (SUBGROUPS)) (D4))) ())
((E) (RA3 E) (RA2 E) (RA E) (RA3 RA A2 E) (R E) (RA3 RA2 RA R A A2 A3 E)
(RA2 R A2 E) (A2 E) (A A2 A3 E))
```

In addition to HULL, two subsidiary functions are used in the definition of SUBGROUPS, (CYCLE X), and (CYCLICGROUPS). The first of these calculates the cyclic subgroup generated by the elements X, while the second yields all the distinct cyclic subgroups of the group. For example, in D^4, we calculate the cycle generated by the element A, and the cyclic subgroups of D^4.

```
(APPLY (LAMBDA (X) ((LAMBDA (T) (CYCLE X)  (D4))) (A))
(A A2 A3 E)

(APPLY (LAMBDA () ((LAMBDA (T) (CYCLICSGROUPS)) (D4))) ())
((E) (RA3 E) (RA2 E) (RA E) (R E) (A2 E) (A A2 A3 E))
```

Another important problem is that of calculating the invariant, or normal subgroups of the group. By its defining property a normal subgroup is a subgroup stable under conjugation by all the elements

of the group. Consequently, it must be a union of certain of the classes of the group, since the classes are the only complexes stable under conjugation. Moreover, two elements sit in the same class if and only if their inverses do, and if a particular union of classes is to be a subgroup, the inverse classes must be present as well. Therefore to calculate the normal subgroups we first adjoin each class to its inverse class and consider all possible unions of the resulting set. By testing to see if each individual such union is a subgroup, and discarding those which are not, we have a method to give all the possible normal subgroups of the group. The LISP function (NSUBGROUPS) performs this calculation, giving a list of the normal subgroups. For an example we calculate the normal subgroups of D4.

```
(APPLY (LAMBDA () ((LAMBDA (T) (NSUBGROUPS)) (D4))) ())
((E) (E A2 A A3) (E A2) (E R RA2 A2) (E RA RA3 A2) (E RA RA3 R RA2 A2
                A A3))
```

Finally the functions (CENTER, (COMSGROUP), (NORMALIZER X), (STABILIZER C), and (CENTRALIZER C) calculate the center of a group, the commutator subgroup, the normalizer of an element, and the stabilizer and centralizer of any complex respectively. By the stabilizer of C we mean the subgroup of all those elements in the group which conjugate the complex into itself, while the centralizer of C is the subgroup whose elements commute with all the elements of C. The following are examples of these functions.

```
(APPLY (LAMBDA () ((LAMBDA (T) (CENTER)) (D4))) ())
(E A2)

(APPLY (LAMBDA () ((LAMBDA (T) (COMSGROUP)) (D4))) ())
(E A2)
```

```
(APPLY (LAMBDA (X) ((LAMBDA (T) (NORMALIZER X)) (D4))) (RA))
(E A2 RA RA3)
(APPLY (LAMBDA (C) ((LAMBDA (T) (STABILIZER C)) (D4))) ((A A2)))
(E A A2 A3)

(APPLY (LAMBDA (X) ((LAMBDA (T) (CENTRALIZER X)) (D4))) ((R A2)))
(E A2 R RA2)
```

The second main classification of the structure of a group is according to its factor structure. By this we mean the possible equivalence relations definable on a group such as the cosets of a subgroup or the classes and systems derivable from the equivalence structure, such as factor-groups. Since an equivalence relation on a group partitions the group into disjoint fibers, after calculating the elements in the first fiber, we need consider only those elements of the group not in that fiber for the next calculation, and so on until the group elements are exhausted. The principal function used in the calcuation of such equivalence classes is the function (EQRELATION R L) which is defined:

```
(EQRELATION (LAMBDA (R L) (IF (NULL L) L (( LAMBDA (W)
    (CONS W (EQRELATION R (ERASE W L)))) (R (CAR L))))))
```

In this case, R is a function of one variable which yields all the elements equivalent to a given element. Using this function we can define the function (LCOSETS S) to calculate the left cosets of the subgroup S,

```
(LCOSETS (LAMBDA (S) (LCOSETS* (GELEMENTS))))

(LCOSETS* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X)
          (LGP X S))) G)))
```

and in an identical way, the function (RCOSETS S) to calculate the right cosets of S. As an example, we compute the left and right cosets of the subgroup (E RA) of D4.

```
(APPLY (LAMBDA (X) ((LAMBDA (T) (LCOSETS X)) (D4))) ((E RA)))
((E RA)(A R) (A2 RA3) (A3 RA2))

(APPLY (LAMBDA (X) ((LAMBDA (T) (RCOSETS X)) (D4))) ((E RA)))
((E RA)(A RA2) (A2 RA3)  (A3 R))
```

The double cosets of two subgroups H and K are the complexes of the form HaK. They are the fibers of an equivalence relation so that we may in similar fashion define the function (DBLCOSETS H K) to compute the double cosets.

```
(DBLCOSETS (LAMBDA (H K) (DBLCOSETS* (GELEMENTS))))

(DBLCOSETS* (LAMBDA (G)

    (EQRELATION (FUNCTION (LAMBDA (X) (COMPLEXGP  H (LGP X K)))) G)))
```

Again, for an example, we calculate the double cosets of the subgroups (E RA) and (E RA2) of D^4.

```
(APPLY (LAMBDA (X Y) ((LAMBDA (T) (DBLCOSETS X Y)) (D4)))
((E RA)(E RA2)))

((E RA2 RA A) (A2 R RA3 A3))
```

One of the equivalence relations of greatest importance is that defining the classes of a group. In order to find all the classes, we need to be able to compute the class of a given element. To do this effectively we make use of the theorem that two elements p and q conjugate an element a into the same element if and only if they lie in the same right coset of the normalizer of a. That is,

$$p^{-1}ap = q^{-1}aq \Leftrightarrow qp^{-1}a = aqp^{-1}$$

$$\Leftrightarrow qp^{-1} \in N_a \Leftrightarrow q \in N_a p$$

To find the equivalence class of an element, or the distinct conjugates of that element, we need only conjugate the element by one

member in each right coset of the normalizer of that element. Consequently, again using the function EQRELATION, we define the LISP function (CLASSES), to calculate the classes of the group.

```
(CLASSES (LAMBDA () (CLASSES* (GELEMENTS))))
(CLASSES* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X)
    (CONJUGATES X (XSECTION (RCOSETS (NORMALIZER X)))))) G)))
```

Each class is thus obtained by conjugating the element X by each of the elements in a cross section of the right cosets of the normalizer of X. Using this function we compute the classes of D_4:

```
(APPLY (LAMBDA () ((LAMBDA (T) (CLASSES)) (D4))) ())
((E) (A A3) (A2) (R RA2) (RA RA3))
```

Analogously, with the function (HCLASSES H) we can calculate the H-classes of the group. These are the classes in which the conjugating element comes from a particular subgroup H. In this case we obtain distinct conjugates of an element a by selecting the conjugating elements from a cross section of the right cosets of $N_a \cap H$, in H. As an example we calculate the H-classes of the subgroup (E A A2 A3).

```
(APPLY (LAMBDA (X) ((LAMBDA (T) (HCLASSES X)) (D4))) ((E A A2 A3)))
((E) (A) (A2) (A3) (R RA2) (RA RA3))
```

Finally, there is the problem of computing the group table of a factor group of G by some normal subgroup N of G. By selecting a cross section of the cosets of N, we can discover which coset any particular product of these elements is in, and consequently generate the multiplication table of the cosets. This table is obtained in the LISP format for a group table, so that we may be able to deal with factor

groups by using the same LISP functions as for groups. The LISP function (FACTORGROUP N), where N is the normal subgroup, performs this calculation. As an example we compute the factor group of the normal subgroup (E A2) of D4.

```
(APPLY (LAMBDA (X) ((LAMBDA (T) (FACTORGROUP X)) (D4))) ((E A2)))
(E (E E A A R R RA RA) A (E A A E R RA RA R) R (E R A RA R E RA A) RA
(E RA A R R A RA E))
```

By rebinding the free variable T to this group table after it has been calculated, we may extract properties such as the classes from this factor group.

```
(APPLY (LAMBDA (N) ((LAMBDA (T) ((LAMBDA (T) (CLASSES)) (FACTORGROUP N)))
(D4))) ((E A2)))
((E) (A) (R) (RA))
```

The final classification of the group structure is its product structure. An important element of this structure concerns the question of whether or not the group may be a direct or semi-direct product of certain of its constituent subgroups. If this is the case, we need only the multiplication tables of these smaller subgroups, and hence quite large groups can be handled by giving their product decomposition. Another case where this decomposition is useful is in calculating representations since the matrix representations of a semi-direct product may often be obtained from the subgroup factors themselves.

Given two groups G and G', the elements of a direct or semi-direct product are elements of the cartesian product GxG', and thus can best be handled as ordered pairs (a b) where $a \in G$, $b \in G'$. As was mentioned before, all that is necessary to use such elements in the program is

to redefine the multiplication and inverses, GP and A-1, along with (GELEMENTS) and (UNIT) for such ordered pairs.

For a direct product this might be done as follows: we let the free variable T be (T1 T2) where, if G = G1xG2, then T1 is the group table for G1, and T2 the group table for G2. Since multiplication for the direct product is defined coordinatewise:

$$(a\ b)(a'\ b') = (aa'\ bb')$$

the function GP would then be:

```
(GP (LAMBDA (A B) (LIST
        (ASSOC (CAR B) (ASSOC (CAR A) (CAR T)))
        (ASSOC (CADR B) (ASSOC (CADR A) (CADR T))))))
```

and similarly for A-1, etc.

In the case of the semi-direct product, the situation is somewhat more complicated since three group tables need to be specified. If $G = G:G' = [(a\ b) \mid a \in G, b \in G']$, then G' is an operator group on G with the multiplication defined:

$$(a\ b)(a'\ b') = (ab(a')\ bb').$$

Consequently, along with the group tables of G and G' respectively, we also need a table of the group G' acting on G. This latter table may be kept in the standard form by extracting b(a) from the table as though it were a product b·a. By letting T = (T1 T2 T3) be a list of the three tables in question, with T1 the table of G, T2 the table of G', and T3 the table of G' acting on G, we have the somewhat complicated, but nevertheless adequate definition for the function GP:

```
(GP (LAMBDA (A B) (LIST
    (ASSOC (ASSOC (CAR B) (ASSOC (CADR A) (CADDR T)
                          (ASSOC (CAR A) (CAR T)))
           (ASSOC (CADR B) (ASSOC (CADR A) (CADR T))))))
```

An example of this is again in the group D4, which is the semi-direct product of the two subgroups A = (E A A2 A3) and R = (E R); that is, D4 = A:R, where the automorphisms of R on A are given by conjugation. If $x \in A$, and $y \in R$, then $y(x) = y^{-1}xy$. For an example, we compute the functions (GELEMENTS), (CLASSES), and (CENTER) of D4 in the semi-direct product notation.

```
(APPLY (LAMBDA () ((LAMBDA (T) (GELEMENTS)) (D4))) ())
((E E) (E R) (A E) (A R) (A2 E) (A2 R) (A3 E) (A3 R))

(APPLY (LAMBDA () ((LAMBDA (T) (CLASSES)) (D4))) ())
((( E E)) ((E R) (A2 R)) ((A E) (A3 E)) ((A R) (A3 R)) ((A2 E)))

(APPLY (LAMBDA () ((LAMBDA (T) (CENTER)) (D4))) ())
((E E) (A2 E))
```

APPENDIX

I. General List Processing Functions

```
(EQUAL (LAMBDA (X Y) (OR
  (EQ X Y)
  (AND (NULL X) (NULL Y))
  (AND (NOT (OR (NULL X) (NULL Y) (ATOM X) (ATOM Y)))
       (EQUAL (CAR X) (CAR Y))
       (EQUAL (CDR X) (CDR Y))))))
```

The time required for computing group theory functions can be significantly decreased by using instead of EQUAL, a function EQUAL*, which is to be defined in each case to test equality of group elements which are of a specific form. For example, if it is known that the group elements are atoms, the definition

(EQUAL* EQ)

will suffice.

```
(LIST (LAMBDA L L))

(APPEND (LAMBDA (L M)
    (IF (NULL L) M (CONS (CAR L) (APPEND (CDR L) M)))))

(ELEM (LAMBDA (X L) (AND
    (NOT (NULL L))
    (OR (EQUAL* X (CAR L)) (ELEM X (CDR L))))))

(SUBSET (LAMBDA (S L) (OR
    (NULL S)
    (AND (ELEM (CAR S) L) (SUBSET (CDR S) L)))))

(SAME* (LAMBDA (L M) (AND (SUBSET L M) (SUBSET M L))))

(MEMBER (LAMBDA (L M) (AND
     (NOT (NULL M))
     (OR (SAME* L (CAR M)) (MEMBER L (CDR M))))))

(REMOVE (LAMBDA (X L)
    (IF (NULL L) L (IF (EQUAL* X (CAR L)) (CDR L) (CONS (CAR L) (REMOVE X
               (CDR L)))))))
```

```
(ERASE (LAMBDA (L M)
    (IF (NULL L) M (ERASE (CDR L) (REMOVE (CAR L) M)))))

(AMONG* (LAMBDA (X L) (IF (ELEM X L) L (CONS X L))))

(TALLYCOMPLEMENT (LAMBDA (L M)
    (IF (NULL L) M (TALLYCOMPLEMENT (CDR L) (CDR M)))))

(AMONGLIST (LAMBDA (L M)
    (IF (NULL L) M (IF (ELEM (CAR L) M) (AMONGLIST (CDR L) M)
    (CONS (CAR L) (AMONGLIST (CDR L) M))))))

(INTERSECTION (LAMBDA (L M)
    (IF (NULL L) L (IF (ELEM (CAR L) M) (CONS (CAR L) (INTERSECTION
    (CDR L) M)) (INTERSECTION (CDR L) M)))))

(UNIONS (LAMBDA (L) (UNIONS* L (LIST))))

(UNIONS* (LAMBDA (L M)
    (IF (NULL L) M (UNIONS* (CDR L) (CONS (CAR L) (APPEND (UNIONS**
    (CAR L) M) M))))))

(UNIONS** (LAMBDA (X L)
    (IF (NULL L) (LIST (CONS (APPEND X (CAR L)) (UNIONS** X (CDR L)))))))

(ASSOC (LAMBDA (X L)
    (IF (EQUAL* X (CAR L)) (CADR L) (ASSOC X (CDDR L)))))

(ASSOC* (LAMBDA (X L)
    (IF (EQUAL* X (CADR L)) (CAR L) (ASSOC* X (CDDR L)))))
```

II. Basic Group Theory Functions

```
(GP (LAMBDA (A B) (ASSOC Y (ASSOC X T))))

(A-1 (LAMBDA (X) (ASSOC* (UNIT) (ASSOC X T))))

(UNIT (LAMBDA () (CAR T)))

(GELEMENTS (LAMBDA () (GELEMENTS* T)))

(GELEMENTS* (LAMBDA (L)
    (IF (NULL L) L (CONS (CAR L) (GELEMENTS* (CDDR L))))))

(LGP (LAMBDA (X L)
    (IF (NULL L) L (CONS (GP X (CAR L)) (LGP X (CDR L))))))
```

```
(RGP (LAMBDA (X L)
    (IF (NULL L) L (CONS (GP (CAR L) X) (RGP X (CDR L))))))

(MULTIGP (LAMBDA L (MONOGP L)))

(MONOGP (LAMBDA (L) (GP (CAR L) (IF (NULL (CDDR L)) (CADR L) (MONOGP
    (CDR L))))))

(COMPLEXGP (LAMBDA (L M)
    (IF (NULL L) L (AMONGLIST (LGP (CAR L) M) (COMPLEXGP (CDR L) M)))))

(COMMUTATOR (LAMBDA (A B) (MULTIGP A B (A-1 A) (A-1 B))))

(CONJUGATE (LAMBDA (P X) (MULTIGP (A-1 P) X P)))

(INVERSES (LAMBDA (L)
    (IF (NULL L) L (CONS (A-1 (CAR L)) (INVERSES (CDR L))))))

(CONJUGATES (LAMBDA  (X L)
    (IF (NULL L) L (CONS (MULTIGP (A-1 (CAR L)) (CONJUGATES (CDR L)))))))

(GP X  Y);
(APPLY (LAMBDA (X Y) ((LAMBDA (T) (GP X Y))( D4))) (A RA3))
RA2
(A-1 X);
(APPLY (LAMBDA (X) ((LAMBDA (T) (A-1 X)) (D4))) (A3))
A
(GELEMENTS);
(APPLY (LAMBDA () ((LAMBDA (T) (GELEMENTS)) (D4))) ())
    (E A A2 A3 R RA RA2 RA3)

(LGP X L);
(APPLY (LAMBDA (X Y) ((LAMBDA (T) (LGP X Y)) (D4))) (R (A A2 A3)))
    (RA RA2 RA3)

(RGP X  L);
(APPLY (LAMBDA (X Y) ((LAMBDA (T) (RGP X Y)) (D4))) (R (A A2 A3 )))
    (RA3 RA2 RA)

(MULTIGP X Y Z W ...);
(APPLY (LAMBDA (X Y Z W) ((LAMBDA (T) (MULTIGP Z W Y X)) (D4))) (A R A2
    RA))
A2
(COMPLEXGP X Y);
(APPLY (LAMBDA (X Y) ((LAMBDA (T) (COMPLEXGP X Y)) (D4))) ((R RA) (A RA2)))
    (RA A2 RA2 A)
```

III. Substructure Functions

```
(HULL (LAMBDA (C) ((LAMBDA (W) (HULL1  C (ERASE C W))) (COMPLEXGP  C C))))

(HULL1  (LAMBDA (U V)
    (IF (NULL V) U ((LAMBDA (X) (HULL1 X (HULL2 X V))) (APPEND V U)))))

(HULL2 (LAMBDA (C1 C2) (ERASE C1 (AMONGLIST (COMPLEXGP C1 C2) (COMPLEXGP
    C2 U)))))

(CYCLE (LAMBDA (A) (CYCLE* A)))

(CYCLE* (LAMBDA (X)
    (IF (EQUAL* X (UNIT) (LIST X) (CONS X (CYCLE* (GP A X)))))))

(CYCLICSGROUPS (LAMBDA () (CYCLICSGROUPS* (REMOVE (UNIT) (GELEMENTS))
    (LIST))))

(CYCLICSGROUPS* (LAMBDA (G L)
    (IF (NULL G) L ((LAMBDA (X) (IF (MEMBER X L) (CYCLICSGROUPS* (CDR G) L)
    (CONS X L))) (CYCLE (CAR G))))))

(SUBGROUPS (LAMBDA () (CONS (LIST (UNIT)) (( LAMBDA (X)
    (SUBGROUPS1 X X (LIST))) (CDR (CYCLICSGROUPS))))))

(SUBGROUPS1 (LAMBDA (C* C L)
    (IF (NULL C) C ((LAMBDA (X)
        (SUBGROUPS2 (CAR C) X X)) (SUBGROUPS1 (CDR C*) (CDR C) L)))))

(SUBGROUPS2 (LAMBDA (C L* L)
    (IF (NULL L*) (CONS C L) ((LAMBDA (H)
    (IF (OR (MEMBER H L) (MEMBER H C*)) (SUBGROUPS2 C  CDR L*) L)
    (SUBGROUPS2 C (CDR L* (CONS H L)))) (HULL (AMONGLIST C (CAR L*)))))))

(NSUBGROUPS (LAMBDA () (CONS (LIST (UNIT)) (NSUBGROUPS* (CLASSPAIRS
    (UNIONS (CDR (CLASSES))) (LIST)))))

(NSUBGROUPS* (LAMBDA (L M)
    (IF (NULL L) M ((LAMBDA (X)
    (IF (NULL (TALLYCOMPLEMENT X (COMPLEXGP X X))) (NSUBGROUPS* (CDR L)
    (CONS X M)) (NSUBGROUPS* (CDR L) M))) (CONS (UNIT) (CAR L))))))

(CLASSPAIRS (LAMBDA (L)
    (IF (NULL L) L (IF (ELEM (A-1 (CAAR L)) (CAR L)) (CONS (CAR L)
    (CLASSPAIRS (CDR L)) ((LAMBDA (Z) (CONS (CAR Z) (CLASSPAIRS (CDR Z))))
    (PAIR (CAR L) (CDR L))))))
```

```
(PAIR (LAMBDA (X L)
    (IF (NULL X) L (IF (ELEM (A-1 (CAR X)) (CAR L))
    (CONS (APPEND X (CAR L)) (PAIR (LIST) (CDR L)))
    (CONS (CAR L) (PAIR X (CDR L)))))))

(CENTER (LAMBDA () ((LAMBDA (X) (CENTER* (CAR X) (CDR X))) (GELEMENTS))))

(CENTER* (LAMBDA (A G L)
    (IF (NULL G) (CONS (UNIT) L) (CENTER* (CAR G) (CDR G) (NORMALIZER* L)))))

(NORMALIZER (LAMBDA (A) (NORMALIZER* (GELEMENTS))))

(NORMALIZER* (LAMBDA (G)
    (IF (NULL G) G (IF (EQUAL* (GP (CAR G) A) (GP A (CAR G)))
    (CONS (CAR G) (NORMALIZER* (CDR G))) (NORMALIZER* (CDR G))))))

(COMSGROUP (LAMBDA () (HULL (COMSGROUP* (GELEMENTS)))))

(COMSGROUP* (LAMBDA (G)
    (IF (NULL G) G (AMONGLIST (COMOP (CAR G) (CDR G)) (COMSGROUP* (CDR G))))))

(COMOP (LAMBDA (X L)
    (IF (NULL L) L (CONS (COMMUTATOR X (CAR L)) (COMOP X (CDR L))))))

(STABILIZER (LAMBDA (C) (STABILIZER* (GELEMENTS))))

(STABILIZER* (LAMBDA (G)
    (IF (NULL G) G (IF (SAME* (LGP (CAR G) C) (RGP (CAR G) C))
    (CONS (CAR G) (STABILIZER* (CDR G))) (STABILIZER* (CDR G))))))

(CENTRALIZER (LAMBDA (C) (CENTRALIZER* (GELEMENTS))))

(CENTRALIZER* (LAMBDA (G)
    (IF (NULL G) G (IF (CENTRAL (CAR G) C) (CONS (CAR G) (CENTRALIZER*
    (CDR G))) (CENTRALIZER* (CDR G))))))

(CENTRAL (LAMBDA (A C) (OR (NULL C)
    (AND (EQUAL* (GP A (CAR C)) ((GP (CAR C) A)) (CENTRAL A (CDR C))))))
```

IV. Factor Structure Functions

```
(EQRELATION (LAMBDA (R L)
    (IF (NULL L) L ((LAMBDA (W)
    (CONS W (EQRELATION R (ERASE W L))) (R (CAR L))))))

(LCOSETS (LAMBDA (S) (LCOSETS* (GELEMENTS))))

(LCOSETS* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X) (LGP X S))) G)))
```

```
(RCOSETS (LAMBDA (S) (RCOSETS* (GELEMENTS))))

(RCOSETS* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X) (RGP X S))) G)))

(DBLCOSETS (LAMBDA (H K) (DBLCOSETS*(GELEMENTS))))

(DBLCOSETS* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X)
    (COMPLEXGP H (LGP X K)))) G)))

(CLASSES (LAMBDA () (CLASSES* (GELEMENTS))))

(CLASSES* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X)
    (CONJUGATES X (XSECTION (RCOSETS (NORMALIZER X)))))) G)))

(HCLASSES (LAMBDA (H) (HCLASSES* (GELEMENTS))))

(HCLASSES* (LAMBDA (G) (EQRELATION (FUNCTION (LAMBDA (X)
    (CONJUGATES X (XSECTION ((LAMBDA (A) ((LAMBDA (S)
    (LCOSETS* H)) (NORMALIZER* H))) X))))) G)))

(FACTORGROUP (LAMBDA (N) ((LAMBDA (C*) ((LAMBDA (G*) (FACTORGROUP* G*))
    (XSECTION C*))) (LCOSETS N))))

(FACTORGROUP* (LAMBDA (G)
    (IF (NULL G) G (CONS (CAR G) (CONS (FACTORROW (CAR G) G*)
    (FACTORGROUP* (CDR G)))))))

(FACTORROW (LAMBDA (A G)
    (IF (NULL G) G (CONS (CAR G) (CONS (COSETREP (GP A (CAR G)) C*)
    (FACTORROW A (CDR G)))))))

(COSETREP (LAMBDA (X L)
    (IF (ELEM X (CAR L)) (CAAR L) (COSETREP X (CDR L)))))
```

V. Product Structure Functions

A. Direct Product Functions

```
(GP (LAMBDA (A B) (LIST (ASSOC (CAR B) (ASSOC (CAR A) (CAR T)))
                        (ASSOC (CADR B) (ASSOC (CADR A) (CADR T))))))

(A-1 (LAMBDA (X) (LIST (ASSOC* (CAAR T) (ASSOC (CAR X)(CAR T)))
                       (ASSOC* (CAADR T) (ASSOC (CADR X) (CADR T))))))

(UNIT (LAMBDA () (LIST (CAAR T) (CAADR T))))

(GELEMENTS (LAMBDA () (CARTESIAN (GELEMENTS* (CAR T)) (GELEMENTS* (CADR T)))))
```

```
(CARTESIAN (LAMBDA (L)
    (IF (NULL L) (LIST L) (BICARTESIAN (CAR L) (CARTESIAN (CDR L))))))

(BICARTESIAN (LAMBDA (U V)
    (IF (NULL U) U ((LAMBDA (U*) (BICARTESIAN* V)) (CAR U)))))

(BICARTESIAN* (LAMBDA (V*)
    (IF (NULL V*) (BICARTESIAN (CDR U) V)
    (CONS (CONS U* (CAR V*)) (BICARTESIAN* (CDR V*))))))
```

B. Semi-Direct Product Functions

```
(GP (LAMBDA (A B) (LIST (ASSOC (ASSOC (CAR B) (ASSOC (CADR A) (CADR T)))
                                    (ASSOC (CAR A) (CAR T)))
                        (ASSOC (CADR B) (ASSOC (CADR A) (CADR T))))))

(A-1 (LAMBDA (X) ((LAMBDA (Z) (LIST
        (ASSOC (ASSOC* (CAAR T) (ASSOC (CAR A) (CAR T))) (ASSOC Z (CADDR
                                T))) Z)))
        (ASSOC* (CAADR T) (ASSOC (CADR A) (CADR T))))))

(UNIT (LAMBDA () (LIST (CAAR T) (CAADR T))))

(GELEMENTS (LAMBDA () (CARTESIAN (GELEMENTS* (CAR T)) (GELEMENTS*
                            (CADR T)))))

(D4 (LAMBDA () (QUOTE
    ((E (E E A A A2 A2 A3 A3)
      A (E A A A2 A2 A3 A3 E)
     A2 (E A2 A A3 A3 E A3 A)
     A3 (E A3 A # A2 A A3 A2))
     (E (E E R R)
      R (E R R E))
     (E (E E A A A2 A2 A3 A3)
      R (E E A A3 A2 A2 A3 A))))))
```