Massachusetts Institute of Technology
Cambridge, Massachusetts
Project MAC

A. I. 89B

Memorandum MAC-M-282
December 1, 1965

Computer Experiments in Finite Algebra - II

by W. D. Maurer

In a previous memo (Computer Experiments in Finite Algebra, MAC-M-246) we described a computer system for the handling of finite groups, semigroups, subsets, finite maps, and constants. This system has been extended to read and write disk files; a mechanical procedure has been developed for extending the system; and a program (the Inferential Compiler) has been written which accepts a source language consisting of mathematical statements in a standard format and compiles code which verifies these statements over a file or files of special cases (including possible counterexamples).

Three limitations of the system were mentioned in the previous memo. Of these, (1) and (3) have been effectively eliminated in the current system. Limitation (2) still

exists and will be overcome only in ALGBRA III, which is briefly described in section 4.

CONTENTS

1. The ALGBRA II System 3

2. The Inferential Compiler 8

3. Directions for Writing Functions 13

4. Extensions of the Current System 17

APPENDIX A. Names, Calling Phrases, and Descriptions of Functions 18

APPENDIX B. Primitive Functions and their Frames for the Inferential Compiler Source Language 38

1. The Algbra II System

ALGBRA II is a medium-scale computer system which operates on the time-shared IBM 7094. It is written in MAD and FAP.

Using ALGBRA II, a student may manipulate the objects studied in modern algebra: finite groups, semigroups, maps between them (including homomorphisms and isomorphisms), subsets of groups and semigroups, and sets of maps (including sets of permutations). He may test whether a semigroup is a group; whether a group is abelian; whether a semigroup has an identity; whether a map is a homomorphism, or one-to-one, or onto; or whether two semigroups are isomorphic. He maytake the direct product of two groups or the factor group of a group by a normal subgroup. He may generate all subgroups of a group, or all normal subgroups, or all maximal subgroups. He may introduce any group or semigroup into the system by typing its Cayley table, or call specifically for symmetric, cyclic, alternating, or dihedral groups.

A mathematician wishing to test some conjecture involving finite groups and semigroups may type his conjecture in a standard form which resembles the English-language statement of the conjecture. He may then compile a routine which will test this conjecture over a standard file of special cases, or a file which he may create. This is done with the Interpretive Compiler.

Use of the ALGBRA II Source Language from the Console

Each routine is called by a calling phrase, which is an English sentence terminated by a period. The name of each routine currently in the system, its calling phrase, and a description of its function is given in Appendix A.

The routines are requested, one by one, from the console. Upon entry to ALGBRA, the program types

*

The user should type the calling phrase of some routine, with parameters which he specifies, according to the rules given below. For example (using the first calling phrase listed below) he might type INPUT CAYLEY TABLE G1, ORDER 8. (The period is essential.)

If a calling phrase is mistyped, ALGBRA will print

?

followed by another asterisk as above. Normally, ALGBRA executes the called routine and returns to type another asterisk.

Parameters

A parameter consists of a string of alphanumeric symbols from one to six characters long. A parameter is the name of either a constant, a table, a file, or a routine. A constant may have the same name as a table, etc. Each routine specifies, for each parameter, whether this

parameter is a constant, a table (of various types) a file, or a routine.

In the descriptions of the phrases given in Appendix A, parameters are represented by special symbols. Constant parameters are represented by the symbols $C1 through $C7. Thus if the fourth parameter is a constant, $C4 appears. (Note: this is the fourth parameter, not the fourth constant parameter.) Boolean constants are represented by the symbols $BC1 through $BC7. Such constants have the value 0 (false) or 1 (true).

Parameters referring to tables are represented by the symbols $T1 through $T7, if any type of table is permissible. Otherwise, they are represented by symbols which signify the type of table. There are five types of objects which may be represented by tables:

(1) Finite Maps ($M1 through $M7). A finite map is a map whose domain is finite. A finite map whose domain has cardinality n is specified by an n-position array, giving the image of each element of the domain.

(2) Sets of Maps ($SM1 through $SM7). A (finite) set of finite maps is represented by a rectangular array. Each row of the array is a finite map.

(3) Subsets ($S1 through $S7). A subset of a finite set of order n is given within the computer by a table containing non-zeros (usually ones) in each position corresponding to an element of the subset.

(4) Sets of Subsets ($SS1 through $SS7). A (finite) set of subsets is represented by a rectangular array, each row of which is a subset as described above.

(5) Cayley Tables ($CT1 through $CT7). A Cayley table of order n is an n x n array, giving the multiplication table of a group or a semigroup of order n. The elements of the semigroup are taken to be the integers from 1 to n for internal calculations, and the integers from 10 to n-1 for communication with the console. Since a semigroup is closed under multiplication, the product of the integers i and j will again be an integer from 1 to n. This integer is contained in the (i, j)-th position in the table. A constant may be regarded as an element of a Cayley table; in this case, its value is increased by 1 for internal use.

A file name is specified by one of the symbols $F1 through $F7; a routine name is specified by one of the symbols $RN1 through $RN7.

The asterisk (*) after a parameter symbol, such as $C1*, signifies that this constant or table is inserted into the system by the given routine. If the asterisk is not present, the constant or table is assumed to be in the system at the start of the routine (otherwise the routine is not executed), and is used by it.

Numerical parameters, which consist of from one to six digits from 0 through 9, are treated separately. A numerical parameter may be used anywhere that a constant parameter is called for. Numerical parameters are inserted into the

system during the interpretation of the calling phrase.

2. The Inferential Compiler

By an inferential compiler we mean a program which accepts source statements expressed as inferences, i. e., mathematical statements, and produces as output a relocatable program which, when executed, proves or disproves, or verifies, the given statements. The current inferential compiler verifies statements; it does not prove them. That is, it checks their validity over a file of special cases. This file is specified in the statements themselves; one may write FOR ALL G1 IN F1, for example, where F1 is a file name.

The source statements to the inferential compiler are written in a language which extends the usual LISP formalism for mathematical statements. The process by which a statement such as

> If the groups G1 and G2 are abelian, then the direct product of G1 and G2 is also abelian

is transformed into an S-expression such as

```
(IMPLIES (AND (ABELIAN G1) (ABELIAN G2))
(ABELIAN (DIRECTPRODUCT G1 G2)))
```

is quite well known; in fact, one of the major objectives of the S-expression formalism was to provide a concise standard form for logical statements, and in particular mathematical theorems. We shall, however, state this transformation

formally, in order to provide a framework for our extension of it.

A well-formed S-expression of the first kind may be defined relative to a finite set of function specifications. Each specification includes the name of the function, the number of arguments, the type of each argument, and the type of the value of the function. Thus AND, in the above example, takes two arguments which are of type Boolean, and its value is of type Boolean; ABELIAN takes one argument, of type "group," and its value is of type Boolean. In formal terms, if we are given:

(a) a set X of objects,

(b) a set T of types,

(c) a function t: X → T, assigning a type to each object,

(d) a set F of function specifications f, where each f is of the form (n l vt), with n the name of the function, vt the type of its value (a member of T), and l a list (t1 t2 ... tn) of members of T, whose length is the number of arguments of f, and such that ti is the type of the i-th argument,

then a well-formed S-expression of the first kind is either an object x, in which case its type is t(x), or an expression of the form (n x1 x2 ... xn), in which n is the name of a function f in F and each xi is a well-formed S-expression of the first kind whose type is ti, as it occurs in the specification for f; and in this case the type

of the defined expression is v(f).

The extension which we make in our input format is to add to the function specification a frame, which is an English sentence (for a function whose value is Boolean) or noun phrase (for a function whose value is not Boolean) with blanks to be filled in. For example, instead of saying (ABELIAN G) we now say (G IS ABELIAN). Here G is an object that was substituted into the original frame, which was ((1, group) IS ABELIAN). The "1" here signifies the first parameter (there is only one) and the "group" signifies the type of this parameter. Another expression can be substituted instead of an object. To illustrate this, suppose we take the functions of the example above, which were

```
(IMPLIES X Y)
(AND X Y)
(ABELIAN G)
(DIRECTPRODUCT G1 G2)
```

and replace them by frames, thus:

```
(IF (1, Bool), THEN (2, Bool))
((1, Bool) AND (2, Bool))
((1, group) IS ABELIAN)
(THE DIRECT PRODUCT OF (1, group) AND (2, group))
```

Then instead of the source statement

```
(IMPLIES (AND (ABELIAN G1) (ABELIAN G2))
(ABELIAN (DIRECTPRODUCT G1 G2)))
```

we have the source statement

```
(IF ((G1 IS ABELIAN) AND (G2 IS ABELIAN)),
THEN ((THE DIRECT PRODUCT OF G1 AND G2) IS
ABELIAN))
```

Clearly the second formalism includes the first as a special case; we can just as easily structure our frames to look like standard LISP functions.

In formal terms, if we are given:

(a) a set X of objects,

(b) a set T of types,

(c) a function t: $X \rightarrow T$, assigning a type to each object;

(d) a set F of function specifications f, where each f is of the form (fr vt), where vt is the type of the value of the function (a member of T), and fr, the frame, is a list (w1 w2 ... wk), where each wi is either a word, or is of the form (z t), where z is an integer and t is a member of T;

then a well-formed S-expression of the second kind is either an object x, in which case its type is t(x), or the result of substituting a well-formed S-expression of the second kind, of type t, for each wi of the form (z t) in the frame of a function specification f; and in this case the type of the defined expression is v(f).

The input to the inferential compiler, then, consists of well-formed S-expressions of the second kind, relative to the function specifications given in Appendix B. At the present time, the compiler takes one expression at a time and compiles it. The compiler is itself called as a function in the standard console source language. It asks for its own

needed data--i. e., what the new function is to be called, what calling phrase is to be used for it, etc. It then compiles the routine and inserts the resulting function into the system.

Each function specification, as given in Appendix B, consists of a routine name (whose function is described in Appendix A), a frame, and a value type. The frame is either an English sentence or an English noun phrase. It is a sentence if and only if the value type is Boolean. The parameters of the frame, of the form (z t) described above, are expressed by the same special symbols as are used in Appendix A.

3. Directions for Writing Functions

Any routine written in MAD, FAP, or any other language which produces a standard BSS deck, may communicate with the ALGBRA system, provided that it uses the primitives of the ALGBRA system and obeys the ground rules associated with these primitives.

Each routine in the system is called by the console calling phrase interpreter, a program named PHRA, using a standard MAD calling sequence:

```
        TSX     RTN,4
        TXH     PARAM1
        TXH     PARAM2
        ...     ......
        TXH     PARAMN
        NORMAL RETURN
```

The words PARAM1, PARAN2, ..., PARAMN are the BCD names of the corresponding parameters. In order to retrieve the actual parameters, system routines must be called. These routines differ depending on whether the parameters is supplied to the routine from the outside, or supplied to the outside by the routine.

For constant parameters, the routines used are COSR (Constant Search) and COSI (Constant Search and Insert). These are called (from MAD programs) by EXECUTE COSR.(C, KC) or EXECUTE COSI.(C, KC), here C is the BCD name of the constant and KC is a cell which contains the location of the

constant. To use a constant, use the function XCONT ("contents of"), applied to KC; i. e., N=XCONT.(KC). To return a value to a constant, use the routine REPL (replace), which replaces the cell pointed to by the first parameter with the second parameter. Thus to place N in the constant pointed to by KC, EXECUTE REPL.(KC, N). Do not use the statements N=KC, KC=N, or the like.

For table parameters, the routines used are CTSR (Cayley Table Search), CTSI (Cayley Table Search and Insert), CTSRNM, CTSINM, and XMSI. (In spite of their names, these functions work for any table.) In order to insure compatibility with a table collapsing scheme now under development, each table has associated with it a cell called a movable, which points to the table and records the fact that the table may be moved around in core by the collapsing scheme at some later time. Collapsing can never occur on pure searching, only on searching and inserting, and thus provision for collapsing is not needed when only searching is to be done; in addition, the last insertion, and all searches following this insertion, do not need movables. Hence there are two alternate routines, CTSRNM and CTSINM (CTSR - No Movable and CTSI - No Movable) which save a little time. The calling sequences for CTSR and CTSRNM are EXECUTE CTSR.(CT, KCT, LM) and EXECUTE CTSRNM.(CT, KCT), where CT is the BCD name of the table, KCT is a cell which contains the location of the table, and LM is a cell containing the movable. If a movable is created, it must be

removed at the end of the routine with ERMV (Erase Movable), which has one parameter; i. e., EXECUTE ERMV.(LM).

In the case of CTSR and CTSRNM, the dimensions of the corresponding table may be found by using the functions DIM1 and DIM2; i. e., N1=DIM1.(KCT) and N2=DIM2.(KCT). All arrays are kept as double arrays; a single array is kept as a 1 x n double array. Thus in this case N2 is the only relevant dimension.

The calling sequences for CTSI and CTSINM are EXECUTE CTSI.(CT, KCT, DIM1, DIM2, LM) and EXECUTE CTSINM.(CT, KCT, DIM1, DIM2), where CT is the BCD name of the table, KCT is a cell which contains the location of the table, DIM1 and DIM2 are the dimensions of the (new) table, and LM is a cell containing the movable. If a movable is used, it must be removed at the end of the routine using ERMV. For a new single array, DIM1 will be equal to 1. To reference an element of a table, use the function ITEM; the element of the table pointed to by KCT, in the (I, J)-th position, is ITEM.(KCT, I, J). To place an element in a table, first locate the index of the item by using IITEM (Index of Item); IITEM.(KCT, I, J) is the index, in a COMMON array called MEMORY, of ITEM.(KCT, I, J). Thus one may write N1=IITEM.(KCT, I, J), followed by MEMORY(N1)=N2, to place N2 in the (I, J)-th element of this table. Tables are stored backward in memory, which corresponds to a forward arrangement of the indices, the statement N1=N1+1 after the preceding two will position us to the (I, J+1)-st element of

the table, or to the (I+1, 1)st if J is equal to the second dimension of the table.

There is also a provision for reserving blocks of "scratch memory." This is given by a routine XMSI (Scratch Memory Search and Insert), whose calling sequence is EXECUTE XMSI.(KCT, DIM1, DIM2, LM). All parameters have the same significance as in CTSI. (There is also an XMSINM, or XMSI - No Movable.) One area of expandable scratch memory may be reserved, as, for example, to accumulate a set of subsets. In the expandable scratch memory feature, the first dimension of the scratch memory table may be increased by N using the routine XPXM (Expand Scratch Memory); i. e., EXECUTE XPXM.(KCT, N).

4. Extensions to the Current System

A set of routines to manipulate finite rings and fields will soon be written for ALGBRA, including a routine to calculate the Galois group of a given polynomial.

It is planned to greatly expand the number of data types in the system. Some of the new data types will involve lists. Both SLIP and the AED free storage routines will be used to manipulate these lists.

A large-scale mathematical system, including theorem proving, is under study for either the PDP-6 or the GE 635.

* * * * * * * * * * * *

APPENDIX A

Names, Calling Phrases, and Descriptions of Functions

IPCT INPUT CAYLEY TABLE $CT1*, ORDER $C2.

The Cayley table $CT1* is input to ALGBRA from the console. IPCT and the next six routines (IPST, IPSC, IPMA, IPMD, IPSM, IPCO) give their own further instructions.

IPST INPUT SUBSET $S1*.

The subset $S1* is input to ALGBRA from the console. The user is asked how large the domain of the subset is; the answer must be a two-digit integer.

IPSC INPUT SUBSET $S1* of $CT2.

The subset $S1* of the Cayley table $CT2 is input to ALGBRA from the console.

IPMA INPUT MAP $M1*.

The map $M1* is input to ALGBRA from the console. The user is asked how large the domain of the map is; the answer must be a two-digit integer.

IPMD INPUT MAP $M1* WITH DOMAIN $CT2.

The map $M1* with the Cayley table $CT2 as domain is input to ALGBRA from the console.

IPSM INPUT SET $SM1* of $C2 MAPS.

The set of maps $SM1*, containing $C2 maps, is input to ALGBRA from the console. The user is asked how large the domain of the maps is; the answer must be a two-digit integer.

IPCO INPUT CONSTANT $C1*.

The constant $C1* is input to ALGBRA from the console.

DUCT PRINT CAYLEY TABLE $CT1.

The Cayley table $CT1 is displayed on the console.

DUST PRINT SUBSET $S1.

The subset $S1 is displayed on the console, together with a count of its elements.

DUMA PRINT MAP $M1.

The map $M1 is displayed on the console.

DUSM PRINT SET OF MAPS $SM1.

The set of maps $SM1 is displayed (as a rectangular array) on the console.

DUCO PRINT CONSTANT $C1.

The constant $C1 (this may also be a Boolean constant $BC1) is displayed on the console. Often a routine will return a constant (e.g. a number of subgroups) which the

user should know in order to proceed.

ERTA ERASE TABLE $T1.

The table $T1, of whatever type (Cayley table, subset, map, set of subsets, set of maps) is erased. This provides more space in the computer for other tables. (Note: ERTA and the next routine, ERCO, erase quantities in core by setting the corresponding names to zero. The space does not actually become available until the tables are collapsed by means of COLAPS.)

ERCO ERASE CONSTANT $C1.

The constant (or Boolean constant) $C1 is erased.

RNTA TABLE $T2* IS $T1.

The table T1, of whatever type, is renamed; that is, it now has two names, $T1 and $T2*.

RNCO CONSTANT $C2* IS $C1.

The constant $C1, of whatever type, is renamed; that is, it now has two names, $C1 and $C2*.

COLA COLLAPSE TABLES.

The internal tables of ALGBRA are collapsed. Zero entries in the tables, corresponding to deleted constants or tables, are eliminated.

QUIT QUIT.

This routine brings the user to CTSS command level. He may save his core image and restart at a later time. If any new functions have been entered, the disk files containing the function tables are updated.

ALCT ALTERNATING GROUP $CT2* ON $C1 LETTERS.

The alternating group on $C1 letters is generated in core and given the name $CT2*.

SYCT SYMMETRIC GROUP $CT2* ON $C1 LETTERS.

This routine

The symmetric group on $C1 letters is generated in core and given the name $CT2*.

CYCT CYCLIC GROUP $CT2*, ORDER $C1.

The cyclic group of order $C1 is generated in core and given the name $CT2*.

DHCT DIHEDRAL GROUP $CT2* ON $C1 VERTICES.

The dihedral group on $C1 vertices (i.e., the group of rotations and reflections of a regular polygon with $C1 vertices) is generated in core and given the name $CT2*.

LACT LEFT ZERO SEMIGROUP $CT2*, ORDER $C1.

The left zero semigroup of order $C1 (i.e., the set of $C1 elements, with multiplication defined by xy = x) is

generated in core and given the name $CT2*.

RACT RIGHT ZERO SEMIGROUP $CT2*, ORDER $C1.

The right zero semigroup of order $C1 (i.e., the set of $C1 elements, with multiplication defined by xy = y) is generated in core and given the name $CT2*.

NACT NULL SEMIGROUP $CT2*, ORDER $C1.

The null semigroup of order $C1 (i.e., the set of $C1 elements, with multiplication defined by xy = z, where z is the null element, in this case 0) is generated in core and given the name $CT2*.

CTDP DIRECT PRODUCT $CT3* OF $CT1 AND $CT2.

The direct product of the semigroups represented by the Cayley tables $CT1 and $CT2 is generated in core and given the name $CT3*.

DASS GENERATE SET $SS2* OF ALL SUBSEMIGROUPS OF $CT1.

The set of all subsemigroups of $CT1 (all subgroups of $CT1, if $CT1 is a group) is generated in core as a set of subsets, and given the name $SS2*.

DALI GENERATE SET $SS2* OF ALL LEFT IDEALS OF $CT1.

The set of all left ideals of the semigroup $CT1 is generated in core as a set of subsets and given the name $SS2*.

DARI GENERATE SET $SS2* OF ALL RIGHT IDEALS OF $CT1.

The set of all right ideals of the semigroup $CT1 is generated in core as a set of subsets and given the name $SS2*.

DATI GENERATE SET $SS2* OF ALL TWO-SIDED IDEALS OF $CT1.

The set of all two-sided ideals of the semigroup $CT1 is generated in core as a set of subsets and given the name $SS2*.

DANS GENERATE SET $SS2* OF ALL NORMAL SUBGROUPS OF $CT1.

The set of all normal subgroups of the groups $CT1 is generated in core as a set of subsets and given the name $SS2*.

DMSS GENERATE SET $SS2* OF ALL MAXIMAL SUBSEMIGROUPS OF $CT1.

The set of all maximal subsemigroups of $CT1 (all maximal subgroups of $CT1, if $CT1 is a group) is generated in core as a set of subsets, and given the name $SS2*.

DMLI GENERATE SET $SS2* OF ALL MAXIMAL LEFT IDEALS OF $CT1.

The set of all maximal left ideals of the semigroup $CT1 is generated in core as a set of subsets and given the name $SS2*.

DMRI GENERATE SET $SS2* OF ALL MAXIMAL RIGHT IDEALS OF $CT1.

The set of all maximal right ideals of the semigroup $CT1 is generated in core as a set of subsets and given the name $SS2*.

DMTI GENERATE SET $SS2* OF ALL MAXIMAL TWO-SIDED IDEALS OF $CT1.

The set of all maximal two-sided ideals of the semigroup $CT1 is generated in core as a set of subsets and given the name $SS2*.

DMNS GENERATE SET $SS2* OF ALL MAXIMAL NORMAL SUBGROUPS OF $CT1.

The set of all maximal normal subgroups of the group $CT1 is generated in core as a set of subsets and given the name $SS2.

EGSS GENERATE SUBSEMIGROUP $S3* FROM ELEMENT $C2 OF $CT1.

The subsemigroup of the semigroup $CT1 (or the subgroup of the group $CT1) generated by the element $C2 is constructed in core and given the name $S3*.

EGLI GENERATE LEFT IDEAL $S3* FROM ELEMENT $C2 of $CT1.

The left ideal of the semigroup $CT1 generated by the element $C2 is constructed in core and given the name $S3*.

EGRI GENERATE RIGHT IDEAL $S3* FROM ELEMENT $C2 OF $CT1.

The right ideal of the semigroup $CT1 generated by the element $C2 is constructed in core and given the name $S3*.

EGT1 GENERATE TWO-SIDED IDEAL $S3* FROM ELEMENT $C2 OF $CT1.

The two-sided ideal of the semigroup $CT1 generated by the element $C2 is constructed in core and given the name $S3*.

ABTE IS $CT1 ABELIAN, ANSWER $BC2*.

If the semigroup (or group) $CT1 is abelian, then BC2* is set to 1 (true); otherwise, it is set to 0 (false).

GRTE IS $CT1 A GROUP, ANSWER $BC2*.

If the semigroup $CT1 is a group, then $BC2* is set to 1; otherwise, to 0.

SGTE IS $CT1 A SEMIGROUP, ANSWER $BC2*.

If the Cayley table $CT1 is associative, then $BC2* is set to 1; otherwise, to 0.

GRTI IS $CT1 A GROUP, ANSWER $BC2*, INVERSE TABLE $M3*.

If the semigroup $CT1 is a group, then $BC2* is set to 1, and a map is generated, giving for each element its inverse, and given the name $M3*. If $CT1 is not a group, $BC2* is set to 0.

SGIT ARE $CT1 AND $CT2 ISOMORPHIC, ANSWER $BC3*,
ISOMORPHISM $M4*.

If the semigroups $CT1 and $CT2 are isomorphic, then $BC3* is set to 1, and an isomorphism $M4* is constructed from $CT1 to $CT2. If the given semigroups are not isomorphic, $BC3* is set to 0.

U2ST UNION $S3* OF SUBSETS $S1 and $S2.

The union of the subsets $S1 and $S2 is constructed as a subset $S3*. All subsets are presumed to be of the same set.

I2ST INTERSECTION $S3* OF SUBSETS $S1 and $S2.

The intersection of the subsets $S1 and $S2 is constructed as a subset $S3*. All subsets are presumed to be of the same set.

CPST COMPLEMENT $S2* OF SUBSET $S1.

The complement of the subset $S1 is constructed and given the name $S2*.

HMTE IS MAP $M3 FROM $CT1 TO $CT2 A HOMOMORPHISM,
ANSWER $BC4*.

If the map $M3 from the semigroup $CT1 to the semigroup $CT2 is a homomorphism, $BC4* is set to 1; otherwise, it is set to 0.

MMTE IS MAP $M3 FROM $CT1 TO $CT2 A MONOMORPHISM, ANSWER $BC4*.

If the map $M3 from the semigroup $CT1 to the semigroup $CT2 is a monomorphism, $BC4* is set to 1; otherwise, it is set to 0.

FMTE IS MAP $M3 FROM $CT1 TO $CT2 AN EPIMORPHISM, ANSWER $BC4*.

If the map $M3 from the semigroup $CT1 to the semigroup $CT2 is an epimorphism, $BC4* is set to 1; otherwise, it is set to 0.

IMTE IS MAP $M3 FROM $CT1 TO $CT2 AN ISOMORPHISM, ANSWER $BC4*.

If the map $M3 from the semigroup $CT1 to the semigroup $CT2 is an isomorphism, $BC4* is set to 1; otherwise, it is set to 0.

CTST CAYLEY TABLE $CT3* OF SUBSET $S2 OF SEMIGROUP $CT1.

The subset $S2 of the semigroup $CT1 is expanded into a full Cayley table $CT3*. The subset is assumed to be closed. (Note in review: Subsets are expressed by a table which has length equal to the order of the original semigroup. This may be greater than the size of the Cayley table of the subset; besides, the Cayley table format may be needed specifically, e.g., as input to other routines. On the other hand, converting to Cayley table format means that the structure of the subset within the original semigroup is

disregarded by any reference to the new Cayley table.)

EGCT CAYLEY TABLE $CT3* OF FACTOR GROUP $CT1/$S2.

The subset $S2 is assumed to be a normal subgroup of the group $CT1. The factor group is constructed as a Cayley table $CT3*.

CTXB CAYLEY TABLE $CT3* IS IMAGE OF $CT1 UNDER PERMUTATION $M2.

The elements (from 0 to n-1) of the Cayley table $CT1 are renamed by the map $M2, producing a new Cayley table $CT3; $M2 is then an isomorphism from $CT1 onto $CT3.

DUCM DUMP MEMORY.

A diagnostic dump of all constants and tables is provided. Note: For Cayley tables, the numbers given in a memory dump will be one greater than the numbers typed in or printed out. This is because ALGBRA handles elements of a Cayley table in a manner compatible with FORTRAN (MADTRAN) conventions. The same will be true for maps.

EGNS GENERATE NORMAL SUBGROUP $S3* FROM ELEMENT $C2 OF $CT1.

The normal subgroup of the group $CT1 generated by the element $C2 is constructed in core and given the name $S3*.

SGSS GENERATE SUBSEMIGROUP $S3* FROM SUBSET $S2 OF $CT1.

The subsemigroup of the semigroup $CT1 (the subgroup of the group $CT1) generated by the subset $S2 of $CT1 is constructed in core and given the name $S3*.

SGLI GENERATE LEFT IDEAL $S3* FROM SUBSET $S2 OF $CT1.

The left ideal of the semigroup $CT1 generated by the subset $S2 of $CT1 is constructed in core and given the name $S3*.

SGRI GENERATE RIGHT IDEAL $S3* FROM SUBSET $S2 OF $CT1.

The right ideal of the semigroup $CT1 generated by the subset $S2 of $CT1 is constructed in core and given the name $S3*.

SGTI GENERATE TWO-SIDED IDEAL $S3* FROM SUBSET $S2 OF $CT1.

The two-sided ideal of the semigroup $CT1 generated by the subset $S2 of $CT1 is constructed in core and given the name $S3*.

SGNS GENERATE NORMAL SUBGROUP $S3* FROM SUBSET $S2 OF $CT1.

The normal subgroup of the group $CT1 generated by the subset $S2 of $CT1 is constructed in core and given the name $S3*.

CTAZ ADD ZERO TO SEMIGROUP $CT1 GIVING $CT2*.

A zero element is added to the semigroup $CT1, giving a new semigroup $CT2*.

CTAU ADD UNIT TO SEMIGROUP $CT1 GIVING $CT2*.

A unit element is added to the semigroup $CT1, giving a new semigroup $CT2*.

LCMA CONSTRUCT LEFT COSET MAP $M3* OF SUBSET $S2 OF $CT1.

The natural map of the group $CT1 into the set of its left cosets modulo the subgroup $S2 is constructed and given the name $M3*.

RCMA CONSTRUCT RIGHT COSET MAP $M3* OF SUBSET $S2 OF $CT1.

The natural map of the group $CT1 into the set of its right cosets modulo the subgroup $S2 is constructed and given the name $M3*.

PHRA CLEAR MEMORY.

All constants and tables are cleared.

ZETE DOES $CT1 HAVE A ZERO, ANSWER $BC2*.

If the semigroup $CT1 contains a zero (an element z such that zs = sz = z for all s) then $BC2* is set to 1; otherwise, it is set to 0.

IDNU HOW MANY IDEMPOTENTS IN SEMIGROUP $CT1, ANSWER $C2*.

The constant $C2* is set to the number of idempotents (elements e with ee = e) in the semigroup $CT1.

EONU SEMIGROUP $CT1, UNIT $C2, IS FOUND TO HAVE $C4* ELEMENTS OF ORDER $C3.

The group $CT1 with the unit element $C2 is searched for all elements of order $C3, and the constant $C4* is set to the number of these elements. (The word #semigroup# in the calling phrase is used for purposes of future expansion of the capabilities of this routine.)

SUGT IS SEMIGROUP $CT1 WITH UNIT $C3 A GROUP, ANSWER $BC2*.

If the semigroup $CT1 is a group, the constant $BC2* is set to 1; otherwise, it is set to 0. The specification of the unit element $C3 saves the search for a unit element that must otherwise be made.

SUGI IS SEMIGROUP $CT1 WITH UNIT $C3 A GROUP, ANSWER $BC2*.

If the semigroup $CT1 is a group, the constant $BC2* is set to 1; and a map $M4* is constructed, giving for each elemtn of $CT1 its inverse. Otherwise, $BC2* is set to 0. The specification of the unit element $C3 saves the search for a unit element that must otherwise be made.

GRTU IS $CT1 A GROUP, ANSWER $BC2*, UNIT $C3*.

If the semigroup$CT1 is a group, the constant $BC2* is set to 1, and the constant $C3* is set to the unit element

of the group. Otherwise, $BC2* is set to 0.

GRUT IS $CT1 A GROUP, ANSWER $BC2*, UNIT $C3*,
INVERSE TABLE $M4*.

If the semigroup $CT1 is a group, the constant $BC2* is set to 1, the constant $C3* is set to the unit element of the group, and a map $M4* is constructed, giving for each element of $CT1 its inverse. Otherwise, $BC2* is set to 0.

UNTE DOES $CT1 HAVE A UNIT, ANSWER $BC2*.

If the semigroup $CT1 contains a unit (an element e such that es = se = s for all s) then the constant $BC2* is set to 1. Otherwise, it is set to 0.

UNTU DOES $CT1 HAVE A UNIT, ANSWER $BC2*, UNIT $C3*.

If the semigroup $CT1 contains a unit, then the constant $BC2* is set to 1, and the constant $C3* is set to the value of the unit. Otherwise, $BC2* is set to 0.

DUSS PRINT SET OF SUBSETS $SS1.

The set of subsets $SS1 is displayed on the console. Each subset is displayed, together with a count of its elements.

ADSM ADD $M3 AS MAP NUMBER $C2 IN SET $SM1.

The map $M3 is added to the set of maps $SM1. If $C2 is greater than the number of maps in $SM1, then $SM1 is expanded.

MASN MAP $M3* IS NUMBER $C2 IN SET $SM1.

The map which is map number $C2 in the set of maps $SM1 is extracted and given the name $M3*.

DUSY SYSTEM DUMP.

A dump is given of all tables within the system in a form which does not depend on their being meaningful. This routine may thus be used to debug the ALGBRA system.

OLCT OUTPUT LISP CAYLEY TABLE $CT1.

The Cayley table $CT1 is output on the console in a standard format compatible with LISP. The 26 x 26 symbols AA through ZZ are used in alphabetical order.

OLCG OUTPUT LISP CAYLEY TABLE $CT1, SYMBOL TABLE $M2.

The Cayley table $CT1 is output on the console in a standard format compatible with LISP. Symols are used from the symbol table $M2, in the order given there.

ILCT INPUT LISP CAYLEY TABLE $CT1*, ORDER $C2.

The Cayley table $CT1* is input from the console in a standard format compatible with LISP. The 26 x 26 symbols AA through ZZ are used in alphabetical order.

ILCG INPUT LISP CAYLEY TABLE $CT1*, ORDER $C2,
SYMBOL TABLE $M3*.

The Cayley table $CT1* is input from the console in a standard format compatible with LISP. The symbols used in the table as it is input are stored in the symbol table $M3*.

DOCT DISK OUTPUT CAYLEY TABLE $CT1.

The Cayley table $CT1 is output to disk as a file with primary name $CT1 and secondary name 'MATH', in a standard format compatible with LISP. The 26 x 26 symbols AA through ZZ are used, in alphabetical order.

DOCG DISK OUTPUT CAYLEY TABLE $CT1, SYMBOL TABLE $M2.

The Cayley table $CT1 is output to disk as a file with primary name $CT1 and secondary name 'MATH', in a standard format compatible with LISP. Symbols are used from the symbol table $M2, in the order given there.

DICT DISK INPUT CAYLEY TABLE $CT1*, ORDER $C2.

The Cayley table $CT1* is input from a disk file with primary name $CT1* and secondary name 'MATH', in a standard format compatible with LISP. The 25 x 26 symbols AA through ZZ are used, in alphabetical order.

DICG DISK INPUT CAYLEY TABLE $CT1*, ORDER $C2,
SYMBOL TABLE $M3*.

The Cayley table $CT1* is input from a disk file with primary name $CT1* and secondary name 'MATH', in a standard format compatible with LISP. The symbols used in the table

as it is input are stored in the symbol table SM3*.

CTSM CAYLEY TABLE $CT2* OF SEMIGROUP OF MAPS $SM1.

The set of finite maps $SM1 is assumed to be a semigroup under composition. The Cayley table of this semigroup is constructed and given the name $CT2*. The finite maps may be permutations, in which case the semigroup is a group.

FSCT CAYLEY TABLE $CT3* OF FACTOR SEMIGROUP $CT1/$S2.

The subset $S2 of the semigroup $CT1 is assumed to be a two-sided ideal. The corresponding factor semigroup is constructed and given the name $CT3*.

IPGH INPUT STATE GRAPH $SM1* WITH $C2 STATES AND $C3 INPUTS.

A sequential machine $SM1 (thought of as the set of finite maps of the set of states into itself induced by the inputs, with outputs ignored) is input from the console. The number of states is $C2, and the number of inputs is $C3.

RSSG STRUCTURE GROUP $S2* OF SIMPLE SEMIGROUP $CT1.

The Rees structure group of the simple semigroup $CT1 is constructed as a subset $S2* of $CT1. This subgroup is of the form eSe, where S is the semigroup and e is any idempotent of S.

SGMA GENERATE SEMIGROUP OF MAPS $SM2 FROM SET OF MAPS $SM1.

The set of finite maps $SM1 generates a semigroup of finite maps under composition. This semigroup is constructed and given the name $SM2.

OWTA OPEN TO WRITE FILE $F1.

A multiple table storage file is opened to write. The file has primary name $F1 and secondary name 'MATH'.

WRTA WRITE TABLE $T2 ONTO FILE $F1.

The table $T2, of whatever type, is written in standard format, together with its name, onto the disk file $F1, which must have previously been opened.

CWTA CLOSE WRITE FILE $F1.

The multiple table storage file with primary name $F1 and secondary name 'MATH', which has been written on, is closed.

ORTA OPEN TO READ FILE $F1.

The disk file with primary name $F1 and secondary name 'MATH' is opened to read.

RDTA READ FILE $F1, END-OF-FILE FLAG $SC2*.

One table is read from the multiple table storage file with primary name $F1 and secondary name 'MATH'. This file

must nave been opened to read. The table name is contained in the file itself, along with the dimensions.

CRTA CLOSE READ FILE $F1.

The multiple table storage file with primary name $F1 and secondary name 'MATH', which has been read, is closed.

NWFN NEW FUNCTION.

A new function is added to the ALGBRA system. The function is assumed to exist as a BSS file on disk, with a given entry point name. More than one entry point of the same routine may be entered as a function in several calls to NWFN.

DUCP PRINT THE CALLING PHRASE FOR THE ENTRY $RN1.

The name $RN1 is assumed to be a routine name (i.e., one of the underlined names described in this section, or the name of a new routine added with NWFN). The calling phrase is displayed on the console.

* * * * * * * * * * * * *

APPENDIX B

Primitive Functions and their Frames

For the Inferential Compiler Source Language

FUNCTION NAME	VALUE TYPE	FRAME
	Boolean	($BC1 OR $BC2)
	Boolean	($BC1 AND $BC2)
	Boolean	(NOT $BC1)
	Boolean	(IF $BC1, THEN $BC2)
	Boolean	(FOR ALL $T1 IN $F2, $BC3)
	Boolean	(THERE EXISTS $T1 IN $F2 SUCH THAT $BC3)
ALCT	Group	(THE ALTERNATING GROUP ON $C1 LETTERS)
SYCT	Group	(THE SYMMETRIC GROUP ON $C1 LETTERS)
DHCT	Group	(THE DIHEDRAL GROUP ON $C1 VERTICES)
CYCT	Group	(THE CYCLIC GROUP OF ORDER $C1)
LACT	Semigroup	(THE LEFT ZERO SEMIGROUP OF ORDER $C1)
RACT	Semigroup	(THE RIGHT ZERO SEMIGROUP OF ORDER $C1)
NACT	Semigroup	(THE NULL SEMIGROUP OF ORDER $C1)
CTDP	Semigroup	(THE DIRECT PRODUCT OF $CT1 AND $CT2)
DASS	Set of Ss.	(THE SET OF ALL SUBSEMIGROUPS OF $CT1)
DALI	Set of Ss.	(THE SET OF ALL LEFT IDEALS OF $CT1)
DARI	Set of Ss.	(THE SET OF ALL RIGHT IDEALS OF $CT1)
DATI	Set of Ss.	(THE SET OF ALL TWO-SIDED IDEALS OF $CT1)
DANS	Set of Ss.	(THE SET OF ALL NORMAL SUBGROUPS OF $CT1)
DMSS	Set of Ss.	(THE SET OF ALL MAXIMAL SUBSEMIGROUPS OF $CT1)
DMLI	Set of Ss.	(THE SET OF ALL MAXIMAL LEFT IDEALS OF $CT1)
DMRI	Set of Ss.	(THE SET OF ALL MAXIMAL RIGHT IDEALS OF $CT1)
DMTI	Set of Ss.	(THE SET OF ALL MAXIMAL TWO-SIDED IDEALS OF $CT1)
DMNS	Set of Ss.	(THE SET OF ALL MAXIMAL NORMAL SUBGROUPS OF $CT1)
EGSS	Subset	(THE SUBSEMIGROUP GENERATED BY $C1 .MEMBOF. $CT2)
EGLI	Subset	(THE LEFT IDEAL GENERATED BY $C1 .MEMBOF. $CT2)
EGRI	Subset	(THE RIGHT IDEAL GENERATED BY $C1 .MEMBOF. $CT2)
EGTI	Subset	(THE TWO-SIDED IDEAL GENERATED BY $C1 .MEMBOF. $C
EGNS	Subset	(THE NORMAL SUBGROUP GENERATED BY $C1 .MEMBOF. $C
U2ST	Subset	(THE UNION OF $S1 AND $S2)
I2ST	Subset	(THE INTERSECTION OF $S1 AND $S2)
CPST	Subset	(THE COMPLEMENT OF $S1)
FGCT	Group	(THE FACTOR GOUP $CT1/$CT2)
SGSS	Subset	(THE SUBSEMIGROUP GENERATED BY $S1 .CTNDIN. $CT2)
SGLI	Subset	(THE LEFT IDEAL GENERATED BY $S1 .CTNDIN. $CT2)
SGRI	Subset	(THE RIGHT IDEAL GENERATED BY $S1 .CTNDIN. $CT2)
SGTI	Subset	(THE TWO-SIDED IDEAL GENERATED BY $S1 .CTNDIN. $C
SGNS	Subset	(THE NORMAL SUBGROUP GENERATED BY $S1 .CTNDIN. $C
CTAZ	Semigroup	($CT1 .UNION. 0)
CTAU	Semigroup	($CT1 .UNION. 1)
LCMA	Map	(THE LEFT COSET MAP OF $CT1 MOD $S2)

RCMA	Map	(THE RIGHT COSET MAP OF $CT1 MOD $S2)
FSCT	Semigroup	(THE FACTOR SEMIGROUP $CT1/$CT2)
RSCG	Group	(THE REES STRUCTURE GROUP OF $CT1)
ABTE	Boolean	($CT1 IS ABELIAN)
GRTE	Boolean	($CT1 IS A GROUP)
SGTE	Boolean	($CT1 IS A SEMIGROUP)
SGIT	Boolean	($CT1 AND $CT2 ARE ISOMORPHIC)
HMTE	Boolean	($M3 IS A HOMOMORPHISM FROM $CT1 TO $CT2)
MMTE	Boolean	($M3 IS A MONOMORPHISM FROM $CT1 TO $CT2)
EMTE	Boolean	($M3 IS AN EPIMORPHISM FROM $CT1 TO $CT2)
IMTE	Boolean	($M3 IS AN ISOMORPHISM FROM $CT1 TO $CT2)
ZETE	Boolean	($CT1 HAS A ZERO)
UNTE	Boolean	($CT1 HAS A UNIT)

* * * * * * * * * * * *