TECHNICAL REPORT TR-200
NGL 21-002-008 ANI
N00014-67-A-0239-n021
(NR-044-431)

F G R A A L
TECHNICAL DOCUMENTATION

BY
C. K. MESZTENYI
H. BREITENLOHNER
J. C. YEH

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND
This report describes the implementation of FGRAAL, a FORTRAN extended graph algorithmic language (technical report TR-179) for the UNIVAC 1108. The report contains the description of the implemented data structures for sets, lists, and graphs. It summarizes the changes made for the RALPH compiler to accommodate the special statements of FGRAAL. It gives the calling sequences generated by the changed compiler. It describes the object time subroutine package.

This research was supported in part by the Office of Naval Research under grant N00014-67-A-0239-021(NR-044-431), and the National Aeronautics and Space Administration under grant NGL 21-002-008.
# TABLE OF CONTENTS

1. INTRODUCTION 1-1
2. HINTS FOR EFFICIENT PROGRAMMING IN FGRAAL 2-1
   2.1. EFFICIENT USE OF SETS 2-1
   2.2. SET EXPRESSIONS AND ASSIGNMENTS 2-2
   2.3. PROPERTIES 2-3
3. FGRAAL STORAGE STRUCTURE ON THE UNIVAC 1108. 3-1
   3.1. DYNAMIC STORAGE 3-1
   3.2. DATA REPRESENTATION 3-5
   3.2.1. UNIVERSAL SEQUENCE AND THE PROPERTY BLOCKS 3-5
   3.2.2. SETS 3-7
   3.2.3. STAQUES (LISTS) 3-12
   3.2.4. PROPERTIES 3-14
   3.2.5. GRAPH STRUCTURES 3-16
   3.2.6. TYPE ASSIGNMENTS 3-18
   3.2.7. STRUCTURE OF HEADERS 3-19
   3.3. REPRESENTATION OF A CONTRACTED GRAPH 3-20
4. EXAMPLE OF A GRAPH STRUCTURE 4-1
5. THE FGRAAL COMPILER 5-1
   5.1. THE RALPH COMPILER 5-1
   5.2. MODIFICATIONS TO RALPH 5-2
   5.2.1. DATA TYPES 5-2
   5.2.2. IMPLEMENTATION OF NEW STATEMENTS 5-2
   5.2.3. IMPLEMENTATION OF NEW SYNTACTIC FEATURES 5-3
   5.2.4. HANDLING OF TEMPORARIES 5-3
   5.3. OPTIMIZATION 5-4
6. COMPILER GENERATED INSTRUCTION SEQUENCES 6-1
   6.1. DECLARATION STATEMENTS 6-1
   6.2. CALLING SEQUENCES 6-2
   6.3. FREEING A SET OR LIST 6-2
   6.4. SET - OPERATIONS, -RELATIONS, -ASSIGNMENT 6-3
   6.5. SPECIAL SET FUNCTIONS 6-3
   6.5.1. CREATE FUNCTION 6-3
   6.5.2. SUBSET FUNCTION 6-4
   6.5.3. ELT, INDEX, SIZE AND PARITY FUNCTIONS 6-5
   6.6. PROPERTY ASSIGNMENT AND RETRIEVAL 6-5
   6.7. LIST ASSIGNMENT AND LIST FUNCTIONS 6-6
   6.7.1. LIST ASSIGNMENT STATEMENT 6-6
   6.7.2. LIST FUNCTIONS 6-6
   6.8. GRAPH OPERATIONS 6-7
   6.8.1. ASSIGN STATEMENT 6-7
   6.8.2. DETACH STATEMENT 6-8
   6.8.3. GRAPH FUNCTIONS 6-8
   6.9. ITERATIVE STATEMENTS 6-9
   6.9.1. WHILE STATEMENT 6-9
   6.9.2. FOR ALL STATEMENT 6-9
   6.10. REMOVE STATEMENTS 6-10
7. LIBRARY PROGRAMS 7-1
8. APPENDIX, SUMMARY OF FGRAAL 8-1
1. INTRODUCTION


THE UNIVAC 1108 IS A WORD ADDRESSABLE COMPUTER WITH 36 BIT WORDS. IT HAS PARTIAL WORD INSTRUCTIONS THUS CERTAIN DATA PACKING INTO ONE WORD CAN BE IMPLEMENTED EFFICIENTLY. IT ALSO HAS A SET OF MASKED SEARCH INSTRUCTIONS WHICH ARE USED FOR CERTAIN SET OPERATIONS IN FGRAAL VERY EFFICIENTLY.

IN CHAPTER 2 SOME HINTS ARE GIVEN FOR ACHIEVING EFFICIENCY BOTH IN STORAGE AND EXECUTION TIME, BASED ON THE IMPLEMENTED DATA STRUCTURE.


WE TAKE THIS OPPORTUNITY TO THANK PROF. V. R. BASILI FOR HIS HELP AND ADVICE IN PREPARING THIS REPORT.
2. HINTS FOR EFFICIENT PROGRAMMING IN FGRAAL

THE READER SHOULD BE ABLE TO GAIN SUFFICIENT INFORMATION ABOUT MODIFYING HIS PROGRAM TO ACHIEVE MAXIMUM EFFICIENCY UNDER THE GIVEN IMPLEMENTATION BY READING THIS CHAPTER ALONE, HOWEVER TO UNDERSTAND FULLY WHY THESE RECOMMENDATIONS FOR EFFICIENCY ARE VALID, THE REST OF THIS REPORT MUST BE READ.

THE PRIMARY GOAL OF FGRAAL IS THE EASY IMPLEMENTATION OF GRAPH ALGORITHMS ARISING IN APPLICATION, USUALLY THE EASY IMPLEMENTATION OF THE ALGORITHM ALSO MEANS THAT THE PROGRAM CAN BE EASILY DE-BUGGED AND DOCUMENTED. UNFORTUNATELY AN EASILY IMPLEMENTED ALGORITHM CAN BE VERY INEFFICIENT IN MEMORY SIZE REQUIREMENTS AND IN EXECUTION TIME, OBVIOUSLY, THE EFFICIENCY OF FGRAAL PROGRAMS DEPEND ON THE ARRANGEMENT OF SPECIAL DATA (SETS, PROPERTIES, ETC.) IN THE IMPLEMENTED FGRAAL SYSTEM, THE ARRANGEMENT OF THESE DATA ARE DESCRIBED IN THE NEXT CHAPTER, IN THIS CHAPTER, WE GIVE SOME HINTS TO ACHIEVE EFFICIENT PROGRAMS BASED ON THE IMPLEMENTED DATA STRUCTURE IN FGRAAL.

GENERALLY WE SUGGEST THAT THE PROGRAM BE WRITTEN AND DEBUGGED WITH AN EFFICIENT IMPLEMENTATION IN MIND, THIS DOES NOT MEAN THAT THE FIRST VERSION BE AS EFFICIENT AS POSSIBLE, ONLY THAT IT LEND ITSELF TO EASY IMPROVEMENT. ONCE A PROGRAM IS DEBUGGED, THE EFFICIENCY OF THE PROGRAM CAN BE IMPROVED BY INSERTING FORTRAN EQUIVALENCE AND DEFINE STATEMENTS, LINEAR ARRAYS AND SOME CHANGE OF THE ORIGINAL PROGRAM. THESE TYPES OF IMPROVEMENT WILL BE ILLUSTRATED IN THE FOLLOWING SECTIONS.

2.1. EFFICIENT USE OF SETS

* STORAGE CAN BE SAVED DURING EXECUTION
* OF A PROGRAM IF SETS ARE MADE EMPTY
* AS SOON AS THEY ARE NOT NEEDED.

THIS IS BECAUSE EACH SET WITH NO ELEMENT (EMPTY) OR WITH ONE ELEMENT (ATOMIC) USES ONLY ONE MEMORY LOCATION WHICH WAS ASSIGNED TO IT BY THE COMPILER.

* IT IS ADVANTAGEOUS TO KEEP THE NUMBER
* OF NOT EMPTY OR ATOMIC SETS UNDER 15
* AT ANY TIME DURING EXECUTION.

SETS WITH TWO OR MORE ELEMENTS ARE REPRESENTED IN EITHER COLUMN OR BLOCK FORM (SEE NEXT CHAPTER FOR DETAILS). THE COLUMN FORM REPRESENTATION OF SETS ARE ATTACHED TO THE UNIVERSAL SEQUENCE AND AS SUCH THEY DO NOT REQUIRE EXTRA STORAGE SPACE. FGRAAL PROVIDES UP TO 15 SETS TO BE REPRESENTED IN COLUMN FORM, AS SOON AS MORE
THAN 15 SETS HAVE TWO OR MORE ELEMENTS, SOME OF THE SETS WILL AUTOMATICALLY BE TRANSFORMED INTO BLOCK FORM. A SET WITH N ELEMENTS OCCUPIES 2*N/3 MEMORY LOCATIONS WHEN IT IS IN BLOCK FORM. THE RESULT OF A SET OPERATION IS ALWAYS PLACED IN COLUMN FORM UNLESS IT IS EMPTY OR ATOMIC. SINCE SETS IN BLOCK FORM OCCUPY EXTRA SPACE AND REQUIRE EXTRA TIME TO GET THEM INTO THIS FORM, ONE AVOIDS THIS OVERHEAD BY KEEPING THE NUMBER OF NOT EMPTY OR ATOMIC SETS UNDER 15 AT ANY TIME DURING EXECUTION.

ONE CAN KEEP DOWN THE NUMBER OF SETS BY USING EQUIVALENCE STATEMENTS, E.G. ASSUME THAT THE SETS S AND T ARE USED IN TWO NON-OVERLAPPING SEGMENTS OF A PROGRAM. AFTER DEBUGGING THE PROGRAM, ONE CAN MAKE THE TWO SETS EQUIVALENT BUT STILL KEEP THEIR NAME:

```
SET S, T
EQUIVALENCE (S,T)
(SEMGNENT 1 USES S)
(SEMGNENT 2 USES T)
```

2.2. SET EXPRESSIONS AND ASSIGNMENTS

* THE MODIFICATION OF A SET BY OTHER SETS
* IS MOST EFFICIENTLY IMPLEMENTED BY A SEQUENCE OF SIMPLE SET ASSIGNMENT OF THE FORM S = S .OP. X, RATHER THAN BY USE OF A COMPLEX EXPRESSION.

THE PRESENT IMPLEMENTATION OF FGRAAL DOES NOT MAKE ANY ATTEMPT TO OPTIMIZE THE EVALUATION OF COMPLEX SET EXPRESSIONS. E.G. THE SET EXPRESSION

```
(S .DF. X) .UN. Y
```

IS EVALUATED FIRST BY USING A TEMPORARY SET T' FOR

```
S .DF. X --> T'
```

AND THE FINAL RESULT IS OBTAINED BY THE EVALUATION OF

```
T' .UN. Y
```

A VERY FREQUENTLY OCCURRING SET ASSIGNMENT STATEMENT IS RECOGNIZED BY THE FGRAAL COMPILER. THIS STATEMENT,

```
S = S .OP. X
```

MODIFIES A SET S WITH ANOTHER SET X BY TAKING THE UNION, DIFFERENCE, INTERSECTION OR SYMMETRIC SUM OF THEM, I.E. OP IS ANY
ONE OF UN, DF, IT OR SM. THIS STATEMENT IS IMPLEMENTED SUCH THAT THE RESULT OF THE SET EXPRESSION, $S_{OP,X}$, IS GENERATED IN THE SPACE PROVIDED FOR $S$. THIS FEATURE OF FGRAAL MAKES IT MORE EFFICIENT TO USE MORE STATEMENTS INSTEAD OF ONE STATEMENT WITH COMPLEX EXPRESSION. E.G., THE EXECUTION OF THE TWO STATEMENTS,

\[
S = S_{DF,X} \\
S = S_{UN,Y}
\]

IS MORE EFFICIENT THEN THE EXECUTION OF THE COMBINED STATEMENT

\[
S = (S_{DF,X})_{UN,Y}
\]

2.3. PROPERTIES

* USING FORTRAN ARRAYS FOR STORING
* PROPERTY VALUES CAN BE MORE EFFICIENT
* THAN USING DECLARED PROPERTIES.

IN THE IMPLEMENTED FGRAAL DATA STRUCTURE, THE PROPERTY VALUES WITH PROPERTY IDENTIFIERS ARE LINKED TO THE ELEMENTS IN THE UNIVERSAL SEQUENCE. THIS REQUIRES TWICE AS MANY MEMORY LOCATIONS AS THE ACTUAL MEMORY SPACE NEEDED TO STORE THE PROPERTY VALUES ONLY. THE RETRIEVAL AND MODIFICATION OF A PROPERTY VALUE OF AN ELEMENT IS ACCOMPLISHED BY A SEARCH IN THE LINKED PROPERTY BLOCKS OF THE ELEMENT. CLEARLY, THIS REQUIRES MUCH MORE TIME THEN THE RETRIEVAL AND MODIFICATION OF PROPERTY VALUES IF THEY COULD BE ARRANGED IN A FORTRAN TYPE OF LINEAR ARRAY. SINCE THE ELEMENTS OF THE UNIVERSAL SEQUENCE ARE IDENTIFIED BY POSITIVE INTEGERS, ATOMIC SETS CAN BE USED AS INDICES FOR DIMENSIONED VARIABLES, THIS FEATURE OF FGRAAL MAKES IT POSSIBLE TO CHANGE PROPERTY VARIABLES INTO FORTRAN TYPE DIMENSIONED VARIABLES TO ACHIEVE BETTER EFFICIENCY.

THERE ARE SOME RESTRICTIONS ON THE USE OF LINEAR ARRAYS FOR PROPERTIES:

(1) THE USE OF LINEAR ARRAYS FOR PROPERTY VALUES DEFINES THE PROPERTY FOR ALL VALID INDEX VALUES CORRESPONDING TO ELEMENTS IN THE UNIVERSAL SEQUENCE. A SPECIAL VALUE MUST BE USED IF ONE WANTS TO MAKE DISTINCTION FOR UNDEFINED PROPERTY FOR AN ELEMENT IN THIS REGION.

(2) THE ELEMENTS FOR WHICH THE PROPERTY IS DEFINED SHOULD OCCUPY A CONTIGUOUS AREA IN THE UNIVERSAL SEQUENCE IF A LINEAR ARRAY IS USED. FURTHERMORE, THE FIRST OF THESE ELEMENTS, AND THE SIZE OF THE REGION SHOULD BE KNOWN.

WITH THESE RESTRICTIONS IN MIND, ONE COULD ATTEMPT TO CHANGE A DEBUGGED FGRAAL PROGRAM TO ACHIEVE HIGHER EFFICIENCY. THE STEPS TO
BE TAKEN FOR THIS CHANGE ARE AS FOLLOWS:
ASSUME THAT THE PROPERTY P IS USED FOR ELEMENTS IN A CONTIGUOUS
AREA OF THE UNIVERSAL SEQUENCE, THE FIRST OF THESE ELEMENTS IS X1,
AND THE SIZE OF THIS AREA IS NX.
THEN REPLACE THE ORIGINAL DECLARATION STATEMENT
PROPERTY P
WITH THE FOLLOWING TWO STATEMENTS
DIMENSION PA(NX)
DEFINE P(X) = PA(X-X1+1)
ANY APPEARANCE OF THE STATEMENT,
REMOVE P
MUST BE REPLACED WITH A SMALL LOOP
DO ZZZ 1=1,NX
ZZZ PA(I) = V
WHERE V IS A SPECIALLY DEFINED VALUE FOR 'UNDEFINED' PROPERTY. IN
THIS CASE, THE SPECIAL FUNCTION,
CHECK(P,X)
SHOULD BE REPLACED BY THE EXPRESSION
P(X) ,NE. V
3. FGRAAL STORAGE STRUCTURE ON THE UNIVAC 1108.

This chapter describes the representation of the special data structures during the execution of an FGRAAL program. In Section 1, the dynamic storage is described. This storage holds all the special data of the FGRAAL programs, except for the headers corresponding to the declared variables: sets, lists, properties and graphs. Section 2 describes the representation of the universal sequence, and the sets, lists, properties and graphs in the dynamic storage. Section 3 shows the representation of a contracted graph, that is, the graph representation modified by the library routine CONTR.

3.1. DYNAMIC STORAGE.

The dynamic storage area is the working space of the FGRAAL system and it is a contiguous array of computer words where all data values except those assigned by the compiler are stored. The size of this storage area varies and can be optionally designated by the user. Logically speaking, the dynamic storage area can be subdivided into three dynamic, variable-sized portions. The upper portion is the universal sequence, the middle portion is the unused storage area and the lower portion is the linked-block area. The two end portions, starting from the two end boundaries of the dynamic storage area expand their sizes by seizing storage from the middle portion. Whenever these two end-portions meet (and the middle portion disappears) the available storage in the dynamic storage area is exhausted and program execution is terminated with an appropriate message.

In FGRAAL, from the data structure point of view, there are two types of storage structure: (1) linear array, (2) linked-block. The linear array is used mainly for the representation of the universal sequence, while the linked-block structure is used for the representation of all other types of data (e.g., stack, graph, property, etc.). The linear array (or the universal sequence) is located on the upper portion of the dynamic storage area, the linked-blocks are located on the lower portion of the dynamic storage area. However, the storage management routine keeps track of the storage allocated to each portion and prevents them from overlapping into each other. The storage management routine also maintains a set of global pointer variables which are updated whenever the status of the dynamic storage changes. The values of these pointers are either relative or absolute addresses. A relative address, i.e., refers to the location of D(i), where array D is the dynamic storage area, an absolute address refers to the ac-
TUAL LOCATION IN THE COMPUTER MEMORY. THE INITIAL (I.E. BEFORE ANY DATA IS CREATED) VALUES OF THESE POINTERS ARE AS FOLLOWS:

(NOTE: LOC(A) DENOTES THE ABSOLUTE LOCATION OF VARIABLE A).

<table>
<thead>
<tr>
<th>NAME</th>
<th>INITIAL VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NU - LAST ELEMENT CREATED</td>
<td>0</td>
</tr>
<tr>
<td>ND - SIZE OF THE DYNAMIC STORAGE</td>
<td>OPTION X 4096</td>
</tr>
<tr>
<td>NF - FIRST AVAILABLE BLOCK</td>
<td>LOC(D(ND-1))</td>
</tr>
<tr>
<td>NL - LAST AVAILABLE BLOCK</td>
<td>LOC(D(ND-1))</td>
</tr>
<tr>
<td>D - DYNAMIC STORAGE, D(1),I=1,...,ND</td>
<td>D(ND-1)=0</td>
</tr>
</tbody>
</table>

INITIALLY, THE ENTIRE DYNAMIC STORAGE AREA CAN BE VIEWED AS A LARGE UNUSED BLOCK. WHENEVER A NEW UNIVERSAL ELEMENT IS CREATED BY THE EXECUTION OF 'CREATE' STATEMENTS, ONE COMPUTER WORD IS CARVED OUT FROM THE TOP OF THIS BLOCK AND THE POINTER NU IS INCREASED BY 1. THE CREATION OF NEW ELEMENTS MAY BE CONTINUED UNTIL THE LOC(D(NU)) IS EQUAL TO NL (I.E. THE AVAILABLE SPACE IS EXHAUSTED). THE ALLOCATION OF 2-WORD BLOCKS IS CARRIED OUT BY TAKING AWAY THE BOTTOM TWO WORDS FROM THE UNUSED BLOCK. WHENEVER A BLOCK IS RELEASED (I.E. AVAILABLE FOR OTHER USES) IT IS LINKED TOGETHER TO FORM A SINGLY-LINKED AVAILABLE-BLOCK LIST WITH NF AS ITS LIST HEADER. THE SUBSEQUENT REQUESTS FOR A 2-WORD BLOCK WILL BE SATISFIED THROUGH THE ALLOCATION OF A BLOCK FROM THIS AVAILABLE-BLOCK LIST. THE NEW BLOCK WILL NOT BE CREATED OUT OF THE UNUSED PORTION OF THE DYNAMIC STORAGE AREA UNTIL THE AVAILABLE-BLOCK LIST IS EXHAUSTED. IN OTHER WORDS, THE STORAGE MANAGEMENT ROUTINE TRIES TO SATISFY THE REQUEST FOR A 2-WORD BLOCK BY ASSIGNING AN EXISTING AVAILABLE BLOCK BEFORE AN ATTEMPT IS MADE TO ALLOCATE A NEW BLOCK. THE ALGORITHM FOR ALLOCATING A 2-WORD BLOCK (WHICH IS TO BE POINTED TO BY 'AVAIL') IS AS FOLLOWS:

1. THE AVAILABLE-BLOCK LIST IS NOT EMPTY
   IF ( NF = 0 ) THEN GO TO 2.
   ELSE
   \[ \text{AVAIL} = NF \]
   \[ \text{LINK}(\text{NF}) = \text{LINK}(\text{LINK}(\text{NF})) \]
   RETURN

2. THE AVAILABLE-BLOCK LIST IS EMPTY, CHECK THE UNUSED PORTION
   IF ( NL < LOC(D(NU)) + 2 ) THEN GO TO 3.
   ELSE
   \[ \text{AVAIL} = NF \]
   \[ \text{LINK}(\text{NF}) = \text{LOC}(\text{NF}) - 2 \]
   \[ \text{NL} = \text{LINK}(\text{NF}) \]
   RETURN

3. THE AVAILABLE SPACE IS EXHAUSTED
   OUTPUT MESSAGE
TERMINATE PROGRAM

THE ALGORITHM TO FREE A 2-WORD BLOCK \( (D(K), D(K+1)) \) IS:

1. LINK THE 2-WORD BLOCK INTO THE AVAILABLE-BLOCK LIST
   \[ \text{LINK}(D(K)) = \text{LINK}(NF) \]

2. UPDATE THE LIST HEADER, NF
   \[ \text{LINK}(NF) = \text{LOC}(D(K)) \]
   RETURN

THE SCHEMATIC DIAGRAM OF THE DYNAMIC STORAGE AREA IS GIVEN IN FIGURE 1. \((K1, K2, \ldots, KM)\) ARE BLOCKS IN THE AVAILABLE-BLOCK LIST.
FIGURE 1. DYNAMIC STORAGE AREA.
3.2. DATA REPRESENTATION

All links between the blocks representing data are with absolute addresses except references to the elements in the universal sequence where their relative addresses are used, e.g. 'I' refers to the element in the universal sequence D(I).

3.2.1. UNIVERSAL SEQUENCE AND THE PROPERTY BLOCKS

The universal sequence is a sequence of elements generated by the FGRAAL 'CREATE' functions. It is the union of all set elements in the FGRAAL system (e.g. nodes(G), arcs(G), elements of any set), hence it is called the universal sequence. This sequence is implemented as a linear array located on the upper end of the dynamic storage area, D(I), I=1, 2, ..., NU. Each memory word in this array, called an element ID word, represents one element in the universal sequence. The relative location of an element in this sequence is called its universal sequence number. Different types of properties may be assigned to an atomic set (hence to its only element in the set) by the property assignment or conditional 'CREATE' statements in FGRAAL. The properties associated with an element are implemented as a property-ring where its element ID word is a ring header and property blocks are the ring elements.

Each element ID word is divided into two halfword fields. The left half of the element ID word is reserved for the representation of sets in column form (see sets). The right half of the element ID word contains the address pointer to a linked list of property blocks which is associated with this element (if there is one), or the complement of its universal sequence number to signify the nonexistence of the property list. The link field of the last property block on the property list contains the complement of the universal sequence number (which points indirectly to the element ID word). The complement is necessary to signify the end of a property list. The element ID word together with its associated property-list form a property-ring where element ID word is its ring header. Figure 3 shows the layout of the universal sequence and the property block lists.
<table>
<thead>
<tr>
<th>UNIVERSAL SEQUENCE</th>
<th>PROPERTY BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>D(I) * * AI1 *</td>
<td>AI1 ************</td>
</tr>
<tr>
<td>* * * * * * *</td>
<td>* * AI2 * ..</td>
</tr>
<tr>
<td>* * * * * * *</td>
<td>* * -I *</td>
</tr>
<tr>
<td>D(J) * * AJ1 *</td>
<td>AJ1 ************</td>
</tr>
<tr>
<td>* * * * * * *</td>
<td>* * AJ2 * ..</td>
</tr>
<tr>
<td>* * * * * * *</td>
<td>* * -J *</td>
</tr>
<tr>
<td>D(K) * * -K *</td>
<td>AJN ************</td>
</tr>
</tbody>
</table>

**Figure 3. Universal Sequence and Property Blocks**
3.2.2. SETS

IN THE FGRAAL SYSTEM, THERE ARE THREE CATEGORIES OF SETS: EMPTY, ATOMIC, AND COMPOSITE. AN EMPTY SET IS A SET WHICH CONTAINS NO ELEMENTS. AN ATOMIC SET IS A SET WHICH CONTAINS EXACTLY ONE ELEMENT. A COMPOSITE SET IS A SET WHICH IS THE UNION OF A NUMBER OF ATOMIC SETS. SETS, REGARDLESS OF THEIR CATEGORIES, ARE ALWAYS REPRESENTED WITH A SET HEADER. IF A SET IS EMPTY OR ATOMIC, IT WILL BE REPRESENTED BY A SET HEADER ALONE. HOWEVER, IF A SET IS A COMPOSITE SET THEN IT WILL BE REPRESENTED BY A SET HEADER TOGETHER WITH A LIST OF SET ELEMENTS. THERE ARE TWO WAYS OF REPRESENTING A LIST OF SET ELEMENTS: (1) LINKED-BLOCK REPRESENTATION AND (2) COLUMN REPRESENTATION. A SET WHOSE ELEMENTS ARE REPRESENTED IN THE LINKED-BLOCK FORM IS SOMETIMES CALLED BLOCK SET. A SET REPRESENTED IN COLUMN FORM IS SOMETIMES CALLED COLUMN SET.

THE SET HEADER OF A DECLARED SET IS A MEMORY WORD ASSIGNED BY THE COMPILER, WHICH IS LOCAL TO THE SUBPROGRAM WHERE IT HAD BEEN DECLARED. DURING THE PROGRAM EXECUTION, THE LOCATION OF THIS ASSIGNED MEMORY WORD (OR SET HEADER) BECOMES THE UNIQUE IDENTIFIER FOR THE ASSOCIATED, DECLARED SET. THE CONTENTS OF THE SET HEADER IDENTIFIES ITS ASSOCIATED SET AS: EMPTY, ATOMIC, OR COMPOSITE. FOR A COMPOSITE SET, THERE IS A LIST OF SET ELEMENTS ASSOCIATED WITH ITS SET HEADER. THE LINKED-BLOCK REPRESENTATION CONSISTS OF SINGLY LINKED 2-WORD BLOCKS WHICH CONTAIN THE SET ELEMENTS. THE COLUMN REPRESENTATION CONSISTS OF THREE MAJOR COMPONENTS: (1) ASSIGNMENT OF A COLUMN POSITION IN THE UNIVERSAL SEQUENCE TO THIS SET, (2) SETTING OF THE ASSIGNED COLUMN POSITION IN THE UNIVERSAL ELEMENT TO INDICATE THE ELEMENT IS ALSO CONTAINED IN THIS SET, AND (3) SUPPLEMENTARY INFORMATION CONCERNING THIS SET (E.G. NO. OF ELEMENTS, FIRST AND LAST ELEMENT, ETC.). IF A SET IS EMPTY OR ATOMIC THEN IT IS SUFFICIENT TO REPRESENT THE SET SOLELY BY ITS ASSOCIATED SET HEADER. THE REPRESENTATIONS OF THESE SETS ARE GIVEN IN FIGURE 4. THE EMPTY SET IS REPRESENTED BY A SET HEADER CONTAINING ALL 0'S. THE ATOMIC SET WITH THE ELEMENT WHOSE UNIVERSAL SEQUENCE NUMBER IS 1, IS REPRESENTED BY A SET HEADER WHICH CONTAINS THE NUMBER 1 IN ITS RIGHT HALF.
FIGURE 4. EMPTY AND ATOMIC SET

FOR A COMPOSITE SET, THE SET HEADER IS DIVIDED INTO TWO HALF-
WORD FIELDS. THE LEFT HALF CONTAINS THE ADDRESS POINTER TO THE
LIST OF SET ELEMENTS BELONGING TO THIS SET, WHILE THE RIGHT HALF
CONTAINS 0'S. THE LIST OF SET ELEMENTS REPRESENTED IN
LINKED-BLOCK FORM ARE ARRANGED IN A SINGELY-LINKED LIST CONSIS-
TING OF 2-WORD BLOCKS. EACH 2-WORD BLOCK IS SUBDIVIDED INTO 4
HALFWORD FIELDS. THE SECOND HALFWORD FIELD (OR THE RIGHT HALF OF
THE FIRST WORD) IS THE LINK FIELD WHICH CONTAINS ADDRESS POINTER
TO THE NEXT 2-WORD BLOCK (IF THERE IS ONE) OR IT CONTAINS 0'S, IF
THIS BLOCK IS THE LAST ONE ON THE LINKED LIST. THE REMAINING
THREE FIELDS OF EACH BLOCK, CALLED ELEMENT FIELDS, CONTAIN SET
ELEMENTS. THE SET ELEMENTS ARE ARRANGED IN THE ORDER OF THEIR
UNIVERSAL SEQUENCE NUMBER. THE ELEMENTS WITH SMALLER UNIVERSAL
SEQUENCE NUMBERS ARE STORED FIRST. THE SEQUENCE IN WHICH THE
ELEMENT FIELDS ARE UTILIZED IS: FIRST, THIRD, FOURTH (OR EQUIVA-
LENTLY, LEFT HALF OF THE FIRST WORD, LEFT HALF OF THE SECOND WORD,
AND RIGHT HALF OF THE SECOND WORD). WHEN THE NUMBER OF ELEMENTS
IN A SET IS NOT DIVISIBLE BY THREE, THE LAST BLOCK WILL CONTAIN
UNUSED ELEMENT FIELD(S). THE UNUSED ELEMENT FIELDS ARE ZERO TO
DENOTE THEIR EMPTINESS. AN EXAMPLE OF A BLOCK SET IS SHOWN IN
FIGURE 5.

FIGURE 5. SET IN BLOCK FORM CONTAINING ELEMENTS
I1 < I2 < ... < IN
FOR EACH COLUMN K (0 ≤ k < 16) THERE ARE TWO INFORMATION WORDS, C1(K) AND C2(K), ASSOCIATED WITH IT. THE ARRAYS C1 AND C2 CONTAIN INFORMATION FOR MANAGING THESE 15 COLUMNS. IF COLUMN K IS NOT ASSIGNED TO ANY SET, THE RIGHT HALF OF C1(K) CONTAINS A COLUMN NUMBER WHICH IS THE NEXT FREE (UNASSIGNED) COLUMN, OR IT CONTAINS 0'S TO INDICATE THAT THIS IS THE LAST FREE COLUMN. THIS IS, IN FACT, A LINKED LIST OF AVAILABLE COLUMN SPACES. HOWEVER, IF COLUMN K IS CURRENTLY ASSIGNED TO A SET, THEN THESE TWO WORD CONTAIN INFORMATION AS FOLLOWS: (1) LEFT HALF OF C1(K) CONTAINS AN ADDRESS POINTER TO THE 'INFO' BLOCK, (2) RIGHT HALF OF C1(K), CALLED TIME FIELD, CONTAINS A GLOBAL REFERENCE COUNT VALUE WHICH RANKS SETS BY THEIR LATEST USE, AND (3) C2(K) IS SUBDIVIDED INTO SIX SIXTHWORD FIELDS. THE FIRST SIXTHWORD OF C2(K) IS THE USE-TAG FIELD WHICH IS SET TO ONE WHenever THE SET IS USED IN THE CURRENT OPERATION, THE SECOND SIXTHWORD FIELD IS THE PROPERTY-TAG FIELD WHICH IS SET TO ONE IF THE SET IS REPRESENTING NODES(G), ARCS(G), OR DOMAIN(P). THE REMAINING 4 FIELDS OF C2(K) ARE NOT USED AND ARE AVAILABLE FOR FUTURE EXPANSION. AN EXAMPLE OF A COLUMN SET IS SHOWN IN FIGURE 6.
THE RESULTS OF SET OPERATIONS MUST BE STORED IN COLUMN FORM (IN ORDER TO SPEED UP THE SET OPERATION). SINCE A TOTAL OF 15 COLUMN SETS MAY EXIST IN THE SYSTEM AT ANY ONE GIVEN TIME, A MANAGEMENT SCHEME IS REQUIRED TO TRANSFER SETS FROM COLUMN FORM TO BLOCK FORM IN ORDER TO MAKE COLUMN SPACE AVAILABLE. TO PREVENT THE TRANSFER OF A COLUMN SET WHICH IS CURRENTLY BEING USED THE USE-TAG FILED ASSOCIATED WITH THIS COLUMN SET IS CHECKED. TO PREVENT THE TRANSFER OF THE SET OF NODES(G) OR ARCS(G) TO A BLOCK FORM (WHICH WOULD PRODUCE A SECOND LINKED LIST COPY OF THIS SET WHICH ALREADY EXISTS IN PROPERTY BLOCKS) THE PROPERTY-TAG FIELD ASSOCIATED WITH THIS COLUMN SET IS CHECKED.

DURING THE PROGRAM EXECUTION, A GLOBAL COUNTER IS MAINTAINED (FOR COLUMN SETS REFERENCES). WHENEVER ANY COLUMN SET (SAY WITH COLUMN INDEX K) IS REFERENCED (E.G. DURING SET OPERATIONS) THE GLOBAL COUNTER IS INCREMENTED BY ONE AND ITS UPDATED VALUE IS STORED INTO THE RIGHT HALF OF C1(K) (THE TIME FIELD). IF A RE-
3.2.3. STAQUES (LISTS)

The staque (or list) is one of the new data structures introduced in the FGHAA. From the storage structure point of view, there are three different categories of staques: (1) single word—logical, integer and real types of staque; (2) double word—double precision and complex types of staque; and (3) variable size—set type of staque. Staques, regardless of their data types, are always represented by a staque header and a circular doubly-linked list of its staque elements.

The staque header of a declared staque is a memory word assigned by the compiler, which is local to the subprogram where it had been declared. During the program execution, the address of this memory word becomes the unique identifier of the associated staque. The staque header contains two halfword fields. The left half of thestaque header contains the address pointer to the linked list of its staque elements. The right half of the header is the type field which contains the data type identifier of its associated staque.

The staque elements are arranged in the form of a circular doubly-linked list consisting of 2-word blocks. The first word of the 2-word block is divided into two halfword link fields. The left half is the backward link field pointing to the previous block and the right half is the forward link field pointing to the next block. The first and last blocks of the staque element list are linked together circularly (i.e., the backward pointer of the first points to the last and the forward pointer of the last points to the first). The second word of the 2-word block contains different information depending on the data type of its staque. For the data types in the single word category, it contains the data value of the element, for the set data type, it contains the set header, for the data types in the double word category, it contains the address pointer to another 2-word block which, in turn, contains the data values of the element. The representation of different categories of the staques are given in Figure 7.
**LOGICAL, INTEGER, REAL TYPES OF LISTS:**

<table>
<thead>
<tr>
<th>LIST HEADER</th>
<th>K1</th>
<th>K2</th>
<th>KN</th>
</tr>
</thead>
<tbody>
<tr>
<td>* K1 * TYPE*</td>
<td>* KN * K2 *</td>
<td>* K1 * K3 * ...</td>
<td>* KN=1* K1 *</td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td>*************</td>
</tr>
<tr>
<td>* VALUE *</td>
<td>* VALUE *</td>
<td>* VALUE *</td>
<td></td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td></td>
</tr>
</tbody>
</table>

**SET TYPE OF LISTS:**

<table>
<thead>
<tr>
<th>LIST HEADER</th>
<th>K1</th>
<th>K2</th>
<th>KN</th>
</tr>
</thead>
<tbody>
<tr>
<td>* K1 * TYPE*</td>
<td>* KN * K2 *</td>
<td>* K1 * K3 * ...</td>
<td>* KN=1* K1 *</td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td>*************</td>
</tr>
<tr>
<td>* SET HEADER *</td>
<td>* SET HEADER *</td>
<td>* SET HEADER *</td>
<td></td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td></td>
</tr>
</tbody>
</table>

The set headers are similar to the set headers of the normal sets except no column form is used.

**DOUBLE PRECISION AND COMPLEX TYPES OF LISTS:**

<table>
<thead>
<tr>
<th>LIST HEADER</th>
<th>K1</th>
<th>K2</th>
<th>KN</th>
</tr>
</thead>
<tbody>
<tr>
<td>* K1 * TYPE*</td>
<td>* KN * K2 *</td>
<td>* K1 * K3 * ...</td>
<td>* KN=1* K1 *</td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td>*************</td>
</tr>
<tr>
<td>* 0 * I1 *</td>
<td>* 0 * I2 *</td>
<td>* 0 * IN *</td>
<td></td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td>I2</td>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td></td>
</tr>
<tr>
<td>* VALUE 1 *</td>
<td>* VALUE 1 *</td>
<td>* VALUE 1 *</td>
<td></td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td></td>
</tr>
<tr>
<td>* VALUE 2 *</td>
<td>* VALUE 2 *</td>
<td>* VALUE 2 *</td>
<td></td>
</tr>
<tr>
<td>*************</td>
<td>*************</td>
<td>*************</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 7. REPRESENTATIONS OF STAQUES**
3.2.4. Properties

The property is another new data structure introduced in the FGRAAL system. From the storage structure point of view there are three different categories of properties (as it is in the STAQUE): (1) Single word — logical, integer and real types of property, (2) double word — double precision and complex types of property, and (3) variable size — set type of property. Properties, regardless of their data types, are always represented by a property header, property info block and property value block(s).

The property header of a declared property is a memory word assigned by the compiler (as it is with the set header). During the program execution, the absolute address of this assigned memory word becomes the unique identifier of the associated property. The property header contains two halfword fields. The right half of the property header is the type field which contains the data type identifier of its associated property. The left half of the property header is the pointer field which contains the complement of the absolute address of a 2-word block called property info block. The complement of the address is used in order to distinguish a property header from a set header. The property info block pointed to by the property header has the same format as that of a column set info block (see sets), except that the column index field is zero unless the domain set is represented by a column set.

The assignment of properties to an atomic set can be realized through the execution of property assignment or conditional 'create' statements. The property value(s) thus assigned to an (universal) element is stored in a property block which is linked into the property-ring associated with this element (see universal sequence and property blocks). The property block consists of two contiguous memory words. The first memory word is subdivided into two halfword fields. The left half is the property identification field which contains the unique identifier, the address of the property header, of the property with which this block is associated. The right half of the first word is the link field which is used in the linkage of the property-ring (see universal sequence and property block). The second word of the property block contains different information depending on the data type of the property (this is in the same format as that of the STAQUE). For the data types in the single word category, it contains the data value of the property. For the set data type, it contains the set header. For the data types in the double word category, it contains the address pointer to another 2-word block which, in turn, contains the data values of the property. The representations of properties are given in Figure 8.
FIGURE 8. REPRESENTATION OF PROPERTY BLOCKS.
3.2.5. GRAPH STRUCTURES


\[
\begin{align*}
L(G) & : 0 \quad T1 \quad T2 \quad 3 \\
L(G)+1 & : -NA \quad 1 \\
L(G)+2 & : -AA \quad 2 \\
\end{align*}
\]

\[
\begin{align*}
NA & : 0 \quad N \quad 0 \\
AA & : -M \quad 0 \\
\end{align*}
\]

\[
\begin{align*}
T1 & = 0 \quad \text{FOR NODE GRAPH} \\
& = 1 \quad \text{FOR NODE/ARC GRAPH} \\
N & = \text{NUMBER OF NODES} \\
M & = \text{NUMBER OF ARCS} \\
T2 & = 0 \quad \text{FOR UNDIRECTED GRAPH} \\
& = 1 \quad \text{FOR DIRECTED GRAPH} \\
\end{align*}
\]

THE INTEGERS 3, 1 AND 2 IN L(G), L(G)+1 AND L(G)+2, RESPECTIVELY, ARE USED AS IDENTIFIERS.

FIGURE 9. GRAPH HEADER WITH NODE AND ARC INFORMATION BLOCKS

FIGURE 10. GRAPH STRUCTURE LINKAGE.
THE NEGATIVE COBOUNDARY OF NODE K,
L(G) = LOCATION OF THE GRAPH HEADER

PROP. BLOCKS OF ELEMENTS
K, J1, ..., JK
*******************************
NODE K
B * L(G)+1 * LINK *
*******************************

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

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

ARC J1
AN1 * L(G)+2 * LINK *
***********************

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

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

ARC JK
ANK * L(G)+2 * LINK *
***********************

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

***************
3.2.6. TYPE ASSIGNMENTS.

THE HEADERS OF LISTS, PROPERTIES AND GRAPHS CONTAIN A FIELD CONSISTING OF A TYPE CODE. THIS TYPE CODE IS AN INTEGER AS FOLLOWS:

0  SET
1  NODES
2  ARCS
3  GRAPH
4  LOGICAL
5  INTEGER
6  REAL
7  DOUBLE PRECISION
8  COMPLEX
3.2.7. STRUCTURE OF HEADERS

THE FOLLOWING TABLE SUMMARIZES THE CONTENTS OF THE HEADERS:

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>VALID</th>
<th>REMOVED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>10000000000000000</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*VALID</td>
<td>*REMOVED</td>
</tr>
<tr>
<td></td>
<td>*COLUMN SET BITS</td>
<td><em>ADDRESS OF PROP.</em></td>
</tr>
<tr>
<td></td>
<td><em>BLOCK OR COMPL.</em></td>
<td><em>OF THE ELEMENT</em></td>
</tr>
<tr>
<td>EMPTY</td>
<td><em>ZERO</em></td>
<td><em>ZERO</em></td>
</tr>
<tr>
<td>ATOMIC</td>
<td><em>ZERO</em></td>
<td><em>ELEMENT</em></td>
</tr>
<tr>
<td>BLOCK</td>
<td><em>ADDRESS OF THE</em></td>
<td><em>ZERO</em></td>
</tr>
<tr>
<td>COLUMN</td>
<td><em>FIRST BLOCK</em></td>
<td><em>ZERO</em></td>
</tr>
<tr>
<td></td>
<td>INFO, BLOCK</td>
<td></td>
</tr>
<tr>
<td>LIST</td>
<td>ADDRESS</td>
<td>0</td>
</tr>
<tr>
<td>SET</td>
<td>OF</td>
<td>4</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>THE</td>
<td>5</td>
</tr>
<tr>
<td>INTEGER</td>
<td>FIRST</td>
<td>6</td>
</tr>
<tr>
<td>REAL</td>
<td>ELEMENT</td>
<td>7</td>
</tr>
<tr>
<td>DBL,PR.</td>
<td>BLOCK</td>
<td>8</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>*PROPERTY</td>
<td><em>INFORMATION</em></td>
</tr>
<tr>
<td>PROPERTY</td>
<td>ADDRESS</td>
<td>0</td>
</tr>
<tr>
<td>SET</td>
<td>OF THE</td>
<td>4</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>PROPERTY</td>
<td>5</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INFORMATION</td>
<td>6</td>
</tr>
<tr>
<td>REAL</td>
<td>BLOCK</td>
<td>7</td>
</tr>
<tr>
<td>DBL,PR.</td>
<td>*</td>
<td>8</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>*</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH</td>
</tr>
<tr>
<td>ADDRESS</td>
</tr>
<tr>
<td>NODES</td>
</tr>
<tr>
<td>ADDRESS OF INFO.</td>
</tr>
<tr>
<td>ARCS</td>
</tr>
<tr>
<td>BLOCK</td>
</tr>
<tr>
<td>INFO,BLOCK</td>
</tr>
<tr>
<td>ADDRESS OF ELEMENTS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FIRST W.</td>
</tr>
<tr>
<td>FIRST ELEMENT</td>
</tr>
</tbody>
</table>

NOTES:

X = ZERO OR ONE FOR NOUE OR NODE/ARC GRAPH, RESPECTIVELY.
Y = ZERO OR ONE FOR UNDIRECTED OR DIRECTED GRAPH, RESP.
- (MINUS) INDICATES COMPLEMENT IN THE PROPER HALF WORD,
CONTRACTION OF A SUBGRAPH CORRESPONDING TO THE GIVEN SET OF NODES IS ACCOMPLISHED IN TWO STEPS:

STEP 1.
A NEW NODE, \( N \), IS CREATED WHICH WILL CORRESPOND TO THE CONTRACTED SUBGRAPH. ALL NODES INVOLVED IN THE CONTRACTION, INCLUDING THOSE NODES WHICH WERE PART OF PREVIOUS CONTRACTION, ARE ASSIGNED NEW PROPERTY BLOCKS. CALL THESE NODES \( N_1, N_2, \ldots, N_M \). THE OLD PROPERTY BLOCKS ARE LINKED TO THE NEW BLOCKS BY STORING THEIR NEGATIVE ADDRESSES IN THE LEFT HALVES OF THE FIRST WORD OF THE NEW PROPERTY BLOCKS:

\[
\begin{array}{c}
\text{NEW PROPERTY BLOCK OF } N_i \\
\text{********} & \text{********} & \text{*******} \\
* \leftrightarrow & * -A* & * \\
\text{********} & \text{********} & \text{*******} \\
\end{array}
\]

A \text{ **********} OLD PROPERTY BLOCK OF \( N_i \) 
\text{********} (CONTENTS UNCHANGED)

\[
\begin{array}{c}
* * * \\
\text{********} \\
\end{array}
\]

STEP 2.
THE ARCS IN EACH ORIGINAL POSITIVE AND NEGATIVE COBOUNDARY CYCLE ARE REARRANGED SO THAT THE ARCS INTERNAL TO THE CONTRACTED SUBGRAPH PRECEDE THE ARCS CONNECTING THE SUBGRAPH WITH THE REST OF THE GRAPH. THE LINKAGES IN THE SECOND WORD OF THE NEWLY CREATED PROPERTY BLOCKS, ASSOCIATED WITH THE NODES \( N, N_1, \ldots, N_M \) ARE DEFINED AS FOLLOWS:

1. THE LEFT FIELD OF THE SECOND WORD (NCOB POINTER) OF \( N \) POINTS TO THE FIRST NON-INTERNAL ARC OF THE NCOB OF \( N_1 \). IF NO NON-INTERNAL ARCS EXIST, THEN IT POINTS TO \( N_1 \).

2. FOR \( N_i (1 \leq i \leq M-1) \), THE LEFT FIELD OF THE SECOND WORD OF \( N_i \) POINTS TO THE FIRST NON-INTERNAL ARC OF THE NCOB OF \( N_{i+1} \). IF NO NON-INTERNAL ARCS EXIST, THEN IT POINTS TO \( N_{i+1} \).

3. THE LEFT HALF OF THE SECOND WORD OF \( N_M \) POINTS TO \( N \).

A CORRESPONDING LINKAGE STRUCTURE IS DEFINED FOR PCOB OF \( N \) USING THE RIGHT FIELDS OF THE SECOND WORD OF THE NODES.
THE LINKAGE STRUCTURE IS ILLUSTRATED BELOW FOR A 2-NODE CONTRACTION:

GIVES THE FOLLOWING LINKAGE (ONLY THE SECOND WORDS OF THE BLOCKS ARE SHOWN):

CREATED NEW NODE

NEW PROP. BLOCK

OLD PROP. BLOCK

A1

A2

A3

A4

A5

A6

A7

A8

A9

A10

A1

A2

A3

A4

A5

A6

A7

A8

A9

A10
4. EXAMPLE OF A GRAPH STRUCTURE

IN THIS EXAMPLE, IT IS ASSUMED THAT 10 ELEMENTS WERE CREATED. THE ELEMENTS HAVE AN ALPHANUMERIC PROPERTY WITH '3Q0' AS THE PROPERTY IDENTIFIER (ADDRESS OF THE PROPERTY HEADER). FURTHERMORE THE ELEMENTS WERE ASSIGNED TO A GRAPH AS NODES AND ARCS SUCH THAT THE GRAPH IDENTIFIER IS '500', THUS '501' AND '502' CORRESPONDS TO THE NODE AND ARC IDENTIFIER, RESPECTIVELY. THE GRAPH HAS THE FOLLOWING STRUCTURE:

THE FOLLOWING FIGURE GIVES THE STORAGE STRUCTURE OF THE ABOVE EXAMPLE:

<table>
<thead>
<tr>
<th>D(1)</th>
<th>7998</th>
<th>5998</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 7998 *</td>
<td>300 * 5998 *</td>
<td>501 * -1 *</td>
</tr>
<tr>
<td>* 'A' *</td>
<td>*</td>
<td>* 5988 * 5998 *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(2)</th>
<th>7996</th>
<th>5996</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 7996 *</td>
<td>300 * 5996 *</td>
<td>501 * -2 *</td>
</tr>
<tr>
<td>* 'B' *</td>
<td>*</td>
<td>* 5986 * 5988 *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(3)</th>
<th>7994</th>
<th>5994</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 7994 *</td>
<td>300 * 5994 *</td>
<td>501 * -3 *</td>
</tr>
<tr>
<td>* 'C' *</td>
<td>*</td>
<td>* 5982 * 5984 *</td>
</tr>
</tbody>
</table>
ASSUMING THE GRAPH IS CONTRACTED, SUCH THAT THE NODES B, C AND D WERE CONTRACTED INTO A NEW NODE WITH ALPHANUMERIC PROPERTY 'BCD':

THE CHANGED STORAGE STRUCTURE IS AS FOLLOWS:

D(1)
***************
* * 7998 *
***************

D(2)
***************
* * 7996 *
***************

D(3)
***************
* * 7994 *
***************

A  V  C/D  W  E

0(1) 7998 .
***************
* * 7998 *
***************

5998
***************
* 501 * -1 *
***************

0(2) 7996 *
***************
* * 7996 *
***************

5996
***************
* 501 * -2 *
***************

0(3) 7994 *
***************
* * 7994 *
***************

5994
***************
* 501 * -3 *
***************
<table>
<thead>
<tr>
<th>D(4)</th>
<th>7992</th>
<th>5992</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>7992</td>
<td>*</td>
</tr>
<tr>
<td>D'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(5)</th>
<th>7990</th>
<th>5990</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>7990</td>
<td>*</td>
</tr>
<tr>
<td>E'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(6)</th>
<th>7988</th>
<th>5988</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>7988</td>
<td>*</td>
</tr>
<tr>
<td>V'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(7)</th>
<th>7986</th>
<th>5986</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>7986</td>
<td>*</td>
</tr>
<tr>
<td>X'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(8)</th>
<th>7984</th>
<th>5984</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>7984</td>
<td>*</td>
</tr>
<tr>
<td>Y'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D(9)</th>
<th>7982</th>
<th>5982</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>7982</td>
<td>*</td>
</tr>
<tr>
<td>Z'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>D(10)</td>
<td>7980</td>
<td>5980</td>
</tr>
<tr>
<td></td>
<td>***************</td>
<td>***************</td>
</tr>
<tr>
<td></td>
<td>* 7980 *</td>
<td>* 300 * 5980 *</td>
</tr>
<tr>
<td></td>
<td>* -10 *</td>
<td>* 502 *</td>
</tr>
<tr>
<td></td>
<td>***************</td>
<td>***************</td>
</tr>
<tr>
<td>D(11)</td>
<td>7978</td>
<td>5978</td>
</tr>
<tr>
<td></td>
<td>***************</td>
<td>***************</td>
</tr>
<tr>
<td></td>
<td>* 7978 *</td>
<td>* 300 * 5978 *</td>
</tr>
<tr>
<td></td>
<td>* -11 *</td>
<td>* 501 *</td>
</tr>
<tr>
<td></td>
<td>***************</td>
<td>***************</td>
</tr>
<tr>
<td></td>
<td>'BCD'</td>
<td>5978</td>
</tr>
<tr>
<td></td>
<td>***************</td>
<td>***************</td>
</tr>
<tr>
<td></td>
<td>* 5996 * 5988 *</td>
<td>* 5992 * 5990 *</td>
</tr>
<tr>
<td></td>
<td>***************</td>
<td>***************</td>
</tr>
</tbody>
</table>
5. THE FGRAAL COMPILER

THE FGRAAL COMPILER TRANSLATES FGRAAL SOURCE PROGRAMS INTO OBJECT CODE. TO SIMPLIFY ITS USE, IT ACCEPTS THE SAME INPUT (SYMBOLIC ELEMENTS OR CARUS), PRODUCES THE SAME OUTPUT (RELOCATABLE BINARY ELEMENTS) AND IS CALLED IN A SIMILAR WAY AS ALL LANGUAGE PROCESSORS ON THE UNIVAC 1108 UNDER EXEC 8. THE FOLLOWING SECTIONS DESCRIBE THE UNIVERSITY OF MARYLAND RALPH COMPILER, AND THE MODIFICATIONS NECESSARY TO EXTEND IT TO FGRAAL.

FGRAAL IS IMBEDDED INTO FORTRAN, AND MOST OF THE SYNTACTIC RULES OF FORTRAN APPLY ALSO TO THE EXTENSION. THE STATEMENTS MUST BE ANALYZED FAIRLY THOROUGHLY TO DETERMINE THEIR NATURE. FOR INSTANCE, THE EXPRESSION 'A(I)' CAN BE A FORTRAN FUNCTION CALL, A REFERENCE TO A SUBSCRIPTED FORTRAN VARIABLE, OR A REFERENCE TO A PROPERTY IN FGRAAL, DEPENDING ONLY ON THE WAY IN WHICH 'A' AND 'I' ARE DECLARED. ANY OF THESE CAN OCCUR IN A FORTRAN EXPRESSION. SIMILARLY, THE STATEMENT 'S=T' MUST BE TRANSLATED DIFFERENTLY WHEN 'S' AND 'T' ARE SETS OR LISTS THAN WHEN THEY ARE NORMAL FORTRAN VARIABLES.

FOR THIS REASON, AND BECAUSE THE UNIVERSITY OF MARYLAND RALPH COMPILER WAS READILY AVAILABLE AND COULD BE MODIFIED EASILY TO ACCEPT THE NEW FEATURES, WE CHOSE TO MODIFY IT FOR OUR PURPOSE. WRITING A PREPROCESSOR INSTEAD, WHICH WOULD TRANSLATE FGRAAL INTO FORTRAN (WITH APPROPRIATE SUBROUTINE CALLS SUBSTITUTED FOR FGRAAL CONSTRUCTS) WOULD HAVE REQUIRED A VERY THOROUGH ANALYSIS OF THE PROGRAM, DUPLICATING MUCH OF THE WORK OF A COMPILER.

THE OTHER EXTREME, WRITING A COMPILER FROM SCRATCH, WOULD HAVE HAD THE ADVANTAGE OF BEING ABLE TO CONTROL THE METHODS OF COMPIATION MORE COMPLETELY, BUT THE AMOUNT OF WORK INVOLVED WOULD HAVE BEEN GREAT, AND MUCH OF IT WOULD HAVE BEEN A DUPLICATION OF FUNCTIONS OF ALREADY EXISTING COMPILERS.

5.1. THE RALPH COMPILER

THE FOLLOWING IS A BRIEF DESCRIPTION OF RALPH, WHICH CANNOT DO FULL JUSTICE TO ALL ITS FEATURES.

RALPH (REENTRANT ALGORITHMIC LANGUAGE PROCESSOR) IS A FOUR PASS COMPILER, DESIGNED TO COMPILE PROGRAMS QUICKLY, WITH LESS EXTENSIVE OPTIMIZATION THAN THE UNIVAC FORTRAN V COMPILER.

PASS ONE READS THE SOURCE PROGRAM AND TRANSLATES IT TO INTERMEDIATE OBJECT CODE (TRIPLES). AT THE SAME TIME IT BUILDS THE SYMBOL TABLES. THE STATEMENT RECOGNIZER IS TABLE DRIVEN. THE VARIOUS STATEMENTS ARE PROCESSED IN INDIVIDUAL PARTS OF THE COMPILER, WHICH USE A COMMON POOL OF SUBROUTINES. THESE INCLUDE A SYNTAX SCANNER, AN EXPRESSION COMPILER, THE SYMBOL TABLE ROUTINES, AND A ROUTINE FOR OUTPUTTING TRIPLES TO A SCRATCH FILE, WHICH DOES
ALL OPTIMIZATION.
PASS TWO CLEANS UP THE SYMBOL TABLE, PROCESSES EQUIVALENCE
STATEMENTS, AND ALLOCATES DATA STORAGE.
PASS THREE CONVERTS THE INTERMEDIATE OBJECT CODE TO MACHINE
INSTRUCTIONS, SOME OF WHICH MAY NOT YET BE COMPLETE. IT READS THE
TRIPLES, DETERMINES THE MODES OF THE OPERANDS, AND FINDS A TABLE
ENTRY FOR THAT PARTICULAR COMBINATION, FROM THAT IT DETERMINES
THE MODE OF THE RESULT, IF ANY, AND FINDS THE DEFINITION SEQUENCE
FOR THE TRIPLE. THE DEFINITION SEQUENCES CONTAIN PSEUDO INSTRUCTIONS (FOR REGISTER ALLOCATION, TRANSFER OF CONTROL, DEFINITION OF
REGISTER CONTENTS, ETC.) AS WELL AS MACHINE INSTRUCTIONS, WHOSE
REGISTER AND OPERAND FIELDS ARE FILLED WITH SPECIAL CODES, AN
INTERPRETER ANALYZES THESE, REPLACES THE FIELDS BY REAL ADDRESSES,
AND WRITES THESE, WITH ANY RELOCATION INFORMATION, TO A SCRATCH
FILE. INCOMPLETE ADDRESSES (FORWARD REFERENCES TO LABELS, ETC.)
ARE FLAGGED.
PASS FOUR READS THE OUTPUT FROM PASS THREE, MODIFIES THE IN-
COMPLETE INSTRUCTIONS, AND WRITES THE FINISHED RELOCATABLE ELE-
MENT. IT ALSO PRINTS AN OBJECT CODE LISTING WHEN REQUESTED.

5.2. MODIFICATIONS TO RALPH

THESE FALL INTO TWO GROUPS: SOME ARE LANGUAGE RELATED, AND
WOULD HAVE TO BE TREATED SIMILARLY IN MOST IMPLEMENTATIONS, OTHERS
ARE DIRECTLY RELATED TO THE COMPILER WHICH IS BEING MODIFIED.

5.2.1. DATA TYPES

THERE ARE SEVERAL NEW VARIABLE TYPES: SETS, STAQUES, PROPER-
TIES, AND GRAPHS. SETS, STAQUES, AND PROPERTIES OCCUPY SINGLE
WORDS IN THE PROGRAMS, WHICH MAY POINT INTO DYNAMIC STORAGE AT
EXECUTION TIME. GRAPHS CONSIST OF THREE WORDS, ONE HAS INFOR-
MATION ABOUT THE GRAPH, THE OTHER TWO ABOUT THE ARCS AND NODES,
RESPECTIVELY. SET IS A VARIABLE TYPE, LIKE INTEGER OR REAL.
STAQUES AND PROPERTIES HAVE ONE OF THE TYPES INTEGER, REAL, DOUBLE
PRECISION, COMPLEX, LOGICAL, OR SET ASSOCIATED WITH THEM.
MOST OF THESE NEW VARIABLES REQUIRE THAT AN INITIAL VALUE IS
COMPILED FOR THEM. THIS IS DONE AT THE END OF PASS THREE, USING
THE INFORMATION IN THE SYMBOL TABLE.

5.2.2. IMPLEMENTATION OF NEW STATEMENTS

DECLARATION STATEMENTS ARE HANDLED IN THE SAME WAY AS FORTRAN
DECLARATIONS. THE SYMBOL IS ENTERED INTO THE MODE TABLE, AND, IF
NECESSARY, INTO THE DIMENSION TABLE.
ASSIGN, DETACH, AND REMOVE STATEMENTS ARE TRANSLATED INTO
CALLS TO APPROPRIATE SUBROUTINES.
The SAVE AND RESET STATEMENTS ARE NOT YET IMPLEMENTED.
5.2.3. IMPLEMENTATION OF NEW SYNTACTIC FEATURES

THE SET OPERATIONS ARE SIMPLY ENTERED INTO A TABLE OF LEGAL OPERATORS. THEY ALL PRODUCE SUBROUTINE CALLS.

THE GRAPH OPERATIONS (BOUNDARY OPERATIONS) ARE HANDLED SIMILARLY, EXCEPT THAT THEY ALL CALL THE SAME SUBROUTINE, AND THE SPECIFIC OPERATION IS TRANSMITTED THROUGH A FLAG WORD.

THE FUNCTIONS 'NODES', 'ARCS', AND 'DOMAIN' ARE TRANSLATED INTO CONSTANT OFFSETS FROM THE GRAPH OR PROPERTY, AND THEIR RESULTS ARE SETS. THIS CAUSED SOME DIFFICULTY BECAUSE OF THE WAY IN WHICH RALPH PASSES CONSTANT OFFSETS FROM PASS ONE TO PASS THREE.

PROPERTIES CAN APPEAR ON EITHER SIDE OF ASSIGNMENT STATEMENTS. THE TWO CASES HAVE A DIFFERENT MEANING, AND EACH MUST BE TRANSLATED DIFFERENTLY.

THE SUBSET FUNCTION PRESENTED A PARTICULARLY DIFFICULT PROBLEM. IF IT IS NOT TO BE TRANSLATED INLINE, WHICH WOULD GENERATE TOO MUCH CODE, IT REQUIRES A CALL BY NAME TO EVALUATE THE LOGICAL EXPRESSION. THIS IS ACHIEVED BY TRANSLATING THE LOGICAL EXPRESSION INTO AN INTERNAL FUNCTION, AND PASSING THE ADDRESS OF THAT FUNCTION TO THE SUBSET ROUTINE.

THE ARGUMENTS TO LIST ASSIGNMENT STATEMENTS ARE PASSED IN ONE ARRAY, CONTAINING THE ADDRESS OF EACH ITEM, AND A FLAG INDICATING WHETHER OR NOT THE VARIABLE IS ANOTHER LIST, AND WHETHER OR NOT IT NEEDS TO BE COPIED (FOR LISTS AND SETS).

5.2.4. HANDLING OF TEMPORARIES

TEMPORARY RESULTS FROM SET OPERATIONS PRESENT A NEW PROBLEM: THEY DO NOT CONTAIN A VALUE WHICH CAN BE SIMPLY FORGOTTEN, BUT THEY POINT TO AN ALLOCATED AREA IN DYNAMIC STORAGE WHICH HAS TO BE FREEED. THIS OCCURS IN THE EVALUATION OF EXPRESSIONS, WHEN PASSING SET EXPRESSIONS TO A SUBROUTINE AS AN ARGUMENT, AND DURING THE EXECUTION OF 'FOR ALL' LOOPS, FOR WHICH A SET HAS TO BE KEPT THROUGHOUT THE EXECUTION OF THE LOOP.

FREEING EACH TEMPORARY EXPLICITLY WOULD CREATE MUCH EXTRA CODE, SO THAT A BETTER METHOD HAD TO BE FOUND. SINCE ALL SET OPERATIONS ARE EXECUTED IN SUBROUTINES, A FLAG IS PASSED IN THE CALLING SEQUENCE, INDICATING WHEN A TEMPORARY SET RESULT IS NO LONGER NEEDED. THE SUBROUTINE CAN THEN USE THE SET WITHOUT COPYING IT, IF IT WISHES TO DO SO. OTHERWISE IT HAS TO FREE THE SET BEFORE IT EXITS. THIS IS MADE THE RESPONSIBILITY OF THE SUBROUTINE, SET ARGUMENTS TO FORTRAN SUBROUTINES CAN BE HANDLED IN THE SAME WAY, PROVIDED THAT THE SUBROUTINE EXIT SEQUENCE TAKES CARE OF
FREEING ALL THE SETS.

TRANSFERS OUT OF 'FOR ALL' LOOPS ARE MORE DIFFICULT, SINCE THE TRANSFER MAY BE IMBEDDED INTO A CALLING SEQUENCE, IT IS NOT POSSIBLE TO SIMPLY REDEFINE THE JUMP INSTRUCTION. INSTEAD, FOR EACH TRANSFER OUT OF A 'FOR ALL' LOOP, A DUMMY LABEL IS CREATED AT THE END OF THAT LOOP, FOLLOWED BY A RELEASE OF THE SET, AND A JUMP TO THE REAL LABEL, OR A JUMP TO A DUMMY LABEL AT THE END OF THE NEXT 'FOR ALL' LOOP.

5.3. OPTIMIZATION

REPEATED SET OPERATIONS ARE ELIMINATED IN THE SAME WAY AS ALL REPEATED ARITHMETIC OPERATIONS, AS LONG AS NO LABELS INTERVENE. THE SAME HOLDS FOR PROPERTY RETRIEVALS AND THE NONDESTRUCTIVE LIST RETRIEVALS. THIS FEATURE CAN BE TURNED OFF BY USING THE 'O' OPTION ON THE COMPILER CALL CARD.

OPERATIONS OF THE FORM

\[
\text{SET} = \text{SET .UN. } x \quad \text{OR} \quad \text{SET} = \text{SET .DF. } x
\]

ARE FREQUENT ENOUGH THAT IT SEEMED WORTHWHILE TO HAVE SPECIAL ENTRIES FOR THEM. THIS REDUCES THE LENGTH OF THE CALLING SEQUENCE BY ONE HALF, AND MAY ALSO BE USED BY THE SUBROUTINE TO AVOID UNNECESSARY COPYING OF THE SET.
6. COMPILER GENERATED INSTRUCTION SEQUENCES

THIS CHAPTER CONTAINS THE DESCRIPTION OF THE INSTRUCTION SEQUENCES GENERATED BY THE FGRAAL COMPILER FOR THE SPECIAL FEATURES OF FGRAAL. THE FGRAAL COMPILER IS AN EXTENSION OF THE RALPH COMPILER, THUS IT GENERATES INSTRUCTION FOR THE FORTRAN STATEMENTS. ALSO, THERE IS NO ATTEMPT MADE TO DESCRIBE THESE INSTRUCTION SEQUENCES. ONLY THE SEQUENCES RELEVANT TO FGRAAL FEATURES ARE DESCRIBED HERE.

6.1. DECLARATION STATEMENTS

DECLARED SETS, PROPERTIES, LISTS AND GRAPHS RECEIVE INITIAL DATA ASSIGNMENTS BY THE COMPILER. THESE DATA ARE INITIALIZED AS FOLLOWS:

- **SET**
  
<table>
<thead>
<tr>
<th>*</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **PROPERTY**
  
<table>
<thead>
<tr>
<th>*</th>
<th>-0</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **LIST**
  
<table>
<thead>
<tr>
<th>*</th>
<th>-0</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **GRAPH**
  
<table>
<thead>
<tr>
<th>G</th>
<th>-0</th>
<th>XXXX</th>
<th>YYYY</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G+1</th>
<th>-0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G+2</th>
<th>-0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WHERE

- XXXX = 0 FOR NODE GRAPH
  = 1 FOR NODE/ARC GRAPH
- YYYY = 0 FOR UNDIRECTED GRAPH
  = 1 FOR DIRECTED GRAPH
- TYPE = 0 FOR SET TYPE
  = 4 FOR LOGICAL TYPE
  = 5 FOR INTEGER TYPE
  = 6 FOR REAL TYPE
  = 7 FOR DOUBLE PRECISION TYPE
  = 8 FOR COMPLEX TYPE
6.2. CALLING SEQUENCES

THE COMPILER GENERATES CALLING SEQUENCES FOR THE FORRAAL LIBRARY ROUTINES. THESE CALLING SEQUENCES ARE IN THE SAME FORM AS THE FORTRAN CALLING SEQUENCES, I.E.,

\[
\begin{align*}
LMJ & \quad X11,\text{LIBR} \\
+ & \quad \text{ARGl} \\
\ldots & \quad \text{ARN} \\
+ & \quad \text{W.B.}
\end{align*}
\]

WHERE 'LIBR' IS THE NAME OF A LIBRARY PROGRAM, ARGl, ..., ARGN ARE THE ADDRESSES OF THE ARGUMENTS, W.B. IS THE WALK BACK WORD, REGISTERS A0-A5 ARE ASSUMED TO BE FREE TO USE BY THE LIBRARY PROGRAMS WITHOUT SAVING THEM. RESULTS ARE RETURNED IN A0 OR IN A0-A1.

SETS IN THE CALLING SEQUENCES HAVE AN ADDITIONAL FEATURE. BESIDES THE ADDRESS OF THEIR SET HEADER, THERE IS A FLAG IN BIT POSITION 12:

\[
\begin{align*}
\text{******} & \quad \text{ADDRESS OF SET} \quad \text{**********}
\end{align*}
\]

WHERE THE FLAG IS SET BY THE COMPILER TO ONE, IF THE SET IS A TEMPORARY SET SUCH THAT IT SHOULD BE CLEARED AFTER IT IS USED AS AN INPUT TO THE LIBRARY PROGRAM. OTHERWISE IT IS SET TO ZERO. IN ALL ARGUMENTS THE X FIELD (INDEX REGISTER MODIFICATION) MAY BE NONZERO, AND INDIRECT ADDRESSING MAY ALSO BE SPECIFIED.

THE FLAG POSITION IS ALSO USED IN LIST ASSIGNMENTS FOR SIMILAR REASONS BUT IT IS EXTENDED FOR IDENTIFYING LISTS.

6.3. FREEING A SET OR LIST

THE COMPILER RECOGNIZES THE SIMPLE ASSIGN STATEMENTS,

\[
\begin{align*}
S & = \text{\textasciitilde EMPTY.} \quad \text{AND} \quad L = \text{\textasciitilde NIL.} \\
(S & = \text{\textasciitilde B}) \quad \text{AND} \quad (L = \text{\textasciitilde #})
\end{align*}
\]

AND TRANSLATES THEM INTO THE FOLLOWING CALLING SEQUENCES, RESPECTIVELY:

\[
\begin{align*}
\text{LMJ} & \quad X11,\text{\textasciitilde SZRO} \\
+ & \quad \text{LOC}(S) \\
+ & \quad \text{W.B.}
\end{align*}
\]

\[
\begin{align*}
\text{LMJ} & \quad X11,\text{\textasciitilde SLZRO} \\
+ & \quad \text{LOC}(L) \\
+ & \quad \text{W.B.}
\end{align*}
\]
6.4.  SET - OPERATIONS, -RELATIONS, -ASSIGNMENT

LET S AND T BE SETS, THE FOLLOWING SET OPERATIONS:

\[
\begin{align*}
S \cup T &= S,DF, T \\
S \cap T &= S,SM, T
\end{align*}
\]

AND SET RELATIONS:

\[
\begin{align*}
S \equiv T &= S,NE, T \\
S \in T &= S,NIN, T
\end{align*}
\]

ARE TRANSLATED INTO THE CALLING SEQUENCES OF THE FORM:

\[
\begin{align*}
LMJ &\quad X11,G$XXXX \\
+ &\quad F*S \\
+ &\quad F*T \\
+ &\quad W,B.
\end{align*}
\]

WHERE G$XXXX IS AS FOLLOWS:

\[
\begin{align*}
G$SUN &\quad \text{FOR } \cup \text{,} \\
G$SIT &\quad \text{FOR } \cap , \\
G$SDF &\quad \text{FOR } \DF, \\
G$SM &\quad \text{FOR } \SM, \\
G$SNIN &\quad \text{FOR } \IN, \\
G$SNE &\quad \text{FOR } \NE, \text{ AND } \Eq.
\end{align*}
\]

THE SET HEADER FOR THE RESULT OF THE ABOVE SET OPERATIONS WILL BE RETURNED IN REGISTER A0. THE LOGICAL RESULT OF THE SET RELATIONAL OPERATION WILL ALSO BE RETURNED IN REGISTER A0.

6.5.  SPECIAL SET FUNCTIONS

6.5.1.  CREATE FUNCTION

THE FUNCTION

\[
\text{CREATE}(0)
\]

IS TRANSLATED INTO THE CALLING SEQUENCE

\[
\begin{align*}
LMJ &\quad X11,G$CRT0 \\
+ &\quad W,B.
\end{align*}
\]
AND THE LIBRARY ROUTINE RETURNS THE ATOMIC SET IN A0.
THE CONDITIONAL CREATE FUNCTION,

\[
\text{CREATE}(P_1, V_1, P_2, V_2, \ldots, P_N, V_N)
\]

WITH PROPERTY IDENTIFIERS \( P_i \) AND PROPERTY VALUES \( V_i \) ARE TRANSLATED
INTO THE CALLING SEQUENCE

\[
\text{LMJ} \quad X_{11}, \text{GSCRT} \\
+ A \\
+ N \\
+ \text{W.B.}
\]

WHERE \( N \) IS THE NUMBER OF PAIRS OF PROPERTY IDENTIFIERS AND VALUES,
A IS THE ADDRESS OF ONE ARRAY CONTAINING THE ADDRESSES OF ALL
PROPERTY HEADERS FOLLOWED BY ALL PROPERTY VALUES:

\[
A + P_1 \\
+ P_2 \\
\ldots \\
+ P_N
\]

\[
A + N + V_1 \\
+ V_2 \\
\ldots \\
+ V_N
\]

THOSE ARRAYS ARE GENERATED BY THE COMPILER. WHEN A PROPERTY VALUE
IS OF TYPE SET, THEN THE ADDRESS OF THE VALUE, \( V_i \), IS FLAGGED. IF
THE FLAG IS ONE THEN THE SET WILL BE FREED BY THE LIBRARY ROUTINE
BEFORE RETURN.

6.5.2. SUBSET FUNCTION

THE EXPRESSION

\[
\text{SUBSET} (X, \text{<LOGICAL EXP.>})
\]

IS TRANSLATED BY THE COMPILER AS

\[
\text{LMJ} \quad X_{11}, \text{G$SUBS} \\
+ 0 \\
+ X \\
J \quad \text{'FUNCTION'} \\
+ (\text{W, B,})
\]

WHERE 'FUNCTION' IS THE ADDRESS OF AN INTERNAL SUBPROGRAM WHICH
EVALUATES THE LOGICAL EXPRESSION.

THE FUNCTION

\[
\text{SUBSET} (S, X, \text{<LOGICAL EXP.>})
\]

IS TRANSLATED SIMILARLY, BUT THE FIRST ARGUMENT IS THE ADDRESS OF
6.5.3. ELT, INDEX, SIZE AND PARITY FUNCTIONS

The functions,

\[ \text{ELT}(I,S) \cdot \text{INDEX}(X,S) \cdot \text{SIZE}(S) \cdot \text{PARITY}(S), \]

are translated as standard Fortran functions, but their names are replaced by \text{GSEL}, \text{GSINDEX}, \text{GSSIZE}, and \text{GSPARI}.

6.6. PROPERTY ASSIGNMENT AND RETRIEVAL

The retrieval of a property value, \( p(x) \)

of an atomic set \( x \) is translated into the following calling sequence

\[ \text{L} \text{M} \text{J} \quad x_{11}, \text{GSPRRT} \]
\[ + \quad p \]
\[ + \quad x \]
\[ + \quad \text{W.B.} \]

where \( p \) and \( x \) are addresses of the property header and set header, respectively. The value is returned in \( a_0 \) (or \( a_0-a_1 \) for double precision and complex) by the library routine.

The assignment of a property value, \( p(x) = v \)

to an atomic set \( x \) is translated into the following calling sequence

\[ \text{L} \text{M} \text{J} \quad x_{11}, \text{GSPRST} \]
\[ + \quad p \]
\[ + \quad v \]
\[ + \quad x \]
\[ + \quad \text{W.B.} \]

where \( p \) and \( x \) are as before, and \( v \) is the address where the value is found. In case of a set-value, \( v \) is flagged. If the flag is zero, then the set is copied; if it is one, the set is linked into the property chain directly.
6.7.  LIST ASSIGNMENT AND LIST FUNCTIONS

6.7.1.  LIST ASSIGNMENT STATEMENT

THE STATEMENT,

L = # OR L = .NIL.

IS TRANSLATED INTO THE CALLING SEQUENCE

LMJ X11, G$LZRO
+ L
+ W.B.

THE STATEMENT,

L = A1 : A2 : ... : AN

IS TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

LMJ X11, G$LIST
+ ARG
+ N
+ L
+ W.B.

WHERE N IS THE NUMBER OF ARGUMENTS, L IS THE ADDRESS OF THE LIST HEADER AND ARG IS THE ADDRESS OF A COMPILER GENERATED ARRAY CONTAINING THE ADDRESSES OF THE N ARGUMENTS INVOLVED IN THE CONCATENATION.

ARG + F1, LOC(A1)
+ F2, LOC(A2)
+ ... , LOC(AN)

THESE ADDRESSES ARE FLAGGED IN BIT POSITIONS 12 AND 13 WITH THE FOLLOWING CODE:

FI = 0 AI IS A VARIABLE WHICH HAS TO BE COPIED
= 1 AI IS A LIST WHICH HAS TO BE COPIED
= 2 AI IS A SET WHICH NEED NOT BE COPIED
= 3 AI IS A LIST WHICH CAN BE CONCATENATED WITHOUT COPYING IT

6.7.2.  LIST FUNCTIONS

THE FUNCTIONS,
FIRST(L) , DFIRST(L) ,
LAST(L) , DLAST(L)

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

\[ \text{LMJ} \ x11, \text{G$SLGET} \]
\[ + \ T \]
\[ + \ L \]
\[ + \ W.B. \]

WHERE

\[ T = 0 \quad \text{FOR FIRST} \]
\[ T = 1 \quad \text{FOR DFIRST} \]
\[ T = 2 \quad \text{FOR LAST} \]
\[ T = 3 \quad \text{FOR LAST AND DLAST} \]

AND \( L \) IS THE ADDRESS OF THE LIST HEADER,
THE VALUE IS RETURNED IN AO (AO-A1 FOR DBL.PR. AND COMPLEX) BY
THE LIBRARY ROUTINE.

6.8. GRAPH OPERATIONS

6.8.1. ASSIGN STATEMENT

The statements:

\[ \text{ASSIGN G, X} \]
\[ \text{ASSIGN G, X - Y} \]
\[ \text{ASSIGN G, X = Y, Z} \]

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCES, RESPECTIVELY:

\[ \text{LMJ} \ x11, \text{G$ASG1} \]
\[ + \ G \]
\[ + \ X \]
\[ + \ W.B. \]

\[ \text{LMJ} \ x11, \text{G$ASG2} \]
\[ + \ G \]
\[ + \ X \]
\[ + \ Y \]
\[ + \ W.B. \]

\[ \text{LMJ} \ x11, \text{G$ASG3} \]
\[ + \ G \]
\[ + \ X \]
6.8.2. DETACH STATEMENT

THE STATEMENTS,

\[
\begin{align*}
\text{DETACH G} \\
\text{DETACH G, S} \\
\text{DETACH G, S = T}
\end{align*}
\]

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCES, RESPECTIVELY:

\[
\begin{align*}
\text{LMJ} & \quad x_{11}, g \# \text{DET1} \\
& \quad + \quad g \\
& \quad + \quad \text{W.B.}
\end{align*}
\]

\[
\begin{align*}
\text{LMJ} & \quad x_{11}, g \# \text{DET2} \\
& \quad + \quad g \\
& \quad + \quad s \\
& \quad + \quad \text{W.B.}
\end{align*}
\]

\[
\begin{align*}
\text{LMJ} & \quad x_{11}, g \# \text{DET3} \\
& \quad + \quad g \\
& \quad + \quad s \\
& \quad + \quad t \\
& \quad + \quad \text{W.B.}
\end{align*}
\]

WHERE G IS THE ADDRESS OF THE GRAPH HEADER, S AND T ARE THE ADDRESSES OF THE CORRESPONDING SET HEADERS.

6.8.3. GRAPH FUNCTIONS

THE GRAPH FUNCTIONS,

\[
\begin{align*}
\text{STAR}(g, s), \quad \text{COB}(g, s), \quad \text{PBD}(g, s), \quad \text{ETC.}
\end{align*}
\]

ARE TRANSLATED INTO THE FOLLOWING CALLING SEQUENCE

\[
\begin{align*}
\text{LMJ} & \quad x_{11}, g \# \text{BDRY} \\
& \quad + \quad t \\
& \quad + \quad g \\
& \quad + \quad s \\
& \quad + \quad \text{W.B.}
\end{align*}
\]

WHERE G IS THE ADDRESS OF THE GRAPH HEADER, S IS THE ADDRESS OF
THE SET HEADER, AND T IS DEFINED AS FOLLOWS (OCTAL):

T

STAR 202
PSTAM 102
NSTAR 002
COB 212
PCOB 112
NCOB 012
ADJ 200
PADJ 100
NADJ 000
INC 201
PINC 001
NINC 101
BU 211
PBD 011
NBW 111

6.9, ITERATIVE STATEMENTS

6.9.1. WHILE STATEMENT

THE STATEMENT

DO # WHILE <LOG.EXPR>

IS TRANSLATED AS

FSXXXXA (TRANSLATION OF LOGICAL EXPRESSION)
JZ AO, FSXXXXB

# ............
# ............
J FSXXXXA
FSXXXXB ............

6.9.2. FOR ALL STATEMENT

THE STATEMENT

DO # FOR ALL X IN S

IS TRANSLATED AS

LMJ X11, G$DO1
WHERE 'TEMP' IS A TEMPORARY LOCATION ASSIGNED BY THE COMPILER.
THE THREE LIBRARY ROUTINES, G$DO1-3, ARE FOR INITIALIZATION, ITERATION AND TERMINATION OF THE LOOP, RESPECTIVELY, THE COMMUNICATION BETWEEN THESE ROUTINES IS ACCOMPLISHED BY THE CONTENTS OF 'TEMP'.
THE LAST ROUTINE, G$DO3, IS ALSO CALLED IF THE DO RANGE IS LEFT BY TRANSFER STATEMENTS (ABNORMAL EXIT).

6.10. REMOVE STATEMENTS

THE STATEMENT

REMOVE P,...,Q(5),...T,...

WITH PROPERTIES P,Q,... SETS S,T,... ARE TRANSLATED INTO CONSECUTIVE SUBROUTINE CALLS, ONE FOR EACH ARGUMENT IN THE LIST OF THE STATEMENT.
IF THE ARGUMENT IN THE STATEMENT IS A PROPERTY, THEN THE CALLING SEQUENCE IS

\[
\begin{align*}
LMJ & \quad X11,G$SRMV1 \\
+ & \quad P \\
+ & \quad W.B.
\end{align*}
\]

IF THE ARGUMENT IS A PROPERTY ON A SET, THEN

\[
\begin{align*}
LMJ & \quad X11,G$SRMV2 \\
+ & \quad Q \\
+ & \quad S \\
+ & \quad W.B.
\end{align*}
\]
IF THE ARGUMENT IS A SET, THEN

LMJ X11,GSRMV3
+ T
+ W,B.
THIS CHAPTER DESCRIBES THE OBJECT TIME LIBRARY PACKAGE OF FGRAAL. THE ENTRY POINTS ARE SUCH THAT THE FIRST TWO CHARACTERS OF THEIR NAMES ARE 'GS', EXCLUDED FROM THIS CONVENTION ARE THE SPECIAL FUNCTIONS OF FGRAAL SUCH AS EXPAND, CONTR, ETC.

THE LIBRARY PACKAGE IS DIVIDED FUNCTIONALLY INTO TEN GROUPS. THE FIRST FOUR OF THESE GROUPS, DATA REFERENCES IN THE DYNAMIC STORAGE, PROCEDURES, DYNAMIC STORAGE ROUTINE AND THE SET ROUTINES, ARE BASIC IN THE PACKAGE IN THE SENSE THAT THEY ARE CALLED BY THE ROUTINES IN THE OTHER PACKAGE. THE OTHER GROUPS ARE INDEPENDENT FROM EACH OTHER. THE FOLLOWING TABLE SUMMARIZES THE ELEMENTS IN THE TEN GROUPS.

<table>
<thead>
<tr>
<th>ELT.</th>
<th>ENTRIES</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>COMMON DATA</td>
</tr>
<tr>
<td>GSDATA</td>
<td>G$NU0</td>
<td>ZERO AS LOWER LIMIT OF UNIV_SEQ.</td>
</tr>
<tr>
<td>*GNSU</td>
<td>LAST CHEATED ELEMENT</td>
<td></td>
</tr>
<tr>
<td>GSNF</td>
<td>FIRST FREE BLOCK</td>
<td></td>
</tr>
<tr>
<td>*G$NL</td>
<td>LOWEST BLOCK ALLOCATED</td>
<td></td>
</tr>
<tr>
<td>*G$DA</td>
<td>ADDRESS OF DYNAMIC STORAGE</td>
<td></td>
</tr>
<tr>
<td>*G$D</td>
<td>DYNAMIC STORAGE</td>
<td></td>
</tr>
<tr>
<td>*G$MASK</td>
<td>MASKING BITS FOR COLUMN SETS</td>
<td></td>
</tr>
</tbody>
</table>

| PROCES | IDYST3 | FREE A 2-WORD BLOCK |
| *ELEMNT | OBTAIN ELEMENT FROM ITS PROP BLOCK |
| *REMOV | UNCHAIN A 2-WORD BLOCK |
| *REMPR | REMOVE A PROPERTY BLOCK |
| *INSRT | INSERT A 2-WORD BLOCK INTO A CHAIN |
| *SAVEX | SAVE RETURN ADDRESS |
| *PTYP | GETS TYPE GROUP OF PROPERTY |
| *SETONE | SET A HIT IN A COLUMN |
| *SETZH0 | RESET A BIT IN A COLUMN |
| *PTAG | SET PTAG ON A COLUMN SET |

<p>| DYNAMIC STORAGE ROUTINES |
| G$DYN | G$DYN | OBTAIN A 2-WORD BLOCK |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G$SZRO</td>
<td>Free a set</td>
</tr>
<tr>
<td>G$SZKO</td>
<td>Free a set if flagged</td>
</tr>
<tr>
<td>G$SZR</td>
<td>Free a set in column form</td>
</tr>
<tr>
<td>G$FREE</td>
<td>Free a set in block form</td>
</tr>
<tr>
<td>G$CRTRN</td>
<td>Free a column index</td>
</tr>
<tr>
<td>G$ASG</td>
<td>Set assignment</td>
</tr>
<tr>
<td>G$ASG</td>
<td>Set union</td>
</tr>
<tr>
<td>G$SUM</td>
<td>Symmetric sum</td>
</tr>
<tr>
<td>G$ADF</td>
<td>Set difference</td>
</tr>
<tr>
<td>G$UN</td>
<td>Set intersection</td>
</tr>
<tr>
<td>G$UNT</td>
<td>Set containment</td>
</tr>
<tr>
<td>G$EQ</td>
<td>Set equivalence</td>
</tr>
<tr>
<td>G$NE</td>
<td>Set not equal</td>
</tr>
<tr>
<td>G$CPY</td>
<td>Copy a set into a column set</td>
</tr>
<tr>
<td>G$CP1</td>
<td>Copy a set into a certain column</td>
</tr>
<tr>
<td>G$DIS</td>
<td>Put a column set into block form</td>
</tr>
<tr>
<td>G$GET</td>
<td>Obtain a free column</td>
</tr>
<tr>
<td>G$ZRO</td>
<td>Clear a column</td>
</tr>
<tr>
<td>G$SETI</td>
<td>Initialize for get-next elt.</td>
</tr>
<tr>
<td>G$INI</td>
<td>Register init. for get next</td>
</tr>
<tr>
<td>G$INQ</td>
<td>Initialization for internal use</td>
</tr>
<tr>
<td>G$NS</td>
<td>Data word for get next</td>
</tr>
<tr>
<td>G$INIC</td>
<td>Register set-up for col. search</td>
</tr>
<tr>
<td>G$NXT</td>
<td>Get next element of set</td>
</tr>
<tr>
<td>G$NXTQ</td>
<td>For internal use</td>
</tr>
<tr>
<td>G$NTB</td>
<td>Get next elt. of block set</td>
</tr>
<tr>
<td>G$NH</td>
<td>Generate header for column set,</td>
</tr>
<tr>
<td>G$NM1</td>
<td>Previous header existed</td>
</tr>
<tr>
<td>G$NM2</td>
<td>Previous header existed but new</td>
</tr>
<tr>
<td>G$NM3</td>
<td>Limits are given</td>
</tr>
<tr>
<td></td>
<td>No previous header</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G$LZRO</td>
<td>Empty a list</td>
</tr>
<tr>
<td>G$LZKO</td>
<td>List functions</td>
</tr>
<tr>
<td>G$LGET</td>
<td>List assignment</td>
</tr>
<tr>
<td>G$LIST</td>
<td>List assignment</td>
</tr>
</tbody>
</table>
* Property Routines
  * GSRMV1 * GSRMV1 * Remove Property P
  * GSRMV2 * GSRMV2 * Remove Property on a Set
  * GSRMV3 * GSRMV3 * Remove Elements from Universe
  * GSPROP * GSPRS1 * Property Assignment
  * * GSPRT * Property Retrieval
  * GSPREM * GSPRKL * Remove a Property from an Element
  * GSPADD * GSPADD * Add Elt. to Property Set
  * GSPADD * GSPADD * Remove Elt. from Property Set
  * GSFNDF * GSFNDF * Find First Elt. with Property
  * GSFNUL * GSFNUL * Find Last Elt. with Property
  * GSGP11 * GSGP11 * Find Prop. Block of Elt.

* Graph Routines
  * GSA01 * GSA01 * Assign G, X
  * GSA02 * GSA02 * Assign G, X, Y
  * GSA03 * GSA03 * Assign G, X, Y, Z
  * GSDET1 * GSDET1 * Detach Full Graph
  * GSDET2 * GSDET2 * Detach Elements from Graph
  * GSDET3 * GSDET3 * Detach Links from Graph
  * GSDOMN * GSDOMN * Domain of a Property (Nodes, Arcs)
  * GSDETA * GSDETA * Detach an Arc
  * GSA0N1 * GSA0N1 * Assign a Node
  * GSGP21 * GSGP21 * Find Graph Property Block

* Special Functions
  * GSCRT * GSCRT * Conditional Create
  * GSCRT0 * GSCRT0 * Create Element
  * GS0 * GS01 * Initialization for Forall St.
  * GS02 * GS02 * Retrieve Element for Forall St.
  * GS03 * GS03 * Terminate Forall Statement
  * GSCICA * GSCICA * Property Check
  * GSELT * GSELT * Elt Function
  * GSN0X * GSN0X * Index Function
  * GSIZE * GSIZE * Size Function
  * GSPAKI * GSPAKI * Parity Function
  * GSUBS * GSUBS * Subset Function

* Diagnostic Routines
  * GSERR * GSERR * Error and Warning Routine
  * GSERED * GSERED * Error Message Data

* Special Library Routines
  * CONTR * CONTR * Contract Node Set
  * EXPAND * EXPAND * Expand Contracted Node Set
  * NODSET * NODSET * Obtain Set of Contracted Nodes
# FUNCTION TABLE

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Args, Args.</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIRST</td>
<td>1 LIST</td>
<td>RETURNS THE FIRST OR LAST ELEMENT OF THE LIST, WITH 'D', OR OF THE LIST.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DFIRST</td>
<td>*</td>
<td>MENT OF THE LIST, WITH 'D', OR OF THE LIST.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LAST</td>
<td>*</td>
<td>IT ALSO DELETES THE ELEMENT FROM THE LIST.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLAST</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- LIST FUNCTIONS: ---

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>1</td>
<td>ZERO</td>
<td>CREATED ELEMENT AS AT. SET</td>
</tr>
<tr>
<td>CREATE</td>
<td>2N</td>
<td>PROP,NAME*</td>
<td>GIVES AT. SET WITH MATCHING PROPERTIES, CREATES ONE IF NOT EXISTS</td>
</tr>
</tbody>
</table>

--- SET FUNCTIONS: ---

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOM</td>
<td>1</td>
<td>INT.</td>
<td>AT. SET WITH THE GIVEN SEQ. NUMBER</td>
</tr>
<tr>
<td>EUT</td>
<td>2</td>
<td>INT. SET</td>
<td>AT. SET IN SPEC. PLACE IN THE SET</td>
</tr>
<tr>
<td>INDEX</td>
<td>2</td>
<td>AT. SET</td>
<td>INDEX NO. OF AT. SET IN THE SET</td>
</tr>
<tr>
<td>SIZE</td>
<td>1</td>
<td>SET</td>
<td>NUMBER OF ELEMENTS IN THE SET</td>
</tr>
<tr>
<td>PARITY</td>
<td>1</td>
<td>SET</td>
<td>TRUE FOR ODD, FALSE FOR EVEN</td>
</tr>
<tr>
<td>COUNT</td>
<td>1</td>
<td>ZERO</td>
<td>MAXIMALSEQUENCE NUMBER</td>
</tr>
<tr>
<td>COUNT</td>
<td>1</td>
<td>AT. SET</td>
<td>SEQUENCE NUMBER OF THE AT. SET</td>
</tr>
<tr>
<td>SUBSET</td>
<td>3</td>
<td>SET, LOG</td>
<td>ELEMENTS OF FIRST, DUMMY ARGUMENT WHICH SATISFY LOG. EXPR.</td>
</tr>
<tr>
<td>CHECK</td>
<td>2</td>
<td>PROP, NAME*</td>
<td>TRUE IF PROPERTY IS DEFINED FOR AT. SET, OTHERWISE FALSE</td>
</tr>
</tbody>
</table>

--- PROPERTY FUNCTIONS: ---

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODERS</td>
<td>1</td>
<td>GRAPH</td>
<td>NODES OF THE GRAPH</td>
</tr>
<tr>
<td>ARCS</td>
<td>1</td>
<td>GRAPH</td>
<td>ARCS OF THE GRAPH</td>
</tr>
<tr>
<td>INC</td>
<td>2</td>
<td>GRAPH</td>
<td>UNION OF ALL BOUNDARY NODES OF THE GIVEN SET OF ARCS</td>
</tr>
<tr>
<td>STAR</td>
<td>2</td>
<td>GRAPH</td>
<td>UNION OF ALL ARCS INCIDENT TO THE GIVEN SET OF NODES</td>
</tr>
<tr>
<td>RD</td>
<td>2</td>
<td>GRAPH</td>
<td>SYMMETRIC SUM OF ALL BOUNDARY NODES OF THE GIVEN SET OF ARCS</td>
</tr>
<tr>
<td>COB</td>
<td>2</td>
<td>GRAPH</td>
<td>SYMMETRIC SUM OF ALL ARCS INCIDENT TO THE GIVEN SET OF NODES</td>
</tr>
<tr>
<td>ADJ</td>
<td>2</td>
<td>GRAPH</td>
<td>UNION OF ALL NODES ADJACENT TO THE NODES IN THE SET</td>
</tr>
</tbody>
</table>

--- GRAPH FUNCTIONS: ---

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODERS</td>
<td>1</td>
<td>GRAPH</td>
<td>NODES OF THE GRAPH</td>
</tr>
<tr>
<td>ARCS</td>
<td>1</td>
<td>GRAPH</td>
<td>ARCS OF THE GRAPH</td>
</tr>
<tr>
<td>INC</td>
<td>2</td>
<td>GRAPH</td>
<td>UNION OF ALL BOUNDARY NODES OF THE GIVEN SET OF ARCS</td>
</tr>
<tr>
<td>STAR</td>
<td>2</td>
<td>GRAPH</td>
<td>UNION OF ALL ARCS INCIDENT TO THE GIVEN SET OF NODES</td>
</tr>
<tr>
<td>RD</td>
<td>2</td>
<td>GRAPH</td>
<td>SYMMETRIC SUM OF ALL BOUNDARY NODES OF THE GIVEN SET OF ARCS</td>
</tr>
<tr>
<td>COB</td>
<td>2</td>
<td>GRAPH</td>
<td>SYMMETRIC SUM OF ALL ARCS INCIDENT TO THE GIVEN SET OF NODES</td>
</tr>
<tr>
<td>ADJ</td>
<td>2</td>
<td>GRAPH</td>
<td>UNION OF ALL NODES ADJACENT TO THE NODES IN THE SET</td>
</tr>
<tr>
<td>Category</td>
<td>Form</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Declaration</td>
<td>SET X,Y,...</td>
<td>X,Y,... are set variables.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'TYPE' STAQUE L,P,...</td>
<td>L,P,... are lists of 'TYPE' = REAL, INTEGER, LOGICAL or SET.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'TYPE' PROPERTY A,B,...</td>
<td>A,B,... property functions of 'TYPE' = REAL, INT., LOG., or SET.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GRAPH G('MOD'), T('MOD')...</td>
<td>G,T,... are graph of 'MOD' = DIRECTED or UNDIRECTED, and PSEUDO-, MULTI- or NODE-GRAPHS.</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>S='SET-EXPRESSION'</td>
<td>SET ASSIGNMENT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>L=...:X:L:Y:...</td>
<td>LIST ASG.: X,L,Y,... are constants, VBL., or list of same type.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P(X)= 'EXPRESSION'</td>
<td>PROPERTY 'P' assigned to atomic set X with value 'EXPRESSION'.</td>
<td></td>
</tr>
<tr>
<td>Graph</td>
<td>ASG. G,X</td>
<td>AT, set X assigned to G as node.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASG. G,X-Y</td>
<td>ATOMIC SETS X, Y are assigned as adjacent nodes in graph G.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASG. G,X-Y,Z</td>
<td>AT, SETS X, Y, Z assigned to graph G as arc Z with end-nodes X, Y.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DETACH G,S</td>
<td>ELEMENTS OF SET S ARE REMOVED FROM GRAPH G.</td>
<td></td>
</tr>
<tr>
<td>Iterative</td>
<td>EXECUTES STATEMENTS THROUGH ONE * LABELED WITH 'ST#'.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DO 'ST#' FOR ALL X, IN, S</td>
<td>FOR EACH ELEMENT X OF SET S.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DO 'ST#' WHILE LOG, EXPRESSION IS TRUE</td>
<td>WHILE LOG, EXPRESSION IS TRUE.</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>REMOVE S,T,..., A,B,...</td>
<td>REMOVES SETS S,T FROM UNIVERSE, PROPERTIES A,B FROM AT, SETS WHERE DEFINED.</td>
<td></td>
</tr>
<tr>
<td>Save-Reset</td>
<td>SAVE G,..., 'P',...</td>
<td>GRAPHS G,..., PROPERTIES P,... SAVED ON AUXILIARY STORAGE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RESET G,..., 'P',...</td>
<td>RESET FROM AUXILIARY STORAGE.</td>
<td></td>
</tr>
</tbody>
</table>