A Rule Language for the GLISP Programming System

Christopher A. Rath

AI TR86-31 August 1986

Support for this research was provided by the Army Research Office, under grant number #ARO DAAG29-84-K-0060
A Rule Language for the GLISP Programming System

Abstract:

GLISP is a public domain programming language written by Dr. Gordon Novak at the University of Texas at Austin. It is an object oriented language based on LISP, a list processing language used in artificial intelligence programming. GLISP is often used in the university environment for the construction of expert systems programs, yet it has no formal rule language.

The thesis is the design and implementation of a general rule language incorporated into GLISP. The project includes an investigation into currently implemented rule languages, an implementation of the rule language compiler in InterLisp, and sample applications. The resulting system is expected to be efficient, general purpose, and easy to learn for those programmers already familiar with GLISP.
To Christine, Mom, and Dad
# Table of Contents

1. Introduction .............................................. 1  
3. GLISP RuleSets .......................................... 16  
4. Reasoning With Uncertainty ............................ 25  
5. Contexts .................................................. 38  
6. Conclusions .............................................. 44  
Appendix A The GLISP Rule Language User's Manual ...... 46  
Appendix B HPSR.LSP ...................................... 79  
Appendix C HPSC.LSP ...................................... 93  
Appendix D HPSFUZZY.LSP .................................. 101  
Appendix E HPSRUN.LSP ................................... 113  
Appendix F Examples ....................................... 120  
Bibliography ............................................... 128  
Vita ......................................................... 131
Chapter 1

Introduction

There are many expert systems tools currently available, ranging from completely preprogrammed systems to highly flexible rule languages embedded within programming languages. Their domains of applicability vary in proportion to the degree of freedom given to the programmer in specifying rules and data structures.

In a marketing report, [Bauman 1984], on the use of expert systems tools, it was found that GLISP [Novak 1982, 1983a, 1983b] was, at the time the survey was taken, the second most widely used language for writing expert systems applications, (OPS5 being first). GLISP, however, does not have a production rule language, forcing programmers to write their own rule system for each project if they wish to use rules.

The purpose of this project is the design and implementation of a general purpose rule language embedded in GLISP. The language preserves the advantages of the GLISP programming language:

1. Readability and conciseness of program code
2. Rich data typing facilities
3. Optimization of code for efficient execution.
This makes the language easy to learn for those already familiar with GLISP. In addition, the language is to provide a basis upon which complete expert systems can be built. To this end, the project can be thought of as consisting of a rule language, plus various packages supporting different problem solving paradigms.

In Chapter two, various expert systems tools which have influenced the design of this project will be discussed. In chapters three through five the design of the rule language is presented, along with examples of its use. Conclusions about the merits and faults along with suggestions for future extensions of this work comprise the final chapter.

This report also serves as a guide to the use of the GLISP rule language. Therefore, a user's manual has been included as the first appendix. The various files which comprise the program code are included for documentation purposes.
Chapter 2

Survey of Expert Systems Tools

There are too many expert systems tools available to include a comprehensive survey in this work. Instead, some of the more well known systems whose design influenced the GLISP rule language will be described, with some note to their relevance to this project. Before discussing the various specific systems, a general introduction to production systems will be given.

2.1 Production Systems

A production system consists of a set of rules, a database, and an interpreter for the rules, [Davis 1977]. Rules are an encoding of an expert's knowledge in the form:

If x is true  
Then do y.  

If x is true  
Then conclude y.

The If part of a rule is alternately called the antecedent, the left hand side (LHS), the condition, or the premise of the rule. The LHS of a rule is typically a conjunction (AND) of several conditions, all of which must be true for the rule to fire. The
Then part is the consequent, right hand side (RHS), or action of the rule.

A rule is said to be eligible to fire if its LHS is true, with respect to the currently available data. A rule is fired by executing the actions in the RHS. In general the selection of an eligible rule and firing its actions constitutes a complete cycle.

The database in a production system is the data for a particular problem. It may contain facts observed by the user and hypotheses generated by the rules of the production system. In a pure production system, the database is the only medium through which rules may communicate with each other.

The rule interpreter is a procedure for executing rules. It specifies a method for choosing which rules to evaluate, determining when they are eligible to fire, and executing the actions of a rule when it does fire. There are two broad classes of interpreters: forward chaining and backward chaining.

A forward chaining system uses known data to derive new hypotheses by scanning the LHS of its rules and choosing one to fire. The actions of the chosen rule may add new hypotheses to the database, causing more rules to be eligible. This process continues until no LHS is satisfied. Forward chaining may be likened to breadth first search in a tree structure.
Backward chaining systems are goal directed. That is, given the goal of determining some piece of information, the RHS of each rule is examined to see if it satisfies that goal. If so, the goal of satisfying the LHS of the rule is added to the system and the process continues. Backward chaining is more like depth first search, moving backwards through the tree until a node is reached whose conditions are all fulfilled or have been shown to be impossible to fulfill.

None of the systems which are described below can be considered to be "pure" production systems. Generally purity is sacrificed for efficiency or flexibility of control.

2.2 EMYCIN

EMYCIN, [van Melle 1981] is a domain independent expert system tool which arose out of the MYCIN [Shortliffe 1976] medical diagnosis consultant. It is a member of the class of restricted, preprogrammed systems; one data type is supported, basically one method of rule evaluation and one control mechanism are allowed. The domain in which EMYCIN works well is thus fairly restricted, consisting mostly of identification and diagnosis problems.

Rules in EMYCIN follow the traditional production form, consisting of a premise (the If part), and an action (the Then part). The premise is an expression consisting of predefined
predicates for comparing data in the database to usually constant values; the predicates are combined using a special set of logical connectives which support reasoning with uncertainty. In general the premise evaluates to a number in the range \([-1,1]\) which represents the extent to which the system believes it is true. A rule fires if this value exceeds the system threshold, 0.2.

The actions of a rule also consist of predefined functions which assert values for elements of the database. There is some capability for the user to define LISP functions for use in the action of a rule, but this is limited.

Data specific to a problem is stored in a database structure called the context tree. A node in the tree can be considered a self-contained database which may have other context trees defined within it. The data itself is stored in fields called parameters. In addition to values, parameters contain descriptive information which is used to help users supply information to the expert system or to present parameter values in an English-like form. Other restriction information verifies that parameter values are of the proper type specified by the programmer.

EMYCIN uses a backchaining interpreter for evaluating rules. However, the set of rules is not scanned at each cycle. When the system is created, the rules are indexed according to
the parameter values they are able to conclude. The execution cycle consists of examining the current goal (parameter whose value is to be determined) and taking the first rule in the index for that goal. The premise for the rule is then examined. If it contains any parameters whose value is yet undetermined, those parameters are made into goals and the cycle starts over again. Once all parameters for a premise are known it is evaluated. If its certainty exceeds 0.2, the actions are executed, else it is skipped. In either event, the rule is removed from consideration for later cycles.

The EMYCIN compiler gives additional efficiency by compiling rules into decision trees. Often several rules will have common conditions within their premises; by creating a decision tree out of them, each premise need only be evaluated once. When this is not possible, common subexpressions within the premises may be evaluated and bound to a variable which replaces the expression in the rules. Together these serve to make EMYCIN a fairly efficient system.

Because the allowable actions and premises are limited, EMYCIN is able to employ a sophisticated user interface. Explanation facilities are given, translations both from rules to English, and English to rules are available, and many development tools aid in the design of expert systems with EMYCIN.
The GLISP rule language takes the opposite approach; completeness is sacrificed for flexibility. The use of uncertainty in EMYCIN strongly influenced the design of both the fuzzy evaluation system and contexts in the GLISP rule language. Although the GLISP rule language is primarily a forward chaining system, there is a limited capability for backchaining included for parameter evaluation.

2.3 OPS5

OPS5 [Forgy 1981] is the latest in the OPS family of production system languages. It is a forward chaining system, widely used as an expert system tool. An OPS5 program consists of a set of production rules, often called the long term memory element, a database called the working memory, and one of several conflict resolution algorithms for choosing the rule to fire during a given cycle.

An OPS5 rule is of the form:

LHS \rightarrow \text{RHS}

where the LHS is a list of patterns to be matched against the database and the RHS is a list of actions to be performed. A pattern is a list consisting of constants and variables. The LHS is considered true if each of its patterns matches some element of the database.

The RHS actions include functions to add and delete elements in the database, along with trace features and null
actions. The user may define new RHS actions as LISP functions and declare them to the system, thus increasing the flexibility of the language.

Working memory is a list of arbitrary list structures or atoms against which the LHS of a rule is matched. For example, the working memory may contain:

( (ANIMAL ↑NAME REX ↑SPECIES DOG)
    (PARENT ↑NAME REX ↑OFFSPRING KING) )

The following rule would add the fact that King is also a dog:

( PPARENT-RULE
    (ANIMAL ↑NAME <NAME1> ↑SPECIES <SPECIES1>)
    (PARENT ↑NAME <NAME1> ↑OFFSPRING <NAME2>)
    → (MAKE ANIMAL ↑NAME <NAME2> ↑SPECIES <SPECIES1>) )

The interpreter for OPS5 matches all the LHSs of rules against working memory, placing those which match in a conflict set. The conflict resolution algorithm is then called to choose one of the rules in the conflict set to fire. Different implementations use different algorithms. Some possibilities include: Recency - fire the most or least recently fired rule, Specificity - fire the rule with the longest LHS, etc. This cycle is repeated until no rule's LHS matches the database.

Efficient implementations of OPS do not actually consider every rule in each cycle. This would make execution of large programs intolerably slow. The OPS compiler [Forgy 1979]
transforms rules into a network of node programs. The interpreter then examines only those nodes where changes have been made. Forgy was able to demonstrate improvements of several orders of magnitude over non-compiled systems.

2.4 LOOPS

LOOPS [Bobrow 1981] is an object oriented language built on top of Interlisp. It contains an embedded rule language for programming in the production system style. Rules in the rule language are unrestricted in content, thus giving the system a wide range of applicability. However, user interaction features are very limited. There are no explanation facilities built in and reasoning with uncertainty is left up to the programmer as well.

Rules in the LOOPS language are defined in small packages called rule sets. Rule sets are compiled into Interlisp code and can be invoked by user programs, by messages to objects, or from other rule sets. A rule has the form:

If (expr1 expr2 ...)

Then (expr1 expr2 ...).

The expressions in the if and then parts are allowed to be any arbitrary LOOPS expression. The If part is evaluated from left to right until an expression evaluates to NIL, or all have been evaluated. In the first case the rule is skipped, in the second
it is eligible to fire. The Then expressions in a rule selected
to fire are also evaluated from left to right; the value of the
last expression is returned as the value of the rule.

Rules may also contain meta-descriptions which are used to
control their execution or provide descriptive information to the
tracing functions. A common meta-description is the !
description. When present, the rule is marked after it fires and
is then not eligible to fire again.

Four forward chaining control methods are available to each
rule set. They are:

1. DO1 - Fire the first rule whose If condition is true
   and exit.
2. DOALL - Fire all rules whose If condition is true
   and exit.
3. WHILE1 <condition> - As long as <condition>
   evaluates to true, fire the first rule whose If
   condition is true.
4. WHILEALL <condition> - As long as <condition>
   evaluates to true, fire all rules whose If condition is
   true.

Because rule sets are compiled into LISP functions, their
component actions should be relatively efficient. However, we
have not determined the efficiency of the rule set construct.
The GLISP rule language has many similarities to the LOOPS rule language. Both use the concept of a rule set, both allow general data structures and unrestricted rule expressions. Although the choice of control methods differs, the idea of offering the programmer a choice is present in both systems. The GLISP rule language, however, provides packages which support predefined problem solving components, thus relieving the programmer from writing commonly used concepts such as reasoning with uncertainty.

2.4 NEOMYCIN

NEOMYCIN [Clancy 1981] was a project to give MYCIN the ability to control its reasoning processes. The traditional rule, which could only refer to the internals of a particular problem, was retained and called a domain rule. Meta-rules, tasks, and a task interpreter were added for control over the execution of domain rules.

A task is a set of meta-rules and control information for solving a problem. The task interpreter evaluates the meta-rules, which in turn call upon domain rules for the solution of problems. Meta-rules may also invoke other tasks for the solution of subproblems. The execution of meta-rules is controlled by the programmer, execution of domain rules is by backchaining.
One advantage of this system is that tasks can be domain-independent. Thus complete problem solving paradigms can be stated once, generically, for an entire class of problems. This makes it possible to create a library of systems, eliminating the need to reprogram them for each application.

The advantages provided by NEOMYCIN are available in the GLISP rule language since rules can call other rules. The form of meta-rules in GLISP does not differ from domain rules however. This makes the GLISP rule language more like the CRYsalIS project described below.

2.5 CRYsalIS

CRYsalIS, [Terry 1983], is an expert system for deducing crystal structures from molecular and chemical information. Although it is not a general expert systems tool, it implements the concept of hierarchical production systems which was influential in the design of the GLISP rule language.

A hierarchical production system consists of several layers, each of which is a production system which examines the current situation and chooses actions from the next layer below it. Some of the advantages stated by Terry of hierarchical systems are clarity of control, reduction in control conditions for domain rules, and efficiency. These results are due to the
ability of the system to focus its attention on relevant parts of the problem chosen by the meta-rules.

2.6 AGE

AGE, [Aiello 1981], is a collection of problem solving tools connected with a user interface to aid in designing complete expert systems. The programmer selects from various predefined sets of functions and variables called components which support a particular problem solving technique. Some of the available components are blackboards, production rules, credibility factors, and search routines. The programmer combines the predefined functions with his own to build a complete problem solving system. Experimentation with different techniques is made easy by the packages, reducing the amount of recoding needed.

A commonly used component in AGE is the blackboard. A blackboard is a common data structure through which various knowledge sources communicate intermediate hypotheses, [Hayes-Roth 1983]. In AGE a blackboard is a common data structure with which various sets of rules called knowledge sources are associated. The programmer defines an algorithm for selecting which knowledge sources to invoke during execution; changes in the blackboard data may also automatically call knowledge sources to update their hypotheses.
Although the programmer may define evaluation systems for rules, two are provided with AGE. The first, called $\text{AND}$, operates as the logical AND, that is, all of the conditions must be true in order for the rule to fire. $\text{ANDMIN}$ works with uncertain data, returning the minimum certainty of its component conditions. Functions are provided for asserting values to the blackboard with certainty, and adjusting the certainty of current values on the blackboard.

The AGE knowledge source is very similar to the GLISP RuleSet. Overall, a system like AGE is a goal towards which the GLISP rule language is but a single step. AGE provides several problem solving techniques, allowing the programmer to experiment, choosing the best one for the application.
Chapter 3

GLISP RuleSets

Rules in the GLISP rule language are structured into small groups with related purposes called RuleSets. A RuleSet may be considered to be a miniature production system, complete with a set of rules, a database, and a control scheme. A complete expert system will consist of several RuleSets combined with GLISP functions for user interaction and procedural problem solving. (See Appendix A, Section 2 for syntax of RuleSets.)

3.1 RuleSet Definitions

A RuleSet consists of several parts which describe the database, rules, and control method to be employed for a particular problem. The RuleSet compiler uses the information from the declaration to compile a RuleSet into a GLISP function. The GLISP compiler translates this code to Interlisp, which can be further compiled into machine code for efficiency. Once compiled, RuleSets can be invoked from user programs or from other RuleSets. An example of a RuleSet is given below:
(MINERAL-NAME-RULES  (M:MINERAL)

RESULT  (M MINERAL)

TERMINATE ( ((PRIN1 "No rules fired.")(TERPRI))

((SCVT ($KNOWN NAME)) (REPORT M 'NAME)))

CONTROL  CERTAINTY

RULES (

(RULE-1  (* if the color is black and it is very hard, it is diamond)
  IF  ($AND ($EQ COLOR 'BLACK) ($GEQ HARDNESS 9.5))
  THEN  (($ASSERT M 'NAME 'DIAMOND 0.8)))

(RULE-2  (* if the color is black and it is not very hard, it is coal)
  IF  ($AND ($EQ COLOR 'BLACK) ($LESSP HARDNESS 9.5))
  THEN  (($ASSERT M 'NAME 'COAL 0.6)))

(RULE-3  (* if the color is red it is iron)
  IF  ($EQ COLOR 'RED)
  THEN  (($ASSERT M 'NAME 'IRON 0.8))))

This particular RuleSet would be one of many for identifying minerals. It would be invoked by the user program or another RuleSet when some information about the color of the mineral was known. Using RuleSets, a problem can be modularized to ease the testing of groups of rules. This will speed the development process.
3.2 Rules

Rules in a GLISP RuleSet maintain the traditional production format of an If condition followed by the Then actions. Additionally, a rule has several descriptive fields used for control over its execution and for auditing information. The first field is the name field. (See Appendix A, Section 1 for syntax of Rules.) Rule names are printed when they are fired, if the trace variable HPSTRACERULES is set to T. Following the name field are two optional elements, the one shot rule declaration and the priority declaration. One shot is indicated by the symbol !, which indicates that the rule is only to be fired once and then not considered again. This feature is useful for rules which perform some physical action.

For example, in a chemical plant, the rule which starts the alarm when a chemical leak is found need only be fired once, while other monitoring rules continue to be checked.

(ALARM-RULE ! 8 (* Sound the alarm when a spill is detected)
  IF ($TRUE SPILL-DETECTION)
  THEN ((SOUND-WARNING-ALARM)))

(MONITOR-RULE (* Check to see if gas is coming in the vents)
  IF ($TRUE EVENT-MONITOR T)
  THEN (($ASSERT SPILL 'PLAN 'TRACE-AIR-SYSTEM 1.0))

The priority of a rule is a number, indicating the order of execution when several rules are available. In general the higher the priority number, the better chance the rule has of firing. Priorities default to 0. Thus in the example above, if both SPILL-DETECTION is true and VENT-MONITOR is true, ALARM-RULE will fire first. Note: Priorities are ignored in RuleSets which use either the FIRST or ALL control methods.

The If expression is allowed to be any arbitrary GLISP expression. This allows the user to choose the form of rule evaluation which best fits the domain of the application. One specific evaluation system, the fuzzy evaluation system, is provided as a package. (See Chapter 4.)

If the If expression is true, the Then actions are evaluated, from left to right. Again any GLISP expression may be part of the Then expression. This provides not only added flexibility in actions, (e.g. SOUND-WARNING-ALARM is a user function which interacts with the environment), but also the ability to create arbitrary hierarchies of RuleSets by invoking RuleSets as actions. In the chemical spill example, one RuleSet may monitor environmental conditions and call other RuleSets to solve problems as they occur.

3.3 Database

RuleSets accept arbitrary GLISP objects as arguments and use them as their database. Through local and global variable
declarations, private and public databases may be maintained as well. A local database, for example, may be a plan for cleaning up a chemical spill. The argument passed to the RuleSet contains information relative to a particular spill, while a global database may include a table of counter-agents for various chemicals. The three together make up the RuleSet database. The partial declaration below demonstrates this example.

(SPIll-CLEANUP-RULESET (S:SPIll)
  LOCAL (P:PLAN)
  GLOBAL (SOLVENT-TABLE)

RULES(
  (ACID-SPILL1 (* Add alkaline for an acid spill)
    IF ($EQ S:TYPE 'ACID)
    THEN ((ADD-TO-PLAN (ASSOC 'ACID SOLVENT-TABLE)) ... ))

3.4 Control

RuleSets execute in a forward chaining manner. Several parts of a RuleSet declaration contribute to the complete control scheme. The most important of these is the CONTROL field.

Four specific control methods for the selection of rules are provided:

1. FIRST - Fire the first rule whose If part is true
2. ALL - Fire all rules whose If part is true
3. CERTAINTY - Fire the rule whose If part has the highest certainty.
4. PRIORITY - Fire the rule with the highest priority whose If part is true.

Under the CERTAINTY scheme, priority is used to break any ties. With PRIORITY, the certainty of an If condition is used to break ties. Rules which do not use any form of certainty factor in evaluating their If part are assumed to have a certainty of 1.0 if true, and 0.0 if NIL.

Iteration is implicit in a RuleSet, controlled by the TERMINATE clause, which acts like the UNTIL condition in a REPEAT-UNTIL loop. A TERMINATE clause has two parts: exception handling actions, and a termination condition. An example is given below:

```
TERMINATE ( ((PRINT "No Rule fired.") (TERPRI))
            (($CVT ($KNOWN NAME)) (REPORT M 'NAME)))
```

The first element in the TERMINATE clause is a list of actions to be taken when no rule is able to fire. This enables the programmer to trap this exceptional condition, perhaps invoking an information gathering routine, or simply informing the user that the RuleSet terminated without reaching the termination condition, as above.

The termination condition is a list whose first element is a GLISP expression and whose remaining elements are actions to be
taken when the expression is non-NIL. Iteration of a RuleSet halts when the termination condition is met, and the actions are then performed, or when no rule fires, and the exception actions are performed.

Iteration can be inhibited by specifying ONE as the termination condition, followed by any number of actions. Since repeated evaluation of rules can be inefficient, this condition was added for cases where iteration is unnecessary. For example, a RuleSet may be designed to find a path through a maze. If we are only interested in a single answer, using the ONE condition will inhibit the generation of alternative paths.

Finally, as RuleSets compile into Lisp functions, they should be allowed to return a value. This value is specified as the RESULT clause, which consists of an expression and an optional type for the expression. When a RuleSet terminates, the expression is evaluated and returned as its value. This value can then be used within GLISP expressions, or as data to another RuleSet.

Although RuleSets are basically forward chained, there is some capability for backchaining. The BACKCHAIN declaration of a RuleSet tells the compiler which parameters have values asserted for them by rules in the RuleSet. A backchain declaration is an association list whose elements are lists consisting of a context type and a list of parameter names. This declaration is compiled
into an index. To use backchain evaluation, the programmer must specifically call the BACKCHAIN function, which takes a context and a parameter name, (See Chapter 5). All RuleSets in the index for the given parameter are then invoked in an attempt to get a value for the parameter. Example:

\[
\begin{align*}
\text{(MINERAL-NAME} & \quad \text{(M:MINERAL)} \\
\text{BACKCHAIN} & \quad \text{((MINERAL(NAME))})
\end{align*}
\]

Another RuleSet which needs the name of a mineral to determine its worth would contain rules such as:

\[
\begin{align*}
\text{(WORTH-1} & \quad \text{(* Gold is worth about $340 an ounce)} \\
\text{IF} & \quad \text{($EQ (BACKCHAIN M 'NAME) 'GOLD)} \\
\text{THEN} & \quad \text{($ASSERT M 'WORTH (TIMES WEIGHT 340) 0.9))}
\end{align*}
\]

If the parameter already has a value, no backchaining is done. In a sense, control can be considered to be backchaining over the set of RuleSets, and forward chaining within RuleSets.

3.6 Advantages

There are several advantages to the RuleSet design given above. Through compilation into GLISP and eventually Interlisp, a certain amount of execution speed can be gained. Further, if the number of rules in a RuleSet is kept small, efficiency and readability can be maintained.

In order to take advantage of small RuleSets it is necessary to be able to determine which of them should be
invoked. Because the Then actions are not restricted, rules are able to invoke other RuleSets, allowing development of a hierarchical system. The advantages mentioned for the CRYsalis system are thus obtained.

RuleSets encourage a modular design of expert systems. Problems will be divided into small parts, each of which is solvable by a RuleSet. The individual RuleSets can be tested independent of each other, thus speeding the development process for expert systems. As functions, RuleSets can be directly invoked from GLISP programs. This gives the programmer the ability to mix procedural and production systems, delegating to each what it solves best. The author feels that this will make future systems stronger and more efficient.

3.7 Disadvantages

The generality of the system reduces the chance for good optimization of the rules. Thus the efficiency of the system will be impaired. Forward chaining can lead to inefficiency as rules are repeatedly evaluated, even if previously found to be false. Generality also makes it more difficult to include user interaction functions such as explanation and translation features. These must be left to the programmer.
Chapter 4
Reasoning With Uncertainty

Because rule evaluation in the CLISP rule language is
ultimately performed by Lisp functions, the programmer may
implement the form of reasoning with uncertainty which best fits
the problem domain. Some methods are described below. The rule
language contains a package of functions which support the fuzzy
logic paradigm for the evaluation of LHS clauses and three
methods of asserting values with uncertainty. These are
discussed at the end of the chapter.

In solving real world problems, humans must often draw
inferences using inexact or uncertain data. Part of the problem
lies in the difficulty of stating things precisely in natural
language where most words are imprecise by nature, and observers
are frequently unfamiliar with the exact terminology even if it
is available. Another source of imprecision lies in the
certainty of observations made in solving a problem. A wildlife
expert may only catch a brief glimpse of the underbelly of some
bird. It may be in shadows or partially obstructed by branches.
In general, observers will not be absolutely certain of what was
seen.

Inexactitude of the first kind can be recognized in
statements such as: "It was sort of red.", "The temperature was
pretty hot.", "I live close to the university.", "She lives about two miles from the lake." It would be exceedingly difficult to communicate if humans could not work with statements having varying degrees of precision. The second type of imprecision arises from imperfect observational circumstances such as the wildlife example above. Example statements include: "I am pretty sure it was red.", "It was probably the moon."

If an expert system is to perform at a level comparable to that of a human expert, it must be able to draw conclusions and make decisions based on inexact data. That expert systems designers have realized this is evidenced by the number of systems which incorporate some form of inexact reasoning. Some such systems will be discussed below.
4.1 Probabilistic Methods

Bayes' Theorem is one of the most widely used of probabilistic inference methods. Its use arises when determining probabilities in a two stage experiment. That is, given a probability that an event $S$ occurs, an experiment is done and it is found that some other event $X$ occurs with some probability. If the probability of $X$ occurring independent of $S$ is known, then the probability for $S$ can be strengthened or weakened based on the experiment. The formula for determining the probability that an object belongs to some class $C$ based on evidence $e$ is given in Figure 1. For example, assume that the probability of contracting a certain disease is one in 100,000 and that if a patient has this disease she will have red spots with a
probability of 50%. Also assume that the probability of a person having red spots in general is one in 10,000. By Bayes' theorem, the probability that a patient with red spots has this disease is given by \( P(\text{disease} \mid \text{red spots}) = (0.5)(0.00001)/(0.0001) \), which is 5%.

There are problems with implementing Bayesian analysis which make it difficult for expert systems design [Stefik 83]. First, it requires a large number of statistical measures to be taken. These statistics may be difficult or impossible to obtain, especially when working with data from imprecise observations. It would be necessary, for example to calculate probabilities that an observer who actually saw a red underbelly would report seeing black. A second problem with this method is that the probabilities derived often differ with those derived by experts from their personal experience [Novak 85]. The statistics given by experts are in general accurate only to within 10% of the actual values. In Bayes' equation, such tolerances introduce sizeable estimation errors.

Similar to Bayesian analysis are the methods of hypothesis testing and decision theory [Weiss 1984]. These methods assign a value of true or false to a statement, based upon the costs of making the wrong decision. If the expected cost of mistakenly concluding \( X \) exceeds a given threshold, then not \( X \) is asserted. In the disease example, the cost of not diagnosing a disease that
is present is higher than diagnosing one that is not present. In hypothesis testing a threshold probability is used to determine whether or not a value should be asserted. Figure 2 gives the decision rule for hypothesis testing.

\[
\text{Hypothesis Testing}
\]

\[
\text{If } \quad \frac{P(e|\mathcal{C})}{P(e|\neg \mathcal{C})} > t(a)
\]

then decide \( \mathcal{C} \)

else decide \( \neg \mathcal{C}. \)

\( t(a) \) is the threshold function for positive and negative errors.

Figure 2

incorporates a subjective cost function in the threshold. Figure 3 gives the rules for decision theory.

As with Bayes' Theorem, these methods suffer the problems of gathering accurate statistics. In addition, the decision theory method relies on subjective cost functions which will reduce its accuracy.

A further problem with statistical methods is that an assumption of observational independence is necessary in order to make the evaluations of antecedent probabilities tractable[Stefik 83]. That is, it is assumed that the observed values do not influence each other. Stefik argues that such simplifications undermine the rigorous statistical basis upon which Bayes'
### Decision Theory

If

\[
P(e \mid C) / P(e \mid \neg C) > P(\neg C) \cdot \text{Cost}(FP) / [P(C) \cdot \text{Cost}(FN)]
\]

then decide \( C \).

if

\[
P(e \mid -C) / p(e \mid C) > P(C) \cdot \text{Cost}(FP)/[P(-C)\cdot \text{Cost}(FN)]
\]

then decide \( -C \).

Cost(FP) is the subjective cost of making a false positive assertion.
Cost(FN) is the subjective cost of making a false negative assertion.

**Figure 3**

Theorem depends. It does not make sense to apply exact methods to inexact data.

#### 4.2 Fuzzy Logic

Fuzzy set theory was postulated by L.A. Zadeh in a paper entitled "Fuzzy Sets", [Zadeh 65]. In traditional set theory a set is defined by a two-valued membership predicate over the universe, 1 indicating membership and 0 indicating exclusion. Fuzzy set theory extends the definition of a set by allowing its membership function to range over the entire interval \([0,1]\). The value indicates the degree to which an object in the universe
belongs to a given set. Figure 4 demonstrates a fuzzy set of hot things.

\[ S = \{(x,y) \mid y = \text{IsHot}(x)\} \]

Sample elements of \( S \):

- (Sun, 1.0)
- (Steam, 0.8)
- (100°, 0.5)
- (Ice, 0.0)

Figure 4

Given this set definition, there are several consistent ways to define set operations such as intersection, union, and complementation. Zadeh's original definitions are intuitive and computationally simple. In these definitions, \( U_S(x) \) is the membership function which defines the set \( S \). (See figure 5.)

**Fuzzy Set Operations**

\[
\begin{align*}
U_A \cap B(x) &= \min\{U_A(x), U_B(x)\} \\
U_A \cup B(x) &= \max\{U_A(x), U_B(x)\} \\
U_A(x) &= 1 - U_A(x)
\end{align*}
\]

Figure 5

In a later paper, "'Fuzzy Logic' and Its Application to Approximate Reasoning," [Zadeh 79], Zadeh defines fuzzy logic
in terms of fuzzy set theory. There are two stages to the "fuzzification" of logic. The first step is accomplished by allowing fuzzy predicates (from fuzzy set theory) into the logic. The logical connectives are defined in a manner similar to the set connectives, [Turner 84]. (See Figure 6.)

<table>
<thead>
<tr>
<th>Fuzzy Logical Connectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A = 1 - [A]</td>
</tr>
<tr>
<td>A &amp; B = min{[A], [B]}</td>
</tr>
<tr>
<td>A or B = max{[A], [B]}</td>
</tr>
<tr>
<td>A -&gt; B = min{1, (1 - [A]) + [B]}</td>
</tr>
</tbody>
</table>

A and B are logical statements. [A] is the fuzzy value associated with A.

Figure 6

The second stage is to make truth and falsehood into fuzzy values. This is done by dividing [0,1] into a finite number of fuzzy subsets, referred to as linguistic truth values. Examples of linguistic truth values include true, false, not true, not false, not very true, not very false etc. In general the membership function for true is defined first, the others being defined in terms of true.

Turner points out several difficulties with fuzzy logic. There is no guarantee that the set of linguistic truth values is closed under the logical connectives. To avoid this, an
approximation function is used to map arbitrary fuzzy subsets of \([0,1]\) into the set of truth values. This however leads to questions of determining what the best approximation to a set is.

One of the first systems to use a fuzzy logic formalism was MYCIN [Shortliffe 76]. A MYCIN certainty factor is a number in the range \([-1,1]\) associated with each attribute-object-value triple in the context tree. Negative certainty factors represent evidence for disbelief, positive factors indicate belief. Antecedent clauses are evaluated using the rules for logical connectives given for a fuzzy logic. A certain amount of fuzziness in truth is afforded by the use of varying thresholds for allowing rules to fire. For example (DEFIS CNTXT PARM value) is considered true if PARM is equal to value and its certainty is 1, while (SAME CNTXT PARM value) is true if PARM is equal to value and its certainty is \(>0.2\). The formula for computing the certainty of a concluded clause takes into account the previous certainty of the parameter in order that certainty factors can be strengthened by later data. The formula is given in Figure 7.

4.3 Fuzzy Reasoning in the GLISP Rule Language

Because there is not one clear "best" system for reasoning with uncertainty, the GLISP rule language leaves the choice up to the programmer. At present, by using GLISP objects with no certainty factor associations, the programmer can completely
ignore fuzzy reasoning; or with context types, use the fuzzy reasoning package provided. There is always the capability for a programmer designed system.

The fuzzy reasoning package consists of two parts: fuzzy evaluation functions used in the LHS of a rule, and three fuzzy assertion functions used in the RHS of a rule. (See Appendix A, Section 4.) A fuzzy evaluation function is either a logical connective (i.e. $\text{AND}$, $\text{OR}$, $\text{NOT}$), or a predicate (i.e. $\text{EQ}$, $\text{LESSP}$, $\text{GEQ}$). The logical connectives implement Zadeh's original scheme, shown in Figure 5 above. Fuzzy predicates are conceived as membership functions, returning the degree of membership of its argument to the set of objects for which the predicate is true. For example ($\text{EQ X 'RED}$) defines the fuzzy set of RED objects. The LHS of a rule is considered true if the fuzzy value it returns exceeds the threshold set with the global variable HPSTHRESHOLD.
For example, consider a context with the following fuzzy parameter values: \(
\text{COLOR} = \text{RED} (0.8) \)
\[
\text{HARDNESS} = 7.5 (1.0).
\]
Then the fuzzy expression:
\[
(\& (\equiv \text{COLOR} \ '\text{RED}) (\leq \text{HARDNESS} 7))
\]
evaluates to the fuzzy value 0.8 which exceeds \text{HPSTHRESHOLD}.
Therefore if this were the LHS of a rule, the rule would be eligible to fire.

A second form of the fuzzy predicate compares two objects, returning the degree to which both belong to some fuzzy set. If the two arguments do belong to the same set, the minimum of their fuzzy values is returned. For example, assume the color of \text{MINERAL-1} is \text{RED} with certainty 0.7, and \text{MINERAL-2} is \text{RED} with certainty 0.5, then \((\equiv \text{MINERAL-1:COLOR} \text{MINERAL-2:COLOR})\)
returns 0.5.

In addition to the predefined fuzzy predicates, programmers are given the ability to define their own fuzzy predicates. The functions \text{DEFINEFP} and \text{DEFINEFPP} create fuzzy functions of the first form and the second form respectively. See Chapter 6, the User's Manual, Section 4.1.5.

Finally, functions are provided to associate a certainty factor with a value when new assertions are made. There are three sources of uncertainty in making an assertion: the certainty of the LHS, the certainty of the assertion, (i.e. the
extent to which it is believed that the evidence implies the conclusion), and the prior certainty of the asserted hypothesis. The three predefined assertion functions, $ASIS, $ASSERT, and $ADJUST differ in which of the three factors influence the final certainty associated with an hypothesis.

The $ASIS function considers only the certainty of the assertion itself. It allows the assertion of strong belief in a conclusion regardless of the strength of the evidence. In a chemical plant, if an employee smelled something which was thought to be a volatile substance leak, the system would want to assert that the leak should be investigated because the potential danger is much higher than the cost of a false alarm. Example:

(RULE-1 (*Sound the alarm when toxic)
   IF ($EQ ODOR 'GASEOUS)
   THEN (($ASIS SPILL 'DANGER 'HIGH 1.0)))

Drilling for oil is a very expensive proposition however. An expert system advisor should make recommendations based on the strength of the evidence. With $ASSERT, the strength of the hypothesis relies on the evidence and the strength of the assertion. $ADJUST takes the current certainty of a hypothesis into consideration as well, using the MYCIN equation, (Figure 7). The equation is computed with the function HPSCOMPUTECF, which the programmer may modify to suit the particular domain. This
allows a wide range of certainty calculations to be possible without rewriting the entire package. Example:

(RULE-2 (* If natural gas is detected, that is further evidence of oil)
  IF ($GEQ GAS-LEVEL 1.5)
  THEN (($ADJUST SITE 'OIL? T 0.7)))

(RULE-3 (* Dig if the evidence warrants it.)
  IF ($EQ OIL T)
  THEN (($ASSERT SITE 'DRILL T 0.5)))

In RULE-2, the presence of natural gas provides additional evidence that an oil well is nearby. Thus the certainty of OIL? is adjusted to record this fact. RULE-3 however, does not consider that previous evidence may also support the decision to drill.
Chapter 5
Contexts

There are many possible forms that the database can take in the GLISP rule language. A package for defining a specific type of database called a context is provided. The purpose of the context description package is to simplify the declaration of objects which will be accessed by the fuzzy evaluation functions discussed in the previous chapter.

5.1 Context Description

A context type is declared with the function HPSCONTEXTS which compiles context descriptions into GLISP object descriptions. Once compiled, context types may be used anywhere that a GLISP object type is used. Additional information is kept about contexts which is used in translating parameters and asserting values for them. This information is kept on the property list of the atom which is the name of the context type. A sample context declaration is given below. (See Appendix A, Section 3.)

(HPSCONTEXTS
(MINERAL
  PRINT-ID MINERAL-
  INIT INITIALIZE-MINERAL
  PARMS(}
(NAME MULTIVALUE ATOM
TRANS (The name of * is))

(COLOR SINGLEVALUE ATOM
TRANS (The color of * is) PROMPT (What is the color of *?)

(VALUE PROP (LOOKUP-VALUE NAME)
TRANS (The monetary value of * is))...

CONTEXTS ((EMBEDDED-IN MINERAL)))

The PRINT-ID is an atom which is used in printing translations of parameters. When a context instance is created with the HPSCREATE function, a unique identifying number is appended to the PRINT-ID and placed on the object's property list under the property PRINT-ID. INIT is a function which is called to set parameters to default values, request data from users, or automatically call RuleSets to deduce values for parameters when a new context instance is created with HPSCREATE. (See 5.5).

5.2 Parameters

The actual data associated with a context is stored in parameters. A parameter has a name by which it is accessed, a type and description which determine its structure in the compiled GLISP object, and optional translation and prompt fields used for reporting and requesting values of parameters.

There are six types of parameters: Singlevalue, absolute, multivalue, prop, adj, and isa. The first three are stored parameters; their values are property list elements of the
context type. Singlevalue parameters are allowed to have one value, with an associated certainty factor; absolute parameters have a single value, but their certainty is always assumed to be 1.0; multivalues are parameters which have a list of values, each with its own certainty factor. The description for a stored parameter is a GLISP object structure description. The remaining three types are computed parameters; their values are derived from other parameters in the context. Each of the types prop, adj, and isa corresponds to a GLISP object property; its description is a function name or a GLISP expression which computes its value.

Within fuzzy evaluation predicates, parameter data is accessed by its name alone. For example, the function to test if the color of a mineral M is red could be written as \((\text{\$EQ M:COLOR 'RED})\) or \((\text{\$EQ COLOR 'RED})\) if COLOR is not ambiguous within the RuleSet. In all other cases, (i.e. outside of fuzzy predicate evaluation), the value of a parameter is accessed as Parameter-name:VALUE.

5.5 Subcontexts

Contexts can be arranged in a tree structure similar to EMYCIN's context tree. The CONTEXT declaration is a list of pairs, an access name and a context type which are to be subcontexts of the declared context. A context instance may have one or more instances of each of its subcontexts stored with it.
An example of subcontext use would be in disease diagnosis. A patient context is defined in part below:

(PATIENT
  PRINT-ID  PATIENT-
  INIT      GET-LAB-DATA
  PARMS(...)  
  CONTEXTS((DISEASES DISEASE) (TREATMENTS TREATMENT)) ...)

As the diagnosis progresses, new disease contexts will be added. RuleSets could be invoked for each of them to suggest treatments.

5.6 GLISP Object Compatibility

The fields MSG, SUPERS, and VALUES can be added to a context declaration, using the same form used in GLISP object declarations. These fields were added to provide the full expressive power of GLISP objects to contexts.

5.7 Context Instances

Context declarations create types; the HPSCREATE function creates instances of those types. HPSCREATE takes a type name and a GLISP object of that type as arguments. Generally it is called as below:

  (HPSCREATE 'type (A type))

and returns a context instance. The creation function gives the context its unique print identification, binds it to its respective type (for purposes of finding translations and prompts), and applies the initialization function to it. Context
instances are atoms, each of whose property list contains its stored parameters and subcontexts. Below is an example of context creation within a RuleSet.

\[(\text{DISEASE-RULE-1}) \quad (* \text{If spots are evident, start a disease context for measles})\]

\[\text{IF} \quad (\text{SEQ HAS-SPOTS T})\]

\[\text{THEN} \quad ((\text{DISEASES} \leftarrow +\text{SPOTS}))\]

\[(\text{HPSCREATE 'DISEASE (A DISEASE WITH SYMPTOMS = (LIST 'SPOTS))}))\]

The initialization function for the disease context could then request more data about the symptoms and call RuleSets for recommending a treatment.

5.8 Advantages of Context Use

Contexts hide the implementation of parameter type structures, which makes them clear and understandable. They are specially formulated for use with fuzzy evaluation and assertion functions: any of the parameter types can be referenced within a fuzzy evaluation function. Any stored type can be updated with a fuzzy assertion function.

Contexts also allow the storage of descriptive information about parameters, which can be used in building a user interface. The descriptions are kept in one place, reducing redundant storage of data. This information is automatically updated by the HPSCREATE function for context instances. Further, the use
of initialization functions allows RuleSets to be called implicitly when new subproblems arise.

5.9 Disadvantages

One major disadvantage of both contexts and GLISP objects as database elements for RuleSets is that the programmer can define structures which may look efficient in GLISP, but drastically slow the system because they are inefficient to implement in Interlisp. It is up to the programmer to carefully choose a data structure which is both useful and efficient.
Chapter 6
Conclusions

A project such as this cannot be considered complete. At present, the GLISP rule language is very flexible, giving it a wide range of applicability. It makes use of GLISP's English-like syntax for readability and concise code, as well as the optimization which the GLISP compiler affords. These qualities make it feasible to extend the language with additional packages, supporting more problem solving techniques.

In doing this project I have learned much about the tradeoffs one faces in designing any system, especially expert systems tools. In order to make the tool flexible, a certain amount of efficiency and completeness were sacrificed. To provide the user with some support, e.g. certainty factors and fuzzy evaluation, some flexibility had to be given up.

Some of the problems with the system can be solved with more research. At present rule optimization is minimal. Specific techniques for each of the control mechanisms supported could be implemented; other control methods could be found. Another limitation is the lack of strong user interface support. It may be necessary to define strict subsets of the rule language in order to implement explanation facilities.
Other future extensions could add more predefined packages from which the user can choose. An alternative to contexts which supports a different method of reasoning with uncertainty would be beneficial, as would the addition of other rule evaluation predicates. Contexts themselves could be extended to include more descriptive information, which would in turn aid in designing an explanation system.

Blackboards and meta-rules are possible architectures already; but the user must design them for each project. As expert systems are built with these architectures, a generic design may suggest itself and warrant inclusion as a package for the rule language.

Designing and implementing the GLISP rule language has been a valuable experience for me. Researching previously designed systems gave me familiarity with what is currently available; building the new system taught me what to look for in a good tool. It is this authors hope that the GLISP rule language will spur interest in designing complete problem solving environments.
Appendix A

GLISP Rule Language
User's Manual

1. Rules

Rules in the GLISP rule language are productions of the form, IF <expr> THEN <exprs>, where <expr> is an arbitrary GLISP expression, and <exprs> is a list of GLISP expressions. From now on, the IF <expr> will be referred to as the left hand side (LHS) of the rule; the THEN <exprs> will be referred to as the right hand side (RHS) of the rule. Rules are defined within RuleSets (See Section 2. of this chapter), and carry additional information which is used to control their execution within their respective RuleSets. The complete syntax of a rule declaration is given below:

```
[<Rule-Name> <options> (* Comments)
 IF <expr>
 THEN(<expr1> <expr2> ... <exprn>))
```

Rule Declaration Syntax

1.1 Description of Rule Fields

The <Rule-Name> field is the only required field. If the others are present, they must appear in the order specified above.

1.1.1 Rule Name

The rule name is an atomic identifier which must be unique within the RuleSet in which the rule is defined. It serves to
identify a rule for auditing, tracing, and execution control purposes.

1.1.2 Options

There are two options which may be specified for a rule. One, both, or neither may appear in the option field; their order within the option field is irrelevant. The two options are ! (one shot rules), and a priority number.

If ! is present, the rule will fire only once in an invocation of the RuleSet. The priority number is used during conflict resolution to determine which of several eligible rules will fire. In general the rule with the highest priority will fire. If no priority is declared, a default value of 0 is assumed. Priority has no effect in the FIRST and ALL control schemes.

1.1.3 The LHS

The LHS of a rule is indicated by the keyword IF, which must be present if the rule has a nontrivial condition. If absent, the default LHS is T, indicating the LHS is always satisfied. The LHS expression can be any GLISP expression which evaluates to a truth value, (i.e. NIL or non-NIL) or to a fuzzy truth value (See Section 4. of this chapter).

A rule's LHS is evaluated implicitly with the function $SETCVT which returns T if the expression is not a fuzzy value and non-NIL, or if it is a fuzzy value and it exceeds the value of the threshold variable HPSTHRESHOLD. Otherwise it returns NIL. In
addition, the variable ANTECEDENT is set to the fuzzy value of
the expression for use in the RHS. $\text{SETCVT}$ treats NIL as 0 and
non-NIL, non-fuzzy values as 1.

1.1.4 The RHS

The RHS of a rule is indicated by the keyword THEN which must
be present if the rule has a nontrivial consequent. If absent,
the RHS is assumed to be NIL, indicating no action is to be taken
when the rule fires. The RHS expression is a list of GLISP
expressions which are evaluated, in the order declared, when the
rule is chosen to fire.

Frequently the RHS of a rule serves to assign a value to a
previously unknown parameter or to strengthen an assertion made
erlier, due to further evidence. In order to allow the
assertion to be modified by the strength of the evidence in the
rule, the variable ANTECEDENT, local to each RuleSet is set to
the certainty of the LHS and is available for use in RHS
expressions. This variable should not be modified by user
functions.

1.2 Example Rules

Below are several rule declarations as they would appear
within a RuleSet.

1.2.1 Robot Grocery Bagger

This is an example of a rule which uses only GLISP objects and
expressions in its evaluation. No options are specified.
(BLR-1  (* Get a new bag if the others are too full.)
  IF   (NOT (THE GROCERY-BAG WITH WEIGHT < = 22))
  THEN ((GROCERY-BAGS  + (A GROCERY-BAG WITH GROCERY-
                          ITEMS = (LIST 'BOTTOM))))

              Grocery Bagger Rule

1.2.2 Mineral Identification

This example uses a fuzzy predicate ($EQ) and a fuzzy
assertion function ($ASSERT) to determine the name of a mineral
from color evidence.

(NAME-1 3  (* if the color is red it could be iron.)
  IF   ($EQ COLOR 'RED)
  THEN (($ASSERT MINERAL 'NAME 'IRON .5))

              Mineral Name Rule

1.2.3 Chemical Spill Monitoring

Here, an expert system monitors the environment for possible
chemical spills. This rule has a high priority because it warns
workers of possible hazard. Once they are warned, they need no
be warned again, so the rule is set to fire once only.

(DANGER-1 ! 10  (* If the spill is possibly toxic, alert employees.)
  IF   ($AND ($UNKNOWN CONTAINED)
            ($GEQ TOXICITY 50))
  THEN (($ASSERT SPILL 'DANGER 'HIGH 1.0)
         (SOUND-ALERT SPILL))

              Chemical Spill Rule
2. RuleSets

RuleSets are collections of related rules which are executed with a common control paradigm. RuleSets compile into GLISP functions with GLISP objects or contexts as arguments. These functions are further compiled by GLISP into Interlisp, which can be compiled to machine code for faster execution.

A RuleSet is defined with the functions HPSRULESETS, which takes one or more RuleSet definitions as arguments. The original definition is stored on the property list of the RuleSet name atom, under the property HPSRULESET.

If the global variable HPSTRACERULES is non-NIL, RuleSets will be compiled to contain commands for printing trace information. If the global variable HPSNOISYFLAG is non-NIL, the compiled version of the RuleSet code will be prettyprinted after compilation.

2.1 RuleSet Definitions

A RuleSet definition is a list whose first element is the name of the RuleSet to be defined and whose remaining elements describe the form of the RuleSet. The syntax is given below. Only the RuleSet-Name and arguments are required. All other fields are optional and may appear in any order.

2.1.1 RuleSet Name

The RuleSet name is the atomic identifier to which the RuleSet definition and compiled GLAMBDAs definition are bound. The name
( <RuleSet-Name> <arguments>
RESULT <result declaration>
TERMINATE <terminate declaration>
BACKCHAIN <backchain declaration>
CONTROL <control type>
GLOBAL <global variable declarations>
LOCAL <local variable declarations>
RULES <rule declarations> )

RuleSet Definition Syntax

is also used in auditing and tracing the execution of programs. Consequently, RuleSet names must be unique.

2.1.2 Arguments

The arguments to a RuleSet are the data which the rules reference and modify. A RuleSet argument declaration is identical to a GLISP argument declaration. That is, it is a list of identifiers with optional type information. (See the GLISP User's Manual, pp. 15-16 for more information on type declarations.)

2.1.3 Result

The result declaration determines the value to be returned when the RuleSet terminates. It is a list whose first element is the expression to be evaluated and returned and whose second element is an optional type specification, used by the GLISP compiler to help determine the type of expressions which call the RuleSet. The default value for RESULT is NIL.
2.1.4 Terminate

The terminate declaration is used to specify the conditions under which iteration of the RuleSet should halt, and the actions to be performed at that time. The terminate condition acts as the UNTIL condition of a REPEAT-UNTIL loop, that is, it is checked after each iteration. The terminate declaration has the form given below.

\[(\langle\text{No-fire actions}\rangle (\langle\text{terminate condition}\rangle \langle\text{actions}\rangle))\]

Syntax of the Terminate Declaration

2.1.4.1 NoFire

A RuleSet always terminates when none of its rules are eligible to fire. The No-fire actions is a list of expressions to be evaluated, in the order declared, if execution of the RuleSet halts due to no rule firing.

2.1.4.2 Normal Termination

After each iteration of the select/fire cycle in a RuleSet, the termination condition is evaluated. If the value is non-NIL, the actions, a list of GLISP expressions, are evaluated in the order declared, the execution of the RuleSet is halted, and the value specified in the result declaration is returned.

If in place of an expression, the keyword ONE is used as the termination condition, only one iteration of the RuleSet is
performed at each invocation. It is still possible for the RuleSet to terminate because no rule has fired.

2.1.5 Backchain

RuleSets provide for a certain amount of backchaining. The backchain declaration is an association list whose elements are of the form: (Context-type (parameter1 parameter2 ... parameterN) ). This declaration informs the compiler that the RuleSet contains rules which assign a value to each of the parameters listed for the context type. The compiler builds an index from the declaration, which is used when evaluating parameters with the BACKCHAIN function. (See Section 5.7 of this chapter.)

2.1.6 Control

There are four control paradigms which determine the conflict resolution algorithm to be used within the RuleSet. The control declaration is the statement of one of the types: FIRST, ALL, PRIORITY, CERTAINTY. The default value is FIRST.

2.1.6.1 FIRST

The FIRST scheme indicates that the first rule whose LHS is satisfied will fire. Rules are evaluated in the order in which they were declared, until the LHS of one is true. The priority, if present, of a rule is ignored.
2.1.6.2 ALL
Conflict resolution by ALL specifies that every rule whose LHS
is satisfied is to be fired. The LHS's are evaluated in the
order declared and their RHS's are executed immediately upon
determining that the rule is eligible to fire.
2.1.6.3 PRIORITY
If the priority paradigm is chosen, the rule with the highest
priority out of all rules which are eligible to fire is chosen.
Ties are resolved by choosing the rule whose LHS certainty value
is the highest.
The order of rule declaration is irrelevant under either the
PRIORITY or CERTAINTY schemes. If no priority information is
given for a rule, a default value of 0 is assumed. Note: In
order to increase execution speed, once it has been determined
that no rule has a higher priority than one already in the
conflict set, the LHS's of remaining rules are not evaluated.
2.1.6.4 CERTAINTY
the CERTAINTY control paradigm is similar to the priority
paradigm. The difference lies in that the rule with the highest
LHS certainty is chosen to fire. Ties are resolved by priority
values. As in priority, not all LHS's will necessarily be
evaluated, since if a rule with certainty 1.0 is in the conflict
set and no remaining rule has higher priority than that rule, the
remaining rules have no chance to fire.
2.1.7 Local

Local declarations are a list of variables which will be used locally within the RuleSet. The syntax of a local variable declaration is identical to a GLISP PROG variable declaration.

2.1.8 Global

Global declarations are a list of global variables which will be known within the RuleSet. The syntax of a global variable declaration is identical to a GLISP GLOBAL variable declaration.

2.1.9 Rules

The rule declarations in a RuleSet is a list of rules as described in Section 1. of this chapter. The rules may contain arguments from the argument list, the local variable list, and the global variable list.

2.2 Example RuleSets

Below are two sample RuleSets using different types of databases and rule expressions.

2.2.1 Robot Bagger

This RuleSet contains the rules for putting large groceries in grocery bags. It uses the GLISP object GROCERY-ORDER as its database and GLISP functions and expressions in the rules.

2.2.2 Mineral Name Identification

This simplistic RuleSet asserts a name value for a mineral according to color evidence. The purpose here is to demonstrate
[BAG-LARGE-RULES (ORDER:GROCERY-ORDER)
  RESULT (ORDER GROCERY-ORDER)
  TERMINATE (NIL ((NOT (THE GROCERY-ITEM WITH SIZE = ‘LARGE))
                  (PRINL "No more large items to bag.")
                  (TERPRI))) )

CONTROL  FIRST
LOCAL  (I:GROCERY-ITEM)
RULES (  
  (BLR-1 (* Get a new bag if all are too full.)
     IF ((NOT (THE GROCERY-BAG WITH WEIGHT < = 22))
        THEN ((GROCERY-BAGS __ + (A GROCERY-BAG WITH
                       GROCERY-ITEMS = (LIST 'BOTTOM))))
  )

  (BLR-2 (* Put bottles in the bottom of the bag.)
    IF (I (THE GROCERY-ITEM WITH CONTAINER = 'BOTTLE))
    THEN ((THE GROCERY-BAG WITH WEIGHT <= 22) __ + I)
       (GROCERY-ITEMS __ - I))

  (BLR-3 (* Add other large items to bags.)
    IF (I (THE GROCERY-ITEM WITH SIZE = 'LARGE))
    THEN ((THE GROCERY-BAG WITH WEIGHT <= 22) __ + I)
       (GROCERY-ITEMS __ - I))) }

Robot Bagger RuleSet

the use of contexts, fuzzy predicates, and fuzzy assertion
functions within RuleSets.
[NAME-COLOR-RULES (M:MINERAL).
RESULT (M MINERAL)
BACKCHAIN ((MINERAL (NAME)))
CONTROL CERTAINTY
TERMINATE ((PRIN1 "No rule fired.")(TERPRI))
((C VT ($ KNOWN M : COLOR)) (REPORT M 'COLOR)) )

RULES (  
(NCR-1 3 (* If the color is black it could be coal)
  IF ($EQ M:COLOR 'BLACK)
  THEN (($ASSERT M 'NAME 'COAL 0.7)) )

(NCR-2 2 (* If the color is red it could be iron)
  IF ($EQ M:COLOR 'RED)
  THEN (($ASSERT M 'NAME 'IRON 0.7)) ) ]

Mineral Name Identification RuleSet
3. Contexts

A context is a special object type consisting of several fields called parameters which may be accessed and modified by the fuzzy evaluation and assertion functions, (See Section 4. of this chapter). Parameters have properties which indicate certainty, give descriptive information, or provide prompts for user input. Additionally, contexts contain fields found in regular GLISP object declarations such as MSG, SUPERS, etc.. Thus the context description language is as expressive as the GLISP object description language.

3.1 Context Declarations

Context types are declared with the function HPSCONTEXTS which takes one or more context declarations as arguments. The original declaration is stored on the property list of the name of the context, under the property HPSDEFINITION. The declarations are then compiled into GLISP object declarations and can be used in GLAMBDA functions and RuleSets. Context instances are created with the function HPSCREATE. (See Section 5.4 of this chapter.) The form of a context declaration is given below.

The context name is the only required field; the other property-value pairs are optional and may appear in any order. There may be only one PRINT-ID and one INIT field in a
( <Context Name>
PRINT-ID <identifier>
INIT <initialization function>
PARMS <parameter declarations>
CONTEXTS <subcontext declarations>
MSG <message declarations>
SUPERS <super classes>
VALUES <value declarations> )

Context Declaration Syntax

declaration. If more than one appears, the latest occurrence is used and the others are ignored.

3.1.1 Print Identifier

The print identifier is an atom which is used to uniquely name instances of a given context type. Whenever a context is instantiated with the function HPSCREATE, a unique sequence number is appended to the print identifier of the context type and the resulting atom is placed on the property list of the instance under the field name PRINT-ID. The print identifier is used when printing translations, prompts and values of parameters.

3.1.2 Initialization Function

The initialization function is any function which takes a single argument of the type defined by the context declaration. It is applied to a new context instance, created with the HPSCREATE function. Typically initialization functions will set parameters to constants, defaults, or user supplied values.
3.1.3 Parameter Declarations

Parameters are the data holding fields of a context. There are two basic types of parameters: stored and computed. Each of these types has three subtypes. The parameter declaration field is a list of parameter declarations which have the form given below.

( <Parameter Name> <type> <type dependent> TRANS <translation> PROMPT <prompt> )

Parameter Declaration Syntax

The parameter name, type, and type dependent fields are all required and must appear in the order given above. TRANS and PROMPT are optional and can appear in either order if both are present.

A translation or prompt is a list of text which may contain one or more instances of the atom *. When a translation or prompt is requested with GETPROMPT or GETTRANS, (See Section 5.3), * is replaced by the print identifier of the context to which the parameter belongs.

3.1.3.1 Stored Parameters (Absolute, SingleValue, MultiValue)

The data associated with stored parameters is kept on the property list of the atom which represents the context instance. Stored parameters are the only types which may have values asserted with the fuzzy assertion functions, (See Section 4.2).
The type dependent field of all stored parameters is a CLISP structure description. (See the GLISP User's Manual, pp. 6-10.) The subtypes of stored parameters designate how many values the parameter may have at one time and whether or not there is any uncertainty involved in the value(s).

1. **Absolute** - Parameters of the type absolute maintain a single value with which no uncertainty is associated. Absolute parameters compile to atoms whose binding is of the type given by their structure description. If a certainty factor is to be inferred from an absolute parameter, it has the value 1.0 if the parameter is non-NIL.

2. **SingleValue** - SingleValue parameters are similar to absolutes in that only one value is maintained and they compile to atoms with bindings of the type described in their structure description. However, SingleValue parameters also have a certainty factor which is stored as the CF property, which has the type `CERTAINTYF`. `CERTAINTYF` is predefined to be `REAL`, but may be modified by the user to be any GLISP type.

3. **MultiValue** - A MultiValue parameter has the form of a list of SingleValue parameters. Most often MultiValues represent the several hypotheses which may be formed about a problem from the given data. Each of the values of a MultiValue has its own certainty factor, CF, which has the type `CERTAINTYF`. 

3.1.3.2 Computed Parameters (PROP, ADJ, ISA)

It is often desirable not to store the value of a parameter, but to compute its value from other parameter values. For example, storing the age of a patient would require that the context be updated frequently. If the date of birth were stored and age computed from it, no updates would be needed. To provide this capability, the three types PROP, ADJ, and ISA may be specified as the type of a parameter.

The type dependent field of a computed parameter is a function name or a GLISP expression, as detailed in the GLISP User's Manual, pp. 5-6. Each of the subtypes compiles to a property of the respective type in the GLISP object generated by the compiler.

Computed parameters may be accessed by the fuzzy evaluation functions, provided the result type of the computation is one of the predefined object types: ABSOLUTEPARAM, SINGLEVALUEPARAM, or, MULTIPLEVALUEPARAM. They may not have values asserted by fuzzy assertion functions.

3.1.4 Subcontexts

A context may contain one or more instances of other contexts, called subcontexts. A subcontext declaration is a list of pars, (Name Type), where Name is the field name by which the subcontext is to be accessed and type is the GLISP object type or context
type name. Subcontexts compile into structures whose form is
(Name (LISTOF Type)).

3.1.5 MSG, SUPERS, VALUES

Each of these fields has the same syntax as its GLISP
counterpart. See the GLISP manual for a more detailed
description.

3.2 Example

The following example shows the declaration of a context which
could be used in the mineral identification system used as an
eexample earlier.
[HPSCONTEXTS
(MINERAL
PRINT-ID MINERAL-
INIT INITIALIZE-MINERAL
PARMS {
  (NAME MULTIVALUE ATOM
   TRANS (The hypothesized names of *)
  (COLOR SINGLEVALUE ATOM
   TRANS (The overall color of *)
   PROMPT (What is the overall color of *)
  (SCRATCHES-COMMON ABSOLUTE (LISTOF COMMON-OBJECT)
   TRANS (The objects that * scratches)
   PROMPT (What objects does * scratch ?)
  (SCRATCHES-MINERAL ABSOLUTE (LISTOF KNOWN-MINERAL)
   TRANS (The known minerals that * scratches)
   PROMPT (What known minerals does * scratch ?)
  (SCRATCHED-BY-COMMON ABSOLUTE (LISTOF COMMON-OBJECT)
   TRANS (The objects that scratch *)
   PROMPT (What objects scratch * ?)
  (SCRATCHED-BY-MINERAL ABSOLUTE (LISTOF KNOWN-MINERAL)
   TRANS (The known minerals that scratch *)
   PROMPT (What known minerals scratch * ?)
  (HARDNESS PROP (LOOKUP HARDNESS SCRATCHES-COMMON
   SCRATCHES-MINERAL SCRATCHED-BY-COMMON
   SCRATCHED-BY-MINERAL)
   TRANS (The hardness of *)
  (WHERE-FOUND ABSOLUTE LOCATION
   TRANS (The place * was found in)
   PROMPT (Where did you find * ?))
CONTEXTS (TESTS CHEMICAL-TEST))]

Mineral Context Declaration
4. Fuzzy Evaluation and Assertion

The rule language contains a set of predefined predicates which return fuzzy values in the range [0,1] rather than simply T or NIL, as well as a set of functions for asserting values for context parameters with fuzzy values. These are called the fuzzy evaluation and fuzzy assertion functions respectively.

Within RuleSets, LHS expressions are always considered to evaluate to a fuzzy value which is tested against the threshold variable HPSTHRESHOLD to determine if the rule is to fire or not. If an LHS expression evaluates to a non-fuzzy value, it is converted to 1.0 if non-NIL and 0.0 if NIL.

4.1 Fuzzy Evaluation Predicates

The FEPs, (Fuzzy Evaluation Predicates) are distinguished from regular predicates by the presence of one or two dollar signs before the predicate name. (For example: $EQ or $$MEMP). Every FEP has a non-fuzzy predicate with which it is associated. The FEP operates in a manner similar to the non-fuzzy predicate except it returns a real value in the range [0,1].

The first argument of an FEP must be one of the types ABSOLUTEPARM, SINGLEVALUEPARM, or, MULTIVALUEPARM. FEPs preceded by two dollar signs require that the first two arguments be one of these types. The remaining arguments may be of any type valid for the non-fuzzy version of the predicate, (i.e. the second argument of $LESSP must be a number). Note: parameters declared
as Absolutes, SingleValues, and MultiValues correspond to the above valid types.

4.1.1 Semantics of FEP Evaluation

FEPs are evaluated according to the type of parameter given to them. In the following definitions, $P$ and $$P$ are fuzzy predicates whose non-fuzzy counterpart is $P$. Parm is an object with one of the valid types for first arguments of fuzzy predicates.

1. SINGLEVALUEPARM and ABSOLUTE PARM

\[ ($P\ \text{Parm}\ \text{Value}_1\ \text{Value}_2\ ...) \]

If \( (P \{\text{value of Parm}\} \ \text{Value}_1\ \text{Value}_2\ ...) \)

Then, if the certainty factor of Parm is non-NIL, return it else return 1.0

Else, \( (\{P \text{ Parm} \ \text{Value}_1\ \text{Value}_2\ ...\} = \text{NIL}) \) return 0.0

Example: Assume MINERAL:COLOR is RED with certainty 0.8 then \( ($EQ\ \text{MINERAL:COLOR} \ '\text{RED}') \) returns 0.8

\[ ($$$P\ \text{Parm}_1\ \text{Parm}_2\ \text{Value}_1\ \text{Value}_2\ ...) \]

If \( (P \{\text{value of Parm}_1\} \ \{\text{value of Parm}_2\} \ \text{Value}_1\ \text{Value}_2\ ...) \)

Then, if at least one of the parm has a non-NIL certainty factor, return the minimum certainty of the two parameters, else return 1.0

Else return 0.0
Example: Assume MINERAL1:HARDNESS is 7 with certainty 0.8

Assume MINERAL2:HARDNESS is 8 with certainty 0.6

then ($$LESSP MINERAL1:HARDNESS MINERAL2:HARDNESS) returns 0.6

($$EQ MINERAL1:HARDNESS MINERAL2:HARDNESS) returns 0.0

2. MULTIVALUEPARAM

($$P Param Value1 Value2 ...)

Assume Param has values $v_1$, $v_2$, ...

If there exists a value $v_i$ of Param such that ($P v_i Value1

Value2 ...)

Then, return the maximum certainty factor of all $v_i$ for which

the predicate is true. If all certainty factors are NIL,

return 1.0

Else, return 0.0

Example: Assume MINERAL:NAME has values IRON 0.8, COAL 0.7, and

DIAMOND 0.2

then ($$EQ MINERAL:NAME 'COAL) returns 0.7

($$P Param1 Param2 Value1 Value2 ...)

Assume Param1 has values $v_{1,1}$ $v_{1,2}$ ... and Param2 has values

$v_{2,1}$ $v_{2,2}$ ...

Collect the set of all pairs ($v_{1,i}$ $v_{2,j}$) such that
(P v1,i v2,j Value1...) is true.

If this set is empty, return 0.0

If it is non-empty: the certainty of a pair is the minimum of
the certainties of the two values, (Null certainty = 1.0).
Return the maximum certainty of the set of pairs.

Example: Assume MINERAL1:NAME has the values given above.

Assume MINERAL2:NAME has values DIAMOND 0.7 COAL 0.5.
then ($EQ MINERAL1:NAME MINERAL2:NAME) returns 0.5 because
the set of pairs which satisfy (EQ X Y) is {((DIAMOND(0.2)
DIAMOND(0.7)), ((COAL(0.7) COAL(0.5))}. The certainties of
the pairs are {0.2, 0.5} and the maximum of this set is 0.5

4.1.2 Predefined FEPs

Each of the predefined FEPs listed below is defined in terms
of its usual LISP predicate.

$EQ  $NEQ  $EQUAL  $NEQUAL  $LESSP  $LEQ
$GEQ  $GREATERP  $ZEROP  $MEMB  $MEMBER  $TRUE
$$EQ  $$NEQ  $$EQUAL  $$NEQUAL  $$LESSP  $$LEQ
$$GREATER
4.1.3 Fuzzy Logical Connectives

4.1.3.1 $\text{AND}$

$\text{AND}$ takes an arbitrary number of fuzzy expressions and/or S-expressions and returns the minimum. Non-NIL S-expressions are considered fuzzy values of 1.0.

4.1.3.2 $\text{OR}$

$\text{OR}$ takes an arbitrary number of fuzzy expressions and/or S-expressions and returns the maximum. Non-NIL S-expressions are considered fuzzy values of 1.0.

4.1.3.3 $\text{NOT}$

$\text{NOT}$ takes a single fuzzy expression and returns its complement. The complement of a fuzzy expression is 1.0 minus its fuzzy value. Note: ($\text{NOT}$ ($\text{EQ}$ $X$ $Y$)) is not the same as ($\text{NEQ}$ $X$ $Y$). For example, if MINERAL:COLOR is RED with certainty 0.8, then ($\text{NOT}$ ($\text{EQ}$ MINERAL:COLOR 'RED')) returns 0.2; ($\text{NEQ}$ MINERAL:COLOR 'RED) returns 0.0.

4.1.4 Other Special Fuzzy Functions

4.1.4.1 $\text{KNOWN}$

$\text{KNOWN}$ takes a single parameter as an argument and returns the certainty associated with its value if the value is non-NIL, in which case it returns 0.0. For ABSOLUTEPARMS, 1.0 is returned for non-NIL values; for MULTIVALUEPARMS, the maximum certainty over all of its values is returned.
4.1.4.2 $UNKNOWN

$UNKNOWN takes a single parameter as an argument and returns
the degree to which the parameter is not known. It is defined as
1 minus ($KNOWN P).

4.1.4.3 $CVT

$CVT takes a single fuzzy expression and compares it to the
threshold variable HPSTHRESHOLD, whose default value is 0.2. If
the certainty of the fuzzy expression exceeds the threshold, $CVT
returns T, else it returns NIL.

4.1.4.4 $SETCVT

$SETCVT takes an atom and a fuzzy expression as its two
arguments. The certainty of the fuzzy expression is assigned to
the atom and the value which would be returned by $CVT on the
fuzzy expression is returned.

4.1.5 User Defined Fuzzy Predicates

There are two functions provided which allow programmers to
define their own fuzzy evaluation predicates. These functions
are DEFINEFP and DEFINEFPP. Both take an identifier for their
first argument and a predicate form for their second. Both
arguments are evaluated.

A predicate form is a list whose first element is the name of
the LISP predicate by which the fuzzy predicate is to be defined,
and whose remaining elements are dummy arguments to the
predicate. DEFINEFP creates a predicate which takes one
parameter and any number of values; DEFINEFP creates a predicate which takes two parameters and any number of values.

Example: $EQ was defined by (DEFINEFP '$EQ '(EQ X Y))

$$EQ was defined by (DEFINEFPP '$$EQ '(EQ X Y))

4.2 Fuzzy Assertion Functions

The rule language provides three different methods of asserting values and certainty factors to parameters. They differ in their use of the certainty of the LHS, (stored in the variable ANTECEDENT), and their propagation of previous certainty values.

All three functions take four arguments: the context in which the parameter value is being asserted, the name of the parameter, the new value, and a certainty factor in the range [0,1], indicating the certainty of the assertion.

4.2.1 $ASIS

$ASIS ignores the LHS certainty, simply giving the parameter the value and certainty stated as its arguments. For MULTIVALUEPARMS, if the parameter does not already have the value asserted, a new value is added with the certainty given. If the value is already present, its certainty is simply changed to the certainty given.

4.2.2 $ASSERT

$ASSERT assigns a new value to a parameter, using the LHS certainty in the calculation of the certainty of the parameter.
The value is assigned to the parameter as described for $ASIS$; the certainty is the certainty of the LHS times the certainty given in the $ASSERT$ call.

4.2.3 $ADJUST$

$ADJUST$ takes into account the current value of a parameter and its certainty, thus allowing the assertion to not only add new values, but to strengthen the certainty of old values. The function $HPSCOMPUTE_C F$ is used to determine the new certainty factor. It takes three arguments: the current certainty factor of the parameter, the certainty given as an argument to $ADJUST$, and the certainty of the LHS. It returns a fuzzy value using an EMYCIN-like calculation. In all other respects it operates as $ASSERT$ and $ASIS$. 
5. Run Time Functions

5.1 Editing

Both RuleSets and contexts can be edited with the DEdit interactive editor. To edit a context, DEdit is invoked with the function HPSEDC, which takes the name of a context type as an argument. If any changes are made, the context is recompiled automatically.

RuleSets are edited by calling the function HPSEDR with the RuleSet name as an argument. As with contexts, if any changes are made, the RuleSet is recompiled automatically.

5.2 Saving Definitions to a File

File package commands have been added to allow both context declarations and RuleSet definitions to be saved in files. To save a context, add the list (HPSCONTEXTS context1 context2 ...) to the <file>COMS variable. Similarly, RuleSets may be saved by adding (HPSRULESETS RuleSet1 RuleSet2 ...) to the <file>COMS variable. The compiled version of a context is saved as a GLISP object with (GLISPOBJECTS context1 context2 ...) and compiled RuleSets as (FNS RuleSet1 RuleSet2 ...).

Associated with the file package commands are the functions HPSPRETTYPRINTCONTEXT and HPSPRETTYPRINTRULESET which take a list of their respective type and prettyprint their definitions. The functions HPSGETDEFC and HPSGETDEFR retrieve the original definitions from the property list of the name atom for RuleSets.
and contexts. The definitions are returned as the value of the function call.

5.3 Translations and Prompts

The functions GETTRANS and GETPROMT take two arguments: a context instance and the name of a parameter in that context. GETTRANS returns the translation text for the parameter with the print identifier for the context substituted for *. GETPROMPT returns the prompt text for the parameter with the print identifier for the context substituted for *.

5.4 Creating Context Instances

The function HPSCREATE is used to initialize a context instance. It takes two arguments, a context type name, and an instance of that context. A unique print identifier is given to the instance, the binding of the instance is set to the context type (for purposes of retrieving information stored on the type name atom), and the initialization function is applied to the context before it is returned. Generally a call to create will have the following form: (SETQ INSTANCE (HPSCREATE 'TYPE-NAME (A TYPE-NAME))).

5.5 Recompiling RuleSets

Once a system has been tested, the tracing of rules may no longer be necessary. The programmer would then set HPSTRACERULES to NIL and call HPSRECOMPILE for each of the RuleSets in the
system. HPSRECOMPILE takes a list of RuleSet names as its argument and returns NIL.

5.6 REPORT

The REPORT function is given to provide a minimal method of printing the values of parameters. REPORT takes two arguments: a context instance and a parameter name. The output of the REPORT function is the translation for the parameter, followed by its values and their certainties.

Example: (REPORT MINERAL1 'NAME)

The NAME of MINERAL-1 is COAL (0.7)
IRON (0.5)

5.7 Backchaining

The BACKCHAIN declaration in a RuleSet creates an index of RuleSets which assert values for given parameters in a context. The function BACKCHAIN uses this index to determine the value of an unknown parameter.

BACKCHAIN takes two arguments: a context instance and a parameter name. If the parameter already has a value asserted for it, that parameter is returned. If no value has been asserted, all of the RuleSets in the index which may provide a value for it are applied to the context. The parameter is then returned. Note: the parameter is returned, not its value. Therefore calls to BACKCHAIN can be nested in calls to fuzzy evaluation functions.
BACKCHAIN maintains a stack of parameters for which it is attempting to find a value. If an attempt is made to BACKCHAIN over a parameter which is already on the stack, the evaluation stops and the parameter, with no value asserted is returned.

Example: RuleSet-1 has been declared to assert a value for the NAME parameter of a mineral. The following rule is in another RuleSet:

IF ($EQ (BACKCHAIN MINERAL1 'NAME) 'COAL)
THEN (($ASSERT MINERAL1 'LOCATION 'MINNESOTA 0.7))
6. Code Files

The code for the rule language is in four files:

HPSR.LSP    - RuleSet compiler
HPSC.LSP    - Context compiler
HPSFUZZY.LSP - Fuzzy predicates
HPSRUN.LSP  - Run time functions
Appendix B

HPSR.LSP
(DEFINEQ
(HPSRULESETS
[FUNCTION HPSCOMPILERULESET])

(HPSCOMPILERULESET
[FUNCTION HPSRULESETS]

("* edited: 05-May-90 10:24" This function is called
by the user to compile a rule set into GLISP code.)

("* edited: 15-Jan-90 03:45" This is the main compiler function. It takes a rule set as input, parses it, places
the original definition on the property list, and returns the code compiled by the GLISP compiler.)

(PROG (NAME ARGS RESULT PROPDEF GLOBAL RULES FNCODE TERMINATE CONTROL BACKCHAIN OUTLOOP DOLOOP
*HSOCOD* INCODE COMPARATOR)
  (COND
    ((LISTP (SEQ NAME (pop RSDOF)))
      (HPSCOMPILERULESET NAME NAME)
      (T NIL))
    ((T (COND
        ((NOT (MEMQ NAME HPSRULESETS))
         (SETQ HPSRULESETS (CONS NAME HPSRULESETS))
         (HPSRULESETS RSDOF)
         (PUTPROP NAME (QUOTE HPSBACKCHAINNAME))
         (BACKCHAIN HPSBACKCHAIN NAME BACKCHAIN)
         (HPSRULESETTER (SEQ INCODE (LIST (QUOTE HPSBACKPRINTS))
                           (QUOTE NAME))
         (QUOTE HPSAUDIT)
         (LIST (QUOTE CONS)
               (QUOTE NAME)
               (QUOTE HPSAUDIT))
         (PUTPROP NAME (QUOTE HPSRULESET)
                      (QUOTE RSDOF))
         (SEQ ARGS (LIST (QUOTE GLAMBA) ARGS))
         (SETQ GLOBAL (COND
                         (GLOBAL (CONS "GLAMBA GLOBAL")
                         (T NIL))
                         (SETQ TERMINATE (HPSGLOBAL INCODE TERMINATE RESULT))
                         (QUOTE RESULT)
                         (QUOTE (QUOTE CONS)
                                 (QUOTE NAME)
                                 (QUOTE HPSAUDIT))
                         (T NIL))
         (SELECTQ CONTROL
                       (QUOTE GLOBAL)
                       (QUOTE Certain))
         (T NIL))

    )
  )
)

("* edited: 22-May-90 10:24" This function is called
by the user to compile a rule set into GLISP code.)

("* edited: 15-Jan-90 03:45" This is the main compiler function. It takes a rule set as input, parses it, places
the original definition on the property list, and returns the code compiled by the GLISP compiler.)

(PROG (NAME ARGS RESULT PROPDEF GLOBAL RULES FNCODE TERMINATE CONTROL BACKCHAIN OUTLOOP DOLOOP
*HSOCOD* INCODE COMPARATOR)
  (COND
    ((LISTP (SEQ NAME (pop RSDOF)))
      (HPSCOMPILERULESET NAME NAME)
      (T NIL))
    ((T (COND
        ((NOT (MEMQ NAME HPSRULESETS))
         (SETQ HPSRULESETS (CONS NAME HPSRULESETS))
         (HPSRULESETS RSDOF)
         (PUTPROP NAME (QUOTE HPSBACKCHAINNAME))
         (BACKCHAIN HPSBACKCHAIN NAME BACKCHAIN)
         (HPSRULESETTER (SEQ INCODE (LIST (QUOTE HPSBACKPRINTS))
                           (QUOTE NAME))
         (QUOTE HPSAUDIT)
         (LIST (QUOTE CONS)
               (QUOTE NAME)
               (QUOTE HPSAUDIT))
         (PUTPROP NAME (QUOTE HPSRULESET)
                      (QUOTE RSDOF))
         (SEQ ARGS (LIST (QUOTE GLAMBA) ARGS))
         (SETQ GLOBAL (COND
                         (GLOBAL (CONS "GLAMBA GLOBAL")
                         (T NIL))
                         (SETQ TERMINATE (HPSGLOBAL INCODE TERMINATE RESULT))
                         (QUOTE RESULT)
                         (QUOTE (QUOTE CONS)
                                 (QUOTE NAME)
                                 (QUOTE HPSAUDIT))
                         (T NIL))
         (SELECTQ CONTROL
                       (QUOTE GLOBAL)
                       (QUOTE Certain))
         (T NIL))

    )
  )
)
(HPSPARSERULES)

(LUMEDA (RDEF))

(PROG NIL)

(HPSSKPCCOMMENTS)

(SETO ARG3 (POP RSEDF))

(COND
  [(ATOM ARG3) (HPSSKPCCOMMENT ARG3 ARG3)]
  [(SETQ ARG3 NIL)])

LOOP(COND
  [(NULL RSEDF) RETURN])

(SETO RSEDF (HPSSKPCCOMMENTS RSEDF))

(SELECTQ (CAR RSEDF)
  [(BACKCHAIN BACKCHAIN BACKCHAIN)
     (SETQ BACKCHAIN (APPEND BACKCHAIN (CAR RSEDF))]
  [(RESULT Result result)
     (RESULT (CAR RSEDF))
     (TERMINATE (CAR RSEDF))]
  [(LOCAL Local local)
     (APPEND LOCAL LOCAL (CAR RSEDF))]
  [(RULES Rules rules)
     (HPSSKPCCOMMENTS (CAR RSEDF))]
  [(CONTROL Control control)
     (COND
       [(NOT (MEMB (CAR RSEDF)
                     (QUOTE FIRST FIRST PRIORITY Priority priority))]
       [(HPSSKPCCOMMENT (CAR RSEDF))]])]}

RETURN

(HPSPARSERULES)

("Launched:" "22-May-0676.28")

("Updated:" "22-May-0676.28")
(QUOTE HPSAUDIT)
  (CONS (QUOTE CONS)
    (QUOTE (CADDR RULE))
    (QUOTE HPSAUDIT)))
  (CCONS (QUOTE SETO)
    (CADDR RULE)
    T)
  (APPEND (CADDR RULE)
    (QUOTE (GO HPSOUI))
  (SETQ PROGVARS (APPEND PROGVARS (LIST (CADDR RULE))
    (T (LIST (QUOTE CONS)
      (CONS (LIST (QUOTE $SETC$))
        (QUOTE ANTECEDENT)
        (CAR RULE))
      (CCONS (QUOTE HPCSTRACEPRINT)
        (QUOTE (CADDR RULE))
      (CCONS (LIST (QUOTE SETO)
        (QUOTE HPSAUDIT)
        (LIST (QUOTE CONS)
          (QUOTE (CADDR RULE))
          (QUOTE HPSAUDIT)))
      (APPEND (CADDR RULE)
        (QUOTE (GO HPSOUI)))
    (T (CONS (QUOTE SETO)
      (QUOTE HPSAUDIT)
      (LIST (QUOTE CONS)
        (QUOTE (CADDR RULE))
        (QUOTE HPSAUDIT))))
  (APPEND (CADDR RULE)
    (QUOTE (GO HPSOUI)))
(HPSCOMPALL
  (LAMBDA NIL
    (PROG (BINDINGS HPSCONVERTVARS CODE)
      (HPSOPTIMIZE T)
      (SETQ PROGVARS (APPEND PROGVARS (APPEND HPSCONVERTVARS (LIST (QUOTE ANTECEDENT)
        (QUOTE NOFIRE))
      (FOR X IN BINDINGS do (COND
        ((ATOM X)
          (SETQ CODE (APPEND CODE (LIST HPSCOMPRULEALL))
            (POP RULES)
          (RETURN (CONS (QUOTE (SETQ NOFIRE T))
            CODE))
          (HPSCOMPRULEALL
            (LAMBDA (RULE)
              (COND
                ((CADDR RULE)
                  (PROCI (LIST 'COND (CONS (LIST 'AND (LIST 'NOT (CADDR RULE))
                    (LIST 'SETC$ ANTECEDENT (CAR RULE))
                  (CCONS (QUOTE HPCSTRACEPRINT)
                    (QUOTE (CADDR RULE))
                  (CCONS (LIST 'SETO 'HPSAUDIT)
                    (LIST 'CONS (QUOTE HPSAUDIT)
                      (QUOTE HPSAUDIT)))
                  (CONS (LIST 'SETO (CADDR RULE))
                    (APPEND (CADDR RULE)
                    (QUOTE (GO HPSOUI))))
      (QUOTE HPSAUDIT)
      (LIST (QUOTE CONS)
        (QUOTE (CADDR RULE))
        (QUOTE HPSAUDIT)))
      (APPEND (CADDR RULE)
        (QUOTE (GO HPSOUI)))
  (SETQ PROGVARS (APPEND PROGVARS (LIST (CADDR RULE))
    (T (LIST (QUOTE CONS)
      (CONS (LIST (QUOTE $SETC$))
        (QUOTE ANTECEDENT)
        (CAR RULE))
      (CCONS (QUOTE HPCSTRACEPRINT)
        (QUOTE (CADDR RULE))
      (CCONS (LIST (QUOTE SETO)
        (QUOTE HPSAUDIT)
        (LIST (QUOTE CONS)
          (QUOTE (CADDR RULE))
          (QUOTE HPSAUDIT)))
      (APPEND (CADDR RULE)
        (QUOTE (GO HPSOUI)))
  (SETQ PROGVARS (APPEND PROGVARS (LIST (CADDR RULE))
    (T (LIST (QUOTE CONS)
      (CONS (LIST (QUOTE $SETC$))
        (QUOTE ANTECEDENT)
        (CAR RULE))
      (CCONS (QUOTE HPCSTRACEPRINT)
        (QUOTE (CADDR RULE))
      (CCONS (LIST (QUOTE SETO)
        (QUOTE HPSAUDIT)
        (LIST (QUOTE CONS)
          (QUOTE (CADDR RULE))
          (QUOTE HPSAUDIT)))
      (APPEND (CADDR RULE)
        (QUOTE (GO HPSOUI))))
    (QUOTE (GO HPSOUI)))
(HPSCOMPALL
  (LAMBDA NIL
    (PROG (BINDINGS HPSCONVERTVARS CODE)
      (HPSOPTIMIZE T)
      (SETQ PROGVARS (APPEND PROGVARS (APPEND HPSCONVERTVARS (LIST (QUOTE ANTECEDENT)
        (QUOTE NOFIRE))
      (FOR X IN BINDINGS do (COND
        ((ATOM X)
          (SETQ CODE (APPEND CODE (LIST HPSCOMPRULEALL))
            (POP RULES)
          (RETURN (CONS (QUOTE (SETQ NOFIRE T))
            CODE))
          (HPSCOMPRULEALL
            (LAMBDA (RULE)
              (COND
                ((CADDR RULE)
                  (PROCI (LIST 'COND (CONS (LIST 'AND (LIST 'NOT (CADDR RULE))
                    (LIST 'SETC$ ANTECEDENT (CAR RULE))
                  (CCONS (QUOTE HPCSTRACEPRINT)
                    (QUOTE (CADDR RULE))
                  (CCONS (LIST 'SETO 'HPSAUDIT)
                    (LIST 'CONS (QUOTE HPSAUDIT)
                      (QUOTE HPSAUDIT)))
                  (CONS (LIST 'SETO (CADDR RULE))
                    (APPEND (CADDR RULE)
                    (QUOTE (GO HPSOUI))))
  (QUOTE HPSAUDIT)
  (LIST (QUOTE CONS)
        (QUOTE (CADDR RULE))
        (QUOTE HPSAUDIT)))
  (APPEND (CADDR RULE)
    (QUOTE (GO HPSOUI)))
(HPSCOMPALL
  (LAMBDA NIL
    (PROG (BINDINGS HPSCONVERTVARS CODE)
      (HPSOPTIMIZE T)
      (SETQ PROGVARS (APPEND PROGVARS (APPEND HPSCONVERTVARS (LIST (QUOTE ANTECEDENT)
        (QUOTE NOFIRE))
      (FOR X IN BINDINGS do (COND
        ((ATOM X)
          (SETQ CODE (APPEND CODE (LIST HPSCOMPRULEALL))
            (POP RULES)
          (RETURN (CONS (QUOTE (SETQ NOFIRE T))
            CODE))
          (HPSCOMPRULEALL
            (LAMBDA (RULE)
              (COND
                ((CADDR RULE)
                  (PROCI (LIST 'COND (CONS (LIST 'AND (LIST 'NOT (CADDR RULE))
                    (LIST 'SETC$ ANTECEDENT (CAR RULE))
                  (CCONS (QUOTE HPCSTRACEPRINT)
                    (QUOTE (CADDR RULE))
                  (CCONS (LIST 'SETO 'HPSAUDIT)
                    (LIST 'CONS (QUOTE HPSAUDIT)
                      (QUOTE HPSAUDIT)))
                  (CONS (LIST 'SETO (CADDR RULE))
                    (APPEND (CADDR RULE)
                    (QUOTE (GO HPSOUI))))
  (QUOTE HPSAUDIT)
  (LIST (QUOTE CONS)
        (QUOTE (CADDR RULE))
        (QUOTE HPSAUDIT)))
  (APPEND (CADDR RULE)
    (QUOTE (GO HPSOUI)))
(TEASVOL:COMPUTER SCIENCE:GODFREYASSAUSTIN@STUDENT:CHRISTOPHER:HPMSR2.LSF:2  10-Jul-06 22:20:08

(SETQ INITCODE (APPEND INITCODE (LIST (QUOTE SETO) (QUOTE RULES) (QUOTE COPY) (QUOTE RZ)))

(SETQ PROGVARS (APPEND PROGVARS HPSCONVERTVARS)

(Return (LIST (QUOTE (SETQ NOTIRE (HPSAURV rule) RULES))))

(QUOTE (COND (NOTIRE (COND

HPSTRAILERULES (HPSTRAILERULES (CADR (RIS-RULE))

(SETQ HPSAURV (CONS (CADR FIRE-RULE) HPSAURV))

(FUNCTION (null)

(QUOTE (COND (NOTIRE)

(CDR NOTIRE))

(QUOTE (RIS-RULES))

(T (SETQ NOTIRE T))

(HPSGCOMPARE:RULE)

(PROC (CD CD))

("edited: "30-Mar-06 09:30")

(SETQ CD (APPEND ARG'S (LIST (QUOTE COND))

(CONS (COND

(QUOTE)

(T (CAR RULES)))

(CDAR RULE)

(SETQ CD (GCOMP (CADR RULE))

(CD NIL NIL NIL))

(QUOTE)

((NULL CD))

(RETURN (ERROR))

((AND (LISTP (CAR CD)))

(RETURN (APPEND (LIST (COND

(QUOTE)

(T (CAR RULES)))

(CDAR CD))

(T (SETQ CD (CDR CD))

(GO UP)))

(HPSORTBYPRIOR:ETY)

((RIS RULES)

(LOOP (COND

(= (RIS RULES)

(RETURN CDAR RULES))

(SETQ TMP NIL)

(= (RIS RULES)

(GO LOOP)))

(QUICKER (CADR CDAR RULES))

(= (RIS RULES)

(CDR CDAR RULES))

(= (RIS RULES)

(GO LOOP)))

(= (RIS RULES)

(CDR CDAR RULES))

(= (RIS RULES)

(GO LOOP)))

(HPSAGENSY:TY)

((RIS TMP)

(= (RIS TMP)

(RETURN (QUOTE HPSAGENSY)

(QUOTE SEQUENCE))))
(DEFINITE)

(HPSOPTIMIZE (LAMBDA NIL)  
  (PROG (RSYM)  
    (SETQ RSYM (HPSGENSYM))  
    (SETQ RULES (COPY RULES))  
    (MAP RULES (FUNCTION LAMBDA (R)  
                  ((CAR (CADDR (CAR R)))  
                   (HPSOPTIMIZE (CADDR R)  
                                 R))  
                  (T (HPSOPTIMIZE (CADDR R)  
                                 R)))  
    (SETQ SEDINGS (APPEND SEDINGS (LIST (CAR (CADDR R)))))  
  )  
)

(HPSOPTIMIZE1 (LAMBDA (EXP RESTRULES)  
       (COND  
       (OR (ATOM EXP)  
            (EQ (CAR EXP)  
                 (QUOTE QUOTE)))  
       (NIL)  
       (HPSSESPEECP (CAR EXP)  
                    (MAP (CONDʧ  
                           (FUNCTION LAMBDA (X)  
                                            (HPSOPTIMIZE1 X RESTRULES))))  
       ))  
)

(HPSOPTIMIZE (LAMBDA (EXP RESTRULES)  
       (COND  
       (OR (ATOM EXP)  
            (EQ (CAR EXP)  
                 (QUOTE QUOTE)))  
       (NIL)  
       (HPSSESPEECP (CAR EXP)  
                    (MAP (CONDʧ  
                           (FUNCTION LAMBDA (X)  
                                            (HPSOPTIMIZE1 X RESTRULES))))  
       ))  
)
(MAP THEN-LIST
  (FUNCTION (LAMBDA (EXPR)
                (COND
                 ((EQUAL Expr (CAR Expr))
                  (PLACA Expr RSYM))
                 (SETQ SUCCESS T))
                 ((ATOM (CAR Expr))
                  (EQ (CAAR Expr)
                       (QUOTE QUOTE))
                  (HPSHONGOREPO (CAAR Expr))))
                NIL)
     T (HPSSUBSTAUX-THEN Expr (CAR Expr))))

(HPSSUBSTAUUXP
  (LAMBDA (EXPR IF-LIST)
    (MAP (CAR Expr)
      (FUNCTION (LAMBDA (EXPR)
                  (COND
                   ((EQUAL Expr (CAR Expr))
                    (PLACA Expr RSYM))
                   ((ATOM (CAR Expr))
                    (EQ (CAAR Expr)
                         (QUOTE QUOTE))
                    (HPSHONGOREPO (CAAR Expr))))
                  NIL)
     T (HPSSUBSTAUUXP Expr (CAR Expr))
     (RETURN SUCCESS)))

(HPSSUBST1
  (LAMBDA (EXPR HOMERULE)
    (PROG (FOUND-FIRST)
      (SETQ SUCCESS NIL)
      (COND
       ((EQUAL Expr (CAR HOMERULE))
        (PLACA HOMERULE (LIST (QUOTE SETQ)
                        RSYM Expr))
       (SETQ FOUND-FIRST T)
       (HPSSUBSTAUUXP-THEN Expr (CADR HOMERULE))))
      (COND
       ((ATOM (CAR HOMERULE))
        (EQ (CAAR HOMERULE)
             (QUOTE QUOTE))
        (HPSHONGOREPO (CAAR HOMERULE))))
      NIL)
     (HPSSUBST1P Expr HOMERULE)
     (HPSSUBSTAUUXP-THEN Expr (CADR HOMERULE))
     T NIL))

(HPSSUBST1P
  (LAMBDA (EXPR HOMERULE)
    (PROG (SUCCESS)
      (SETQ SUCCESS NIL)
      (MAP (CADR HOMERULE)
        (FUNCTION (LAMBDA (EXPR)
                    (COND
                     ((EQUAL Expr (CAR Expr))
                      (COND
                       ((FOUND-FIRST (PLACA Expr RSYM))
                        T (SETQ FOUND-FIRST T))
                       (PLACA Expr (LIST (QUOTE SETQ)))))
                     NIL)
        T (SUCCESS))
        (RETURN SUCCESS))
     (HPSHONGOREPO (CAAR HOMERULE)))
     (HPSSUBST1P Expr HOMERULE)
     (HPSSUBSTAUUXP-THEN Expr (CADR HOMERULE))
     T NIL))
(DEFINITE)

(HPSGETDEFR)
(LAMBD (NAME TYPE)
 (LIST (QUOTE HPSRULESET)
          (CONS NAME (GETPROP NAME (QUOTE HPSRULESET))))

(HPSDLODEFR)
(LAMBD (NAME TYPE)
 (PUTPROP NAME "HPSRULESET NIL"))

(HPSPRETTYPRUNTRULESETS)
(LAMBD (LST)
 (PROG (TMP OBJ)
     (TEPPR)
     (TEPPR)
     (PRINT (QUOTE "("))
     (PRINT (QUOTE HPSETLUSETS))
     (COND
      ((NULL LST)
       (TEPPR)
       (PRINT (QUOTE ")"))
      (TEPPR)
      (RETURN)))
     (SETQ OBJ (PROG LST))
     (COND
      ((SETQ TMP (GETPROP OBJ (QUOTE HPSRULESET)))
       (PRINT NIL T T "(" . FONT DEFAULT FONT T T 3 .PPV (CAR TMP))
       (MAP CAR TMP)
       (FUNCTION LAMBD (REST)
          (PRINT NIL T T 3 (CAR REST)
          10 .PPV (CARN REST)
          (PRINT (FUNCTION COD))
          (PRINT NIL "")
          (GO LP))))
     (GO LP))

(HPSDEF)
(LAMBD (RULESET)
     (EDIT (GETPROP (SETQ LAST HPSETLUSETS) (OR RULESET HPSETLASTRULESETEDITED))
           (QUOTE HPSRULESET))
     (RULESET) (QUOTE HPSETLUSETS)
     (QUOTE HPSETLCHANGE))

(HPSETLCHANGE)
(LAMBD (ATM EXP TYPE FLAG)
 (HPSCOMPLEMENTRULESET (CONS ATM EXP)))

("edited: "23-May-86 12:24")
("edited: "23-May-86 12:55")
(DEFUN RULE-SET (NAME) RULES)

(RULE-SET "COAL" """")

(RULE-SET "IRON" """"
  (RULE-1 """" If it is soft and black it is coal"
    (IF (AND (SEQ (COLOR (QUOTE BLACK))
               (SHORT (HARDNESS 9)))
         (ASSERT (NAME "COAL") .8)
         (REPORT (QUOTE NAME))
         (RULE-2 """" If it is red it is iron"
           (IF (SEQ (COLOR 'RED))
               (ASSERT (NAME "IRON") .8)
               (REPORT (QUOTE NAME))
               (RULE-3 """" Differentiate between coal and black diamonds"
                 (IF (AND (SEQ (COLOR (QUOTE BLACK))
                              (GREATERP (HARDNESS 9)))
                         (ASSERT (QUOTE DIAMOND) .8)
                         (REPORT (QUOTE NAME)))))

(DECLARE: DOEVAL LOAD DOECALC COMPIL DECODER COMPILER"

(ASSERT NLAMA HPSRULES)

(ASSERT NLAMA )

(ASSERT NLAMA )

STOP
Appendix C

HPSC.LSP
(DEFUN HPSDECONTEXT)
  ((NAME ID PROP ADJ ISA AGG SUPERS VALUES OBST))
  (COND
    ((LISTP (SETQ NAME (PROP CONTEXT)))
     (ERROR "BAD NAME FOR A CONTEXT"))
    ((NOT (MEMQ NAME HSCONTEXTS))
     (SETQ HSCONTEXTS (CONS NAME HSCONTEXTS)))
    (HPSINIT NAME)
    (SETQ SUPERS (QUOTE CONTEXT)))
  (LOOP (COND
      ((NULL CONTEXT)
       (GO LP3))
    (SELECTQ (CAR CONTEXT))
      ((PRINT-ID Print-Id Print-id print-id)
       (HPSDEERVEDP NAME (QUOTE PRINT-ID))
       (CAR CONTEXT))
    (PUTPROP NAME (QUOTE PRINT-ID)
       (CAR CONTEXT)))
    (INIT INIT TEST)
    (HPSDEERVEDP NAME (QUOTE INIT)
       (CAR CONTEXT))
    (PUTPROP NAME (QUOTE INIT)
       (CAR CONTEXT))))
  (PREFIX NAMES PREFIX)
  (HPSDEERVEDP NAME (QUOTE PREFIX)
       (CAR CONTEXT))
  (NPCSNAMEFS NAME (CAR CONTEXT)))
  (PRINT NAME (QUOTE NAME)
       (HPSDEERVEDP NAME (QUOTE NAME)
       (CAR CONTEXT)))
  (SETQ MSG (APPEND MSG (CAR CONTEXT))
      (SUPERS SUPERS (CAR CONTEXT))
    (HPSDEERVEDP NAME (QUOTE SUPERS)
       (CAR CONTEXT)))
  (VALUES VALUES VALUES)
  (HPSDEERVEDP NAME (QUOTE VALUES)
       (CAR CONTEXT))
  (VALUES VALUES (APPEND VALUES (CAR CONTEXT)))
  (CONTEXTS CONTEXTS CONTEXTS CONTEXTS CONTEXTS)
  (HPSDEERVEDP NAME (QUOTE CONTEXTS)
       (CAR CONTEXT))
  (NPCSNAMEFS (CAR CONTEXT))
  (HPSDEERVEDP NAME (CAR CONTEXT)
       (CAR CONTEXT)))
  (SETQ CONTEXT (CAR CONTEXT))
(((OR ATOM X)
  (NULL (CAR X)))
 (HPSBAOPARM X)
 (T SELECT) (CAR X))

((ABSOLUTE Absolute absolute)
 (HPSBAOPABS NAME X)
 (SINGLEVALUE Singlevalue Singlevalue Singlevalue)
 (HPSBAOPNM NAME X)
 (MULTIVALEU Multivalue Multivalue Multivalue)
 (HPSBAOPHUL NAME X))
 (ADJ AD AD)
 (HPSBAOPADS NAME X)
 (EQA E QA)
 (HPSBAOPAG NAME X) (HPSBAOP22 PROP X) (HPSBAOP22PROP NAME X) (HPSBAOP22PARM X) (HPSBAOP22PARM X)))

(HPSBAOPARM
 (LAMEGA (PARM))
 (PRG NIL
 (PRINT "HPSBAOPABS: THE PARAMETER ")
 (PRINT NAME)
 (PRINT " HAS BAD FORM. ")
 (TERM)
 (PRINT "THIS PARAMETER WAS IGNORED."
 (TERM))))

(HPSBAOAD
 (LAMEGA (PARNAME SD))
 (PRG NIL
 (PRINT "HPSBAOPABS: THE PARAMETER ")
 (PRINT NAME)
 (PRINT " HAS A BAD STRUCTURE DESCRIPTION."
 (TERM)
 (PP SD)
 (TERM)
 (PRINT "THIS PARAMETER WAS IGNORED."
 (TERM))))

(HPSBAOPARPROP
 (LAMEGA (PARNAME PROP))
 (PRG NIL
 (PRINT "HPSBAOPABS: THE PROPERTY ")
 (PRINT PROP)
 (PRINT " IN THE PARAMETER ")
 (PRINT NAME)
 (PRINT " IS NOT A RECOGNIZED PARAMETER PROPERTY. ")
 (TERM)
 (PRINT "THIS PROPERTY WAS IGNORED."
 (TERM))))

(HPSBAODARM
 (LAMEGA (CONTEXT PARAMETERFORM))
 (PUTPROP CONTEXT (QUOTE HPSBAOPARM)
 (CONS (APEND PARAMETERFORM (LIST (LIST (PRINT (QUOTE BACCHUS/HOPSC)))
 (GETPROP CONTEXT (QUOTE HPSBAOPARM))))))

(HPSPARSEOPARM
 (LAMEGA (PARM))
 (PRG NIL
 LOOP(COND
 (NULL PARM)
 (RETURN))
 (SELECT (CAR PARM)
 ((TRANS Trans Trans)
 (SETQ TRANS (CAR PARM)))
 (PROMPT PROMPT GROUP)
 (SETQ PROMPT (CDR PARM)))
 (HPSBAOPARM22PROP NAME (CAR PARM)))
 (SETQ PARM (CDR PARM))
 (GO LOOP))))
(RETURN))))

(HPSCOMPSA)
(LAMBDA (CONTEXT PARM)  (*edited: "22-May-8619:52")
  (COND
   ((LISTP (SETQ NAME (POP PARM)))
    (HPSCOADPARM PARM)
    (RETURN)))
   (POP PARM)
   (SETQ CD (LIST NAME (POP PARM)))
   (HPSPARSEPARM PARM)
   (HPSCOADPARM CONTEXT (LIST NAME (LIST 'TRANS TRANS)
    (LIST 'PROMPT PROMPT)
    (LIST 'TYPE 'ISA))
   (SETQ ISA (APPEND ISA (LIST CD)))
   (RETURN)))))

(HPSCOMPADJ)
(LAMBDA (CONTEXT PARM)  (*edited: "22-May-8619:51")
  (COND
   ((LISTP (SETQ NAME (POP PARM)))
    (HPSCOADPARM PARM)
    (RETURN)))
   (POP PARM)
   (SETQ CD (LIST NAME (POP PARM)))
   (HPSPARSEPARM PARM)
   (HPSCOADPARM CONTEXT (LIST NAME (LIST 'TRANS TRANS)
    (LIST 'PROMPT PROMPT)
    (LIST 'TYPE 'ADJ))
   (SETQ ADJ (APPEND ADJ (LIST CD)))
   (RETURN)))))

(HPSCOMPPROP)
(LAMBDA (CONTEXT PARM)  (*edited: "22-May-8619:54")
  (COND
   ((LISTP (SETQ NAME (POP PARM)))
    (HPSCOADPARM PARM)
    (RETURN)))
   (POP PARM)
   (SETQ CD (LIST NAME (POP PARM)))
   (HPSPARSEPARM PARM)
   (HPSCOADPARM CONTEXT (LIST NAME (LIST 'TRANS TRANS)
    (LIST 'PROMPT PROMPT)
    (LIST 'TYPE 'PROP))
   (SETQ PROP (APPEND PROP (LIST CD)))
   (RETURN)))))

(HPSCOMPCONS)
(LAMBDA (CONTEXTS)
  (MAPC CONTEXTS (FUNCTION (LAMBDA (X)
    (COND
      ((OR (ATOM X)
        (NOT (AND (ATOM (CAR X))
        (CDR X))
        (ATOM (CADR X))))
      (PRINT "HPSCOMPCONS: BAD SUBCONTEXT *")
      (PRINT X)
      (PRINT "SKIPPING...")
      (TERMP))
      (T (SETQ SD (APPEND SD (LIST (CAR X))
        (LIST (QUOTE LISTP)
        (CADR X)))))"))))
)

(DEFINE)

(HPSPRETTYPRETCCONTEXTS)
(LAMBDA (LIST)
  (COND)
  (TERMP)
  (TERMP)
(TEASLET:COMPUTER SCIENCE:UOF TEASLASTPRINT:STUDENT:CHRISTOPHER) LISP;5 20-Jun-86 07:56:42

(PRINT (QUOTE '))
(PRINT (QUOTE HPSCONTENTS))
(LP (COND ((NULL LST) (T (PRIVP ')))
          (CONSP LST)
          (RETURN)))
(SETP OBJ (POP LST))
(COND ((SETP TMP (GETPROP OBJ (QUOTE HPSDEFINITION)))
       (PRINT (= T T ""))
       (PRINT (LIST OBJ FONT DEFAULT FONT)
               (= T T 3 (CAR REST)))
       (FUNCTION COOR)
       (PRINT "'"))
       (GO UP)))

(HPSGETDEF)
  (LAMBDA (NAME TYPE)
    (LIST (QUOTE HPSCONTENTS)
           (CONS NAME (GETPROP NAME (QUOTE HPSDEFINITION)))))

(HPSDELDEF)
  (LAMBDA (NAME)
    (PUTPROP NAME "HPSDEFINITION NIL")
    (PUTPROP NAME "LSTRUCTURE NIL"))

(HPSDEL)
  (LAMBDA (CONTENTS)
    (DELETE (GETPROP (SETP LASTHPSCONTENTSEDITED OR CONTENTS HPSLASTCONTENTSEDITED)
                   "HPSDEFINITION")
         CONTENTS)

(HPSIFCHANGEDEC)
  (LAMBDA (ATN (NAME TYPE FLAG)
                (HPSDEFCONTEXT (CONS ATN (EXPR))))
    (DEFINE)

(HPSNEXTSEQ)
  (LAMBDA (CONTEXT)
    (PUTPROP CONTEXT (QUOTE HPSSEQUENCES)
     (ADD0 (GETPROP CONTEXT (QUOTE HPSSEQUENCES))))))

(HPSMPPROMPT)
  (LAMBDA (CONTEXT PARMNAME)
    (CAR (ASSOC (QUOTE PROMPT)
                 (CADR (ASSOC PARMNAME (GETPROP CONTEXT (QUOTE HPSPAREMPS)))))))

(HPSMPTRANS)
  (LAMBDA (CONTEXT PARMNAME)
    (CAR (ASSOC 'TRANS (CADR (ASSOC PARMNAME (GETPROP CONTEXT "HPSPAREMPS"))))))

(HPSMPRTYPE)
  (LAMBDA (CONTEXT PARMNAME)
    (CAR (ASSOC (QUOTE TYPE)
                 (CADR (ASSOC PARMNAME (GETPROP CONTEXT (QUOTE HPSPAREMPS))))))

(HPSMPRMS)
  (LAMBDA (CONTEXT PARMNAME)
    (CAR (ASSOC (QUOTE NAME)
                 (CADR (ASSOC PARMNAME (GETPROP CONTEXT (QUOTE HPSPAREMPS))))))}

(RFAQ TESTCONTEXT (MINERAL:PRINT-ID MINERAL: INIT:INITIALIZE-MINERAL-PARKS)
  (NAME: MULTIVALUE-ATOM-TRANS (THE NAME OF *))
  (GROUP: MULTIVALUE-ATOM-TRANS (THE GROUP BELONGS TO))
  (COLOR: SIMPLEVALUE-ATOM-TRANS)
(THE OVERALL COLOR OF *)
(PROMPT)
(WHAT IS THE OVERAL COLOR OF *)
(HARDNESS SINGLE VALUE REAL TRANS)
(THE GEOLOGICAL HARDNESS OF *)
(SCRAECHES ABSOLUTE (LIST OF ATOM)
(TANKS)
(THE THINGS * SCRATCHES)
(PROMPT)
(WHAT DOES * SCRATCH ?)
(SCRAECHED-BY ABSOLUTE (LIST OF ATOM)
(TANK)
(THE THINGS THAT SCRATCH *)
(PROMPT)
(WHAT SCRATCHES * ?)
(WHERE-FOUND ABSOLUTE LOCATION TRANS)
(THE PLACE * WAS FOUND)
(PROMPT)
(WHERE DID YOU FIND * ?))

CONTENTS
((TEST CHEMICAL-TEST)))

(READMS(1-1-TESTCONTENTED .REL))

(READMS HPSCONTEXTS (PRINT-ID MINERAL))

(REDEF (QUOTE HPSCONTEXTS) (QUOTE FILEPAGCOMS) (QUOTE
(COMP
(MACRO
(HPSCONTENT
(E

HPSRETR:PRINTCONTEXTS (QUOTE HPSCONTEXTS))))

(TYPE DESCRIPTION
"HPSCONTEXT DEFINITIONS"

GETDEF HPSCONTEXTDEFPC
DEFDEF HPSCONTEXTDEFPC))

(DECLARE: DONT-EVAL-LOAD DONT-EVAL-COMPILE DONT-COPY-COMPILER-VARS

(ADDTOVAR NAMMA HPSCONTEXTS)

(ADDTOVAR NAML)

(ADDTOVAR NAM)

(PUTPROP HPSC.LSP COPYRIGHT "CHRISTOPHER RATH" 1980)

STOP
Appendix D

HPSFUZZY.LSP
(defineq)

(sand)

(lambda fargs (cond (null fargs) 't) (t (apply quote 'min) (cons 1 (mapcon fargs (function (lambda (x) (cond ((fuzzyvalp (eval x)) (list (eval x))))))))))

(sor)

(lambda fargs (apply 'max (cons 0 (mapcar fargs (function (lambda (x) (cond ((fuzzyvalp (eval x)) (eval x)) (eval 1))) (null (eval x)) (t 0))))))

(snot)

(lambda (x) (cond ((fuzzyvalp 1) (difference 1 x)) (null x) (t 0))))

(sknown)

(lambda (park) (cond ((atom park) (cond (getprop park 'quote ch) (null (eval park)) (t 1)) (t (apply 'max (mapcar park (function known)))))))

(sunknown)

(lambda (park) (difference 1 (sknown park))))
(STRUE
  (LAMBDA ()
    (CONQ
      ((TRUE (VAL 1))
       (OR (GETPROP 'SF)
        (T)))))
  (T)
  (SEQ VAL 0)
  (MAP (FUN LAMBDA (DEFINITEP))
    (SEQ VAL (MAX VAL (STRUE DEFINITEP))
    (RETURN VAL)))))

(DEFINEP
  (LAMBDA (FRAME PFORM)
    (PROG (TM)
      (SETQ TMP
        (LIST
          FRNAME
          (LIST
            (QUOTE LAMBDA)
            (CAR PFORM)
            (LIST
              (QUOTE CONQ)
              (LIST
                (QUOTE PFORM)
                (CONS (QUOTE CONQ)
                  (CONS (QUOTE PFORM))
                  (CONS (QUOTE (CAR PFORM))
                    (CONS (QUOTE EVAL)
                      (CONS (QUOTE (CAR PFORM))
                        (CONS (QUOTE OR)
                          (CONS (QUOTE SETQ)
                            (QUOTE PFORM)
                            (QUOTE PFORM)))))))))))))

(CONQ
  (QUOTE T)
  (CONS
    (QUOTE PFORM)
    (CONS
      (QUOTE (CAR PFORM))
      (CONS
        (QUOTE SETQ VAL 0))
      (CONS (QUOTE NAPC)
        (CONS (QUOTE LAMBDA)
          (LIST
            (QUOTE PFORM)
            (LIST
              (QUOTE (DEFINITEP))
              (LIST
                (QUOTE SETQ)
                (QUOTE PFORM)
                (LIST
                  (QUOTE NAPC)
                  (CONS FRNAME)
                  (CONS (QUOTE PFORM)))))))))

(QUOTE ((RETURN PFORM)

  (CONS
    (QUOTE (DEFINITEP))
    (CONS
      (QUOTE PFORM)))))

(DEFINEP
  (LAMBDA (NAME FORM)
    (PROG (TM)
      (SETQ TMP
        (QUOTE T)
        (CONS (QUOTE PFORM))
        (CONS (QUOTE FRNAME)
          (CONS (QUOTE (DEFINITEP))
            (CONS (QUOTE PFORM)))))))))
(DEFUN NEQUAL (LAMBDA (X Y)) (NOT (EQUAL X Y)))

("edited: '20-May-86 12:49")

(DEFUN TRUE (LAMBDA (X)) (EQ X 1))

("edited: '18-Jul-86 14:09")

(DEFUN FUZZYVALP (LAMBDA (X) (AND (NUMBERP X) (GEO 1 X) (LIC X 1)))

("edited: '20-May-86 12:19")
(T T)

(T (PROG (RVAL)
    [SETQ RVAL 0]
    [MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
                        (SETQ RVAL (MAX RVAL (SAMEQUAL DEFINEFPX T)))
                        (RETURN RVAL))]

(SLESSP)
    [LAMBDA (X Y)
       (COND
           [(ATOM X)
              (COND
               [(LESSP (EVAL X) Y)
                (OR (GETPROP X 'CF)
                    T))]
           [(T 0)])
           (T (PROG (RVAL)
                 [SETQ RVAL 0]
                 [MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
                                     (SETQ RVAL (MAX RVAL (LESSP DEFINEFPX Y)))
                                     (RETURN RVAL))]

(SLEQ)
    [LAMBDA (X Y)
       (COND
           [(ATOM X)
              (COND
               [(LEQ (EVAL X) Y)
                (OR (GETPROP X 'CF)
                    T))]
           [(T 0)])
           (T (PROG (RVAL)
                 [SETQ RVAL 0]
                 [MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
                                     (SETQ RVAL (MAX RVAL (SLEQ DEFINEFPX Y)))
                                     (RETURN RVAL))]

(SGEQ)
    [LAMBDA (X Y)
       (COND
           [(ATOM X)
              (COND
               [(GEQ (EVAL X) Y)
                (OR (GETPROP X 'CF)
                    T))]
           [(T 0)])
           (T (PROG (RVAL)
                 [SETQ RVAL 0]
                 [MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
                                     (SETQ RVAL (MAX RVAL (SGEQ DEFINEFPX Y)))
                                     (RETURN RVAL))]

(SGREATERP)
    [LAMBDA (X Y)
       (COND
           [(ATOM X)
              (COND
               [(GREATERP (EVAL X) Y)
                (OR (GETPROP X 'CF)
                    T))]
           [(T 0)])
           (T (PROG (RVAL)
                 [SETQ RVAL 0]
                 [MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
                                     (SETQ RVAL (MAX RVAL (SGREATERP DEFINEFPX Y)))
                                     (RETURN RVAL))]

(SGREATERRP)
    [LAMBDA (X)
       (COND
           [(ATOM X)
              (COND
               [(SGREATERRP (EVAL X) Y)
                (OR (GETPROP X 'CF)
                    T))]
           [(T 0)])
           (T (PROG (RVAL)
                 [SETQ RVAL 0]
                 [MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
                                     (SETQ RVAL (MAX RVAL (SGERP DEFINEFPX Y)))
                                     (RETURN RVAL))]
(COND
  ((ATOM X)
    (COND
     ((EQ (EVAL Z) 0)
      (OR (GETPROP X 'CF)
       (T O)))
     (T (PROG (RVAL)
       (SETQ RVAL 0)
       (MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
         (SETQ RVAL (MAX RVAL (ZEROP DEFINEFPX)))
       (RETURN RVAL)))))))
  (SMEMB
    (LAMBDA (X Y)
      (COND
       ((ATOM X)
        (COND
         ((MEMBER (EVAL X) Y)
          (OR (GETPROP X 'CF)
           (T O)))
         (T (PROG (RVAL)
           (SETQ RVAL 0)
           (MAPC X (FUNCTION (LAMBDA (DEFINEFPX)
             (SETQ RVAL (MAX RVAL (SMEMBER DEFINEFPX Y)))
           (RETURN RVAL)))))))
       ))
    (DEFINEXP)
    (SSEQ
      (LAMBDA (X Y)
        (COND
         ((AND (ATOM X)
           (ATOM Y))
          (COND
           ((EQ (EVAL X) (EVAL Y))
            (MIN (OR (GETPROP X 'CF)
             (OR (GETPROP Y 'CF)
              (T O))))
            (T (PROG (RVAL)
              (SETQ RVAL 0)
              (MAPC Y (FUNCTION (LAMBDA (DEFINEFPX)
                (SETQ RVAL (MAX RVAL (SSEQ DEFINEFPX X)))
              (RETURN RVAL)))))))
         (T (ATOM Y))
         (T (DEFINEXP)
          (SSEQ Y X)
          (RETURN RVAL)))))
    (SSNEQ
      (LAMBDA (X Y)
        (COND
         ((AND (ATOM X)
           (ATOM Y))
          (COND
           ((EQ (EVAL X) (EVAL Y))
            (MIN (OR (GETPROP X 'CF)
             (OR (GETPROP Y 'CF)
             (T O))))
            (T (PROG (RVAL)
              (SETQ RVAL 0)
              (MAPC Y (FUNCTION (LAMBDA (DEFINEFPX)
                (SETQ RVAL (MAX RVAL (SSNEQ DEFINEFPX X)))
              (RETURN RVAL)))))))
         (T (ATOM Y))
         (T (DEFINEXP)
          (SSNEQ Y X)
          (RETURN RVAL))))
(INEQ (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1) (OR (GETPROP Y 'CF) 1)))

(ATOM X) (SSEQUAL X X)
' (PROG (RVAL) (SETO RVAL 0) (MAPC Y (FUNCTION (LAMBDA (DEFINEFPK) (SETO RVAL (MAX RVAL (SSEQUAL DEFINEFPK) X) (RETURN RVAL))))

(SSEQUAL X Y)
(COND ((AND (ATOM X) (ATOM Y)))
(COND ((EQUAL (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1) (OR (GETPROP Y 'CF) 1)))
(T 0))
((ATOM Y) (SSEQUAL Y X))
' (PROG (RVAL) (SETO RVAL 0) (MAPC Y (FUNCTION (LAMBDA (DEFINEFPK) (SETO RVAL (MAX RVAL (SSEQUAL DEFINEFPK) X) (RETURN RVAL))))

(SSEQUAL X Y)
(COND ((AND (ATOM X) (ATOM Y)))
(COND ((INEQUAL (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1) (OR (GETPROP Y 'CF) 1)))
(T 0))
((ATOM Y) (SSEQUAL Y X))
' (PROG (RVAL) (SETO RVAL 0) (MAPC Y (FUNCTION (LAMBDA (DEFINEFPK) (SETO RVAL (MAX RVAL (SSEQUAL DEFINEFPK) X) (RETURN RVAL))))

(S$LESSP)
(LAMBDA (X Y)
(COND ((AND (ATOM X) (ATOM Y)))
(COND ((LESSP (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1) (OR (GETPROP Y 'CF) 1)))
(T 0))
((ATOM X) (S$LESSP X X))
' (PROG (RVAL) (SETO RVAL 0))

(SETQ RVAL 0)
(MAPC Y (FUNCTION (LAMBDA (DEFINEPPP))
(SETQ RVAL (MAX RVAL (SSLEQ DEFINEPPP X))
(RETURN RVAL)))

(SSLEQ)
(LAMBDA (X Y)
(COND
((AND (ATOM X) (ATOM Y))
(COND
(((LEQ (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1)
(OR (GETPROP Y 'CF) 1)))
(T 0)))
((ATOM Y)
(SSLEQ Y X))
(T (PROG (RVAL)
(SETQ RVAL 0)
(MAPC Y (FUNCTION (LAMBDA (DEFINEPPP))
(SETQ RVAL (MAX RVAL (SSLEQ DEFINEPPP X))
(RETURN RVAL))))

(SSGEQ)
(LAMBDA (X Y)
(COND
((AND (ATOM X) (ATOM Y))
(COND
(((GEQ (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1)
(OR (GETPROP Y 'CF) 1)))
(T 0)))
((ATOM Y)
(SSGEQ Y X))
(T (PROG (RVAL)
(SETQ RVAL 0)
(MAPC Y (FUNCTION (LAMBDA (DEFINEPPP))
(SETQ RVAL (MAX RVAL (SSGEQ DEFINEPPP X))
(RETURN RVAL))))

(SSGREATERP)
(LAMBDA (X Y)
(COND
((AND (ATOM X) (ATOM Y)))
(COND
((GREATERP (EVAL X) (EVAL Y))
(MIN (OR (GETPROP X 'CF) 1)
(OR (GETPROP Y 'CF) 1)))
(T 0)))
((ATOM Y)
(SSGREATERP Y X))
(T (PROG (RVAL)
(SETQ RVAL 0)
(MAPC Y (FUNCTION (LAMBDA (DEFINEPPP))
(SETQ RVAL (MAX RVAL (SSGREATERP DEFINEPPP X))
(RETURN RVAL))))

(SSMEMB)
(LAMBDA (X Y)
(COND
((AND (ATOM X) (ATOM Y))
(COND
((MEMB (EVAL X) X)
(DEFUN MEMBER (X Y)  
  (COND  
    ((ATOM X)  
      T)  
    ((MEMBER X Y)  
      T)  
    (T  
      NIL))
)

(DEFUN SMEMBER (X Y)  
  (T  
    (PROG (RVAL)  
      (SETQ RVAL 0)  
      (MAP Q (FUNCTION (LAMBDA (DEFINEPPPP)  
        (SETQ RVAL (MAX RVAL (SMEMBER DEFINEPPPP X)))))  
      (RETURN RVAL)))))

(DEFUN SASSERT (CENIT PARNAME VALUE CF)  
  ("* EDITED: "OCT-OCT-86 12:37")  
  ("* ASSERT A CEXACTY VALUE WITH RESPECT TO THE ANTECEDENT")  
  (SELECTQ (HPSPARNTYPE (EVAL CENIT))  
    PARNAME)  
  (MAKLIST (CONS (CAR PARNAME)  
    (MAP Q (FUNCTION (LAMBDA (DEFINEPPPP)  
        (SETQ RVAL (MAX RVAL (SMEMBER DEFINEPPPP X)))))  
      (RETURN RVAL))))

(DEFUN HPASSERTSABS (CENIT PARNAME VALUE)  
  (PROG (VAR)  
    (SETQ VAR (OR (GETPROP CENIT PARNAME)  
      (GERSYN)))  
    (SET VAR VALUE)  
    (PUTPROP CENIT PARNAME VALUE)))

(DEFUN HPASSERTSIN (CENIT PARNAME VALUE CF)  
  (PROG (VAR)  
    (SETQ VAR (OR (GETPROP CENIT PARNAME)  
      (GERSYN)))  
    (SET VAR VALUE)  
    (PUTPROP CENIT PARNAME VALUE)  
      CF)  
    (PUTPROP CENIT PARNAME VAR)))

(DEFUN HPASSERTSMUL (CENIT PARNAME VALUE CF)  
  (PROG (VAR)  
    (SETQ PLIST (GETPROP CENIT PARNAME))  
    (IF X IN LIST DO  
      (CONQ (EQUAL X)))
    (RETURN RVAL)))))

(DEFUN HPASSERTSMUL (CENIT PARNAME VALUE CF)  
  (PROG (VAR)  
    (SETQ PLIST (GETPROP CENIT PARNAME))  
    (IF X IN LIST DO  
      (CONQ (EQUAL X)))
    (RETURN RVAL)))))

VALUE
(SETO VAR 1)
(SETO VAR VALUE)
(PUTPROP VAR QUOTE CF)
(PUTPROP CNTAT PARNNAME CONS VAR PLIST)

($SADJUST)
(LAMBDA (CNTAT PARNNAME VALUE CF)
 (SELECTQ (HPSADJUST CNTAT PARNNAME))
 (ABSOLUTE)
 (HPSASSERTARS CNTAT PARNNAME VALUE))
 (EQUAL CF)
 (HPSADJUST CNTAT PARNNAME VALUE CF))
 ((MULTIVALE)
 (HPSADJUSTMUL CNTAT PARNNAME VALUE CF))
 (ERROR "PARAMETER NOT ADJUSTABLE")
)

(HPSADJUSTSIN)
(LAMBDA (CNTAT PARNNAME VALUE CF)
 (PROG (VAR))
 (COND
 (SETO VAR (GETPROP CNTAT PARNNAME))
 (COND
 (EQUAL (EVAL VAR)
 (PUTPROP VAR CF (HPSCOMPUTECF (GETPROP VAR CF)
 CF ANTECEDENT))
 (T (SETO VAR VALUE)
 (PUTPROP VAR QUOTE CF (HPSCOMPUTECF O CF ANTECEDENT))
 (PUTPROP CNTAT PARNNAME VAR))
)

(HPSADJUSTMUL)
(LAMBDA (CNTAT PARNNAME VALUE CF)
 (PROG (PLIST VAR))
 (COND
 (SETO PLIST (GETPROP CNTAT PARNNAME))
 (FOR X IN PLIST DO (COND
 (EQUAL (EVAL X)
 (SETO VAR X)
 (REMOVE X PLIST))
 (COND
 (VAR (PUTPROP VAR CF (HPSCOMPUTECF (GETPROP VAR CF)
 CF ANTECEDENT))
 (T (SETO VAR VALUE)
 (PUTPROP VAR CF (HPSCOMPUTECF O CF ANTECEDENT))
 (PUTPROP CNTAT PARNNAME CONS VAR PLIST))

(HPSCOMPUTECF)
(LAMBDA (OLD NEW ANT)
 (PLUS (OR OLD 9)
 (TIMES NEW (TIMES (DIFFERENCE 1 OLD)
 ANT))

(SASIS)
(LAMBDA (CNTAT PARNNAME VALUE CF)
 (SELECTQ (HPSARIYTYPE (EVAL CNTAT)
 PARNNAME))
 (ABSOLUTE)
 (HPSASSERTARS CNTAT PARNNAME VALUE))
 (EQUAL VALUE)
 (HPSASSERTIN CNTAT PARNNAME VALUE CF))
 (MULTIVALE)
 (HPSASSMUL CNTAT PARNNAME VALUE CF))
 (ERROR "PARAMETER NOT ASSERTABLE")
)

(HPSASSSIN)
(LAMBDA (CNTAT PARNNAME VALUE CF)
 ("edited: '24-May-96 12:55"
)
(PROC (VAR
  (ST 'Q )
  (GETPROP CNTXT PARNAME)
  (GENSYM))
  (SET VAR VALUE)
  (PUTPROP VAR (QUOTE CF)
  CF)
  (PUTPROP CNTXT PARNAME VAR))

(HPSASIMUL
  (LAMBDA (GETPROP CNTXT PARNAME VALUE CF)
  (PROC (PLIST VAR)
  (SETQ PLIST (GETPROP CNTXT PARNAME))
  (FOR I IN PLIST DO (COND
  ((EQUAL (VAL I) VALUE)
  (SETQ VAR 1))
  (DREMOVE I PLIST))
  (SETQ VAR (OR VAR (GENSYM)))
  (SET VAR VALUE)
  (PUTPROP VAR 'CF CF)
  (PUTPROP CNTXT PARNAME 'CONS VAR PLIST))
  )
  )
  )

(RPAQO HPSITTHRESHOLD .2)
(RPAQO HPSITNOISYFLAG T)
(DECLARE: OORTVALU;OLOG DO;OVALC;OMP;DO;OCOPY;COMPILEVARS

(ADDTOVAR NLAMA ;OR ;SAND)
(ADDTOVAR NLANL ;SILN)
(ADDTOVAR LAMA ;)

(PUTPROPS HPSFUZZY.LSP COPYRIGHT "CHRISTOPHER RATH" 1986)

STOP
Appendix E

HPSRUN.LSP
(DEFINE)

(SASSERT)

(LAMBDA (CNTXT PARNNAME VALUE CF)
  (SELECTQ (HPSPARNTYPE (EVAL CNTXT) PARNNAME))
    (ABSOLUTE)
    (HPSSASSERTAS CNTXT PARNNAME VALUE)
    (SINGLEVALUE)
    (HPSSASSERTMUL CNTXT PARNNAME VALUE CF)
    (MULTIVALE)
    (HPSSASSERTMUL CNTXT PARNNAME VALUE CF)
    (ERROR "PARAMETER NOT ASSERTABLE")
)

(HPSSASSERTAS)

(LAMBDA (CNTXT PARNNAME VALUE)
  (PROG (VAR)
    (SETQ VAR (OR (GETPROP CNTXT PARNNAME) (GENSYM)))
    (SETQ VAR VALUE)
    (PUTPROP CNTXT PARNNAME VAR))

(HPSSASSERTMUL)

(LAMBDA (CNTXT PARNNAME VALUE)
  (PROG (PLIST VAR)
    (SETQ PLIST (GETPROP CNTXT PARNNAME))
    (FOR X IN PLIST DO (COND ((EQUAL (EVAL X) VALUE)
      (SETQ VAR X) (GRENMOVE X PLIST))
      (SETQ VAR (OR VAR (GENSYM)))
      (SETQ VAR VALUE)
      (PUTPROP VAR (QUOTE CF) (LIST CF) (QUOTE TIMES CF ANTECEDENT))
      (PUTPROP CNTXT PARNNAME (CONS VAR PLIST))))

(SADIUST)

(LAMBDA (CNTXT PARNNAME VALUE)
  (SELECTQ (HPSPARNM (EVAL CNTXT) PARNNAME))
    (ABSOLUTE)
    (HPSSASSERTAS CNTXT PARNNAME VALUE)
    (SINGLEVALUE)
    (HPSSASSERTMUL CNTXT PARNNAME VALUE CF))

("* SASSERT HPSASSERTAS -HPSSASSERTMUL SADIUST HPSASSERTMUL -SADIUST"
  "select CNTXT PARNNAME VALUE CF")

("* HPSSASSERT AS singlevalue AS hypothesis")

("* HPSSASSERTMUL multiplevalue AS hypothesis")

("* HPSCREATE HPSCRECOMPILE REPORT HPSEVALUATE HPSEXPRHPSEXPR HPSEXPRHPSEXPR")

("* HPSCreateHPSCrecompletrHPSEvaluateHPSEXPRHPSEXPR HPSEXPRHPSEXPR")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")

("* ASSERT A CERTAINTY VALUE WITH RESPECT TO THE ANTECEDENT")
(MULTIVALUE)

(HPSADJUSTMUL CNMT PARNAME VALUE CF)
(READ "PARAMETER NOT ADJUSTABLE")

(HPSADJUSTSU)
(LAMQDA (CNMT PARNAME VALUE CF)
(PROG (VAR)
 [COND
 [SETQ VAR (GETPROP CNMT PARNAME)]
 [COND
 [EQUAL (EVAL VAR)
 [PUTPROP VAR 'CF (HPSCOMPUTECF (GETPROP VAR 'CF)
 OF ANTECEDENT])
 [T (SET VAR VALUE)
 [PUTPROP VAR 'CF (HPSCOMPUTECO OF ANTECEDENT)
 [T (SET VAR (GENSYM))
 [SET VAR VALUE]
 [PUTPROP VAR 'CF (HPSCOMPUTECO OF ANTECEDENT)
 [PUTPROP CNMT PARNAME VAR])]

(HPSADJUSTMUL)
(LAMQDA (CNMT PARNAME VALUE CF)
(PROG (PLIST VAR)
 [SETQ PLIST (GETPROP CNMT PARNAME)]
 [FOR X IN PLIST DO (COND
 [EQUAL (EVAL X)
 [SETQ VAR X]
 [REMOVE X PLIST]
 [COND
 [VAR (PUTPROP VAR 'CF (HPSCOMPUTECF (GETPROP VAR 'CF)
 OF ANTECEDENT))
 [T (SETQ VAR (GENSYM))
 [SET VAR VALUE]
 [PUTPROP VAR 'CF (HPSCOMPUTECO OF ANTECEDENT)
 [PUTPROP CNMT PARNAME (CONS VAR PLIST)])]

(HPSCOMPUTECF)
(LAMQDA (OLD NEW ANT)
 [PLUS (OR OLD 0)
 [TIMES NEW (TIMES (DIFFERENCE 1 OLD)
 ANT)])

(SASIS)
(LAMQDA (CNMT PARNAME VALUE CF)
 [SELECTQ (HPSMANTYPE (EVAL CNMT)
 PARNAME)
 [ABSOLUTE]
 [HPSASSERTANS CNMT PARNAME VALUE)]
 [SINGLEVALUE]
 [HPSASSISORS CNMT PARNAME VALUE (CF)]
 [MULTIVALUE]
 [HPSASSUM CNMT PARNAME VALUE CF]
 (ERROR "PARAMETER NOT ASSERTABLE")]

(HPSASSUM)
(LAMQDA (CNMT PARNAME VALUE CF)
(PROG (VAR)
 [SETQ VAR (OR (GETPROP CNMT PARNAME)
 (GENSYM))]
 [SET VAR VALUE]
 [PUTPROP VAR (QUOTE CF)
 CF]
 [PUTPROP CNMT PARNAME VAR])

(HPSASSIMUL)
(LAMQDA (CNMT PARNAME VALUE CF)
(PROG (PLIST VAR)
 [SETQ PLIST (GETPROP CNMT PARNAME)]
 [FOR X IN PLIST DO (COND
 [EQUAL (EVAL X)
 [SETQ VAR X]])]
ABSOLUTE
(ATOM (BINDING (VALUE ANY)))

SINGLEVALUE
(ATOM (BINDING (VALUE ANY)))
(PREADLIST (CT CERTAINTY))

MULTIVALUE
(SINGLEVALUE)

CONTEXT
(ATOM (BINDING (HPSDEF ATOM)))
(CERTAINTY REAL)

DEFINED

HPSCREATE
(defined "16-Jun-66 09:18")

HPSRECOMPILE
(defined "16-Jun-66 09:18")
(DEFUN F (X) (+ X X))

(defun (f x) (+ x x))
Appendix F

Examples
The following pages of code are expanded versions of examples used in the text. The first, ROBOT.LSP, is a robot grocery bagger. It is invoked with the MANAGER RuleSet, given a GROCERY-ORDER as an argument. e.g.

(MANAGER (A GROCERY-ORDER WITH GROCERY-ITEMS = (LIST:
   (A GROCERY-ITEM WITH NAME = 'PRETZELS SIZE = 'MEDIUM
    CONTAINER = 'BOX)
   (A GROCERY-ITEM WITH NAME = 'ICE-CREAM SIZE = 'MEDIUM
    CONTAINER = 'BOX FROZEN? = T) ) )

The second example is a very small part of a mineral identification consultant. It demonstrates the use of contexts within RuleSets.
(GROCERY-ORDER
   (LIST (GROCERY-ITEMS (LISOF GROCERY-ITEM))
         (GROCERY-RASS (LISTOF GROCERY-BAG)))
)

(GROCERY-BAG
   (GROCERY-ITEMS (LISOF GROCERY-ITEM))
   (PROP ((LARGE NUM-LARGE-ITEMS)
           (MEDIUM NUM-MED-ITEMS)
           (SMALL NUM-SMALL-ITEMS)
           (WEIGHT (4 * LARGE + 2 * MEDIUM + SMALL)))
   MSG ((PRINT SPILL-BAG))
)

(GROCERY-ITEM
   (ATOM (PROPLIST (SIZE ATOM)
                   (FROZEN BOOLEAN)
                   (FR-BAGGED BOOLEAN)
                   (CONTAINER ATOM))
         (BAGTAG (NAME ATOM)))
   MSG ((FREEZER-BAG (FR-BAGGED = T)))))
)

(HMSRULESETS

(MANAGER
   (ORD GROCERY-ORDER)
   (RESULT (ORD GROCERY-ORDER)
   TERMINATE (NIL ((NULL GROCERY-ITEMS)
                    (REPORT-CONTENTS ORD)))
   )
   CONTROL FIRST
   RULES ((COMPLETE-ORDER 1 (+ SEE IF ANYTHING IS MISSING)
           IF T THEN (FILL-ORDER-RULES ORD)))
           (BAG-LARGE (+ PUT HEAVY THINGS AT THE BOTTOM OF THE BAGS)
           IF
           THOSE GROCERY-ITEMS WITH SIZE = (QUOTE LARGE)
           THEN
           )
(BAG-LARGE-RULES ORD))
(BAG-MEDIUM (+ ADD MEDIUM ITEMS TO THE BAGS)
  IF (THOSE GROCERY-ITEMS WITH SIZE = (QUOTE MEDIUM))
  THEN ((BAG-MED-RULES ORD))))
(BAG-SMALL (* FINALLY FILL THE BAGS UP WITH SMALL ITEMS)
  IF (THOSE GROCERY-ITEMS WITH SIZE = (QUOTE SMALL))
  THEN ((BAG-SMALL-RULES ORD))))

FILL-ORDER-RULES
(ORD:GROCERY-ORDER)
TERMINATE
 NIL (ONE))
CONTROL
 ALL
RULES ((FOR-1 (* YOU NEED PEPSI WITH SALTY FOOD)
  IF (AND (OR (THE GROCERY-ITEM WITH NAME = (QUOTE POTATO-CHIPS))
              (THE GROCERY-ITEM WITH NAME = (QUOTE PRETZELS)))
         (NOT (THE GROCERY-ITEM WITH NAME = (QUOTE PEPSI))))
  THEN ((GROCERY-ITEMS -- (A GROCERY-ITEM WITH NAME = (QUOTE PEPSI))
         SIZE = (QUOTE LARGE)
         CONTAINER = (QUOTE BOTTLE))
         (PRINT "$ I ADDED SOME PEPSI FOR YOU "$))
         (TERM)))
(FOR-2 (* YOU NEED SYRUP WITH ICE CREAM)
  IF (AND (THE GROCERY-ITEM WITH NAME = (QUOTE ICE CREAM))
         (NOT (THE GROCERY-ITEM WITH NAME = (QUOTE SYRUP))))
  THEN ((GROCERY-ITEMS -- (A GROCERY-ITEM WITH NAME = (QUOTE SYRUP))
         SIZE = (QUOTE SMALL)
         CONTAINER = (QUOTE CAR))
         (PRINT "$ I ADDED SOME SYRUP FOR YOUR ICE CREAM "$)
         (TERM))))

(BAG-SMALL-RULES
(ORD:GROCERY-ORDER)
TERMINATE
 NIL (NOT (THOSE GROCERY-ITEMS WITH SIZE = (QUOTE SMALL))))
CONTROL
FIRST
LOCAL (B:GROCERY-BAG I:GROCERY-ITEM)
RULES ((BSP-1 (* START A NEW BAG IF NECESSARY)
  IF (NOT (THE GROCERY-BAG WITH WEIGHT <= 25))
  THEN ((GROCERY-BAGS -- (A GROCERY-BAG WITH GROCERY-ITEMS =
                       (LIST (QUOTE BOTTOM)))))
(BSP-2 (* PUT THINGS IN BAGS WITHOUT BOTTLES)
  IF (B = (THE GROCERY-BAG WITH (THOSE GROCERY-ITEMS WITH
                                CONTAINER = (QUOTE BOTTLE))
                      * NIL AND WEIGHT <= 25))
  THEN ((B -- (I = (THE ORD:GROCERY-ITEMS WITH SIZE = (QUOTE SMALL)))))
  (ORD:GROCERY-ITEMS -- I))
(BSP-3 (* FILL ANY OTHER AVAILABLE BAGS)
  IF (I = (THE ORD:GROCERY-ITEMS WITH SIZE = (QUOTE SMALL)))
  THEN ((I -- (THE ORD:GROCERY-ITEMS WITH SIZE = (QUOTE SMALL)))))
(BAG-MED-RULES
 (GRID:GROCERY-ORDER)
 TERMINATE ((NIL (NOT (THOSE GROCERY-ITEMS WITH SIZE = (QUOTE MEDIUM))))
 CONTROL :FIRST
 LOCAL (I:GROCERY-ITEM)
 RULES ((BMR-1 (* START A FRESH BAG IF NECESSARY)
 (NOT (THE GROCERY-BAG WITH WEIGHT <= 24))
 THEN ((GROCERY-BAGS -- (A GROCERY-BAG WITH GROCERY-ITEMS =
 (LIST (QUOTE BOTTOM))))
 (BMR-2 (* BAG FROZEN ITEMS)
 IF (NOT (THE GROCERY-ITEM WITH FROZENT = T AND FR-BAGGED = NIL
 AND SIZE = (QUOTE MEDIUM)))
 THEN ((FR-BAGGED = T) ((THE GROCERY-BAG WITH WEIGHT <= 24)
 "-- 1") (GROCERY-ITEMS "-- 1"))
 (BMR-3 (* BAG FROZEN ITEMS THAT ARE ALREADY FR-BAGGED)
 IF (NOT (THE GROCERY-ITEM WITH FROZENT = T AND FR-BAGGED = T
 AND SIZE = (QUOTE MEDIUM)))
 THEN ((FR-BAGGED = T) "-- 1") (GROCERY-ITEMS "-- 1"))
 (BMR-4 (* BAG ANY OTHER MEDIUM ITEMS)
 IF (NOT (THE GROCERY-ITEM WITH SIZE = (QUOTE MEDIUM)))
 THEN ((THE GROCERY-BAG WITH WEIGHT <= 24)
 "-- 1") (GROCERY-ITEMS "-- 1"))
 (BAG-LARGE-RULES
 (GRID:GROCERY-ORDER)
 TERMINATE ((NIL (NOT (THOSE GROCERY-ITEMS WITH SIZE = (QUOTE LARGE))))
 CONTROL :FIRST
 LOCAL (I:GROCERY-ITEM)
 RULES ((BLR-1 (* GET A NEW BAG IF ALL ARE FULL)
 (NOT (THE GROCERY-BAG WITH WEIGHT <= 22))
 THEN ((GROCERY-BAGS -- (A GROCERY-BAG WITH GROCERY-ITEMS =
 (LIST (QUOTE BOTTOM))))
 (BLR-2 (* PUT BOTTLES IN FIRST IF ANY)
 IF (NOT (THE GROCERY-ITEM WITH SIZE = (QUOTE LARGE)
 AND CONTAINER = (QUOTE BOTTLE)))
 THEN ((THE GROCERY-BAG WITH WEIGHT <= 22)
(DEFUN (NUM-LARGE-ITEMS) NIL
  (LET ((RESULT (LENGTH (THOSE GROCERY-ITEMS WITH SIZE (QUOTE LARGE)))))
    (VALUE RESULT))

(DEFUN (NUM-MED-ITEMS) NIL
  (LET ((RESULT (LENGTH (THOSE GROCERY-ITEMS WITH SIZE (QUOTE MEDIUM))))
    (VALUE RESULT))

(DEFUN (NUM-SMALL-ITEMS) NIL
  (LET ((RESULT (LENGTH (THOSE GROCERY-ITEMS WITH SIZE (QUOTE SMALL)))))
    (VALUE RESULT))

(DEFUN (SPILL-BAG) NIL
  (LET ((RESULT (LENGTH (THOSE GROCERY-ITEMS WITH SIZE (QUOTE SMALL)))))
    (VALUE RESULT))

(DEFUN (REPORT-CONTENTS) NIL
  (LET ((RESULT (LENGTH (THOSE GROCERY-ITEMS WITH SIZE (QUOTE SMALL)))))
    (VALUE RESULT))

(DEFUN (STOP) NIL
  (STOP))
(DEFUN INITIALIZE-MINERAL
  (MINERAL) (*edited "18-Jun-96 09:59:43")
  (PRINT (GETPROMPT M (QUOTE COLOR)))
  (SPACES 5)
  (SASIS M (QUOTE COLOR))
  (READ)
  (READ)
  (PRINT "WHAT IS THE ESTIMATED HARDNESS OF *")
  (SPACES 5)
  (SASIS M (QUOTE HARDNESS))
  (READ)
  (READ)))
)

(MINERAL-INIT)

(INIT INITIALIZE-MINERAL)

(PARMS ((NAME MULTIVALUE ATOM TRANS (THE NAME OF *))
  (GROUP MULTIVALUE ATOM TRANS (THE GROUP * BELONGS TO))
  (COLOR SINGLEVALUE ATOM TRANS (THE OVERALL COLOR OF *))
  (HARDNESS SINGLEVALUE REAL TRANS (THE GEOLOGICAL HARDNESS OF *))
  (SCRATCHES ABSOLUTE (LISTOF ATOM)
    TRANS (THE THINGS * SCRATCHES) PROMPT
    (WHAT DOES * SCRATCH *))
  (SCRATCHED-BY ABSOLUTE (LISTOF ATOM)
    TRANS (THE THINGS THAT SCRATCH *)
    PROMPT
    (WHAT SCRATCHES *)
    (WHERE-FOUND ABSOLUTE LOCATION TRANS (THE PLACE * WAS FOUND)
    PROMPT
    (WHERE DID YOU FIND *))
  )
)

(DEFMPS RULES (MPSRULESETS MPSRULESET-1)

   (RULESET-1)

(M:MINERAL)
RESULT (M:MINERAL)
BACKCHAIN (((MINERAL (NAME))))
CONTROL
PRIORITY
TERMINATE
() (PRINT "No rule fired."))
() (SHOW M:COLOR)
() (HEIGHT M:LINE)
RULES (((RULE-1) (* If it is soft and black it is coal)
IF
(SAND M:PShVARZ (SLEQ M:HARDNESS 9)).
THEN
((ASSERT M:NAME "COAL").)
(REPORT M (QUOTE NAME))).
)

RULE-1 (* If it is red it is iron)
IF
(SLEQ M:COLOR "RED")
THEN
((ASSERT M:NAME "IRON").)

RULE-2 (* Differentiate between coal and black diamonds)
IF
(SAND M:PShVARZ (SGREATP M:HARDNESS 9)).
THEN
((ASSERT M:QUOTE NAME)
(QUOTE DIAMOND)
("c").))
)

STOP
Bibliography


Vita

Christopher Rath was born in Aberdeen Maryland on May 13, 1963 to Allan and Ann Rath. He grew up in Mankato Minnesota, where he graduated from Loyola High School in June of 1981. He attended Saint Mary's College in Winona Minnesota, graduating with a Bachelor of Arts degree in May of 1985. He was hired by AT&T Bell Laboratories and was subsequently admitted to the graduate school at The University of Texas at Austin in the department of Computer Science.

This thesis was typed by the author using the XEROX STAR workstation.