File No. UIUCDCS-F-83-911

AQINTERLISP:
An INTERLISP Program for Inductive Generalization of VL1 Event Sets

USER'S & PROGRAMMER'S GUIDE

Jeffrey M. Becker

Department of Computer Science
University of Illinois
at Urbana-Champaign

ISG 83-~~11~~ 28

September 1983

This was work supported by the National Science Foundation under grant NSF MCS 82-05166.

Table of Contents

Acknowledgments

This paper draws heavily on the work of Paul Richards, and is in part a modification of his paper "AQLISP: A LISP Program for Inductive Generalization of VL1 Event Sets", University of Illinois, 1979.

I would also like to express my appreciation to Prof. R.S. Michalski for giving me the opportunity to work on this project, and to Robert Stepp for helping me get started.

This work was supported by the National Science Foundation under grant NSF MCS 82-05166.

Changes since AQLISP (1979)

The program has been entirely recoded, and many changes have been made. The changes of greatest importance are 1) structured domain variables are now supported 2) linear domains can now have negative integer ranges, symbolic ranges, and real number ranges, 3) the data entry phase has been separated from the initialization phase to make the program more flexible and easier to use, 4) a compiled version now exists which is significantly faster than the interpreted version, and 5) internal representations of all variable types have been revised to allow the program to work on much bigger problems.

The documentation has been revised and extended to cover new features of the program. The sections on data sets, structured variables, and the global variables list are totally new. The sample dialog, machine dependent instructions, comparison with another version of AQ, and the program listing have been replaced with new versions.

AQINTERLISP:
An INTERLISP Program for Inductive Generalization of VL1 Event Sets

This paper describes the operation and internal structure of a program called AQINTERLISP, revision 15-July-1983. AQINTERLISP is an interactive INTERLISP-10 program for generalization and optimization of discriminant descriptions of object classes. The descriptions are expressed as disjunctive normal expressions in the variable valued logic system VL1 [7]. Such expressions are unions of conjunctive statements (complexes) involving relations on multiple-valued variables. Input data to the program are sets of VL1 events (sequences of attribute-value pairs) describing individual objects. Each event is associated with a given class name.

For further information on the VL1 system the reader is referred to [7], and for a detailed description of the theory behind the AQ algorithm the reader is referred to [8, 9].

1. Why AQ in INTERLISP?

There are several motivations for implementing algorithm AQ in INTERLISP. The most significant one is the ease with which dynamic structures are constructed and destroyed, which is essential to the algorithm. Data objects may be given nearly any name by virtue of the flexible naming conventions for LISP atoms. Additionally, INTERLISP lends itself to interactive communication with the user. This allows the program to be used as a "sounding board" for testing new ideas quickly and easily. There are excellent break and trace facilities which allow checking of intermediate results, as well as an in-core editor for quickly implementing modifications and extensions to data used by the program and to the program itself. Finally, many of the applications for AQ are in the areas of pattern recognition and artificial intelligence, which are often explored using LISP programs.

Section 2 describes interactive operations within the AQINTERLISP system. Described are the formats of data sets, event and class input, parameter specification, mode selection, and output format. Sections 2.7 and 2.8 provide an example of the interactive system dialog and the output generated by the various modes. Section 3 provides a description of the data structures used by the system, and a description of the principal interface to the implementation, function (AQ), which permits use of the AQ system within user written programs. Section 4 offers some comparisons of AQINTERLISP and AQ11p, a Pascal implementation of algorithm AQ. Installation dependent operating instructions and listings of system functions are provided in the appendices.

2. User's Guide

See Appendix A for installation specific instructions on starting up Interlisp and loading AQINTERLISP. It is recommended that the compiled version be used since it is approximately twenty times faster than the interpreted version. If the interpreted version is used, type (UNDO-AQMACROS) after loading AQINTERLISP and before running it to redefine the set of compiler macros as standard functions. Once the AQINTERLISP system is loaded into the INTERLISP system, a data set must be provided. A data set (described in detail below) is a special list which is the value of an atom. A menu driven interface to the top level AQINTERLISP functions may be invoked by typing (RUN). These functions may also be called directly as described below.

Data sets may be created and modified by typing (ENTERDATA DataSetName). This invokes an interactive facility for entering data.

An existing data set may be edited using the INTERLISP editor by typing (EDITV DataSetName).

A data set may also be -

Saved on disk by typing (SAVEDATA 'DataSetName 'FileName),
retrieved from disk by typing (GETDATA 'FileName),
and listed by typing (LISTDATA DataSetName).

< Note where single quotes are required! >.

After a data set is in memory, the AQ algorithm may be invoked by typing (AQVAL DataSetName). The user will be prompted for parametric information and operating mode selection. After a run is completed, the user will be allowed to select another mode of operation.

2.1. Data Sets

A data set is a list of three elements:

(Variables&Domains Order Events&Classes)

which contains all the necessary information for specifying an event space and sets of events in that event space.

2.1.1. Variables&Domains

Variables&Domains is a list of variable domain specifications each having the following form:

((var1 var2 ... varn) D (domain))

where each "vari" is a variable name, "D" is "N","L", or "S" indicating a NOMINAL, LINEAR, or STRUCTURED domain type, and "domain" is a domain specification for the variable(s).

NOMINAL domains have the form (val1 val2 ... valn) where each "vali" is one of the values which the variable may assume.

LINEAR domains have two possible forms:
1) The form (low .. high Epsilon) where "low" is the low limit and "high" is the high limit (inclusive) of the range of numeric values the variable may assume. The two dots ".." are a Lisp literal atom which serves as a place holder, and improves readability. "Epsilon" is the minimum allowed distance between values of selectors in positive and negative events along a given dimension. For Integer valued domains, "Epsilon" is always 1. For Real valued domains, "Epsilon" should be at most 1/2 of the minimum distance between data points in the current data set, and may be smaller.
2) The form (val1 val2 ... valn) where each vali is the symbolic name for a value in an ordered sequence (eg. Small, Medium, Large).

STRUCTURED domains have the form ((siblings => parent) ... (siblings => parent)) where "siblings" is a list of sibling nodes and "parent" is their immediate parent.

2.1.2. Order

The order of variables used in events must be specified since in the abbreviated form for specifying events (section 2.2.3) the variable names are not explicitly stated when a list of values is given. This list must contain all variables used in the current data set. The order must correspond to the order of values in abbreviated form events. If no abbreviated form events are specified, this list still must contain all variables used in the current data set. The form is:

(var1 var2 ... varn)

where each "vari" is the name of a variable.

2.1.3. Events&Classes

Events&Classes is a list of event specifications, each of the form:

(event => class)

where "event" is a list of values the variables assume for the event or a list of selectors, and "class" is the name of the class of which the event is an example. For example (1 2 7 => I) specifies an event where the first variable in the specified order has the value 1, the second has value 2 and the third has value 7, and the class name is "I". For variables V1, V2, and V3 this may also be specified as ((V1 = 1)(V2 = 2)(V3 = 7) => I).

Data sets may be created directly using the INTERLISP structure editor, or may be created using a text editor on another system provided that the file created will load using the INTERLISP (LOAD 'filename) command, and the result of the load is setting the value of an atom to be the data set. Convenient facilities for entering data are provided in AQINTERLISP.

2.2. Entering Data Sets

A data set may be created using a special facility in AQINTERLISP which provides modest on line help and prompting for information. This facility is invoked either via the AQINTERLISP top level menu (which is invoked by typing (RUN)), or directly by typing (ENTERDATA DataSetName). DataSetName is a name provided by the user for identifying a particular data set.

When ENTERDATA is called with a data set name which is not bound to any value, the user is directed through the following sequence by calls to functions described below:

1) Entering the Order of variables in events.
2) Entering of Variables and Domains.
3) Entering Event and Class information.

After the above steps are completed the program goes to the ENTERDATA command level where the user may exercise any of the commands described below.

Subsequent calls to ENTERDATA with a data set which has been previously defined will place the user at the ENTERDATA command level. The ENTERDATA command level prompt is ">>". Commands recognized by ENTERDATA may be listed by typing "?", or any other letter which is not recognized as a command. The list provided is:

```
Type V to enter Variables and Domains
Type O to enter Order of variables in events
Type E to enter an Event and Class information
Type L to list the data set
Type C to make changes using the LISP structure editor
Type . to exit ENTERDATA
Type any other key to see this list
```

As soon as one of these letters is typed, ENTERDATA completes a word starting with the given letter and prompts for more information. Word completion is indicated in the discussion below as "W-ord". The "-" is not actually printed by AQINTERLISP (see the sample dialog). Typing a letter not in the above list causes the list to be printed. When a prompt-input sequence has been completed, ENTERDATA returns to the command level and waits for a keypress.

2.2.1. Entering the Order of Variables in Events

Upon typing "O" ENTERDATA will prompt with:

Order of variables in events:
List all variables for the current data set in the
desired order, separated by spaces or commas.

> Type a list of all variable names for the current data set. When events are entered in the abbreviated form, the order of values given must correspond to the order of variables given here. If the order is changed after events have been entered, a warning message is printed. Values in abbreviated event specifications will NOT be rearranged to match the new order.

2.2.2. Entering Variables and Domains

Every variable to be used in the current AQINTERLISP run must be explicitly declared. Additionally, the type and domain of these variables must be provided. Variables may be NOMINAL (assume discrete values), LINEAR (assume interval values on the range [0 .. <maximum integer on system>]), or STRUCTURED (assume discrete values in a tree structure). Typing a "V" at the ENTERDATA command level starts the following dialog:

V-ariable name(s):
List variables having the same domain, separated by spaces or commas.
Or, type '.' if there are no more variables to be entered.

> Enter names of all variables that belong to the next group of identical types and domains, then type <RETURN>. Variable names should be composed of the letters A..Z and digits 0..9, and must begin with a letter (conforming to the Lisp conventions for non-numeric atoms). Several variables may be entered on a line, separated by spaces or commas.

After the variable names have been entered, ENTERDATA will prompt for information about the variable domain:

Domain Type? (N, L, S):

Three possible answers may be specified here:

N-OMINAL, L-INEAR, or S-TRUCTURED.

N-OMINAL
: to specify that these variables may assume only discrete values that are elements of the domain specified in the next question:

List the permissible values, separated by spaces or commas.

Enter all of the possible values the current variable(s) may assume. These may be any atomic symbol permitted by the INTERLISP implementation. No limit is imposed on the number of values in a domain.

L-INEAR

to specify that these variables assume interval values on the range of values specified. The domain of these variables is also specified in the next questions:

What is the range of values?
Type as '(Low ".." High)' for Integer or Real valued variables,
or '(Val1 Val2 ... Valn)' for Symbolic valued variables.

For Integer or Real valued variables, enter the low and high limits (inclusive) of the range of values the current variables may assume. The limits should both be integers within the range of integers supported on the current Lisp system, or real numbers, with Low < High. Only one interval may be specified for the domain of a linear variable.

If either of Low or High is a Real number the program will prompt with:
"What is the Epsilon value?"
Enter a small value, such as the smallest decimal units used in the events for these variables. For example, if values such as (1.01, 3.34, 7.69, ...) are to be used in the events then a good value for Epsilon would be 0.01 .

For Symbolic valued variables, enter the list of values in order from lowest to highest (eg. (small medium large)). Any atomic symbol may be used for a value.

S-TRUCTURED

to specify that these variables assume discrete values that are elements of the tree-structured domain which is specified in the next question:

What is the structure?
Type as '((siblings => parent) ... (siblings => parent))'

Enter a list of nodes separated by spaces or commas, and the immediate parent of those nodes. The tree structure may have any number of levels, but no node may have more than one parent (general graph structures are not supported). For example, ((1 2 => 8)(3,4,5 => 7)) specifies a structured domain where 8 is the parent of nodes 1 and 2, and 7 is the parent of nodes 3, 4 and 5. Any atomic symbol may be used for a node.

After the domain of the current group of variables is specified, ENTERDATA will prompt for more variable names. If there are more variable names to be entered, repeat the steps described above. Otherwise, type a period "." to continue.

2.2.3. Entering an Event and Class

Events are entered to AQINTERLISP either in an abbreviated form, or as a list of selectors in VL1 form. The VL1 form for selectors is discussed first, then the abbreviated form which may be used for rapid data entry is given. The VL1 formulae are formed by the conjunction of terms (or selectors). Each selector takes one of three forms:

1) for a NOMINAL variable, the selector format is:

[<var> <relation> <val-1> ... <val-n>]

where

<var> is a NOMINAL type variable

<relation> is either "=" or "#" to specify equality or inequality to the values specified

<val-i> is a valid value from this variable's domain. more than one value may be specified to indicate that several values belong to the same event.

2) for a LINEAR variable, the selector format is:

[<var> <relation> <lowvalue> .. <highvalue>]

or

[<var> <relation> <value>]

where

<var> is a LINEAR type variable

<relation> is either "=" or "#" as above

<lowvalue> is the lower bounding value of the interval for the event

<highvalue> is the upper bounding value of the interval for the event

<value> may be specified if only a specific point value is to be included, i.e. when <lowvalue> = <highvalue>.

3) for a STRUCTURED variable, the selector format is:

[<var> <relation> <val-1> ... <val-n>]

where

<var> is a STRUCTURED type variable

<relation> is either "=" or "#"

<val-i> is a valid value from this variable's domain. More than one value may be specified to indicate that several values belong to the same event.

Thus, if L1 is a linear variable on [0 .. 4], N1 is a nominal variable with domain (A,B,C), and S1 is a structured variable with domain ((1,2,3 => 6),(4,5 => 7)), a complete VL1 formula could be:

[L1 = 2 .. 3][N1 # B][S1 = 1 v 2]

Normally, an event will specify a single point in the event space. Thus, for each selector it is necessary to specify only the value of the variable at that point. The abbreviated form of an event is:

val1,val2, .. , valn

The order of variables associated with the values should be specified as described above. The relation in all selectors is implicitly "=".

Typing "E" at the ENTERDATA command level causes the following prompt to be printed:

E-vents and Classes:
List {all/additional} class names, separated by spaces or commas.

Enter a list of class names. Class names may be any valid Interlisp atom.

Next ENTERDATA will prompt for event information in one of two modes: 1) By individual class, or 2) For all classes. The first mode is useful when adding additional events to an existing data set. The second mode is useful for creating new data sets. The prompts are:

For each event in a class, enter a list of values
for variables {list of variable names in specified order}
in the order shown, separated by spaces or commas.
Or, enter a list of selectors as ((sel)(sel)...(sel)),
where each "(sel)" has the form (Variable Relation Vals).
Type '.' instead of an event specification to proceed to the next class.

The list of class names is: {list of all class names}

Type a class name to enter events for a particular class,

or 'ALL' to enter events for all classes.

ENTERDATA will then allow the user to enter event information for any or all classes.

Enter events for class {class name}.
Type '.' to proceed to the next class.
Class {class name} event:

Enter a list of values for the previously declared variables in the specified order, separated by spaces or commas, or a list of selectors. The values must be valid values from the variable's domain. Events in different classes should have a null intersection.

2.2.4. Listing the Data Set

Typing "L" at the ENTERDATA command level will cause the data that has been entered so far to be pretty-printed using (LISTDATA DataSet).

2.2.5. Changing data already entered

Type "C" at the ENTERDATA command level to invoke the Lisp structure editor. When editing is complete, the system will return to the ENTERDATA command level. Typing (EDITV DataSetName) at the Lisp top-level will also allow editing of the data set.

2.2.6. Exiting ENTERDATA

Type a period "." to exit the ENTERDATA function. The data set name will be printed on the screen by Lisp.

2.3. Parametric Specifications

After a data set has been provided, the AQVAL interactive driver for AQ may be invoked either from the AQINTERLISP functions menu, or directly by typing (AQVAL DataSetName). First AQVAL will prompt for parametric information. Then AQVAL will initialize certain data structures using information from the data set and will print resource utilization information for the initialization phase. Currently, two parameters must be specified interactively to the AQINTERLISP system: MaxStar, the maximum star size permitted when generating stars, and CutStar, the size to which a star will be trimmed when it exceeds MaxStar in size. These questions run as follows:

Enter maximum Star size for this run:

Enter an integer number greater than zero. The upper bound on this parameter is determined by space available to the program, and CPU time available for processing. Too large a value may cause excessive

garbage collections, complete exhaustion of free-space, or use very large amounts of computer time. Too small a value will cause excessive trimming of the stars generated, which lowers the probability that a truly optimal cover will be found. (Note that if any trimming is done at all, the covers generated by AQ are not necessarily optimal).

Enter size to cut Star to when truncating:

Enter an integer number greater than zero, and less than MaxStar, mentioned above. The closer this value is to CutStar, the more overhead caused by star trimming will increase, with a subsequent increase in CPU time used.

The parameter that determines optimality criteria, Criteria&ToleranceList, is always initialized to ((#COVERED 0.0) (NUMBEROFSELECTORS 0.0)), which causes covers produced by the system to always be selected from complexes that cover the largest number of events, and in the case of ties, those with the fewest number of terms in VL1 format. (See section 3.3 for more information on the parameters.)

2.4. Operating mode selection

There are four "modes" in which the AQ algorithm may be applied to the event clusters:

1) Intersecting covers
2) Disjoint covers (with possibly intersecting complexes)
3) Disjoint covers and Disjoint complexes
4) Sequential (VL)

2.4.1. Intersecting covers

Intersecting covers are generated by applying AQ in the following manner: let Ei represent the set of events to be covered for class i, and F be the set of all events specified to the system. Each cover Ci is constructed by applying AQ to Ei against F - Ei. Thus, the intersection of any two covers Ci ^ Cj, i#j may be non-null. The intersection will not contain any event points originally specified as an event, it only can occur over unspecified events.

2.4.2. Disjoint covers (with intersecting complexes)

Disjoint covers are generated by applying AQ in this manner: Sort the classes into alphabetic order by classname, and assign each class a unique index i. Thus E1 = <events in class with 'first' alphabetic name>, and so

on. Let F = U{Ei}, i.e., all events explicitly specified. Let Cj = <cover of class j> and Ej = <events covered by Cj>. The covers for class i are generated by applying AQ to Ei against F - Ei + U{Cj} - U{Ej}, i≠j. In this manner, it is guaranteed that Ci ^ Cj = {}, i≠j.

2.4.3. Disjoint covers and disjoint complexes

Disjoint complex covers are produced in a similar manner to Non-intersecting covers, except that the star generation of AQ is performed in such a way as to guarantee each complex in the cover is disjoint. In the previous two modes, this may not be the case - complexes within the same cover can have non-null intersections in those modes.

2.4.4. Sequential mode

Sequential mode produces covers in the following manner: first, the classes are sorted as before. Then the cover Ci for class i is produced by applying AQ to Ei against F - Ej. To utilize the covers produced by this mode, they must be tested in the same sequential order that they were constructed (i.e., alphabetic classname order). Each cover may contain any event points allocated to a previously generated cover. This mode is useful in classifying new events with a minimum of variable testing, since fewer complexes are needed to specify some covers.

2.4.5. Specifying the operating mode

The operating mode choice is entered in response to:

```
Select mode of operation by typing 1, 2, 3, or 4:
   1: Intersecting Covers
   2: Disjoint Covers (with possibly intersecting complexes)
   3: Disjoint Covers and Disjoint Complexes
   4: Sequential (VL)

Type the corresponding number to select the mode desired.
```

After the mode is selected, AQ will be applied as described, and the covers will be computed and printed, along with resource utilization information. After all of the covers are printed the following question is printed:

Do you want to try another mode?

Enter YES if you want to try another mode on the same data, otherwise enter NO. This allows you to try any of the four modes on the same data. If YES is entered, the "Mode of operation?" prompt is reissued. If NO is entered, a garbage collection is performed, and (AQVAL) returns to the calling function (either the AQINTERLISP menu, or top-level EVAL). As an added convenience, you may also type in the mode number at this point and skip the "Mode of operation?" prompt.

2.5. Program Output

Interactive AQINTERLISP produces its covers as a series of disjunctive VL1 formulas for each class. Each is printed in VL1 format. The output appears as:

The Covers are:

Cover of Class: <first class name in alphabetical order>
[<vl term 1>][<vl term 2] ... [<vl term n>]
[<vl term n+1>][<vl term n+2>]...[<vl term n+m>]
.
.
.

where each line represents a separate interval. The cover is the union of all intervals' printed under the heading line. This is repeated for each class covered by AQ. If no intervals are printed, the cover is the entire event space.

Under a heading and to the left of each complex information about the number of input events in the current class which the complex covers will be printed.

2.6. Other messages printed by AQINTERLISP

Certain questions accept only a few specific answers, such as when a mode is selected, or the domain type of a variable is defined. Generally, if an unexpected response is entered the user will be given a list of choices and the program will wait for another response.

During initialization, if an error in the data set is detected INITIALIZE will invoke the Interlisp editor on the data set, then restart itself after editing is completed. Where this is not possible, due to the current structure of the program, function (SHOULDNT) is invoked to abort the run, print "Shouldn't happen!" and cause a break. Type "^" to exit the break, then (RUN) to re-enter the AQINTERLISP functions menu.

Several other places within AQINTERLISP, consistency checks are made on the internal structures of AQ. If unexpected forms are detected, function (SHOULDNT) is invoked. These aborts are usually caused by improper entry of variables, events, or class names which INITIALIZE failed to detect.

If the message "Error in AQ" is printed, a potential infinite loop has been detected in the AQ function. This can be caused by placing events in separate classes which have a non-empty intersection.

After one of these aborts, exit the break and use the AQINTERLISP facilities to correct the data set and restart AQVAL.

2.7. Sample AQINTERLISP Dialog

Below is an example terminal dialog, entering variables, events/classes, and associated output that will be used in section 2.8. User input is underlined. Comments are in brackets: "{ comment }". This run was done on the CRL Xerox Dolphin using a compiled version of AQINTERLISP.

```
_(LOAD 'JMBSAQLISP.DCOM)

        { Use invokes the AQINTERLISP functions menu }
_(RUN)


    AQINTERLISP Functions

A: AQVAL
    The AQVAL driver for the AQ algorithm.

E: ENTERDATA
    Interactive facility for entering and modifying data sets.

G: GETDATA
    Loads a data set from a file.

L: LISTDATA
    Pretty prints a data set.

S: SAVEDATA
    Saves a data set on a file.

Q: Quit
    Leave this menu.

Type the corresponding letter to invoke the desired function: ENTERDATA

Data Set Name ? DATAX
                        { This is a new data set, so ENTERDATA }
                        { goes directly to Order List entry.   }

Order List:
List all variables for the current data set in the
desired order, separated by spaces or commas.
V1 V2 V3 V4

Variable declarations:
List variables having the same domain, separated by spaces or commas.
Or, type '.' if there are no more variables to be declared.
V1 V3

Domain Type? (N, L, S): NOMINAL
List the permissible values, separated by spaces or commas.
```

1, 2

Variable declarations:
List variables having the same domain, separated by spaces or commas.
Or, type '.' if there are no more variables to be declared.
V2

Domain Type? (N, L, S): LINEAR
What is the range of values?
Type as "Low '..' High" for Integer or Real valued variables,
  or as "Val1 Val2 ... Valn" for Symbolic valued variables.
1 .. 3

Variable declarations:
List variables having the same domain, separated by spaces or commas.
Or, type '.' if there are no more variables to be declared.
V4

Domain Type? (N, L, S): STRUCTURED
What is the structure?
Type as '((siblings => parent) ... (siblings => parent))'.
((1 2 => 5)(3 4 => 6))

Variable declarations:
List variables having the same domain, separated by spaces or commas.
Or, type '.' if there are no more variables to be entered.
.

Events and Classes:
List all class names, separated by commas or spaces.
A B C

For each event in a class, enter a list of values
for variables (V1 V2 V3 V4)
in the order shown, separated by spaces or commas.
Or, enter a list of selectors as ((sel)(sel) ... (sel)),
where each '(sel)' has the form (Variable Relation Vals).
Type '.' instead of a list of values to proceed to the next class.

The list of class names is: (A B C).

Type a class name to enter events for a particular class,
or 'ALL' to enter events for all classes.
ALL

Enter events for class A.
Type '.' to proceed to the next class.
Class A event: 1 1 1 1
Class A event: 1 3 1 3
Class A event: 2 2 2 3
Class A event: .

```
Enter events for class B.
Type '.' to proceed to the next class.
Class B event: 1 2 1 2
Class B event: 1 2 2 1
Class B event: 1 1 2 w
Class B event: .

Enter events for class C.
Type '.' to proceed to the next class.
Class C event: 2 1 1 2
Class C event: 2 3 1 2
Class C event: ((V1 = 2)(V2 = 3)(V3 = 2)(V4 = 2)) {VL1 format shown here}
Class C event: .



Type V to enter Variables and Domains
Type O to enter Order of variables in events
Type E to enter an Event and Class information
Type L to list the data set
Type C to make changes using the LISP structure editor
Type . to exit ENTERDATA
Type any other key to see this list


>> List          { User lists the data set to check for errors. }

Variables & Domains:
Variable(s) V1 V3  have NOMINAL domain (1 2)
Variable(s) V2  have LINEAR domain (1 .. 3)
Variable(s) V4  have STRUCTURED domain ((1 2 => 5) (3 4 => 6))

Order of variables in events is: ALPHA

Events & Classes:
(1 1 1 1 => A)
(I 3 1 3 => A)
(2 2 2 3 => A)
(1 2 1 2 => B)
(1 2 2 1 => B)
(1 1 2 w => B)    { oops - the w should be a 2 }
(2 1 1 2 => C)
(2 3 1 2 => C)
(2 3 2 2 => C)

>> Change Data Set     { Lisp editor is invoked, editor dialog not shown }
DATAX

>> .      { User exits ENTERDATA }
```

```
    AQINTERLISP Functions        { Control returns to the functions menu. }

A: AQVAL
    The AQVAL driver for the AQ algorithm.

E: ENTERDATA
    Interactive facility for entering and modifying data sets.

G: GETDATA
    Loads a data set from a file.

L: LISTDATA
    Pretty prints a data set.

S: SAVEDATA
    Saves a data set on a file.

Q: Quit
    Leave this menu.        { User invokes AQVAL driver for the AQ algorithm }

Type the corresponding letter to invoke the desired function: AQVAL

Data Set Name ? DATAX

Welcome to AQINTERLISP <Revision 15-July-1983>

Enter maximum Star size for this run: 25

Enter size to cut Star to when truncating: 10

Initialization:
351 conses
0.747 seconds
1.068 seconds, real time

Select mode of operation by typing 1, 2, 3, or 4:
   1: Intersecting Covers
   2: Disjoint Covers (with possibly intersecting complexes)
   3: Disjoint Covers and Disjoint Complexes
   4: Sequential (VL)

  #1

674 conses
1.55 seconds
1.966 seconds, real time


Event Coverage    The Covers are:
 Total
                  Cover of Class: A
    2             [V3 = 1][V4 = 6,1]
```

```
   2              [V4 = 6]

                  Cover of Class: B
   2              [V1 = 1][V2 = 2]
   2              [V1 = 1][V3 = 2]

                  Cover of Class: C
   3              [V1 = 2][V4 = 5]


Do you want to try another mode? 2

297 conses
0.713 seconds
0.728 seconds, real time


Event Coverage    The Covers are:
 Total
                  Cover of Class: A
   2              [V3 = 1][V4 = 6,1]
   2              [V4 = 6]

                  Cover of Class: B
   2              [V1 = 1][V4 = 2]
   2              [V1 = 1][V3 = 2][V4 = 5]

                  Cover of Class: C
   3              [V1 = 2][V4 = 2]


Do you want to try another mode? YES

Select mode of operation by typing 1, 2, 3, or 4:
   1: Intersecting Covers
   2: Disjoint Covers (with possibly intersecting complexes)
   3: Disjoint Covers and Disjoint Complexes
   4: Sequential (VL)

  #3

311 conses
0.704 seconds
0.717 seconds, real time


Event Coverage    The Covers are:
 Total
                  Cover of Class: A
   2              [V3 = 1][V4 = 6,1]
   1              [V3 = 2][V4 = 6]
```

```
                    Cover of Class: B
   2                [V1 = 1][V4 = 2]
   1                [V3 = 2][V4 = 1]

                    Cover of Class: C
   3                [V1 = 2][V4 = 2]


Do you want to try another mode? 4

251 conses
0.599 seconds
0.609 seconds, real time


Event Coverage      The Covers are:
 Total
                    Cover of Class: A
   2                [V3 = 1][V4 = 6,1]
   2                [V4 = 6]

                    Cover of Class: B
   3                [V1 = 1]

                    Cover of Class: C
   3


Do you want to try another mode? NO

{ At this point a garbage collection is invoked.  This may take over a }
{ minute on the Vax and 5 - 15 seconds on the Dolphin.                  }

    AQINTERLISP Functions

A: AQVAL
    The AQVAL driver for the AQ algorithm.

E: ENTERDATA
    Interactive facility for entering and modifying data sets.

G: GETDATA
    Loads a data set from a file.

L: LISTDATA
    Pretty prints a data set.

S: SAVEDATA
    Saves a data set on a file.

Q: Quit
    Leave this menu.
```

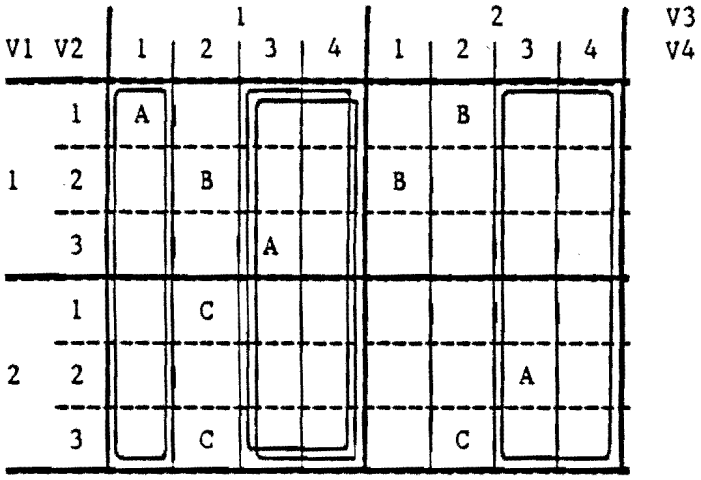
```
Type the corresponding letter to invoke the desired function: SAVEDATA

Data Set Name ? DATAX      { User saves data set that was just created. }

File Name ? JMBSDATAX


    AQINTERLISP Functions

{ list of functions omitted to save space }

Type the corresponding letter to invoke the desired function: GETDATA

File Name ? JMBSPUTX  { User loads previously created data set }
FILE CREATED 30-MAY-83 12:36:23
((VARS PUTX))   { Note that the data set name is PUTX }


    AQINTERLISP Functions

{ list of functions omitted to save space }

Type the corresponding letter to invoke the desired function: LISTDATA

Data Set Name ? PUTX

                    { User lists the data set which was just loaded }
Variables & Domains:
Variable(s) COLOR  have NOMINAL domain (RED BLUE GREEN WHITE ORANGE YELLOW)
Variable(s) LENGTH  have NOMINAL domain (12 14 16 18 20)
Variable(s) ENGINE-CID  have NOMINAL domain (260 340 440)
Variable(s) PASSENGERS  have LINEAR domain (3 .. 6)

Order of variables in events is: (COLOR LENGTH ENGINE-CID PASSENGERS)

Events & Classes:
(RED 14 260 3 => FORD)
(BLUE 14 260 3 => FORD)
(GREEN 16 260 3 => FORD)
(GREEN 16 340 3 => FORD)
(WHITE 16 340 3 => FORD)
(ORANGE 16 340 3 => FORD)
(RED 12 260 3 => CHEVY)
(BLUE 12 260 3 => CHEVY)
(WHITE 12 260 3 => CHEVY)
(ORANGE 12 260 3 => CHEVY)
(BLUE 14 340 3 => CHEVY)
(WHITE 14 340 3 => CHEVY)
(YELLOW 16 440 5 => CHEVY)
(YELLOW 18 440 5 => CHEVY)
(YELLOW 14 260 3 => DODGE)
(ORANGE 16 440 3 => DODGE)
```

```
(WHITE 18 440 5 => DODGE)
(RED 20 340 6 => DODGE)
(BLUE 20 440 6 => DODGE)


Press any key to continue. _


    AQINTERLISP Functions

A: AQVAL
    The AQVAL driver for the AQ algorithm.

E: ENTERDATA
    Interactive facility for entering and modifying data sets.

G: GETDATA
    Loads a data set from a file.

L: LISTDATA
    Pretty prints a data set.

S: SAVEDATA
    Saves a data set on a file.

Q: Quit
    Leave this menu.

Type the corresponding letter to invoke the desired function: Quit

{ User exits AQINTERLISP functions menu, control is returned to }
{ Interlisp top_level EVAL.                                      }
```

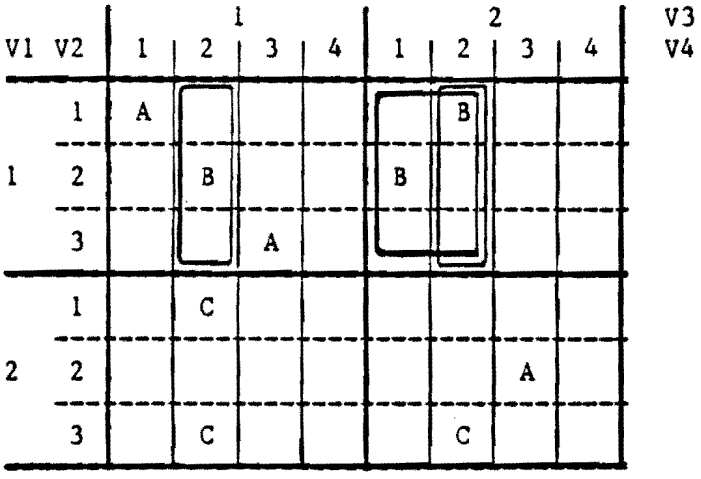
2.8. Examples of the different operating modes using GLD's

Below are displayed using "generalized logic diagrams" the covers produced by the preceeding dialog on the interactive system for data set DATAX.

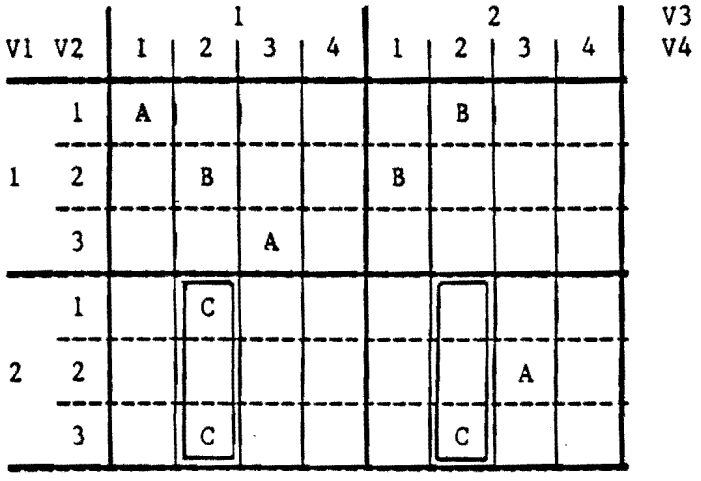
2.8.2. Disjoint covers (with intersecting complexes)



Cover of Class A
[V3 = 1][V4 = 6,1]
[V4 = 6]

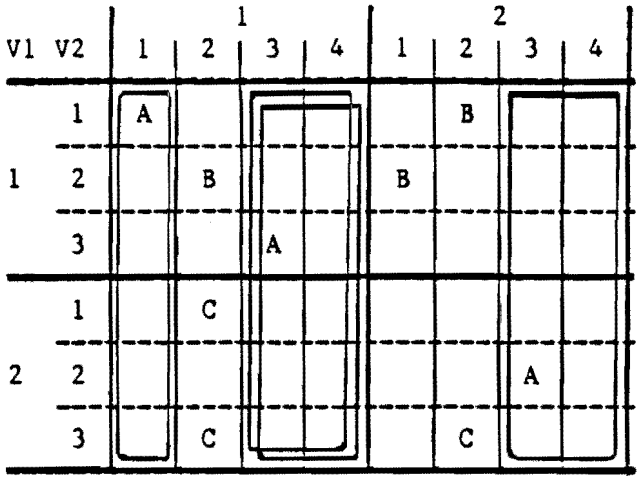


Cover of Class B
[V1 = 1][V4 = 2]
[V1 = 1][V3 = 2][V4 = 5]

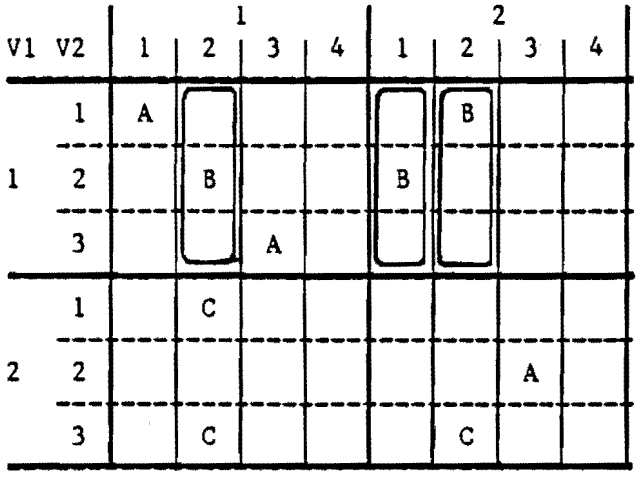


Cover of Class C
[V1 = 2][V4 = 2]

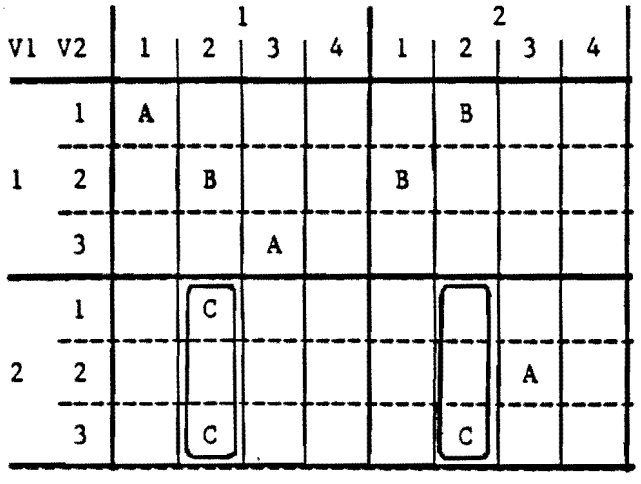
2.8.3. Disjoint covers and disjoint complexes



Cover of Class A
[V3 = 1][V4 = 6,1]
[V3 = 2][V4 = 6]

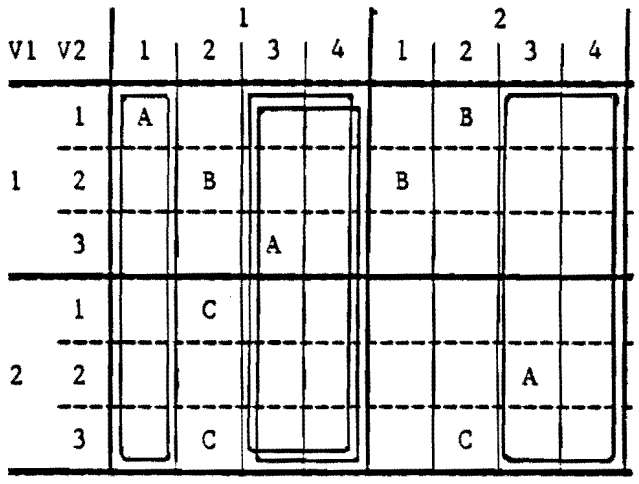


Cover of Class B
[V1 = 1][V4 = 2]
[V3 = 2][V4 = 1]

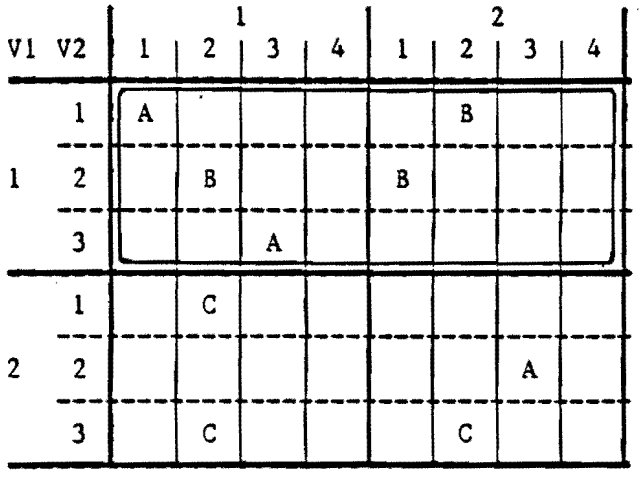


Cover of Class C
[V1 = 2][V4 = 2]

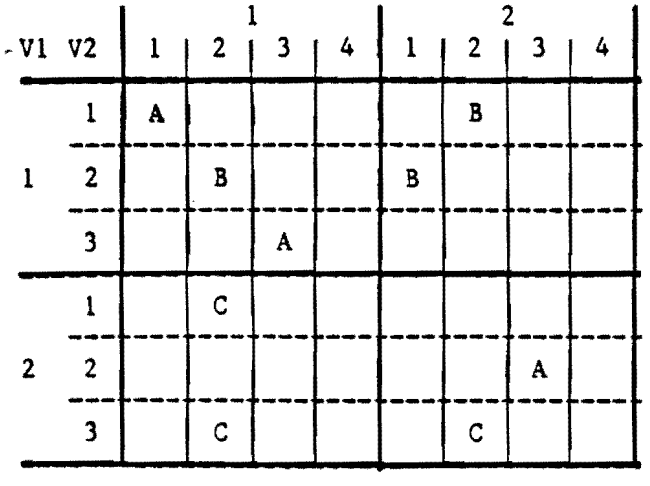
2.8.4. Sequential



Cover of Class A
[V3 = 1][V4 = 6,1]
[V4 = 6]



Cover of Class B
[V1 = 1]



Cover of Class C

(entire event space)

3. Programmer's Guide

In addition to using the AQVAL driver for the AQ algorithm, it is possible to use the AQ function within other INTERLISP programs, provided that the proper data structures are constructed prior to calling AQ. The next few sections deal with the organization of these structures, and the actual invocation of AQ. The user need not be concerned with creating these structures himself. The function (INITIALIZE DataSetName) is available for creating all of the following described structures from a data set (section 2.1). The function AQ forms the cover of complexes in PosClass against the complexes in NegClasses. AQ calls AQ-MR to do the real work. AQ-MR generates a single new complex which covers all or part of the complexes in PosClass against the complexes in NegClasses and returns the list of as yet uncovered events from PosClass. The function (REENTRANT-AQ-INIT) is available for initializing the property lists associated with the class names on list ClassNames prior to running AQ-MR.

3.1. Variable definitions

The most basic structure in AQ is the variable. These are represented by non-numeric atoms with several special properties on their property lists. All variables referenced in any events must be prepared prior to calling AQ. We will divide variables into three different types: NOMINAL, LINEAR, and STRUCTURED. For reasons of computational efficiency, bit strings are used to represent domain and event information. A bit string in AQINTERLISP is either a single Integer numeric atom or a list of Integer numeric atoms. There is no restriction on the length of a bit string (other than memory and computational time restrictions) so any number of values may be used in any domain. Internal functions for handling bit strings, most of which are compiler macros, are generally named MASKxxxxx. See Appendix B for a complete listing.

3.1.1. Nominal Variables

Nominal variables are those that can only assume distinct discrete values. Each value permitted to a nominal variable is represented by a bit position in a bit string. Bit positions are assigned by starting with the least significant bit and moving towards the most significant bit position.

The property "DOMAINTYPE" on each variable has the value "NOMINAL".

The property "VALUESET" on each variable is a list of possible values (literal atoms) that the variable may assume. The values are in the order of the assignments of values to bits.

The property "MASK" has as its value a bit string consisting of all possible bits in this variables group (all possible values). These bits correspond to the values in the VALUESET list and must be adjacent. For example, if N1 is a nominal variables with:

```
        VALUESET of N1 = (A B C)

    then
                              octal          binary

        MASK    of N1 =    7Q          0 ... 0000 0111

    where
        bit 0 (1Q)  represents value  A of N1
        bit 1 (2Q)                    B of N1
        bit 2 (4Q)                    C of N1
```

Bits are set to indicate the presence of a value. The INITIALIZE function will set up all masks, and put nominal variables on the list "AllVars".

3.1.2. Linear variables

Linear variables are those that can assume either point values or interval values. Each linear variable is a member of list "AllVars", and has six special properties set to appropriate values:

The property "DOMAINTYPE" is set to "LINEAR".

The property "FLAVOR" is set to one of "SMALL", "SYMBOLIC", or "LARGE", corresponding to a Small integer, Symbolic, or Large integer or real valued range respectively. The number of values which must be present in a domain before the "LARGE" flavor is used is controlled by the global variable "LargeLinearThreshold" which is initially set to 32.

The property "LOW" is set to the lowest value this variable may assume, whether numerical or symbolic.

The property "HIGH" is set to the highest value this variable may assume. HIGH must be greater than or equal to LOW.

The property "VALUESET" is set to a list of the values which this variable may assume. For Symbolic ranges this is just the list of values entered by the user. For Small integer ranges, this is a list of the integers from LOW to HIGH. For Large integer or real ranges, the list is derived from the set of events in the data set. The list created consists of observed values from the given events and open intervals between observed values.

The property "MASK" is set to a bit string which is the logical "OR" of the bits assigned to all values in the domain. Bit assignments are made for the VALUESET of a Linear variable in exactly the same way as for a Nominal variable.

3.1.3. Structured variables

Structured variables are those that can assume discrete values in a tree structure. Structured variables have a MASK property set up for the terminal nodes (leaves) of the domain in exactly the same way that the MASK property is set up for the VALUESET of nominal variables. Altogether, six special properties are initialized:

The property "DOMAINTYPE" is set to the value "STRUCTURED".

The property "VALUESET" is set to a list of all terminal nodes in the structure definition. The order of the nodes in this list corresponds to the order of bit assignments within a bit string. (See example below).

The property "SIBLINGS" is set to a list of elements of the form (parent siblings), with one element for each parent node in the domain. This list is sorted so that no parent node is contained in the list of siblings for a parent node on its left. (More than one correct ordering may exist.)

The property "MASK" has as its value a numeric atom consisting of all possible bits for the terminal nodes in this variables group. The method for constructing this mask is identical to the method for constructing NOMINAL variable masks.

The property "PARENTMASKS" has as its value a list of numeric atoms in order corresponding to the order of parent nodes in the "SIBLINGS" list. Each parent mask is the logical OR of the bits assigned to all descendants of the corresponding parent node.

In addition, each structured variable appears on the list "AllVars".

For example, let X be a structured variable such that:

```
domain of X      = ((1 2 3 => 7)(4 5 => 6)(6 7 => 8))
```

Then,

```
VALUESET of X    = (1 2 3 4 5)
SIBLINGS of X    = ((7 1 2 3)(6 4 5)(8 6 7))

MASK     of X    = 0 ... 0000 0001 1111

PARENTMASKS of X = (0 ... 0000 0000 0111
                    0 ... 0000 0001 1000
                    0 ... 0000 0001 1111)

Bit assignments:   0 ... 0000 0005 4321
```

where a name placed in a bit position indicates an assignment to that position, and 0 indicates no value is assigned to that bit position.

Note that these structures only define the domain of each variable, and do not assign values to the variable.

3.2. Event and Class structures

Events in AQINTERLISP are specified by the conjunction of VL1 selectors. Selectors specify the specific values variables hold at particular event points. This conjunction of selectors forms a complex. A complex in AQINTERLISP is represented by a list of selectors of the form:

((Variable . Value(s))(Variable . Value(s)) ... (Variable . Value(s)))

where "Variable" is the symbolic name of a variable, and "Value(s)" is a bit string with bit(s) set to indicate the presence of value(s). This structure is commonly referred to in Lisp programming references as an assoc-list or a-list. An arbitrary number of any type of selector may be specified for an event, limited only by free space. The list of selectors is sorted into order using the function (EARLIERVAR Sel1 Sel2) to indicate ordering. Two global lists are maintained - "ClassList", which is a list of the events in all classes, and "ClassNames", a list of all names associated with the corresponding items in "ClassList". The list of complexes associated with each class is also stored under the property "EVENTS" for each class name.

3.3. Parameter specifications

Several parameters are supplied to AQ by global lists or variables. These are:

MaxStar
: Set to the non-negative integer value that specifies the maximum size a star is allowed to have during star generation in AQ.

CutStar
: Set to the non-negative integer number less than MaxStar that specifies the size to which a star is trimmed when its size exceeds MaxStar.

Criteria&ToleranceList
: A list of doublets in the following format: (<function-name><tolerance>), where <tolerance> is between 0.0 and 1.0 . These are used to compute the cost factors used to trim complexes from a star when necessary. When called, the function is supplied with three arguments:

    1) a complex
    2) a list of complexes from all 'other' classes (those events not to be covered)
    3) all complexes in the class to be covered that have not yet been covered.

Using these arguments, the function should compute and return a numeric "cost" representing the expense of including the specified complex in the star. The numeric values of cost can be of an arbitrary scale, as long as increasing cost is represented by a more positive number returned. The <tolerance> is used to compute an allowable range of optimality. This range is computed by first applying the function to all complexes in the star, to determine the maximum and minimum cost of elements in the star. A limit is then computed as

$$lim = min + (max - min) * \langle tolerance \rangle$$

if a complex's cost is less than or equal to "lim", it is considered optimal, and is not trimmed unless only optimal intervals remain in the star, and the star must still be trimmed further.

These cost functions are applied in the order in which they appear in Criteria&ToleranceList. The first criterion is the most significant, and successive criteria are used for tie-breaking.

Two cost functions are provided in the AQINTERLISP source - #COVERED which computes how many events in the yet-to be-covered list are covered by the complex, and NUMBEROFSELECTORS which computes the cost of the number of selectors used to represent the complex.

3.4. Global variables

This is a complete list of the global variables used by AQINTERLISP:

AllOnes
: A numeric atom initialized so that all bits are set. Used for various functions involving bit strings.

AllZeros
: Like AllOnes except that no bits are set.

AllVars
: A list of all variables for the current data set. This list is used in INITIALIZE to ensure that a variable is not declared more than once.

HighBit
: A numeric atom with only the highest order bit set.

LowBit
: A numeric atom with only the lowest order bit set.

LargeLinearThreshold
: The largest number of values that a "SMALL" linear integer domain can have. When the number of values in a linear integer domain exceeds LargeLinearThreshold, the "LARGE" linear representation will automatically be used. Initially set to 32.

ExtendStrucMode
: Used to select the mode of "extension against" for structured domain variables. Takes values "NOMINAL", "MAXIMAL", and "MINIMAL". Initially set to "MAXIMAL", which implements the standard definition of extension against.

MultiFlag
: Used to control order of generating covering complexes. If T, then function AQ-MULTI is used to generate covers, otherwise function AQ-SINGLE is used to generate covers. Initially set to T.

IOTimerFlag
: Controls printing of I/O resource utilization information. If T, information is printed. Initially set to NIL.

ClassNames
: A list of the names of all classes in the current data set.

ClassList
: A list of all events in the classes of the current data set, in order corresponding to ClassNames.

MaxStar
: The maximum star size, as described in 3.3.

CutStar
: The size a star is trimmed to when MaxStar is exceeded, as described in 3.3.

Criteria&ToleranceList
: The criteria and tolerance list for evaluating complexes, as described in 3.3. Set to ((#COVERED 0)(NUMBEROFSELECTORS 0)).

VLIREADTBL
: The read table used for all AQINTERLISP input. It is a copy of the original Lisp read table with the syntax of "," and "v" reset to SEPRCHR so that these characters will be treated as blanks.

3.5. The AQ function

Once all variables are defined, and all events are constructed and organized into classes, AQ may be invoked by the following INTERLISP call:

```
(AQ PosClass NegClasses DisjointCoverFlag DisjointComplexFlag)
```

where

PosClass the name of the class to be covered by AQ

NegClasses is a list of the names of all classes that are NOT to be covered by AQ

DisjointCoverFlag is NIL if the cover may intersect with covers of classes in NegClasses, T if the covers must be non-intersecting.

DisjointComplexFlag is NIL if the cover can be constructed of intersecting compexes, or T if the cover must be constructed of disjoint complexes.

MultiFlag controls whether AQ-SINGLE or AQ-MULTI is used when generating class covers. AQ-SINGLE produces a complete cover for one class before proceeding to the next class. AQ-MULTI cycles through the classes forming one complex per class on a cycle. AQ-MULTI is slightly faster in the disjoint covers modes. Initially set to T, AQ-MULTI is used.

IOTimerFlag controls printing of I/O resource utilization information. Initially NIL, no I/O resource info. is printed.

AQ returns modified values on the property lists associated with the class names:

COVER - Cover of the class

COVEREDEVENTS - A list of the events covered by complexes in COVER in corresponding order.

UNCOVEREDEVENTS - A list of events in the class which are not covered by any complex in COVER.

SEEDLIST - A list of seed events for forming a star.

OLDEVENT - Used to trap infinite loops, contains the previous event used as a seed.

The union of events covered by all complexes is the COVER of the specified class PosClass. The complexes may be decoded by reversing the process used to construct complexes - see the function "PPCOMP" in Appendix B for an example of this.

3.6. Other functions available

Many internal functions may prove useful to the user utilizing AQ as a callable function. Appendix B lists the entire AQINTERLISP system, with brief comments about each function.

4. Comparison with another implementation of AQ

AQINTERLISP is one of a number of implementations of AQ. Another readily accessible implementation is AQ11p, a PASCAL based system which is available on the CSO Cyber 175 and the CRL uiuccsb (Vax B). There are a number of differences between the two systems. These differences are outlined below.

AQ11p supports multi-step synthesis of VL1 hypotheses, the current implementation of AQINTERLISP does not. AQ11p also has "restriction" capabilities, and facilities for generating confusion matrices. AQINTERLISP does not have these features. The reader is referred to [10] for further discussion of these topics.

The internal data structures of the programs are quite different. In AQ11p, all variable types are implemented using Pascal sets. AQINTERLISP uses variable sized bit strings for all variable types. Upper bounds are placed on the problem size in AQ11p by compiler constants which limit domain set size to 59 levels. In AQINTERLISP, domain sizes are limited only by available space and CPU time. Internal representations are automatically adjusted to fit the problem size. AQINTERLISP also supports Real valued linear ranges.

The user interfaces of the two systems are also quite different. AQ11p operates only in batch processing mode. To get symbolic output, two input files must be used - one containing parameters and data, and the other containing the mappings of the classes and variables from integers to English names. AQINTERLISP operates in an interactive environment. Special facilities are provided for prompting for information and handling of data sets. Data may be entered directly using appropriate symbolic names, no translation table is needed.

Some test runs were done to get a good idea of the relative performance of AQ11p vs AQINTERLISP, and AQINTERLISP on the Vax vs AQINTERLISP on the Dolphin. To compare AQINTERLISP on the Dolphin vs the Vax B (a Vax 780), the data set used in the sample dialog, "DATAX" was run on both machines in both compiled and interpreted versions in all four modes of operation. On both machines, the compiled version runs about 21 times faster than the interpreted version in terms of CPU time, and 15 times faster in terms of real time. On the average, the Vax is about 1.9 times faster than the Dolphin in terms of CPU time, and about 1.5 times faster than the Dolphin in terms of real time (with 5 users on the Vax and an average of 1.12 jobs in the run queue). The reader should note that often the load on Vax B is much greater, so it is not unusual for the Dolphin to return results faster than Interlisp on the Vax in terms of real time.

To compare AQ11p and AQINTERLISP, a slightly modified version of "DATAX" was run in both programs - the domain of "V4" was changed to NOMINAL with valueset (1 2 3 4). For AQ11p all ranges were normalized to start at 0 by subtracting one from all values in the data (as required by the program). AQ11p was also set to generate covers in 2 passes with MAXSTAR set to 5. AQINTERLISP was run with MAXSTAR=10 and CUTSTAR=5. Only the com-

piled version of AQINTERLISP was run. The programs were run in both Intersecting and Disjoint Cover modes. On the average, AQ11p was 4 times faster than AQINTERLISP in terms of CPU time. However, the turn-around times were nearly the same since AQ11p must do I/O to read the data from a file and write the results to a file, whereas the data for AQINTERLISP was already in memory and converted to internal format.

Other factors should also be considered. The power and flexibility of Interlisp does not come without some inconvenience. AQINTERLISP, whether interpreted or compiled, must be run from "inside" Interlisp. Thus, it is necessary to load Interlisp before running the program. This can take several minutes. Once Interlisp is loaded, the AQINTERLISP file must be loaded into Interlisp. This can also take several minutes. Because the Interlisp system is very large it operates in a virtual memory environment. A lot of computing resources are tied up and paging behavior becomes noticeable. On the other hand, compiled AQ11p is a machine code load module which runs independently of any other systems. Once a data file has been created the program can be run immediately with simple operating system commands. In certain situations this can be a more convenient way to interact with a program.

Overall, AQINTERLISP has more flexible naming conventions, can handle larger data sets, and is more "user friendly", but lacks some of the features and is not as fast as AQ11p.

5. Conclusions

The operating characteristics of AQINTERLISP have been described and demonstrated. An interactive interface to the AQ function has been documented, and a sample execution shown. Finally, the calling requirements of AQ are illustrated, and the primary data structures diagrammed. A complete listing of AQ appears in Appendix B.

References

(1) -- , Interlisp Reference Manual, Warren Teitelman, ed., Xerox Palo Alto Research Center, California, 1978.

(2) —,Interlisp-D Users Guide, Xerox Electro-Optical Systems, Pasadena, California, February 1982.

(3) Bobrow, D.G., and R. S. Michalski, Source listings for AQLISP on Interlisp-10, Installed on Bolt, Beranek, and Newman system BBND, 1978.

(4) Greenwalt, E. M., Johnathan Slocum, and Robert A. Amsler, UT LISP Documentation, Version 4.0, University of Texas at Austin Computation Center, 1975.

(5) Larson, James, and R. S. Michalski, "AQVAL/1 (AQ7) Users Guide and Program Description", Report number 731, Department of Computer Science, University of Illinois, Urbana, June 1975.

(6) Michalski, R. S., "A Geometric Model for the Synthesis of Interval Covers", Report number 461, Department of Computer Science, University of Illinois, Urbana, June, 1971.

(7) Michalski, R. S., "Variable-Valued Logic: System VL1", 1974 International Symposium on Multiple-Valued Logic, West Virginia University, Morgantown, West Virginia, May 1974.

(8) Michalski, R. S., and McCormick, B. H., "Interval Generalization of Switching Theory", Report number 442, Department of Computer Science, University of Illinois, Urbana, May 1971.

(9) Michalski, R. S., and Larson, J. B., "Selection of Most Representative Training Examples and Incremental Generation of VL1 Hypotheses: the underlying methodology and description of programs ESEL and AQ11", Report Number 867, Department of Computer Science, University of Illinois, Urbana, May 1978.

(10) Michalski, R. S., and Larson, J. B., Revised by K. Chen, "Incremental Generation of VL1 Hypothesis: the underlying methodology and description of program AQ11", File # UIUCDCS-F-83-905, ISG 83-5, Department of Computer Science, University of Illinois, Urbana, Jan. 1983.

(11) Richards, Paul, "AQLISP: A LISP Program for Inductive Generalization of VL1 Event Sets", Internal Report 12-15-79, Department of Computer Science, University of Illinois, Urbana, December 1979.

Using AQINTERLISP on the Xerox DOLPHIN

See the "dolphin" notes file on uiucdcs (Vax A) for current discussion about the Dolphin and new users information. Power up the Dolphin and start INTERLISP according to instruction in the INTERLISP-D users guide (see [2]). The following instructions assume that the appropriate files have been stored on the Dolphin disk partition 1 or can be accessed using connected directories.

It is much more convenient to use the display editor "DEDIT" and the line editor "TTYIN" for editing data sets than the standard Lisp editor. These facilities are not present in the standard Interlisp environment. They may be loaded by typing LOAD(DEDIT.DCOM) and LOAD(TTYIN.DCOM). The standard editing function calls will be redefined in terms of these new editors. Instructions for use of DEDIT and TTYIN can be found in the INTERLISP-D users guide.

The compiled version of AQINTERLISP is roughly 20 times faster than the interpreted version, and will probably be favored by most users for most purposes. It can be loaded by typing LOAD(JMB$AQLISP.DCOM) The interpreted version may be loaded in a similar manner. It is stored under the name JMB$AQLISP.

Some special control characters that the user should be aware of are:

^C Stops Lisp and enters the RAID debugger (see ref. [2]). Type ^N to get back to Lisp.

^E Stops the currently running Interlisp program, if any, and returns to the top-level eval function.

^H DO NOT USE. Use the [BS] key to backspace.

Up-arrow
Use this key to exit unintentional breaks and return to the calling level. See [1] for further information about the Interlisp Break Package.

Using AQINTERLISP on the CRL Vax 8

Preparing to run Interlisp-VAX involves setting up a special subdirectory and adding commands to your ".login" file. A subdirectory should be created as an immediate descendant of your home directory. Copy /mntb/2/michalski/jbecker/interlisp/INIT.LISP to this subdirectory. INIT.LISP is an initialization file which contains commands which are executed at the beginning of every Interlisp-VAX session. The above file will rebind ^H from the Help interrupt to the character delete function and ^J to the help interrupt function. Also, ^C is disabled and "LOGOUT" is advised to prompt the user to save variable and function definitions. The following lines should be added to your ".login" file:

```
alias ilisp  /usr/interlisp/unixbin/ilisp
alias ls     /usr/interlisp/unixbin/ls
setenv       VERSIONS 4
setenv       SysLispInitFile "$HOME/interlisp/INIT.LISP
setenv       SHELL /usr/interlisp/unixbin/csh
```

The new shell and ls are the same as the standard ones, but in addition know about Interlisp-VAX file naming conventions. To run Interlisp type "ilisp". Documentation of interest to Interlisp-VAX users may be found in the /usr/interlisp directory. NOTE: the Interlisp-VAX system is still under development. Certain problems exist with file naming conventions, and certain Interlisp facilities are not implemented in the current release of Interlisp-VAX. The user should review the available documentation.

No special display editors are available under Interlisp-VAX. The Interlisp structure editor may still be used for editing data sets. AQINTERLISP is stored under:

```
/mntb/2/michalski/isg/AQLISP/AQLISPV      (interpreted), and
/mntb/2/michalski/isg/AQLISP/AQLISPV.v    (compiled).
```

Load the desired file by typing LOAD(filename) once Interlisp has been started.

Control characters in Interlisp-VAX must be followed by a <RETURN> before they will take effect. ^E may be used to stop a running program when it is requesting input. It may not be possible to stop a running program which is not requesting input - this bug is being worked on.

Appendix B

AQINTERLISP Function Listings

Appendix B

AQINTERLISP Funtion Listings

```
FUNCTION: RUN
CALLS: AQVAL ENTERDATA GETDATA LISTDATA READNAME SAVEDATA WAIT-FOR-KEYPRESS
CALLED BY: <none>

(RUN
  [LAMBDA NIL

          (* Top level menu for AQINTERLISP)


    (PROG (Choice DSName FLName)
      LOOP(printout T T T "    AQINTERLISP Functions" T T)
          (printout T "A: AQVAL" T "    The AQVAL driver for the AQ algorithm."
                    T T)
          (printout T "E: ENTERDATA" T
           "    Interactive facility for entering and modifying data sets."
                    T T)
          (printout T "G: GETDATA" T "    Loads a data set from a file." T T)
          (printout T "L: LISTDATA" T "    Pretty prints a data set." T T)
          (printout T "S: SAVEDATA" T "    Saves a data set on a file." T T)
          (printout T "Q: Quit " T "    Leave this menu." T T)
          (printout T
           "Type the corresponding letter to invoke the desired function: ")
      CHOOSE
          (CONTROL T)
          (SETQ Choice (U-CASE (READC)))
          (CONTROL NIL)
          (SELECTQ Choice
                   (A (printout T "QVAL" T)
                      (SETQ DSName (READNAME (QUOTE DataSet)))
                      (AQVAL (EVAL DSName)))
                   (E (printout T "NTERDATA" T)
                      (SETQ DSName (READNAME (QUOTE DataSet)))
                      (TERPRI)
                      (ENTERDATA DSName))
                   (G (printout T "ETDATA" T)
                      (SETQ FLName (READNAME (QUOTE File)))
                      (GETDATA FLName))
                   (L (printout T "ISTDATA" T)
                      (SETQ DSName (READNAME (QUOTE DataSet)))
                      (LISTDATA (EVAL DSName))
                      (WAIT-FOR-KEYPRESS))
                   (S (printout T "AVEDATA" T)
                      (SETQ DSName (READNAME (QUOTE DataSet)))
                      (SETQ FLName (READNAME (QUOTE File)))
                      (SAVEDATA DSName FLName))
                   (Q (printout T "uit" T)
                      (GO OUT))
```

```
                    (PROGN (printout T T "Type one of the above letters: ")
                           (GO CHOOSE)))
            (CLEARBUF)
            (GO LOOP)
          OUT (RETURN])
```

```
FUNCTION: AQVAL
CALLS: READMAXSTAR&CUTSTAR READSX REENTRANT-AQ-INIT SHOWCOVERS
CALLED BY: RUN

(AQVAL
  [LAMBDA (DataSet)

          (* Interactive driver for the AQ function)


    (printout T T "Welcome to AQINTERLISP <Revision 15-July-1983>" T)
    (PROG (Mode)
          (READMAXSTAR&CUTSTAR)
          (printout T "Initialization:" T)
          (TIME (INITIALIZE DataSet)
                1 0)
          (SETQ ClassList (MAPCAR ClassNames (FUNCTION EVENTS)))
      LP  (printout T T
                    "Select mode of operation by typing 1, 2, 3, or 4:     "
                     T "   1: Intersecting Covers" T
             "   2: Disjoint Covers (with possibly intersecting complexes)"
                     T "   3: Disjoint Covers and Disjoint Complexes" T
                     "   4: Sequential (VL)")
      ENTER
          (printout T T T "  #")
          (SETQ Mode (READSX))
      SHORTCUT
          (TERPRI)
          (REENTRANT-AQ-INIT ClassNames)
          (SELECTQ Mode
                   (1 (TIME (INTERSECTINGCOVERS ClassNames)
                            1 0))
                   (2 (TIME (DISJOINTCOVERS ClassNames)
                            1 0))
                   (3 (TIME (DISJOINTCOMPLEXCOVER ClassNames)
                            1 0))
                   (4 (TIME (ORDEREDCOVER ClassNames)
                            1 0))
                   (PROGN (printout T
                                    "Type a number (1 .. 4) to make a selection."
                                  T)
```

```
                    (GO ENTER)))
        (COND
          (IOTimerFlag (TIME (PROGN (SHOWCOVERS ClassNames)
                                    (printout T "I/O Statistics:" T))
                             1 0))
          (T (SHOWCOVERS ClassNames)))
        (printout T T "Do you want to try another mode? ")
      AGAIN
        [SETQ Mode (GNC (U-CASE (READSX]
        (AND (NUMBERP Mode)
             (IGREATERP Mode 0)
             (ILESSP Mode 5)
             (GO SHORTCUT))
        (SELECTQ Mode
                 (Y (GO LP))
                 (N (RETURN (RECLAIM)))
                 (PROGN (printout T "Type YES, NO, or a number (1 .. 4)" T)
                        (GO AGAIN])
```

FUNCTION: ENTERDATA
CALLS: AQHELP ENTEREVENTS&CLASSES ENTERORDER ENTERVARIABLES&DOMAINS LISTDATA
CALLED BY: RUN

```
(ENTERDATA
  [NLAMBDA (DataSet)

          (* ENTERDATA command level function, allows interactive
          entry of a data set)


    (PROG (Char AQData)
          [COND
            ((EQ DataSet (QUOTE DSName))
              (SETQ DataSet (EVAL DataSet]
          [COND
            ((NOT (BOUNDP DataSet))
              (SET DataSet (LIST NIL (QUOTE ALPHA)
                                 NIL))
              (SETQ AQData (EVAL DataSet))
              (PRIN1 "O")
              (ENTERORDER AQData)
              (PRIN1 "V")
              (ENTERVARIABLES&DOMAINS AQData)
              (PRIN1 "E")
              (ENTEREVENTS&CLASSES AQData))
            (T (SETQ AQData (EVAL DataSet]
          (AQHELP (QUOTE ENTER))
```

```
        (TERPRI)
    LP  (CONTROL T)
        (CLEARBUF)
        (PRIN1 ">> ")
        (SETQ Char (READC))
        (CONTROL NIL)
        (SELECTQ Char
                 (V (ENTERVARIABLES&DOMAINS AQData))
                 (O (ENTERORDER AQData))
                 (E (ENTEREVENTS&CLASSES AQData))
                 (L (PRIN1 "ist ")
                    (LISTDATA AQData))
                 (C (PRIN1 "hange Data Set ")
                    (TERPRI)
                    (EDITV AQData)
                    (PRINT DataSet)
                    (TERPRI))
                 (%. (GO DONE))
                 (AQHELP (QUOTE ENTER)))
        (GO LP)
    DONE(CONTROL NIL)
        (RETURN DataSet])
```

```
FUNCTION: GETDATA
CALLS: <none>
CALLED BY: RUN

(GETDATA
  [LAMBDA (FileName)

          (* Load a data set from a disk file)


    (LOAD FileName])
```

```
FUNCTION: LISTDATA
CALLS: PRINLIST
CALLED BY: ENTERDATA RUN

(LISTDATA
  [LAMBDA (DataSet)
```

```
          (* Pretty-print a data set)


    (TERPRI)
    (TERPRI)
    (PRIN1 "Variables & Domains: ")
    (TERPRI)
    [MAPC (CAR DataSet)
          (FUNCTION (LAMBDA (V&D)
              (PRIN1 "Variable(s) ")
              (PRINLIST (CAR V&D))
              (PRIN1 " have ")
              (PRIN1 (SELECTQ (CADR V&D)
                              (N (QUOTE NOMINAL))
                              (L (QUOTE LINEAR))
                              (S (QUOTE STRUCTURED))
                              (QUOTE ?)))
              (PRIN1 " domain ")
              (PRINT (CADDR V&D]
    (TERPRI)
    (PRIN1 "Order of variables in events is: ")
    (PRINT (CADR DataSet))
    (TERPRI)
    (PRIN1 "Events & Classes: ")
    (TERPRI)
    [MAPC (CADDR DataSet)
          (FUNCTION (LAMBDA (E&C)
              (PRINT E&C]
    (TERPRI])
```

```
FUNCTION: SAVEDATA
CALLS: <none>
CALLED BY: RUN

(SAVEDATA
  [LAMBDA (DataSetName FileName)

          (* Save a data set on a disk file)


    (COND
      ((NOT (BOUNDP DataSetName))
        (PRINT "Data set not found."))
      (T (PRETTYDEF NIL FileName (LIST (LIST (QUOTE VARS)
                                             DataSetName])
```

```
FUNCTION: EDITDATA
CALLS: <none>
CALLED BY: SETUPVBLS&DOMAINS

(EDITDATA
  [LAMBDA (DataSet)

          (* Invoke Lisp Editor)


    (EDITV DataSet])
```

```
FUNCTION: ENTERORDER
CALLS: READLIST
CALLED BY: ENTERDATA

(ENTERORDER
  [LAMBDA (DataSet)

          (* ENTERDATA function for inputing order of variables in
          events)


    (printout T "rder List:" T
              "List all variables for the current data set in the"
              T "desired order, separated by spaces or commas." T)
    (COND
      ((NOT (NULL (CADDR DataSet)))
        (printout T
              "Warning! Events have already been entered using the order: "
                  (CADR DataSet)
                  T)))
    (SETQ Ord (READLIST))
    (RPLACA (CDR DataSet)
            Ord)
    (CLEARBUF)
    (TERPRI)
    DataSet])
```

```
FUNCTION: ENTERVARIABLES&DOMAINS
CALLS: AQHELP READLIST READSX
CALLED BY: ENTERDATA
```

```
(ENTERVARIABLES&DOMAINS
  [LAMBDA (DataSet)

          (* ENTERDATA function for inputing variables and domain
          specifications)


    (PROG (Vars DomainType Domain)
      LOOP(printout T "ariable declarations: " T
    "List variables having the same domain, separated by spaces or commas."
                  T
              "Or, type '.' if there are no more variables to be declared."
                  T)
          (SETQ Vars (READLIST))
          (TERPRI)
          (AND (EQUAL (CAR Vars)
                      (QUOTE %.))
               (GO DONE))
      CHOOSE
          (CLEARBUF)
          (PRIN1 "Domain Type? (N, L, S): ")
          (CONTROL T)
          (SETQ DomainType (READC))
          (CONTROL NIL)
          (SELECTQ DomainType
                   (N (printout T "OMINAL" T
              "List the permissible values, separated by spaces or commas."
                              T)
                      (SETQ Domain (READLIST)))
                   [L (printout T "INEAR" T "What is the range of values?" T
          "Type as %"Low '..' High%" for Integer or Real valued variables,"
                              T
            "  or as %"Val1 Val2 ... Valn%" for Symbolic valued variables."
                              T)
                      (SETQ Domain (READLIST))
                      (COND
                        ((NEQ (CADR Domain)
                              (QUOTE ..))
                          NIL)
                        ((AND (FIXP (CAR Domain))
                              (FIXP (CADDR Domain)))
                          (NCONC Domain (LIST 1)))
                        (T (NCONC Domain (PROGN (printout T T
                                                  "What is the Epsilon Value? ")
                                                  (LIST (READSX]
                   (S (printout T "TRUCTURED" T "What is the structure?" T
                "Type as '((siblings => parent) ... (siblings => parent))'."
                              T)
                      (SETQ Domain (READSX)))
                   (PROGN (AQHELP (QUOTE DOMAINTYPE))
                          (GO CHOOSE)))
          [RPLACA DataSet (NCONC (CAR DataSet)
```

```
                                    (LIST (LIST Vars DomainType Domain]
          (TERPRI)
          (PRIN1 "V")
          (GO LOOP)
      DONE(RETURN DataSet])
```

```
FUNCTION: ENTEREVENTS&CLASSES
CALLS: ENTEREVENT READLIST READSX
CALLED BY: ENTERDATA

(ENTEREVENTS&CLASSES
  [LAMBDA (DataSet)

          (* ENTERDATA function for inputing event and class
          information)


    (PROG (NewClasses Class)
          (SETQ ClassNames NIL)
          [MAPC (CADDR DataSet)
                (FUNCTION (LAMBDA (E&C)
                    (SETQ ClassNames (UNION ClassNames (LAST E&C]
          (printout T "vents and Classes:" T "List ")
          (COND
            ((NULL (CADDR DataSet))
              (PRIN1 "all"))
            (T (PRIN1 "additional")))
          (printout T " class names, separated by commas or spaces." T)
          (AND (CADDR DataSet)
               (printout T
                "Or, type '.' if no additional classnames are to be added."
                        T))
          (SETQ NewClasses (READLIST))
          (AND (NOT (EQUAL (CAR NewClasses)
                           (QUOTE %.)))
               (SETQ ClassNames (UNION ClassNames NewClasses)))
          (SORT ClassNames)
      ALLORONE
          (printout T T "For each event in a class, enter a list of values" T
                    "for variables  "
                    (CADR DataSet)
                    T "in the order shown, separated by spaces or commas." T
                  "Or, enter a list of selectors as ((sel)(sel) ... (sel))," T
                  "where each '(sel)' has the form (Variable Relation Vals)."

                 "Type '.' instead of an event to proceed to the next class."
```

```
                T)
        (printout T T "The list of class names is: " ClassNames "." T T)
        (printout T
              "Type a class name to enter events for a particular class,"
                  T "or 'ALL' to enter events for all classes." T)
        (CLEARBUF)
        (SETQ Class (READSX))
        (COND
          ((MEMBER Class ClassNames)
            (GO ONECLASS))
          ((EQUAL Class (QUOTE ALL))
            (GO ALLCLASSES))
          ((EQUAL Class (QUOTE %.))
            (GO OUT))
          (T (GO ALLORONE)))
    ONEMORE
        (printout T T "Type a class name to enter events for another class."
                  T "Type '.' if there are no more events to be added." T)
        (CLEARBUF)
        (SETQ Class (READSX))
        (COND
          ((EQUAL Class (QUOTE %.))
            (GO OUT))
          ((NOT (MEMBER Class ClassNames))
            (CLEARBUF)
            (GO ONEMORE)))
    ONECLASS
        (COND
          ((EQUAL (ENTEREVENT DataSet Class)
                  (QUOTE %.))
            (GO ONEMORE))
          (T (GO ONECLASS)))
        (GO ONEMORE)
    ALLCLASSES
        [MAPC ClassNames (FUNCTION (LAMBDA (Class)
                  (printout T T "Enter events for class " Class "." T
                            "Type '.' to proceed to the next class."
                            T)
                  (PROG NIL
                    LOOP(SELECTQ (ENTEREVENT DataSet Class)
                                 (%. (GO OUT))
                                 (GO LOOP))
                    OUT]
    OUT (TERPRI)
        (RETURN DataSet])
```

FUNCTION: ENTEREVENT

```
CALLS: READLIST
CALLED BY: ENTEREVENTS&CLASSES

(ENTEREVENT
  [LAMBDA (DataSet Class)

          (* Input an Event)


    (PROG (Event)
          (printout T "Class " Class " event: ")
          (SETQ Event (READLIST))
          [COND
            ((LISTP (CAR Event))
              (SETQ Event (CAR Event]
          (COND
            ((EQUAL (CAR Event)
                    (QUOTE %.))
              (RETURN (QUOTE %.)))
            (T [RPLACA (CDDR DataSet)
                       (NCONC (CADDR DataSet)
                              (LIST (APPEND Event (QUOTE (=>))
                                            (LIST Class]
               [SORT (CADDR DataSet)
                     (FUNCTION (LAMBDA (X Y)
                         (ALPHORDER (CAR (LAST X))
                                    (CAR (LAST Y]
               (RETURN T])



FUNCTION: AQHELP
CALLS: <none>
CALLED BY: ENTERDATA ENTERVARIABLES&DOMAINS

(AQHELP
  [LAMBDA (HelpIn)

          (* Help facilities text for AQINTERLISP)


    (TERPRI)
    (TERPRI)
    (SELECTQ HelpIn
             (ENTER (printout T "Type V to enter Variables and Domains" T
                              "Type O to enter Order of variables in events"
                               T
                             "Type E to enter an Event and Class information"
                               T "Type L to list the data set" T
```

```
                "Type C to make changes using the LISP structure editor"
                          T "Type . to exit ENTERDATA" T
                          "Type any other key to see this list"
                          T))
          (DOMAINTYPE (printout T "Type N to indicate a NOMINAL domain" T
                                  "Type L to indicate a LINEAR domain"
                                  T "Type S to indicate a STRUCTURED domain" T)
                      )
          (PROGN (printout T T T "    AQINTERLISP Functions" T T)
                 (printout T "ENTERDATA" T
        "    Interactive facility for entering and modifying data sets."
                          T "    Type (ENTERDATA DataSetName))." T T)
                 (printout T "LISTDATA" T "    Pretty print a data set." T
                          "    Type (LISTDATA DataSetName)."
                          T T)
                 (printout T "GETDATA" T "    Load a data set from a file."
                          T "    Type (GETDATA 'DataSetName)." T T)
                 (printout T "SAVEDATA" T "    Save a data set on a file." T
                          "    Type (SAVEDATA 'DataSetName 'FileName)."
                          T T)
                 (printout T "AQVAL" T
                     "    Invoke the AQVAL driver for the AQ algorithm."
                          T "    Type (AQVAL DataSetName)." T T)
                 (printout T T "<note where single quotes are required>")))
    (TERPRI)
    (TERPRI])
```

```
FUNCTION: READMAXSTAR&CUTSTAR
CALLS: READSX
CALLED BY: AQVAL

(READMAXSTAR&CUTSTAR
  [LAMBDA NIL

          (* Input MaxStar and CutStar parameters)


    (TERPRI)
    (PRIN1 "Enter maximum Star size for this run: ")
    (SETQ MaxStar (READSX))
    (TERPRI)
    (PRIN1 "Enter size to cut Star to when truncating: ")
    (SETQ CutStar (READSX))
    (TERPRI])
```

```
FUNCTION: READLIST
CALLS: READSX
CALLED BY: ENTEREVENT ENTEREVENTS&CLASSES ENTERORDER ENTERVARIABLES&DOMAINS

(READLIST
  [LAMBDA NIL

          (* Reads a list of s-expressions from the terminal)


    (CONS (READSX)
          (READLINE VL1READTBL])




FUNCTION: READNAME
CALLS: <none>
CALLED BY: RUN

(READNAME
  [LAMBDA (Type)

          (* Reads a File or DataSet name)


    (TERPRI)
    (SELECTQ Type
             (File (printout T "File Name ? ")
                   (READ))
             (DataSet (printout T "Data Set Name ? ")
                      (READ))
             (SHOULDNT])



FUNCTION: READSX
CALLS: <none>
CALLED BY: AQVAL ENTEREVENTS&CLASSES ENTERVARIABLES&DOMAINS READLIST
READMAXSTAR&CUTSTAR

(READSX
  [LAMBDA NIL

          (* Reads a single s-expression from the terminal)
```

```
    (READ NIL VL1READTBL])



FUNCTION: WAIT-FOR-KEYPRESS
CALLS: <none>
CALLED BY: RUN

(WAIT-FOR-KEYPRESS
  [LAMBDA NIL

          (* Stop everything until a key is pressed)


    (printout T T "Press any key to continue.  ")
    (CLEARBUF)
    (CONTROL T)
    (READC)
    (CONTROL NIL)
    (TERPRI])



FUNCTION: INITIALIZE
CALLS: SETUPEVENTS&CLASSES SETUPVBLS&DOMAINS
CALLED BY: <none>

(INITIALIZE
  [LAMBDA (DataSet)

          (* Transform a data set into the internal structures
          required by the AQ functions)


    (SETQ AllVars (CADR DataSet))
    (SETUPVBLS&DOMAINS DataSet)
    (SETUPEVENTS&CLASSES DataSet])



FUNCTION: REENTRANT-AQ-INIT
CALLS: <none>
CALLED BY: AQVAL
```

```
(REENTRANT-AQ-INIT
  [LAMBDA (Classes)

          (* Initialize property lists for AQ-MR)


    (MAPC Classes (FUNCTION (LAMBDA (Class)
              (SETCOVER Class NIL)
              (SETSEEDLIST Class (EVENTS Class))
              (SETUNCOVEREDEVENTS Class (EVENTS Class))
              (SETOLDEVENT Class NIL)
              (SETCOVEREDEVENTS Class NIL])
```

```
FUNCTION: RESETCLASSES
CALLS: <none>
CALLED BY: SETUPEVENTS&CLASSES
```

```
(RESETCLASSES
  [LAMBDA NIL

          (* Reset the list of events associated with the classes in
          "ClassNames" to NIL)


    (MAPC ClassNames (FUNCTION (LAMBDA (Class)
              (SETEVENTS Class NIL])
```

```
FUNCTION: SETUPVBLS&DOMAINS
CALLS: EDITDATA SETUPLINEAR SETUPNOMINAL SETUPSTRUCTURED
CALLED BY: INITIALIZE

(SETUPVBLS&DOMAINS
  [LAMBDA (DataSet)

          (* Transform the list of variables and domains from a data
          set into the corresponding structures used by the AQ
          function)


    (PROG (V&D TempList DeclaredVars)
      START
          (SETQ TempList (CAR DataSet))
```

```
      LOOP(AND (NULL TempList)
               (GO OUT))
           (SETQ V&D (CAR TempList))
           [COND
             ([SETQ Vbl (CAR (SOME (CAR V&D)
                                   (FUNCTION (LAMBDA (V)
                                       (MEMBER V DeclaredVars]
              (printout T "The variable " Vbl " has been declared twice." T
                        "Calling editor on the data set."
                        T)
              (EDITDATA DataSet)
              (GO START))
             (T (SETQ DeclaredVars (APPEND DeclaredVars (CAR V&D]
           (SELECTQ (CADR V&D)
                    (N (SETUPNOMINAL (CAR V&D)
                                     (CADDR V&D)))
                    (L (SETUPLINEAR (CAR V&D)
                                    (CADDR V&D)
                                    (CADDR DataSet)))
                    (S (SETUPSTRUCTURED (CAR V&D)
                                        (CADDR V&D)))
                    (PROGN (printout T "Unknown domain type in " V&D "." T
                                     "Calling editor on the data set."
                                     T)
                           (EDITDATA DataSet)
                           (GO START)))
           (SETQ TempList (CDR TempList))
           (GO LOOP)
      OUT (RETURN])
```

```
FUNCTION: SETUPNOMINAL
CALLS: SETUPMASK
CALLED BY: SETUPVBLS&DOMAINS

(SETUPNOMINAL
  [LAMBDA (Vars Domain)

          (* Set up structures for a set of nominal variables with the
          same domain)


     (PROG [(Mask (SETUPMASK 1 (FLENGTH Domain]
           (MAPC Vars (FUNCTION (LAMBDA (Var)
                      (PUTPROP Var (QUOTE DOMAINTYPE)
                               (QUOTE NOMINAL))
                      (PUTPROP Var (QUOTE VALUESET)
                               Domain)
```

```
                    (PUTPROP Var (QUOTE MASK)
                             Mask])



FUNCTION: SETUPLINEAR
CALLS: SETUPLARGELINEARVALS SETUPMASK
CALLED BY: SETUPVBLS&DOMAINS

(SETUPLINEAR
  [LAMBDA (Vars Domain E&CList)

          (* Set up internal structures for a set of Linear variables
          with the same domain)


    (PROG (Flavor Low High ValueSet Mask Epsilon)
          (SETQ Low (CAR Domain))
          (SETQ High (CADDR Domain))
          (SETQ Epsilon (CAR (LAST Domain)))
          [COND
            ((NOT (EQ (CADR Domain)
                      (QUOTE ..)))
              (SETQ Flavor (QUOTE SYMBOLIC))
              (SETQ High (CAR (LAST Domain)))
              (SETQ ValueSet Domain))
            ((AND (FIXP Low)
                  (FIXP High)
                  (ILESSP (IDIFFERENCE High Low)
                          LargeLinearThreshold))
              (SETQ Flavor (QUOTE SMALL))
              (SETQ ValueSet (for I from Low to High collect I)))
            (T (SETQ Flavor (QUOTE LARGE))
               (SETQ ValueSet (SETUPLARGELINEARVALS Vars E&CList Low High
                                                    Epsilon]
          (SETQ Mask (SETUPMASK 1 (FLENGTH ValueSet)))
          (MAPC Vars (FUNCTION (LAMBDA (Var)
                    (PUTPROP Var (QUOTE DOMAINTYPE)
                             (QUOTE LINEAR))
                    (PUTPROP Var (QUOTE FLAVOR)
                             Flavor)
                    (PUTPROP Var (QUOTE LOW)
                             Low)
                    (PUTPROP Var (QUOTE HIGH)
                             High)
                    (PUTPROP Var (QUOTE VALUESET)
                             ValueSet)
                    (PUTPROP Var (QUOTE MASK)
                             Mask])
```

```
FUNCTION: SETUPLARGELINEARVALS
CALLS: PRINLIST
CALLED BY: SETUPLINEAR

(SETUPLARGELINEARVALS
  [LAMBDA (Vars E&CList Low High Epsilon)

          (* Set up the ValueSet for a large linear domain)


    (PROG ((ValueSet (LIST Low High))
           Event Interval Vals)
          [for E&C in E&CList
             do (SETQ Event (CAR (LASTN E&C 2)))
                (SETQ ValueSet
                  (UNION ValueSet
                         (COND
                           ((ATOM (CAR Event))
                             (for Val in Event as Var in AllVars
                                collect Val when (MEMBER Var Vars)))
                           (T (APPENDX (for Sel in Event
                                          collect (for Val in (CDDR Sel)
                                                     collect Val
                                                     when (NEQ Val
                                                               (QUOTE ..)))
                                          when (MEMBER (VAR Sel)
                                                       Vars]
          [for Val in ValueSet do (COND
                                    ((OR (GREATERP Val High)
                                         (LESSP Val Low))
                                      (printout T "Value " Val
                                 " out of range in Linear domain for variables "
                                                (PRINLIST Vars)
                                                "."))
                                    (SHOULDNT]
          (SETQ ValueSet (SORT ValueSet (FUNCTION LESSP)))
          (SETQ Vals ValueSet)
      LOOP(AND (NULL (CDR Vals))
               (GO OUT))
          (SETQ Interval (DIFFERENCE (CADR Vals)
                                     (CAR Vals)))
          [COND
            ((LESSP Interval (PLUS Epsilon Epsilon))
              (SETQ Vals (CDR Vals)))
            ((LESSP Interval (TIMES 3 Epsilon))
              (SETQ Interval (LIST (PLUS (CAR Vals)
                                         Epsilon)))
              (RPLACD Interval (CDR Vals))
              (RPLACD Vals Interval)
              (SETQ Vals (CDDR Vals)))
            (T [SETQ Interval (LIST (CONS (PLUS (CAR Vals)
                                                Epsilon)
```

```
                                        (DIFFERENCE (CADR Vals)
                                                    Epsilon]
                (RPLACD Interval (CDR Vals))
                (RPLACD Vals Interval)
                (SETQ Vals (CDDR Vals]
            (GO LOOP)
        OUT (RETURN ValueSet])
```

FUNCTION: SETUPSTRUCTURED
CALLS: SETUPLEAVES SETUPMASK SETUPPARENTMASKS SETUPSIBLINGS
CALLED BY: SETUPVBLS&DOMAINS

```
(SETUPSTRUCTURED
  [LAMBDA (Vars Domain)

          (* Set up internal structures for a set of structured
          variables with the same domain)


    (PROG (ValueSet Siblings Mask ParentMasks (DummyVar (QUOTE DummyVar)))
          (SETQ Siblings (SETUPSIBLINGS Domain))
          (SETQ ValueSet (SETUPLEAVES Domain Siblings))
          (SETQ Mask (SETUPMASK 1 (FLENGTH ValueSet)))
          (PUTPROP DummyVar (QUOTE MASK)
                   Mask)
          (SETQ ParentMasks (SETUPPARENTMASKS ValueSet Siblings DummyVar))
          (MAPC Vars (FUNCTION (LAMBDA (Var)
                     (PUTPROP Var (QUOTE DOMAINTYPE)
                              (QUOTE STRUCTURED))
                     (PUTPROP Var (QUOTE VALUESET)
                              ValueSet)
                     (PUTPROP Var (QUOTE SIBLINGS)
                              Siblings)
                     (PUTPROP Var (QUOTE MASK)
                              Mask)
                     (PUTPROP Var (QUOTE PARENTMASKS)
                              ParentMasks])
```

FUNCTION: SETUPSIBLINGS
CALLS: <none>
CALLED BY: SETUPSTRUCTURED

```
(SETUPSIBLINGS
  [LAMBDA (SubTreeList)

          (* Generate a Siblings List for a Structured domain
          specification)


    (PROG (SibList Group Parent Siblings)
          [MAPC SubTreeList (FUNCTION (LAMBDA (SubTree)
                    (SETQ Parent (CAR (LAST SubTree)))
                    (SETQ Siblings (CAR (LASTN SubTree 2)))
                    (COND
                      ((SETQ Group (ASSOC Parent SibList))
                        (RPLACD Group (UNION (CDR Group)
                                             Siblings)))
                      (T (SETQ SibList (NCONC SibList
                                              (LIST (CONS Parent Siblings]
          [SORT SibList (FUNCTION (LAMBDA (N1 N2)
                    (SOME (CDR (MEMBER N1 SibList))
                          (FUNCTION (LAMBDA (RightSib)
                              (MEMBER (CAR N1)
                                      (CDR RightSib]
          (RETURN SibList])
```

```
FUNCTION: SETUPLEAVES
CALLS: <none>
CALLED BY: SETUPSTRUCTURED

(SETUPLEAVES
  [LAMBDA (SubTreeList SiblingList)

          (* Generate a list of all leaves <terminal nodes> from a
          structured domain specification and the associated Siblings
          List)


    (PROG (Leaves)
          [MAPC SubTreeList (FUNCTION (LAMBDA (SubTree)
                    (MAPC (APPEND (CAR (LASTN SubTree 2))
                                  (LAST SubTree))
                          (FUNCTION (LAMBDA (Node)
                              (COND
                                ((ASSOC Node SiblingList))
                                (T (SETQ Leaves (UNION Leaves (LIST Node]
          (RETURN Leaves])
```

```
FUNCTION: SETUPPARENTMASKS
CALLS: SETUPPARENTMASK
CALLED BY: SETUPSTRUCTURED

(SETUPPARENTMASKS
  [LAMBDA (ValueSet SiblingList DummyVar)

          (* Set up a list of Parent Masks for the parent nodes of a
          structured domain)


    (MAPCAR SiblingList (FUNCTION (LAMBDA (Group)
                (CONS (CAR Group)
                      (SETUPPARENTMASK (CDR Group)
                                       DummyVar ValueSet SiblingList
                                       (ZEROMASK DummyVar])




FUNCTION: SETUPPARENTMASK
CALLS: POSN SETUPPARENTMASK
CALLED BY: SETUPPARENTMASK SETUPPARENTMASKS

(SETUPPARENTMASK
  [LAMBDA (Siblings DummyVar ValueSet SiblingList Mask)

          (* Create a parent mask for a parent node in a structured
          domain)


    [PROG (Node)
          (MAPC Siblings (FUNCTION (LAMBDA (Sib)
                    (COND
                      [(SETQ Node (ASSOC Sib SiblingList))
                        (SETQ Mask (DMASKLOGOR Mask (SETUPPARENTMASK
                                                      (CDR Node)
                                                      DummyVar ValueSet SiblingList
                                                      Mask]
                      (T (SETQ Mask (DMASKLOGOR Mask
                                                (MASKLLSH (ONEMASK DummyVar)
                                                          (POSN Sib ValueSet 0]
    Mask])



FUNCTION: SETUPMASK
```

```
CALLS: WORDMASK
CALLED BY: SETUPLINEAR SETUPNOMINAL SETUPSTRUCTURED

(SETUPMASK
  [LAMBDA (LowBitPos HighBitPos)

          (* Build a bit string mask with all bits from LowBitPos to
          HighBitPos (inclusive) set.)


    (PROG (Val)
      LOOP[COND
            ((IGREATERP LowBitPos WordSize)
              (SETQ Val (NCONC Val (LIST AllZeros)))
              (SETQ HighBitPos (IDIFFERENCE HighBitPos WordSize))
              (SETQ LowBitPos (IDIFFERENCE LowBitPos WordSize)))
            [(IGREATERP LowBitPos 0)
              (COND
                ((IGREATERP HighBitPos WordSize)
                  [SETQ Val (NCONC Val (LIST (WORDMASK LowBitPos WordSize]
                  (SETQ HighBitPos (IDIFFERENCE HighBitPos WordSize))
                  (SETQ LowBitPos 0))
                (T (COND
                     ((NULL Val)
                       (RETURN (WORDMASK LowBitPos HighBitPos)))
                     (T (RETURN (NCONC Val (LIST (WORDMASK LowBitPos HighBitPos]
            (T (COND
                 ((IGREATERP HighBitPos WordSize)
                   (SETQ Val (NCONC Val (LIST AllOnes)))
                   (SETQ HighBitPos (IDIFFERENCE HighBitPos WordSize)))
                 (T (RETURN (NCONC Val (LIST (WORDMASK 1 HighBitPos]
          (GO LOOP])



FUNCTION: WORDMASK
CALLS: <none>
CALLED BY: SETUPMASK

(WORDMASK
  [LAMBDA (LowBitPos HighBitPos)

          (* Return a bit string with bits from LowBitPos to
          HighBitPos (inclusive) set. Bits are numbered 1 ..
          WordSize.)


    (PROG [[LowSeg (SUB1 (LLSH 1 (SUB1 LowBitPos]
           (HighSeg (COND
```

```
                    ((ILESSP HighBitPos WordSize)
                      (SUB1 (LLSH 1 HighBitPos)))
                    (T (LOGOR 1 (LLSH (SUB1 (LLSH 1 (SUB1 HighBitPos)))
                                      1]
          (RETURN (LOGXOR HighSeg LowSeg])
```

```
FUNCTION: SETUPSELECTOR
CALLS: BUILDLINEAR BUILDNOMINAL BUILDSTRUCTURED
CALLED BY: SETUPEVENTS&CLASSES

(SETUPSELECTOR
  [LAMBDA (Selector)

          (* Transform a selector from external to internal
          representation)


    (BUILDSELECTOR (VAR Selector)
                   (SELECTQ (DOMAINTYPE (VAR Selector))
                            (NOMINAL (BUILDNOMINAL (VAR Selector)
                                                   (RELATION Selector)
                                                   (CDDR Selector)))
                            (LINEAR (BUILDLINEAR (VAR Selector)
                                                 (RELATION Selector)
                                                 (CDDR Selector)))
                            (STRUCTURED (BUILDSTRUCTURED (VAR Selector)
                                                         (RELATION Selector)
                                                         (CDDR Selector)))
                            (SHOULDNT])
```

```
FUNCTION: BUILDNOMINAL
CALLS: NEGATEVALS
CALLED BY: SETUPSELECTOR

(BUILDNOMINAL
  [LAMBDA (Var Relation ValList)

          (* Build a mask corresponding to a nominal selector with
          variable Var and list of values VaList)


    (PROG ((ValueSet (VALUESET Var))
```

```
          (Mask (ZEROMASK Var))
          (MaskBit (ONEMASK Var)))
         (for V in ValList do (printout T V
                                    " is NOT a valid value for variable "
                                       Var "." T)
                              (SHOULDNT)
           unless (MEMBER V ValueSet))
         (for Val in ValueSet do [COND
                                   ((MEMBER Val ValList)
                                     (SETQ Mask (DMASKLOGOR Mask MaskBit]
                                  (SETQ MaskBit (MASKLLSH MaskBit 1)))
         (SELECTQ Relation
                  (= (RETURN Mask))
                  (# (RETURN (NEGATEVALS Var Mask)))
                  (SHOULDNT])
```

FUNCTION: BUILDLINEAR
CALLS: NEGATEVALS POSN
CALLED BY: SETUPSELECTOR

```
(BUILDLINEAR
  [LAMBDA (Var Relation ValList)

          (* Transform the valueset of a linear selector to internal
          form)


    (PROG ((ValueSet (VALUESET Var))
           (Mask (ZEROMASK Var))
           MaskBit ValRange)
          (COND
            ((NULL ValList)
              (printout T "A selector for variable " Var "has no values." T)
              (SHOULDNT)))
          [for V in ValList do (printout T V
                                    " is NOT a valid value for variable "
                                       Var "." T)
                               (SHOULDNT)
             unless (OR (MEMBER V ValueSet)
                        (EQ V (QUOTE ..]
      LOOP[COND
            ((NULL ValList)
              (GO OUT))
            ((EQ (CADR ValList)
                 (QUOTE ..))
              (SETQ MaskBit (MASKLLSH (ONEMASK Var)
                                      (POSN (CAR ValList)
```

```
                                          ValueSet 0)))
            (SETQ ValRange (MEMBER (CAR ValList)
                                   ValueSet))
            (for Val in ValRange do (SETQ Mask (DMASKLOGOR Mask MaskBit))
                                    (SETQ MaskBit (MASKLLSH MaskBit 1))
               repeatuntil (OR (EQ Val (CADDR ValList))
                               (NULL Val)))
            (SETQ ValList (CDDDR ValList)))
          (T [SETQ Mask (DMASKLOGOR Mask (MASKLLSH (ONEMASK Var)
                                                   (POSN (CAR ValList)
                                                         ValueSet 0]
             (SETQ ValList (CDR ValList]
        (GO LOOP)
     OUT (SELECTQ Relation
                (= (RETURN Mask))
                (# (RETURN (NEGATEVALS Var Mask)))
                (SHOULDNT])
```

FUNCTION: BUILDSTRUCTURED
CALLS: NEGATEVALS
CALLED BY: SETUPSELECTOR

```
(BUILDSTRUCTURED
  [LAMBDA (Var Relation ValList)

          (* Build a mask corresponding to a structured selector with
          variable Var and list of values ValList)


    (PROG ((ValueSet (VALUESET Var))
           (ParentMasks (PARENTMASKS Var))
           (Mask (ZEROMASK Var))
           (MaskBit (ONEMASK Var)))
          [for V in ValList do (printout T V
                                         " is NOT a valid value for variable "
                                           Var "." T)
                               (SHOULDNT)
            when (AND (NOT (MEMBER V ValueSet))
                      (NULL (ASSOC V ParentMasks]
          (for Val in ValueSet do [COND
                                    ((MEMBER Val ValList)
                                      (SETQ Mask (DMASKLOGOR Mask MaskBit]
                                  (SETQ MaskBit (MASKLLSH MaskBit 1)))
          (for PM in ParentMasks do (SETQ Mask (DMASKLOGOR Mask (CDR PM)))
             when (MEMBER (CAR PM)
                          ValList))
          (SELECTQ Relation
```

```
                (= (RETURN Mask))
                (# (RETURN (NEGATEVALS Var Mask)))
                (SHOULDNT])
```

```
FUNCTION: SETUPEVENTS&CLASSES
CALLS: RESETCLASSES SETUPSELECTOR
CALLED BY: INITIALIZE

(SETUPEVENTS&CLASSES
  [LAMBDA (DataSet)

          (* Convert the Events&Classes list from a data set to
          internal structures used by the AQ functions.)


    (PROG (E&C TempList Event Class Complex)
          (RESETCLASSES)
          (SETQ ClassNames NIL)
          [for E&C in (CADDR DataSet)
             do (SETQ Event (CAR (LASTN E&C 2)))
                (SETQ Class (CAR (LAST E&C)))
                (SETQ Complex NIL)
                [COND
                  [(ATOM (CAR Event))
                    (for Val in Event as Var in AllVars
                       do (SETQ Complex (NEWSELECTOR Complex
                                                     (SETUPSELECTOR
                                                       (LIST Var (QUOTE =)
                                                             Val]
                  (T (for Sel in Event do (SETQ Complex
                                                 (NEWSELECTOR Complex
                                                              (SETUPSELECTOR Sel]
                (SETEVENTS Class (NCONC (EVENTS Class)
                                        (LIST Complex)))
                (SETQ ClassNames (UNION ClassNames (LIST Class]
          (RETURN (SORT ClassNames])
```

```
FUNCTION: INTERSECTINGCOVERS
CALLS: AQ-MULTI AQ-SINGLE
CALLED BY: <none>

(INTERSECTINGCOVERS
```

```
  [LAMBDA (Classes)

          (* Generate a cover of a class in an intersecting-cover
          mode)


    (COND
      (MultiFlag (AQ-MULTI Classes NIL NIL))
      (T (AQ-SINGLE Classes NIL NIL])
```

```
FUNCTION: DISJOINTCOVERS
CALLS: AQ-MULTI AQ-SINGLE
CALLED BY: <none>

(DISJOINTCOVERS
  [LAMBDA (Classes)

          (* Apply AQ to produce disjoint (but with possibly
          intersecting interval) covers)


    (COND
      (MultiFlag (AQ-MULTI Classes T NIL))
      (T (AQ-SINGLE Classes T NIL])
```

```
FUNCTION: DISJOINTCOMPLEXCOVER
CALLS: AQ-MULTI AQ-SINGLE
CALLED BY: <none>

(DISJOINTCOMPLEXCOVER
  [LAMBDA (Classes)

          (* Apply AQ to produce disjoint covers with disjoint
          complexes in the covers)


    (COND
      (MultiFlag (AQ-MULTI Classes T T))
      (T (AQ-SINGLE Classes T T])
```

```
FUNCTION: ORDEREDCOVER
CALLS: AQ
CALLED BY: <none>

(ORDEREDCOVER
  [LAMBDA (Classes)

          (* Apply AQ to produce covers in the ordered
          (VL1) mode)


    (MAP Classes (FUNCTION (LAMBDA (ClassTail)
             (AQ (CAR ClassTail)
                 (CDR ClassTail)
                 NIL NIL])
```

```
FUNCTION: AQ
CALLS: AQ-MR
CALLED BY: AQ-SINGLE ORDEREDCOVER

(AQ
  [LAMBDA (PosClass NegClasses DisjointCoverFlag DisjointComplexFlag)

          (* Performs the top level of the AQ algorithm to generate a
          cover of the complexes in PosClass aqainst the complexes in
          NegClasses)


    (PROG NIL
      LOOP(AND (AQ-MR PosClass NegClasses DisjointCoverFlag DisjointComplexFlag)
               (GO LOOP))
          (RETURN])
```

```
FUNCTION: AQ-SINGLE
CALLS: AQ
CALLED BY: DISJOINTCOMPLEXCOVER DISJOINTCOVERS INTERSECTINGCOVERS

(AQ-SINGLE
  [LAMBDA (Classes DisjointCoverFlag DisjointComplexFlag)

          (* Produce covers for Classes in the mode indicated by
          DisjointCoverFlag and DisjointComplexFlag by covering one
```

```
          class at a time)


    (MAPC Classes (FUNCTION (LAMBDA (Class)
              (AQ Class (REMOVE Class Classes)
                  DisjointCoverFlag DisjointComplexFlag])
```

```
FUNCTION: AQ-MULTI
CALLS: AQ-MR
CALLED BY: DISJOINTCOMPLEXCOVER DISJOINTCOVERS INTERSECTINGCOVERS

(AQ-MULTI
  [LAMBDA (Classes DisjointCoverFlag DisjointComplexFlag)

          (* Produce covers for Classes in the mode indicated by
          DisjointCoverFlag and DisjointComplexFlag by cycling through
          the classes, producing one covering complex per class, until
          the events in each class are covered.
          Slightly faster than AQ-SINGLE.)


    (PROG ((ActiveClasses Classes))
      LOOP[SETQ ActiveClasses (SUBSET ActiveClasses
                                      (FUNCTION (LAMBDA (Class)
                                          (AQ-MR Class (REMOVE Class Classes)
                                                 DisjointCoverFlag
                                                 DisjointComplexFlag]
          (AND ActiveClasses (GO LOOP])
```

```
FUNCTION: AQ-MR
CALLS: BESTCOMP COVEREDBYCOMPLEX KNOCKOUT PPCOMP STAR
CALLED BY: AQ AQ-MULTI

(AQ-MR
  [LAMBDA (PosClass NegClasses DisjointCoverFlag DisjointComplexFlag)

          (* Reentrant AQ function, finds and adds one covering
          complex to the COVER of PosClass. Returns the list of
          remaining uncovered events from PosClass)


    (PROG (Event Star FList BestComplex CoveredEvents)
```

```
        (AND (NULL (UNCOVEREDEVENTS PosClass))
             (GO OUT))
        (SETQ Event (CAR (SEEDLIST PosClass)))
        (AND (EQ Event (OLDEVENT PosClass))
             (PRIN1 "Error in AQ: infinite loop ")
             (SHOULDNT))
        [COND
          [DisjointCoverFlag
            (SETQ FList (MAPCONC NegClasses
                                 (FUNCTION (LAMBDA (NegClass)
                                     (APPEND (COVER NegClass)
                                             (APPEND (UNCOVEREDEVENTS
                                                                 NegClass]
          (T (SETQ FList (MAPCONC NegClasses (FUNCTION (LAMBDA (NegClass)
                                      (APPEND (EVENTS NegClass]
        [COND
          (DisjointComplexFlag (SETQ FList (APPEND FList (COVER PosClass]
        (SETQ Star (STAR Event FList (EVENTS PosClass)
                         (UNCOVEREDEVENTS PosClass)))
        (AND (NULL Star)
             (printout T "Error in AQ:" T "The event " # (PPCOMP Event)
                       T "in class " PosClass
                       " overlaps with an event in another class."
                       T "Please correct the data set.")
             (SHOULDNT))
        (SETSEEDLIST PosClass (KNOCKOUT Star (SEEDLIST PosClass)))
        (SETQ BestComplex (BESTCOMP Star (EVENTS PosClass)
                                    (UNCOVEREDEVENTS PosClass)))
        (SETQ CoveredEvents (COVEREDBYCOMPLEX BestComplex (EVENTS PosClass)))
        (SETCOVER PosClass (CONS BestComplex (COVER PosClass)))
        (SETCOVEREDEVENTS PosClass (CONS CoveredEvents
                                         (COVEREDEVENTS PosClass)))
        (SETUNCOVEREDEVENTS PosClass (LDIFFERENCE (UNCOVEREDEVENTS PosClass)
                                                  CoveredEvents))
        (SETOLDEVENT PosClass Event)
    OUT (AND (NULL (SEEDLIST PosClass))
             (SETSEEDLIST PosClass (UNCOVEREDEVENTS PosClass)))
        (RETURN (UNCOVEREDEVENTS PosClass])
```

```
FUNCTION: BESTCOMP
CALLS: TRUNCATE
CALLED BY: AQ-MR

(BESTCOMP
  [LAMBDA (Star PositiveEvents UnCoveredEvents)

          (* Finds the best complex from a list of complexes
```

```
          (Star) given the list of events to cover
          (PositiveEvents) and those events yet to be covered
          (UnCoveredEvents))


    (CAR (TRUNCATE Star 1 1 Criteria&ToleranceList PositiveEvents
                   UnCoveredEvents])
```

```
FUNCTION: #COVERED
CALLS: INCLUDES
CALLED BY: <none>

(#COVERED
  [LAMBDA (Comp **DUMMY** Events)

          (* Determines the number of events in "Events" covered by
          "Comp", this is a cost function used in the default
          Criteria&ToleranceList)


    (PROG ((**Val 0))
          [MAPC Events (FUNCTION (LAMBDA (Ev)
                    (COND
                      ((INCLUDES Comp Ev)
                        (SETQ **Val (ADD1 **Val]
          (RETURN (MINUS **Val])
```

```
FUNCTION: NUMBEROFSELECTORS
CALLS: <none>
CALLED BY: <none>

(NUMBEROFSELECTORS
  [LAMBDA (Comp **Dummy1** **Dummy2**)

          (* Counts the number of selector in complex Comp , this is a
          cost function used in the default Criteria&ToleranceList)


    (FLENGTH Comp])
```

```
FUNCTION: COVEREDBYCOMPLEX
CALLS: INCLUDES
CALLED BY: AQ-MR

(COVEREDBYCOMPLEX
  [LAMBDA (Cover EventList)

          (* Return a list of all events in EventList that are covered
          by complex cover)


    (SUBSET EventList (FUNCTION (LAMBDA (Event)
                (INCLUDES Cover Event])
```

```
FUNCTION: KNOCKOUT
CALLS: INCLUDES
CALLED BY: AQ-MR

(KNOCKOUT
  [LAMBDA (OuterComps InnerComps)

          (* Removes all complexes in InnerComps that are covered by
          some complex in OuterComps)


    (SUBSET InnerComps (FUNCTION (LAMBDA (InComp)
                (NOTANY OuterComps (FUNCTION (LAMBDA (OutComp)
                            (INCLUDES OutComp InComp])
```

```
FUNCTION: EARLIERVAR
CALLS: <none>
CALLED BY: NEWSELECTOR

(EARLIERVAR
  [LAMBDA (X Y)

          (* Used to order selectors within a complex, sorts by print
          name of the variables in the selectors)


    (ALPHORDER (VAR X)
               (VAR Y])
```

```
FUNCTION: EXTENDAGAINST
CALLS: EXTENDAGAINSTVALS
CALLED BY: STAR

(EXTENDAGAINST
  [LAMBDA (C1 C2)

          (* Build a new star of selectors by extending the cover C1
          against C2)


    (PROG (FS NewVals)
          (RETURN (MAPCONC C1
                           (FUNCTION (LAMBDA (ES)
                               (SETQ FS (FASSOC (VAR ES)
                                                C2))
                               (COND
                                 ((NULL FS)
                                   NIL)
                                 (T (SETQ C2 (CDR C2))
                                    (SETQ NewVals (EXTENDAGAINSTVALS
                                        (VAR ES)
                                        (VALS ES)
                                        (VALS FS)))
                                    (AND NewVals
                                         (LIST (BUILDSELECTOR (VAR ES)
                                                              NewVals])


FUNCTION: EXTENDAGAINSTVALS
CALLS: EXTENDAGAINSTLINEAR EXTENDAGAINSTNOMINAL EXTENDAGAINSTSTRUCTURE
CALLED BY: EXTENDAGAINST

(EXTENDAGAINSTVALS
  [LAMBDA (Var PosVals NegVals)

          (* Extend PosVals against NegVals for variable Var)


    (SELECTQ (DOMAINTYPE Var)
             (NOMINAL (EXTENDAGAINSTNOMINAL Var PosVals NegVals))
             (LINEAR (EXTENDAGAINSTLINEAR Var PosVals NegVals))
             (STRUCTURED (EXTENDAGAINSTSTRUCTURE Var PosVals NegVals))
             (SHOULDNT])
```

```
FUNCTION: EXTENDAGAINSTNOMINAL
CALLS: NEGATEVALS
CALLED BY: EXTENDAGAINSTVALS

(EXTENDAGAINSTNOMINAL
  [LAMBDA (Var PosVals NegVals)

          (* Extend nominal selector with vals PosVals against nominal
          selector with vals NegVals)


    (AND (MASKZEROPLOGAND PosVals NegVals)
         (NEGATEVALS Var NegVals])
```

```
FUNCTION: EXTENDAGAINSTLINEAR
CALLS: <none>
CALLED BY: EXTENDAGAINSTVALS

(EXTENDAGAINSTLINEAR
  [LAMBDA (Var PosVals NegVals)

          (* Extend linear selector with values PosVals against linear
          selector with values NegVals)


    (PROG (Marker NewVals)
          (AND (NOT (MASKZEROPLOGAND PosVals NegVals))
               (RETURN NIL))
          (SETQ Marker (COPY PosVals))
          (SETQ NewVals (COPY PosVals))
          (SETQ NegVals (DMASKLOGOR (MASKNEGATE (MASK Var))
                                    NegVals))
      RIGHT
          (AND (MASKZEROP Marker)
               (GO NEXT))
          (SETQ Marker (MASKLRSH Marker 1))
          (SETQ Marker (DMASKERASE Marker NegVals))
          (SETQ NewVals (DMASKLOGOR NewVals Marker))
          (GO RIGHT)
      NEXT(SETQ Marker (COPY PosVals))
      LEFT(AND (MASKZEROP Marker)
               (GO OUT))
          (SETQ Marker (MASKLLSH Marker 1))
          (SETQ Marker (DMASKERASE Marker NegVals))
          (SETQ NewVals (DMASKLOGOR NewVals Marker))
          (GO LEFT)
      OUT (RETURN NewVals])
```

```
FUNCTION: EXTENDAGAINSTSTRUCTURE
CALLS: NEGATEVALS
CALLED BY: EXTENDAGAINSTVALS

(EXTENDAGAINSTSTRUCTURE
  [LAMBDA (Var Pos Neg)

          (* Extend structured selector with values Pos against Neg)


    (PROG (NewPos)
          (AND (NOT (MASKZEROPLOGAND Pos Neg))
               (RETURN NIL))
          (SELECTQ ExtendStrucMode
                   (NOMINAL (RETURN (NEGATEVALS Var Neg)))
                   (MAXIMAL (SETQ NewPos (COPY Pos))
                            [MAPC (PARENTMASKS Var)
                                  (FUNCTION (LAMBDA (PM)
                                      (COND
                                        ((MASKZEROPLOGAND (CDR PM)
                                                          Neg)
                                          (SETQ NewPos (DMASKLOGOR NewPos
                                                                   (CDR PM]
                            (RETURN NewPos))
                   (MINIMAL
                     (SETQ NewPos (COPY Pos))
                     (SETQ Pos (COPY Pos))
                     [MAPC (PARENTMASKS Var)
                           (FUNCTION (LAMBDA (PM)
                               (COND
                                 ((AND (MASKZEROPLOGAND (CDR PM)
                                                        Neg)
                                       (NOT (MASKZEROPLOGAND (CDR PM)
                                                             Pos)))
                                  (SETQ Pos (DMASKERASE Pos (CDR PM)))
                                  (SETQ NewPos (DMASKLOGOR NewPos (CDR PM]
                     (RETURN NewPos))
                   (SHOULDNT])



FUNCTION: INCLUDES
CALLS: INCLUDESVALS
CALLED BY: #COVERED ABSORB COVEREDBYCOMPLEX KNOCKOUT

(INCLUDES
  [LAMBDA (Comp Event)

          (* Determines if the complex "Comp" includes all points in
```

```
          "Event")

    (PROG (InSel)
          (RETURN (EVERY Comp (FUNCTION (LAMBDA (OutSel)
                                (SETQ InSel (FASSOC (VAR OutSel)
                                                    Event))
                                (COND
                                  ((NULL InSel)
                                    NIL)
                                  (T (SETQ Event (CDR Event))
                                     (INCLUDESVALS (VALS OutSel)
                                                   (VALS InSel])
```

```
FUNCTION: INCLUDESVALS
CALLS: <none>
CALLED BY: ABSORBP INCLUDES

(INCLUDESVALS
  [LAMBDA (OutVals InVals)

          (* Determine if OutVals includes all points in InVals)

    (MASKINCLUDESP OutVals InVals])
```

```
FUNCTION: MULTIPLY
CALLS: ABSORBP PRODUCTSC
CALLED BY: STAR

(MULTIPLY
  [LAMBDA (CompSet SelSet)

          (* Multiply a list of complexes with a list of selectors)

    (MAPCONC CompSet (FUNCTION (LAMBDA (Comp)
                (COND
                  ((ABSORBP Comp SelSet)
                    (LIST Comp))
                  (T (MAPCONC SelSet (FUNCTION (LAMBDA (Sel)
                                     (PRODUCTSC Sel Comp])
```

```
FUNCTION: ABSORBP
CALLS: INCLUDESVALS
CALLED BY: MULTIPLY

(ABSORBP
  [LAMBDA (PComp ERComp)

          (* Determine if some selector in ERComp includes some
          selector in PComp. If so, the intersection of PComp and the
          negative event used to form ERComp is null, so multiplying
          PComp by the selectors in ERComp is unnecessary.)


    (PROG (PSel)
          (RETURN (SOME ERComp (FUNCTION (LAMBDA (ERSel)
                               (SETQ PSel (FASSOC (VAR ERSel)
                                                  PComp))
                               (COND
                                 ((NULL PSel)
                                   NIL)
                                 (T (SETQ PComp (CDR PComp))
                                    (INCLUDESVALS (VALS ERSel)
                                                  (VALS PSel])
```

```
FUNCTION: PRODUCTSC
CALLS: VALSPRODUCT
CALLED BY: MULTIPLY

(PRODUCTSC
  [LAMBDA (Sel Comp)

          (* Return a list of complex<es> which is the product of a
          selector and a complex)


    (PROG (NewVals OldSelector)
          (SETQ OldSelector (FASSOC (VAR Sel)
                                    Comp))
          (RETURN (COND
                    ((NULL OldSelector)
                      (LIST (ADDSELECTOR Comp Sel)))
                    (T (SETQ NewVals (VALSPRODUCT (VALS Sel)
                                                  (VALS OldSelector)))
                       (COND
                         ((NULL NewVals)
                           NIL)
                         ((OR (FIXP NewVals)
```

```
                        (FIXP (CAR NewVals)))
                   (LIST (REPLACESELECTOR (BUILDSELECTOR (VAR Sel)
                                                         NewVals)
                                          OldSelector Comp)))
              [(EQ (CAR NewVals)
                   (QUOTE //DISJUNCTION))
               (MAPCAR (CDR NewVals)
                       (FUNCTION (LAMBDA (NewV)
                           (REPLACESELECTOR
                             (BUILDSELECTOR (VAR Sel)
                                            NewV)
                             OldSelector Comp]
              ((SHOULDNT])
```

```
FUNCTION: VALSPRODUCT
CALLS: <none>
CALLED BY: PRODUCTSC

(VALSPRODUCT
  [LAMBDA (SelVals CompVals)

          (* Finds the product (intersection) of all points in "SelVals"
          and "CompVals")


    (PROG (NewVals)
          (SETQ NewVals (MASKLOGAND SelVals CompVals))
          (COND
            ((MASKZEROP NewVals)
              (RETURN NIL))
            (T (RETURN NewVals])
```

```
FUNCTION: STAR
CALLS: ABSORB EXTENDAGAINST MULTIPLY TRUNCATE
CALLED BY: AQ-MR

(STAR
  [LAMBDA (E FList PositiveEvents UnCoveredEList)

          (* Generates the star of complexes E against FList)
```

```
    (PROG (Product ExtSelList SelList)
          (SETQ Product (LIST NIL))
          [SETQ ExtSelList (MAPCAR FList (FUNCTION (LAMBDA (F)
                                          (EXTENDAGAINST E F]
          [MAPC ExtSelList (FUNCTION (LAMBDA (SelList)
                    (SETQ Product (TRUNCATE (MULTIPLY Product SelList)
                                            MaxStar CutStar
                                            Criteria&ToleranceList
                                            PositiveEvents UnCoveredEList]
          (RETURN Product])
```

```
FUNCTION: TRUNCATE
CALLS: ABSORB CUTSTAR
CALLED BY: BESTCOMP STAR

(TRUNCATE
  [LAMBDA (Star MaxSize TrimSize C&TList PositiveEvents UnCoveredEvents)

          (* Cuts (trims) the size of a star to TrimSize if it exceeds
          MaxSize in size, using the optimality criteria specified in
          "C&TList")


    (COND
      ((ILEQ (FLENGTH Star)
             MaxSize)
        Star)
      (T (PROG (C&T)
               (SETQ Star (ABSORB Star))
          LOOP(COND
                ((OR (NULL C&TList)
                     (ILEQ (FLENGTH Star)
                           TrimSize))
                  (RETURN)))
               (SETQ C&T (CAR C&TList))
               (SETQ Star (CUTSTAR Star (CAR C&T)
                                   (CADR C&T)
                                   PositiveEvents UnCoveredEvents))
               (SETQ C&TList (CDR C&TList))
               (GO LOOP))
         (BESTN Star TrimSize])
```

```
FUNCTION: ABSORB
CALLS: INCLUDES
CALLED BY: STAR TRUNCATE

(ABSORB
  [LAMBDA (PStar)

          (* Remove complexes from PStar that are redundant with
          respect to inclusion)


    (PROG (Outer Inner)
          [SORT PStar (FUNCTION (LAMBDA (A B)
                    (ILEQ (FLENGTH A)
                          (FLENGTH B]
          (SETQ PStar (CONS (QUOTE DUMMY)
                            PStar))
          (SETQ Outer PStar)
      OLOOP
          (AND (NULL (CDR Outer))
               (GO OUT))
          (SETQ Inner PStar)
      ILOOP
          (COND
            ((NULL (CDR Inner))
              (SETQ Outer (CDR Outer))
              (GO OLOOP))
            ([OR (EQ Outer Inner)
                 (NOT (INCLUDES (CADR Outer)
                                (CADR Inner]
              (SETQ Inner (CDR Inner))
              (GO ILOOP))
            (T (RPLACD Inner (CDDR Inner))
               (AND (EQ (CDR Outer)
                        (CDR Inner))
                    (SETQ Outer Inner))
               (GO ILOOP)))
      OUT (RETURN (CDR PStar])



FUNCTION: CUTSTAR
CALLS: <none>
CALLED BY: TRUNCATE

(CUTSTAR
  [LAMBDA (Star CritFN Tolerance PositiveEvents UnCoveredEvents)

          (* Trims "Star" to those complexes that are optimal
```

```
          according to function "CritFN" applied to "Tolerance")


    (PROG (VList Max Min Tol)
          [SETQ VList (MAPCAR Star (FUNCTION (LAMBDA (Comp)
                                   (APPLY CritFN (LIST Comp PositiveEvents
                                                       UnCoveredEvents]
          (SETQ Max (APPLY (FUNCTION MAX)
                           VList))
          (SETQ Min (APPLY (FUNCTION MIN)
                           VList))
          (SETQ Tol (PLUS (TIMES Tolerance (DIFFERENCE Max Min))
                          Min))
          (RETURN (SUBSET Star (FUNCTION (LAMBDA (Comp)
                               (PROG1 (NOT (GREATERP (CAR VList)
                                                     Tol))
                                      (SETQ VList (CDR VList])
```

FUNCTION: REFUNION
CALLS: VALSUNION
CALLED BY: <none>

```
(REFUNION
  [LAMBDA (CompList)

          (* Finds the REFUNION of a list of complexes)


    (PROG (Union NewVals Sel2)
          (SETQ Union (CAR CompList))
          [MAPC
            (CDR CompList)
            (FUNCTION (LAMBDA (Comp)
                (SETQ Union
                  (MAPCONC
                    Comp
                    (FUNCTION (LAMBDA (Sel1)
                        (SETQ Sel2 (FASSOC (VAR Sel1)
                                           Union))
                        (COND
                          ((NULL Sel2)
                            NIL)
                          (T (SETQ Union (CDR Union))
                             (SETQ NewVals (VALSUNION (VAR Sel1)
                                                      (VALS Sel1)
                                                      (VALS Sel2)))
                             (COND
```

```
                                ((EQ NewVals (QUOTE *))
                                  NIL)
                                (T (LIST (BUILDSELECTOR (VAR Sel1)
                                                        NewVals]
          (RETURN Union])
```

```
FUNCTION: VALSUNION
CALLS: <none>
CALLED BY: REFUNION

(VALSUNION
  [LAMBDA (Var Vals1 Vals2)

          (* Finds the union of Vals1 and Vals2 for variable Var.
          Returns "*" if the union is all of the values in the
          domain.)


    (PROG (NewVals)
          (SETQ NewVals (MASKLOGOR Vals1 Vals2))
          (COND
            ((MASKEQUAL NewVals (MASK Var))
              (RETURN (QUOTE *)))
            (T (RETURN NewVals])
```

```
FUNCTION: NEGATECOMPLEX
CALLS: NEGATEVALS
CALLED BY: <none>

(NEGATECOMPLEX
  [LAMBDA (Complex)

          (* Returns the "negative" of a complex, ie all points not
          within the complexes space)


    (MAPCAR Complex (FUNCTION (LAMBDA (Sel)
              (LIST (BUILDSELECTOR (VAR Sel)
                                   (NEGATEVALS (VAR Sel)
                                               (VALS Sel])
```

FUNCTION: NEGATEVALS
CALLS: <none>
CALLED BY: BUILDLINEAR BUILDNOMINAL BUILDSTRUCTURED EXTENDAGAINSTNOMINAL
EXTENDAGAINSTSTRUCTURE NEGATECOMPLEX PRINTNOMINAL

```
(NEGATEVALS
  [LAMBDA (Var Vals)

          (* Finds the negative of the Vals for variable Var.)


    (MASKLOGXOR (MASK Var)
                Vals])
```

FUNCTION: SHOWCOVERS
CALLS: SHOWCOVER
CALLED BY: AQVAL

```
(SHOWCOVERS
  [LAMBDA (Classes)

          (* Pretty-print the covers of a list of classes)


    (printout T T T "Event Coverage" .TAB 18 "The Covers are: " T " Total    " T)
    (for ClassName in Classes
       do (printout T .TAB 18 "Cover of Class: " ClassName T #
                    (SHOWCOVER (REVERSE (COVER ClassName))
                               (REVERSE (COVEREDEVENTS ClassName))
                               18)
                    T T])
```

FUNCTION: SHOWCOVER
CALLS: PPCOMP
CALLED BY: SHOWCOVERS

```
(SHOWCOVER
  [LAMBDA (Cover CoveredEvents Column)

          (* Pretty print a class cover with event coverage stats.)
```

```
    (for Complex in Cover as CoveredComps in CoveredEvents
       do (TAB 3)
          (PRIN1 (LENGTH CoveredComps))
          (TAB Column)
          (PPCOMP Complex])
```

```
FUNCTION: PPCOMPS
CALLS: PPCOMP
CALLED BY: <none>

(PPCOMPS
  [LAMBDA (CompList)

          (* Pretty-print a list of complexes)

    (for Comp in CompList do (PPCOMP Comp)
                             (TERPRI])
```

```
FUNCTION: PPCOMP
CALLS: PRINTLINEAR PRINTNOMINAL PRINTSTRUCTURE
CALLED BY: AQ-MR PPCOMPS SHOWCOVER

(PPCOMP
  [LAMBDA (Comp)

          (* Pretty-print a complex)

     (for Sel in Comp do (SELECTQ (DOMAINTYPE (VAR Sel))
                                  (NOMINAL (PRINTNOMINAL Sel))
                                  (LINEAR (PRINTLINEAR Sel))
                                  (STRUCTURED (PRINTSTRUCTURE Sel))
                                  (SHOULDNT])
```

```
FUNCTION: PRINTNOMINAL
CALLS: MASKONBITS NEGATEVALS
```

CALLED BY: PPCOMP

```
(PRINTNOMINAL
  [LAMBDA (Sel)

          (* Pretty-print a single nominal selector)


    (PROG ((VarName (VAR Sel))
           (Vals (VALS Sel))
           (Mask (MASK (VAR Sel)))
           (MaskBit (ONEMASK (VAR Sel)))
           Relation Printed)
          (COND
            ((MASKEQUAL Vals Mask)
              (RETURN NIL)))
          (COND
            ((GREATERP (MASKONBITS Vals)
                       (MASKONBITS (NEGATEVALS VarName Vals)))
              (SETQQ Relation #)
              (SETQ Vals (NEGATEVALS VarName Vals)))
            (T (SETQQ Relation =)))
          (printout T "[" VarName " " Relation " ")
          (for VarValue in (VALUESET VarName)
             do (COND
                  ((NOT (MASKZEROPLOGAND MaskBit Vals))
                    (COND
                      (Printed (PRIN1 ",")))
                    (PRIN1 VarValue)
                    (SETQ Printed T)))
                (SETQ MaskBit (MASKLLSH MaskBit 1)))
          (PRIN1 "]"])
```

FUNCTION: PRINTLINEAR
CALLS: <none>
CALLED BY: PPCOMP

```
(PRINTLINEAR
  [LAMBDA (Sel)

          (* Pretty-print a linear selector (nothing fancy))


    (PROG ((VarName (VAR Sel))
           (Vals (VALS Sel))
           (Mask (MASK (VAR Sel)))
           (MaskBit (ONEMASK (VAR Sel)))
```

```
          (ICount 0)
          LastVal Printed)
         (COND
           ((MASKEQUAL Vals Mask)
             (RETURN NIL)))
         (printout T "[" VarName " = ")
         (for V in (VALUESET VarName)
            do [COND
                 ((MASKZEROPLOGAND Vals MaskBit)
                   (COND
                     ((IGREATERP ICount 1)
                       (printout T " .. " LastVal)))
                   (SETQ ICount 0))
                 (T (SETQ ICount (ADD1 ICount))
                    [COND
                      ((EQP ICount 1)
                        (COND
                          (Printed (PRIN1 ",")))
                        (SETQ Printed T)
                        (COND
                          ((LISTP V)
                            (PRIN1 (CAR V))
                            (SETQ ICount (ADD1 ICount)))
                          (T (PRIN1 V]
                     (SETQ LastVal (COND
                         ((LISTP V)
                           (CDR V))
                         (T V]
               (SETQ MaskBit (MASKLLSH MaskBit 1)))
         (COND
           ((IGREATERP ICount 1)
             (printout T " .. " LastVal)))
         (PRIN1 "]"])
```

FUNCTION: PRINTSTRUCTURE
CALLS: <none>
CALLED BY: PPCOMP

```
(PRINTSTRUCTURE
  [LAMBDA (Sel)

          (* Pretty-print a single structured selector)


    (PROG ((UsedBits (ZEROMASK (VAR Sel)))
           (VarName (VAR Sel))
           (Vals (VALS Sel))
```

```
            (MaskBit (ONEMASK (VAR Sel)))
            (Mask (MASK (VAR Sel)))
            Printed)
           (COND
             ((MASKEQUAL Vals Mask)
               (RETURN))
             (T (printout T "[" VarName " = ")
                [for PMask in (REVERSE (PARENTMASKS VarName))
                   do (AND Printed (PRIN1 ","))
                      (PRIN1 (CAR PMask))
                      (SETQ UsedBits (MASKLOGOR UsedBits (CDR PMask)))
                      (SETQ Printed T)
                   when (AND (MASKINCLUDESP Vals (CDR PMask))
                             (NOT (MASKINCLUDESP UsedBits (CDR PMask]
                (for V in (VALUESET VarName)
                   do (COND
                        ((AND (MASKINCLUDESP Vals MaskBit)
                              (NOT (MASKINCLUDESP UsedBits MaskBit)))
                          (COND
                            (Printed (PRIN1 ",")))
                          (PRIN1 V)
                          (SETQ UsedBits (DMASKLOGOR UsedBits MaskBit))
                          (SETQ Printed T)))
                      (SETQ MaskBit (MASKLLSH MaskBit 1)))
                (PRIN1 "]"]
```

FUNCTION: PRINLIST
CALLS: <none>
CALLED BY: LISTDATA SETUPLARGELINEARVALS

```
(PRINLIST
  [LAMBDA (PList Char)

          (* Print the elements of a list without outer parenthesis)


    (PROG ((SeprChr (OR Char (QUOTE " ")))
           Printed)
          (for Item in PList do (AND Printed (PRIN1 SeprChr))
                                (PRIN1 Item)
                                (SETQ Printed T])
```

```
FUNCTION: MASKONBITS
CALLS: ONBITS
CALLED BY: PRINTNOMINAL

(MASKONBITS
  [LAMBDA (Mask)

          (* Returns the count of the number of on
          (set) bits in a bit string.)


    (COND
      ((FIXP Mask)
        (ONBITS Mask))
      (T (for A in Mask sum (ONBITS A])
```

```
FUNCTION: ONBITS
CALLS: <none>
CALLED BY: MASKONBITS

(ONBITS
  [LAMBDA (Word)

          (* Count the number of on (set) bits in an integer word)


    (for I from 1 to WordSize count (PROG1 (NOT (ZEROP (LOGAND Word LowBit)))
                                           (SETQ Word (LRSH Word 1])
```

```
FUNCTION: POSN
CALLS: POSN
CALLED BY: BUILDLINEAR POSN SETUPPARENTMASK

(POSN
  [LAMBDA (X XLIST I)

          (* Return an integer value corresponding to the position of
          X in XLIST)


    (COND
      ((NULL XLIST)
```

```
        NIL)
       ((EQ X (CAR XLIST))
        I)
       (T (POSN X (CDR XLIST)
                (ADD1 I])
```

```
FUNCTION: PRINBITS
CALLS: <none>
CALLED BY: PRINMASK

(PRINBITS
  [LAMBDA (Word)

          (* Print the bits of an integer word from low
          (left) to high (right))


    (for I from 1 to WordSize do (PRIN1 (LOGAND Word LowBit))
                                 (SETQ Word (LRSH Word 1)))
    (SPACES 1])
```

```
FUNCTION: PRINMASK
CALLS: PRINBITS
CALLED BY: <none>

(PRINMASK
  [LAMBDA (Mask)

          (* Print the bits of a bit string for low
          (left) to high (right))


    [COND
      ((FIXP Mask)
        (PRINBITS Mask))
      (T (MAPC Mask (FUNCTION PRINBITS]
    (TERPRI])
```

```
FUNCTION: PP-AQMACROS
CALLS: <none>
CALLED BY: <none>

(PP-AQMACROS
  [LAMBDA NIL
    (SETQ SYSPRETTYFLG T)
    (for X in (FILECOMSLST (QUOTE JMB$AQ)
                           (QUOTE MACRO))
       do (printout T X T # (SHOWPRINT (GETPROP X (QUOTE MACRO)))
                    T T))
    (SETQ SYSPRETTYFLG NIL])
```

```
FUNCTION: REDO-AQMACROS
CALLS: <none>
CALLED BY: <none>

(REDO-AQMACROS
  [LAMBDA NIL
    (for X in (FILECOMSLST (QUOTE JMB$AQ)
                           (QUOTE MACRO))
       do (PUTPROP X (QUOTE MACRO)
                   (GETD X])
```

```
FUNCTION: UNDO-AQMACROS
CALLS: <none>
CALLED BY: <none>

(UNDO-AQMACROS
  [LAMBDA NIL
    (for X in (FILECOMSLST (QUOTE JMB$AQ)
                           (QUOTE MACRO))
       do (PUTD X (GETPROP X (QUOTE MACRO])
```

```
FUNCTION: MACRO
CALLS: <none>
CALLED BY: <none>
```

```
(MACRO
  [LAMBDA (X)
    (GETPROP X (QUOTE MACRO])
```

AQINTERLISP Compiler Macros

```
MACRO: ADDSELECTOR
[LAMBDA (Complex Selector)

        (* Add a selector to a complex)
        (MERGE (LIST Selector)
               (APPEND Complex)
               (FUNCTION EARLIERVAR]
```

```
MACRO: APPENDX
[LAMBDA (L)
        (* Append together the top level elements of a list)
        (APPLY (FUNCTION APPEND)
               L]
```

```
MACRO: BESTN
[LAMBDA (LL N)

        (* Returns the first N elements of list LL. Actually modifies LL.)
        (COND ((ILEQ (FLENGTH LL)
                     N)
               LL)
              (T (RPLACD (FNTH LL N)
                         NIL)
                 LL]
```

```
MACRO: BUILDSELECTOR
[LAMBDA (Var Vals)
```

```
        (* Construct a selector)
        (CONS Var Vals]



MACRO: COVER
[LAMBDA (Class)
        (GETPROP Class (QUOTE COVER]



MACRO: COVEREDEVENTS
[LAMBDA (Class)
        (GETPROP Class (QUOTE COVEREDEVENTS]



MACRO: DOMAINTYPE
[LAMBDA (Var)
        (* Return the domain type of a variable)
        (GETPROP Var (QUOTE DOMAINTYPE]



MACRO: EPSILON
[LAMBDA (Var)
        (* Return the minimum difference allowed between linear selectors)
        (OR (GETPROP Var (QUOTE EPSILON))
            1]



MACRO: EVENTS
[LAMBDA (Name)
        (* Return the list of events associated with a class name)
        (GETPROP Name (QUOTE EVENTS]
```

```
MACRO: HIGH
[LAMBDA (X)

        (* Return the upper limit of the range of values in a disjunctive list
           of values of a linear selector)
        (CDAR X]



MACRO: LOW
[LAMBDA (X)

        (* Return the lower limit of the range of values in a disjunctive list
           of values of a linear selector)
        (CAAR X]



MACRO: MASK
[LAMBDA (VAR)

        (* Returns the mask corresponding to the nominal or structured variable
           VAR)
        (GETPROP VAR (QUOTE MASK]



MACRO: MASKEQUAL
[LAMBDA (X Y)

        (* Returns T if two variable masks are equal)
        (COND ((FIXP X)
               (EQP X Y))
              (T (for A in X as B in Y always (EQP A B]



MACRO: MASKINCLUDESP
[LAMBDA (Outer Inner)

        (* Determine if bit string Outer includes all bit in bit string Inner)
        (COND ((FIXP Outer)
               (EQP (LOGAND Outer Inner)
                    Inner))
```

```
            (T (for OutV in Outer as InV in Inner always
                   (EQP (LOGAND OutV InV)
                        InV]


MACRO: MASKLLSH
[LAMBDA (X N)

        (* Left-shift a bit string N places)
        (COND ((FIXP X)
               (LLSH X N))
              (T (PROG (CarryIn CarryOut)
                       (for I from 1 to N do (SETQ CarryIn 0)
                            (for Ptr on X do (SETQ
                                   CarryOut
                                   (COND ((ZEROP (LOGAND (CAR Ptr)
                                                         HighBit))
                                          0)
                                         (T 1)))
                                 (RPLACA Ptr (LOGOR CarryIn (LLSH (CAR Ptr)
                                                                  1)))
                                 (SETQ CarryIn CarryOut)))
                       (RETURN X]


MACRO: MASKLRSH
[LAMBDA (X N)

        (* Right-shift a bit string N places)
        (COND ((FIXP X)
               (LRSH X N))
              (T (PROG (Carry)
                       [for I from 1 to N do
                            (for Ptr on X do
                                 (SETQ Carry (COND
                                        ((ZEROP (LOGAND (OR (CADR Ptr)
                                                            0)
                                                        LowBit))
                                         0)
                                        (T HighBit)))
                                 (RPLACA Ptr (LOGOR Carry (LRSH (CAR Ptr)
                                                                1]
                       (RETURN X]
```

```
MACRO: MASKLOGAND
[LAMBDA (X Y)

        (COND ((FIXP X Y)
               (LOGAND X Y))
              (T (for A in X as B in Y collect (LOGAND A B]



MACRO: MASKLOGOR
[LAMBDA (X Y)

        (COND ((FIXP X)
               (LOGOR X Y))
              (T (for A in X as B in Y collect (LOGOR A B]



MACRO: MASKLOGXOR
[LAMBDA (X Y)

        (COND ((FIXP X)
               (LOGXOR X Y))
              (T (for A in X as B in Y collect (LOGXOR A B]



MACRO: DMASKERASE
[LAMBDA (X Y)

        (COND ((FIXP X)
               (LOGXOR X (LOGAND X Y)))
              (T [for A on X as B on Y do (RPLACA A (LOGXOR (CAR A)
                                                            (LOGAND (CAR A)
                                                                    (CAR B]
                 X]



MACRO: DMASKLOGOR
[LAMBDA (X Y)

        (COND ((FIXP X)
               (LOGOR X Y))
```

```
            (T [for A on X as B on Y do (RPLACA A (LOGOR (CAR A)
                                                         (CAR B]
               X]



MACRO: DMASKLOGAND
[LAMBDA (X Y)

        (COND ((FIXP X Y)
               (LOGAND X Y))
              (T [for A on X as B on Y do (RPLACA A (LOGAND (CAR A)
                                                            (CAR B]
                 X]



MACRO: DMASKLOGXOR
[LAMBDA (X Y)

        (COND ((FIXP X)
               (LOGXOR X Y))
              (T [for A on X as B on Y do (RPLACA A (LOGXOR (CAR A)
                                                            (CAR B]
                 X]



MACRO: MASKNEGATE
[LAMBDA (Mask)

        (COND ((FIXP Mask)
               (LOGXOR Mask AllOnes))
              (T (for X in Mask collect (LOGXOR X AllOnes]



MACRO: MASKZEROP
[LAMBDA (X)
        (COND ((FIXP X)
               (ZEROP X))
              (T (for A in X always (ZEROP A]
```

```
MACRO: MASKZEROPLOGAND
[LAMBDA (X Y)

        (COND ((FIXP X)
               (ZEROP (LOGAND X Y)))
              (T (for A in X as B in Y always (ZEROP (LOGAND A B]



MACRO: MAXBOUND
[LAMBDA (Var)

        (* Return the upper limit of the range of a linear variable domain)
        (GETPROP Var (QUOTE HIGH]



MACRO: MINBOUND
[LAMBDA (Var)

        (* Return the lower limit of the range of a linear variable domain)
        (GETPROP Var (QUOTE LOW]



MACRO: NEWSELECTOR
[LAMBDA (Complex Selector)

        (MERGE (LIST Selector)
               Complex
               (FUNCTION EARLIERVAR]



MACRO: NON-TRIVIAL
[LAMBDA (X)
        T]



MACRO: OLDEVENT
[LAMBDA (Class)
```

```
        (GETPROP Class (QUOTE OLDEVENT]



MACRO: ONEMASK
[LAMBDA (Var)

        (COND ((FIXP (MASK Var))
               LowBit)
              (T (CONS LowBit (for A on (CDR (MASK Var))
                                 collect AllZeros]



MACRO: PARENTMASKS
[LAMBDA (Var)
        (* Return the list of parentmasks associated with Structured variable
           Var)
        (GETPROP Var (QUOTE PARENTMASKS]



MACRO: RELATION
[LAMBDA (X)
        (* Return the relation <= or #> used in a linear selector)
        (CADR X]



MACRO: REPLACESELECTOR
[LAMBDA (NewSel OldSel Complex)

        (* Replace OldSel by NewSel in Complex. Returns a copy of "Complex"
           with appropriate changes.)
        (SUBST NewSel OldSel Complex]



MACRO: SEEDLIST
[LAMBDA (Class)
        (GETPROP Class (QUOTE SEEDLIST]
```

```
MACRO: SETCOVER
[LAMBDA (Class Value)
        (PUTPROP Class (QUOTE COVER)
                 Value]
```

```
MACRO: SETCOVEREDEVENTS
[LAMBDA (Class Value)
        (PUTPROP Class (QUOTE COVEREDEVENTS)
                 Value]
```

```
MACRO: SETEVENTS
[LAMBDA (Name Val)
        (PUTPROP Name (QUOTE EVENTS)
                 Val]
```

```
MACRO: SETOLDEVENT
[LAMBDA (Class Value)
        (PUTPROP Class (QUOTE OLDEVENT)
                 Value]
```

```
MACRO: SETSEEDLIST
[LAMBDA (Class Value)
        (PUTPROP Class (QUOTE SEEDLIST)
                 Value]
```

```
MACRO: SETUNCOVEREDEVENTS
[LAMBDA (Class Value)
        (PUTPROP Class (QUOTE UNCOVEREDEVENTS)
                 Value]
```

```
MACRO: SIBLINGS
[LAMBDA (Var)
        (* Return the SIBLINGS list associated with structured variable VAR)
        (GETPROP Var (QUOTE SIBLINGS]



MACRO: UNCOVEREDEVENTS
[LAMBDA (Class)
        (GETPROP Class (QUOTE UNCOVEREDEVENTS]



MACRO: VALS
[LAMBDA (Sel)

        (CDR Sel]



MACRO: VALUESET
[LAMBDA (Var)

        (GETPROP Var (QUOTE VALUESET]



MACRO: VAR
[LAMBDA (X)
        (* Returns the variable from a linear selector)
        (CAR X]



MACRO: ZEROMASK
[LAMBDA (Var)

        (COND ((FIXP (MASK Var))
               AllZeros)
              (T (for A on (MASK Var)
                      collect AllZeros]
```

INITIAL VALUES OF VARIABLES

```
VARIABLE: JMB$AQLISPCOMS
VALUE:    [(FNS RUN AQVAL ENTERDATA GETDATA LISTDATA SAVEDATA EDITDATA
                ENTERORDER ENTERVARIABLES&DOMAINS ENTEREVENTS&CLASSES
                ENTEREVENT AQHELP READMAXSTAR&CUTSTAR READLIST READNAME READSX
                WAIT-FOR-KEYPRESS INITIALIZE REENTRANT-AQ-INIT RESETCLASSES
                SETUPVBLS&DOMAINS SETUPNOMINAL SETUPLINEAR SETUPLARGELINEARVALS
                SETUPSTRUCTURED SETUPSIBLINGS SETUPLEAVES SETUPPARENTMASKS
                SETUPPARENTMASK SETUPMASK WORDMASK SETUPSELECTOR BUILDNOMINAL
                BUILDLINEAR BUILDSTRUCTURED SETUPEVENTS&CLASSES
                INTERSECTINGCOVERS DISJOINTCOVERS DISJOINTCOMPLEXCOVER
                ORDEREDCOVER AQ AQ-SINGLE AQ-MULTI AQ-MR BESTCOMP #COVERED
                NUMBEROFSELECTORS COVEREDBYCOMPLEX KNOCKOUT EARLIERVAR
                EXTENDAGAINST EXTENDAGAINSTVALS EXTENDAGAINSTNOMINAL
                EXTENDAGAINSTLINEAR EXTENDAGAINSTSTRUCTURE INCLUDES
                INCLUDESVALS MULTIPLY ABSORBP PRODUCTSC VALSPRODUCT STAR
                TRUNCATE ABSORB CUTSTAR REFUNION VALSUNION NEGATECOMPLEX
                NEGATEVALS SHOWCOVERS SHOWCOVER PPCOMPS PPCOMP PRINTNOMINAL
                PRINTLINEAR PRINTSTRUCTURE PRINLIST MASKONBITS ONBITS POSN
                PRINBITS PRINMASK PP-AQMACROS REDO-AQMACROS UNDO-AQMACROS MACRO)
          (MACROS ADDSELECTOR APPENDX BESTN BUILDSELECTOR COVER COVEREDEVENTS
                  DOMAINTYPE EPSILON EVENTS HIGH LOW MASK MASKEQUAL
                  MASKINCLUDESP MASKLLSH MASKLRSH MASKLOGAND MASKLOGOR
                  MASKLOGXOR DMASKERASE DMASKLOGOR DMASKLOGAND DMASKLOGXOR
                  MASKNEGATE MASKZEROP MASKZEROPLOGAND MAXBOUND MINBOUND
                  NEWSELECTOR NON-TRIVIAL OLDEVENT ONEMASK PARENTMASKS
                  RELATION REPLACESELECTOR SEEDLIST SETCOVER SETCOVEREDEVENTS
                  SETEVENTS SETOLDEVENT SETSEEDLIST SETUNCOVEREDEVENTS
                  SIBLINGS UNCOVEREDEVENTS VALS VALUESET VAR ZEROMASK)
          (VARS AllOnes AllZeros ClassNames Criteria&ToleranceList CutStar
                ExtendStrucMode HighBit IOTimerFlag LargeLinearThreshold
                LowBit MaxStar MultiFlag NORMALCOMMENTSFLG WordSize)
          (GLOBALVARS AllOnes AllVars AllZeros ClassNames
                      Criteria&ToleranceList CutStar ExtendStrucMode HighBit
                      IOTimerFlag LargeLinearThreshold LowBit MaxStar
                      MultiFlag VL1READTBL WordSize)
          [P (RPAQ VL1READTBL (COPYREADTABLE (QUOTE ORIG)))
             (SETSYNTAX (QUOTE ,)
                        (QUOTE SEPRCHAR)
                        VL1READTBL)
             (SETSYNTAX (QUOTE v)
                        (QUOTE SEPRCHAR)
                        VL1READTBL)
             (RPAQ HighBit (LLSH 1 (SUB1 WordSize]
          (DECLARE: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS
                    (ADDVARS (NLAMA)
                             (NLAML ENTERDATA)
                             (LAMA]
```

```
VARIABLE: AllOnes
VALUE:    -1

VARIABLE: AllZeros
VALUE:    0

VARIABLE: ClassNames
VALUE:    (I II)

VARIABLE: Criteria&ToleranceList
VALUE:    ((#COVERED 0)
           (NUMBEROFSELECTORS 0))

VARIABLE: CutStar
VALUE:    10

VARIABLE: ExtendStrucMode
VALUE:    MAXIMAL

VARIABLE: HighBit
VALUE:    -2147483648

VARIABLE: IOTimerFlag
VALUE:    NIL

VARIABLE: LargeLinearThreshold
VALUE:    32

VARIABLE: LowBit
VALUE:    1

VARIABLE: MaxStar
VALUE:    25

VARIABLE: MultiFlag
VALUE:    T

VARIABLE: NORMALCOMMENTSFLG
VALUE:    T
```

VARIABLE: WordSize
VALUE: 32