AGE Reference Manual

AGE-1

31 October 1981

Nelleke Aiello
Conrad Bock
H. Penny Nii
William C. White

Copyright © 1981 The Board of Trustees of the Leland Stanford Junior University. All Rights Reserved.

Heuristic Programming Project
Computer Science Department
Stanford University
Stanford, Ca 94305
Table of Contents

1. Introduction ............................................................................................................. 1

2. The AGE User Program .............................................................................................. 3
   2.1. User Programs in the Forward Driven Framework ........................................ 3
       2.1.1. Hypothesis Structure (The Blackboard) ........................................... 4
       2.1.2. Knowledge Sources ........................................................................... 6
       2.1.3. Control Information ........................................................................... 15
       2.1.4. Input Data ........................................................................................... 20
       2.1.5. The UNITS Package ........................................................................... 20
   2.2. User Programs in the Backchain Framework .................................................. 24
       2.2.1. Structure of Chaining Data ................................................................ 24
       2.2.2. Knowledge Sources .......................................................................... 26
       2.2.3. Control Information ........................................................................... 29
       2.2.4. Input Data ........................................................................................... 29

3. User Program Execution ............................................................................................. 31
   3.1. Forward Driven Execution .............................................................................. 31
   3.2. Backward Chained Execution .......................................................................... 37

4. The AGE System ........................................................................................................ 41
   4.1. System Initialization ...................................................................................... 41
   4.2. The AGE Executive ...................................................................................... 41
       4.2.1. AGE Tasks ......................................................................................... 42
       4.2.2. Hacker Mode Tasks ............................................................................ 43

5. User Program Design and Specification ................................................................... 45

6. User Program Syntax Checking ............................................................................... 51

7. Functions and Variables ............................................................................................ 53
   7.1. Auxiliary Functions ...................................................................................... 53
   7.2. Output Functions ......................................................................................... 59
   7.3. User Prompting Functions ............................................................................ 63
   7.4. Miscellaneous Functions .............................................................................. 75
   7.5. Backchain Utility Functions ........................................................................ 80
   7.6. Task Execution Functions ............................................................................ 81
   7.7. Global Variables .......................................................................................... 86

Appendix I. Glossary ..................................................................................................... 101

Appendix II. Reserved Function Names ...................................................................... 109

Appendix III. Reserved Global Variable Names ........................................................... 115

Index ............................................................................................................................. 119
1. Introduction

AGE is an information processing system whose purpose is to support the design, construction, and execution of knowledge based programs.

The AGE system runs in INTERLISP under the TENEX or TOPS-20 operating system on DEC PDP-10 type computer systems. AGE exists in the form of a system image which may be invoked directly from the operating system. To enter AGE, a user simply types the name of the system image file and answers a few questions to initialize the system.

The user's interaction with AGE is then controlled by the AGE executive program (refer to Section 4.2), which allows the user to:

- specify a definition of the user program and the necessary domain knowledge;
- check the syntactic correctness of the user program thus defined;
- save the program definition; and,
- run the program.

AGE contains comprehensive debugging aids that include dynamic tracing, execution coreak, and trace-back explanation (also see Joy of AGE-ing).

The AGE system provides several different frameworks within which a user system may be constructed. A framework is a distinct conceptual skeleton around which an AGE user system is defined. Each AGE framework may share some facilities with other frameworks, but one or more facilities of one framework are unique to that framework and generally are incompatible with the unique facilities of other frameworks. A framework is conceptually similar to a prefabricated house -- the basic design decisions have been made, but many of the final decisions have been left to suit the occupants' tastes and needs.

There are two frameworks currently supported by AGE:

- the Forward driven framework, and
- the Backchain framework.

The Forward driven framework, is useful primarily for building programs whose designs are rooted in the Blackboard Model [Lesser 77]. The Blackboard Model uses the concepts of:

- a globally accessible data structure called a blackboard, and
- independent knowledge sources that cooperate to incrementally form hypotheses on the blackboard.

In AGE, the implementatio : of the Blackboard Model differs from the original by providing for a variety of different control and representational forms. Some of the differences include:
• production rule and object centered representations of knowledge,

• extended blackboard to include control related data, and

• a scheme for generating and processing expectations and goals.

The Backchain framework, is useful for building a program that uses production rules as its primary knowledge representation, and goal directed backward chaining of rules as its inference engine. Using this framework the user can build a program whose behavior is similar to programs written in EMYCIN -- a predefined skeletal program (see EMYCIN User’s Manual). However, in contrast to EMYCIN, the user of the AGE Backchain framework must provide the rule predicates, specify the structure of the objects that are to be chained, define the computation to be applied to the certainty factors, and so on (see Section 2.2).
2. The AGE User Program

An AGE user program represents the user's knowledge and model of the task domain. This program will be interpreted by the AGE system to produce whatever results are sought by the user. Every program must contain one or more sets of production rules called knowledge sources (KSs). Additional knowledge may be contained in the user program in the form of:

- control information defining the way KSs and their rules are to be interpreted;
- input data definitions defining information to be supplied by a user during program execution;
- a hierarchically organized global data base, or blackboard;
- a semantic network (using the UNITS package); and,
- LISP functions and variables used by the program. AGE contains many predefined functions and variables designed to be used within the user program. In addition to these, a user program may contain any additional definitions specified by the user.

As mentioned earlier, AGE currently provides two frameworks for building a user program: the Forward Driven framework and the Backchain framework.

2.1. User Programs in the Forward Driven Framework

The Forward Driven framework can be described as follows:

- There are diverse Knowledge Sources (KSs) which are kept separate and independent;
- There is a global data base, the blackboard, that is used as a means of communication and interaction among the KSs; and
- The KSs respond to changes in the blackboard.

In AGE the user program in the Forward Driven framework consists of three major components. These components are:
• The blackboard: The blackboard is designed to hold input data, intermediate results and solutions. It is a hierarchical data structure that is organized to represent the problem domain as a hierarchy of analysis levels. In addition, the blackboard holds dynamic control related data that are accessible to the user program. Knowledge about the problem domain can also be represented in an object centered representation as implemented by the Units Knowledge Base. In this representation, the description of the objects, both conceptual and actual, and the relationships among them, are maintained in a semantic network.

• The knowledge sources: The knowledge that uses information on the blackboard to perform a specific task is represented in production rules. A set of rules that belong together is called a knowledge source.

• The control information: The selection and activation of the knowledge sources, and the selection of items on the blackboard for focus of attention, are designed to be independent of the domain knowledge. By specifying different control mechanisms, programs can be made to display different behavior even with the same knowledge base.

The combined process of KS selection and incremental changes to the blackboard is viewed as a general process of hypothesis formation. Consequently, the data structure that holds the input data and the intermediate, as well as the final, results is referred to as the hypothesis structure. In addition, control information on the blackboard can be accessed by the KSs, and KSs can be built that act as control or meta-knowledge sources.

2.1.1. Hypothesis Structure (The Blackboard)

Overview

The hypothesis structure in a forward driven program is assumed to be hierarchically organized. The hierarchy may be "flat", consisting, in the extreme case, of input data and a single inference level drawn from the data. Depending on the problem, there may be more than one hierarchy needed to define the hypothesis structure. Each hierarchy consists of hypothesis elements integrated by links that represent support from above and support from below; the links are called expectation links (or model-derived links) and reduction links, respectively.

An hypothesis element is a named node (object) in the hypothesis structure that represents an aggregation (summary, interpretation, integration, abstraction, etc.) of lower level hypothesis elements. The properties of the elements are represented in the form of attribute-value pairs that are meaningful at the particular hypothesis level.

A link in the hypothesis structure represents a relationship between the elements of the hypothesis. Since the links are created by the actions of the rules, they indirectly represent the rule(s) that created them. A link can point to a hypothesis element from any other element, including one from its own level. Thus, the user can structure the hypothesis as a general directed graph, as a simple linear sequence, or more usually, as a strict hierarchy. An hypothesis structure appropriate for the problem at hand is specified by the user during the design phase.
The actual solution hypothesis is built incrementally by rules that add or modify the hypothesis elements, or relationships between the elements. Hypothesis formation can be thought of as a process whereby the rules:

1. Interpret data (support from below);
2. Specialize or instantiate a more inclusive hypothesis element (support from above); or
3. Generate expectations that must be verified by data. These expectations can be generated based on:
   a. A model (theoretical support), or
   b. A higher level hypothesis element.

**Definition**

The hypothesis structure consists of the values of a number of global variables: plus each of the atoms in the list which is the value of the global variable NODELEVELS; with the atom’s value and its associated property list of attribute-value pairs.

The variables which contain the hypothesis information are:

- NODELEVELS
- LEVELLIST
- DEFINEDSLOTS
- DEFINEDLINKS
- LINKS&OPPOSITES

**NODELEVELS** - The value of NODELEVELS: = (level) ...

This is a list of the names of each of the hypothesis levels, where (car NODELEVELS) is the highest or top level.

The property list of (level): =

(NUM (availnum)
 LLIST (level) [ (node) ] ...)
 { (attribute) (attrvalue) }
 { (linkname) (linkvalue) } ...)

where:

- (availnum): = a positive integer used to generate unique node names at the given level.
- (node): = a node name; an element of the hypothesis. This is either the name of a pre-created element or a name generated from the (level) name concatenated with an integer determined by (availnum). Initially, before running the user program, only pre-created nodes appear in this list.
- (attribute): = an attribute name. One of the names in DEFINEDSLOTS.
2.1.1 Hypothesis Structure (The Blackboard)

\[ \text{<attrvalue>} := (\{ \text{<atval>} \} ...) \]

where:

\[ \text{<atval>} \text{ is a value of the attribute.} \]

\[ \text{<linkname>} := \text{a link name. One of the names in} \]
\[ \text{DEFINEDLINKS.} \]

\[ \text{<linkvalue>} := (\{ \text{<node>} \} ...) \]

Initially \text{<attrvalue>} is NIL and there are no links associated with any nodes. These may be added to the hypothesis structure by the execution of the user's program.

The property list of \text{<node>} is the same as that of \text{<level>} except that it does not contain either NUM or LLIST properties.

\[ \text{LEVELLIST} - \text{The value of LEVELLIST is a list of the level names of all the hypothesis levels currently used by the user program.} \]

\[ \text{DEFINEDSLOTS} - \text{The value of DEFINEDSLOTS is a list of all the names that will be used as attributes by the user's program.} \]

\[ \text{DEFINEDLINKS} - \text{The value of DEFINEDLINKS is a list of all names that will be used as links by the user's program.} \]

\[ \text{LINKS&OPPOSITES} - \text{The value of LINKS&OPPOSITES:= (\{ \text{<link>} . \text{<link>} \}) ...} \]

\[ \text{a list of all pairs of links which have opposites and are therefore 2-way links.} \]

2.1.2 Knowledge Sources
Overview

Knowledge necessary to accomplish the goals of the program is represented in production rules. These rules are organized into one or more sets of rules called Knowledge Sources (KSs). The user program may consist of one KS that contains a set of homogeneous rules. Or, the user program may contain many KSs that are used as hypothesis generators and hypothesis validators [Nii 78].

A KS is a labeled set of production rules that are a priori deemed to belong together. For example, a model-like KS may contain rules that are organized around some objects or concepts; a data-oriented KS may contain rules that generate hypotheses from data elements.

A KS can be thought of as a mega-chunk of knowledge or, because of its organization, as a mega-rule. Each has associated with it preconditions for its invocation that indicates the specific situations under which the rules in the KS are applicable. A KS is invoked when one of its preconditions matches the currently focused event which represents the situation. An event is a user-specified token that summarizes the actions taken by the rules. For example, a rule may modify an attribute $x$ in some hypothesis element. It will generate a user-specified token, $x$-modified, which becomes an event name. A KS that knows what to do with $x$ will have in its preconditions, $x$-modified -- the KS will be invoked after $x$ is changed. A KS can thus be dynamically chained in different order, depending on the events generated by the firing of different rules.

Local context applicable to the rules within the KS can be established before the rules are actually executed. A context includes a strategy for rule evaluation for the KS (single hit, multiple hit, once only), links that are allowed to be generated by the rules in the KS, and binding of local variables.

The rules in a KS can access and use control related data. Such a KS can manipulate other KSs, and it can be thought of as a part of the control structure. These KSs can contain rules that know about problem-solving strategies for the specific application at hand, manipulation of other knowledge sources, and other control heuristics.

Rules

Each production rule consists of a left hand side (LHS) and a right hand side (RHS). The LHS specifies a set of conditions or patterns for the applicability of the rule. The term applicability can mean that all of the specified conditions must be true, or only some need be true, in order to fire a rule (i.e. execute the RHS). Because of the wide range of possibilities available for the definition of applicability, AGE asks the user to define applicability in the form of a function to serve as the LHS Evaluator. Some simple LHS Evaluators, such as, $\text{AND (all-conditions-must-be-true)}$, are provided; the user can use one of them rather than programming his own.

The RHS is the action taken by the rule, and represents the implication to be drawn under the situation specified in the LHS. These implications are represented in the form of changes that are to be made to the hypothesis structure or to the Units Knowledge Base. Currently the changes that the RHS can make are limited to the following:

- actual changes to the hypothesis elements, attributes of the elements, links, or units. These changes are called events and are posted on the Event list on the blackboard, after the changes have been made. As mentioned earlier, the event
list is a part of the control related data on the blackboard that are available to the KSs.

- expected changes, either to the hypothesis structure or units. These are be posted on the Expectation list. No actual changes are made until the expected changes occur. The expectation list is also available to the KSs as a part of the control information.

- desired changes to the hypothesis structure or units. These are posted on the Goal list. No actual changes are made until the goals have been achieved. The goal list is also available to the KSs.

A change inferred by the RHS of a rule can have associated with it as its value, a probability, or some informal weight, that reflects the confidence in the implication to be drawn under the particular conditions specified in the LHS. This informal probability will need to be reflected in the attribute-value pair of the affected hypothesis element when the rule is executed. The a priori specified weights in the rules, and the probability already assigned to the attribute of the hypothesis element to be modified need to be integrated. This can be accomplished in many ways -- one possible method is described in EMYCIN and another in KAS. To allow maximum flexibility, AGE asks the user to provide the necessary computation in the form of a function to perform value adjustment.

Definition

The complete syntax and semantics of the KSs and the rules are detailed below.

The names of the Knowledge Sources are contained in KSLIST.

The value of KSLIST is a list of the names of the knowledge sources which exist. Each name is a global variable whose value is the knowledge source.

......

<knowledge-source> := (<preconditions> <inference-levels> <links> <hit-strategy> <variable-bindings> . <rulelist>)

A set of rules and information about their applicability and execution context.

<preconditions> := ( { <event-type> } ... )

A list, possibly empty, of event types. The event types specified in the precondition define the applicability of the set of rules in the KS. The KS is not invoked unless the currently focussed step contains one of the event types specified in this list. Every precondition must be the event type mentioned in a rule in some KS or the atom T. The atom T is a special precondition, which, when included in the KS, will cause that KS to be invoked whenever there are no others.
selected. In the event that multiple KS's are selected by a
given event type, the KS’s will be invoked in the order of
their occurrence in KSLIST.

\[\text{<inference - levels> := ( ( <level/node> , <level/node> ) ) ... }\]

where:

\[\text{<level/node> := <level> | <node> | DATA}\]

A list, possibly empty, of the pairs of names of the hypothesis
levels or hypothesis nodes that are referred to in this
KS. AGE checks to insure that no other levels or nodes are
referenced during the evaluation of the rules. DATA is an
acceptable hypothesis level name, as are the names of nodes
defined at the time the user program is specified.

\[\text{<links> := ( ( <link> , <link> ) ) ... }\]

These are the relationships between the hypothesis elements
that the rules in the KS will either create or modify. When a
link is specified by a rule, the program automatically infers
any associated opposite link. These are monitored in the same
way as levels during rule evaluation.

\[\text{<hit - strategy> := ( <single - or - multiple - hit> } \{ \text{ONCEONLY } \})\]

where:

\[\text{<single - or - multiple - hit> := SINGLE | MULTIPLE}\]

SINGLE or MULTIPLE determines whether the KS should be
scanned for more than one rule whose conditions are satisfied.
When SINGLE, the execution of the rules in the KS will stop
after the first rule is fired. When MULTIPLE, the KS
context will be preserved until all relevant rules have been fired.
ONCEONLY refers to whether each of the rules of the KS
should be executed only once during the running of the user's
system, or whether they may be executed every time the KS is
invoked and the LHS is true. In a ONCEONLY KS, the
LHS of a rule may be evaluated many times until its value is
true. Then the <action>s in the rule are executed once and the
rule is never evaluated again.

\[\text{<variable - bindings> := ( ( <variable> ) ) ... }\]

where:

\[\text{<variable> := <variable - name> | ( <variable - name>}\]
A list of variable bindings to be in effect during the execution of the rules in the KS. Like LISP PROG variables, these are limited in scope to the execution of the KS. Unlike the variables in a PROG, as each variable is set, its value is available in the following binding expressions. In order to avoid recomputation of similar conditions in the rules, variables can be set to the computed value and the variables used in the rules. The user may also specify values for system variables so as to provide values for those variables which are local to a single KS. In particular, the user may define values of \#VALUEADJUSTER and \#LHSEVALUATOR which apply only to a single KS.

\texttt{<rulelist> := (<rule> ...)}

A list of rules.

\texttt{<rule> := (<rule #> <lhs> <action> ...)}

\texttt{<rule #> := an integer number identifying the rule.}

\texttt{<lhs> := (<condition> ...)}

A list of the conditions which are evaluated by the user specified LHS evaluator function. The value of the LHS evaluator function must be true in order for the actions in the rule to be executed. Each condition is a LISP expression to be evaluated.

\texttt{<action> := (<action - type> . <action - tail>)}

The \texttt{<action - type>} determines the form and meaning of the remaining \texttt{<action>}. The types of \texttt{<action>}s defined are :

\texttt{<iteration> | <expectation> | <goal> | <proposition> | <execution>}

\texttt{<action - type> := LOOP | EXPECT | ACHIEVE | PROPOSE | EXECUTE}

The type of \texttt{<action>} to be performed. The form of \texttt{<action - tail>} is determined by the \texttt{<action - type>}. 

\texttt{October 10 1981}
<iteration> := (LOOP <variable> <values> <action> ...)

<variable> := a variable name. The iteration variable which will receive each of the values in the iteration list.

<values> := the iteration list. This is a LISP expression which will be evaluated to produce a list of values. Each of the values in the list will be assigned to the iteration variable and the following <action>s evaluated in that context.

......

<expectation> := (EXPECT { <key> <value> } ...)

where:

<key> := function | expr | actions | comment

The <value> associated with each of the allowable keys is as follows:

function - A function: either a function name or function expression. This is optional, and so may be omitted. The function will be used in conjunction with the expr of the expression to construct the expression which will be used by the expectation <matcher> to determine if the expectation has been met.

expr - An expression which will be evaluated to create the expectation. The value of the expression will be used in conjunction with the expectation function to construct the expression to be used by the expectation <matcher>. The <matcher> expression will normally consist of (function . expr); however, when no function is specified in the expectation, the value of the expectation expression alone will be used as the <matcher> expression.

actions - An <action> or list of <action>s to be performed when the expectation is met. When the value of the <matcher> is true, the <action>s are executed.

comment - Any text description provided by the user.

......

<goal> := (ACHIEVE { <key> <value> } ...)

where:

<key> := function | expr | actions | comment
The \textit{value} associated with each of the allowable keys is as follows:

\textbf{function} - A function: either a function name or function expression. This is optional, and so may be omitted. The \textit{function} will be used in conjunction with the \textit{expr} of the goal to construct the expression which will be used by the \textit{goal} \textit{matcher} to determine if the goal has been met.

\textbf{expr} - An expression which will be evaluated to create the goal. The value of the expression will be used in conjunction with the \textit{goal function} to construct the expression to be used by the \textit{goal} \textit{matcher}. The \textit{matcher} expression will normally consist of \textit{(function . expr)}; however, when no \textit{function} is specified in the goal, the the value of the \textit{goal expression} alone will be used as the \textit{matcher} expression.

\textbf{actions} - An \textit{action} or list of \textit{actions} to be performed when the goal is met. When the value of the \textit{matcher} is true, the \textit{actions} are executed. When the goal has not been met, the \textit{goal seekingmethod} is applied in an attempt to achieve the goal. When the value of the \textit{seekingmethod} is true, the \textit{actions} are executed.

\textbf{comment} - Any text description provided by the user.

\begin{verbatim}

\textbf{proposition} := (PROPOSE \{ \textit{key} \textit{value} \} ...)

where:

\textit{key} := ev.type | ch.type | level | hypo-element | attr-value | link-node | unit | slot | parent | relation | progeny? | token | arglist | role | datatype | slotvalue | keepinprogeny? | support | comment

The value associated with each of the allowable keys is as follows:

\textbf{ev.type} - An atom which is the name of this type of event. This \textit{event type} name is used as a \textit{precondition} in KS's to focus attention on what to do next. The atom \textit{T} is a special precondition and may not be used as an event type.

\textbf{ch.type} - \texttt{ADD} | \texttt{MODIFY} | \texttt{SUPERSEDE} | \texttt{NULL} | \texttt{ADDUNIT} | \texttt{DELETEUNIT} | \texttt{MSGUNIT} | \texttt{ADDSLOT} | \texttt{CHANGESLOT} | \texttt{DELETESLOT} | \texttt{MSGSLOT}

\end{verbatim}
The name of the type of change to be made by this proposition.

ADD creates a new node in the hypothesis; MODIFY changes attribute-values and/or links to an existing node of the hypothesis; SUPERSEDE replaces the value of an attribute of a node in the hypothesis with a new value; and NULL specifies no change. The remaining types cause changes in the UNITS knowledge base.

The actual <key> <value> pairs which occur in any particular proposition are determined by the ch.type of the proposition. The standard keys associated with all propositions are: ev.type, ch.type, support, comment. The distinctive keys associated with each change type are:

ADD - level { attr-value } { link-node }
MODIFY - hypo-element { attr-value } { link-node }
SUPERSEDE - hypo-element { attr-value } { link-node }
NULL - hypo-element
ADDUNIT - unit parent relation
DELETEUNIT - unit progeny?
MSGUNIT - unit token arglist
ADDSLOT - unit slot role datatype
CHANGESLOT - unit slot value
DELETE SLOT - unit slot keepinprogeny?
MSGSLOT - unit slot token arglist

level - The name of a hypothesis level. When the ch.type value is ADD, a new hypothesis node will be added to the given level.

hypo-element - An expression designating a hypothesis element. The expression will be evaluated to determine the name of the element.

attr-value - (<attribute> <value>) | { (<attribute> <value>) } ...

where:

<attribute> ::= An atom which is the name of an attribute in the hypothesis.

<value> ::= An expression which will be evaluated to give the value of the attribute.

link-node - (<link> <element>) | { (<link> <element>) } ...

where:
\(\text{link} :=\) An atom which is the name of a defined link. This identifies the given link from the node specified in the proposition to the node specified by \(\text{element}\). The inverse link, if any, will be inferred automatically by the program.

\(\text{element} :=\) An expression which will be evaluated to give the name of a hypothesis node.

\text{unit} - the name of a unit in the UNITS knowledge base.

\text{slot} - the name of a slot in the UNITS knowledge base.

\text{parent, relation} - refer to the UNITS function MAKEUNIT.

\text{progeny?} - refer to the UNITS function DELETEUNIT.

\text{token, arglist} - refer to the UNITS functions UNITMSG and SLOTMSG.

\text{role, datatype} - refer to the UNITS function MAKESLOT.

\(\text{slot-value} := (\text{slot} \text{slotval}) | ((\text{slot} \text{slotval}) ...)

\text{where:}

\(\text{slot} :=\) The name of a slot in the UNITS knowledge base.

\(\text{slotval} :=\) An expression which will be evaluated to give the value of the slot.

Refer to the UNITS function PUTVALUE.

\text{keepinprogeny?} - Refer to the UNITS function DELETESLOT.

\text{support} - The name of the knowledge source responsible for this proposition.

\text{comment} - Any text description provided by the user.

\(\text{execution} := (\text{EXECUTE} \{ \text{key} \text{value} \} ...)

\text{where:}

\(\text{key} := \) function | args | comment

The \(\text{value}\) associated with each of the allowable keys is as follows:
function - A function to be executed. When the action is to be performed, the function and args will be combined to produce an expression which will be evaluated for its effect. The function may be omitted, in which case the args alone will constitute the expression to be evaluated.

args - An expression which will be interpreted in one of two ways depending on the execution function. When a function is specified, the expression must be a list of the arguments to which the function will be applied. When no function is specified, the expression itself will be evaluated.

comment - Any text description provided by the user.

2.1.3. Control Information

Overview

The control information in AGE is for directing the invocation of the KSs, manipulating the rules in the KSs, and focusing the attention within the user program. There are several functional components grouped under the heading of Control. They are:

• Input component: The format and the names of the input data, as well as the manner in which the data is to be acquired, need to be specified by the user. For example, the input may come from a terminal or a file; the input may be acquired at the beginning of the program or only when a rule first needs the information.

• Initialization component: preprocessing of data, if necessary, and returning the name of the first KS to be invoked.

• Kernel component: focus of attention and inference generation. This component will be discussed in some detail below.

• Termination component: terminating the program. There is no uniform way in which forward driven programs terminate. The user must specify the condition under which the program is to terminate; for example, the occurrence of some specific event.

• Post-processing component: processing after the termination of rule executions; for example, printing the hypothesis or printing an explanation.

Kernel Control Information

The functions of the kernel control information are:

1. to select an item on the blackboard to process next, and
2. to invoke KSs appropriate to that item and consistent with the goal of the program.

Thus, the kernel control component consists of two conceptually distinct sub-components whose separate functions are focus of attention and inference generation:

- **Focus of Attention**: First, select a step type (an event, an expectation, or a goal) to process next. From the appropriate control information associated with the step type (Event list, Expectation list, or Goal list), choose a specific step; this has the affect of selecting a hypothesis element or a unit for focus of attention. Then, choose a relevant KS, a KS whose precondition matches the focused event. Finally, invoke that KS. KS selection may require complex processing associated with each step type; for example, for a goal step type, backward-chaining of rules might be an appropriate method for achieving a goal.

- **Inference generation**: Within the invoked KS the rules either (1) PROPOSE changes in the hypothesis elements or units, (2) indicate that some changes are EXPECTed to occur, or (3) indicate that the KS desires a particular value or state to be ACHIEVEd within the hypothesis structure. Each of these actions taken by the RHS is called a step.

Because the control components are full of details that are confusing to the novice users, AGE provides two rather simple, prepackaged control structures, called Control Macros. They are useful for event-driven and expectation-driven control.

**Event-driven Control Macro**: Event-driven hypothesis formation is characterized by incremental formation of hypothesis elements from evidence found in data, or in lower level hypothesis elements. The elements that are modified can be focussed for further processing, on the basis of first-in-first-out, first-in-last-out, or best-first. These generally correspond to breadth-first, depth-first, and best-first processing of the hypothesis space. When an element is chosen to be processed, it is termed focussed. The event name (a token assigned by the user) associated with the focussed element determines which KS is to be invoked next. Thus, event-driven processing can be summarized as a two-step process:

1. A rule modifies the hypothesis elements or units and causes an event, with associated event token.
2. If the focussed event name (a token) matches a precondition of a KS, then invoke that KS.
3. [loop back to 1]

**Expectation-driven Control Macro**: Within an expectation-driven system, states which are expected to occur within the hypothesis structure are generated by the rules. The expectation can be some expected input data, or some expected intermediate results. The expectation generation can be based on some previously generated hypothesis elements, or on models or schemata represented in the Units Knowledge Base or as production rules.

In order to determine if a specified expectation has been met, or can be met, the user must provide an Expectation Evaluation function. Independent of the user-provide Expectation Evaluation function, AGE will always monitor the incoming data and events to see if the expected situation has occurred. Thus, expectation-driven processing is a simple three-step process:
1. A rule generates expectation(s).
2. If an expectation is met, then the hypothesis elements or units are modified as specified. This action generates an event with an associated event token.
3. If the focussed event name (a token) matches a precondition of a KS, then that KS is invoked.
4. [loop back to 1]

The AGE Control Macros are simply collections of preset control variables and preprogrammed control functions. For example, both macros use $AND function for the LHS evaluator -- $AND is one of the AGE functions available to the user. It takes as its argument Lisp expressions and returns T, if all the expressions evaluates to non-NIL. Both the macros also use a common initialization function, $INITIALIZE, which is a null function that returns as its value the name of the first KS to be invoked (specified by the user during the design phase). If the macros do not perform exactly as the user wishes, unsatisfactory parts can be modified; for example replace the $INITIALIZE function with one more suitable to the problem at hand. In this manner, the user can evolve an arbitrarily complex control structure.

Definition

The complete definition of the various components of the control information in the AGE system follow:

The value of STEPTYPELIST is a list of the names of each of the valid step types defined in AGE. Execution of an AGE user program involves the sequential execution of inference generation steps. Each individual step may be any one of the defined step types.

Each step type name is a global variable which has a value which is the <control-info> for that particular step type. Each step type name is also defined as a function which is the step execution function for that step type.

Also, associated with each step type during user program execution are a step list and the current focussed step. The step list is a global variable, <steptypelist>, whose value is a list of any steps remaining to be processed for the individual step type. The current focussed step is a global variable, FOCUS<steptype>, whose value is the step currently being processed by the AGE system.

The names of the step types currently defined are:

EVENT - The value of EVENT is the current control information for events.

EXPECTATION - The value of EXPECTATION is the current control information for expectations.

GOAL - The value of GOAL is the current control information for
goals.

<control - info> : = ( <selectionmethod> <collectionrules> <matcher> <seekingmethod> )

A list identifying various control elements for each of the step types. For each of the recognized step types, this list will be found as the value of the step type name.

<selectionmethod> : = the name of a function which, when applied to a step list, will return one of the elements of the list.

<collectionrules> : = a list of event types whose events may be collected.

<matcher> : = the name of a predicate which can be applied to an expectation expression or goal expression to determine if the expectation or goal has been met. When the expectation or goal has been met, the predicate must return a non-NIL value.

<seekingmethod> : = the name of a function which can be applied to a goal step in an attempt to achieve the desired goal. When the function is successful in achieving its goal, it must return a non-NIL value.

EVENTLIST · A list of <step – entry>s which are the events remaining to be processed.

EXPECTATIONLIST · A list of <step – entry>s which are the expectations remaining to be processed.

GOALLIST · A list of <step – entry>s which are the goals remaining to be processed.

FOCUSEVENT · The value of FOCUSEVENT is a step from the event list. This is the current focused event during the execution of an event step.

FOCUSEXPECTATION · The value of FOCUSEXPECTATION is a step from the expectation list. This is the current focused expectation during the execution of an expectation step.

FOCUSGOAL · The value of FOCUSGOAL is a step from the goal list. This is the current focused goal during the execution of a goal step.

<step – entry> : = ( <type> <node> <hypochange> <support> <enum> <effect> )

a step entered in one of the step lists.
\langle \text{type}\rangle := \text{an event type.}

\langle \text{node}\rangle := \text{the name of a hypothesis node, or a unit in the UNITS knowledge base. The value of the hypo-element or unit property of the proposition creating this step.}

\langle \text{hypochange}\rangle := \text{The list of terms causing changes to the hypothesis or UNITS knowledge base.}

\langle \text{support}\rangle := \text{The name of some knowledge source. The KS which was being invoked when this step was created.}

\langle \text{enum}\rangle := \text{A unique name assigned to identify this step.}

\langle \text{effect}\rangle := \text{NIL | (expr <action> ..)}

the expression and \langle \text{action}\rangle s associated with expectations or goals.

\text{USERINITFN} - \text{The value of USERINITFN is a function name or function expression. This function will be evaluated at the start of the user program to determine the name or names of the user's initial knowledge source. The value must be either a single name or a list of names of knowledge sources. The knowledge sources named will be invoked to begin the user's program.}

\text{CONTROLRULES} - \text{The value of CONTROLRULES is a list of \langle \text{control-rule}\rangle s which will be used during program execution to select the next step type to be performed.}

\langle \text{control-rule}\rangle := (\langle \text{condition}\rangle \langle \text{steptype}\rangle)

A rule used to determine the focus of attention during user program execution.

\langle \text{condition}\rangle := \text{the condition for the selection of the rule. This may be any LISP expression which will be evaluated.}

\langle \text{steptype}\rangle := \text{the name of a step type.}

\text{# LHSEVALUATOR} - \text{The value of \# LHSEVALUATOR is a function name. This function will be applied to the LHS of a rule to determine if the rule should be executed.}

\text{# VALUEADJUSTER} - \text{The value of \# VALUEADJUSTER is a function name or function expression. The function will be evaluated to determine the adjusted value associated with a link or attribute. The}
function to be executed may use the global variables $NODE, $ATTR, and $VAL. If a function is not given, then no value adjustment is performed.

**TERMINATIONCONDITION** - The value of TERMINATIONCONDITION is a function name or function expression. The function designated will be evaluated and the value returned will be used to decide when to end execution of the user program. Program execution will terminate as soon as the function returns a non-NIL value.

**USERPOSTPROCESSINGFN** - The value of USERPOSTPROCESSINGFN is a function name or function expression. This function will be evaluated at the end of the execution of the user program.

### 2.1.4. Input Data

**INPUTDATANAMES** - All input data names are stored on the global variable INPUTDATANAMES. The user can define the names of input data at the time the user program is specified. In addition to specifying the names of data items, the user may also write prompts for any of those data items. These data items are calls to AGE input functions and are stored on the property list of the data item name under the property AGPROMPT. If a data item has no prompt the user will be asked for its value at the beginning of a run. For data items with prompts, the user specified prompt is evaluated to retrieve the data value the first time it is referenced during the program’s execution.

The function ASKFORDATA asks for input data values at the beginning of each run. It skips data items with their own prompts. $DATA is used in the rules to retrieve data values. $DATA first checks to see if a value is available and returns it if it is. If no value has been entered yet, the prompt is evaluated and the answer is saved on INPUTDATA. This list of data names and values can be saved on a file after it is created by ASKFORDATA.

**INPUTDATA** - The value of INPUTDATA is an association list which associates the user’s data names with the actual data structures.

### 2.1.5. The UNITS Package

The Unit Editor is not available from AGE. If a predefined Units representation is desired it must be built using the UNITS Package. The Units information can be built dynamically from the RHS of rules as allowed by the ch.type of PROPOSE. In addition, all UA and UM functions are available from the
LHS of rules. When using UNITS, the user needs to execute \texttt{(OPENNETWORK)} before accessing the Units Knowledge Base, and \texttt{(CLOSENETWORK)} to close the knowledge base. Refer to the \textit{UNITS User's Manual} for a more complete description of the Units representation and its uses.

\begin{verbatim}
OPENNETWORK (NETSPEC NCFLG) - lambda-spread

NETSPEC - a specification for an existing UNITS network. This is a network name which may be preceded by a directory specification.

NCFLG - a flag to control loading of functions with the network. When the flag is NOFNS, no LISP functions will be read with the network.

OPENNETWORK opens the appropriate files for a knowledge base. This involves (1) reading the relations into core and (2) opening the dynamic units file - which is used as an extended memory. If NCFLG is NOFNS, then no attempt will be made to read LISP functions with the units. If a network is already open, then an error occurs unless it is the one specified by NETSPEC.

value - the name of the network which was opened; or NIL if the network could not be opened successfully.

CLOSENETWORK (FLG) - lambda-spread

FLG - a flag whose non-NIL value indicates that the files are to be re-opened to continue processing.

CLOSENETWORK causes the current network to be written out to disk. If FLG is T, then the network is reopened to continue processing.

MAKEUNIT (UNIT RELATIVE RELATION CLASS) - lambda-spread

MAKEUNIT creates a new unit in the network. If the creation process is successful, MAKEUNIT returns the name of the new unit. Otherwise, it prints an appropriate error message and returns NIL.

Check that RELATIVE exists, UNIT not a unit already, and RELATION is valid.

DELETEUNIT (UNIT KEEPSPECFLG) - lambda-spread
\end{verbatim}
DELETEUNIT deletes a unit from the network. It attempts to be thorough about removing all traces of the unit. First it deletes all slots in the unit. Then it deletes all instances of the unit. The treatment of specializations depends on the value of KEEPSPECFLG. If KEEPSPECFLG is NIL (the default case) all specializations are also deleted. If KEEPSPECFLG is T, all specializations of the deleted unit become specializations of its generalization. DELETEUNIT returns UNIT if the operation was successful and NIL otherwise.

UNITMSG (ARGS) · lambda-nospread

UNITMSG sends a message to a unit. UNIT is the name of the unit and TOKEN is the name of the slot containing the relevant attached procedure. UNITMSG applies the function giving it ARG1 thru ARG4 and the elements of ARGLIST as arguments and returns the value of the function.

MAKESLOT (SLOT UNIT ROLE DATATYPE) · lambda-spread

MAKESLOT creates a new slot named SLOT in the unit named UNIT and all of its progeny. It returns SLOT if the operation is successful and NIL otherwise.

DELETESLOT (SLOT UNIT KEEPFLG) · lambda-spread

DELETESLOT deletes a slot from UNIT and its progeny. It returns SLOT if the operation is successful and NIL otherwise. KEEPFLG controls the extent of the deletion. If KEEPFLG is NIL, the slot is deleted in all progeny (default case); if KEEPFLG is T, the slot is kept in all progeny; if KEEPFLG is NOTSPEC, the slot is deleted from all instances and kept in all specializations and their progeny.

SLOTMSG (SLOT UNIT TOKEN FAKEVALUE ARG2 ARGLIST) · lambda-spread

SLOTMSG sends a message to a particular slot in a unit. First it looks for the procedure in a field of the slot. If it is not there, it looks for it in the datatype. If the datatype unit is a SLOTUNIT, it chases through the datatype of the value slot of
the SLOTUNIT. If there is still no procedure, it looks for one in a unit with the name of the slot.

PUTVALUE (SLOT UNIT VALUE NOTESTFLG ROLE TV PFLG)
- lambda-spread

PUTVALUE is used to set a value in a slot. It verifies that the value satisfies the requirements for being a terminal value according to the role associated with the slot.
2.2. User Programs in the Backchain Framework

For some problems it is possible to achieve a solution by chaining the production rules before evaluating them. Starting with a goal rule, the backchain framework looks for rules whose actions will satisfy the conditions of the goal rule. Each of these rules, or subgoals, is then traced to find its subgoals, recursively, until all the conditions of a subgoal either refer to input data or have been previously chained. The rules are then evaluated in the reverse order that they were found, starting with the lowest subgoal.

The Backchain framework consists of the following components:

- A data structure which is used to generate the solution path and to store intermediate results.

- A Knowledge Source whose rules can be chained in a backward manner, so that values referenced in the conditions of the rules are always available because of the earlier execution of rules whose actions conclude those values.

- Some control information.

The critical aspect of setting up backward chained rules is the selection of the structure of objects appearing on the LHS and the RHS of rules that serve as the links between those rules when they are chained. AGE provides several different ways to chain the rules depending on the structure of the chaining data.

2.2.1. Structure of Chaining Data

Overview

There are three types of chaining data available in the AGE Backchain framework:

1. attribute-only,
2. fixed object-attribute, and
3. variable object-attribute.

With the attribute-only chaining data (BCDATA.ATTR) the names of the attributes are used to chain the rules. In this simple case each attribute will be a global variable and the value of the attribute, the value of the variable. See the PUFF implementation using EMYCIN for an example.

With the fixed object-attribute chaining data (BCDATA.FIXED-OBJ) the object must be given explicitly in the conditions and actions of the rules, i.e. as a literal atom. This does not allow for the specification of relationships between objects. You will be asked for the names of the objects you want to use, and for the attributes to be associated with each object. The same attribute may be associated with several objects. Each object-attribute pair must be unique and these pairs will be used to backchain the rules.

With the variable object-attribute chaining data (BCDATA.VAR-OBJ), objects are not stated explicitly in each rule, but are computed as the rules are evaluated. AGE requires that the attributes
be unique to each level of objects for this method (same constraint as in MYCIN). With this restriction, AGE can determine the level of an object from the attribute, and by keeping track of the context, compute the relevant object within that level. Thus the same rule may be applied to several objects (or contexts). You will be asked to specify the names of the levels here, and the names of the attributes to be associated with those levels.

Definition

The variables which contain the data structure information are:

CHAININGCASE  NODELIST  ATTLIST

CHAININGCASE - There are three datatypes currently available in the AGE Backchain Model. These three datatypes correspond to the three backchaining methods (or cases) implemented in AGE. The chaining cases are:

- 1A ATTR-ONLY attribute only
- 2A EXPLICITOBJ explicit objects with attributes
- 2B VARIABLEOBJ variable objects grouped by levels, with attributes

In the ATTR-ONLY case, each attribute is treated as a variable with a value. In EXPLICITOBJ, the objects names have property list associated with them and the attributes and their values are stored as property names and values. The VARIABLEOBJ case uses levels with property lists to store the objects associated with each level and the attributes applicable to objects on that level. Attributes can only be used on one level. Each object also has a property list with the attributes and their values.

Rules are chained by the attributes and objects they refer to, either just the attribute name for ATTR-ONLY, the object and attribute names for EXPLICITOBJ, and the level, object and attribute names for VARIABLEOBJ.

NODELIST - The value of NODELIST is a list of objects used in the EXPLICITOBJ chaining case, a list of levels for the VARIABLEOBJ case, and NIL for ATTR-ONLY.

ATTLIST - The value of ATTLIST is a list of attributes used in a backchained user system.
2.2.2. Knowledge Sources

The rules in the Backchain knowledge source have a more rigid structure than those in the Forward driven KSs. The Backchain rules must be set up so that they can be chained. Thus each condition and each action must specify explicitly all the data items they refer to. The specific structure of the rules in the Backchain knowledge source depends on the data structure used. The data structure determines how the rules will be chained and therefore, what information must be associated with each rule.

- **attribute-only case**: The rules are chained through references to the attributes. If a rule contains a reference to attribute X in one of its conditions, then all rules with the attribute X in their actions become subgoals of the first rule.

- **fixed object-attribute case**: The rules are chained by attribute-object pairs.

- **variable object-attribute case**: The rules are chained by the attributes, which must be unique.

**Definition**

The name of the Backchain KS is contained in KSLIST. The name of the KS is also a global variable, whose value is the knowledge source.

```
<knowledge-source> := (<preconditions> <rule> ...)
```

```
<rule> := (<rule #> <lhs> <action> ...)
```

```
<rule #> := an integer number identifying the rule.
```

```
<lhs> := (<bccondition> ...)
```

A list of the conditions which determine the applicability of a rule.

```
<bccondition> := (<premise> <object> <attribute> <arg> ...)
```

```
<premise> := the name of a premise function. Each premise function has one or more arguments of the form (fn <value> { <arg> } ...)
```

```
and will be invoked by using the <object> and <attribute> to determine the <value> to be used in the function call.
```

```
<object> := the name of an object in the data structure. This is NIL for ATTR-ONLY data types. It is evaluated for VARIABLE-OBJ data types.
```

```
<attribute> := an explicit attribute in the data structure.
```
<arg> := other arguments needed by the premise function. Eg: \((\text{BETWEEN} \ <\ \text{val}\rangle\)

\(<\text{low}\rangle \ <\text{high}\rangle\))

Let 

<action> := (PROPOSE <key> <value> ...)

Backchain actions are always propositions. Their syntax is a
restricted version of the forward driven action syntax.

<key> := ch.type | element | attr.value | comment

The value associated with each of the allowable keys is as follows:

ch.type - BCADD | BCMODIFY | BCSUPERSEDE

The name of the type of change to be made by this proposition.
For backward chained execution, this is the name of a function
which will be applied to the element and attr.value of the
proposition to produce the desired effect.

BCADD (LEVEL ATTR-VALUES) - lambda-spread

Add a new object to the data structure.

LEVEL - The level to which the new object should be added.

ATTR-VALUES - A list of attribute value pairs to associate with the new
object.

Effect - Creates a new object in the data structure. Can only be used
with variable object or hypothesis datatypes. The new object
name is stored in the global variable NEWNODENAME.

BCMODIFY (OBJECT ATTR-VALUES) - lambda-spread

Modifies a value in the data structure.

OBJECT - An expression which evaluates to a data object. (NIL for
ATTR-ONLY datatype.)

ATTR-ONLY - A list of attribute-value pairs. The attribute is an
explicit atom, not evaluated. The value is an expression to be
evaluated.
Effect: adds the new value to the list of existing values for the
given object and attribute.

BCSUPERSEDE (OBJECT ATTR-VALUES) - lambda-spread

Supersede a value in the data structure with a new value.

OBJECT - an expression which evaluates to an object in the data
structure. (NIL for ATTR-ONLY data.)

ATTR-VALUES - list of attribute-value pairs, the attribute is an
explicit atom, not evaluated and the value is an expression to
be evaluated.

Effect: Replaces the value associated with the attribute of the object
in the data structure, with the new value in ATTR-VALUES.

element - NIL | <object> | <level>

where:

NIL is for attribute only chaining case.

<object> := an explicit object name for attribute of explicit
object case.

<level> := an expression evaluating to a level name for the
attribute of variable object case.

attr-value - (attribute <value>) | ((attribute <value>) ...)

where:

attribute := an attribute in the data structure.

<value> := an expression which will be evaluated for the new
value of the attribute.

comment := Any text description provided by the user.
2.2.3. Control Information

In addition to the rules and preconditions, the Backchain KS must also have an Evaluation function and a value adjustment function. Since the rules are executed in the order determined by the chain, no focussing or termination information is needed. The user may specify a post-processing function.

# LHSEVALUATOR - The value of # LHSEVALUATOR is a function name. This function will be applied to the LHS of a rule to determine if the rule should be executed.

# VALUEADJUSTER - The value of # VALUEADJUSTER is a function name or function expression. The function will be evaluated to determine the adjusted value associated with a link or attribute. The function to be executed may use the global variables $NODE, $ATTR, and $VAL. If a function is not given, then no value adjustment is performed.

USERPOSTPROCESSINGFN - The value of USERPOSTPROCESSINGFN is a function name or function expression. This function will be evaluated at the end of the execution of the user program.

2.2.4. Input Data

The user may specify some data items to be input data items. These data items are stored on the global variable INPUTDATANAMES with the following structure.
If chaining case = 1A then INPUTDATANAMES := (<data-name> ...) Else INPUTDATANAMES := ((<object> <data-attribute> ...) ...)

Usually the user is asked for values for the input data at the beginning of each run. However the user may also specify his own prompts for some of the data items. These prompts are saved on the property list of the combined attribute - object name under the property AGPROMPT. The prompt is evaluated when the input data item is referenced during execution.
3. User Program Execution

The user program is executed by instructing the AGE executive (see Section 4.2) to perform the task RUN. This invokes execution of the LISP function \texttt{UXRUN}.

\texttt{UXRUN} \cdot \text{lambda}\text{-spread}

Execute the user program.

This is the function responsible for controlling execution of the AGE user program. In programs specifying backward chained rule interpretation, execution of the program is performed by the function \texttt{RUNBACKCHAIN}. Otherwise, execution of the user program is performed by the function \texttt{AGESYSTEMCONTROL}.

3.1. Forward Driven Execution

\texttt{AGESYSTEMCONTROL} \cdot \text{lambda}\text{-spread}

This is the function responsible for executing the AGE user program.

Execution of a user program consists of:

1. initialization;
2. execution; and
3. termination.

Initialization includes:

a. initializing system variables and data structures;
b. prompting for user input data;
c. invoking the user initialization function and selecting initial knowledge sources;
d. invoking the initial knowledge sources.

Execution consists of a loop involving:

a. termination check;
b. focus of attention; and

This loop continues until the termination check indicates that the user program has ended.

Within the AGE execution loop, focus of attention consists of invoking the user specified control rules to select the name of the step type to be executed next.

Inference generation consists of executing the step execution function for the type of step selected. If executing the focussed step type does not cause any action to be taken, then
alternate step types are tried.

Termination of the AGE user program occurs either:
   a. when no action was taken during the previous inference
generation step, or
   b. the user supplied termination condition has a non-NIL value.

When the user program terminates, the user post-processing
function is invoked and execution ends.

Prompting for user input data is controlled by the function ASKFORDATA:

`ASKFORDATA (NAMELST) · lambda-spread`

`NAMELST` - a list of the names of the global data structures which are
required by the user program.

`effects` - prompts the user for values for each of the named data
structures. The user has the option of specifying a file to be
loaded containing the data structures in the global variable
`INPUTDATA`, or of entering each data structure at the terminal.
When the values are entered at the terminal, the user is given
the opportunity to save them on a file before continuing.

`value` - an association list in which each variable name is associated
with the data structure which is its value. If a data structure
has a prompt under the property AGPROMPT, the user will not be
asked for its value here (from ASKFORDATA). Instead the value
`UNPROMPED` will be returned on association list with each
unprompted data name.

Inference generation occurs during the execution of the various step execution functions:

`EVENT () · lambda-spread`

This is the step execution function for an event step type. It selects
and invokes one or more knowledge sources. First it uses the
selection method for events to select a step from the event
list as the current focussed event. The value of the global
variable FOCUSEVENT will be set to the selected event step for
the duration of the execution of the step. Then it invokes the
appropriate knowledge sources determined by the event type of
the focussed event.

The control information associated with the event step type
determines the selection method used to select a step from the
event list to be focussed. Then the collection rules are
checked to consolidate all equivalent steps in the event list.
The event type of the focussed event is used to determine the
set of knowledge sources to be invoked next. Invoke each ks
whose precondition contains the event type of the focussed event. If none of the knowledge sources contain the desired precondition, then invoke each ks whose precondition includes T.

---

**EXPECTATION()** - lambda-spread

This is the step execution function for an expectation step type. Execution of an expectation consists of the following actions:

a. use the expectation selection method to select the first step in the expectation list whose effect expression satisfies the expectation matcher; The value of the global variable FOCUSEXPECTATION will be set to the selected EXPECTATION step for the duration of the execution of the step;

b. remove the selected expectation and any duplicates from the expectation list;

c. perform the actions which are the effect of the expectation.

---

**GOAL()** - lambda-spread

This is the step execution function for a goal step type. Execution of a goal consists of the following actions:

a. use the goal selection method to select the first step in the goal list whose effect expression satisfies either the goal matcher or the goal seeking method; The value of the global variable FOCUSGOAL will be set to the selected GOAL step for the duration of the execution of the step;

b. remove the selected goal and any duplicates from the goal list;

c. perform the actions which are the effect of the goal.

---

During the execution of the user program, each action of a rule which is executed creates a new step entry in one of the step-type lists. These entries determine what steps remain to be performed by the program.

**EVENTLIST** - A list of (step-entry)s which are the events remaining to be processed.
EXPECTATIONLIST - A list of <step - entry>s which are the expectations remaining to be processed.

GOALLIST - A list of <step - entry>s which are the goals remaining to be processed.

FOCUSEVENT - The value of FOCUSEVENT is a step from the event list. This is the current focused event during the execution of an event step.

FOCUSEXPECTATION - The value of FOCUSEXPECTATION is a step from the expectation list. This is the current focused expectation during the execution of an expectation step.

FOCUSGOAL - The value of FOCUSGOAL is a step from the goal list. This is the current focused goal during the execution of a goal step.

<step - entry> := (<type> <node> <hypochange> <support> <enum> <effect>)

a step entered in one of the step lists.

<type> := an event type.

<nodes> := the name of a hypothesis node, or a unit in the UNITS knowledge base. The value of the hypo-element or unit property of the proposition creating this step.

<hypochange> := The list of terms causing changes to the hypothesis or UNITS knowledge base.

<support> := The name of some knowledge source. The KS which was being invoked when this step was created.

<enum> := A unique name assigned to identify this step.

<effect> := NIL | (<expr> <action>) ...

the expression and <action>s associated with expectations or goals.

HISTORYLIST - a list of history entries which are used to record the run as it progresses. In order for the explanation system to have enough information to explain later how AGE did what it did, HISTORYLIST must have a structure that reflects the way AGE changes the blackboard. Namely, each KS has separate rules, each...
rule has separate PROPOSE’s, and each PROPOSE has separate
changes on its attr-value slot. This is done using a nested set
of globals that correspond directly to the nesting of change
actions and the nesting of the functions that do the changing. This
nesting is needed only for event, expectation, or goal actions.
The history list also has entries for KS calls, list focuses, and
steps. These require nothing fancy. The variables involved in
the nesting are:

HISTORYENTRY, which collects one entry on HISTORYLIST, i.e.,
one rule firing entry;

HLACTENRY, which collects one entry on HISTORYENTRY, i.e., one
PROPOSE entry of the rule;

HLCHGENTRY, which collects one entry in HLACTENTRY, or one change
entry in the attr-value slot of the rule.

The structure of HISTORYLIST follows:

HISTORYLIST ::= (\textit{historyentry} \textit{...})

\textit{historyentry} ::= (\textit{HL#} \textit{entrytype} \textit{name}
\textit{number} \textit{changes})

\textit{HL#} ::= the name used to identify the entry. This is an atom
created by packing HL onto a unique number.

\textit{entrytype} ::= KS | STEP | FOCUS | EVACTION | EXACTION | GACTION

\textit{changes} ::= (\textit{actionentry} \textit{...})

\textit{actionentry} ::= (\textit{changentry} \textit{...})

\textit{changentry} ::= (\textit{changetype} \textit{node} \textit{attr}
\textit{value} \textit{remlinks} \textit{addlinks})

\textit{changetype} ::= LIST | ADD | MODIFY | SUPERSEDE

The use of \textit{name}, \textit{number}, and \textit{changes} for a given
\textit{entrytype} is explained in the following diagram:
AGE Reference Manual

3.1 Forward Driven Execution

KS -- <name>::= <KSname>
    -- <number>::= <invokation#>

STEP -- <name>::= <new steptype>
    -- <number>::= <step#>
    -- <changes>::= <old steptype>

FOCUS -- <name>::= <list name>
    -- <number>::= <list element ID>
    -- <changes>::= <value of element focused>

ENTRYTYPE ::=<entrytype>

EVATION -- <name>::= <KSname>
    -- <number>::= <rule#>
    -- <changes>::= (<actionentry> ...)
    <actionentry>::= (<changentry> ...)

EXATION -- <name>::= NIL
    -- <number>::= NIL
    -- <changes>::= same as for EVATION

GACTION -- same as for EXATION

The use of <node>, <attr>, <value>, <remlinks>, and <addlinks> for a given <changetype> is explained in the following diagram:

LIST -- <node>::= <list name>
    -- <attr>::= ADD
    -- <value>::= <list element ID>

ADD -- <node>::= <level name>
    -- <attr>::= <node name>

MODIFY -- <node>::= <node name>
    -- <attr>::= <attribute name>
    -- <value>::= <value used in change>
    -- <remlinks>::= links that were removed because of the change.
    -- <addlinks>::= links that were added because of the change.

SUPERSEDE -- same as MODIFY

Also,
<list element ID>'s are EV10, G3, etc.

<steptype>::= <list name>::= | EVENT
| EXPECTATION
| GOAL

<history - record> := (<hnum> <rnum> <pred> <succ>.
<step - entry>)
an entry in the history list.

<hnum> - the name of the step which this record represents.

<rnum> - a rule number.

<pred> - the name of the step which was the predecessor of this record.

<succ> - a list of the names of the steps which are immediate successor
to this record.

3.2. Backward Chained Execution

.....

RUNBACKCHAIN () - lambda: spread

This is the top level function responsible for backward chained
execution of an AGE user program. To execute the user program,
the function performs the following steps:

1. Reads in input data.

2. Asks for number of the top goal rule, unless the system has
been previously run and the goal rule number was saved in
GOALRULENO.

3. Calls BACKCHAINER to chain the rules.

4. Depending on the CHAININGCASE (datatype) calls the
appropriate backchain rule interpreter.

5. Evaluates the user's postprocessing function. If there is no
postprocessing function, PRINTBCDATA is called.

.....
**BCASKFORDATA ()** - lambda spread

**Effect:** First collect all the input data names, from (INPUTDATA? T) property, then ask for data items or a file name, and finally insert the data values into the user's data structure. If a data item has a prompt (on its property list under AGPROMPT) don't ask for its value here. STORE UNPROMPTED instead, it will be prompted for later when needed.

---

**FINDNEWCONTEXT (LEVEL OLDCONTEXT)** - lambda - spread

Generates a new list of contexts from an level name and an old context.

**LEVEL** - an atom, a level name.

**OLDCONTEXT** - an atom, an object name.

**Method** -- We have a choice: (1) Search member/memberof tree that originates from OLDCONTEXT, looking for objects on the LLIST of LEVEL. These are the new contexts. (2) Search LLIST's of blackboard for OLDCONTEXT, giving the level it's on. Then, knowing the level of LEVEL, propagate the member/memberof tree in that direction giving the new contexts. Here, the former is used.

**Effect** -- Used by VARIABLEOBJ datatype to generate the explicit objects a rule refers to from the variable object. The level name gives the applicable object level. Returns a list of the objects at that level which are linked to the old context, either by MEMBER or MEMBER/OF.

---

**BCRHSMANAGER (ACTS CONTEXT PREMISEVALUES)**

- lambda - spread

**BCRHSMANAGER** executes the actions on the RHS of BCRules.

**ACTS** - A list of actions from the RHS of a BCRule.

\[
\langle \text{BCaction} \rangle := (\text{PROPOSE} \text{ chtype} \langle \text{BCchange type} \rangle \text{ element} \langle \text{object} \rangle \text{ atval} \langle \text{attribute-value pairs} \rangle \text{ support} \langle \text{BCKSNAME} \rangle \text{ comment} \langle \text{English comment} \rangle )
\]

**Effect** -- Applies the change type to the object and attribute value
pairs. Records the change on the HISTORYLIST. The BCchange types available are BCMODIFY, BCSUPERSEDE, and BCADD.

PRINTBCDATA () · lambda-spread

Print the backchain data structure.

BC EXPLAIN () · lambda-spread

Effect-- answers questions about the previous backchain execution.

Why did a rule fire?
Why didn't a rule fire?
Why was a particular value concluded for this object-attribute pair?
What rule concluded a particular value for this object-attribute pair?
What was the value of this object-attribute pair before or after a certain rule fired?

HLFIND, the basic history list search function for black board model systems, is also used here to retrieve information from history lists produced by backchain models. A backchain history item has the following syntax.

<backchain history item> := (<history number> BACKCHAIN <NIL <rule number> <list of premise values> <list of changes>>) <changes> := (NIL <node being changed> <attribute> <new value>)

WHY DID (RULENO) · lambda-spread

Why did RULENO fire? | Answer include 1. RULENO never fired. 2. because its premises evaluated satisfactorily using the current # LHSEVALUATOR: a. The premises are b. The values of the arguments to the premises are c. The premises evaluated to ...

Find history list item recording changes made by RULENO and print the premises and premise values.

WHY DIDN'T (RULENO) · lambda-spread

WHY DIDN'T RULENO fire?
1. It did fire. 2. Its results were never needed. 3. Another rule supplied the required information first. 4. One of its
conditions failed.

Find the place in the backchain execution where RULENO could have fired - i.e. other rules which made changes to the same object attribute pairs as RULENO.

---

**WHATRULE** (VAL OBJ ATTR TAIL) \( \cdot \) lambda spread

Why was VAL concluded for ATTR of OBJ?

Effect -- Retrieve history item recording change to ATTR of OBJ. If new value is equal to VAL then return the number of the rule which made that change.

---

**WHATWAS** (OBJ ATTR TIME RULENO) \( \cdot \) lambda spread

What was the value of ATTR of OBJ before or after RULENO fired?

Effect -- Retrieve from the history list the new value of a change to ATTR of OBJ; from the history list item which was made either by RULENO (for AFTER) or by the last change to ATTR of OBJ BEFORE RULENO fired.
4. The AGE System

4.1. System Initialization

SYSINIT () · lambda·spread

Perform system initialization for entry to a SYSOUT file.

- re-initialize any timing breakdown in progress;
- reset any global variables on the list SYSOUTRESETVARS;
- reset the LISP history list;
- read and execute the user’s LISP initialization file;
- print an announcement of the name and date of the SYSOUT file;
- save system use statistics on the system recording file;
- reset the system variables LISPFILE, USERNAME, SYSTEMNAME, and HOSTNAME;
- check system text files to insure use of the latest version, and update if necessary;
- ask the user if he is already familiar with the system, and set NEWUSERFLAG accordingly.

4.2. The AGE Executive

User interaction with the AGE system is controlled by an executive program named UCONTROL. This program allows the user to select from the list of predefined tasks which constitute the system.

UCONTROL (U) · lambda·spread

Task control and execution.

U · an optional name of a task to be executed.

- the given task is executed. If no task name is given, then the program prompts for tasks until instructed to quit; executing each task as it is requested.

The primary list of tasks available at any particular time is contained in the global variable OPTIONLIST. In addition, the variable ALLOPTIONS contains a list of all known tasks. When a
user enters a task name, it must either match one of the entries in these lists, or it must be the name of a task function.

Once a valid task is selected, the program determines the file environment required for the task, initializes various variables, and then executes the function which performs the task. The name of a function which executes a given task is usually found by prefixing the task name with UX; although in some cases, the task function may have the same name as the task.

Instead of specifying a function to be executed, a task may simply call for another list of tasks to be made available to the user. In this case, the control program will simply start a new version of itself at a lower level with OPTIONLIST reset to a new list of tasks.

The execution of tasks is error protected, so the user may terminate any task by simply typing tE.

4.2.1. AGE Tasks

Following is a summary of the tasks available in AGE:

- **RECORD** - Start/finish a typescript file.
- **Design** - Create the user program.
- **SAVE** - Save the user program on a file.
- **GETSYS** - Get the user program from a file.
- **Check** - Check syntax of the user program.
- **Examine** - Examine parts of the user system.
- **Reset** - Reset the user system.
- **Run** - Execute the user program.
- **SYSOUT** - Save everything (ie. SYSOUT).
- **Help** - Examine descriptions of the current list of possible tasks.
- **Lisp** - Enter lower level LISP.
- **Quit** - Exit from LISP.
- **Tasks** - Show a complete list of all tasks available and examine their descriptions.
- **Util** - System utility tasks.

The following system utility tasks are available:

- **Schema** - Display the design schema.
- **Hacker** - Unstructured system access.
- **EX?** - Print any explanation available for a given name.
- **Doc** - Print documentation for a file.
- **Genman** - Generate a reference manual.
- **Userinfo** - Show past user information.
- **Load** - Load files.
- **Mksys** - Create subsystem for immediate execution.
4.2.2. Hacker Mode Tasks

In the general system tasks, the Hacker task consists of an extensive set of additional tasks to be used for system specification. This is the same set of tasks which are available during system Design in the UNGUIDED mode. Following is a list of these tasks:

- **Hypo**: Hypothesis building function.
- **PRecreatednodes?**: Allow the user to predefine some of the nodes at a particular level.
- **EDITNodename**: Specify the LHS evaluator and weight adjustment functions.
- **EDITEvaluator**: Load and/or enter user defined functions.
- **EDITUserfns**: Edit the user defined functions.
- **KERNEL**: Specify control information.
- **MACRO-EVentspec**: Default control specifications for an event driven system.
- **MACRO-EXpectspec**: Default control information for an expectation driven system.
- **EDITMacrocontrol**: Specify the names of the input data structures.
- **INPutformat**: Specify the name and definition of your pre-processing function.
- **Terminate**: Name and/or define the termination condition.
- **POSTprocessing**: Name and/or define your post processing function.
- **BACKchain-ks-acq**: Acquire Backchained rules.
- **EDITBCKs**: Edit Backchained rules.
- **Attr-only**: Acquire attribute-only data for backchained rules.
- **EDIT-ATTR ONLY**: Edit attribute-only data.
- **EXPlicitobj**: Acquire explicit object data for backchained rules.
- **Variableobj**: Acquire variable object data for backchained rules.
- **EDIT-ATTR-OF-obj**: Edit attribute of object data.
- **BCevaluator**: Acquire LHS evaluator for backchained rules.
- **EDITBCEvaluator**: Edit LHS evaluator for backchained rules.
- **END**: End selection.
5. User Program Design and Specification

AGE provides assistance in the process of designing a user program by using an AND/OR graph whose structure reflects the decisions needed to create a user program. The user can use the DESIGN task to guide him through the nodes of the graph, specifying the components of the user program as they are called for.

The design hierarchy is defined by the value of the following variables:

- AGESCHEMENAMAPROPLIST
- AGESCHEMENAMALIST
- AGESCHEMA

AGESCHEMENAMAPROPLIST - a list of the names of all the properties used to describe a node in the design hierarchy.

AGESCHEMENAMALIST - a list of the names of all the nodes in the design hierarchy.

A description of the properties of the nodes in the design hierarchy follows:

AGESCHEMA

- TOUSFR - a keyword for access to the MANUAL
- ADVISE - a keyword for access to the MANUAL. Advice about what this component will do and why it should be done.
- FINALADVISE - a keyword for access to the MANUAL. Advice about what to do when this node and all of its subnodes have been completed.
- PARTOF - a name or list of names. The schema components for which the current component is a sub-component.
- CONTAINS - a name or list of names. The schema components which are sub-parts of the current component.
- OPTIONAL - T if this component is not required in the completed system.
- ACQUISITIONFN - the name of a function which interacts with the user to create this component.
- EDITFN - the name of a function which interacts with the user to change the present state of this component.
- PRINTFN - the name of a function which prints this component.
- CHECKFN - the name of a function which checks the syntactic and semantic correctness of this component.
- BRANCHTYPE - the type of branching which takes place at this node. Either AND or OR.

The design hierarchy consists of the following nodes and properties:

DESIGN.PLAN

- TOUSFR - DESIGN.PLAN.TOUSFR
ADVISE - NIL
FINALADVISE - NIL
CONTAINS - (BLACKBOARD.MODEL BACKCHAIN )
BRANCHTYPE - OR
BLACKBOARD.MODEL
TOUSER - DESIGN.BLACKBOARD.MODEL.TOUSER
ADVISE - DESIGN.PLAN.ADVISE
PARTOF - DESIGN.PLAN
CONTAINS - (HYPOTHESIS KNOWLEDGE CONTROL USERFUNS )
BRANCHTYPE - AND
HYPOTHESIS
TOUSER - DESIGN.HYPOTHESIS.TOUSER
ADVISE - DESIGN.ADVICE.HYPOTHESIS
PARTOF - DESIGN.PLAN
CONTAINS - (HYPOTHESIS.ELEMENT )
ACQUISITIONFN - HYPO
EDITFN - HYPO
PRINTFN - PRHYPO
BRANCHTYPE - OR
HYPOTHESIS.ELEMENT
TOUSER - DESIGN.HYPOTHESIS.ELEMENT.TOUSER
ADVISE - DESIGN.ADVICE.HYPOTHESIS.ELEMENT
PARTOF - DESIGN.PLAN
CONTAINS - (HYPOTHESIS.ELEMENT )
ACQUISITIONFN - PRECREATEDNODES?
EDITFN - PRECREATEDNODES?
KNOWLEDGE
TOUSER - DESIGN.KNOWLEDGE.TOUSER
ADVISE - DESIGN.ADVICE.KNOWLEDGE
PARTOF - DESIGN.PLAN
CONTAINS - (KS RULE.EVALUATOR )
BRANCHTYPE - AND
KS
TOUSER - DESIGN.KS.TOUSER
ADVISE - DESIGN.ADVICE.KS
PARTOF - DESIGN.PLAN
CONTAINS - (KS RULE.EVALUATOR )
ACQUISITIONFN - KNOWLEDGESOURCES
EDITFN - KNOWLEDGESOURCES
PRINTFN - (LAMBDA NIL (PRINT KSLIST )
RULE.EVALUATOR
TOUSER - DESIGN.RULE.EVALUATOR.TOUSER
ADVISE - DESIGN.ADVICE.RULE.EVALUATOR
PARTOF - DESIGN.PLAN
CONTAINS - ( MACRO EVENT-DRIVEN MACRO EXPECTATION-DRIVEN DETAILCONTROL )
BRANCHTYPE - OR
MACRO EVENT-DRIVEN
TOUSER - DESIGN.MACRO EVENT-DRIVEN.TOUSER
PARTOF - DESIGN.PLAN
ACQUISITIONFN - MACRO-EVENTSPEC
EDITFN - MACRO-EVENTSPEC
MACRO.EXPECTATION-DRIVEN
TOUSER - DESIGN.MACRO.EXPECTATION-DRIVEN.TOUSER
PARTOF - CONTROL
ACQUISITIONFN - MACRO-EXPECTSPEC
EDITFN - MACRO-EXPECTSPEC
DETAILCONTROL
TOUSER - DESIGN.DETAILCONTROL.TOUSER
ADVISE - DESIGN.ADVISE.DETAILCONTROL
PARTOF - CONTROL
CONTAINS - (INPUTFORMAT INITIALIZATION TERMINATION
POSTPROCESSING CONTROL.KERNEL )
BRANCHTYPE - AND
INPUTFORMAT
TOUSER - DESIGN.INPUTFORMAT.TOUSER
ADVISE - DESIGN.ADVISE.INPUTFORMAT
PARTOF - DETAILCONTROL
ACQUISITIONFN - INPUTFORMAT
EDITFN - INPUTFORMAT
INITIALIZATION
TOUSER - DESIGN.INITIALIZATION.TOUSER
ADVISE - DESIGN.ADVISE.INITIALIZATION
PARTOF - DETAILCONTROL
ACQUISITIONFN - INITIAL
EDITFN - INITIAL
TERMINATION
TOUSER - DESIGN.TERMINATION.TOUSER
ADVISE - DESIGN.ADVISE.TERMINATION
PARTOF - DETAILCONTROL
ACQUISITIONFN - TERMINATE
EDITFN - TERMINATE
POSTPROCESSING
TOUSER - DESIGN.POSTPROCESSING.TOUSER
ADVISE - DESIGN.ADVISE.POSTPROCESSING
PARTOF - (DETAILCONTROL BACKCHAIN.CONTROL )
OPTIONAL - T
ACQUISITIONFN - POSTPROCESSING
EDITFN - POSTPROCESSING
CONTROL.KERNEL
TOUSER - DESIGN.CONTROL.KERNEL.TOUSER
ADVISE - DESIGN.ADVISE.CONTROL.KERNEL
PARTOF - DETAILCONTROL
ACQUISITIONFN - KERNEL
EDITFN - KERNEL
PRINTF - PRINTCONTROLINFO
USERFNS
TOUSER - DESIGN.USERFUNCTIONS.TOUSER
ADVISE - DESIGN.ADVISE.USERFNS
PARTOF - BLACKBOARD.MODEL
OPTIONAL - T
ACQUISITIONFN - GETUSERFNS
EDITFN - EDITUSERFNS
BACKCHAIN
TOUSER - BACKCHAIN.TOUSER
ADVISE - BACKCHAIN, ADVISE
PARTOF - DESIGN, PLAN
CONTAINS - (BACKCHAIN, DATA BACKCHAIN, RULES BACKCHAIN, CONTROL )
BRANCHTYPE - AND
BACKCHAIN, DATA
TOUSER - BACKCHAIN, DATA, TOUSER
ADVISE - BACKCHAIN, DATA, ADVISE
PARTOF - BACKCHAIN
CONTAINS - (DATA, HYPO BCDATA, OBJ-ATTR BCDATA, ATTR )
BRANCHTYPE - OR
BCDATA, OBJ-ATTR
TOUSER - BCDATA, OBJ-ATTR, TOUSER
ADVISE - BCDATA, OBJ-ATTR, ADVISE
PARTOF - BACKCHAIN, DATA
CONTAINS - (BCDATA, FIXED-OBJ-ATTR BCDATA, VAR-OBJ-ATTR )
BRANCHTYPE - OR
BCDATA, FIXED-OBJ-ATTR
TOUSER - BCDATA, FIXED-OBJ-ATTR, TOUSER
ADVISE - BCDATA, FIXED-OBJ-ATTR, ADVISE
PARTOF - BCDATA, OBJ-ATTR
ACQUISITIONFN - EXPLICITOBJ
EDITFN - EXPLICITOBJ
PRINTFN - PRINTBCDATA
BCDATA, VAR-OBJ-ATTR
TOUSER - BCDATA, VAR-OBJ-ATTR, TOUSER
ADVISE - BCDATA, VAR-OBJ-ATTR, ADVISE
PARTOF - BCDATA, OBJ-ATTR
ACQUISITIONFN - VARIABLEOBJ
EDITFN - VARIABLEOBJ
PRINTFN - PRINTBCDATA
BCDATA, ATTR
TOUSER - BCDATA, ATTR, TOUSER
ADVISE - BCDATA, ATTR, ADVISE
PARTOF - BACKCHAIN, DATA
ACQUISITIONFN - ATTR-ONLY
EDITFN - ATTR-ONLY
PRINTFN - PRINTBCDATA
BACKCHAIN, RULES
TOUSER - BACKCHAIN, RULES, TOUSER
ADVISE - BACKCHAIN, RULES, ADVISE
PARTOF - BACKCHAIN
CONTAINS - (BACKCHAIN, KS BCRULE, EVALUATOR )
PRINTFN - (LAMBDA NIL (PRINT KSLIST ) )
BRANCHTYPE - AND
BACKCHAIN, KS
TOUSER - BACKCHAIN, KS, TOUSER
ADVISE - BACKCHAIN, KS, ADVISE
PARTOF - BACKCHAIN, RULES
ACQUISITIONFN - BACKCHAIN-KS
EDITFN - BACKCHAIN-KS
BCRULE, EVALUATOR
TOUSER - DESIGN, RULE, EVALUATOR, TOUSER
PARTOF - BACKCHAIN, RULES
ACQUISITIONFN - BCEVALUATOR
EDITFN - BCEVALUATOR
BACKCHAIN.CONTROL
  TOUSER - BACKCHAIN.CONTROL.TOUER
  ADVISE - BACKCHAIN.CONTROL.ADVISE
  PARTOF - BACKCHAIN
  CONTAINS - POSTPROCESSING
POSTPROCESSING
  TOUSER - DESIGN.POSTPROCESSING.TOUER
  ADVISE - DESIGN.ADVISE.POSTPROCESSING
  PARTOF - (DETAILCONTROL BACKCHAIN.CONTROL)
  OPTIONAL - T
  ACQUISITIONFN - POSTPROCESSING
  EDITFN - POSTPROCESSING
6. User Program Syntax Checking

The AGE system has the ability to perform a syntax check of the user program before it is actually executed. It is advisable to perform this function before running any user program, in order to avoid errors which are easily detected and corrected. Actual checking of the user program is performed by the function SYSCHK:

```
SYSCHECK 0 lambda.spread
```

This is the function responsible for checking the syntax of the AGE user program. In AGE, the user program is a collection of data structures representing the various sections of the user system. The following section of the user program will be checked:

1. system name;
2. hypothesis structure;
3. knowledge sources;
4. control information.

Each of the sections of the user program will be scanned in succession. If any errors are detected, then an identifying reference number will be assigned to the error, the error reference will be printed at the user's terminal, and the reference will be saved along with an error message and a pointer to the actual data structure containing the error.

When the program has completed its examination of all the parts of the user program, then any errors encountered will be saved in the global variable ERRORLIST, where they can be examined by the user using the functions PRINTERROR and EDITERROR.

In addition to finding program errors, the function will also print messages about unusual program constructs found in the user's program, but which are not necessarily errors.

value - T if no errors were encountered during the system check; NIL otherwise.

```
PRINTERROR (LST) · lambda-spread
```

LST - An error descriptor created during AGE syntax checking: ( # messagelist pointer)

```
EDITERROR (LST) · lambda-spread
```
LST - An error descriptor created during AGE syntax checking: (# messagelist pointer)
7. Functions and Variables

7.1. Auxiliary Functions

The AGE system contains a number of functions designed specifically for use by the user's program. As a general system convention, the names of these functions usually begin with $. Descriptions of the functions presently available are included below.

$ADDNODE (LEVEL!$ LINKLST!$) - lambda-spread

LEVEL!$ - the name of one of the levels in the current hypothesis. The level at which a new node is to be added.

LINKLST!$ - a list of link-node expressions. These are the links to be added to the new node. This list is the same as that found in the proposition of a rule.

effects - creates a new node at the specified level. Then, for each link in LINKLST!$, determines the node value and places the links to the node.

$ADJWT () - lambda-spread

value - A new value computed from the existing values of $NODE, $ATTR, $VAL, and $TALLY. $VAL is assumed to be a value consisting of two parts, val and wt, in the form: (val wt) Each of the values currently associated with $ATTR at $NODE are also assumed to be of the same form. First, the existing wt in $VAL is modified by $TALLY. Then, if there is a current value of val associated with $ATTR at $NODE, the existing wt is further combined with the current wt. The final value returned consists of the current val and the modified wt in the form: (val wt)

$ADJWTBC () - lambda-spread

MYCIN-like weight adjusting function for use with backchained rules.

NEWWEIGHT = OLDWEIGHT + (1 - OLDWEIGHT) * NEWWEIGHT
$\textbf{AND (CONDITIONS!}) - nlambda-nospread$

CONDITIONS!$ - a list of LISP expressions to be tested.

effects - same as the LISP function AND. Each expression in the list is evaluated until a value of NIL results.

value - T if the value of every expression in the list was non-NIL.

$\textbf{SANDBC (PREMISES*)} - nlambda-nospread$

RETRIEVE ATTR OF OBJECT AND TEST PREMISE. RETURN list of values ONLY IF ALL PREMISES ARE TRUE.

$\textbf{SANDMIN (PREMISE!}) - nlambda-nospread$

PREMISE!$ - a list of LISP expressions to be tested.

effects - Each expression in the list is evaluated until a value of NIL results. Evaluation stops as soon as the first NIL value is found. The value of each expression is assumed to be a number, and the variable $TALLY$ is set to the number with smallest absolute value of 1.0 or less. Expressions whose values are not numbers do not change $TALLY$. If none of the numeric values have an absolute value less than 1, then $TALLY$ is 1.0.

value - T if the value of every expression in the list was non-NIL.

$\textbf{SANDMINBC (PREMISE*)} - nlambda-nospread$

MYCIN-like evaluator for use with backchain rules.

PREMISE* - list of conditions in the form (premisefn object attribute \texttt{arg} ...)

Effect - Evaluates each condition. If a condition evaluates to NIL, stop evaluating conditions. Save the smallest numerical result less than 1 on the global variable $TALLY$. (result of evaluating a condition)

Returns list of values if all the conditions evaluate to non-NIL values, otherwise NIL.
$\textbf{DATA (DATANAME)} \cdot \text{nlambda-spread}

\textbf{DATANAME} - the name of a user created data structure defined during program acquisition.

\textbf{value} - the data structure associated with the name. If the value is UNPROMPTED, \$\text{DATA} will invoke (EVAL) the user defined prompt stored on the property list of DATANAME under the property AGPROMPT.

.....

$\textbf{DEFAULTDOMAIN-RULES ()} \cdot \text{lambda-spread}

.....

$\textbf{DEFAULTPOST ()} \cdot \text{lambda-spread}

A function to be used a the post processing function in the user program.

\textbf{effects} - prints the complete hypothesis structure.

.....

$\textbf{DEFAULTTERM1 ()} \cdot \text{lambda-spread}

A function to be used as a termination condition for the user program.

\textbf{value} - T when the event list is empty.

.....

$\textbf{DEFAULTTERM2 ()} \cdot \text{lambda-spread}

A function to be used as a termination condition for the user program.

\textbf{value} - T when both the event list and expectation list are empty.

.....

$\textbf{FIND (AVLNLST)} \cdot \text{lambda-spread}

\textbf{AVLNLST} - a list of attr-value or link-node expressions. This list has the same format as that found in the proposition of a rule.

\textbf{value} - a list of the names of all the nodes in the hypothesis which contain attributes and values the same as all of those
specified in AVLNLST.

$L\text{LEVELNODES} (LEV) \cdot \text{lambda}\text{-spread}$

$LEV \cdot$ the name of one of the hypothesis levels.

$value \cdot$ the list of all user created or generated nodes at the specified level.

$\ldots$

$\$\text{MODIFY} (\$NODE!\$ AVWLNLST!\$ L1NKFLG!\$) \cdot \text{nlambda}\text{-spread}$

$NODE!\$ \cdot$ An expression whose value is a node in the current hypothesis.
   This expression is evaluated to find the node being modified.

$AVWLNLST!\$ \cdot$ lists of properties to be used to modify the current node.
   This list is the same as that found in either the $\text{attr-value}$ or $\text{link-node}$ property of a proposition of a rule.

$L1NKFLG!\$ \cdot$ T if the properties represent links; NIL otherwise.

effects \cdot$ uses $\text{MODIFY1}$ to add the properties to the current node.

$\ldots$

$\$\text{PEQ} (\$PATTERN1\$ PATTERN2) \cdot \text{lambda}\text{-spread}$

$\text{pattern match} \ldots \text{to be elaborated later on}.$

$\ldots$

$\$\text{SUPERSEDE} (\$NODE!\$ AVWLNLST!\$ L1NKFLG!\$)$
   \cdot \text{nlambda}\text{-spread}$

$NODE!\$ \cdot$ An expression whose value is a node in the current hypothesis.
   This expression is evaluated to find the node being modified.

$AVWLNLST!\$ \cdot$ lists of properties to be used to modify the current node.
   This list is the same as that found in either the $\text{attr-value}$ or $\text{link-node}$ property of a proposition of a rule.

$L1NKFLG!\$ \cdot$ T if the properties represent links; NIL otherwise.

effects \cdot$ uses $\text{MODIFY1}$ to replace the properties at the current node.

$\ldots$
$VALUE (NOODE ATTR/LINK WHICH) - nlambda-spread

VALUE retrieval function.

NOODE - an expression evaluating to a hypothesis element containing a desired value, can also be a backchain data item.

ATTR/LINK - the explicit name of an attribute or link. This is not evaluated.

WHICH - allows the user to specify ALL, for all the values, or LATEST, for the last value added. The default is ALL.

Returns the value of the attr of the node, or the node pointed to by the link from node.

ASIS () - lambda-spread

value - $VAL

BCASIS () - lambda-spread

MODIFY 1 (NODE AVWLST LINKFLG RFLG) - lambda-spread

NODE - An expression whose value is a node in the current hypothesis. This expression is evaluated to find the node being modified.

AVWLST - lists of properties to be used to modify the current node; where each property is either an attribute or a link : = (attribute value) | (link node)

A single property will be treated as a list of that property. The value and node in each property are expressions which must be evaluated to determine the actual items to be used.

LINKFLG - T if the properties represent links; NIL otherwise.

RFLG - A flag to indicate that previous values are to be replaced. If RFLG is T then delete the previous values and replace them with the current value, otherwise just add the new value to the existing list.

effects - For each property in AVWLST, determines the evaluated properties. Then changes or replaces the attributes or links in the specified node.
7.2. Output Functions

```
GETDATAITEMNAMES () - 
lambda-spread

PRHYPO () - 
lambda-spread

This prints the hypothesis indenting at each level.

PRHYPOLEVEL (LEVELNAME) - 
lambda-spread

PRHYPOLEVEL2 (LEVELNAME INDENT LEVELNUMBER) - 
lambda-spread

LEVELNAME is the name of a level node. INDENT is the amount of
indentation at this level. LEVELNUMBER is the number of the
level (1 is highest) This prints out this level of the
hypothesis

PRINTALL (DEVICENAME) - 
lambda-spread

PRINTBCDATA () - 
lambda-spread

Print the backchain data structure.

PRINTBCKS (KS) - 
lambda-spread

Print the backchain knowledge source (s)

KS - the name of the BC knowledge source or NIL in which case
BACKCHAINKS is printed.

PRINTCONTROLINFO () - 
lambda-spread
```
PRINTCONTROLRULES () · lambda-spread

PRINTHYPO (NDLEVELS) · lambda-spread

PRINTKS (KSLST) · lambda-spread

PRINTLST (LSTT) · lambda-spread

This asks the user how far down a list s/he wants printed then prints it. This is lambda so the entry should be the name of a list or a quoted list.

PRINTPROPLIST (LIST INDENT1 INDENT2 HOWMANY STARTINGAT ELEMENT) · lambda-spread

This prints a property list. LIST. INDENT1 is the indentation, in number of spaces, of the properties. INDENT2 is the indentation of the values. If INDENT2 is too small, it puts the value as close to the property as possible. The format is a column of <property> · <value>. The next two arguments tell HOWMANY prop-val pairs to print STARTING AT a certain place in the list. (1) HOWMANY = NIL or ALL prints the rest of the list beginning at STARTINGAT. A number means print that many prop-val pairs from STARTINGAT. A prop name for HOWMANY prints from STARTINGAT to that prop-val pair. (2) A null STARTINGAT starts at the beginning. A number for STARTINGAT starts that many prop-val pairs down the list. A propname starts at that prop name. If ELEMENT is given it is assumed that this is a printout of a backchaining node and ** is to be printed after the attributes that are data. Note that the list must still be entered.

PRINTRULE (RULE) · lambda-spread

PRINTRULEACTION (ACTN) · lambda-spread
PRINTRULES () - lambda-spread

PRNODE (ELEMENT INDENT) - lambda-spread

PRNODE2 (ELEMENT INDENT HOWMANY STARTINGAT) - lambda-spread

Outputs the element's value and its property list. INDENT is an integer indicating how far to indent the printing. The last two args are described in PRINTPROPLIST.

PRULES (RULES RHSONLYFLAG) - lambda-spread

This prints out the rules entered. RULES must be a list of rules. RHSONLYFLAG specifies printing only the RHS of the rule.

SAVECHAIN (FCOMS) - lambda-spread

FCOMS - The name of a global variable whose value is the fileCOMS of a file to be saved.

Generate a list of unique attribute names (att-obj names), then place a command to save the BACKCHAINRULES prop for those names, in the fileCOMS of the file.

SAVECONTROL () - lambda-spread

Save control information.

effects - saves all the user program control info, except for the information used in the interpretation of rules, on <usersystemname>-CONTROLINFO. Saves all of the user's control functions on <usersystemname>-USERFNS.

SAVEDATA () - lambda-spread
Saves the users data structure on a file. (<username>-DATA) Saves the CHAININGCASE, list of objects (NODELIST), list of attributes (ATTLIST), and the property lists of the objects.

.....

SAVEDESIGN () · lambda-spread

Save design information.

.....

SAVEHYPOTHESIS () · lambda-spread

Save the hypothesis structure.

.....

SAVEINDATA () · lambda-spread

Save the user's input data.

.....

SAVERULES () · lambda-spread

Save the knowledge sources and related information.

.....

SAVEUSERFNS () · lambda-spread

Save user defined functions.

.....

VALU1 (X) · lambda-spread

.....
7.3. User Prompting Functions

DISPLAYLIST (PRLIST GNAME ULFLG METAFLG DISPLAYFN DESCRPFN NAMEFN SELECTNUM) - lambda -spread

Print/examine a list of items and their associated descriptions.

PRLIST - a list of the items to be printed or examined. Each item must consist of one or more names, which will be determined by applying the function NAMEOF to the item; and an optional description associated with the name.

GNAME - an optional generic name describing the items in LST. E.g: tasks, names, files, etc.

ULFLG - a flag to indicate that names should be printed in upper and lower case. When this is a list, it is assumed to be a list of additional items which must be included in determining an unambiguous representation for the name of each item in LST.

METAFLG - a flag to indicate that the name of one or more of items may be a meta-symbol identifiable by the function METASYMBOLP.

DISPLAYFN - an optional function which prints information about a given item on the current output file. This should be a function of a single argument; or the atom T. The atom T will be interpreted as meaning a function which uses PRIN1FORMS to print CDR of an item as its description. The function will be applied to a single item from PRLIST to print its description. If no function is given, then only the names of the items will be printed.

DESCRPFN - an optional predicate which, when applied to an item in the list, indicates whether or not the item contains an associated description. The predicate should be a function of a single argument; or the atom T. The atom T will be interpreted as meaning a predicate whose value is always T. The value of the predicate should be NIL if no description exists; or non-NIL otherwise. If no predicate is given, then the function assumes that each item to be printed is in the form of a list whose description is the CDR of the list.

NAMEFN - an optional function which may be used by NAMEOF to extract the name of each item in LST.

SELECTNUM - an optional small integer giving the minimum number of items that will cause selection from the list of items. Fewer items will be displayed without selection. When the atom NEVER
is given in place of a number, then the list of items will always be printed without selection; otherwise, when no number is given, selection will be used for MINSELECTDISPLAY or more items.

effects - information about each of the items in the list is printed on the current output file. First, the name of each item is printed using the function DSPLYLST as follows:

The following <n> <gname> are available:
  x  y  ...

Following the printing of this initial list, if no DISPLAYFN is given, then no further printing is performed; otherwise, each item which has a description is printed using DISPLAYFN. However, when the list contains at least the number of items specified by SELECTNUM, then SELECTAPPLY is used to allow the user to select the items to be printed. In each of these cases, the name and associated description of each item is printed using the function DSPLYLST as follows:

  x  - <description>
  y  - <description>
  ...


DSPLYLST (DSLIST NAMEFN DISPLAYFN ULFLG METAFLG
DESCRPFN) - lambda-spread

Print information about each item in a list.

DSLIST - a list of the items to be printed. Each item must consist of one or more names, which will be determined by applying the function NAMEOF to the item; and an optional description associated with the name.

NAMEFN - an optional function which may be used by NAMEOF to extract the name of each item in LST.

DISPLAYFN - an optional function which prints information about a given
item on the current output file. This should be a function of a
single argument; or the atom T. The atom T will be interpreted
as meaning a function which uses PRIN1FORMS to print CDR of an
item as its description. The function will be applied to a
single item from DSLIST to print its description. If no
function is given, then only the names of the items will be
printed.

ULFLG - a flag to indicate that names should be printed in upper and
lower case. When this is a list, it is assumed to be a list of
additional items which must be included in determining an
unambiguous representation for the name of each item in LST.

METAFLG - a flag to indicate that the name of one or more of items may
be a meta-symbol identifiable by the function METASYMBOLP.

DESCRPFN - an optional predicate which, when applied to an item in the
list, indicates whether or not the item contains an associated
description. The predicate should be a function of a single
argument; or the atom T. The atom T will be interpreted as
meaning a predicate whose value is always T. The value of the
predicate should be NIL if no description exists; or non-NIL
otherwise. If no predicate is given, then the function assumes
that each item to be printed is in the form of a list whose
description is the CDR of the list.

effects - prints information about each item in the list on the current
output file. If no DISPLAYFN is given, just the name of each
item is printed; otherwise, the name is printed and DISPLAYFN
is applied to each item in the list. Finally, the list becomes
the new value of the global variable LASTDISPLAYLIST.

......

INPLINES (PROMPTMSG HELPMSG GOODLIST CHECKFN NOSHIFTFLG
DEFAULT KEYPROMPTFLG) - lambda-spread

Input a list of s-expressions from the user's terminal.

PROMPTMSG - a message to be used to prompt the user. This may be in any
form acceptable to PRIN1FORMS. The message is only used if the
user has not yet entered any characters in the input buffer, or
when beginning a new line of input. Otherwise the message is
not printed.

HELPMSG - an optional message to be used to respond to a user's request
for help. If provided, this must be a LISP form or list of
forms to be interpreted by PRIN1FORMS whenever ? is typed by
the user. If the form is the atom NOHELP, or if the atom ? is
included in GOODLIST, then a ? typed by the user will be
returned in the value of the function, and no response will be
given by the function.

GOODLIST - a list of entries which, in conjunction with CHECKFN, define
the acceptable user responses. When the list is omitted and no
CHECKFN is provided, any response except a null response is
considered acceptable.

An acceptable answer must be an atom. Atoms enclosed in <...>'s
are considered to be meta-symbols representing some other set
of atoms. With the exception of <CR>, meta-symbols in GOODLIST
represent atoms which must be interpreted by CHECKFN. If the
meta-symbol <CR> is included in GOODLIST, then a null response
will be allowed as an acceptable answer by the user. An atom
which matches, or partially matches, one of the acceptable
responses will not be considered as a candidate to be
interpreted by CHECKFN. If the atom ? is included in the list
as an acceptable answer, then a ? typed by the user will be
returned as the value of the function.

A message associated with any answer will be used in help
messages about that answer. The messages may be supplied in any
form acceptable to PRINTFORMS.

Each entry in the list is either an atom or a list whose first
element is an atom or list of atoms: (entry ...); where entry
is: atm | (atm <form> ...) | (atm ...) <form> ...); where atm
is an acceptable answer, and form is an associated help
message.

CHECKFN - a function which will be used to check any expressions
entered by the user which do not match an entry in GOODLIST.
The function will be applied to any expression typed by the
user which is either a meta-symbol or which does not match an
entry in GOODLIST. If the value returned by the function is
non-NIL, then that value will be taken as a valid reply by the
user.

NOSHIFTFLG - a flag signifying that input may be in lower case. When
this flag is NIL, all input will be converted to upper case.

DEFAULT - a list of default values to be returned if the user enters no
reply. These values are presumed to be correct and will not be
checked against either GOODLIST or CHECKFN.

KEYPROMPTFLG - a flag determining when a set of abbreviated keys is to
be used in the prompt to the user. If no value is given, then
the value of MAXKEYLISTLENGTH is used. If the flag is NIL, then
no abbreviated list will be used. If the flag is a small
integer, then that is the maximum size keylist that will be
abbreviated. Any other non-NIL value allows any number of keys to be abbreviated.

effects: if the input buffer is empty or if the user is beginning a new line of input, then the prompt message, default values, and abbreviated key list are printed; otherwise no user prompt is printed. The function then reads a list of s-expressions using READLINE, and the expressions are examined one at a time.

If the user enters a null reply, then the function returns the default values, if any are given. If there are no default values, but the meta-symbol <CR> is included in GOODLIST, then NIL is returned. Otherwise, the function continues reading until the user enters one or more expressions.

If the first character of any expression is a semicolon, then the remainder of the expressions are considered to be a comment which is ignored as input.

If any expression is a ?, and there is no ? in GOODLIST, and HELPMSG is not .HELP, then the help message is printed and the user is prompted again.

In responding to a user's request for help, the function recognizes two types of request. When the ? is the last expression read, the function prints all the help information available, consisting of HELPMSG, a list of the acceptable responses in GOODLIST, and any messages associated with each of the entries in GOODLIST.

When the ? is followed by one or more other expressions, then the function selects the items in GOODLIST which match any of the expressions entered, and prints just those responses with their associated messages. In either case, the input buffer is cleared before printing the message.

As each expression is examined, it is interpreted according to GOODLIST and CHECKFN, and then, if acceptable, the appropriate value is included in the input list. When the user's reply is not acceptable, then the input buffer is cleared, the user is reprompted, and reading continues from the next line.

When the last expression is reached, if any acceptable expressions have been read, then their interpreted values are returned as the value of the function; otherwise, the function continues reading from the next line.

value - a list of one or more acceptable expressions read from the user's terminal.
INPLIST (PROMPTMSG HELPMSG GOODLIST CHECKFN NOSHIFTFLG DEFAULT KEYPROMPTFLG SPELLCORRECTFLG)
   - lambda -spread

Input a list of words from a single line at the user's terminal.

PROMPTMSG - a message to be used to prompt the user. This may be in any form acceptable to PRIN1FORMS. The message is only used if the user has not yet entered any characters in the input buffer, or when beginning a new line of input. Otherwise the message is not printed.

HELPMSG - an optional message to be used to respond to a user's request for help. If provided, this must be a LISP form or list of forms to be interpreted by PRIN1FORMS whenever ? is typed by the user. If the form is the atom NOHELP, or if the atom ? is included in GOODLIST, then a ? typed by the user will be returned as the value of the function, and no response will be given by the function.

GOODLIST - a list of entries which, in conjunction with CHECKFN, define the acceptable user responses. When the list is omitted and no CHECKFN is provided, any response except a null response is considered acceptable.

An acceptable answer must be an atom. Atoms enclosed in <...>'s are considered to be meta-symbols representing some other set of atoms. With the exception of <CR>, meta-symbols in GOODLIST represent atoms which must be interpreted by CHECKFN. If the meta-symbol <CR> is included in GOODLIST, then a null response will be allowed as an acceptable answer by the user. An atom which matches, or partially matches, one of the acceptable responses will not be considered as a candidate to be interpreted by CHECKFN. If the atom ? is included in the list as an acceptable answer, then a ? typed by the user will be returned as the value of the function.

A message associated with any answer will be used in help messages about that answer. The messages may be supplied in any form acceptable to PRIN1FORMS.

Each entry in the list is either an atom or a list whose first element is an atom or list of atoms: (entry ...); where entry is: atm | (atm <form> ...) | ((atm ...) <form> ...); where atm is an acceptable answer, and form is an associated help message.

CHECKFN - a function which will be used to check any atoms entered by
the user which do not match an entry in GOODLIST. The function will be applied to any atom typed by the user which is either a meta-symbol or which does not match an entry in GOODLIST. If the value returned by the function is non-NIL, then that value will be taken as a valid reply by the user.

NOSHIFTFLG - a flag signifying that input may be in lower case. When this flag is NIL, all input will be converted to upper case.

DEFAULT - a list of default values to be returned if the user enters no reply. These values are presumed to be correct and will not be checked against either GOODLIST or CHECKFN.

KEYPROMPTFLG - a flag determining when a set of abbreviated keys is to be used in the prompt to the user. If no value is given, then the value of MAXKEYLISTLENGTH is used. If the flag is NIL, then no abbreviated list will be used. If the flag is a small integer, then that is the maximum size keylist that will be abbreviated. Any other non-NIL value allows any number of keys to be abbreviated.

SPELLCORRECTFLG - a flag determining when spelling correction should be used to match entries in GOODLIST. If the flag is non-NIL, then any user answer will be matched to GOODLIST with spelling correction applied if necessary. Normally, when this flag is NIL, no spelling correction is applied to the user responses.

effects - if the input buffer is empty or if the user is beginning a new line of input, then the prompt message, default values, and abbreviated key list are printed; otherwise no user prompt is printed. The function then reads one word at a time from the input buffer until the end of line is reached. If the first character of any word is a semicolon, then the remainder of the line is a comment which is ignored as input. When end of line is reached, if any acceptable words have been read, or there are default values given, then they are returned as the value of the function; otherwise, the function continues reading from the next line. If any word read is a ?, and there is no ? in GOODLIST, and HELPMSG is not NOHELP, then the help message is printed and the user is prompted again.

In responding to a user’s request for help, the function recognizes two types of request. When the ? read is the last word typed on the line, the function prints all the help information available, consisting of HELPMSG, a list of the acceptable responses in GOODLIST, and any messages associated with each of the entries in GOODLIST. When the ? is followed by one or more other words on the line, then the function selects the items in GOODLIST which match any of the words entered, and prints just those responses with their associated messages. In
either case, the input buffer is cleared before printing the message.

When a word has been entered, it is interpreted according to GOODLIST and CHECKFN, and then, if acceptable, the appropriate value is included in the input list. When the user's reply is not acceptable, then the input buffer is cleared, the user is reprompted, and reading continues from the next line.

**value** - a list of one or more acceptable answers read from a line of user input. Normally, each acceptable answer is a word read from the input line; however, when CHECKFN is used to interpret the user responses, each of the answers may be any non-NIL value returned by CHECKFN.

```
INPNAMES (PROMPTMSG HELPMSG GOODLIST CHECKFN NOSHIFTFLG DEFAULT KEYPROMPTFLG SPELLCORRECTFLG AUTOSELECTFLG) = lambda-spread
```

Input an atom from the user's terminal.

**PROMPTMSG** - an optional message to be used to prompt the user. This may be in any form acceptable to PRIN1FORMS. The message is only used if the user has not yet entered any characters in the input buffer, or when beginning a new line of input. Otherwise the message is not printed.

**HELPMSG** - an optional message to be used to respond to a user's request for help. If provided, this must be a LISP form or list of forms which may be interpreted by PRIN1FORMS. Normally, the message will be printed whenever the user asks for help by typing ? in response to the prompt. However, if the message is the single atom NOHELP, or if the atom ? is included as the name of an entry in GOODLIST, then a ? typed by the user will be returned as the value of the function, and no message will be printed. In addition, the user may ask for limited help about specific entries in GOODLIST, in which case, this message will not be used.

**GOODLIST** - an optional list of entries which, in conjunction with CHECKFN, define the acceptable user responses. When the list is omitted and no CHECKFN is provided, any response is considered acceptable.

Each entry in the list must consist of one or more names, which will be determined by the function NAMEOF, and an optional message associated with the names. A name must be an atom. Atoms enclosed in <...>’s are considered to be meta-symbols.
representing some other set of names. With the exception of \(<\text{CR}>\), meta-symbols in GOODLIST represent atoms which must be interpreted by CHECKFN. If the meta-symbol \(<\text{CR}>\) is included as the name of an entry in GOODLIST, then a null response will be allowed as an acceptable answer by the user. An atom which matches, or partially matches, one of the GOODLIST entries will not be considered as a candidate to be interpreted by CHECKFN. If the atom \(?\) is included as the name of an entry in the list, then a \(?\) typed by the user will be returned as the value of the function.

When an entry is a list, the tail of the list beginning with the second element constitutes the message for that entry. A message associated with any entry will be used to describe that entry. The message may be supplied in any form acceptable to PRIN1FORMS.

CHECKFN - an optional function which will be used to check any atoms entered by the user which do not match an entry in GOODLIST. The function will be applied to any atom typed by the user which is either a meta-symbol or which does not match an entry in GOODLIST. If the value returned by the function is non-NIL, then that value will be taken as a valid reply by the user.

NOSHIFTFLG - a flag signifying that input may be in lower case. When this flag is NIL, all input will be converted to upper case.

DEFAULT - an optional default value to be returned if the user enters no reply. If a default value is given, it must be compatible with GOODLIST or CHECKFN.

KEYPROMPTFLG - a flag determining when a set of abbreviated keys is to be used in the prompt to the user. If no value is given, then the value of MAXKEYLISTLENGTH is used. If the flag is NIL, then no abbreviated list will be used. If the flag is a small integer, then that is the maximum size keylist that will be abbreviated. Any other non-NIL value allows any number of keys to be abbreviated.

SPELLCORRECTFLG - a flag determining when spelling correction should be used to match entries in GOODLIST. If the flag is non-NIL, then any user answer will be matched to GOODLIST with spelling correction applied if necessary. Normally, when this flag is NIL, no spelling correction is applied to the user responses.

AUTOSELECTFLG - a flag specifying that the function should automatically select and return the proper value whenever: a. GOODLIST contains a single entry; and b. there is no CHECKFN.

effects - if the input buffer is empty or if the user is beginning a
new line of input, then the prompt message, default value, and abbreviated key list are printed; otherwise no user prompt is printed. The function then reads one word from the input buffer. If the first character of the word is a semicolon, or the word is a string, then the remainder of the line is a comment which is ignored as input. If the word read is a ?, and there is no ? in GOODLIST, and HELPMSG is not NOHELP, then the input buffer is cleared and a help message is printed. In either case, the function reprompts and continues reading from the next input line.

In responding to a user's request for help, the function recognizes two types of request. When the ? read is the last word typed on the line, the function prints all the help information available, consisting of HELPMSG, a list of the acceptable responses in GOODLIST, and any messages associated with each of the entries in GOODLIST. When the ? is followed by one or more other words on the line, then the function selects the items in GOODLIST which match any of the words entered, and prints just those responses with their associated messages.

When an atom has been entered, it is interpreted according to GOODLIST and CHECKFN, and the appropriate value is returned as the value of the function. When the user's reply is not acceptable, then the input buffer is cleared, the user is reprompted, and reading continues from the next line.

value - any acceptable answer typed by the user. Normally, an acceptable answer is an atom; however, when CHECKFN is used to interpret the user responses, the value may be any non-NIL value returned by CHECKFN. All string values will be converted to atoms before they are returned. In addition to returning a value which is the interpretation of the user's input, the function also sets the global variable USERINPUT to be any word which was actually entered by the user, before it is interpreted by the function.

INPYN (PROMPTMSG HELPMSG DEFAULT) - lambda-spread

Input yes/no as T/NIL from the user's terminal.

PROMPTMSG - a message to be used to prompt the user. This may be in any form acceptable to PRIN1FORMS. The message is only used if the user has not yet entered any characters in the input buffer, or when beginning a new line of input. Otherwise the message is not printed.

HELPMSG - a LISP form or list of forms to be interpreted by PRIN1FORMS.
if ? or ?? is typed by the user. When provided, the forms will be used to respond to a user's request for help.

DEFAULT - an optional default value to be used if the user enters no reply. If a default value is given, it must be either of the atoms YES or NO; any other value will be interpreted as no default.

value - T if the user typed YES; NIL if the response was NO.

SELECTAPPLY (LST!s NAMEFN!s APPLYFN!s ULFLG!s GNAME!s NLSTFLG!s OTHERFLG!s METAFLG!s)
- lambda-spread

User selection of items from a list, with an optional function applied to the selected values.

LST!s - a list of items.

NAMEFN!s - a function to be used by NAMEOF to extract the name of each item in LST.

APPLYFN!s - a function to be applied to each list of items as they are selected by the user. This should be a function of one argument, which is a list of elements. If no function is supplied, then no action will be taken on the selected items.

ULFLG!s - a flag to indicate that selection options should be printed in upper and lower case.

GNAME!s - a generic name describing the items in LST. Eg: tasks, names, files, etc.

NLSTFLG!s - a flag which may be used to suppress the initial printing of the list of selections. If this is a small integer, then it is the maximum number of items which may be contained in the list and still have each of their names listed in the initial prompt. If the list contains more items than this number, then none of the names will be printed initially. If the flag is any other non-NIL value, then the initial list of names will be suppressed completely.

OTHERFLG!s - a flag which, when non-NIL, specifies that the user may be allowed to enter names of items not in the initial list. Ordinarily the user responses are limited to selecting items in the given list.

METAFLG!s - a flag to indicate that the name of one or more of items
may be a meta-symbol identifiable by the function \textsc{metasymbolp}.

effects - prompts the user to select items from \textsl{LST}s and applies the \textsc{applyfn}s to each list of items as they are selected.

value - a list of all the items selected by the user.

\textsc{selectfile} (\textit{name\_sf} \textit{mode\_sf} \textit{lst\_sf} \textit{nulflag\_sf})

- \textsc{lambda}\textsc{-spread}

Prompt the user for a file name allowing file recognition on typein.

\textit{name\_sf} - an atom which is the name of a global variable, referred to as a file variable, whose value will be set to the name of the file selected. When \textit{nulflag\_sf} is not set, the initial value of this variable will be used to determine any default file name. If this is not an atom, then it will be evaluated.

\textit{mode\_sf} - The input/output mode of the file; either \textsc{input} or \textsc{output}. If none is given then \textsc{output} is assumed. If this is not an atom, then it will be evaluated.

\textit{lst\_sf} - A list of default extensions. If none is given, then the list of extensions associated with \textit{name\_sf} in \textsc{selectfilelist} will be used. If this is not a list, then it will be evaluated. When the list of file extensions contains the atom \textsc{print}, and there is a task name in the global variable \textsc{controltask}, then the primary default extension will be \textsc{print\_task}.

\textit{nulflag\_sf} - A flag which, when set, allows the entry of a null file name by the user, thereby preventing the use of any default file name. This is always evaluated.

effects - the user is prompted to enter the name of a file for the file variable. The function allows standard file recognition, and sets the value of the file variable to be the name of the file.

value - the fully recognized file name entered by the user.
7.4. Miscellaneous Functions

COPYFROMSCRATCHFILE (LISTFLG) - lambda-spread

COPYFROMSCRATCHFILE returns a string or list (depending on the value of LISTFLG) constructed from the text in the scratch file determined by OPENSCRATCHFILE.

COPYTOSCRATCHFILE (VALU FLG) - lambda-spread

COPYTOSCRATCHFILE copies VALU (a string or list) to a scratchfile. It returns the name of the scratchfile. If FLG is T, then the scratchfile is left open; otherwise it is closed before exit.

CURRENTFILENAMEOF (FILE EXTNLST) - lambda-spread

Get the complete name of a currently existing file.

FILE - A partial file name. ie: directory, extension, and version are optionally included in the name.

EXTNLST - A list of any preferred extension names which the file should have. A single name will be treated the same as a list of that name.

effects - When a file name is found on a directory other than those in DIRECTORYLIST, then the new directory name is added at the end of the list.

value - the fully recognized file name of a currently existing file. If the input file name is already the fully recognized file name of a current file, then that file name will be returned. Otherwise, the function will look at each directory in DIRECTORYLIST, giving preference to any directory in the input file name, to find a file with the specified file name and a valid extension. The valid extensions are all those included in the preferred list plus the extension of the file name.

EOFCHECK (FILE) - lambda-spread

FILE - a file name
effects - reads a single s-expression from file and closes file if an
end indication is found

value - NIL if end is found; otherwise a list whose first element is
the s-expression read from file

NAMEOF (ITEM FN) - lambda-spread

Get the symbolic names of a data structure.

ITEM - a LISP s-expression representing some item.

FN - an optional function which may be used to extract the part of ITEM
which is to be used as its name. When no FN is given, then ITEM
is assumed to be either a name or list whose first element is
either a name or list of names.

value - the name or list of names which denote the item. A name may be
either an atom or a string.

NAMEOF1 (ITEM FN) - lambda-spread

Get a single symbolic name for a data structure.

ITEM - a LISP s-expression representing some item.

FN - an optional function which may be used to extract the part of ITEM
which is to be used as its name. When no FN is given, then ITEM
is assumed to be either a name or list whose first element is
either a name or list of names.

value - a single name which denotes the item. If the item has a list of
names, then only the first is returned. A name may be either an
atom or a string.

OPENSRCATCHFILE (ACCESS) - lambda-spread

OPENSRCATCHFILE opens a scratch file with the appropriate access mode:
INPUT, OUTPUT, BOTH, or APPEND. If a file is already opened in
the appropriate access mode, then that file is used with its
file pointer reset appropriately.
PRIN1CMNT (CMNT!pc COL!pc FILE!pc) - lambda-spread

Print a LISP comment.

CMNT!pc - a LISP comment or comment pointer.

COL!pc - an optional integer number. The number of spaces to indent any continuation lines if the comment cannot be completely printed on the present line. The default is 0 spaces.

FILE!pc - the output file.

effects - the text of the comment is printed on the output file.

Printing starts at the current position of the output pointer, with each expression in the comment printed to the output line separated from the previous expression by a space. A single TERPRICCHAR character in the comment will cause immediate termination of the current print line; otherwise, printing proceeds to the end of a line and continues at the specified column on a continuation line. A single EVALTEXTCHAR character in the comment will cause the following expression to be interpreted as a special print command by the function PRIN1COMMAND, with any non-NIL value which is returned from the function being printed. The presence of TEXTINDEXMARKER in the text being copied is interpreted as identifying text strings to be used as index entries. These index entries will be collected and either placed in TEXTINDEXLIST or written in SCRIBE commands on the output file, depending on the value of SCRIBEINDEXFLAG. Upon completion of printing, the output pointer will be positioned immediately following the last character printed.

-----

PRIN1COMMAND (VAL!pcd COL!pcd FILE!pcd) - lambda-spread

VAL!pcd - a special printing command, or an expression to be evaluated.

COL!pcd - an integer number. The number of spaces that printing is to be indented when a new line is started. The default value is 0 spaces.

FILE!pcd - the output file.

effects - Any special printing commands are interpreted as follows:

- n - a positive integer. Tab to the specified column.
- -n - a negative integer. Print n spaces.

Any other expression will be evaluated for its effect.

value - the results of evaluating a command expression.
PRIN1FORMS (LST!pf FILE!pf COL!pf) - lambda-spread

Interpret a list of expressions for printing.

LST!pf - a LISP form, or a list of forms, to be evaluated and/or printed. Each of these forms will be interpreted so as to produce printed output to the designated file.

FILE!pf - the output file to be used for printing.

COL!pf - an integer number. The number of spaces to indent any continuation lines required to print the forms. If none is given, then 0 is assumed.

effects - in the input forms, strings will be printed by PRINNEXT; comments will be printed by PRIN1CMNT; all other forms will be evaluated for their effect, and any non-NIL value will be printed by PRINNEXT. Comments which are QUOTEd will be treated as simple comments, without regard for the QUOTE. The global variable TEXTINDEXLIST will be set to contain any text strings which are marked for indexing in the comments printed.

value - T if any of the forms are printed; NIL otherwise

PRINNEXT (X COL FILE SPACE) - lambda-spread

X - an s-expression to be printed.

COL - the optional column number in which printing should begin whenever a new line is required.

FILE - the name of the file on which the s-expression will be printed.

SPACE - an optional integer number specifying the number of spaces to be printed following the s-expression. When this is not given, 2 spaces are printed. If this number is negative, then it means that the s-expression should be printed left justified in a fixed length field of the specified length.

effects - if X will fit on the current output line, it is printed followed by the specified number of spaces; otherwise, the current output line is terminated and X is printed on the next line starting in the column specified. Whenever a new line is started, if there are index entries in TEXTINDEXLIST and indexing information is being prepared for SCRIBE (ie.
SCRIBEINDEXFLAG is non-NIL), then a SCRIBE command for each index entry will be printed on a separate line before beginning the next line of text.

.....

TEXTED (EDITOR VALU HASHFILE) - lambda-spread

EDITOR - the name of a text editor to be called: one of the editors named in TEXTEDITORLIST, or some other text editor.

VALU - a string of text or a comment to be edited, or an atom to be used to retrieve the text from a given hash file.

HASHFILE - the name of a hash file containing the text to be edited, if the text is contained in a hash file.

.....

UPLOAD (NAME AGAINFLG LNFLG) - lambda-spread

.....
7.5. Backchain Utility Functions

---

CLEARDATA () - lambda-spread

Clear the data structure, RESET

---

CLEARTRACE () - lambda-spread

Delete the trace information from the property lists of the objects and attributes. The trace information shows which attributes have been previously setup as subgoals, and therefore don't need to be traced again.

---

RESETCHAIN () - lambda-spread

Delete all chaining information from the objects and attributes.

---

RETBC (OBJ ATT WHICH) - lambda-spread

Returns the first value, ie the last value added with modify. Unless the third arg is (quote ALL) then returns all the values.

OBJ - object
ATT - attribute
WHICH - ALL | NIL

---
7.6. Task Execution Functions

For many of the system tasks, calling for the execution of the task invokes the execution of a corresponding task function. Further description of these task functions follows:

---

UXRECORD () - lambda-spread

Start/finish a typescript file.

---

UXDESIGN () - lambda-spread

Create the user program.

---

UXSAVE () - lambda-spread

Save the user program on a file.

---

UXGETSYS () - lambda-spread

Get the user program from a file.

---

UXCHECK () - lambda-spread

Check syntax of the user program.

---

UXEXAMINE () - lambda-spread

Examine parts of the user system.

---

UXRESET () - lambda-spread

Reset the user system.

---

UXRUN () - lambda-spread

---

October 81 1981
Execute the user program.

This is the function responsible for controlling execution of the AGE user program. In programs specifying backward chained rule interpretation, execution of the program is performed by the function RUNBACKCHAIN. Otherwise, execution of the user program is performed by the function AGESYSTEMCONTROL.

**UXSYSOUT ()** - lambda-spread

Save everything (ie. SYSOUT)

**UXHELP ()** - lambda-spread

Examine descriptions of the current list of possible tasks.

**UXLISP ()** - lambda-spread

Enter lower level LISP.

**UXQUIT ()** - lambda-spread

Exit from LISP.

**UXTASKS ()** - lambda-spread

Show a complete list of all tasks available and examine their descriptions.

**HYPO ()** - lambda-spread

Hypothesis building function.

**PRECREATEDNODES? ()** - lambda-spread

Allow the user to predefine some of the nodes at a particular level.
KNOWLEDGESOURCES () - lambda-spread

Specify the LHS evaluator and weight adjustment functions.

GETUSERFNS () - lambda-spread

Load and/or enter user defined functions.

EDITUSERFNS () - lambda-spread

Edit the user defined functions.

KERNEL () - lambda-spread

MACRO-EVENTSPEC () - lambda-spread

MACRO-EXPECTSPEC () - lambda-spread

INPUTFORMAT () - lambda-spread

Input data specification function.

INITIAL () - lambda-spread

TERMINATE () - lambda-spread
POSTPROCESSING () · lambda-spread

ATTR-ONLY () · lambda-spread

Acquisition function for attribute-only datatype.

Effect--The attribute only datatype consists of a list of attribute names, unique names. These attributes will have values associated with them when the system is run. The attributes will also be used to chain the rules in a backward manner. The ATTR-ONLY acquisition function also asks for input data names. Attributes used as input data are marked with (INPUTDATA? T) on their property list and their names are put on the INPUTDATANAMES list.

EXPLICITOBJ () · lambda-spread

Acquire explicit object data.

Effect--Set up an explicit object data structure. This datatype consists of objects and associated attributes. Each object must have a unique name. The attributes must be unique with respect to other attributes of the same object. The attributes and later their values are stored on the property list of the objects.

VARIABLEOBJ () · lambda-spread

Acquisition of the variable level datatype.

Effect--Sets up a datatype allowing variable objects. The user specifies levels of abstraction for specific objects to be generated at runtime. The user also specifies attributes to be associated with those objects. All the objects created at one level will have the same attributes, and those attributes can not be used at any other level.

BCEVALUATOR () · lambda-spread

Specify the LHS evaluator and weight adjustment functions.
UXEND () - lambda-spread

End selection.

.....
7.7. Global Variables

# LHSEVALUATOR - The value of # LHSEVALUATOR is a function name. This function will be applied to the LHS of a rule to determine if the rule should be executed.

# VALUEADJUSTER - The value of # VALUEADJUSTER is a function name or function expression. The function will be evaluated to determine the adjusted value associated with a link or attribute. The function to be executed may use the global variables $NODE, $ATTR, and $VAL. If a function is not given, then no value adjustment is performed.

$ATTR - The value of $ATTR is an attribute or link name. This is the attribute or link under current consideration. It is available for use during the execution of the user's value adjustment function.

$NODE - The value of $NODE is the name of a hypothesis node. This is the node under current consideration. It is available for use during the execution of the user’s value adjustment function.

$TALLY - The value of $TALLY is a probability value. This is the probability associated with the most recently evaluated LHS of a rule. It is available for use during the execution of the user’s value adjustment function. It is set by $ANDMIN.

$VAL - The value of $VAL is a value associated with $ATTR and $NODE. This is a value of the attribute or link under current consideration. It is available for use during the execution of the user’s value adjustment function.

AGECONTROLVARLST - The value of AGECONTROLVARLST is a list of the names of all the AGE variables containing user program control information.

AGEHYPOVARLST - The hypothesis structure consists of the values of a number of global variables; plus each of the atoms in the list which is the value of the global variable NODELEVELS; with the atom’s value and its associated property list of attribute-value pairs.

AGERULECONTROLVARLST - The value of AGERULECONTROLVARLST is a list of the names of all the AGE variables containing user program control information which is used to control the execution of rules in the knowledge sources.

AGESCHEMA - a global variable whose value is the name of the root node in the design hierarchy. The value of each schema property for the variable is a comment describing the meaning of the property.
AGESCHEMALIST - a list of the names of all the nodes in the design hierarchy.

AGESCHEMAPROPLIST - a list of the names of all the properties used to describe a node in the design hierarchy.

ALLTRACEFLAGS - an association list describing every trace mode available in AGE. In this list, the name of a trace mode is associated with a description of what happens when the mode is turned on.

ATTLIST - The value of ATTLIST is a list of attributes used in a backchained user system.

AUTOLOADFLAG - A flag to enable/disable the automatic search for, and loading of, functions which are undefined when they are initially referenced. When the flag is NIL, no attempt will be made to locate undefined functions; otherwise, the system will search for the definition of any undefined functions, looking first in any of the currently loaded files, and then in the files on LOADLIST.

AUTOLOADLIST - a list of any functions which have been loaded by the automatic loading function, LOADABLE. Each time a function is loaded because of a call to LOADABLE, the name of the function will be added to the front of the list.

BBSAVE - This is used for saving the blackboard.

BREAKLIST - a list of break descriptors. Each descriptor is a list of 3 elements:

(<when> <where> <which>)

Following are the allowable options for specifying break descriptors:
(The vertical lines are multi-choice branches, horizontal ones are single choice branches)
CHAININGCASE - There are three datatypes currently available in the AGE Backchain Model. These three datatypes correspond to the three backchaining methods (or cases) implemented in AGE. The chaining cases are:

1A ATTR-ONLY attribute only
2A EXPLICITOBJ explicit objects with attributes
2B VARIABLEOBJ variable objects grouped by levels, with attributes

In the ATTR-ONLY case, each attribute is treated as a variable with a value. In EXPLICITOBJ, the objects names have property list associated with them and the attributes and their values are stored as property names and values. The VARIABLEOBJ case uses levels with property lists to store the objects associated with each level and the attributes applicable to objects on that level. Attributes can only be used on one level. Each object also has a property list with the attributes and their values.

Rules are chained by the attributes and objects they refer to, either just the attribute name for ATTR-ONLY, the object and attribute names for EXPLICITOBJ, and the level, object and attribute names for
VARIABLEOBJ.

CONTROLRULES - The value of CONTROLRULES is a list of <control - rule>s which will be used during program execution to select the next step type to be performed.

CONTROLTASK - a push down list of the names of the tasks which are currently being executed.

DEFINEDLINKS - The value of DEFINEDLINKS is a list of all names that will be used as links by the user's program.

DEFINEDSLOTS - The value of DEFINEDSLOTS is a list of all the names that will be used as attributes by the user’s program.

EVENT - The value of EVENT is the current control information for events.

EVENTLIST - A list of <step - entry>s which are the events remaining to be processed.

EVSAVE - This is used for saving the event list.

EXPECTATION - The value of EXPECTATION is the current control information for expectations.

EXPECTATIONLIST - A list of <step - entry>s which are the expectations remaining to be processed.

EXSAVE - This is used for saving the expectation list.

EXTENSIONCHAR - a string containing a single character which is the extension separator character. This is the character which separates the generic file name from the file extension in a fully recognized file name.

FOCUSEVENT - The value of FOCUSEVENT is a step from the event list. This is the current focused event during the execution of an event step.

FOCUSEXPECTATION - The value of FOCUSEXPECTATION is a step from the expectation list. This is the current focused expectation during the execution of an expectation step.

FOCUSGOAL - The value of FOCUSGOAL is a step from the goal list. This is the current focused goal during the execution of a goal step.

GOAL - The value of GOAL is the current control information for goals.
**GOALIST** - A list of **step-entry**s which are the goals remaining to be processed.

**GSAVE** - used to save the goal list.

**HISTORYLIST** - a list of history entries which are used to record the run as it progresses. In order for the explanation system to have enough information to explain later how AGE did what it did, HISTORYLIST must have a structure that reflects the way AGE changes the **blackboard**. Namely, each KS has separate rules, each rule has separate PROPOSE's, and each PROPOSE has separate changes on its attr-value slot. This is done using a nested set of globals that correspond directly to the nesting of change actions and the nesting of the functions that do the changing. This nesting is needed only for event, expectation, or goal actions. The history list also has entries for KS calls, list focuses, and steps. These require nothing fancy. The variables involved in the nesting are:

**HISTORYENTRY**, which collects one entry on HISTORYLIST, i.e., one rule firing entry;

**HLACTENRY**, which collects one entry on HISTORYENTRY, i.e., one PROPOSE entry of the rule;

**HLCHGENTRY**, which collects one entry in HLACTENTRY, or one change entry in the attr-value slot of the rule.

The structure of HISTORYLIST follows:

**HISTORYLIST** :: = (histor yentry ...)  

**<historyentry>** :: = (<HL #> <entrytype> <name> <number> <changes>)  

**<HL #>** :: = the name used to identify the entry. This is an atom created by packing HL onto a unique number.  

**<entrytype>** :: = **KS | STEP | FOCUS | EVACTION | EXACTION | GACTION**  

**<changes>** :: = (<actionentry> ...)

**<actionentry>** :: = (<changentry> ...)

**<changentry>** :: = (<changetype> <node> <attr> <value> <remlinks> <addlinks>)

**<changetype>** :: = **LIST | ADD | MODIFY | SUPERSEDE**
The use of `<name>`, `<number>`, and `<changes>` for a given
`<entrytype>` is explained in the following diagram:

```
ENTRYTYPE ::=
  KS -- <name>::= <KSname>
  -- <number>::= <invokation#>

  STEP -- <name>::= <new steptype>
  -- <number>::= <step#>
  -- <changes>::= <old steptype>

  FOCUS -- <name>::= <list name>
  -- <number>::= <list element ID>
  -- <changes>::= <value of element focused>

  EVACTION -- <name>::= <KSname>
  -- <number>::= <rule#>
  -- <changes>::= (<actionentry> ...)
  <actionentry>::= (<changentry> ...)

  EXACTION -- <name>::= NIL
  -- <number>::= NIL
  -- <changes>::= same as for EVACTION

  GACTION -- same as for EXACTION
```

The use of `<node>`, `<attr>`, `<value>`, `<remlinks>`, and
`<addlinks>` for a given `<changetype>` is explained in the
following diagram:

```
CHANGETYPE ::=
  LIST -- <node>::= <list name>
  -- <attr>::= ADD
  -- <value>::= <list element ID>

  ADD -- <node>::= <level name>
  -- <attr>::= <node name>

  MODIFY -- <node>::= <node name>
  -- <attr>::= <attribute name>
  -- <value>::= <value used in change>
  -- <remlinks>::= links that were removed
    because of the change.
  -- <addlinks>::= links that were added
    because of the change.

  SUPERSEDE -- same as MODIFY
```
Also,

\[
\text{<list element ID>'}s \text{ are EV10, G3, etc.}
\]

\[
\text{<steptype>::=} \text{<list name>::=} \text{ EVENT} \quad \text{ EXPECTATION} \quad \text{ GOAL}
\]

\text{HLACTENTRY} - collects any changes due to one PROPOSE. When finished, HLACTENTRY is pushed onto the changes field of HISTORYENTRY.

\text{HOSTNAME} - the name of the computer facility which is the current host for this program.

\text{INDATA} - an association list which associates the user's data names with the actual data structures. This is an old form of INPUTDATA. This variable name may be used in old user files, but new systems no longer create it.

\text{INPUTDATA} - The value of INPUTDATA is an association list which associates the user's data names with the actual data structures.

\text{INPUTDATANAMES} - All input data names are stored on the global variable INPUTDATANAMES. The user can define the names of input data at the time the user program is specified. In addition to specifying the names of data items, the user may also write prompts for any of those data items. These data items are calls to AGE input functions and are stored on the property list of the data item name under the property AGPROMPT. If a data item has no prompt the user will be asked for its value at the beginning of a run. For data items with prompts, the user specified prompt is evaluated to retrieve the data value the first time it is referenced during the program's execution.

The function ASKFORDATA asks for input data values at the beginning of each run. It skips data items with their own prompts. \$DATA is used in the rules to retrieve data values. \$DATA first checks to see if a value is available and returns it if it is. If no value has been entered yet, the prompt is evaluated and the answer is saved on INPUTDATA. This list of data names and values can be saved on a file after it is created by ASKFORDATA.

\text{INV # INFO} - an associated list storing the number of times each KS has been invoked.
**KEEPHISTORYFLAG** - This tells whether the user wants the history list kept.

**KSLIST** - The value of KSLIST is a list of the names of the knowledge sources which exist. Each name is a global variable whose value is the knowledge source.

**LEVELLIST** - The value of LEVELLIST is a list of the level names of all the hypothesis levels currently used by the user program.

**LINKS&OPPOSITES** - The value of LINKS&OPPOSITES := ( { <link> , <link> } ... )

a list of all pairs of links which have opposites and are therefore 2-way links.

**LOADCOMFLAG** - A list of the file extension names which are preferred when selecting LISP program files to be loaded. This should contain COM when compiled files are to be loaded in preference to others. When the list is NIL, no attempt will be made to find file names with any particular extension.

**LOADLIST** - A list of the names of files containing program information relevant to the current context. This list will be used to determine where to look for any information which is undefined at the time it is needed for program execution. When an executing program encounters undefined information, the contents of each of the files in this list will be loaded until the desired information is found or there are no more files available. The names included in the list should be generic file names only, since the selection of directory and extension names will be controlled by DIRECTORYLIST and LOADCOMFLAG respectively.

**LOADTYPEFLAG** - an atom designating the type of file loading to be performed by the system. This value is used as the LDFLG parameter when loading user files.

**MAXKEYLISTLENGTH** - a flag which specifies the default value for the maximum number of allowable responses to a prompt that can be displayed as abbreviated keys when prompting a user. If the flag is NIL, then no abbreviated keys should be displayed in the prompt to the user. If the flag is a small integer, then it specifies the maximum number of keys that should be displayed. Any other non-NIL value means that all allowable user responses should be displayed as abbreviated keys in the prompt.

**MINSELECTDISPLAY** - a flag which specifies the default value for the number of items which will invoke interactive selection mode when displaying a list of items to the user. Fewer items will be displayed without interaction.
NEWNODENAME - The value of NEWNODENAME is the name of the node in the hypothesis structure which was created most recently during the execution of the user program.

NLSAVE - used for saving local variables.

NODELEVELS - The value of NODELEVELS : = (level ...)  
This is a list of the names of each of the hypothesis levels, where (car NODELEVELS) is the highest or top level.

The property list of level : =  
(NUM <availnum>  
LLIST (level [ <node> ] ...)  
{ <attribute> : <attrvalue> }  
{ <linkname> : <linkvalue> } ...)

where:

<availnum> : = a positive integer used to generate unique node names at the given level.

<node> : = a node name; an element of the hypothesis. This is either the name of a pre-created element or a name generated from the level name concatenated with an integer determined by <availnum>. Initially, before running the user program, only pre-created nodes appear in this list.

<attribute> : = an attribute name. One of the names in DEFINEDSLOTS.

<attrvalue> : = ([ <atval> ] ...)

where:

<atval> is a value of the attribute.

<linkname> : = a link name. One of the names in DEFINEDLINKS.

<linkvalue> : = ([ <node> ] ...)

Initially <attrvalue> is NIL and there are no links associated with any nodes. These may be added to the hypothesis structure by the execution of the user's program.

The property list of node is the same as that of level except that it does not contain either NUM or LLIST properties.

NODELIST - The value of NODELIST is a list of objects used in the
EXPLICITOBJ chaining case, a list of levels for the VARIABLEOBJ case, and NIL for ATTR - ONLY.

NOTRACEFUNCTIONS - a list of the names of functions whose names should be omitted from the system trace printed by SYSTRACE.

RULESUSEDONCE - The value of RULESUSEDONCE is:

\[ ((\text{<ksname> <rulenumber>}) \ldots) \ldots) \]

This is a list associating the name of each knowledge source in the user's program with a list of the rule numbers from that knowledge source which have already been executed. This variable is only used for those knowledge sources whose hit strategy is ONCEONLY.

SAVEFILEXT - an atom which is the file extension to be used for SYSOUT files. This name must not contain either of the separator characters used for extension or version separators.

SCRIBEINDEXFLAG - a flag used to control the treatment of expressions in the text which are marked, using TEXTINDEXMARKER, as expressions to be indexed. When the flag is NIL, index expressions will be included in the list, TEXTINDEXLIST, and no other information will be printed; otherwise, the index expression will be followed by a SCRIBE index command containing the expression. When the flag is non-NIL, it is assumed to be a SCRIBE environment to be used to print the index expression, eg: b for boldface.

SEPRCHARS - a list containing all the characters which separate lisp expressions in a file. The first character in the list must be the EOL character.

SITENAME - the name of the computer facility which is the current host for this program.

STEPSAVE - used for saving the step.

STEPTYPELIST - The value of STEPTYPELIST is a list of the names of each of the valid step types defined in AGE. Execution of an AGE user program involves the sequential execution of inference generation steps. Each individual step may be any one of the defined step types.

Each step type name is a global variable which has a value which is the <control - info> for that particular step type. Each step type name is also defined as a function which is the step execution function for that step type.

Also, associated with each step type during user program execution are a step list and the current focussed step. The step list is a global variable, <steptype>LIST, whose value is a list of any steps remaining to be processed for the individual step type. The current focussed step is a global variable, FOCUS<steptype>, whose value is...
the step currently being processed by the AGE system.

**SYSTEMEXITCOM** - the operating system command to exit from a lower fork EXEC.

**SYSTEMNAME** - the name of the operating system under which the current program is running.

**SYSTEMNULLFILE** - the name of the system NULL device in the current operating system.

**TERMINATIONCONDITION** - The value of TERMINATIONCONDITION is a function name or function expression. The function designated will be evaluated and the value returned will be used to decide when to end execution of the user program. Program execution will terminate as soon as the function returns a non-NIL value.

**TIMEFLAG** - a flag which calls for the printing of timing information for the execution of tasks.

**TRACEINFO** - a list of the names of the trace modes which are currently turned on in AGE.

**UA.AVAIL** - a file pointer to the first free block in the UNITS file.

**UA.BUMPFLG** - a flag which is set whenever it is time to attempt to remove some inactive units from core. This is normally set by a GCTRP break whenever the number of free list cells goes below some minimum.

**UA.BUMPN** - the number of units to be paged out when more space is needed by the system. If this is a number between 0 and 1, then it is interpreted as a fraction of the number of units currently in memory.

**UA.ERRMSG** - the text of the most recent error message.

**UA.ERRNO** - the error number associated with the most recent error message.

**UA.ERTOKEN** - the name of the item causing the latest error message, or a list of several such names.

**UA.FILENAME** - the name of the units network which is currently active. The name must be an atom which does not contain any file directory or extension specifications.

**UA.FILES** - a list of file description elements which give information about each of the files which may be part of an active knowledge base. Each element is a list containing file extension, file variable name, and required status in the form: (extension variable flag)

**UA.FIXED-FIELDS** - a list of the permanent fields in each slot.
UA.GCTRPCOUNT - the number of free list words, which when the system contains less than this number, will cause the flag to be set to remove units from memory.

UA.HASHFILE - the name of any text hash file associated with the current active knowledge base.

UA.HEADERPOS - a file pointer to the position of the header record for the block being written.

UA.MINGOODBLOCKSIZE - the minimum acceptable size of a free block in the UNITS file. No smaller free blocks will be created.

UA.NONUSERS - a list of the names of directories in UA.USERS which are not connected with individual users of the system.

UA.PFLG - a flag indicating when units paging is allowed to occur. When NIL, no units paging is allowed.

UA.RELFILE - the name of the relations file associated with the current active network.

UA.ROVER - a file pointer to the current active free block in the UNITS file. This pointer allows rotating access to the free blocks to improve storage management.

UA.TIMESTAMP - the current relative access number for unit slot access. This number is incremented each time the slots of a different unit are accessed. It is used to mark units so as to determine relative access times for the paging algorithm.

UA TRACEFLG - a flag controlling the printing of various messages about system events. When the flag is non-NIL, these messages will be printed.

UA.TRAILPOS - a file pointer to the position of the trailer record for the block being written in the UNITS file.

UA.UNIT - the name of the unit which is the current focus of attention of the program. This is the unit whose slots have been accessed most recently.

UA.UNITFILE - the name of the UNITS file in the current active network.

UA.UNITLIST - a list of the names of each of the units in the current active knowledge base.

UA.UNITPAGEPROPS - a list of the property names which contain the paged
information for a unit. These properties only have a value when the unit is in memory. Otherwise they are all NIL. The first property on this list may also be used as a flag to determine when the unit is in core, as it is always non-NIL when in use.

**UA.UNITPROPS** - a list of all the property names which constitute the data structure of a unit. This includes both paged and relation information.

**UA.UNITRELPROPS** - a list of the property names which contain the relation information for a unit. The first property name on the list must always be included in the property list of each unit, as it is used as a flag to indicate the existence of a unit. Other property name are optional.

**UA.UNITSIN** - the total number of times that the slots of any unit have been loaded into memory from the UNITS file.

**UA.UNITSOUT** - the total number of times that the slots for any unit have been written into the UNITS file.

**UA.UPFLG** - a flag indicating when the system should keep track of the numbers of units paged in and out. When NIL, paged units will not be counted.

**UA.USERS** - a list of directories which may contain units knowledge bases. The first directory in the list is expected to contain any help files to be accessed.

**USED # 'S** - The value of USED # 'S is an integer number. This is a counter used to provide a new and unique number to label system steps. The value is the number of the last step label generated.

**USEDHL # 'S** - This keeps track of the history list numbers used.

**USEDLIST # 'S** - This keeps track of the numbers used for the event list, expectation list, and goal list.

**USEDSTEP # 'S** - This is used to keep track of step numbers.

**USERFNS** - The value of USERFNS is a list of the names of the functions which have been defined by the user as a part of the user program.

**USERINITFN** - The value of USERINITFN is a function name or function expression. This function will be evaluated at the start of the user program to determine the name or names of the user's initial knowledge source. The value must be either a single name or a list of names of knowledge sources. The knowledge sources named will be invoked to begin the user's program.

**USERPOSTPROCESSINGFN** - The value of USERPOSTPROCESSINGFN is a function name or function expression. This function will be evaluated at the end of
the execution of the user program.

**VERSIONCHAR** - a string containing a single character which is the version separator character. This is the character which separates the file extension from the file version in a fully recognized file name.
Appendix I
Glossary

Action - is the operating part of a production rule. It also refers to an operation performed by a production rule. Each action represents an implication or conclusion to be drawn in a particular situation.

Applicability - refers to the conditions which must be met in order to fire a rule, i.e., execute the RHS of the rule. Because of the wide range of possibilities available for the definition of applicability, AGE asks the user to define applicability in the form of a function to serve as the LHS Evaluator.

The term also refers to the conditions for the selection of a knowledge source. The event types specified in the precondition define the applicability of the set of rules in the KS. A KS is invoked when one of its preconditions matches the currently focussed event which represents the situation.

Attribute - a property associated with a hypothesis node or unit.

Attribute-only - one of the types of chaining data available in the AGE backchain framework. With the attribute-only chaining data, the names of the attributes are used to chain the rules. In this simple case each attribute will be a global variable and the value of the attribute, the value of the variable. See the PUFF implementation using EMYCIN for an example.

Backchain - a framework for constructing a user program. It is useful for building a program that uses production rules as its primary knowledge representation, and goal directed backward chaining of rules as its inference engine.

In the backchain framework a user program consists of the following components:

- the data structure;
- the knowledge sources; and
- the control information.

Blackboard - a data structure designed to hold input data, intermediate results and solutions. It is a hierarchical data structure that is organized to represent the problem domain as a hierarchy of analysis levels. In addition, the blackboard holds dynamic control related data that are accessible to the user program. In a forward driven framework, the process of incremental changes to the blackboard is viewed as a general process of hypothesis formation. Consequently, the blackboard is also referred to as the hypothesis structure.

Chaining - refers to the process of finding a line of reasoning which results in a particular goal by
working backwards from the goal. This is the process used in the AGE backchain framework. The ability to chain the rules depends on the type of chaining data used. There are three types of chaining data available in the AGE backchain framework:

- attribute-only,
- fixed object-attribute, and
- variable object-attribute.

**Condition** - The left hand side of a rule specifies a set of conditions for the applicability of a rule. Each condition may be any LISP expression which will be evaluated as a predicate.

**Control information** is for directing the invocation of the KSs, and focussing the attention within the user program. There are several functional components grouped under the heading of Control. They are:

- Input,
- Initialization,
- Kernel,
- Termination, and
- Post-processing.

Control information also refers to information specified within the KSs which manipulate rules.

**Control macro** is simply a collection of preset control variables and preprogrammed control functions. Because the control components are full of details that are confusing to the novice users, AGE provides two rather simple, prepackaged control structures, called control macros. They are useful for event-driven and expectation-driven control.

**Domain knowledge** - An AGE user program represents the user’s knowledge and model of the task domain. Every program must contain some knowledge represented as one or more sets of production rules called knowledge sources (KSs). Additional knowledge may be contained in the program in the form of:

- control information defining the way KSs and their rules are to be interpreted;
- input data definitions defining information to be supplied by a user during program execution;
• a hierarchically organized global data base, or blackboard;

• a semantic network (using the UNITS package); and,

• LISP functions and variables used by the program. AGE contains many predefined functions and variables designed to be used within the user program. In addition to these, a user program may contain any additional definitions specified by the user.

Element - a named node (object) in the hypothesis structure (or blackboard) that represents an aggregation (summary, interpretation, integration, abstraction, etc.) of lower level hypothesis elements. The properties of the elements are represented in the form of attribute-value pairs that are meaningful at the particular hypothesis level.

Event - one of the AGE step types. It also refers to an event type.

Event type - a user-specified name that summarizes the actions taken by a rule.

Expectation - one of the AGE step types.

Fixed object-attribute - one of the types of chaining data available in the AGE backchain framework. With the fixed object-attribute chaining data the object name must be given explicitly in the conditions and actions of the rules, i.e. as a literal atom. This does not allow for the specification of relationships between objects. The same attribute may be associated with several objects. Each object-attribute pair must be unique and these pairs are used to chain the rules.

Focus - When an element is chosen to be processed, it is termed focussed. The event name associated with the focussed element is called the focussed event.

Forward driven - a framework for constructing a user program. It is useful primarily for building programs whose designs are rooted in the Blackboard Model [Lesser 77].

In the forward driven framework a user program consists of three major components. These components are:

• the blackboard;

• the knowledge sources; and

• the control information.

Framework - is a distinct conceptual skeleton around which an AGE user system is defined. Each AGE framework may share some facilities with other frameworks, but one or more facilities of one framework are unique to that framework and generally are incompatible with the unique facilities of other frameworks. A framework is conceptually similar to a prefabricated house -- the basic design decisions have
been made, but many of the final decisions have been left to suit the occupants' tastes and needs. There are two frameworks currently supported by AGE:

- the Forward driven framework, and
- the Backchain framework.

Goal - one of the AGEl step types.

History list - a list of entries which are used to record the execution of a user program as it progresses. This information is needed by the explanation system to provide explanations for how AGE did what it did.

Hit strategy - refers to the properties of a knowledge source which identify how its rules are to be executed. SINGLE or MULTIPLE determines whether the KS should be scanned for more than one rule whose conditions are satisfied. When SINGLE, the execution of the rules in the KS will stop after the first rule is fired. When MULTIPLE, the KS context will be preserved until all relevant rules have been fired. ONCEONLY refers to whether each of the rules of the KS should be executed only once during the running of the user program, or whether they may be executed every time the KS is invoked and the LHS is true. In a ONCEONLY KS, the LHS of a rule may be evaluated many times until its value is true. Then the actions in the rule are executed once and the rule is never evaluated again.

Hypothesis - The combined process of KS selection and incremental changes to the blackboard is viewed as a general process of hypothesis formation. Consequently, the data structure that holds the input data and the intermediate, as well as the final, results is referred to as the hypothesis structure.

The hypothesis structure in a forward driven program is assumed to be hierarchically organized. The hierarchy may be "flat", consisting, in the extreme case, of input data and a single inference level drawn from the data. Depending on the problem, there may be more than one hierarchy needed to define the hypothesis structure. Each hierarchy consists of hypothesis elements integrated by links that represent support from above and support from below; the links are called expectation links and reduction links, respectively.

Inference - Execution of an AGE user program involves the sequential execution of inference generation steps. Each individual step may be any one of the defined step types.

Inference generation consists of executing the step execution function for the type of step selected. If executing the focussed step type does not cause any action to be taken, then alternate step types are tried.

Inference generation occurs during the execution of the various step execution functions.
Kernel - the central control information of a forward driven program. The functions of the kernel control information are:

- to select an item on the blackboard to process next, and
- to invoke KSs appropriate to that item and consistent with the goal of the program.

Thus, the kernel control component consists of two conceptually distinct subcomponents whose separate functions are focus of attention and inference generation:

Knowledge source - Knowledge necessary to accomplish the goals of the program is represented in production rules. These rules are organized into one or more sets of rules called knowledge sources (KSs). A set of rules that belong together is called a knowledge source.

A KS is a labeled set of production rules that are a priori deemed to belong together. For example, a model-like KS may contain rules that are organized around some objects or concepts; a data-oriented KS may contain rules that generate hypotheses from data elements.

A KS can be thought of as a mega-chunk of knowledge or, because of its organization, as a mega-rule. Each has associated with it preconditions that indicates the specific situations under which the rules in the KS are applicable. The preconditions define the applicability of the KS. A KS is invoked when one of its preconditions matches the currently focussed event.

KS - a knowledge source.

Left hand side - of a rule is the part whose function is to determine if the rule is applicable in a given situation.

LHS - the left hand side of a production rule.

LHS evaluator - a user specified method by which the 'LHS' of a rule will be evaluated to determine its applicability.

Multiple hit - a hit strategy for the execution of rules in a KS. For a MULTIPLE KS, the context of the KS will be preserved until all relevant rules have been fired.

Node - a single element or node in the hypothesis structure.

Once only - a hit strategy for the execution of rules in a KS. This strategy refers to whether each of the rules of the KS should be executed only once during the running of the user program, or whether they may be executed every time the KS is invoked and the LHS is true. In a ONCEONLY KS, the LHS of a rule may be evaluated many times until its value is true. Then the actions in the rule are executed once and the rule
is never evaluated again.

**Post-processing** - consists of all processing which occurs after the termination of rule execution during execution of the user program. This processing consists of the execution of a single post-processing function.

**Post-processing function** - a function which will be evaluated after the termination of the user program. When termination of the user program occurs, the user post-processing function is invoked and execution ends.

**Pre-created element** - a named node of the hypothesis which is part of the original definition of the hypothesis structure created by the user; and not created during program execution.

**Precondition** - Each KS has associated with it preconditions for its invocation that indicates the specific situations under which the rules in the KS are applicable. The preconditions are one or more event types. The event types specified in the precondition define the applicability of the set of rules in the KS. A KS is invoked when one of its preconditions matches the currently focussed event.

**Production rule** - knowledge represented as a statement of the form:

```plaintext
if condition then action
```

**RHS** - the right hand side of a production rule.

**Right hand side** - of a rule is the part which specifies the actions to be taken when the rule is executed.

**Rule** - a production rule.

**Single hit** - a hit strategy for the execution of rules in a KS. When SINGLE, the execution of the rules in the KS will stop after the first rule is fired.

**Step** - a single inference generation step during the course of execution of a user program.

**Step type** - Execution of an AGE user program involves the sequential execution of inference steps. Each individual step may be any one of the defined step types. Some of the step types currently defined in AGE are: EVENT, EXPECTATION, and GOAL.

The name of each step type is a global variable which has a value which is the control information for that particular step type. The name of each step type is also defined as a function which is the step execution function for that step type.

Also, associated with each step type during user program execution are a step list and the current focussed step. The step list is a global variable, `<steptype>LIST`, whose value is a list of any steps remaining to be processed for
the individual step type. The current focused step is a global variable, FOCUS<steptype>, whose value is the step currently being processed by the AGE system.

**Task** - generally refers to a set of operations which the system can perform. Each task is identified by a name associated with the operations to be performed. Generally, the operations to be performed are embodied in a LISP function to be executed. The name of a function which executes a given task is usually found by prefixing the task name with UX; although in some cases, the task function may have the same name as the task. Also, instead of specifying a function to be executed, a task may simply call for another list of tasks to be made available to the user.

**Termination** - The user must specify the condition under which the program is to terminate; for example, the occurrence of some specific event. Termination of the AGE user program occurs either:

- when no action was taken during the previous inference generation step, or

- the user supplied termination condition has a non-NIL value.

Program execution will terminate as soon as the function specified by the user as the termination condition returns a non-NIL value. When the user program terminates, the user post-processing function is invoked and execution ends.

**Units** - Knowledge about the problem domain can also be represented in an object centered representation as implemented by the Units Knowledge Base. In this representation, the description of the objects, both conceptual and actual, and the relationships among them, are maintained in a semantic network.

**User program** - An AGE user program represents the user's knowledge and model of the task domain. This program will be interpreted by the AGE system to produce whatever results are sought by the user. Every program must contain one or more sets of production rule called knowledge sources (KSs). Additional knowledge may be contained in the user program in the form of:

- **control information** defining the way rules and KSs are to be interpreted;

- **input data** definitions defining information to be supplied by a user during program execution;

- a hierarchically organized global data base, or **blackboard**;

- a **semantic network** (using the UNITS package); and,

- **LISP functions and variables** used by the program. AGE contains many predefined functions and variables designed to be used within the user program. In addition to these, a user program may contain
any additional definitions specified by the user.

Value adjustment - In AGE, value adjustment is an operation performed on each link or attribute computed during the execution of the action of a rule. The actual operation performed is specified in a value adjustment function as part of the user program. As each attribute or link is computed by an action, the function will be evaluated to determine the adjusted value associated with the link or attribute. The function to be executed may use the global variables $NODE, $ATTR, and $VAL. If a function is not given, then no value adjustment is performed.

Variable object-attribute - one of the types of chaining data available in the AGE backchain framework. With the variable object-attribute chaining data, objects are not stated explicitly in each rule, but are computed as the rules are evaluated. AGE requires that the attributes be unique to each level of objects for this method (same constraint as in MYCIN). With this restriction, AGE can determine the level of an object from the attribute, and by keeping track of the context, compute the relevant object within that level. Thus the same rule may be applied to several objects (or contexts).

Weight - A change inferred by the RHS of a rule can have associated with it as its value, a probability, or some informal weight, that reflects the confidence in the implication to be drawn under the particular conditions specified in the LHS. This informal probability will need to be reflected in the attribute-value pair of the affected hypothesis element when the rule is executed. The a priori specified weights in the rules, and the probability already assigned to the attribute of the hypothesis element to be modified need to be integrated. This can be accomplished in many ways -- one possible method is described in EMYCIN and another in KAS. To allow maximum flexibility, AGE asks the user to provide the necessary computation in the form of a function to perform value adjustment.
Appendix II
Reserved Function Names

Following is a list of all names used as functions in AGE.

$ADDNODE
$AND
$ANDMINBC
$DEFAULTPOST
$FINDD
$SPEQ
*AGESFNS
*GPIO
ACHIEVECHK
ADDPROP
ADDTOUSERFNFILE
ADDTOGAALLIST
ADJPAIRNUM
AGEBREAK
AGESYSTEMCONTROL
ALLIDS
AMF
ANALYZEFN
ANALYZEFUNARG
ANALYZEQUOTE
APF
APV
ASIS
ASKFORTRACEINFO
ATLFX
ATTR-ONLY
AZCOMS1
AZFIX
AZPRINNEXT
AZSETUP
BACKCHAIN-KS
BACKCHAINER2B
BACKCHAININTERPRETER2B
BCASIS
BCEVALUATOR
BCINPUTFORM
BCSAVELEVELOBJATTRINFO
BLKS
CANCELNETWORK
CHECKRESTRICTION
CLEARBADVALUES
CLEARTRACE
CLOSEDP
COMBINATIONS
COMMONLENGTH
COMSVAR
COPYBYTES1W
COPYNETWORK

$ADJWT
$ANDBC
$DATA
$DEFAULTTERM1
$LEVELNODES
$SUPERSEDE
*DOC
*GPLISP
ACTNCHK
ADDSASSOC
ADDTOEVENTLIST
ADDIVARSLST
AGEADJUSTWT
AGEBREAKEXAMINE
AGEEXPLAIN
ALLNODES
ANALYZEASSEMBLE
ANALYZEFNCALL
ANALYZEFUNCTION
ANALYZERECORD
APFH
APVF
ASKFORBREAKLIST
ASSIGNNAMETO
ATLFX2
ATVALCHK
AZCOMS2
AZGETDEF
AZPROMPT
AZUPDATEVARS
BACKCHAINER1A
BACKCHAININTERPRETER1A
BASENAMEOF
BCASKFORDATA
BCEXPLAIN
BCMODIFY
BCSUPERSEDE
BREATHFIRST
CATCH10
CLASS?
CLEARDATA
CLEARUSERSYSTEM
CLOSENETWORK
COMBINATIONSOF2
COMPACT
CONNDIRNAME
COPYFROMSCRATCHFILE
COPYPROPS

$ADJWTBC
$ANDMIN
$DEFAULTDOMAIN-RULES
$DEFAULTTERM2
$MODIFY
$VALUE
*FUTIL
*SYSCTL
ADDATM
ADDSymbol
ADDTOEXPECTATIONLIST
ADJACENT
AGEBACKTRACE
AGEBREAKMESS
AGEEXPLAIN1
AlLOPTIONSOF
ANALYZECOMMENT
ANALYZEFORM
ANALYZEPROG
ANCESTOR?
APPREV
ASGNSTORAGE
ASKFORDATA
ASSOCTAIL
ATTACHLIST
AYZ
AZEXPRP
AZMAP
AZPROPCOM
AZY/N
BACKCHAINER2A
BACKCHAININTERPRETER2A
BCADD
BCDATAFILECOMMS
BCHYPO
BCRHSMANAGER
BEGINOUTFILE
BUTLAST
CHANGEROLE
CLASSIFY
CLEARHYPO
CLEARVALUES
COLLECTEVENTS
COMBINE
COMS
COPUNIT
COPYGROUP
COPYRIGHT
COPYTOSCATCHFILE
COUNTTEXT
CURRENTFILENAMEOF
DATATYPEP
DBGLHISFAIL
DEFPREM
DELETE
DELETEROLE
DELETPROP
DESCENDANTS
DESIGNING
DIRECTORYOF
DISJOIN
DISPLAYTASK
DSPLYEX?USER
DWIMLOAD?
DWIMNEWVERSION
EDITBOOLEAN
EDITFNLIST
EDITLISP
EDITSTRING
EDITUNIT
ELOADP
EOFCHECK
EQUALBOOLEAN
EQUALINTERVAL
EQUALSTRING
EQUALUNIT
EVALUATOR
EVSORTFN
EXLISPFILE
EXPECTATION
EXPGIVEACTION
EXPLAINARG
EXPLHSFAIL
EXPNOFIRE
EXPOBJATTR
EXPSAVE
FAULTAPPLY!W
FCV
FILEDESCRIPTIONS
FILEENTRIES
FILEVAR
FILTERLIST
FINDRULE
FIRSTNCHARS
FIX1
FIXOPPLINKS
FNCLUSTER
FNSGROUPS
FUNCTIONS
GCAF
GENLEVEL
GENNEXTINV#
GET-LIST-LENGTH
COPYUNIT
CREATENEWNODE
DATAMEDIA
DATOMIZE
DECLASSIFY
DELIF
DELETEFIELD
DELETESLT
DELUNIT
DESCRIBESYNONYMS
DESIGNSUMMARY
DIRECTROP
DISPLAYFILE
DLOADP
DSPLYEX?USER
FAULTEVAL!W
FCV
FILEDESCRIPTIONS
FILEENTRIES
FILEVAR
FILTERLIST
FINDRULE
FIRSTNCHARS
FIX1
FIXOPPLINKS
FNCLUSTER
FNSGROUPS
FUNCTIONS
GCAF
GENLEVEL
GENNEXTINV#
GET-LIST-LENGTH
COPYVAL
CREATETEXTHIERARCHY
DATATYPEP
DAYSTR
DEEPMEMB
DELETEBYCAR1
DELETEFUNCTION
DELETEUNIT
DEPTHFIRST
DESRIPTUSER
DESIGNADVICE
DISJOIN
DISPLAYLIST
DREMLAST
DSUBSET
DWIMLOADFN
EDITBADVALUES
EDITEXPR
EDITINTERVAL
EDITSLOT
EDITTEXT
EFLISTFN
ENDOUTFILE
EVENTCHK
EXECUTECHK
EXPCHECK
EXECFIRE
EXPKSINV
EXPLAINVAR
EXPLISTADD
EXPSETVAR
EXTENSIONOF
FCF
FIFO
FILENAMEOF
FILETEXT
FILLINBCDATA
FINDONMAP
FIRSTN
FIX-SLOTREFS
FIXFIXEDOPPLINKS
FLOADFNS
FNGRAPH
FUNCTIONP
GC
GENHLSMMRY
GENNEWNODENAME
GET&DEFFUN
GET-REF
<p>| GETACTIONS | GETACTION1 | GETALLOBJCHG | GETALLOBJCHG | GETALLOBJCHG |
| GETALLRULESPCS | GETBCACTIONS | GETBCCONDITION | GETBCCONDITION |
| GETBCRULE | GETCHARS | GETCOMMENTTEXT | GETCOMMENTTEXT |
| GETCONDITIONS | GETCONTROLRULES | GETDATAITEMNAMES | GETDATAITEMNAMES |
| GETEVENTC | GETEVENTS | GETEXPECTM | GETEXPECTM |
| GETHISTORYSEEK | GETFILE | GETFILE | GETFILE |
| GETINVOKINGEVENTS | GETFILENAME | GETFILENAME | GETFILENAME |
| GETLEVELSPAN | GETGOAL | GETGOAL | GETGOAL |
| GETLIST | GETHITSTRATEGY | GETHITSTRATEGY | GETHITSTRATEGY |
| GETMATCHES | GETKSINV# | GETKSINV# | GETKSINV# |
| GETOBJATTRCHG | GETLIST# | GETLIST# | GETLIST# |
| GETPROMPTKEYS | GETMATCHLIST | GETMATCHLIST | GETMATCHLIST |
| GETROLE | GETNAMELIST | GETNAMELIST | GETNAMELIST |
| GETRULENUMBERS | GETOBJECT | GETOBJECT | GETOBJECT |
| GETTCREATEFILE | GETGOALS | GETGOALS | GETGOALS |
| GETUSERFNS | GETINTERVAL | GETINTERVAL | GETINTERVAL |
| GETVORD | GETKRNAME1 | GETKRNAME1 | GETKRNAME1 |
| GLOBALCONTEXT | GETLIS | GETLIS | GETLIS |
| GOSRTFN | GETLISTENTRY | GETLISTENTRY | GETLISTENTRY |
| HYPO | GETLITERAL | GETLITERAL | GETLITERAL |
| INITIAL | INITIALIZE | INITIALIZE | INITIALIZE |
| INITUSED#’S | INPNAME | INPNAME | INPNAME |
| INPUTFORMAT | INPNAMES | INPNAMES | INPNAMES |
| INSERTLEVEL | INPNAMESKEY | INPNAMESKEY | INPNAMESKEY |
| INST* | INPYN | INPYN | INPYN |
| INTERATOM | INSERTPAIR | INSERTPAIR | INSERTPAIR |
| INTERNUMBER | INST*? | INST*? | INST*? |
| INTERSAME | INTERDUCKY | INTERDUCKY | INTERDUCKY |
| INTERSTRING | INTERPRETER | INTERPRETER | INTERPRETER |
| INTRANS | INTERSECTF | INTERSECTF | INTERSECTF |
| ISATOM | INTERTEXT | INTERTEXT | INTERTEXT |
| ISINTEGER | INVALUES | INVALUES | INVALUES |
| ISNUMBER | ISBOOLEAN | ISBOOLEAN | ISBOOLEAN |
| ISTEXT | ISINTERVAL | ISINTERVAL | ISINTERVAL |
| KEYPoINTERS | ISSTRING | ISSTRING | ISSTRING |
| KCH | ISUNIT | ISUNIT | ISUNIT |
| KSCHK | KEYWeRDsof | KEYWeRDsof | KEYWeRDsof |
| KSLINKCHK | KSHSCHK | KSHSCHK | KSHSCHK |
| LADDFILE | KSPRECHK | KSPRECHK | KSPRECHK |
| LEFN | LCFT | LCFT | LCFT |
| LEXT | LEVELCHK | LEVELCHK | LEVELCHK |
| LIFO | LFCX | LFCX | LFCX |
| LISTFIELDS | LINESPACE | LINESPACE | LINESPACE |
| LISTSLOTS | LISTID | LISTID | LISTID |
| LOADFILEOF | LINKNDKCHK | LINKNDKCHK | LINKNDKCHK |
| LONUIN | LOADUSER | LOADUSER | LOADUSER |
| LOOKUPPARAM | LOOKATREFERENCE | LOOKATREFERENCE | LOOKATREFERENCE |
| LUNU | LOOPCHK | LOOPCHK | LOOPCHK |
| LUNUN | MARKGOAL | MARKGOAL | MARKGOAL |
| MACRO-EVENTSPEC | MARKGOAL | MARKGOAL | MARKGOAL |
| MAKEYLIST | MAKENETWORK | MAKENETWORK | MAKENETWORK |
| MAKERULESPCS | MAKESLOT | MAKESLOT | MAKESLOT |
| MAKEUNITNAME | MAKEUNIT | MAKEUNIT | MAKEUNIT |
| MAKEUNITNAME | MATCHHIERARCHY | MATCHHIERARCHY | MATCHHIERARCHY |</p>
<table>
<thead>
<tr>
<th>Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMLASTPROP</td>
</tr>
<tr>
<td>RENAMEUNIT</td>
</tr>
<tr>
<td>RESETTEXTFILE</td>
</tr>
<tr>
<td>RESTOREVALS</td>
</tr>
<tr>
<td>RHSMANAGER1</td>
</tr>
<tr>
<td>RPLACN</td>
</tr>
<tr>
<td>RULECHK</td>
</tr>
<tr>
<td>RULEUSEDONCE</td>
</tr>
<tr>
<td>SASSOCF</td>
</tr>
<tr>
<td>SAVEDATA</td>
</tr>
<tr>
<td>SAVEINDATA</td>
</tr>
<tr>
<td>SAVETASKINFO</td>
</tr>
<tr>
<td>SAVEXINFO</td>
</tr>
<tr>
<td>SELECTFILE</td>
</tr>
<tr>
<td>SETDATE</td>
</tr>
<tr>
<td>SETMODIFIER</td>
</tr>
<tr>
<td>SETUPSTORAGE</td>
</tr>
<tr>
<td>SHOWFILES</td>
</tr>
<tr>
<td>SIZEORDERP</td>
</tr>
<tr>
<td>SLOTMATCH</td>
</tr>
<tr>
<td>SPEC</td>
</tr>
<tr>
<td>SPLIT</td>
</tr>
<tr>
<td>STATIC</td>
</tr>
<tr>
<td>STRICTBOOLEAN</td>
</tr>
<tr>
<td>STRICTINTERVAL</td>
</tr>
<tr>
<td>STRICTNUMBER</td>
</tr>
<tr>
<td>STRICTTEXT</td>
</tr>
<tr>
<td>SUBNAME</td>
</tr>
<tr>
<td>SUPERIORS</td>
</tr>
<tr>
<td>SYSINIT</td>
</tr>
<tr>
<td>TASKFUNCTION</td>
</tr>
<tr>
<td>TERMINATE</td>
</tr>
<tr>
<td>TEXTED</td>
</tr>
<tr>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>TOPLEVELSLOT?</td>
</tr>
<tr>
<td>TRACEPRINT</td>
</tr>
<tr>
<td>TRANSFERTEXT</td>
</tr>
<tr>
<td>TTEST</td>
</tr>
<tr>
<td>TVBOOLEAN</td>
</tr>
<tr>
<td>TVINTERVAL</td>
</tr>
<tr>
<td>TVNUMBER</td>
</tr>
<tr>
<td>TVTEXT</td>
</tr>
<tr>
<td>UA-BUMPUNITS</td>
</tr>
<tr>
<td>UA-DELETEROLE1</td>
</tr>
<tr>
<td>UA-ERMMSG</td>
</tr>
<tr>
<td>UA-GETREL</td>
</tr>
<tr>
<td>UA-GETSLOT</td>
</tr>
<tr>
<td>UA-INITGLOBALVARS</td>
</tr>
<tr>
<td>UA-MAKESLOT</td>
</tr>
<tr>
<td>UA-OKSLOT?</td>
</tr>
<tr>
<td>UA-PACKAGFN?</td>
</tr>
<tr>
<td>UA-PICKSIZE</td>
</tr>
<tr>
<td>UA-PUTREL</td>
</tr>
<tr>
<td>UA-PUTTRAILER</td>
</tr>
<tr>
<td>UA-RELEASEBLOCK</td>
</tr>
<tr>
<td>RENAMESCHEMAPROP</td>
</tr>
<tr>
<td>RESETCHAIN</td>
</tr>
<tr>
<td>RESOLVE-EXPR</td>
</tr>
<tr>
<td>RETBC</td>
</tr>
<tr>
<td>RLOADP</td>
</tr>
<tr>
<td>RSCAN</td>
</tr>
<tr>
<td>RULEEDITOR2</td>
</tr>
<tr>
<td>RUNBACKCHAIN</td>
</tr>
<tr>
<td>SAVECHAIN</td>
</tr>
<tr>
<td>SAVEDESIGN</td>
</tr>
<tr>
<td>SAVERULES</td>
</tr>
<tr>
<td>SAVEUSERFNS</td>
</tr>
<tr>
<td>SEARCHQUOTE</td>
</tr>
<tr>
<td>SELEXPRS</td>
</tr>
<tr>
<td>SETFMODES</td>
</tr>
<tr>
<td>SETPROPCOMMENTSW</td>
</tr>
<tr>
<td>SETUPTEXTFILE</td>
</tr>
<tr>
<td>SHOWREFS</td>
</tr>
<tr>
<td>SLOT-MENTIONS</td>
</tr>
<tr>
<td>SLOTMSG</td>
</tr>
<tr>
<td>SPEC*</td>
</tr>
<tr>
<td>SPLITUNIT</td>
</tr>
<tr>
<td>STIW</td>
</tr>
<tr>
<td>STRICTEXPR</td>
</tr>
<tr>
<td>STRICTLISP</td>
</tr>
<tr>
<td>STRICTSTRING</td>
</tr>
<tr>
<td>STRICTUNIT</td>
</tr>
<tr>
<td>SUBSET</td>
</tr>
<tr>
<td>SYMBOLIZE</td>
</tr>
<tr>
<td>SYSRESET</td>
</tr>
<tr>
<td>TEMPINIT</td>
</tr>
<tr>
<td>TERPRI?</td>
</tr>
<tr>
<td>TEXTPROF</td>
</tr>
<tr>
<td>TMPLTCHK</td>
</tr>
<tr>
<td>TOLEVELUNIT?</td>
</tr>
<tr>
<td>TRACKINGSLOT?</td>
</tr>
<tr>
<td>TRANSFERUNIT</td>
</tr>
<tr>
<td>TTYOUT</td>
</tr>
<tr>
<td>TVEXPR</td>
</tr>
<tr>
<td>TVLISP</td>
</tr>
<tr>
<td>TVSTRING</td>
</tr>
<tr>
<td>TVUNIT</td>
</tr>
<tr>
<td>UA-CHANGEROLE</td>
</tr>
<tr>
<td>UA-DELREL</td>
</tr>
<tr>
<td>UA-GCTRAP</td>
</tr>
<tr>
<td>UA-GETRELFILE</td>
</tr>
<tr>
<td>UA-GETSLOTFILE</td>
</tr>
<tr>
<td>UA-LOADUNIT</td>
</tr>
<tr>
<td>UA-MAKESCALE</td>
</tr>
<tr>
<td>UA-MAKESTANDARD</td>
</tr>
<tr>
<td>UA-OPENFILE</td>
</tr>
<tr>
<td>UA-PATCHDELSLOT</td>
</tr>
<tr>
<td>UA-PROGENYSLOT?</td>
</tr>
<tr>
<td>UA-PUTRELFILE</td>
</tr>
<tr>
<td>UA-PUTUNIT</td>
</tr>
<tr>
<td>UA-RELEASEBLOCK1</td>
</tr>
<tr>
<td>RENAMESLOT</td>
</tr>
<tr>
<td>RESETLINE</td>
</tr>
<tr>
<td>RESOLVE-GROUP</td>
</tr>
<tr>
<td>RHSMANAGER</td>
</tr>
<tr>
<td>ROLE?</td>
</tr>
<tr>
<td>RULEBEFOREP</td>
</tr>
<tr>
<td>RULEFIREDP</td>
</tr>
<tr>
<td>SAMESETF</td>
</tr>
<tr>
<td>SAVECONTROL</td>
</tr>
<tr>
<td>SAVEHYPOTHESIS</td>
</tr>
<tr>
<td>SAVESLOTINFO</td>
</tr>
<tr>
<td>SAVEVALS</td>
</tr>
<tr>
<td>SELECTAPPLY</td>
</tr>
<tr>
<td>SETCREATOR</td>
</tr>
<tr>
<td>SETLOADFILE</td>
</tr>
<tr>
<td>SETUPBINDINGS</td>
</tr>
<tr>
<td>SFCOC</td>
</tr>
<tr>
<td>SIGN</td>
</tr>
<tr>
<td>SLOT?</td>
</tr>
<tr>
<td>SORTSLOTS</td>
</tr>
<tr>
<td>SPECIALIZE</td>
</tr>
<tr>
<td>SPRTCHK</td>
</tr>
<tr>
<td>STRICTATOM</td>
</tr>
<tr>
<td>STRICTINTEGER</td>
</tr>
<tr>
<td>STRICTLIST</td>
</tr>
<tr>
<td>STRICTTABLE</td>
</tr>
<tr>
<td>STRINGTOMINT</td>
</tr>
<tr>
<td>SUBSYSYED</td>
</tr>
<tr>
<td>SYSCHK</td>
</tr>
<tr>
<td>SYSTRACE</td>
</tr>
<tr>
<td>TERMINALVALUE?</td>
</tr>
<tr>
<td>TESTPRINCR</td>
</tr>
<tr>
<td>TEXTRCH</td>
</tr>
<tr>
<td>TOLEVELFUNS</td>
</tr>
<tr>
<td>TOPVALUE</td>
</tr>
<tr>
<td>TRANSFERLIST</td>
</tr>
<tr>
<td>TRIMNIL</td>
</tr>
<tr>
<td>TVATOM</td>
</tr>
<tr>
<td>TVINTERGER</td>
</tr>
<tr>
<td>TVLIST</td>
</tr>
<tr>
<td>TVTABLE</td>
</tr>
<tr>
<td>UA-ASSIGNBLOCK</td>
</tr>
<tr>
<td>UA-COMISSUE</td>
</tr>
<tr>
<td>UA-DELSLOT</td>
</tr>
<tr>
<td>UA-GETLISPFILE</td>
</tr>
<tr>
<td>UA-GETRELREC</td>
</tr>
<tr>
<td>UA-HIERORDER</td>
</tr>
<tr>
<td>UA-MAKEREL</td>
</tr>
<tr>
<td>UA-MAKELEVELSLOT</td>
</tr>
<tr>
<td>UA-OPENUNITFILE</td>
</tr>
<tr>
<td>UA-PATCHRENAME</td>
</tr>
<tr>
<td>UA-PUTHEADER</td>
</tr>
<tr>
<td>UA-PUTSLOTFIELD</td>
</tr>
<tr>
<td>UA-PUTUNITFILE</td>
</tr>
<tr>
<td>UA-RENAMEUNIT</td>
</tr>
</tbody>
</table>
Reserved Function Names

UA-SETDATATYPE
UA-TIMESORT
UCONTROL
ULOAD
ULOADE
UM-NEED-TV?
UNBOUNDVARS
UNIONF
UNIT?
UNITNAMESORT
UPDATEFNGROUP
UPDATETEXTIERARCHY
USEDP
USPELLFIX
UXBROWSE
UXDESIGN
UXEXAMINE
UXGENMAN
UXHELP
UXMKFILE
UXPAGES
UXQUIT
UXRETURN
UXSCHEMA
UXTASKS
UXUTD
VALIDINPUT?
VARCHK
VERSIONOF
WHYDID
WLF
WRITEOUT
XCHK
XFERMAPTEXT

UA-SETROLE
UA-UNRELATE
UDDCHK
ULDADC
ULOADR
UNANALYZE
UNCHANGEDSLOT?
UNIONQ
UNITMATCH
UNSYMBOLIZE
UPDATELCAMESMACROS
UPDATETEXTINDEX
USERDIRNAME
UNIT
UXCF
UXDOC
UXFGRP
UXGETSYS
UXLISP
UXMKSYS
UXPFMAP
UXRECORD
UXRUN
UXSYSFNS
UXTOP
UXRFC
VALU
VARIABLEOBJ
WHATRULE
WHYDIDN'T
WRITE
WRITEVALUEFILE
XDEFCHK
XFNCHECK

UA-STORERELREC
UCEXECUTE
ULIST
ULOADD
UM-CLEARBADVALUES
UNAYZ
UNDATAMEDIA
UNIT-MENTIONS
UNITMSG
UPDATEF
UPDATESHEMA
UPLOWCASE
USERFUNCTIONP
UXANTF
UXCHECK
UXEND
UXFILESTAT
UXHACKER
UXLOAD
UXMSTF
UXPFPNS
UXRESET
UXSAVE
UXSYSOUT
UXUSERINFO
UXXMNFILE
VALU
VARIABLEOBJ
WHATRULE
WHYDIDN'T
WRITE
WRITEVALUEFILE
XDEFCHK
XFNCHECK
Appendix III
Reserved Global Variable Names

Following is a list of all names used as global variables in AGE:

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LHSEVALUATOR</td>
<td>#VALUEADJUSTER</td>
<td>$ATTR</td>
</tr>
<tr>
<td>$NODE</td>
<td>$TALLY</td>
<td>$VAL</td>
</tr>
<tr>
<td><strong>COMMENT</strong>FLG</td>
<td>**RETEVAL</td>
<td>ADDSPELLFLG</td>
</tr>
<tr>
<td>ADVICENODE</td>
<td>AFTERSYSAUTOCOMMENTS</td>
<td>AGECONTROLVARLIST</td>
</tr>
<tr>
<td>AGEFILELIST</td>
<td>AGEFILENAME</td>
<td>AGEHYPOVARLIST</td>
</tr>
<tr>
<td>AGEREQUIREDFNS</td>
<td>AGERULECONTROLVARLIST</td>
<td>AGESHEMA</td>
</tr>
<tr>
<td>AGESCHEMAMAPLST</td>
<td>ALLOPTIONS</td>
<td>AGESUSERFLAG</td>
</tr>
<tr>
<td>ALLFILES</td>
<td>ALREADYEVALUATEDRULES</td>
<td>ALLTRACEFLAGS</td>
</tr>
<tr>
<td>ALREADYCHAINRULES</td>
<td>ATOMPROPERTIES</td>
<td>ARGPPROMPTLIST</td>
</tr>
<tr>
<td>ASGNSTORAGEFLAG</td>
<td>AUTOLOADLIST</td>
<td>ATTLIST</td>
</tr>
<tr>
<td>AUTOLOADFLAG</td>
<td>BEFORE#</td>
<td>AZCOMFLAG</td>
</tr>
<tr>
<td>BACKCHAINKS</td>
<td>BLANK</td>
<td>BCTRACE</td>
</tr>
<tr>
<td>BEFORE#</td>
<td>BLANKSTRING</td>
<td>BREAKLINE</td>
</tr>
<tr>
<td>BREAKLIST</td>
<td>BROKENFNS</td>
<td>BREAKINGLINE</td>
</tr>
<tr>
<td>CHAININGCASE</td>
<td>CLISPARRAY</td>
<td>BUILDMAPFLAG</td>
</tr>
<tr>
<td>CLISPFIPrettyFLG</td>
<td>CLISPARRAY</td>
<td>CLISPBREAK</td>
</tr>
<tr>
<td>COMMON.NOPRINTSLOTS</td>
<td>CLISPTRACE</td>
<td>COMMENTFLG</td>
</tr>
<tr>
<td>CR</td>
<td>CONTROLRULES</td>
<td>CONTRAFLICTASK</td>
</tr>
<tr>
<td>DBQUOTE</td>
<td>CSYMBOL</td>
<td>CXCV</td>
</tr>
<tr>
<td>DESIGNDONE</td>
<td>DEFINEDLINKS</td>
<td>DEFINEDSLOTS</td>
</tr>
<tr>
<td>DIRECTORYLIST</td>
<td>DESIGNOPEN</td>
<td>DFNFLG</td>
</tr>
<tr>
<td>DRIBBLEFILE</td>
<td>DONE</td>
<td>DONEFLG</td>
</tr>
<tr>
<td>DWIMUSERFORMS</td>
<td>DWIMIFYFLG</td>
<td>DWIMUSERFN</td>
</tr>
<tr>
<td>ERRORMESSAGE</td>
<td>ERRORLIST</td>
<td>ERRORLIST</td>
</tr>
<tr>
<td>EVALUATEDRULES&amp;CONTEXTS</td>
<td>EVALTEXTCHAR</td>
<td>EVALUATIONLIST</td>
</tr>
<tr>
<td>EVSAVE</td>
<td>EVENT</td>
<td>EXSYSLIST</td>
</tr>
<tr>
<td>EXHELPLIST</td>
<td>EX?</td>
<td>FILEDATE</td>
</tr>
<tr>
<td>EXPLANATIONLIST</td>
<td>EXPECTATION</td>
<td>FILEDATE</td>
</tr>
<tr>
<td>EXTENSIONCHAR</td>
<td>EXSAVE</td>
<td>FILELINELENGTH</td>
</tr>
<tr>
<td>FILEDIR</td>
<td>EXUSERLIST</td>
<td>FIELEMAPPING</td>
</tr>
<tr>
<td>FILELIST</td>
<td>FILEHIERARCHY</td>
<td>FILEPKGFLG</td>
</tr>
<tr>
<td>FILENAME</td>
<td>FILEMAPLIST</td>
<td>FILEREFSRCHLIST</td>
</tr>
<tr>
<td>FILERTBL</td>
<td>FILEPCT</td>
<td>FIRSTSESSION:SYLLABUS</td>
</tr>
<tr>
<td>FIRSTFLG</td>
<td>FILEREF</td>
<td>FOCUSGOAL</td>
</tr>
<tr>
<td>FOCUSEXPECTATION</td>
<td>FIRSTSESSION</td>
<td>FUNCTIONNAME</td>
</tr>
<tr>
<td>GLOBALVARS</td>
<td>GOAL</td>
<td>GOALLIST</td>
</tr>
<tr>
<td>GOALRULENO</td>
<td>GOLE</td>
<td>GROUPLIST</td>
</tr>
<tr>
<td>GSAVE</td>
<td>GTSIGN</td>
<td>GVVEC</td>
</tr>
<tr>
<td>HACKEROPTIONS</td>
<td>HEADER</td>
<td>HELPDEPTH</td>
</tr>
<tr>
<td>HELPTIME</td>
<td>HERALDSTRING</td>
<td>HISTORYLIST</td>
</tr>
<tr>
<td>HALCTENTRY</td>
<td>HOSTNAME</td>
<td>HYPHEN</td>
</tr>
<tr>
<td>HYPONODEPROPS</td>
<td>HYPONODES</td>
<td>INDATA</td>
</tr>
<tr>
<td>INITFILE</td>
<td>INPUTDATA</td>
<td>INPUTDATANAMES</td>
</tr>
<tr>
<td>INTYPE</td>
<td>INVINFO</td>
<td>KEEPHISTORYFLAG</td>
</tr>
<tr>
<td>KEYMATCHFLAG</td>
<td>KNOWLEDGESOURCENAME</td>
<td>KSEVENTS</td>
</tr>
<tr>
<td>KSLINKS</td>
<td>KSLIST</td>
<td>KSNAMES</td>
</tr>
<tr>
<td>KSNODES</td>
<td>KSVARIABLES</td>
<td>LAMBDAVAR</td>
</tr>
<tr>
<td>Reserved Global Variable Names</td>
<td>Reserved Global Variable Names</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>LASTARGVALUE</td>
<td>LASTDISPLAYLIST</td>
<td></td>
</tr>
<tr>
<td>LASTNODE</td>
<td>LASTDISPLAYLIST</td>
<td></td>
</tr>
<tr>
<td>LINECOUNTLIST</td>
<td>LASTNODE</td>
<td></td>
</tr>
<tr>
<td>LINKS&amp;OPPOSITES</td>
<td>LERLEVELLIST</td>
<td></td>
</tr>
<tr>
<td>LISPXHISTORY</td>
<td>LEVELNODES</td>
<td></td>
</tr>
<tr>
<td>LOADCOMFLAG</td>
<td>LINKEDDFNS</td>
<td></td>
</tr>
<tr>
<td>LPAR</td>
<td>LISPFILE</td>
<td></td>
</tr>
<tr>
<td>MAX.FLOAT</td>
<td>LISTSIGN</td>
<td></td>
</tr>
<tr>
<td>MESSAGELIST</td>
<td>MAX.INTEGER</td>
<td></td>
</tr>
<tr>
<td>MKSYSLIST</td>
<td>MINSELECTDISPLAY</td>
<td></td>
</tr>
<tr>
<td>NEWUSERFLAG</td>
<td>NEWNOFILENAME</td>
<td></td>
</tr>
<tr>
<td>NLDEC</td>
<td>NLAMA</td>
<td></td>
</tr>
<tr>
<td>NODELIST</td>
<td>NLSAVE</td>
<td></td>
</tr>
<tr>
<td>NOSAVEOPTIONS</td>
<td>NOFIXFNLSLIST</td>
<td></td>
</tr>
<tr>
<td>NOTEXT</td>
<td>NOSPELLFLG</td>
<td></td>
</tr>
<tr>
<td>OLDELLEVELNAME</td>
<td>NOTIMEOPTIONS</td>
<td></td>
</tr>
<tr>
<td>PAGEMARK</td>
<td>OLDSCHMAPROP</td>
<td></td>
</tr>
<tr>
<td>PASTREFERENCES</td>
<td>PARACOUNTLIST</td>
<td></td>
</tr>
<tr>
<td>PRECONDITIONS</td>
<td>PERIOD</td>
<td></td>
</tr>
<tr>
<td>PRINTVALUEFLAG</td>
<td>PRETTYHEADER</td>
<td></td>
</tr>
<tr>
<td>PROMPTFILE</td>
<td>PROGVAR</td>
<td></td>
</tr>
<tr>
<td>RECQFLG</td>
<td>PUNCTUATIONCHARS</td>
<td></td>
</tr>
<tr>
<td>RULENAME</td>
<td>RPAR</td>
<td></td>
</tr>
<tr>
<td>RULEUSEDONCE</td>
<td>RULENUMB</td>
<td></td>
</tr>
<tr>
<td>SAVFILEXTEXT</td>
<td>RULEEVENTS</td>
<td></td>
</tr>
<tr>
<td>SAVETASKINFOLIST</td>
<td>SAVESYSOPTIONS</td>
<td></td>
</tr>
<tr>
<td>SCRIBEDOINDEXFLAG</td>
<td>SCRIBECOMMANDCHAR</td>
<td></td>
</tr>
<tr>
<td>SELECTFILELIST</td>
<td>SCRIBEOUBKT</td>
<td></td>
</tr>
<tr>
<td>SETSTORAGELIST</td>
<td>SEPRCASE</td>
<td></td>
</tr>
<tr>
<td>SITENAME</td>
<td>SHOWINPUTVALUEFLAG</td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>SKELNAME</td>
<td></td>
</tr>
<tr>
<td>STRINGCHAR</td>
<td>STEPSAVE</td>
<td></td>
</tr>
<tr>
<td>SYSOUTFILE</td>
<td>SUBSYSRESSCANFLG</td>
<td></td>
</tr>
<tr>
<td>SYSTEMEXITCOM</td>
<td>SYSSOUTRESETVARS</td>
<td></td>
</tr>
<tr>
<td>SYSTEMNULFILE</td>
<td>SYSTEMFUNCTIONS</td>
<td></td>
</tr>
<tr>
<td>TAB</td>
<td>SYSTEMTYPE</td>
<td></td>
</tr>
<tr>
<td>TASKNAME</td>
<td>TAB1</td>
<td></td>
</tr>
<tr>
<td>TERMINATIONCONDITION</td>
<td>TASKPROMPT</td>
<td></td>
</tr>
<tr>
<td>TEXTEDITOR</td>
<td>TERPRICHAR</td>
<td></td>
</tr>
<tr>
<td>TEXTFILE</td>
<td>TEXTEDITORLIST</td>
<td></td>
</tr>
<tr>
<td>TEXTINDEXSTRING</td>
<td>TEXTINDEXLIST</td>
<td></td>
</tr>
<tr>
<td>TEXTREFERENCES</td>
<td>TEXTKEY</td>
<td></td>
</tr>
<tr>
<td>TRACEINFO</td>
<td>TIMEFLAG</td>
<td></td>
</tr>
<tr>
<td>UA.AVAIL</td>
<td>TRACEMODE</td>
<td></td>
</tr>
<tr>
<td>UA.BUMP</td>
<td>UA.BLOCKSIZE</td>
<td></td>
</tr>
<tr>
<td>UA.ERRTOKEN</td>
<td>UA.ERRMSG</td>
<td></td>
</tr>
<tr>
<td>UA.FIXED-FIELDS</td>
<td>UA.FILENAME</td>
<td></td>
</tr>
<tr>
<td>UA.HEADERPOS</td>
<td>UA.GCTRPCOUNT</td>
<td></td>
</tr>
<tr>
<td>UA.MSGFLG</td>
<td>UA.INCORELIST</td>
<td></td>
</tr>
<tr>
<td>UA.RELFILE</td>
<td>UA.INOUSER</td>
<td></td>
</tr>
<tr>
<td>UA.SIZE-INCR</td>
<td>UA.ROVER</td>
<td></td>
</tr>
<tr>
<td>UA.TRAILPOS</td>
<td>UA.TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>UA.UNILIST</td>
<td>UA.UNIT</td>
<td></td>
</tr>
<tr>
<td>UA.UNITRELPROPS</td>
<td>UA.UNITPAGEDPROPS</td>
<td></td>
</tr>
<tr>
<td>UA.UPFLG</td>
<td>UA.UNITSTIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UA.USERS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UA.UNITFILE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UA.UNITPROPS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UEIINDISIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UEIUSERS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UEBREVITY</td>
<td></td>
</tr>
</tbody>
</table>

October 1981
UEHACKER
UM.USERNAME
USEDHL#'S
USEMAPFLG
USERINITFN
USERNAME
UUSCRATCHSTRING
WEDITMACROS

UETRANSMITFLG
UNITFILELIST
USEDLIST#'S
USERDIR
USERINPUT
USERPOSTPROCESSINGFN
VARIABLENAME
WIDEPRINTFLG

UEWITHFLG
USED#'S
USEDSTEP#'S
USERFNS
USERMACROS
 USERSYSTEMNAME
 VERSIONCHAR
  weditmacros

end
Index

#LHSEVALUATOR 10, 19, 29, 86
#VALUEADJUSTER 10, 19, 29, 86

$ADDNODE 53
$ADJVT 53
$ADJVTBCH 53
$AND 7, 17, 53
$ANDBC 54
$ANDMIN 54, 86
$ANDMNBCH 54
$ATTR 20, 29, 86, 108
$DATA 20, 55, 92
$DEFAULTDOMAIN RULES 55
$DEFAULTPOST 55
$DEFAULTTERM1 55
$DEFAULTTERM2 55
$FIND 55
$INITIALIZE 17
$LEVELNODES 56
$MODIFY 56
$NODE 20, 29, 88, 108
$SEQ 56
$SUPERSEDE 56
$TALLY 86
$VAL 20, 29, 86, 108
$VALUE 56

2-way link 6, 93

<action> 9, 10, 11, 12, 19, 26, 27, 34
?actionentry> 35, 90
<action – tail> 10
<action – type> 10
<addlinks> 35, 36, 90, 91
<arg> 26, 27
<attr> 35, 36, 90, 91
<attribute> 5, 13, 26, 28, 94
<attrvalue> 5, 6, 94
<avtnum> 5, 94
<bccondition> 26
<changentry> 35, 90
<changes> 35, 90, 91
<changetype> 35, 36, 90, 91
<collectionrules> 18
<condition> 10, 19
<control – info> 17, 18, 95
<control – rule> 19, 89
<data – attribute> 29
<data – name> 29
<effect> 18, 19, 34
<element> 13, 14
<entrytype> 35, 90, 91
<enum> 18, 19, 34
<event – type> 8
<execution> 10, 14
<expectation> 10, 11
<expr> 19, 34
<goal> 10, 11
<historyentry> 35, 90
<history – record> 37
<hit – strategy> 8, 9
<HL #> 35, 90
<hnum> 37
<hypochange> 18, 19, 34
<inference – levels> 8, 9
<iteration> 10, 11
<key> 11, 12, 13, 14, 27
<knowledge – source> 8, 26
<ksname> 95
<level/node> 9
<level> 5, 6, 9, 28, 94
<lhs> 10, 26
<link> 6, 9, 13, 14, 93
<linkname> 5, 6, 94
<links> 8, 9
<linkvalue> 5, 6, 94
<matcher> 11, 12, 18
<name> 35, 90, 91
<node> 5, 6, 9, 18, 19, 34, 35, 36, 90, 91, 94
<num> 35, 90, 91
<object> 26, 28, 29
<preconditions> 8, 26
<pred> 37
<premise> 26
<proposition> 10, 12
<remlinks> 35, 36, 90, 91
<num> 37
<rule #> 10, 26
<rule> 10, 26
<rulelist> 8, 10
<rulenumber> 95
<seekingmethod> 12, 18
<selectionmethod> 18
<single – or – multiple – hit> 9
<slot> 14
<slotval> 14
<steptype> 19
<steptype LIST> 17, 95
<step – entry> 18, 33, 34, 37, 89, 90
<succ> 37
<support> 18, 19, 34
<type> 18, 19, 34
<value> 9, 11, 12, 13, 14, 26, 27, 28, 35, 36, 90, 91
<values> 11
<variable> 9, 11
<variable – bindings> 8, 9
<variable – name> 9
<when> 87
<where> 87
<which> 87
AGE Reference Manual

ACHIEVE 10, 11, 16
Action 4, 7, 10, 15, 16, 17, 24, 26, 27, 33, 35, 90, 101, 103, 104, 105, 106, 107, 108
Actions 11, 12
Actual change 7
ADD 12, 13
ADDSLOT 12, 13
ADDUNIT 12, 13
Adjusted value 19, 29, 86, 108
AGECONTROLVARLST 86
AGEHYPOVARLST 86
AGERULECONTROLVARLST 86
AGESCHEMA 45, 86
AGESCHEMALIST 45, 86
AGESCHEMAPROPLIST 45, 87
AGESYSTEMCONTROL 31
AGPROMPT 20, 29, 92
ALLTRACEFLAGS 87
Analysis level 101
Applicability 7, 8, 26, 101, 102, 105, 106
Arglist 12, 13, 14
Args 14, 15
ASIS 57
ASKFORDATA 20, 32, 92
ATTLIST 25, 87
ATTR-ONLY 26, 84
Attr-value 12, 13, 27, 28, 35, 90
Attribute 6, 7, 19, 29, 86, 89, 101, 108
Attribute - object 29
Attribute-only 24, 26, 101, 102
Attribute-value pair 4, 5, 8, 86, 103, 108
AUTOLOADFLAG 87
AUTOLOADLIST 87
Backchain 24, 26, 101, 102, 103, 104, 108
Backchain framework 1, 2, 3, 24
Backchain rule 26
BBSAVE 87
BCADD 27
BCASIS 57
BASKFORDATA 38
BCDATA.ATTR 24
BCDATA.FIXED-OBJ 24
BCDATA.VAR-OBJ 24
BCEVALUATOR 84
BCEXPLAIN 39
BCMODIFY 27
BCRHSMANAGER 38
BCSUPERSEDE 27, 28
Best-first 16
Blackboard 1, 2, 3, 4, 7, 15, 34, 87, 90, 101, 102, 103, 104, 105, 107
Blackboard Model 1, 103
Breath-first 16
Break descriptor 87
BREAKLIST 87
Ch.type 12, 13, 27
Chained 24
Chaining 24, 101
Chaining case 29
Chaining data 24, 101, 102, 103, 108
CHAININGCASE 25, 88
CHANGESLOT 12
CLEARDATA 80
CLEARTRACE 80
CLOSENETWORK 21
Comment 11, 12, 13, 14, 15, 27, 28
Component 15, 102
Conclusion 101
Condition 7, 9, 10, 19, 24, 102, 103, 104, 106
Control information 3, 4, 8, 15, 17, 24, 86, 89, 102, 103, 104, 105, 106, 107
Control Macro 16, 102
Control related data 7
Control structure 7
CONTROLRULES 19, 89
CONTROLTASK 89
COPYFROMSCRATCHFILE 75
COPYTOSCRATCHFILE 75
CURRENTFILENAMEOF 75
DATA 9
Data name 20, 92
Data structure 24, 101
Datatype 12, 13, 14
DEFINEDLINKS 6, 89, 94
DEFINEDSLOTS 6, 89
DELETE_SLOT 12, 13, 14, 22
DELETEUNIT 12, 13, 14, 21
Depth-first 16
Design 43
Desired change 8
Directed graph 4
DISPLAYLIST 63
Domain knowledge 1, 4, 102
DISPLAYLIST 64
EDITERROR 51
EDITUSERFNS 83
Element 4, 5, 27, 28, 94, 103
EMYCIN 2, 8, 24, 101, 108
EOFCHECK 75
Ev.type 12, 13
EVALUATOR 83
Event 7, 17, 18, 32, 33, 89, 103, 107
Event list 7, 16, 18, 34, 89, 98
Event step 18, 34, 89
Event type 8, 12, 18, 19, 34, 101, 103, 106
Event-driven 16, 102
Event-driven Control Macro 16
EVENTLIST 18, 33, 89
EVSAVE 89
EXECUTE 10, 14
EXPECT 10, 11, 16
EXPECTATION 17, 18, 33, 34, 89, 103
Expectation <matcher> 11
Expectation Evaluation 16
Expectation expression 11, 18
Expectation function 11
Expectation link 4, 104
Expectation list 8, 16, 18, 34, 89, 98
Expectation step 18, 34, 89
Expectation-driven 16, 102
Expectation-driven Control Macro 16
EXPECTATIONLIST 18, 34, 89
Expected change 8
Explanation system 34, 90, 104
EXPLICITOBJ 84
Expr 11, 12
EXSAVE 89
EXTENSIONCHAR 89
FINDNEWCONTEXT 38
First-in-first-out 16
First-in-last-out 16
Fixed object-attribute 24, 26, 102, 103
Focus 103
Focus of attention 4, 16, 19, 105
FOCUS<steptype> 17, 95
Focused event 18, 34, 89
Focused expectation 18, 34, 89
Focused goal 18, 34, 89
FOCUSEVENT 18, 34, 89
FOCUSEXPECTATION 18, 34, 89
FOCUSGOAL 18, 34, 89
Focussed element 103
Focussed event 7, 16, 101, 103, 105, 106
Focussed step 8, 17, 95, 106
Forward driven 4, 15, 26, 101, 103, 104
Forward driven framework 1, 3
Framework 1, 101, 103
Function 11, 12, 14, 15
GETDATAITEMNAMES 59
GETUSERFNS 83
GOAL 17, 18, 24, 33, 34, 89, 90, 104
Goal <matcher> 12
Goal <seekingmethod> 12
Goal directed backward chaining 2, 101
Goal expression 12, 18
Goal function 12
Goal list 8, 16, 18, 34, 89, 90
Goal rule 24
Goal step 18, 34, 89
GOALLIST 18, 34, 90
GSAVE 90
Hierarchy 4, 104
History list 37, 93, 98, 104
HISTORYENTRY 35, 90
HISTORYLIST 34, 90
Hit strategy 95, 104, 105, 106
HILACTENRY 35, 90
HILACTENTRY 92
HOSTNAME 92
HYPO 82
Hypo-element 12, 13, 19, 34
Hypothesis 19, 34, 104
Hypothesis element 4, 7, 9, 13, 103, 104
Hypothesis formation 4, 101, 104
Hypothesis generator 7
Hypothesis level 4, 5, 6, 9, 13, 93, 94, 103
Hypothesis node 9, 13, 14, 19, 34, 86, 101
Hypothesis structure 4, 5, 7, 16, 86, 94, 101, 103, 104
Hypothesis validator 7
Implication 7, 101
INDATA 92
Inference 104
Inference engine 2, 101
Inference generation 16, 104, 105, 106
Inference level 4, 104
INITIAL 83
Initial knowledge source 19, 98
Initialization 15, 102
INPLINES 65
INPLIST 68
INPNAME 70
Input 15, 102
Input data 3, 4, 102, 104, 107
Input data item 29
INPUTDATA 20, 92
INPUTDATANAMES 20, 29, 92
INPUTFORMAT 83
INPYN 72
INTERLISP 1
INVOKE 92
Inverse link 14
Invoke 8, 19, 92, 98
Invoked 105
KAS 8, 108
KEEPHISTORYFLAG 92
Keepinprogeny? 12, 13, 14
Kernel 15, 16, 83, 102, 104
Knowledge based program 1
Knowledge source 1, 3, 4, 7, 8, 14, 19, 24, 26, 34, 86, 93, 95, 98, 101, 102, 103, 104, 105, 107
KNOWLEDGESOURCES 83
KS 3, 4, 7, 8, 9, 10, 12, 15, 16, 17, 19, 26, 29, 34, 90, 92, 101, 102, 104, 105, 106, 107
KSLIST 8, 9, 93
Left hand side 7, 102, 105
Level 12, 13
LEVELLIST 6, 93
LHS 7, 8, 9, 19, 20, 24, 29, 86, 104, 105, 108
LHS Evaluator 7, 10, 17, 101, 105
Linear sequence 4
Link 4, 6, 7, 9, 19, 29, 86, 89, 93, 104, 108
Link node 12, 13
LINKS&OPPOSITES 6, 93
LISP 3, 103, 107
LLIST 5, 94
LOADCOMFLAG 93
LOADLIST 93
LOADTYPEFLAG 93
LOOP 10, 11
MACRO EVENTSPEC 83
MACRO EXPECTSPEC 83
MAKESLOT 14, 22
MAKEUNIT 14, 21
MAXKEYLISTLENGTH: 93
Mega-chunk 7
Mega-rule 7
MINSELECTDISPLAY 93
Model-derived link 4
MODIFY 12, 13
MODIFY1 57
MSGSLT 12, 13
MSGUNIT 12, 13
MULTIPLE 9, 104, 105
Multiple hit 7, 105
MYCIN 24, 108

NAMEOF 76
NAMEOF1 76
NEWNODENAME 94
NILSAVE 94
Node 4, 86, 94, 103, 105
NODELEVELS 5, 86, 94
NODELIST 25, 94
NOTRACEFUNCTIONS 95
NULL 12, 13
NUM 5, 94

Object centered representation 1, 4, 107
Once only 7, 105
ONCEONLY 9, 95, 104, 105
OPENNETWORK 20, 21
OPENSRCATCHFILE 76
Opposite link 9

Parent 12, 13, 14
Pattern 7
Post processing 15, 29, 102, 106, 107
Post-processing function 106
POSTPROCESSING 83
Pre-created element 5, 94, 106
Precondition 7, 8, 12, 16, 105, 106
PRECREATEDNODES? 82
PRHYPO 59
PRHYPOLEVEL 59
PRHYPOLEVEL2 59
PRIMCNT 76
PRINCOMMAND 77
PRINFORMS 78
PRINNEXT 78
PRINTALL 59
PRINTBCDATA 30, 59
PRINTBCS 59
PRINTCONTROLINFO 59
PRINTCONTROLRULES 60
PRINTERROR 51
PRINTHYPO 60
PRINTKS 60
PRINTLIST 60

PRINTPROPLIST 60
PRINTRULE 60
PRINTRULEACTION 60
PRINTRULES 61
PRNODE 61
PRNODE2 61
Probability 8, 108
Probability value 86
Production rule 1, 2, 3, 4, 7, 16, 101, 102, 105, 106, 107
Progeny? 12, 13, 14
PROPOSE 10, 12, 16, 27, 35, 90
Proposition 14, 19, 34
PRULES 61
PUFF 24, 101
PUTVALUE 14, 23

Reduction link 4, 104
Relation 12, 13, 14
RESETCNT 80
RETBC 80
RHS 7, 8, 16, 20, 24, 106, 108
Right hand side 7, 106
Role 12, 13, 14
Rule 3, 7, 19, 29, 34, 86, 90, 102, 104, 106, 108
Rule evaluation 9
Rule number 37, 95
RULESUSEDONCE 95
RUNBACKCHAIN 37

SAVECHAIN 61
SAVECONTROL 61
SAVEDATA 61
SAVEDESIGN 62
SAVEFILE 95
SAVETHYPOHESIS 62
SAVEINDATA 62
SAVERULES 62
SAVEUSERFNS 62
SCRIBEINDEXFLAG 95
SEPRCHARS 95
SINGLE 9, 104, 106
Single hit 7, 106
SITENAME 95
Situation 101
Slot 12, 13, 14
Slot-value 14
SLOTMSG 14, 22
Slotvalue 12
Step 16, 18, 19, 34, 35, 37, 80, 90, 95, 98, 104, 106, 107
Step execution function 17, 95, 106
Step list 17, 18, 34, 95, 106
Step type 16, 17, 18, 19, 89, 95, 103, 104, 106
STEPSAVE 95
STEPYPELIST 95
SUPERSEDE 12, 13
Support 12, 13, 14
Support from above 4, 104
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE Reference Manual</td>
<td></td>
</tr>
<tr>
<td>Support from below</td>
<td>4, 104</td>
</tr>
<tr>
<td>Syntax check</td>
<td>51</td>
</tr>
<tr>
<td>SYSCHECK</td>
<td>51</td>
</tr>
<tr>
<td>SYSINIT</td>
<td>41</td>
</tr>
<tr>
<td>SYSTEMEXITCOM</td>
<td>96</td>
</tr>
<tr>
<td>SYSTEMNAME</td>
<td>96</td>
</tr>
<tr>
<td>SYSTEMNULFILE</td>
<td>96</td>
</tr>
<tr>
<td>Task</td>
<td>41, 42, 107</td>
</tr>
<tr>
<td>Task domain</td>
<td>107</td>
</tr>
<tr>
<td>TENEX</td>
<td>1</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>83</td>
</tr>
<tr>
<td>Termination condition</td>
<td>107</td>
</tr>
<tr>
<td>TERMINATIONCONDITION</td>
<td>20, 96</td>
</tr>
<tr>
<td>TEXTED</td>
<td>79</td>
</tr>
<tr>
<td>TIMEFLAG</td>
<td>96</td>
</tr>
<tr>
<td>Token</td>
<td>12, 13, 14</td>
</tr>
<tr>
<td>TOPS-20</td>
<td>1</td>
</tr>
<tr>
<td>Trace mode</td>
<td>87, 96</td>
</tr>
<tr>
<td>TRACEINFO</td>
<td>96</td>
</tr>
<tr>
<td>UA.AVAIL</td>
<td>96</td>
</tr>
<tr>
<td>UA.BUMPFLG</td>
<td>96</td>
</tr>
<tr>
<td>UA.BUMPN</td>
<td>96</td>
</tr>
<tr>
<td>UA.ERRMSG</td>
<td>96</td>
</tr>
<tr>
<td>UA.ERRNO</td>
<td>96</td>
</tr>
<tr>
<td>UA.ERRTOKEN</td>
<td>96</td>
</tr>
<tr>
<td>UA.FILENAME</td>
<td>96</td>
</tr>
<tr>
<td>UA.FILES</td>
<td>96</td>
</tr>
<tr>
<td>UA.FIXED-FIELDS</td>
<td>96</td>
</tr>
<tr>
<td>UA.GCTRPCOUNT</td>
<td>97</td>
</tr>
<tr>
<td>UA.HASHFILE</td>
<td>97</td>
</tr>
<tr>
<td>UA.HEADERPOS</td>
<td>97</td>
</tr>
<tr>
<td>UA.MINGOODBLOCKSIZE</td>
<td>97</td>
</tr>
<tr>
<td>UA.NONUSERS</td>
<td>97</td>
</tr>
<tr>
<td>UA.PFLG</td>
<td>97</td>
</tr>
<tr>
<td>UA.RELFILE</td>
<td>97</td>
</tr>
<tr>
<td>UA.ROVER</td>
<td>97</td>
</tr>
<tr>
<td>UA.TIMESTAMP</td>
<td>97</td>
</tr>
<tr>
<td>UA.TRACEFLG</td>
<td>97</td>
</tr>
<tr>
<td>UA.TRAILPOS</td>
<td>97</td>
</tr>
<tr>
<td>UA.UNIT</td>
<td>97</td>
</tr>
<tr>
<td>UA.UNITFILE</td>
<td>97</td>
</tr>
<tr>
<td>UA.UNITLIST</td>
<td>97</td>
</tr>
<tr>
<td>UA.UNITPAGEPROPS</td>
<td>97</td>
</tr>
<tr>
<td>UA.UNITPROPS</td>
<td>98</td>
</tr>
<tr>
<td>UA.UNITRELPROPS</td>
<td>98</td>
</tr>
<tr>
<td>UA.UNITSIN</td>
<td>98</td>
</tr>
<tr>
<td>UA.UNITSOUT</td>
<td>98</td>
</tr>
<tr>
<td>UA.UPFLG</td>
<td>98</td>
</tr>
<tr>
<td>UA.USERS</td>
<td>98</td>
</tr>
<tr>
<td>UCONTROL</td>
<td>41</td>
</tr>
<tr>
<td>ULOAD</td>
<td>79</td>
</tr>
<tr>
<td>UNGUIDED</td>
<td>43</td>
</tr>
<tr>
<td>Unit</td>
<td>7, 12, 13, 14, 19, 34, 101</td>
</tr>
<tr>
<td>Unit Editor</td>
<td>20</td>
</tr>
<tr>
<td>UNITMSG</td>
<td>14, 22</td>
</tr>
<tr>
<td>UNITS</td>
<td>3, 14, 103, 107</td>
</tr>
<tr>
<td>Units Knowledge Base</td>
<td>4, 7, 13, 14, 16, 19, 34, 107</td>
</tr>
</tbody>
</table>

October 123 1981
Copyright © 1985 by KSL and
Comtex Scientific Corporation