b. UNRELATE(*) = * where TYPE(*) is node or arc and x(*) does exist.

If $x(x_k(u(*)))$ is *, then $x_k(u(*))$ and x(*) are removed.

If TYPE(*) is arc then $a^+(u(*))$ is also removed.

Otherwise, (X^+ has more than one component) set $T = x^{-1}(*)$.

If $x_k(u(*))$ is * then $x_k(u(*))$ becomes T.

If TYPE(*) is arc and $a^+(u(*))$ is * then $a^+(u(*))$ becomes T.

In any case, however, x(T) becomes x(*) and x(*) is removed.

c. ATTACH(*) = * where TYPE (*) is node or arc and y(*) does not exist.

Set $T = y_k(b(*))$ if $y_k(b(*))$ exists. y(*) becomes y(T) and y(T) becomes *. If $y_k(b(*))$ does not exist then y(*) and $y_k(b(*))$ become * and if TYPE(*) is arc then $a^-(b(*))$ becomes *.

d. DETACH(*) = * where TYPE(*) is node or arc and y(*) does exist.

If $y(y_k(b(*)))$ is *, then $y_k(b(*))$ and y(*) are removed.

If TYPE(*) is arc then $a^-(b(*))$ is also removed.

Otherwise, (Y^- has more than one component) set $T = y^{-1}(*)$.

If $y_k(b(*))$ is * then $y_k(b(*))$ becomes T.

If TYPE(*) is arc and $a^-(b(*))$ is * then $a^-(b(*))$ becomes T.

In any case, however, y(T) becomes y(*) and y(*) is removed.

In order to show how the GDS is changed as a result of these operations, consider Figure 3.5 as a typical data structure represented by the GDS of Figure 4.6 given by Figure 4.7. Then the following sequence of operations generates a new GDS.

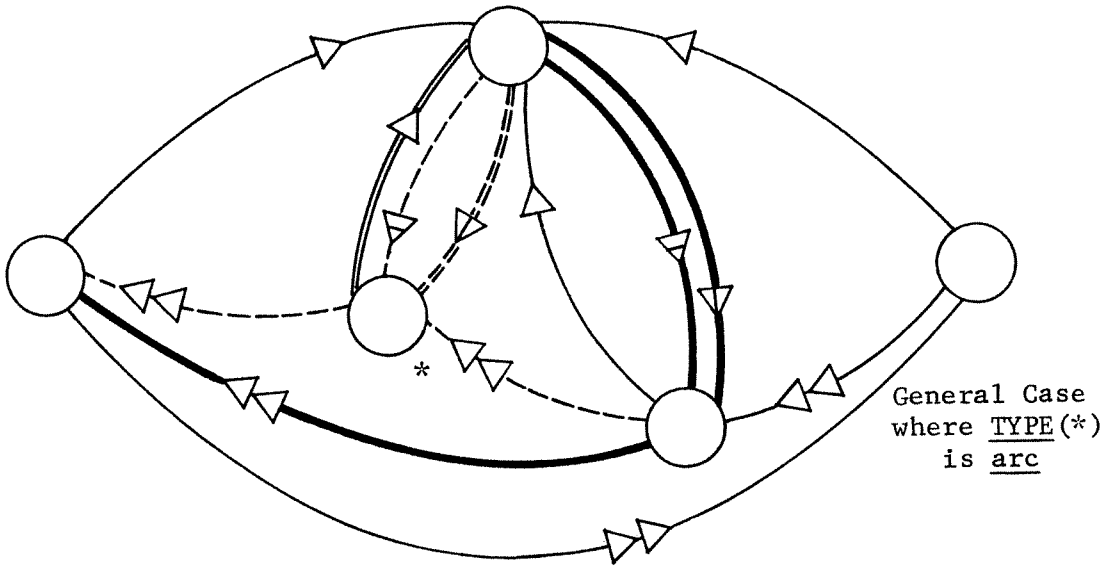


Figure 4.5. Diagram for UNRELATE(*)

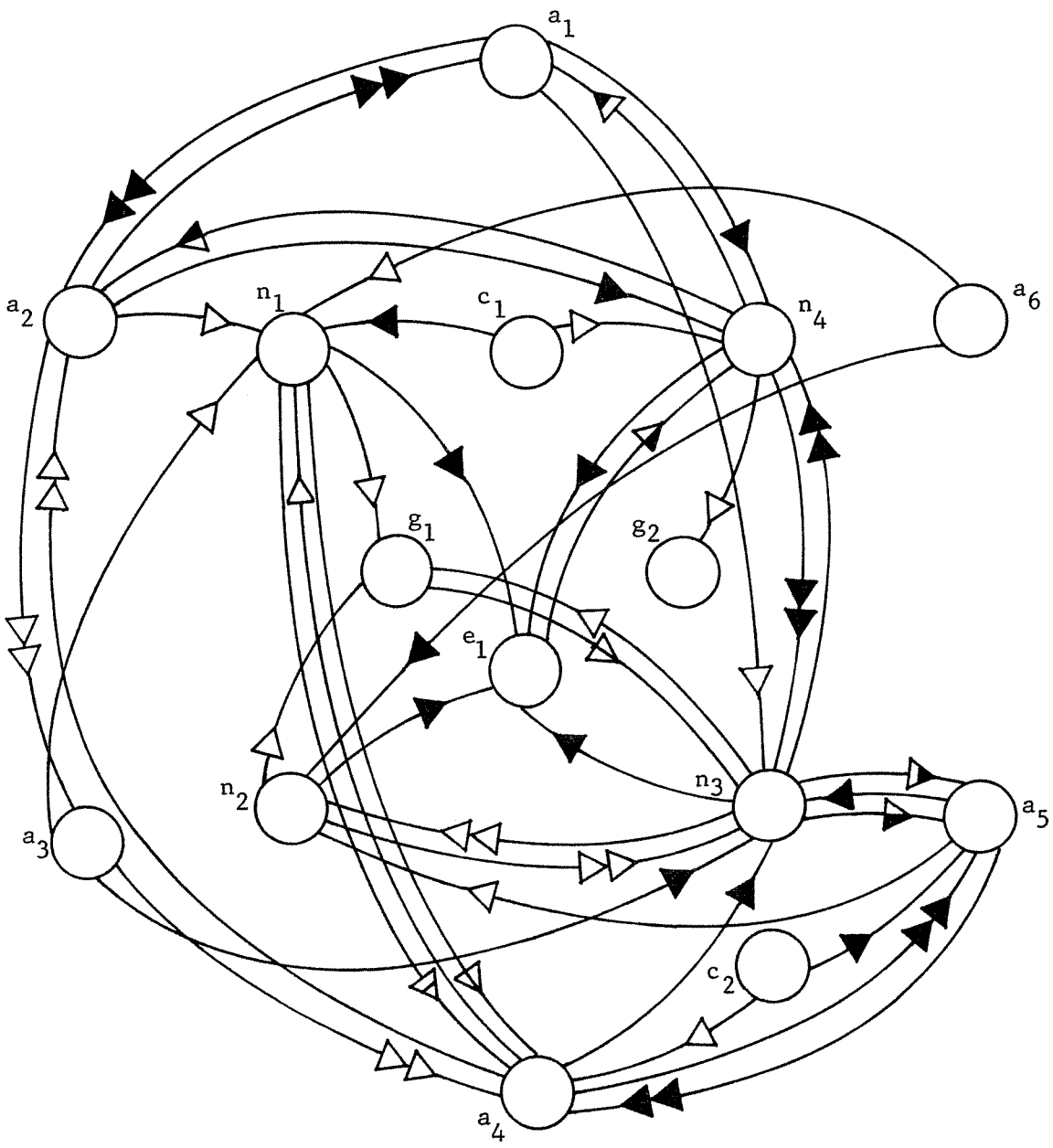


Figure 4.6. GDS for Semantic Examples

ARCS	u	b	x	x_k	$a^{\vert}$	y	y_k	a^{-}	TYPE
e_1							n_4		element
g_1				n_3					graph
g_2									graph
n_1	g_1	e_1		a_4	a_4				node
n_2	g_1	e_1	n_3						node
n_3	g_1	e_1	n_2			n_4	a_5	a_5	node
n_4	g_2	e_1				n_3	a_2	a_1	node
a_1	n_3	n_4				a_2			arc
a_2	n_1	n_4	a_3			a_1			arc
a_3	n_1	n_3	a_4						arc
a_4	n_1	n_3	a_2			a_5			arc
a_5	n_2	n_3				a_4			arc
a_6	n_1	n_2							arc
c_1	n_4	n_1							cursor
c_2	a_4	a_5							cursor

Figure 4.7. Tabular Form of GDS for Semantic Examples

RELATE(a_6) makes a_6 a related-only arc by redefining

$$\underline{ARCS}(a_4,x) = a_6 \text{ and } \underline{ARCS}(a_6,x) = a_2.$$

ATTACH(a_6) makes a_6 a regular arc by redefining

$$\underline{ARCS}(n_2,y_k),\ \underline{ARCS}(n_2,a^-),\ \text{and } \underline{ARCS}(a_6,y) = a_6.$$

UNRELATE(a_4) makes a_4 an attached-only arc by redefining

$$\underline{ARCS}(n_1,a^+) \text{ and } \underline{ARCS}(n_1,x_k) = a_3,\ \underline{ARCS}(a_3,x) = a_6,\ \text{and}$$

$$\underline{ARCS}(a_4,x) = \lambda.$$

DETACH(a_4) makes a_4 an isolated arc by redefining

$$\underline{ARCS}(a_5,y) = a_5 \text{ and } \underline{ARCS}(a_4,y) = \lambda.$$

DETACH(a_5) makes a_5 an isolated arc by redefining

$$\underline{ARCS}(a_5,y),\ \underline{ARCS}(n_3,y_k),\ \text{and } \underline{ARCS}(n_3,a^-) = \lambda.$$

Notice here that if x^{-1} is also stored, then removing structures from the sets is not a function of length. More importantly, a user ought to be able to state for a particular problem whether his algorithm requires detaching or unrelating. Whenever it is necessary to destroy the whole graph, all that must be done is to remove access to the graph, and the garbage colector gobbles up the whole structure. We would recommend for a first implementation that all x^{-1} and y^{-1} arc labels be stored explicitly. The reason for this recommendation is that graphs don't normally grow so large that they can't be stored; on the contrary, the difficult problems in graph processing are the result of the combinatorial nature of the graph algorithms.

4. Structural changing operations

a. HANG(*,**) = * where TYPE(*) ϵ T-set and TYPE(**) ϵ T-set or ** is λ. If ** is λ, v(*) is removed. If not, v(*) becomes **.

b. CHANGE-CURRENT-ARC-OUT(*) = * where TYPE(*) is arc and x(*) exists. $a^{+}(u(*))$ becomes *.

c. CHANGE-LAST-OF-RELATED-SET(*) = * where TYPE(*) is node or arc and x(*) exists. $x_k(u(*))$ becomes *.

d. CHANGE-CURRENT-ARC-IN(*) = * where TYPE(*) is arc and y(*) exists. $a^{-}(b(*))$ becomes *.

e. CHANGE-LAST-OF-ATTACHED-SET(*) = * where TYPE(*) is node or arc and y(*) exists. $y_k(b(*))$ becomes *.

f. CHANGE-ORIGIN(*,**) = * where TYPE(*) is cursor, if TYPE(u(*)) is TYPE(**) then u(*) becomes **; if not, b(*) also becomes **. If TYPE(*) is node and TYPE(**) is graph or TYPE(*) is arc and TYPE(**) is node then u(*) becomes ** if x(*) does not exist.

g. CHANGE-OBJECT(*,**) = * where TYPE(*) is cursor, if TYPE(b(*)) is TYPE(**) then b(*) becomes **; if not, u(*) also becomes **. If TYPE(*) is node and TYPE(**) is element or TYPE(*) is arc and TYPE(**) is node then b(*) becomes ** if y(*) does not exist.

Once again, using Figure 4.6, the following sequence of state and structural changing operations forms a new GDS.

CHANGE-ORIGIN(a_5,n_1) makes a_5 have the ORIGIN, n_1, by redefining

ARCS(a_5,u) = n_1 and

CHANGE-CURRENT-ARC-OUT(a_2) makes a_2 the CURRENT-ARC-OUT(n_1) by redefining

ARCS(n_1,a^+) = a_2 and

RELATE(a_5) makes a_5 a regular arc by redefining

ARCS(a_4,x) = a_5 and ARCS(a_5,x) = a_2 and

CHANGE-LAST-OF-RELATED-SET(a_3) makes a_3 the LAST-OF-RELATED-SET(n_1) by redefining ARCS(n_1,x_k) = a_3.

5. Traversing operations (simple)

a. TRAVERSE-RELATED-SUCCESSOR(*) = $\begin{cases} x(b(*)) \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$,

where TYPE(*) is cursor. If x(b(*)) exists, then b(*) becomes x(b(*)).

b. TRAVERSE-RELATED-PREDECESSOR(*) = $\begin{cases} x^{-1}(b(*)) \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$,

where TYPE(*) is cursor. If $x^{-1}(b(*))$ exists then b(*) becomes $x^{-1}(b(*))$.

c. TRAVERSE-ATTACHED-SUCCESSOR(*) = $\begin{cases} y(b(*)) \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$

where TYPE(*) is cursor. If y(b(*)) exists then b(*) becomes y(b(*)).

d. TRAVERSE-ATTACHED-PREDECESSOR(*) = $\begin{cases} y^{-1}(b(*)) \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$

where TYPE(*) is cursor. If $y^{-1}(b(*))$ exists, then b(*) becomes $y^{-1}(b(*))$.

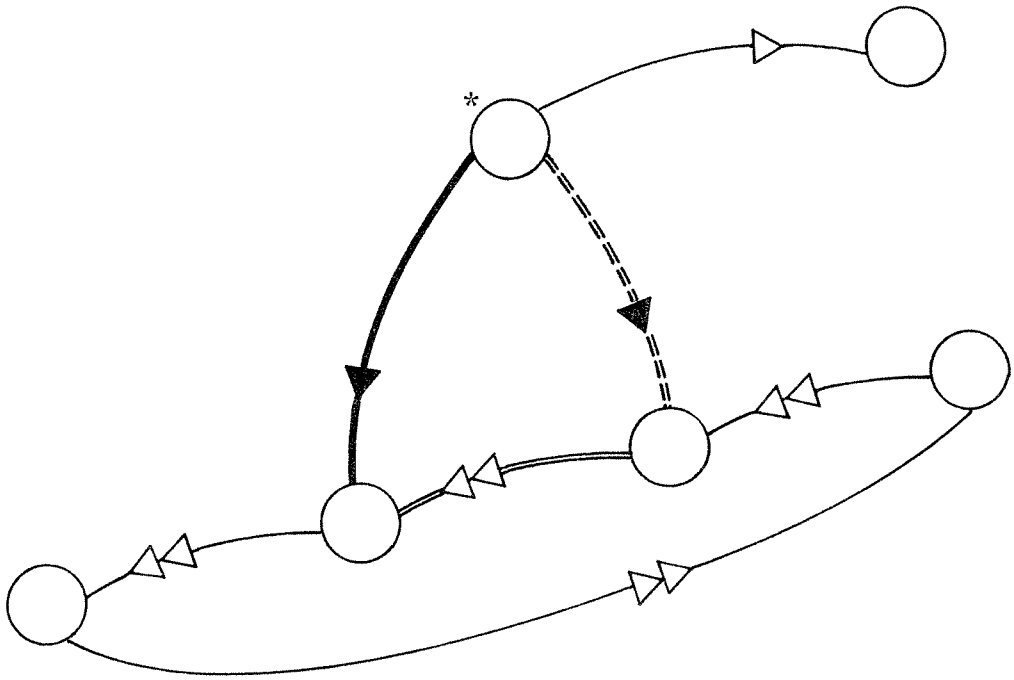


Figure 4.8. Diagram for TRAVERSE-RELATED-SUCCESSOR(*)

Traversing operations (complex)

e. TRAVERSE-NODE-OUT(*) = $\begin{cases} \alpha \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$,

where TYPE(*) is cursor, TYPE(b(*)) is node, and α is $x(a^{+}(b(*)))$. If α exists then $a^{+}(b(*))$ becomes α and b(*) becomes $b(\alpha)$.

f. TRAVERSE-GRAPH-OUT(*) = $\begin{cases} \alpha \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$,

where TYPE(*) is cursor, and TYPE(b(*)) is node. If there is an arc α such that $u(x(\alpha)) = b(*)$ and $u(b(\alpha)) = u(b(*))$ with minimum $\underline{\text{pathlength}}_x(a^{+}(b(*)),\alpha)$ then $a^{+}(b(*))$ becomes α and b(*) becomes $b(\alpha)$. (See note below.)

g. TRAVERSE-NODE-IN(*) = $\begin{cases} \alpha \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$,

where TYPE(*) is cursor, TYPE(b(*)) is node, and α is $y(a^{-}(b(*)))$. If α exists then $a^{-}(b(*))$ becomes α and b(*) becomes $u(\alpha)$.

h. TRAVERSE-GRAPH-IN(*) = $\begin{cases} \alpha \text{ if it exists} \\ \lambda \text{ otherwise} \end{cases}$,

where TYPE(*) is cursor and TYPE(b(*)) is node. If there is an arc α such that $b(y(\alpha)) = b(*)$ and $u(u(\alpha)) = u(b(*))$ with minimum $\underline{\text{pathlength}}_y(a^{-}(b(*)),\alpha)$ then $a^{-}(b(*))$ becomes α and b(*) becomes $u(\alpha)$.

Note: $\underline{\text{pathlength}}_a(n,m) = \begin{cases} 1 \text{ if } \underline{\text{ARCS}}(n,a) = m \\ \underline{\text{pathlength}}_a(\underline{\text{ARCS}}(n,a),m)+1 \text{ otherwise} \end{cases}$

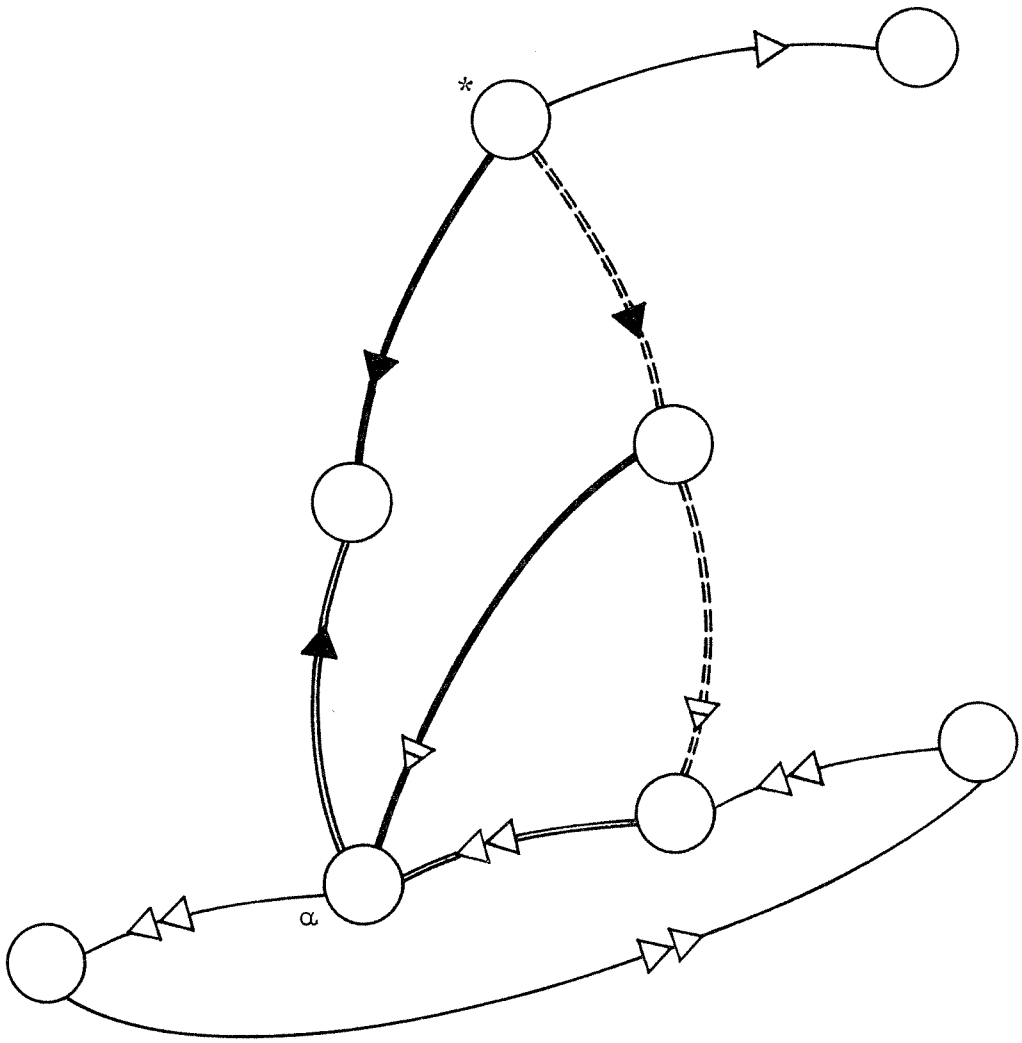


Figure 4.9. Diagram for TRAVERSE-NODE-OUT(*)

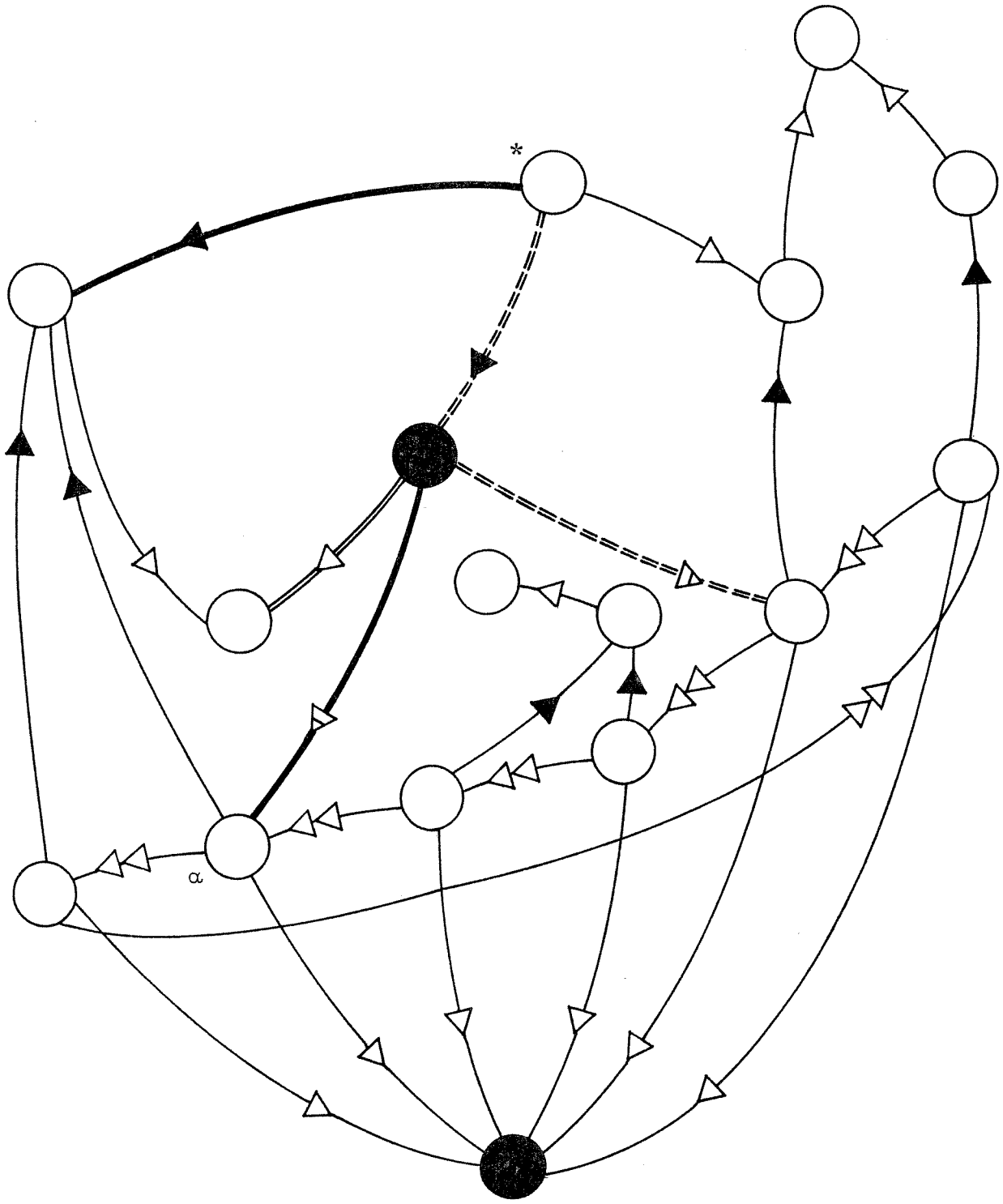


Figure 4.10. Diagram for TRAVERSE-GRAPH-OUT(*)

CHAPTER V

CONCLUSIONS

This dissertation has been concerned with two issues: the design of a graph processing language, and its formal definition. These two research efforts have resulted in the development of a graph processing language extension, GROPE, and the macro-semantics and micro-semantics two-level definitional technique.

GROPE embodies some major new ideas about representation and processing of complex structures while maintaining a serious concern with efficiency. GROPE is like LISP in that everything is dynamic. That is, the graph structures can grow, shrink, and be modified both dynamically and irregularly. In addition, GROPE has many support features which enhance the power of the graph processing primitives and special mechanisms for searching and processing graph structures. From the standpoint of the GROPE programming language extension, there is nothing new about embedding a set of operations in a high-level language. GROPE makes available to the programmer 154 operations and fifteen data types. Despite this complexity, there have been enough users of GROPE to warrant its development.

Considering the formal definitional technique, we accomplished what we set out to accomplish. That is, our formal definition is a relatively readable definition of the operations and structures available to the user (macro-semantics) and our formal definition provides an implementation design and an indication of the cost of the operations (micro-semantics). In both the macro-semantics and the micro-semantics we have used the same

approach, the abstract system approach. Thus we have shown that the abstract system approach is a viable alternative for programming language definition. One weakness of our model, however, is the lack of concern with such issues as control structure and parameter transmission.

Both the design of graph processing languages and the formal definition of programming languages have generated some interesting research problems.

Research Problems

The graph processing language GROPE has many interesting features that may or may not, in the final analysis, be considered useful graph processing tools. For example, graph readers appear to be clever mechanisms for searching graphs when combined with the current-arc-out and current-arc-in. An interesting problem would be to discover under what conditions this tool imparts a powerful algorithmic technique. Also, we introduced the notion of states of arcs and nodes. Recall that arcs and nodes can be related, attached, neither related nor attached, or both related and attached. Allowing a single graph to contain any of four types of nodes and any of four types of arcs could (and has) produced some interesting approaches to graph algorithms and the representation of graphs. Perhaps these structures could be the seed for some mathematical development as an extension of digraph theory. The wide range of flexibility as discussed in Chapter II could also lead to new schemes for representation. Finally, if the ideas of graph processing are further explored, a good research area would be to design a graph processing machine.

In the area of formal definition, there are a number of interesting research problems. The macro-semantics and the micro-semantics are both

abstract systems. An interesting research area would be to develop some mathematical results about these two systems. For example, a formal definition of a garbage collector could be defined, and one might prove that all and only those nodes which were available to become garbage would become so. One could show that by making simplifications or additions to the formal systems, other languages such as SLIP or LISP might be defined. Furthermore, an interesting result would be to prove that the micro-semantics and the macro-semantics were equivalent. Let us map out a strategy for this problem.

Define f as a one-to-one correspondence which shows for any gds (the formal system of the macro-semantics) how it can be mapped into a GDS (the formal system of the micro-semantics). For example, part of the correspondence would contain: "$x \in G$ if and only if TYPE(f(x)) = graph" . In addition, we can extend f over the operations such that f(op) = OP where op is an operation defined over the macro-semantics and OP, similarly named, is defined over the micro-semantics. Thus we try to prove the following theorem.

THEOREM. The macro-semantics are f-equivalent to the micro-semantics if and only if for all op defined over the macro-semantics

$$op(gds,x_1,x_2,\ldots,x_n) = (gds',v) \text{ if and only if}$$

$$f(op)(f(gds),f(x_1),f(x_2),\ldots,f(x_n)) = (f(gds'),f(v)).$$

Similarly, we could prove that the application of op to a gds with legal parameters leaves the gds "well defined." The same analysis could be made on the micro-semantics.

It has long been understood that one of the major bottlenecks in the application of computers to old as well as new problems is the fact that in general each problem must undergo a number of transformations (some very complicated) to make it compatible with the language which the computer scientist decides to use. Languages such as GROPE, in which the data structures correspond more closely to those used to describe problems in applications areas, should hopefully lead to the eventual elimination of this bottleneck.

APPENDIX A

The listing presented here is the GROPE language manual for the CDC 6600 implementation. Although the manual is self-contained, a few words of direction are suggested.

The chart on page 120 is a VENN diagram. For example, a list is a lnsr, a rdsr an object, and a value whereas a node is not a lnsr. In addition, the ∨ can be read as "only contain(s)." For example, the grset only contains graphs, the rseto and rseti only contain arcs. The chart on page 121 is also a Venn diagram.

Page 122 is a table of contents which does not give page numbers and is organized by classification of the function. The *'s are an indication of how important it is to understand a particular subclass of functions in order to program in GROPE (the more *'s, the more important). Because there are 154 functions in GROPE, we decided that placing the functions in alphabetical order was perhaps a reasonable way to organize the functions since the table of contents classification is available.

Also, the following correspondences hold:

Chapter II	Appendix A
mapft	mapft
orft	orft
andft	andft
dmapft	delft
lmapft	loft
smapft	soft
crgraph(x)	relate(cregr(x))
detgraph(x)	unrel(x)
detnode(x)	unrel(detnd(x))
detarc	detrc

Note that no other mapping functions that appear in Chapter II are in Appendix A.

****
* GROPE *
****

THERE ARE CERTAIN CONVENTIONS EMPLOYED IN THIS DESCRIPTION OF THE GROPE FUNCTIONS; THEY ARE BRIEFLY DEFINED AS FOLLOWS :

ARGUMENTS TO FUNCTIONS

ARG : SOME ARGUMENT -- NOT NECESSARILY A GROPE VALUE.

BITS : AN EXTERNAL REFERENCE OR A ONE-DIMENSIONAL ARRAY, POSSIBLY OF LENGTH 1 (HENCE A VARIABLE), WITH USER-SPECIFIED INFORMATION TO BE PASSED TO AN ATOM CREATION FUNCTION.

FUN : AN EXTERNAL REFERENCE OR A FUNCTION-ATOM.

G : A GRAPH.

L : A LIST.

LS : A LINEAR STRUCTURE (A LIST, SET, OR READER).

N : A NODE.

NG : A NODE OR A GRAPH.

NUM : A NUMBER.

OBJ : AN OBJECT; THE FUNCTION DEFINITION MAY STIPULATE RESTRICTIONS ON THE KINDS OF OBJECTS THAT ARE VALID.

P : A PAIR.

PS : A PSET.

Q : A DUMMY ARGUMENT; IT IS ALWAYS IGNORED.

RC : AN ARC.

RDR : A READER; THE FUNCTION DEFINITION MAY STIPULATE RESTRICTIONS ON THE KINDS OF READERS THAT ARE VALID.

RS : A READABLE STRUCTURE; X IS A READABLE STRUCTURE IF IT IS A LINEAR STRUCTURE OR A NODE OR A GRAPH.

TAPENUMBER : AN INTEGER TAPE REFERENCE.

VAL : SOME ARGUMENT -- NECESSARILY A GROPE VALUE.

VECTOR : A ONE-DIMENSIONAL ARRAY WITH NO USER-SPECIFIED INFORMATION WITHIN THE ARRAY.

CONCERNING MNEMONIC FUNCTION NAMES

-ATT- MEANS ATTACH[ED]

CH- OR CHA- MEANS CHANGE

CR- OR CRE- MEANS CREATE

-CUR- MEANS CURRENT

DEL- MEANS DELETE

DET- MEANS DETACH

-END MEANS END

-FT MEANS THE SECOND ARGUMENT IS A FUNCTION -- WHETHER AN EXTERNAL REFERENCE OR A FUNCTION-ATOM.

HS- MEANS HAS

-I MEANS IN

IS- DENOTES A PREDICATE FUNCTION THAT RETURNS ITS ARGUMENT [TRUE] OR ZERO [FALSE].

-IS- MEANS ISOLATED [UNATTACHED]

MA- MEANS MAKE

MOVE- MEANS MOVE [A READER] DIRECTLY

-O MEANS OUT

-RC- MEANS ARC

-REL- MEANS RELATE[D]

SERT- MEANS INSERT

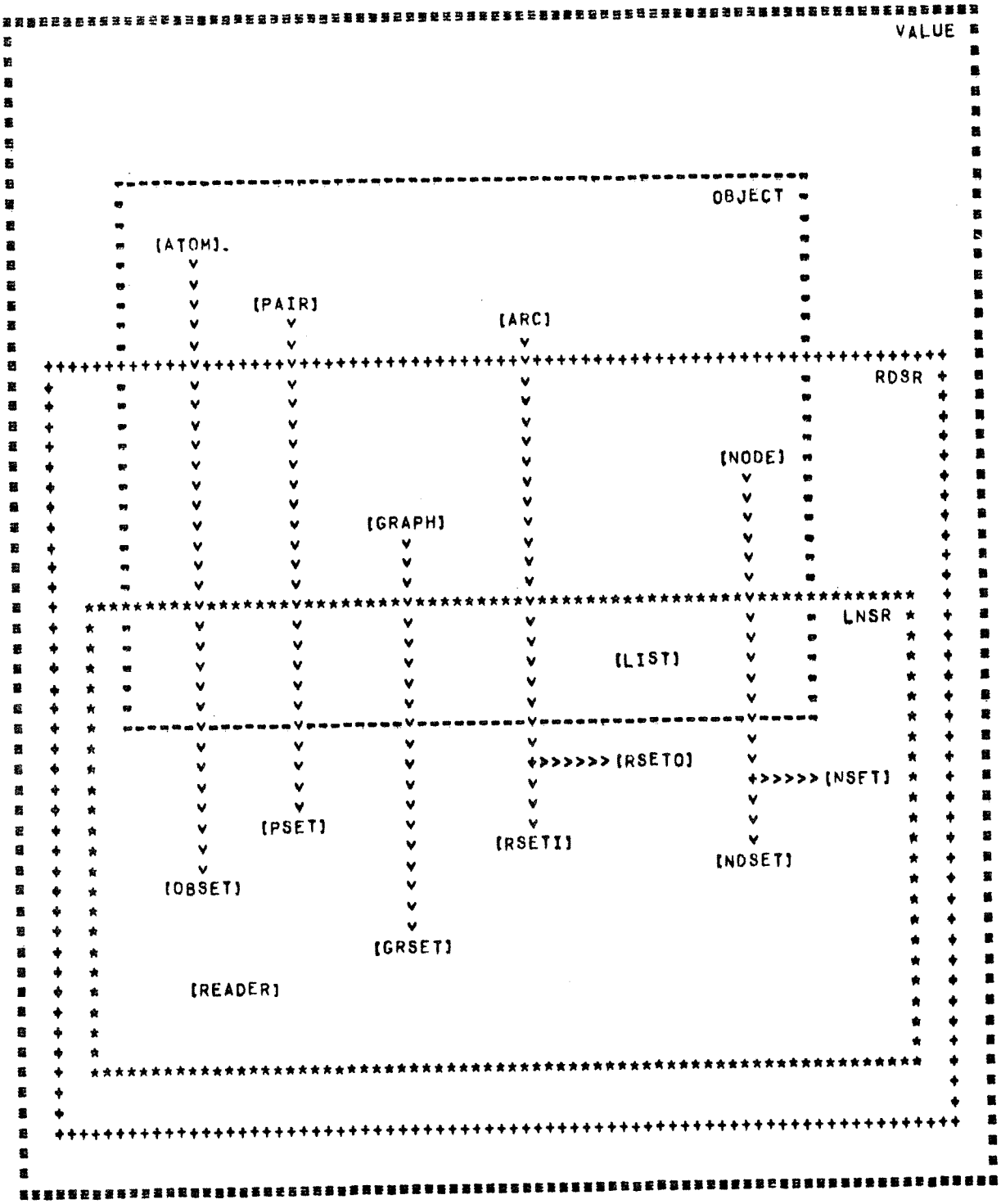
-SET- OR -ST- MEANS SET

-SR MEANS STRUCTURE

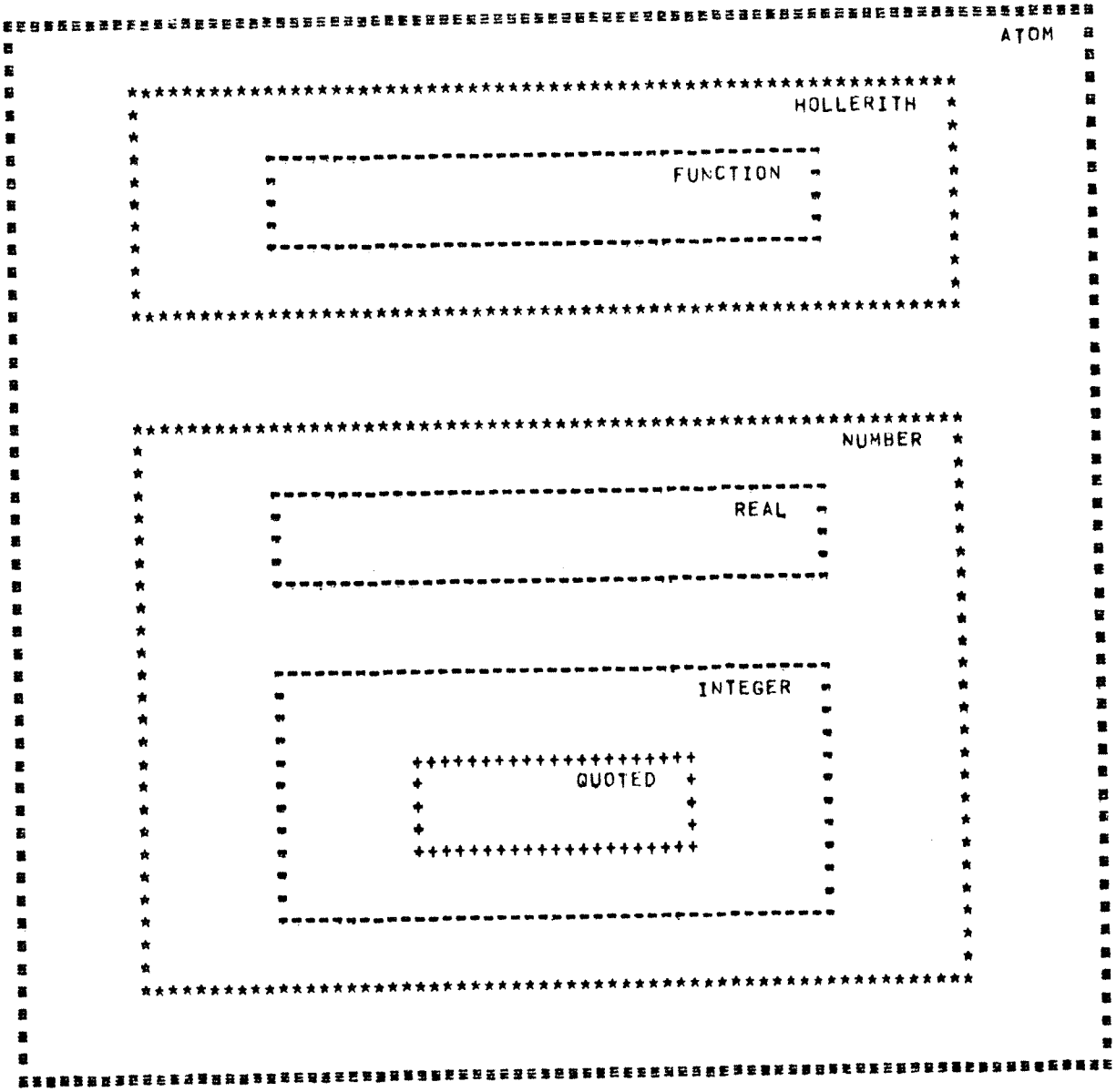
T- MEANS TRAVERSE [A READER]

GROPE ADDS ELEMENTS TO [SYSTEM] SETS BY EITHER STACKING [IF IN MODES] OR QUEUEING [IF IN MODEQ]; THE USER MAY CHANGE THE MODE.

RELATIONSHIPS AMONG THE GROPE DATA-TYPES



RELATIONSHIPS AMONG THE GROPE ATOM-TYPES



THE RELATIVE IMPORTANCE OF FUNCTION-GROUPS

ATOM AND LIST PROCESSING FUNCTIONS

SETUP/SAVAR/SAVCOM/RETURN		********
ATOM/QUOTE	ISATOM/ISHOL/ISINT/ISREAL/ISFUN/ISGINT/ISNUM	********
IMAGE/LENATM		********
CREL/LIST/QUEUE/STACK/CONCAT/POP	ISLIST	********
IDENT/EQUAL		********
APPLY/EQULFT/COMPOS		********

GRAPH PROCESSING FUNCTIONS

	ISGRAF	******
CREGR	ISNODE	******
ANODE/CRNODE/CRISN		******
GRAPH/CHAGR	ISREL	******
RELATE/UNREL	ISARC	******
CRARC/CRARCI/CRARCO/CRISR		******
FRNODE/CHAFRN/TONODE/CHATON/REVRC	ISRCI/ISRCO	****
ATTRC/DETRC/ATTRCI/DETRCI/ATTRCO/DETRCO	ISATTN	**
ATTND/DETND		**
CURCI/MACURI/CURCO/MACURO	ISOBJ	******
OBJECT/CHOBJ	ISVAL	****
VALUE/HANG/UNHANG/BUMP		

PROPERTY-SET FUNCTIONS

		********
PUT/GET	ISPSET	****
PSET/CREPS/HSPSET	ISPAIR	**
CRISP	ISATTP	**
ATTPR/DETPR		**
AFFIX/APPEND		

LINEAR STRUCTURE FUNCTIONS

		********
FIRST/LAST	ISLNSR	********
LENGTH		********
MEMBER		********
ANDFT/ORFT/MAPFT/LOFT/SOFT/DELFT	ISGRST/ISNDSET/ISNSET	******
GRSET/NDSET/NSET/HSNSET	ISRSTI/ISRSTO	******
RSETI/RSETO	ISOBST	**
OBSET		

READER PROCESSING FUNCTIONS

CREEDR/ORIGIN/CHORIG/REED/RESTRT/COPYRD	ISRDR/ISRDSR	********
TO/TI		********
DSEND/DSNDTO/ASEND/REVERT/CONECT	ISDEEP	******
SERTO/SERTI/SUBST/DELETE/MERGE		******
CHEND/MOVEND/MOVETO	ISATND/ISATBG	****
TIFT/TOFT/CURARC		****

INPUT / OUTPUT AND MISCELLANEOUS FUNCTIONS

		********
RDFILE/RDEXP/ECHO		********
PRFILE/PRXPFT/PRINFT/TERPRI/MARGIN/TAB	ISMODS	****
MODEQ/MODES		******
INTGER/REALE/TRUE		**
MAXERR/MESAGE		

AFFIX (OBJ,PS) : = OBJ, THE PSET PS IS AFFIXED TO OBJ [IN ADDITION TO ANY OTHER OBJECTS TO WHICH IT MAY BE AFFIXED], HENCEFORTH HSPSET(OBJ) = PS, [ONLY ONE PSET MAY BE AFFIXED TO ANY GIVEN OBJECT.]

ANDFT (LS,FUN,ARG2,ARG3,ARG4,ARG5) : = LS, PROVIDED THAT APPLY(FUN,ELEMENT, ARG2,ARG3,ARG4,ARG5) RETURNS TRUE [≠0] FOR EACH SUCCESSIVE ELEMENT IN THE LINEAR STRUCTURE LS; IF FUN RETURNS FALSE [=0], THE PROCESS IS TERMINATED AND ZERO IS RETURNED.
NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

ANODE (OBJ,G) : IF THERE IS A NODE N IN THE NSET(OBJ) WITH GRAPH(N) = G THEN THAT NODE IS THE VALUE, ELSE ANODE : = CRNODE(OBJ,G).

APPEND (OBJ,PS) : = OBJ, THE ELEMENTS OF THE PSET PS ARE QUEUED INTO THE PSET(OBJ) AND PS BECOMES EMPTY.

APPLY (FUN,ARG1,ARG2,ARG3,ARG4,ARG5) : = BITS(ARG1,ARG2,ARG3,ARG4,ARG5) WHERE FUN = ATOM(BITS,-3) FOR SOME EXTERNAL REFERENCE BITS; OR IF FUN IS AN EXTERNAL REFERENCE APPLY : = FUN(ARG1,ARG2,ARG3,ARG4,ARG5).
NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

ASEND (RDR) : THE VALUE IS THE READER PASSED IN THE CALL TO THE DESCEND FUNCTION WHICH RETURNED THE VALUE RDR; THIS CORRESPONDS TO THE SECOND READER IN THE STACK TO WHICH RDR REFERS. RDR IS NOT SIDE-AFFECTED.

ATOM (BITS,NUM) : RETURNS AN ATOMIC OBJECT ATM;
(1) IF NUM > 0 THEN ISHOL(ATM) IS TRUE AND NUM IS TAKEN TO BE AN INTEGER INDICATING THE MAXIMUM NUMBER OF CONTIGUOUS WORDS OF BITS CONTAINING THE DISPLAY-CODE PRINT-IMAGE OF THE ATOM, LEFT-JUSTIFIED, ZERO FILL. IF BLANK FILL IS GIVEN, ZERO FILL WILL BE SUBSTITUTED. THE VECTOR BITS IS SCANNED FROM RIGHT TO LEFT FOR THE FIRST NON-BLANK, NON-ZERO CHARACTER.
(2) IF NUM = 0 THEN ISINT(ATM) AND ISNUM(ATM) ARE BOTH TRUE.
(3) IF NUM IS -1 [OR -2] THEN BITS IS SINGLE [OR DOUBLE] PRECISION; ISREAL(ATM) AND ISNUM(ATM) ARE BOTH TRUE.
(4) IF NUM IS -3 THEN ATM IS A FUNCTION-ATOM. THIS IS A REFERENCE TO THE EXTERNAL FUNCTION BITS WHICH MAY BE STORED IN LISTS, ETC. AND MAY BE EXECUTED BY CALLING APPLY WITH ATM AS THE FIRST ARGUMENT. ISFUN(ATM) AND ISHOL(ATM) ARE BOTH TRUE.
(5) IN ALL CASES, ISATOM(ATM) IS TRUE.
[ONLY ONE ATOM WILL EXIST FOR ANY GIVEN BITS AND NUM.]

ATTND (N) : = N, IF ISATTN(N) IS TRUE, THERE IS NO EFFECT; ELSE THE NODE N IS STACKED OR QUEUED INTO THE NSET(OBJECT(N)).

ATTPR (P,PS) : = P. IF ISATTP(P) IS TRUE, THERE IS NO EFFECT; ELSE THE PAIR P IS STACKED OR QUEUED INTO THE PSET PS.

ATTRC (RC) : = ATTRCO(ATTRCI(RC)).

ATTRCI (RC) : = RC, IF ISRCI(RC) IS TRUE, THERE IS NO EFFECT; ELSE THE ARC RC IS STACKED OR QUEUED INTO THE RSETI(TONODE(RC)).
[IT CAN BE SAID THAT THE ARC IS ATTACHED IN.]

ATTRCO (RC) : = RC, IF ISRCO(RC) IS TRUE, THERE IS NO EFFECT; ELSE THE ARC RC IS STACKED OR QUEUED INTO THE RSETO(FRNODE(RC)).
[IT CAN BE SAID THAT THE ARC IS ATTACHED OUT.]

BUMP (OBJ,NUM) : = OBJ, THE [QUOTED INTEGER] VALUE(OBJ) IS INCREMENTED BY NUM. IF VALUE(OBJ) RETURNS ZERO [NO HANGING], THEN QUOTE(NUM) IS HUNG.

CHAFRN (RC,N) : = RC, THE FRNODE(RC) BECOMES N; THE ATTACHED RELATIONSHIP (ISRCO) BETWEEN THE OLD FRNODE AND THE ARC IS MAINTAINED BETWEEN THE NEW FRNODE AND THE ARC.

CHAGR (N,G) : = N, THE GRAPH(N) BECOMES G, AND THE RELATED CONDITION (ISREL) OF THE NODE N IS MAINTAINED. (IT MAY BE SAID THAT THE NODE N NOW RESIDES ON THE GRAPH G.)

CHATON (RC,N) : = RC, THE TONODE(RC) BECOMES N; THE ATTACHED RELATIONSHIP (ISRCI) BETWEEN THE OLD TONODE AND THE ARC IS MAINTAINED BETWEEN THE NEW TONODE AND THE ARC.

CHEND (RDR) : = RDR, THE SET OR LIST BEING READ BY RDR IS ALTERED SO THAT ISATND(RDR) BECOMES TRUE. THAT IS, THE RDR POINTS TO THE NEW END OF THE SET OR LIST.

CHOBJ (OBJ1,OBJ2) : = OBJ1, THE OBJECT(OBJ1) BECOMES OBJ2, AND ALL ATTACHED AND RELATED RELATIONSHIPS ARE MAINTAINED FOR OBJ1.
NOTE : OBJ1 MUST NOT BE AN ATOM.

CHORIG (RDR,RS) : = RDR. FIRST THE READER IS RESTARTED (SEE RESTRT), AND THEN ITS ORIGIN IS CHANGED TO THE READABLE STRUCTURE RS.

COMPOS (VAL,FUN1,FUN2,ARG2,ARG3) : = APPLY(FUN1,APPLY(FUN2,VAL,ARG3), ARG2,ARG3).

CONCAT (L1,L2) : = L1, THE LISTS L1 AND L2 ARE CONCATENATED, AND L2 BECOMES EMPTY. THE LENGTH OF L1 BECOMES THE SUM OF THE FORMER LENGTHS OF L1 AND L2. LAST(L1) BECOMES THE FORMER LAST(L2).

CONECT (RDR1,RDR2) : = RDR1. ESSENTIALLY, RDR1 AND RDR2 ARE CONCATENATED. RDR2 IS UNAFFECTED, BUT RDR1 IS ENLARGED BY THE ADDITION OF THE ELEMENTS OF RDR2 AT THE END OF THE READER (STACK) RDR1.
ISDEEP(RDR1) IS ALSO GUARANTEED TRUE.

COPYRD (RDR) : CREATES AND RETURNS A READER X SUCH THAT:
(1) ORIGIN(X) = ORIGIN(RDR)
(2) REED(X) = REED(RDR)
(3) ISDEEP(X) = Ø.

CRARC (N1,OBJ,N2) : = ATTRC(CRISR(N1,OBJ,N2)).

CRARCI (N1,OBJ,N2) : = ATTRCI(CRISR(N1,OBJ,N2)).

CRARCO (N1,OBJ,N2) : = ATTRCO(CRISR(N1,OBJ,N2)).

CREEDR (RS) : CREATES AND RETURNS A READER X OF THE STRUCTURE RS;
RS MAY BE A NODE, GRAPH, LIST, SET OR READER.
ORIGIN(X) = RS REED(X) = Ø ISDEEP(X) = Ø

CREGR (OBJ) : CREATES AND RETURNS A NEW, EMPTY, UNRELATED GRAPH WITH OBJECT OBJ.

CREL (OBJ) : CREATES AND RETURNS A NEW EMPTY LIST WITH OBJECT OBJ.

CREPS (Q) : CREATES AND RETURNS A NEW, EMPTY, NON-AFFIXED PSET.

CRISN (OBJ,G) : CREATES AND RETURNS A NEW ISOLATED NODE N :
OBJECT(N) = OBJ, GRAPH(N) = G, ISREL(N) = Ø,
AND ISATTN(N) = Ø.

```
CRISP (OBJ,VAL) : CREATES AND RETURNS A NEW ISOLATED PAIR P :
      OBJECT(P) = OBJ AND VALUE(P) = VAL.

CRISR (N1,OBJ,N2) : CREATES AND RETURNS A NEW ISOLATED ARC RC:
      FRNODE(RC) = N1    OBJECT(RC) = OBJ    TONODE(RC) = N2

CRNODE (OBJ,G) : = RELATE(ATTND(CRISN(OBJ,G))).

CURARC (Q) : RETURNS THE MOST RECENT ARC [CURRENT ARC] CROSSED BY A
      NODE OR GRAPH READER.

CURCI (N) : RETURNS THE CURRENT ARC INCOMING TO NODE N; ELSE Ø IF THERE IS
      NONE.  THE EXISTENCE OF ONLY ONE ARC IN THE RSETI(N) CAUSES IT TO BE
      THE CURRENT ARC INCOMING BY DEFAULT.

CURCO (N) : RETURNS THE CURRENT ARC OUTGOING FROM NODE N; ELSE Ø IF THERE IS
      NONE.  THE EXISTENCE OF ONLY ONE ARC IN THE RSETO(N) CAUSES IT TO BE
      THE CURRENT ARC OUTGOING BY DEFAULT.

DELETE (RDR) : = RDR. THE LIST READER RDR IS MOVED IN ONE POSITION [SEE TI],
      AND THE ELEMENT WHICH IT WAS PREVIOUSLY READING [SEE REED] IS
      REMOVED FROM THE LIST. IF THE FINAL REMAINING ELEMENT IS DELETED
      IN THIS MANNER [THE LENGTH(ORIGIN(RDR)) BECOMES Ø], THEN THE
      READER IS RESTARTED [SEE RESTRT]. AFTER DELETE HAS BEEN EXECUTED,
      TO(RDR) WILL MOVE RDR TO THE ELEMENT PREVIOUSLY JUST OUT FROM
      THE DELETED ELEMENT.

DELFT (LS,FUN,ARG2,ARG3,ARG4,ARG5) : = LS. THE ELEMENTS OF THE LINEAR
      STRUCTURE LS FOR WHICH APPLY(FUN,ELEMENT,ARG2,ARG3,ARG4,ARG5)
      IS TRUE ARE DELETED FROM THE STRUCTURE. WHILE IN DELFT, THE USER MAY
      NOT CHANGE THE STRUCTURE LS IN ANY WAY; ANY OTHER STRUCTURES MAY BE
      ARBITRARILY AFFECTED.
      NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

DETND (N) : = N. THE NODE N IS DETACHED FROM THE NSET(OBJECT(N)).

DETPR (P,PS) : = P. THE PAIR P IS DETACHED FROM THE PSET PS.

DETRC (RC) : = DETRCO(DETRCI(RC)).

DETRCI (RC) : = RC.  THE ARC RC IS DETACHED FROM THE RSETI(TONODE(RC)).
      [IT MAY BE SAID THAT THE ARC IS NO LONGER VISIBLE AT THE TONODE.]
      NOTE : IF THE ARC RC WAS THE CURCI(TONODE(RC)) THEN THE ARC
             IMMEDIATELY PRECEDING RC IN THE RSETI BECOMES THE NEW
             CURCI(TONODE(RC)).

DETRCO (RC) : = RC.  THE ARC RC IS DETACHED FROM THE RSETO(FRNODE(RC)).
      [IT MAY BE SAID THAT THE ARC IS NO LONGER VISIBLE AT THE FRNODE.]
      NOTE : IF THE ARC RC WAS THE CURCO(FRNODE(RC)) THEN THE ARC
             IMMEDIATELY PRECEDING RC IN THE RSETO BECOMES THE NEW
             CURCO(FRNODE(RC)).

DSEND (RDR) : = DSNDTO(RDR,REED(RDR)).

DSNDTO (RDR,RS) : = CONECT(CREEDR(RS),RDR).

ECHO (TAPENUMBER) : = TAPENUMBER. THE DATA READ BY RDEXP [SEE RDFILE]
      IS ECHO-PRINTED, A LOGICAL RECORD AT A TIME, ON FILE TAPENUMBER IF
      TAPENUMBER IS POSITIVE; OTHERWISE THE PRINTING IS SUPPRESSED.
      INITIALLY : ECHO(Ø).
```

EQUAL (OBJ1,OBJ2) : = OBJ1 IF OBJ1 AND OBJ2 ARE IDENTICAL, OR IF OBJ1 AND OBJ2 ARE LISTS SUCH THAT THEIR ELEMENTS ARE EQUAL.

EQULFT (ARG1,FUN,ARG2) : = ARG1 IF IDENT(APPLY(FUN,ARG1),ARG2) IS TRUE; ELSE EQULFT : = 0.

FIRST (LS) : RETURNS THE FIRST ELEMENT OF THE LINEAR STRUCTURE LS.

FRNODE (RC) : IN THE ARC <N1,OBJ,N2> , N1 IS THE FROM-NODE.

GET (OBJ1,OBJ2) : GENERALLY SPEAKING, GET : = VAL WHERE PUT(OBJ1,OBJ2,VAL) WAS LAST EXECUTED. MORE ACCURATELY, GET : = VALUE(P), WHERE P IS THE FIRST PAIR IN THE PSET AFFIXED TO OBJ1 SUCH THAT OBJECT(P) = OBJ2. IF NO SUCH PAIR OR PSET EXISTS, THE VALUE IS 0.

GRAPH (N) : RETURNS THE GRAPH ON WHICH NODE N RESIDES.

GRSET (Q) : RETURNS THE SET OF ALL RELATED GRAPHS.

HANG (OBJ,VAL) : = OBJ. THE VALUE(OBJ) BECOMES VAL.

HSNSET (OBJ) : RETURNS THE SET [NSET] OF ATTACHED NODES WITH OBJECT = OBJ. IF NONE HAVE BEEN ATTACHED, HSNSET : = 0.

HSPSET (OBJ) : RETURNS THE PSET AFFIXED TO OBJ; IF NONE, HSPSET : = 0.

IDENT (ARG1,ARG2) : = ARG1 IF ARG1 AND ARG2 ARE IDENTICAL; ELSE IDENT : = 0. SINCE IDENT IS A BITWISE COMPARISON, ARG1 AND ARG2 NEED NOT BE GROPE VALUES.

IMAGE (ATM,VECTOR,NUM1) : RETURNS THE (FIRST WORD OF THE) BITS PASSED IN THE CALL ATM = QUOTE(BITS) OR ATM = ATOM(BITS,NUM2) -- UNLESS NUM2 IS -3, IN WHICH CASE THE (HOLLERITH) NAME OF THAT EXTERNAL REFERENCE IS RETURNED. IF MORE THAN ONE WORD IS REQUIRED TO CONTAIN THE IMAGE OF ATM, THEN (AT MOST) THE FIRST NUM1 WORDS OF BITS IS STORED INTO VECTOR.

INTGER (ARG) : = ARG. THE MOTIVATION FOR THIS FUNCTION IS AS FOLLOWS: M = ARG CAUSES MODE CONVERSION, AND THE EFFECT IS A CATASTROPHE IF M IS TO BE USED AS A GROPE VALUE; HENCE M = INTGER(ARG).

ISARC (VAL) : = VAL IF VAL IS AN ARC; 0 OTHERWISE.

ISATBG (RDR) : = RDR IF RDR IS POINTING AT THE FIRST ELEMENT OF ITS (SET, LIST, OR READER) ORIGIN; 0 OTHERWISE.

ISATND (RDR) : = RDR IF RDR IS POINTING AT THE LAST ELEMENT OF ITS (SET, LIST, OR READER) ORIGIN; 0 OTHERWISE.

ISATOM (VAL) : = VAL IF VAL IS AN ATOM; 0 OTHERWISE. VAL IS AN ATOM IF IT WAS GENERATED BY A CALL TO ATOM OR QUOTE.

ISATTN (N) : = N IF THE NODE N IS ATTACHED TO THE NSET(OBJECT(N)).

ISATTP (P) : = P IF THE PAIR P IS ATTACHED TO SOME PSET.

ISDEEP (RDR) : = RDR IF THE READER RDR MAY BE ASCENDED; 0 OTHERWISE.

ISFUN (VAL) : = VAL IF VAL IS THE RESULT OF SOME VAL = ATOM(BITS,-3); 0 OTHERWISE.

ISGRAF (VAL) : = VAL IF VAL IS A GRAPH; 0 OTHERWISE.

ISGRST (VAL) : = VAL IF VAL IS THE GRAPHSET [GRSET]; Ø OTHERWISE.

ISHOL (VAL) : = VAL IF VAL IS AN ATOM WITH A HOLLERITH [DISPLAY-CODE] IMAGE [THAT IS, IT WAS CREATED BY A CALL TO ATOM(BITS,NUM) WITH NUM ≥ 1 OR NUM = -3]; Ø OTHERWISE.

ISINT (VAL) : = VAL IF VAL IS AN ATOM WITH AN INTEGER [BINARY] IMAGE [THAT IS, IT WAS CREATED BY A CALL TO ATOM(BITS,Ø) OR BY A CALL TO QUOTE]; Ø OTHERWISE.

ISLIST (VAL) : = VAL IF VAL IS A LIST; Ø OTHERWISE.

ISLNSR (VAL) : = VAL IF VAL IS A LINEAR STRUCTURE [THE GRSET, THE OBSET, A LIST, PSET, RSETI, RSETO, NSET, NDSET OR READER]; Ø OTHERWISE.

ISMODS (Ø) : = -1 IF GROPE IS IN THE STACK MODE; Ø OTHERWISE.

ISNDST (VAL) : = VAL IF VAL IS A NODESET [NDSET] OF SOME GRAPH; Ø OTHERWISE.

ISNODE (VAL) : = VAL IF VAL IS A NODE; Ø OTHERWISE.

ISNSET (VAL) : = VAL IF VAL IS AN NSET [OF SOME OBJECT]; Ø OTHERWISE.

ISNUM (VAL) : = VAL IF VAL IS A NUMERIC ATOM [FIXED- OR FLOATING-POINT]; Ø OTHERWISE.

ISOBJ (VAL) : = VAL IF VAL IS A GROPE OBJECT [ATOM, PAIR, ARC, NODE, GRAPH OR LIST]; Ø OTHERWISE.

ISOBST (VAL) : = VAL IF VAL IS THE OBSET; Ø OTHERWISE.

ISPAIR (VAL) : = VAL IF VAL IS A PAIR; Ø OTHERWISE.

ISPSET (VAL) : = VAL IF VAL IS A PSET; Ø OTHERWISE.

ISQINT (VAL) : = VAL IF VAL IS THE RESULT OF SOME VAL = QUOTE(BITS); Ø OTHERWISE. IF ISQINT(VAL) IS TRUE, THEN SO ARE ISINT(VAL) AND ISATOM(VAL).

ISRCI (RC) : = RC IF THE ARC RC IS IN THE RSETI(TONODE(RC)); Ø OTHERWISE.

ISRCO (RC) : = RC IF THE ARC RC IS IN THE RSETO(FRNODE(RC)); Ø OTHERWISE.

ISRDR (VAL) : = VAL IF VAL IS A READER; Ø OTHERWISE.

ISRDSR (VAL) : = VAL IF VAL IS A READABLE STRUCTURE [A LINEAR STRUCTURE, NODE OR GRAPH]; Ø OTHERWISE.

ISREAL (VAL) : = VAL IF VAL IS AN ATOM WITH A FLOATING-POINT [REAL] IMAGE; Ø OTHERWISE.

ISREL (NG) : = NG IF THE NODE OR GRAPH NG IS RELATED; Ø OTHERWISE.

ISRSTI (VAL) : = VAL IF VAL IS AN RSETI [OF SOME NODE]; Ø OTHERWISE.

ISRSTO (VAL) : = VAL IF VAL IS AN RSETO [OF SOME NODE]; Ø OTHERWISE.

ISVAL (ARG) : = ARG IF ARG IS A LEGAL GROPE VALUE : AN OBJECT, A SET, OR A READER; Ø OTHERWISE.

LAST (LS) : RETURNS THE LAST ELEMENT OF THE LINEAR STRUCTURE LS.

LENATM (ATM) : = THE NUMBER OF WORDS CONTAINING THE IMAGE OF THE ATOM ATM.

LENGTH (LS) : RETURNS THE (INTEGER) NUMBER OF TOP-LEVEL ELEMENTS IN THE LINEAR STRUCTURE LS. IF ANY OTHER TYPE OF ARGUMENT IS PASSED, LENGTH : = -1. FOR SETS AND LISTS, THE LENGTH IS IMMEDIATELY AVAILABLE (STORED).

LIST (ARG1,ARG2,ARG3,ARG4,ARG5) : RETURNS A NEW LIST L WITH OBJECT(L) = QUOTE(Ø). IF 5 ARGUMENTS (GROPE VALUES) ARE PASSED TO LIST, THE NEW LIST WILL CONTAIN THOSE 5 ELEMENTS, OTHERWISE THE LIST IS COMPOSED OF K-1 ELEMENTS WHERE ARGK IS THE FIRST ARGUMENT WHICH IS NOT A GROPE VALUE. (Ø IS NOT A GROPE VALUE.) NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 5.

LOFT (LS,FUN,ARG2,ARG3,ARG4,ARG5) : RETURNS A NEW LIST L WITH OBJECT(L) = QUOTE(Ø). FUNCTION FUN IS APPLIED TO THE SUCCESSIVE ELEMENTS IN THE LINEAR STRUCTURE LS. IF FUN RETURNS A GROPE VALUE, VAL, THEN VAL IS QUEUED INTO THE NEW LIST. NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

MACURI (RC) : = RC. CURCI(TONODE(RC)) BECOMES RC. (THAT IS, RC BECOMES THE CURRENT ARC INTO THE TONODE(RC).)

MACURO (RC) : = RC. CURCO(FRNODE(RC)) BECOMES RC. (THAT IS, RC BECOMES THE CURRENT ARC OUT FROM THE FRNODE(RC).)

MAPFT (LS,FUN,ARG2,ARG3,ARG4,ARG5) : = LS. MAPFT APPLIES FUN TO EACH ELEMENT IN THE LINEAR STRUCTURE LS. (X = APPLY(FUN,ELEMENT,ARG2,ARG3, ARG4,ARG5) IS CALLED N TIMES WHERE N = LENGTH(LS).) NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

MARGIN (NUM) : THE MARGIN, INITIALLY SET TO 1, IS THE NUMBER OF COLUMNS ON THE LEFT OF THE GROPE OUTPUT BUFFER WHICH PRINFT WILL NOT FILL. IF NUM IS NONNEGATIVE, THEN THE MARGIN BECOMES NUM. IN ANY EVENT THE MARGIN IS RETURNED AS THE VALUE OF THE FUNCTION.

MAXERR (NUM) : = NUM. HENCEFORTH GROPE WILL ABORT THE PROGRAM AFTER NUM ERRORS. INITIALLY : MAXERR(10)

MEMBER (VAL,LS) : = VAL PROVIDED THAT VAL IS ONE OF THE ELEMENTS IN THE LINEAR STRUCTURE LS; ELSE Ø. IN MANY CASES MEMBERSHIP IN SETS (GRSET, NDSET, NSET, RSETI, RSETO, ETC.) MAY BE TESTED MORE EFFICIENTLY WITH AN APPROPRIATE IS-FUNCTION (ISREL, ISATTN, ISRCI, ISRCO, ETC).

MERGE (RDR,L) : = RDR. THE ELEMENTS IN THE LIST L ARE INSERTED IN ORDER TO THE RIGHT OF (OUT FROM) THE LIST READER RDR. L BECOMES EMPTY, AS IN CONCAT. THE EFFECT IS THAT THE NEXT TO(RDR) WILL RETURN THE VALUE WHICH WAS FORMERLY FIRST(L).

MESAGE (TAPENUMBER) : = TAPENUMBER. IF TAPENUMBER ≤ Ø THEN ERROR MESSAGES WILL NOT APPEAR, ELSE THEY WILL BE WRITTEN ON FILE TAPENUMBER. INITALLY : MESAGE(6)

MODEQ (Q) : = Ø; HENCEFORTH GROPE IS IN THE QUEUE MODE.

MODES (Q) : = -1; HENCEFORTH GROPE IS IN THE STACK MODE. INITIALLY : MODES(Q)

MOVEND (RDR) : = RDR. THE LINEAR STRUCTURE READER RDR IS MOVED DIRECTLY TO THE LAST ELEMENT IN ITS ORIGIN. IF LENGTH(ORIGIN(RDR)) = Ø, THEN RDR IS RESTARTED.

MOVETO (RDR,VAL) : = RDR, THE READER RDR MOVES DIRECTLY TO THE VALUE
VAL WITHIN ITS ORIGIN. [FOR A LIST READER, RDR IS RESTARTED,
THEN MOVED TO THE FIRST OCCURENCE OF VAL.] REED(RDR) BECOMES VAL.

NDSET (G) : RETURNS THE SET OF NODES ON GRAPH G THAT ARE RELATED.

NSET (OBJ) : RETURNS THE SET OF ATTACHED NODES WITH OBJECT = OBJ. [THE EMPTY
NSET IS CREATED IF NECESSARY.]

OBJECT (OBJ) : IF OBJ IS A PAIR, THEN OBJECT : = X WHERE OBJ = CRISP(X,V).
IF OBJ IS AN ARC, THEN OBJECT : = X WHERE OBJ = CRISR(N,X,M).
IF OBJ IS A NODE, THEN OBJECT : = X WHERE OBJ = CRISN(X,G).
IF OBJ IS A GRAPH, THEN OBJECT : = X WHERE OBJ = CREGR(X).
IF OBJ IS A LIST, THEN OBJECT : = X WHERE OBJ = CREL(X).
NOTE : OBJ CANNOT BE AN ATOM.

OBSET (Q) : RETURNS THE OBSET. THE OBSET IS THE SET OF ATOMS CREATED BY
ATOM (BITS,NUM). THE SET IS NOT ORDERED, AND THE USER MAY NOT
DIRECTLY AFFECT IT [SUCH AS WITH DELFT, SERTO, ETC], BUT IT
MAY BE SEARCHED WITH THE READER MECHANISM [USING TO, LOFT, ETC].

ORFT (LS,FUN,ARG2,ARG3,ARG4,ARG5) : = 0 IF APPLY(FUN,ELEMENT,ARG2,ARG3,
ARG4,ARG5) RETURNS FALSE [=0] FOR EACH SUCCESSIVE ELEMENT IN
THE LINEAR STRUCTURE LS; IF FUN RETURNS TRUE [≠0], THE PROCESS IS
TERMINATED AND THAT VALUE [≠0] IS RETURNED.
NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

ORIGIN (RDR) : THE ORIGIN OF THE READER RDR IS THE READABLE STRUCTURE RS USED
IN RDR = CREEDR(RS), WHETHER CALLED DIRECTLY [BY THE USER] OR BY
DSNDTO. [THAT IS, THE ORIGIN IS LOCAL TO THE CURRENT LEVEL OF
THE READER, AND [NORMALLY] CHANGES WHEN ASEND OR DSNDTO IS EXECUTED.]

POP (L) : REMOVES THE FIRST ELEMENT OF THE LIST L AND RETURNS THAT ELEMENT.

PRFILE (TAPENUMBER,NUM) : = TAPENUMBER. TERPRI WILL WRITE THE CONTENTS OF
THE GROPE OUTPUT BUFFER ON FILE TAPENUMBER OF COLUMN LENGTH NUM.
INITALLY : PRFILE(6,136).

PRINFT (VAL,FUN,AGR2,ARG3,ARG4,ARG5) : = VAL. THE FOLLOWING ALGORITHM
DETERMINES WHAT IS WRITTEN INTO THE GROPE OUTPUT BUFFER --

SET X = VAL:
(1) IF X IS AN ATOM, WRITE ITS IMAGE INTO THE BUFFER.
OR (2) IF X IS A PAIR, ARC, NODE, GRAPH, OR READER, THEN SET X =
APPLY(FUN,X,ARG2,ARG3,ARG4,ARG5); IF ISVAL(X) IS TRUE
GO TO STEP (1), ELSE DO NOT WRITE X.
ELSE (3) X MUST BE A LINEAR STRUCTURE [NOT A READER] :
WRITE THE CHARACTER (. IF X IS A LIST, WRITE ITS
OBJECT [AS ABOVE], DELIMITED BY THE CHARACTER ↓,
PROVIDED ITS OBJECT IS NOT A LIST OR QUOTE(0).
AND (4) SET X [SUCCESSIVELY] TO EACH ELEMENT IN THE
LINEAR STRUCTURE AND PROCEED AS (1) ABOVE; THEN
WRITE THE CHARACTER) AND EXIT. THE GROPE
OUTPUT BUFFER IS PRINTED AND EMPTIED [TERPRI]
ONLY AS NECESSARY TO PREVENT BUFFER OVERFLOW.

BE CAREFUL TO NOTE THAT THERE ARE MANY WAYS TO GENERATE
INFINITE LOOPS. A RECOMMENDED CALL IS PRINFT(VAL,OBJECT) WHICH
WORKS SUCESSFULLY IN MOST CASES.
NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

PRXPFT (VAL,FUN,ARG2,ARG3,ARG4,ARG5) : = TERPRI(PRINFT(VAL,FUN,ARG2,ARG3,
ARG4,ARG5)).

NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

```
PSET (OBJ) : RETURNS THE PSET AFFIXED TO OBJ.  IF NONE EXISTS, ONE IS CREATED,
       AFFIXED TO OBJ AND RETURNED.
       TO THE EXTENT TO WHICH OBJECTS SHARE THE SAME PSET, THEY
       SHARE THE SAME VALUE.

PUT (OBJ1,OBJ2,VAL) : = OBJ1. GET(OBJ1,OBJ2) BECOMES VAL. PUT STACKS
       OR QUEUES THE CRISP(OBJ2,VAL) ON THE PSET(OBJ1), UNLESS THERE
       ALREADY IS A PAIR P ON THE PSET(OBJ1) WITH THE OBJECT(P) = OBJ2,
       IN WHICH CASE HANG(P,VAL) IS EXECUTED.

QUEUE (VAL,L) : = L . THE VALUE VAL IS QUEUED ONTO THE LIST L.
       THE QUEUE-STACK MODE IS UNAFFECTED BY QUEUE.

QUOTE (BITS) : RETURNS A QUOTED INTEGER X: ISINT(X), ISQINT(X),
       AND ISATOM(X) ARE TRUE. THE ABSOLUTE VALUE OF THE INTEGER BITS MUST
       BE ≤ 131,071. X DOES NOT APPEAR IN THE GROPE SPACE (VECTOR).
       NOTE : EQUAL(ATOM(BITS,0),QUOTE(BITS)) IS FALSE.

RDEXP (Q) : RETURNS THE NEXT ATOM OR LIST (A BALANCED SET OF PARENTHESES AND
       ATOMS) IN THE INPUT BUFFER, READING LOGICAL RECORDS (CARDS) INTO
       THE BUFFER FROM THE CURRENT RDFILE AS NECESSARY TO COMPLETE THE
       OPERATION. NOTE : IF THE USER WISHES TO USE READ AND RDEXP ON THE SAME
       FILE, WHEN CHANGING THE READING MODE FROM READ TO RDEXP OR RDEXP TO
       READ ALWAYS START THE DATA ON A NEW LOGICAL RECORD (CARD). WHEN
       RDEXP ENCOUNTERS AN END-OF-FILE, RDEXP : = 0. THE OBJECT OF ANY NEW
       LIST IS QUOTE(0).

RDFILE (TAPENUMBER,NUM) : = TAPENUMBER. THE FUNCTION RDEXP WILL READ
       FROM FILE TAPENUMBER OF (COLUMN) LENGTH NUM.
       INITIALLY : RDFILE(5,80)

REALE (IARG) : = IARG. THE MOTIVATION FOR THIS FUNCTION IS AS FOLLOWS:
       X = IARG CAUSES MODE CONVERSION, AND THE EFFECT IS A CATASTROPHE
       IF X IS TO BE USED AS A GROPE VALUE; HENCE X = REALE(IARG).

REED (RDR) : RETURNS THE VALUE AT WHICH THE READER RDR IS POINTING, IF THE
       READER IS UNMOVED (NOT READING ANYTHING), REED : = 0.
       IF RDR IS AN OBSET READER, AN ATOM IS RETURNED.
       IF RDR IS A PSET READER, A PAIR IS RETURNED.
       IF RDR IS AN RSETI OR RSETO READER, AN ARC IS RETURNED.
       IF RDR IS A NODE, GRAPH, NSET OR NDSET READER, A NODE IS RETURNED.
       IF RDR IS A GRSET READER, A GRAPH IS RETURNED.
       IF RDR IS A READER READER, A READER IS RETURNED.
       IF RDR IS A LIST READER, THEN A GROPE VALUE IS RETURNED.

RELATE (NG) : = NG. THE NODE (OR GRAPH) NG IS STACKED OR QUEUED INTO THE
       APPROPRIATE NODESET (OR THE GRAPHSET) IF ISREL(NG) IS FALSE. HENCEFORTH
       ISREL(NG) IS TRUE.

RESTRT (RDR) : = RDR. THE REED(RDR) BECOMES 0. (THE READER BECOMES UNMOVED.)

RETURN (ARG) : WHEN CALL RETURN(ARG) IS THE LAST EXECUTED STATEMENT IN
       A FUNCTION, ALPHA, THEN THE FOLLOWING OCCURS:
                (1) THE LAST VECTOR SAVED BY SAVAR IS NO LONGER
                    PROTECTED FROM THE GARBAGE COLLECTOR.
                (2)         ALPHA = ARG
                            RETURN
       A TYPICAL FORTRAN-GROPE FUNCTION MIGHT BE:
                FUNCTION ALPHA (A,B,C,D)
                .
```

```
                .
                .
                CALL SAVAR(ALPHA,3)
                Y = F(A)
                ALPHA = F1(B,Y)
                Z = F2(C,D,ALPHA)
                CALL RETURN(Z)
                .
                .
                .
                END
        NOTE : THE FUNCTION NAME AND THE TWO LOCAL VARIABLES, Y AND Z,
               ARE PROTECTED FROM THE CALL SAVAR(ALPHA,3) UNTIL THE
               CALL RETURN(Z) STATEMENT.
        WARNING : WHEN RETURN IS USED, SAVAR MUST PROTECT ALL AND ONLY
                  THE LOCAL VARIABLES.

REVERT (RDR) : = RDR IF ISDEEP(RDR) = 0; ELSE REVERT : = REVERT(ASEND(RDR)).
       (REVERT RETURNS THE LAST READER IN THE STACK RDR.)

REVRC (RC) : = RC. THE ARC RC IS REVERSED - THE TONODE(RC) BECOMES THE
       FRNODE(RC), AND VICE-VERSA. THE ATTACHED RELATIONSHIPS ARE MAINTAINED:
       IF ISRCO(RC) WAS TRUE, THEN IT IS STILL TRUE.
       IF ISRCI(RC) WAS TRUE, THEN IT IS STILL TRUE.

RSETI (N) : RETURNS THE SET OF ARCS INTO NODE N THAT ARE ATTACHED IN.
       (ISRCI IS TRUE FOR ALL ARCS IN THE RSETI.)

RSETO (N) : RETURNS THE SET OF ARCS FROM NODE N THAT ARE ATTACHED OUT.
       (ISRCO IS TRUE FOR ALL ARCS IN THE RSETO.)

SAVAR (VECTOR,NUM) : = VECTOR. THE FIRST NUM VARIABLES IN VECTOR ARE SAVED -
       THAT IS, NONE OF THE GROPE STRUCTURES NAMED BY THESE VARIABLES AT
       GARBAGE-COLLECTION TIME WILL BE DESTROYED. (SEE SETUP FOR THE
       DESCRIPTION OF THE GARBAGE COLLECTOR.)
       SAVAR ZEROES OUT THE FIRST NUM VARIABLES IN VECTOR.

SAVCOM (VECTOR,NUM) : IS THE SAME AS SAVAR, BUT SAVCOM DOES NOT ZERO OUT
       THE FIRST NUM VARIABLES IN VECTOR.

SERTI (RDR,VAL) : = RDR. IF RDR IS A LIST READER, THEN THE VALUE
       VAL IS INSERTED INTO THE LIST INWARD FROM RDR; THE NEXT TI(RDR)
       WILL PRODUCE VAL.
       IF RDR IS A NODE OR GRAPH READER, THEN THE ARC VAL IS INSERTED
       IN THE RSETI(REED(RDR)) SO THAT THE NEXT TI(RDR) WILL CROSS THE ARC
       VAL (SUBJECT TO THE RESTRICTION ON THE GRAPH READER). IN THIS CASE,
       ISRCI(VAL) MUST BE FALSE BEFORE SERTI, AND WILL BE TRUE AFTERWARDS.
       ALSO, THE TONODE(VAL) MUST = REED(RDR).

SERTO (RDR,VAL) : = RDR. IF RDR IS A LIST READER, THEN THE VALUE
       VAL IS INSERTED INTO THE LIST OUTWARD FROM RDR; THE NEXT TO(RDR)
       WILL PRODUCE VAL.
       IF RDR IS A NODE OR GRAPH READER, THEN THE ARC VAL IS INSERTED
       IN THE RSETO(REED(RDR)) SO THAT THE NEXT TO(RDR) WILL CROSS THE ARC
       VAL (SUBJECT TO THE RESTRICTION ON THE GRAPH READER). IN THIS CASE,
       ISRCO(VAL) MUST BE FALSE BEFORE SERTO, AND WILL BE TRUE AFTERWARDS.
       ALSO, THE FRNODE(VAL) MUST = REED(RDR).

SETUP (VECTOR,NUM1,NUM2,NUM3) : = 0. THIS SETS UP GROPE, NO OTHER GROPE
       FUNCTION MAY BE EXECUTED UNTIL AFTER SETUP; HOWEVER, THE SYSTEM MAY BE
       RE-INITALIZED (FOR EXECUTION OF A DIFFERENT PROGRAM, FOR EXAMPLE)
       BY CALLING SETUP AGAIN.  SETUP USES VECTOR OF SIZE NUM1 DIMENSIONED
```

BY THE USER, WITH NUM1 * NUM2 [0≤NUM2<1] WORDS RESERVED FOR FULL WORDS AND NUM3 WORDS RESERVED FOR THE GARBAGE COLLECTOR STACK. THE FOLLOWING IS AN EXACT DESCRIPTION OF FULL WORD UTILIZATION IN THIS IMPLEMENTATION OF GROPE, ASSUMING AN ATOM IS ACTUALLY BEING CREATED:

(1) FOR ATOM(BITS,-NUM) - NUM FULL WORDS WHERE NUM IS 1 OR 2.
(2) FOR ATOM(BITS,0) = 1 FULL WORD IF BITS > 2 ↑ 18 - 1 .
(3) FOR NUM ≥ 1, AND MORE THAN 3 CHARACTERS IN BITS = NUM FULL WORDS.
(4) ALL OTHER CASES REQUIRE NO FULL WORDS.

THE FOLLOWING IS A DESCRIPTION OF WHAT GROPE VALUES ARE NOT DESTROYED BY THE GARBAGE COLLECTOR:

(1) THE GRSET AND THE OBSET.
(2) ALL VALUES NAMED BY THE SAVED VARIABLES. [THE USER SHOULD NOTE THAT EACH CALL TO SAVAR CONSUMES ONE WORD OF THE GARBAGE COLLECTOR STACK.]
(3) IF A LINEAR STRUCTURE IS SAVED, THEN SO ARE ITS ELEMENTS.
(4) IF A STRUCTURE X IS SAVED THEN [ASSUMING WELL DEFINED] SO ARE THE NDSET(X), HSNSET(X), RSETO(X), RSET1(X), HSPSET(X), GRAPH(X), TONODE(X), FRNODE(X), OBJECT(X), VALUE(X), ORIGIN(X), AND REED(X).

AN EXAMPLE INITIALIZATION IS :

```
DIMENSION ARRAY(2000)
BEGIN = SETUP(ARRAY,2000,0,04,200)
```

NOTE : ANY EXECUTION PARAMETER THE USER CAN ALTER [E.G. = RDFILE, PRFILE, MODEQ, ECHO, ETC.] IS NOT RE-INITIALIZED BY SETUP.

SOFT (LS,FUN,ARG2,ARG3,ARG4,ARG5) : THE EFFECT AND VALUE IS LIKE LOFT, HOWEVER, IF VAL IS ALREADY A MEMBER OF THE NEW LIST, VAL IS NOT INSERTED. THUS SOFT PRODUCES A LIST WITH NO DUPLICATED ELEMENTS. NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

STACK (VAL,L) : = L . THE VALUE VAL IS STACKED ONTO THE LIST L. THE QUEUE-STACK MODE IS UNAFFECTED BY STACK.

SUBST (RDR,VAL) : = RDR. THE VALUE VAL IS SUBSTITUTED FOR THE ELEMENT IN THE LIST WHICH IS BEING POINTED AT BY RDR. THAT IS, THE READER DOES NOT CHANGE POSITION, BUT REED(RDR) BECOMES VAL.

TAB (NUM) : IF NUM > 0, THE CURRENT OUTPUT TAB POSITION [AS ON A TYPEWRITER] BECOMES NUM - THAT IS, PRINFT BEGINS FILLING THE GROPE OUTPUT BUFFER AT COLUMN NUM. IN ANY EVENT, THE CURRENT TAB IS RETURNED AS THE VALUE OF THE FUNCTION.

TERPRI (ARG) : = ARG. TERPRI WRITES THE CONTENTS OF THE GROPE OUTPUT BUFFER ON THE CURRENT PRFILE, AND EMPTIES THE BUFFER.

TI (RDR) : = REED(RDR) ONCE THE READER RDR HAS MOVED AS DESCRIBED: IF RDR IS A LIST READER, THEN RDR TRAVERSES THE LIST INWARD ONE ELEMENT. [THAT IS, RDR MOVES LEFT ONE POSITION.] LISTS ARE CIRCULAR, SO IF THE READER WAS ON THE FIRST ELEMENT BEFORE TI, IT WILL MOVE AROUND TO THE LAST ELEMENT.
IN THE CASE OF THE UNMOVED GRAPH READER RDR,
TI = REED(MOVETO(RDR,VALUE(ORIGIN(RDR)))) . [NOTE THAT IN THIS CASE THE VALUE HANGING FROM THE GRAPH MUST BE ONE OF ITS NODES.]
IF RDR IS A NODE OR GRAPH READER, LET N=REED(RDR) [OR ORIGIN(RDR) IF THE NODE READER RDR IS UNMOVED], THEN THE READER CROSSES THE NEXT ARC IN THE RSETI(N) AFTER THE CURCI(N), AND THE REED BECOMES THE NODE TO WHICH THE READER MOVES. [IN THE CASE OF A GRAPH READER, RDR WILL SEARCH THE ARCS UNTIL IT FINDS ONE WHICH COMES FROM A NODE ON THE GRAPH WHICH IS THE ORIGIN OF THE READER. IF NONE CAN BE FOUND, THE READER DOES NOT MOVE.] AFTER THE MOVE, THE FUNCTION CURARC(Q) WILL RETURN THE ARC CROSSED BY THE READER. THE ARC CROSSED BECOMES THE CURCI OF ITS TONODE.

IF FOR ANY REASON RDR CANNOT MOVE, TI : = 0.

```
TIFT (RDR,FUN,ARG2,ARG3,ARG4,ARG5) : IS LIKE TI(RDR) FOR THE NODE OR GRAPH
       READER RDR EXCEPT THAT APPLY(FUN,RC,ARG2,ARG3,ARG4,ARG5) MUST ANSWER
       TRUE FOR THE ARC UNDER CONSIDERATION (FOR BEING CROSSED), ELSE THE NEXT
       ARC IN THE RSETI WILL BE CONSIDERED. (THAT IS, FUN IS APPLIED TO THE
       ARCS IN THE RSETI STARTING AFTER THE CURCI, UNTIL FUN ANSWERS TRUE
       OR ELSE IT HAS BEEN UNSUCCESSFULLY APPLIED TO ALL, IN WHICH CASE
       NO MOVE IS MADE.) IF FUN ANSWERS TRUE, THEN TI IS EXECUTED ACROSS
       THAT ARC, BUT THE VALUE OF TIFT IS THAT VALUE RETURNED BY FUN.
       IN THE CASE OF THE UNMOVED GRAPH READER RDR,
       TIFT = REED(MOVETO(RDR,VALUE(ORIGIN(RDR))))
       (IN THE GRAPH READER CASE, FUN IS ONLY APPLIED TO THOSE ARCS WHICH
       COME FROM A NODE ON THE GRAPH.)
       NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

TO (RDR) : = REED(RDR) ONCE THE READER RDR HAS MOVED AS DESCRIBED:
       IF RDR IS A LINEAR STRUCTURE READER, RDR TRAVERSES THE STRUCTURE
       OUTWARD ONE ELEMENT. (THAT IS, RDR MOVES RIGHT ONE POSITION.)
       SUCH STRUCTURES ARE CIRCULAR (WITH THE EXCEPTION OF THE READER),
       SO IF THE READER WAS ON THE LAST ELEMENT BEFORE TO, IT WILL MOVE
       AROUND TO THE FIRST ELEMENT (WITH THE EXCEPTION OF THE READER
       STRUCTURE, IN WHICH CASE NO MOVE IS MADE).
       IN THE CASE OF THE UNMOVED GRAPH READER RDR,
       TO = REED(MOVETO(RDR,VALUE(ORIGIN(RDR)))) . (NOTE THAT IN THIS CASE
       THE VALUE HANGING FROM THE GRAPH MUST BE ONE OF ITS NODES.)
       IF RDR IS A NODE OR GRAPH READER, LET N=REED(RDR) (OR ORIGIN(RDR)
       IF THE NODE READER RDR IS UNMOVED), THEN THE READER CROSSES THE
       NEXT ARC IN THE RSETO(N) AFTER THE CURCO(N), AND THE REED
       BECOMES THE NODE TO WHICH THE READER MOVES. (IN THE CASE OF A GRAPH
       READER, RDR WILL SEARCH THE ARCS UNTIL IT FINDS ONE WHICH GOES TO
       A NODE ON THE GRAPH WHICH IS THE ORIGIN OF THE READER. IF NONE
       CAN BE FOUND, THE READER DOES NOT MOVE.) AFTER THE MOVE,
       THE FUNCTION CURARC(Q) WILL RETURN THE ARC CROSSED BY THE READER.
       THE ARC CROSSED BECOMES THE CURCO OF ITS FRNODE.
       IF FOR ANY REASON RDR CANNOT MOVE, TO : = Ø.

TOFT (RDR,FUN,ARG2,ARG3,ARG4,ARG5) : IS LIKE TO(RDR) FOR THE NODE OR GRAPH
       READER RDR EXCEPT THAT APPLY(FUN,RC,ARG2,ARG3,ARG4,ARG5) MUST ANSWER
       TRUE FOR THE ARC UNDER CONSIDERATION (FOR BEING CROSSED), ELSE THE NEXT
       ARC IN THE RSETO WILL BE CONSIDERED. (THAT IS, FUN IS APPLIED TO THE
       ARCS IN THE RSETO STARTING AFTER THE CURCO, UNTIL FUN ANSWERS TRUE
       OR ELSE IT HAS BEEN UNSUCCESSFULLY APPLIED TO ALL, IN WHICH CASE
       NO MOVE IS MADE.) IF FUN ANSWERS TRUE, THEN TO IS EXECUTED ACROSS
       THAT ARC, BUT THE VALUE OF TOFT IS THAT VALUE RETURNED BY FUN.
       IN THE CASE OF THE UNMOVED GRAPH READER RDR,
       TOFT = REED(MOVETO(RDR,VALUE(ORIGIN(RDR))))
       (IN THE GRAPH READER CASE, FUN IS ONLY APPLIED TO THOSE ARCS WHICH
       LEAD TO A NODE ON THE GRAPH.)
       NOTE : THIS FUNCTION ACCEPTS A VARIABLE NUMBER OF ARGUMENTS UP TO 6.

TONODE (RC) : IN THE ARC <N1,OBJ,N2> , N2 IS THE TO-NODE.

TRUE (ARG) : = ARG. (THUS IF ARG≠Ø, THEN ARG IS TRUE.)

UNHANG (OBJ) : = OBJ. THE VALUE(OBJ) BECOMES Ø. (THE HANGING IS REMOVED.)

UNREL (NG) : = NG. THE NODE (OR GRAPH) NG IS REMOVED FROM THE APPROPRIATE
       NODESET (OR THE GRAPHSET), AND ISREL(NG) BECOMES FALSE.

VALUE (OBJ) : RETURNS THE MOST RECENT VALUE, VAL, SUCH THAT HANG(OBJ,VAL) WAS
       EXECUTED; IF NONE, VALUE : = Ø.
```

REFERENCES

1. Baron, R., L. Shapiro, D. P. Friedman, and J. Slocum, "Graph processing using GROPE/360," University of Iowa Computer Science Technical Report (in preparation).

2. Burstall, R. M., "Formal description of program structure and semantics in first-order logic," in Machine Intelligence 5 (B. Meltzer and D. Michie, eds.), American Elsevier Publishing Co., New York (1970).

3. Cashin, P. M., M. R. Mayson, and R. Podmore, "LINKNET--A structure for computer representation and solution of network problems," Australian Computer Journal 3 (August 1971).

4. Crespi-reghizzi, S., and R. Morpurgo, "A language for treating graphs," Comm. ACM 13 (1970), 319-323.

5. deBakker, J. W., "Semantics of programming languages," in Advances in Information System Science (J. Tou, ed.), Vol. 2, Plenum Press, New York (1969).

6. Earley, J., "Toward an understanding of data structures," Comm. ACM 14, 10 (Oct. 1971), 617-627.

7. Friedman, D. P., D. Dickson, J. Fraser, and T. W. Pratt, "GRASPE 1.5: a graph processor and its application," Department of Computer Science Report RS1-69, University of Houston, Houston, Texas, 1969.

8. _____, "GRASPE: graph processing a LISP extension," Computation Center Report TNN-84, University of Texas, Austin, Texas 1968.

9. _____, "Use of the intersection rules in the development of new models within GRASPE 1.5," University of Texas, April, 1970 (unpublished manuscript).

10. Greenawalt, E. M., private communication.

11. Griggs, Eric R., "Automatic Data Flow Analysis of Computer Programs," unpublished Master's thesis, University of Texas at Austin, May 1973.

12. Griswold, R. E., J. F. Poage, and I. P. Polonsky, The SNOBOL4 Programming Language, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1968.

13. Hart, R., "HINT: a graph processing language," Institute for Social Science Research Technical Report, Michigan State University, East Lansing, Michigan, 1969.

14. Hendrix, G. G., "Question answering via canonical verbs and semantic models: a model of textual meaning," Technical Report NL12, January 1973, Department of Computer Science, The University of Texas at Austin.

15. _____, "Modeling simultaneous actions and continuous processes," to appear in Artificial Intelligence Journal.

16. _____, C. W. Thompson, and J. Slocum, "Language processing via canonical verbs and semantic models," in Proceedings of the Third Annual Joint Conference on Artificial Intelligence, August 1973.

17. Knowlton, K. C., "A programmer's description of L^6, Bell Telephone Laboratories low-level linked list language," Comm. ACM 9, 8 (August 1966).

18. Landin, P. J., "Correspondence between ALGOL-60 and Church's lambda notation, Part I and II," Comm. ACM 8, 2 (February 1965), and Comm. ACM 8, 3 (March 1965).

19. Lawsen, Harold W., Jr., "PL/I list processing," Comm. ACM 6 (June 1967) 385-367.

20. Lee, J., Computer Semantics, Van Nostrand-Reinhold, 1972.

21. Lehmann, W. P., R. Stachowitz, and Bary Allan Gold, "German-English translation system," Technical Report of the Linguistics Research Center, The University of Texas at Austin (in preparation), 1973.

22. Lucas, P., and K. Walk, "On the formal description of PL/I," in Annual Review in Automatic Programming (L. Bolliet, et al., eds.), Vol. 6, Part 3, Pergamon Press,New York (1969).

23. McCarthy, J., et al., LISP 1.5 Programmer's Manual, MIT Press, Cambridge, Massachusetts, 1962.

24. Newell, Allen (ed.), Information Processing Language-V Manual, Prentice-Hall, Englewood Cliffs, New Jersey, 1961.

25. Pohl, Ira, "A method for finding Hamilton paths and Knight's tours," Comm. ACM 7 (July 1967).

26. Pratt, T. W., "A hierarchical graph model of the semantics of programs," Proceedings of AFIPS SJCC (1969), 813-825.

27. _____, "Introduction to a theory of programming language semantics," Report TSN-4, University of Texas Computation Center, 1969, 10 pp.

28. _____, "Semantic modeling by hierarchical graphs," ACM SIGPLAN Symposium on Programming Language Definition, San Francisco, Calif., August 1969.

29. _____, "Pair grammars, graph languages, and string-to-graph translations," J. of Comp. Sys. Sci., 5, 6 Dec. 1971, 560-595.

30. _____, "A formal definition of ALGOL 60 using hierarchical graphs and pair grammars," Report TSN-33, University of Texas Computation Center, 1973, 82 pp.

31. _____, and D. P. Friedman, "A language extension for graph processing and its formal semantics," Comm. ACM 14 (1971), 460-467.

32. Ross, Douglas T., "The AED free storage package," Comm. ACM 8 (August 1967), 481-492.

33. Shneiderman, B., "Data Structures: Description, Manipulation, and Evaluation," unpublished Ph.D. dissertation, State University of New York at Stonybrook, 1973.

34. Slocum, J., "Question answering via canonical verbs and semantic models: generating English for the model," Technical Report NL13, January 1973, Department of Computer Science, The University of Texas at Austin.

35. _____, "The Graph Processing Language GROPE 2.0," Master's thesis in preparation, The University of Texas at Austin.

36. Stachowitz, Rolf, Voraussetzungen für maschinelle Übersetzung: Probleme, Lösungen, Aussichten, Athenäum Verlag, Frankfurt/M, 1973.

37. _____, Ein Modell linguistischer Performanz, Athenäum Verlag, Frankfurt/M. (in vorbereitung).

38. Thompson, C. W., "Question answering via canonical verbs and semantic models: Parsing to canonical verb forms," Technical Report NL13, January 1973, Department of Computer Science, The University of Texas at Austin.

39. Wegner, P., "The Vienna definition language," Computing Surveys 4, 1 (1972), 5-64.

40. Weizenbaum, J., "Symmetric list processor," Comm. ACM 6, 9 (Sept. 1963).

41. Wesson, Robert B., "A pair grammar based string to graph translator writing system," Master's thesis in preparation, The University of Texas at Austin.

42. Wilson, James P., "Graphical representation of semantic structure," unpublished Master's thesis, The University of Texas at Austin, August 1972.

43. Wirth, N., "The programming language, PASCAL," Acta Informtica 1 (1971), 35-63.

44. Woods, W. A., "Transition network grammars for natural language analysis," Comm. ACM 13, 10 (October 1970), 591-606.