INLEN-3: A MULTISTRATEGY SYSTEM FOR LEARNING, INFERENCE AND KNOWLEDGE DISCOVERY IN DATABASES

Overview, Implementation, Experiments, and User's Guide

Kenneth A. Kaufman and Ryszard S. Michalski
Machine Learning and Inference Laboratory
George Mason University
4400 University Drive
Fairfax VA 22030-4444
{kaufman,michalsk}@aic.gmu.edu

Table of Contents

INLEN-3: A MULTISTRATEGY SYSTEM FOR LEARNING, INFERENCE AND KNOWLEDGE DISCOVERY IN DATABASES
Overview, Implementation, Experiments, and User's Guide

1 Introduction

INLEN is a system designed to assist users in data analysis. It uses a multistrategy approach to assisting users by providing different operators which may be called upon to perform various knowledge discovery tasks. By discovering patterns, trends or exceptions that might be difficult for a user to observe in a large database, INLEN may point out information that allows for an advantageous selection of strategies that may otherwise have been missed.

INLEN has a modular conceptual architecture. By adding new learning and discovery programs into the INLEN system, one can make a more versatile and powerful tool. As its name suggests, INLEN-3 is only a single step in the development of a powerful knowledge discovery system. Nonetheless, in itself, INLEN-3 is a complete package, with tools to perform a wide range of knowledge discovery tasks.

INLEN-3 is a direct descendent of the AURORA system (INIS, 1988), an expert system shell that can use machine learning techniques to acquire knowledge. Unlike AURORA, the focus of INLEN is not to simply develop advisory systems for a given task (although it can still be used in that way), but rather to perform various mental manipulations upon a database in order to learn interesting facts and metafacts about the data. The knowledge base can thus be tailored to fit the needs of the user.

Using INLEN, a user can develop one or more *advisory* or *application systems*. These systems contain data and knowledge pertaining to a particular problem. Among the problems INLEN has been applied to are analysis of scientific publications, discovery of economic and demographic patterns in different regions of the world, and detection of patterns in genetic material. Below are three further examples of how INLEN might be used to discover and apply knowledge from a database.

If someone wanted to use INLEN for assistance in the selection of a microcomputer, they could set up a database of the computers on the market and their properties, and use INLEN's learning module to generate a knowledge base consisting of decision rules showing the conditions under which the purchase of a specific module would be recommended. INLEN's advisory module could then be called upon to suggest a machine that would fit the user's needs.

Or if a company maintained a database of on-the-job accidents, they could create an application system based on this domain. Using INLEN's learning and discovery module, they could discover the conditions under which different classes of accidents occurred. By

analyzing this knowledge, the firm might be able to pinpoint areas in which its safety policies needed to be revised.

Another user might be a government analyst interested in the economic or military actions of a foreign power. INLEN could recognize some trends and regularities in the data and make exceptions or changes stand out for the analyst. Hence the analyst could be alerted to potentially important changes in the policies and activities of another country well before they might have been noticed otherwise.

2 Architecture and Implemented Operators

INLEN's approach is to build a synergistic system that allows a human expert and a computer tool to perform the tasks that each party is better suited for. Some patterns are more easily detectable by a machine than by humans; others are obvious to the human eye, but difficult to notice by today's discovery systems. Data and knowledge management functions, searches through large data sets, consistency checking and discovery of certain classes of patterns are relatively easy to perform by a learning and discovery system. On the other hand, defining criteria for judging what is important and what is not, making decisions about what data to process, and evaluating findings from the viewpoint of human utility are easier for a human expert. Working together, such a human-computer data analysis system could exhibit a synergistic effect in extracting useful knowledge from data, and have an increased potential for making discoveries. A machine learning system might also be potentially useful in formulating explicit criteria that experts are using implicitly in evaluating the "interestingness" of a pattern.

INLEN integrates several advanced machine learning capabilities, which until now have existed only as separate experimental programs. Many learning systems are capable of a narow subset of the spectrum of knowledge that can be gained from factual data. By integrating a variety of these tools, a user will have access to a more powerful and versatile system.

The general design of INLEN is shown in Figure 1, and expands upon the initial design presented in [Kaufman, Michalski and Kerschberg, 1991]. As is depicted in Figure 1, the INLEN system consists of a relational database for storing known facts about a domain, and a knowledge base for storing rules, constraints, hierarchies, decision trees, equations accompanied with preconditions, and enabling conditions for performing various actions on the database and/or knowledge base.

The purpose for integrating the above capabilities is to provide a user with a set of advanced tools to search for and extract useful knowledge from a database, to organize that knowledge from different viewpoints, to test this knowledge on a set of facts, and to facilitate its integration within the original knowledge base. These tools in INLEN are known as *knowledge generation operators* (KGOs).

The KGOs are designed to complement one another, and to be capable of performing many types of learning. For example, different operators in INLEN might be employed to learn a set of rules from examples (empirical induction), generalize a descriptor or a set of objects

(constructive deduction, abstraction or induction), hypothesize explanations for events in the data based on rules in the knowledge base (abduction), speculate on unknown attribute values of an object based on known values of similar objects (analogical reasoning), and suggest unknown attribute values by employing rules or formulas in the knowledge base (deduction).

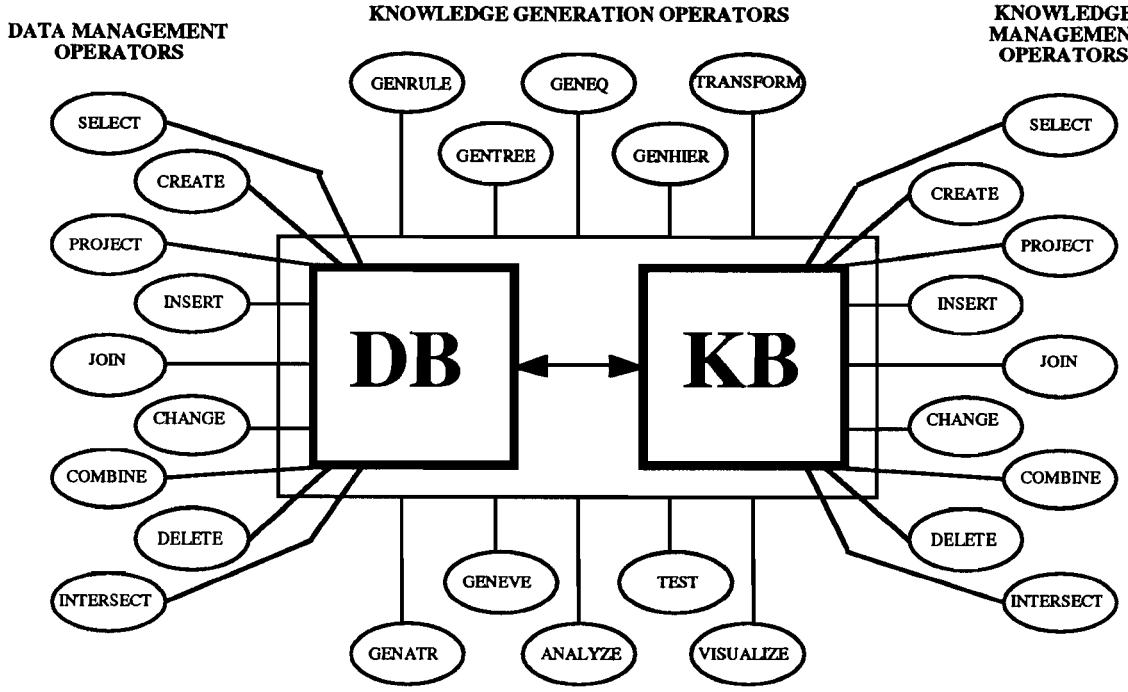


Figure 1. Functional Architecture of INLEN

INLEN-3 is a prototype system, whose focus is on tool integration capabilities and testing of the design. As a result, not all of the knowledge generation operators [Michalski et al, 1992b] have been implemented. Below is a listing of the operators which are part of INLEN-3, organized by category:

GENRULE: Generate Rules

Operators in the GENRULE class take some form of data and/or knowledge as an input, and return a ruleset consisting of facts induced from the input examples. The generated rules consist of a decision part implied by a condition part. The decision part consists of a conjunction of one or more statements or actions, while the condition part consists of a disjunction of conjunctions, each consisting of one or more elementary conditions. Specific GENRULE operators in INLEN-3 are based on the AQ15c program (Wnek et al, 1995). They differ from one another in the type of rules generated (as defined by the user when setting the learning parameters). The user may select from among the following operators:

CHARSET (Characterize Set) determines a description characterizing a class of entities. Input to the operator may be a table representing a group of events and their relevant attributes. It may also be a set of knowledge segments, defined by their own meta-

attributes, that the user wishes to characterize with a rule. CHARSET discovers characteristic rules that describe all of the examples in the input group in as much detail as possible. The output from this operator may include the input set of events in addition to the generated rules that describe the characterization.

DIFFSET (Differentiate Set) takes one set of objects as a primary input, and one or more sets of objects as a controlling input. These sets may be represented by the values of a "decision variable" within an input table. DIFFSET induces simple rules that encapsulate the differences between the primary set and the other classes. The operator may be called upon to treat each of the groups in turn as the primary and discover rules differentiating each group from the others. The output knowledge consists of the ruleset created by the operator, and the input events. Here, the emphasis is on finding the simplest rules possible that will differentiate the objects from the various classes. This operator may be applied in a biased mode, in which the user assigns costs to the different attributes to indicate their desirability in being part of the output rule set. In this case, the system will attempt to learn rules with the least total cost.

DISCSET (Discriminate Set) is similar to DIFFSET, except that the descriptions it generates will be maximally general, while specifying sufficient conditions for distinguishing one class of objects from the other class(es). Unlike DIFFSET, this operator's preference criteria will select a rule which covers more of the input examples instead of one with fewer conditions.

In addition, a user can submit a set of preference criteria and degrees of rigidity to customize the performance of set characterization, differentiation and discrimination.

TRANSFORM: Transform Knowledge

The TRANSFORM operators perform basic inferential transformations on knowledge, hence both the primary inputs and outputs are knowledge segments of the same type, typically decision rules. This class of operators includes two pairs of inverse operators: ABSTRACT and CONCRETIZE, and GENERALIZE and SPECIALIZE, and IMPROVE, an operator that improves knowledge by giving it new examples to learn from. INLEN-3 supports abstraction/concretiztion with respect to structured and numeric attributes (see Section 3) and the IMPROVE operator.

ABSTRACT/CONCRETIZE modifies its input knowledge segment by removing details from or adding details to its description. For example, the known fact, "The Chrysler Dynasty is a mid-sized car that gets 26 miles per gallon", may be abstracted by replacing concepts in it by more general concepts, entailed by the original one. The resulting knowledge segment might contain the fact, "The Chrysler Dynasty is an efficient automobile for its class." Conversely, CONCRETIZE can take a fact such as "The Lincoln Town Car is a luxury automobile" as input, and create an output statement such as "The Lincoln Town Car is expensive, and contains many comforts and conveniences for the driver and passengers." In such an example, the user would have defined the abstraction hierarchy (e.g., 26 MPG is a subvalue of efficient) and asked the program to climb the hierarchy with the discovered knowledge.

The input to IMPROVE is one or more knowledge segments and a new set of examples. From the examples, any exceptions to the input knowledge are detected, and the KSs are modified accordingly by a learning program. The output from this operator consists of the revised rules. The methodology for IMPROVE, as with other rule generation operators in INLEN-3, is based on the AQ15c program.

GENATR: Generate Attributes

The GENATR operators map relational tables to relational tables whose rows are the same but whose columns have been changed, either by the addition of new attributes or by the removal of old ones. In INLEN-3 only attribute removal (selection) is implemented.

SELATR (Select Attribute) determines the attributes in a relational table that are most relevant for differentiating between various classes of objects, and produces a reduced table that retains only those variables chosen by the operator. By keeping only the most relevant attributes in the object (example) descriptions, one can significantly reduce the computation time required by the other learning operators at only a minimal cost to performance. This operator may be run in two modes. The first, based on the PROMISE program (Baim, 1982), chooses the attributes with the best overall capacity to discriminate among classes. The second, which may produce more succinct knowledge in a rule-based representation, favors attributes which have one or more values with high discrimination capability.

To illustrate the difference, consider a knowledge base to distinguish between upper-case letters of the English alphabet. Two attributes that might be considered are whether the letter consists of only straight lines, or whether it has a tail. The former attribute is very good for a decision-tree knowledge representation, it immediately divides the set of letters under consideration into two roughly equal-sized groups. However, in a rule-based representation, this attribute alone is not sufficient to recognize a letter; other conditions must be added.

On the other hand, the has-tail attribute works very well in a rule-based representation. It leads to the simple rule *Letter is Q if has-tail = true*. However, this is not a useful feature on a decision-tree. More often than not it will simply reduce the set of candidates from 26 to 25.

GENEVE: Generate Events

The GENEVE class covers a wide variety of operators that generate a new set of tuples, either from an existing relational table, or from the entire event space to which a table's tuples belong. The output events are selected according to some criterion of desirability, such as typicality, extremity, being contained in two or more classes, etc.

INLEN-3 supports one GENEVE operator – PREDVAL (Predict Value). This operator predicts the value for an attribute based on the values of other attributes, provided by the user. It operates in an expert system shell as a major component of INLEN's Advisory Module (Section 8).

TEST: Test Knowledge

The TEST operator determines the performance of a ruleset on a set of examples by testing the input knowledge segments for consistency and completeness with regard to the input examples (specified in a relational table). Consistency implies that no event in the example set is covered by two different rules. Completeness refers to the condition that every example is covered by the conditions applying to at least one rule. Input consists of a set of examples to be tested and a set of knowledge segments that are to be tested against the examples. The output knowledge consists of several relational tables containing TEST's analysis, including weights that indicate the quality of the knowledge segments. The primary output table is in the form of a *confusion matrix*, i.e. a matrix whose $(i,j)^{th}$ element shows how well the *i*th example matched the rules for class *j*. TEST uses the ATEST methodology (Reinke, 1984) for analyzing consistency and completeness in rules, and generating confusion matrices.

3 The Data

The attributes in INLEN's databases can take on five types: linear, integer, numeric, nominal, or structured. In this section we discuss each of them.

3.1 Linear Type

Linear attributes are attributes whose values can be ordered completely. Their values are names of concepts, and as such, while one can say that one value is greater than another, there is no way of indicating how much greater the value is. Examples of linear attributes and their values might be *size* (small, medium, large, extra large), *temperature* (cold, cool, warm, hot, very hot), and *class* (kindergarten, first, second, ...)

INLEN's knowledge representation can express concepts such as "size is small or large," "size is not large," "temperature is cool to hot," "temperature ≤ mild," or "class is kindergarten or fifth to seventh."

3.2 Integer Type

Integer attributes are a special class of linear attributes whose values consist of the integers 0-50 inclusive. Examples of integer attributes might be *#USstatesVisited, YearsService* , and *AgeWhenMarried* .

Integer attributes are considered to be part of different intervals, so that when concepts involving integer attributes are learned, the values are stretched into intervals. For example, two people who will receive an award have 15 and 23 years service, while a person who will not receive the award has only 7 years service. INLEN may apply a gap-bisecting heuristic and generate the rule: *gets_award if YearsService ≥11.*

3.2 Numeric Type

Numeric attributes are a special class of linear attributes whose values consist of nonnegative real numbers in a range bordered by two integers. Examples of numeric

attributes might be *GPA* (0-4), *body_temp_fahrenheit* (90-108), and *percent_land_forest* (0-100).

Like integer attributes, numerics are considered to be part of different intervals; however, unlike integers, they have their intervals set dynamically when the problem is defined. The ranges are set so as to maximize the likelihood of useful discoveries. For example, in an insurance database, if the goal is to learn about accident-prone drivers, the optimum intervals for the attribute age might be under 23, 23-32, 33-51, 52-68, and above 68. If the goal is to learn about customers who might buy a particular offered service, better ranges might be under 38, 38-47, 48-63, 64-75, and over 75. INLEN automatically discretizes the attribute once the problem is presented.

3.4 Nominal Type

Nominal attributes are the simplest of the data types. The values for nominal attributes are simply different names, and there exist no inclusion or ordering relationships among the values; the only meaningful relation among them is (in)equality. Examples of nominal attributes and their values might be *color* (red, yellow, blue, green, ...), *Citizenship* (Australia, Canada, Germany, Japan, UK, US, ...), *animal* (dog, cat, pig, lion, ...), and *married* (yes, no).

3.5 Structured Type

Like nominal attributes, structured attributes have values that can not be compared by an ordering relation. However, the values of structured attributes can be organized into hierarchies based on inclusion relations. Most nominal attributes with more than a few values can be structured; for example, Figures 2 and 3 show possible structurings of the attributes of nominal attributes *color* and *animal* presented in Section 3.4.

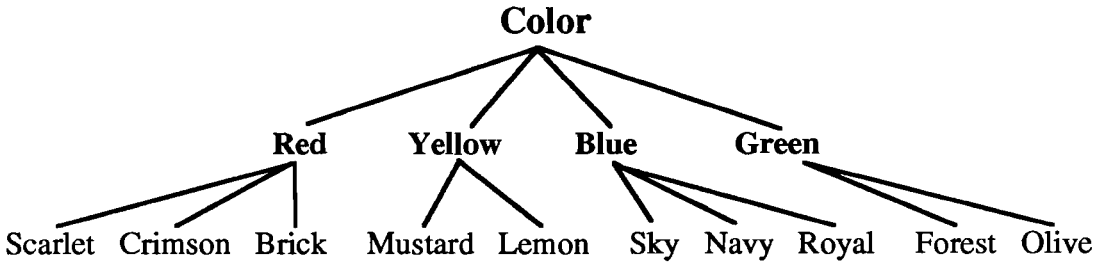


Figure 2. Organization of the structured attribute *color*

The structuring in Figure 2 is straightforword. Each top-level concept is shown to encompass two or three low-level concepts, for example, Red encompasses Crimson, among others. The structuring in Figure 3 is somewhat more complex, and contains four levels of structuring. Several facts should be noted about this structure. First, not all paths down the structure need to go to the same level. The path to Pig, for example, ends at the second level, while the path to Snoopy goes down four levels.

Also notice that items on the same level of the structure do not need to share the same level of generality. Presumably, the fourth level nodes below Beagle represent different

individual dogs, while the ones below Terrier represent different breeds of terrier (we could include specific terriers' names on a fifth level if we so desired.) The level number is not important for INLEN's reasoning, but rather the ancestor/descendant and the common ancestor relationships that are most critical.

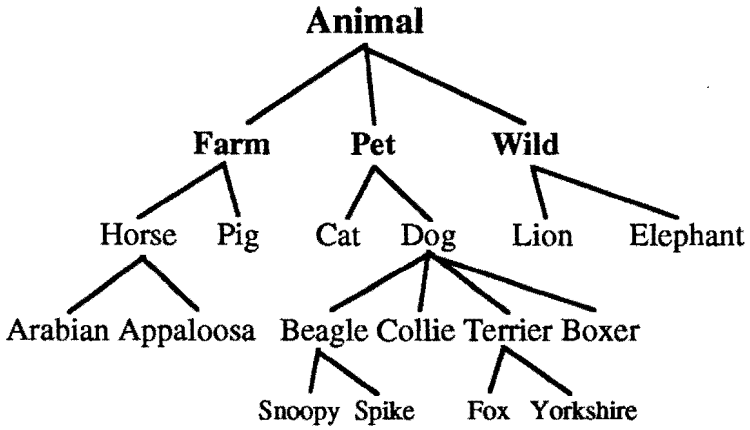


Figure 3. Organization of the structured attribute *animal*

The method of defining simple structured attributes is described along with the use of INLEN's Variables table in Section 5.3. More advanced uses of structured attributes, including the definition of *anchor nodes* to be the focus of generalization/specialization and the use of multiple views of a single structure are described in Appendix A.

4 Getting Started With INLEN

INLEN-3 runs on a DOS platform. All that is required to install the system is to create a directory to serve as INLEN's home and copy the contents of the system disks into that directory using the RESTORE command.

To run INLEN, go to this directory and type "inlen" at the prompt. This takes you directly to the main menu. Alternatively, one can type "inlen demo" to see a series of introductory screens that briefly describe the philosophy behind INLEN, a few concept definitions and the system's major modules.

The main menu (Figure 4) includes the following options:

Create or Modify an Advisory System – where the user may create, edit or delete the various components of an application system.

Invoke Learning and Discovery Module –contains the operators for learning, optimizing, improving and testing the rules in an application system's knowledge base.

Apply an Advisory System to a Problem – uses a knowledge base in conjunction with INLEN's expert system shell for decision making.

Browse Through an Advisory System – allows the user to look at or print some or all of the components of an application system.

Learn How to Use INLEN – an on-line tutorial for using INLEN's major modules.

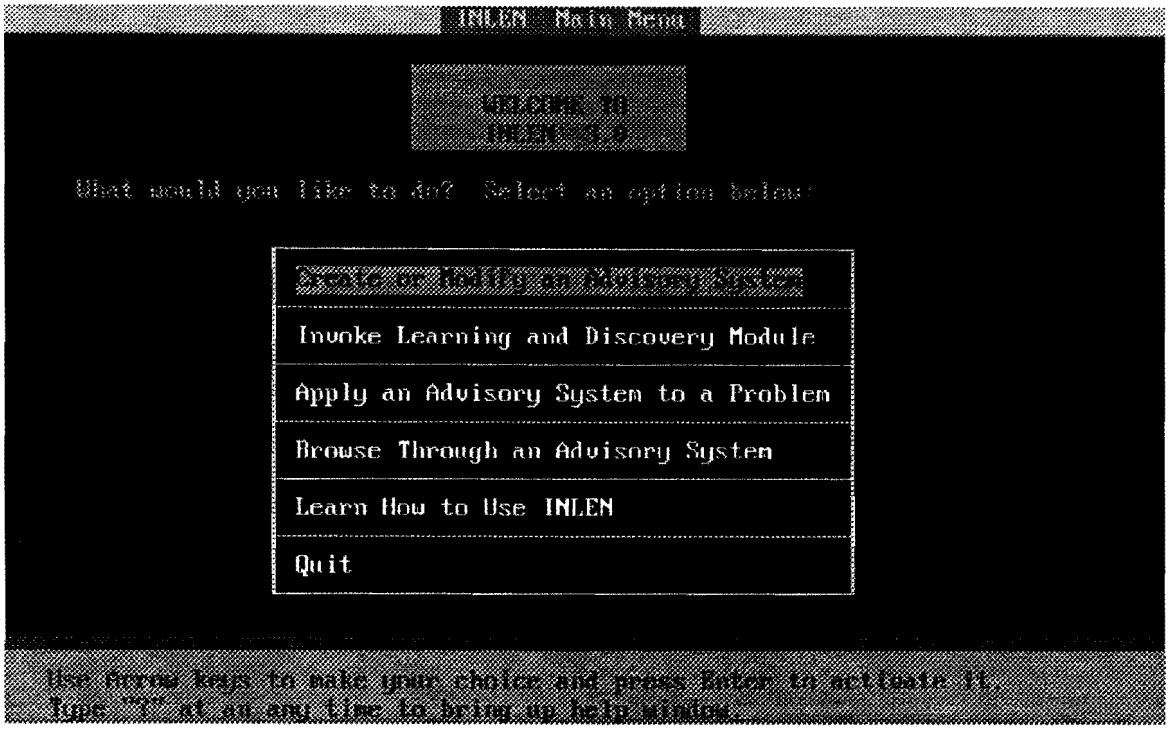


Figure 4. INLEN Main Menu

The user may scroll through the menu using the up and down arrow keys, putting the highlighted choice into effect with the **Return** key. Alternatively, one may select an option by simply typing the first letter of the desired command.

The features included in the Create, Modify and Browse options are discussed in Section 5. The Learning and Discovery Module is described in Section 6. The Advisory System inference engine is described in Section 8.

5 Developing an Application

This section describes the Create/Modify and Browse an Advisory System modules. These are grouped together due to their similar "look and feel"; indeed, the major difference is that when in Browse mode, a user may inspect but not alter the structures that make up an application system.

5.1 The "Develop System" Menu

When either the Create/Modify or Browse option is selected from the main menu, a new menu appears on the screen consisting of three columns (Figure 5). From this menu, the user may customize a command specifying the action to be taken and the application system that the action will affect.

The leftmost column contains a list of application systems in INLEN's memory. The first entry is always <New System>, representing the possibility that the user may wish to create a brand new system to add to the list. (It should be noted that the only legal

operation on <New System> is creation of its System Description. After that, the created system takes its proper place in the list below.) Subsequent items in the list are the names of all of the application systems that have been created, sorted by application domain. For example, the engineering systems would be listed together, separate from the geographic systems. INLEN can currently maintain knowledge of 30 different application systems; in order to add a 31st, a user would have to first remove an existing system description (see Section 5.2).

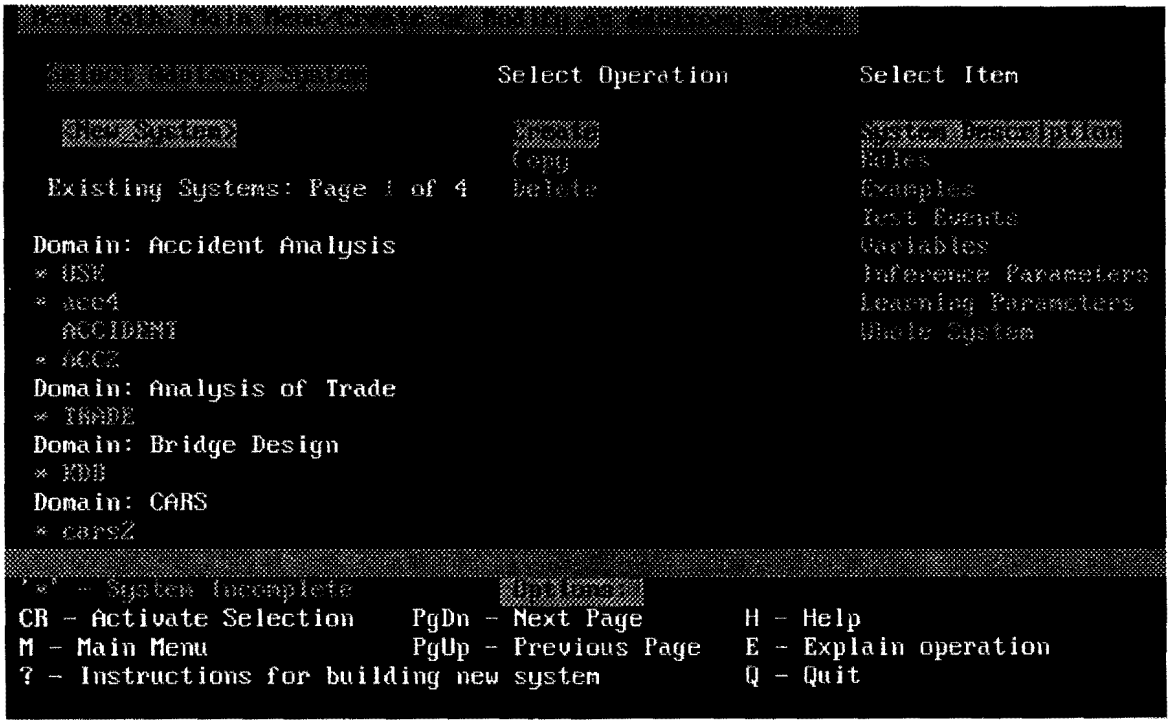


Figure 5. Developing System Menu

An asterisk next to the system name indicates that the system is not yet complete, i.e., it can not be used in an advisory session until further components are built.

The rightmost column lists the components of an application system. These are:

- **System Description**: General information on the application system itself (Section 5.2).
- **Rules**: The rule base associated with the application system (Sections 5.5, 6 and 7).
- **Examples**: The actual database from which learning will take place (Section 5.4).
- **Test Events**: A set of examples from which knowledge may be tested (Section 5.4).
- **Variables**: The schema for the application's database, including descriptions of the attributes and their legal values (Section 5.3).
- **Inference Parameters**: Parameters that guide the expert system's inference engine in a domain's advisory sessions (Section 8.2).
- **Learning Parameters**: Parameters that guide the learning system's generation of rules for the domain (Section 6.1.1).

- **Whole System:** For operations on the application system as a whole (Section 5.6).

The middle column contains the list of permissible operations on the part of the application system specified in the other two columns. When in the Create/Modify module, the options are:

- **Create** or **Access** depending on whether the specific component exists yet. Either way, screens will be brought up allowing the user to develop that component of the selected system.
- **Copy**, in which a component of the selected application system may be copied to another system.
- **Delete**, in which files associated with a component of the selected application may be removed. The system will always prompt the user with an "Are you sure?" message before such a deletion occurs.

When in Browse mode the options are:

- **View**, in which screens appear that are typically similar to the ones presented by the Create and Access commands. In View mode, however, the constructive and destructive capabilities are not available.
- **Print**, in which the contents of the selected component are output directly to the printer for off-line viewing. The output is also written to the file rev.dat, which may then be copied elsewhere for an on-line record.

One of the three columns is highlighted; the user may choose the active column by using the right and left arrow keys. Similarly, the user may scroll to an entry within the active column by using the up and down arrow keys. If there are more application systems than fit in the leftmost column, the user may use the **PgUp** and **PgDn** keys to see other pages of the system list. When the desired items have been selected in all three columns, the user can hit **Return** to effect the chosen operation. Alternatively, the user may hit **M** to return to the main menu or **Q** to quit.

The Copy, Delete and Print commands are typically self-explanatory and similarly used regardless of the application system component involved. Hence the following sections will focus upon the options available in Create, Access and View mode.

5.2 The System Description

The System Description is the information that allows INLEN to recognize an application system as one to be made available to users. It consists of four items, two of which are required and two of which are optional. These are:

- **Domain**: The name of the general domain area for the application. The purpose of the domain is to organize the list of systems when they are presented to the user. When entering a domain name, the user may type in the name or type **?** to select from a menu of already-defined domains.
- **System Name**: The name of the application system (up to 8 characters). Files associated with the system will have this name, followed by an extension indicating the purpose of the file.
- **Short Description** (optional): A brief description (up to 50 characters) that explains the purpose of the application system. These descriptions provide assistance when a user in the advisory module needs to select a system to work with.

- **File Containing Long Description** (optional): The name of a file that contains an extended description of an application system. Users selecting this application in the advisory module will be shown the text in this file in order to ensure that this is indeed the application they want.

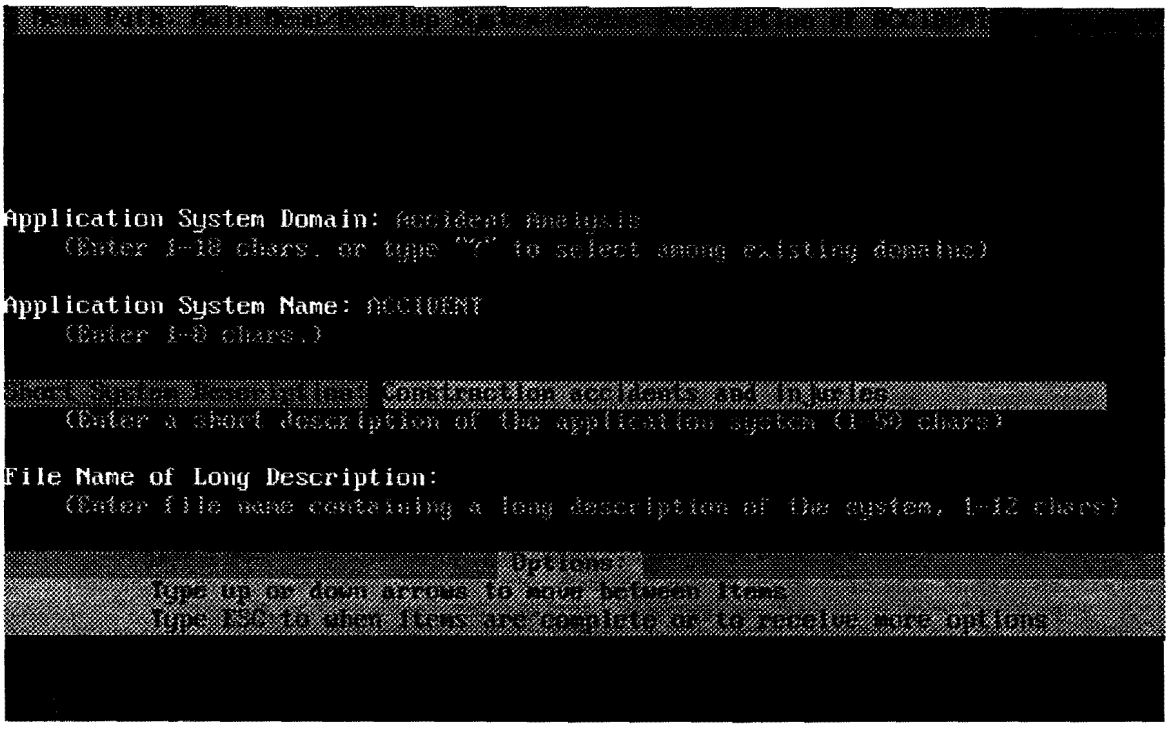


Figure 6. Entering a System Description

In Create or Access mode, the user can scroll between items using the up and down arrow keys (Figure 6). When all four entries are satisfactory, one may hit **Esc** to go to a menu of options for leaving the System Description screen.

In View mode, the entire System Description is shown to the user at once, along with a list of commands for leaving the System Description screen.

5.3 The Variables Table

The Variables table contains information about the attributes in the database for a given application. This information includes attribute type, attribute cost, legal values and annotations. Each of these is described below.

This information appears on the screen in the form of a table. Each column represents an individual variable. The leftmost column by default represents the *decision variable*. This is the attribute whose value we are most interested in determining based on other attributes, both through rule learning and through rule application in the advisory module. For example, if INLEN is being used to select a microcomputer, the decision variable would consist of the various models that were available, while the other variables would represent features such as price, memory, graphics, etc. By exchanging columns, a user may select a new variable to be the decision variable for future sessions.

The top row of each column contains the name of the variable, the next row shows either its type (nominal, linear, structured, integer, or real-valued numeric range) or its cost (the information shown can be toggled using the **F5** key). Subsequent rows contain the legal values of the variable. Values and variable names can be up to 12 characters long, and may not include spaces, hyphens, periods, commas, colons or pound signs.

As explained in Section 3, if the variable type is *nominal*, the order of the values is insignificant. If the type is *integer*, it is unnecessary to enter values; the integer type automatically takes on the integer values from 0 to 50. If the type is *numeric*, the range is defined in the type row of the screen by entering bounding integers separated by a hyphen (example: 2-16). As with integers, no specific entry of legal values is needed; the range as defined will automatically be divided into discrete intervals based on the classification of the training examples, and values will be placed in the proper interval. As long as a training example set is present, the intervals will be reconfigured whenever the variables or training examples table is saved. By remaining in the variables editor, the user can then see the intervals that were selected for the different ranges.

The other variable types require more care in the ordering of values. A *linear* variable implies an ordering among its values, in which each value is less or smaller than the ones listed below it and greater or larger than the ones listed above it. For example, a linear attribute named size might have associated with it the values small, medium, large and extra_large *in that order*. *Structured* attributes have hierarchical orderings among their values, hence some values may represent subclasses of other ones. The user can specify the level in the hierarchy by using the <Tab> key. A top level value is entered normally, a second level value is entered with a single tab, a third level value with two tabs, and so forth. Note that the tabs should be added *after* the name has been typed in.. A value may be moved back to a higher level by typing <SHIFT-Tab>. INLEN supports structures as deep as five levels.

Structured values must be listed in such an order that a value's parent is the next value above it that is on the next higher level. For instance, an attribute describing geometric shapes might be entered as follows:

```
polygon
   triangle
   four-sided
        square
        rhombus
   hexagon
curved
   circle
   ellipse
```

Here square and rhombus would be subclasses of four-sided, which along with triangle and hexagon would be subclasses of polygon. Curved would have subvalues circle and ellipse. Inheritance would work on multiple levels, thus a square would also be considered a polygon.

Structured data representation is the most powerful representation form available in INLEN. A user can represent intricate relationships among the available values. Facilities are also provided to allow users to get around the 5-level depth limitation. Detailed instructions on the use of structured domains are given in Appendix A.

The *cost* of a variable is an indicator of how expensive or difficult it is to ascertain a value for that variable. The purpose of providing attribute costs is to discourage decision rules whose applicability would be costly to determine. For example in a medical domain, if a diagnosis for certain diseases might be made on a patient via either a blood test or exploratory surgery, using the result of the blood test would be preferable because of its ease to perform and the lower physical and financial demand on the patient. The blood test indicator would therefore be assigned a far lower cost in INLEN's database than results based on the surgery. Given a set of learning parameters to minimize cost, INLEN would do all that it could to learn rules that relied on the blood test, only bringing up the need for the surgery when no other course of action would provide a satisfactory result.

The user may assign variables costs from 0 to 100. The default value is 1.

Figure 7 shows an example of a Variables table.

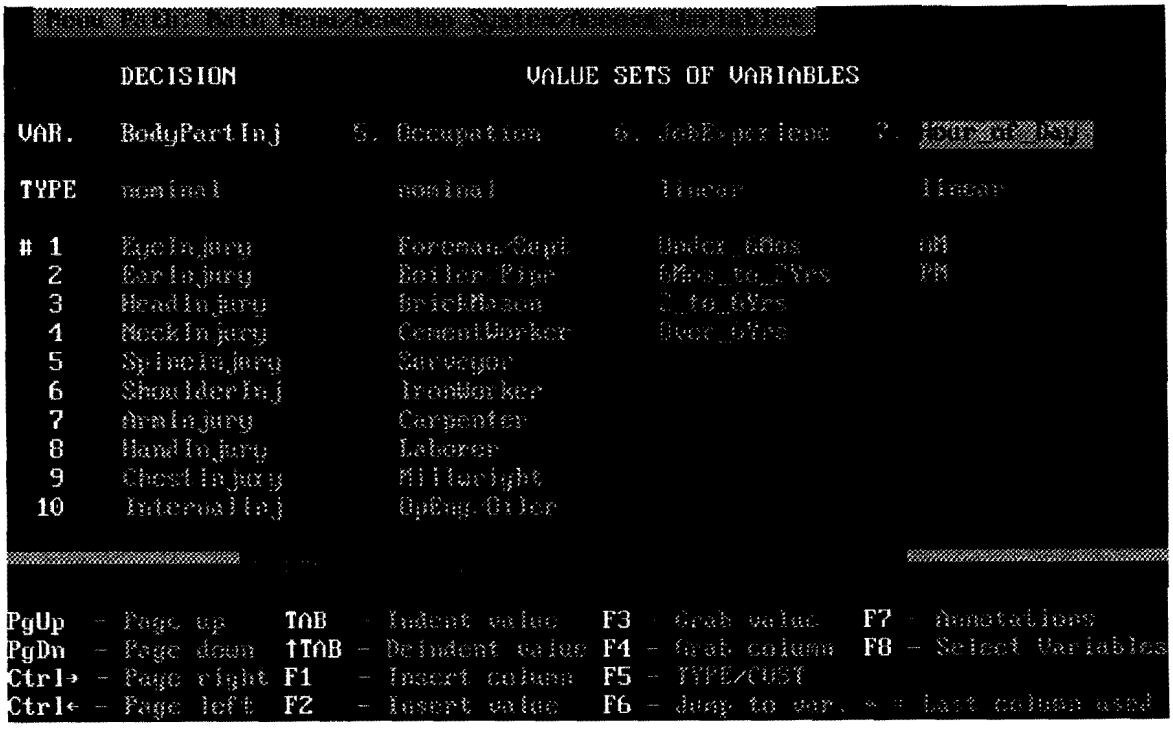


Figure 7. The Variables Table

5.3.1 Using the Variables table

The user may scroll through the Variables table using the arrow keys, the PgUp and PgDn keys, and the Ctrl- right and left arrow keys, the latter to move right or left one full screen at a time. INLEN's limits are currently 95 variables (plus the decision variable), each of which may have as many as 200 different values.

After having scrolled to a desired cell in the table, the user may type in or alter the value in that cell, locking in the change by typing Return (which behaves as a down arrow key), the space bar (which behaves as a right arrow key), one of the movement keys, or one of the Variables table function keys. These include:

F1 (Insert Column) – Insert a blank column at the current location. Shift the column that was there and all other columns to its right one column further to the right in order to accommodate it. The training and testing example tables are automatically modified to maintain database consistency.

F2 (Insert Value) – Insert a blank cell at the current location. Move the value that was there and all below it one row lower in order to accommodate it. Note that this is not permitted when the current cell contains a variable name, type or cost.

F3 (Grab Value) – Call upon a menu of operations that can be performed upon a value. The user may move a value to a new row (overwriting whatever was there), switch a value with another one in the same column, or delete a value, with all lower values in the column moving up a row. Note that this is not permitted when the current cell contains a variable name, type or cost.

F4 (Grab Column) – Call upon a menu of operations that can be performed upon an entire column in the table. The user may insert a duplicate copy of the column at the current location, moving the original column and all of the columns to its right one column to the right; copy a column to another location, overwriting whatever was there previously; move a column to another location, again overwriting whatever was there; switch two columns in the table or delete a column, as all columns that were to the right of it move one column leftward. Note that the decision column may be involved in any of these operations; it is regarded as column 0. The training and testing example tables are automatically modified to maintain database consistency, i.e., if a column is deleted, duplicated, moved, etc., the same operation will be performed on the corresponding column in the example tables.

F5 (Type/Cost) – If the variable types are being displayed on the screen, show the costs instead. If the costs are being displayed, show the types instead.

F6 (Jump to Variable) – Immediately moves the cursor to the top of the current column.

F7 (Annotations) – The user may add or alter the annotations for the current cell, which vary depending on the cell. These annotations are used by the advisory module to allow for a better interaction with the user.

If the current cell is a variable name, INLEN will ask for a question associated with the variable (for use when the inference engine asks for its value. If there is no question, it will just use the form "<variable name> is:". INLEN will also ask for an "importance value" associated with the variable. The importance value may be from 1 to 10, and affects the ordering of the questioning from the inference engine. See Section 8 for more details.

If the current cell is a decision (i.e., a value for the decision variable, INLEN will ask for the name of a file to be displayed should that decision be selected (the file presumably will contain further information about the nature of the decision) and a program to be executed

should the decision be selected. If no file or program are specified, no actions will be taken.

If the current cell is the value of a non-decision variable, the user may type in the name of a help file associated with that value. An advisory system user may then inquire about the meaning of a particular value and see the help file displayed.

There are no annotations associated with variable types or costs.

F8 (Select Variables) – This function calls upon a program to select and retain the "best" variables using a modified version of the PROMISE method (Baim, 1982). Given that learning from a large database may be a computationally expensive process, this operator allows INLEN to reduce the number of attributes in its view of the data by only retaining those which show the most promise of working toward the user's goals.

The attribute selection operator estimates the degree to which an individual variable successfully discriminates among the decision classes given the examples in the database, and assigns each value a score from 0 to 100 with 0 indicating no discriminatory capacity (i.e., total independence from the decision variable) and 100 showing perfect discrimination. A menu allows the user to select the parameters for this operator (Figure 8); one may choose to select the attributes that have individual values that discriminate especially well, or select the ones that show the best overall discrimination capability. The user may also opt to retain a specific number of attributes, or to keep or discard attributes based on a threshold promise score. The default threshold score is 50, and the default number of attributes saved is half the original number of attributes.

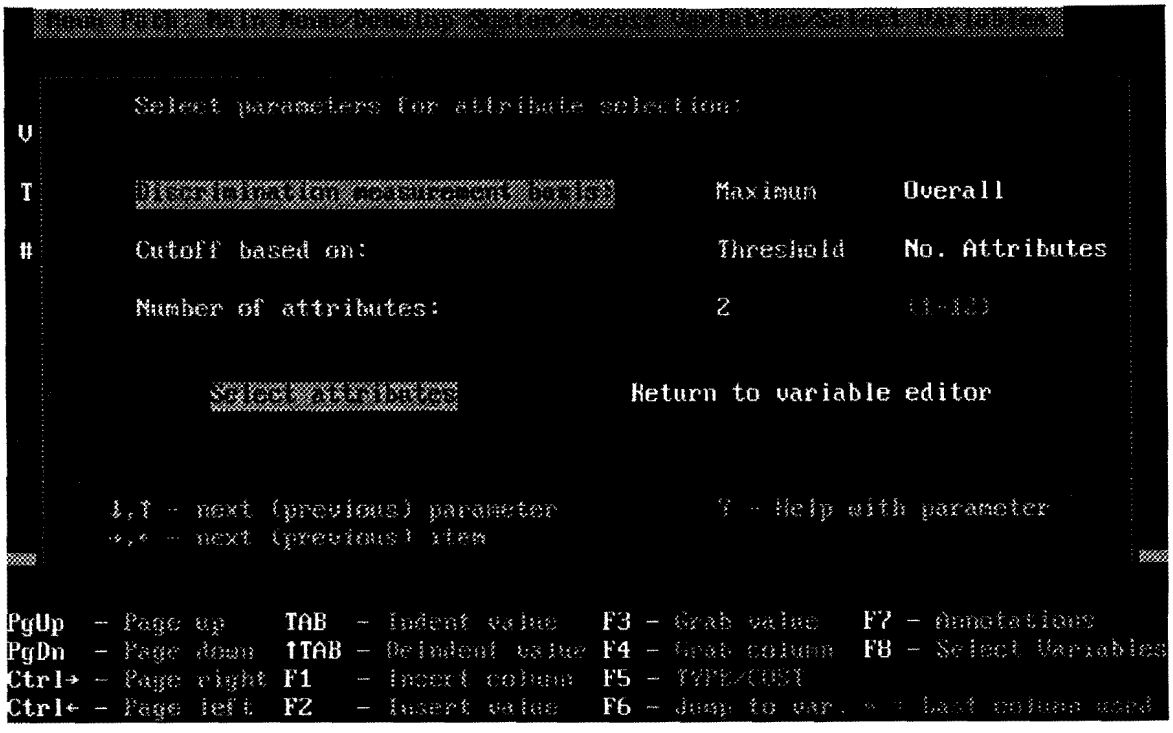


Figure 8. Setting the parameters for attribute selection

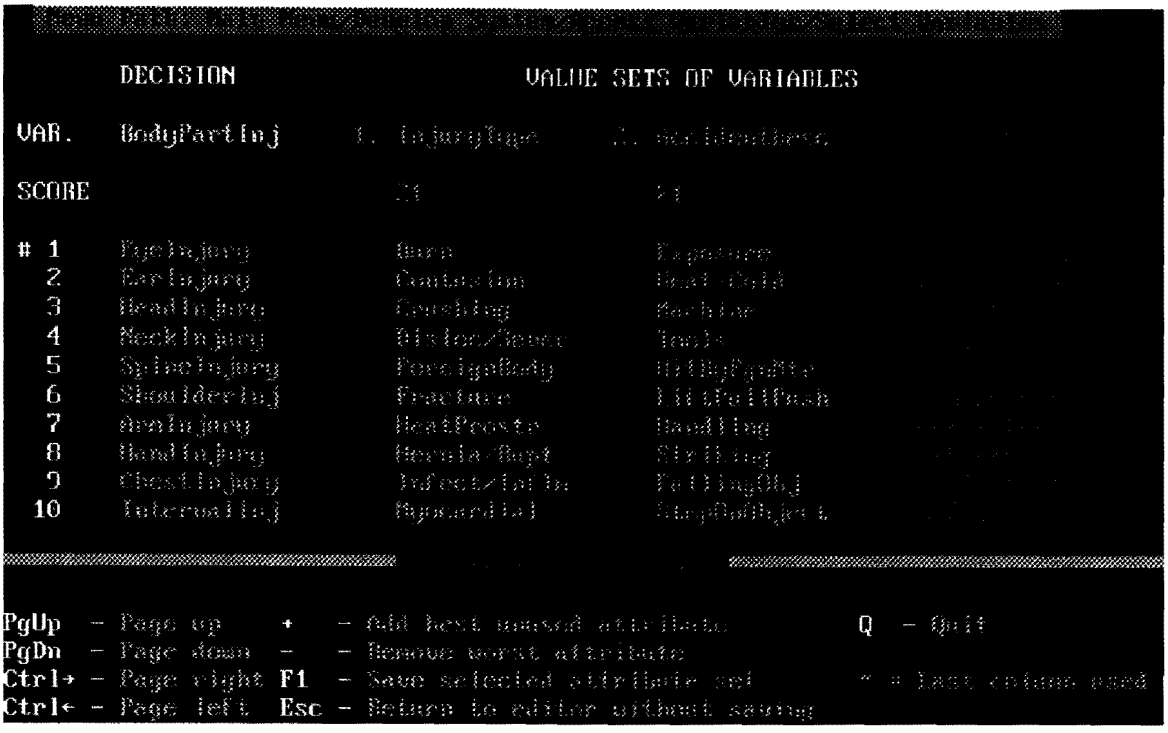


Figure 9. Displaying the most promising attributes

INLEN displays its recommended attributes based upon these parameters (Figure 9), sorting the columns in order of promise score from best to worst, with columns slated for retention shown in white and those slated for deletion appearing in red. The user may either accept the revised database (with the F1 key), modify the proposed database by adding or deleting from the list of suggested features (using the + or - keys respectively), or decide not to reduce the number of active variables after all (via the Esc key). If the user does choose to reduce the variable set, the smaller database may be saved to a different application system, thereby maintaining the full-sized database in its original location. Note that complex structures (i.e, structured variables with substructures, Appendix A) are treated together as single attributes.

Esc (Escape) – A menu offering options for transfer of control (e.g., Save Changes, Return to Parent Menu, Quit) appears, allowing the user to leave the Variables editor. If changes have not been saved, the user will be prompted to ensure that any desired save is performed before exiting.

5.4 The Training and Testing Examples Tables

The Examples tables are where the actual data is stored in an INLEN database. The Examples table contains data from which knowledge will be generated, while the Testing Event table contains data from which knowledge will be tested. Nonetheless, the screens for accessing these tables are identical. The Examples tables are similar in format to the Variables table, with each column representing an attribute in the database. Each row after the header rows (variable name and type/cost) represents a record from the database, an example from which INLEN may learn/test. Blank cells are permitted; these represent

information that is not available. INLEN currently allows up to 650 examples to be used at one time.

One major difference between the Examples and Variables tables is that the Examples tables offer a mode in which the user can view or enter record keys. These will not be involved directly in knowledge discovery, but they can provide the user with links between knowledge and the examples on which that knowledge is based. Keys will be shown on the far left of the screen; as a result, the decision variable is moved over, and only two other variable columns are shown. If the user does not provide a key, INLEN will generate one based on the example number. An Examples table screen in which keys are displayed is shown in Figure 9.

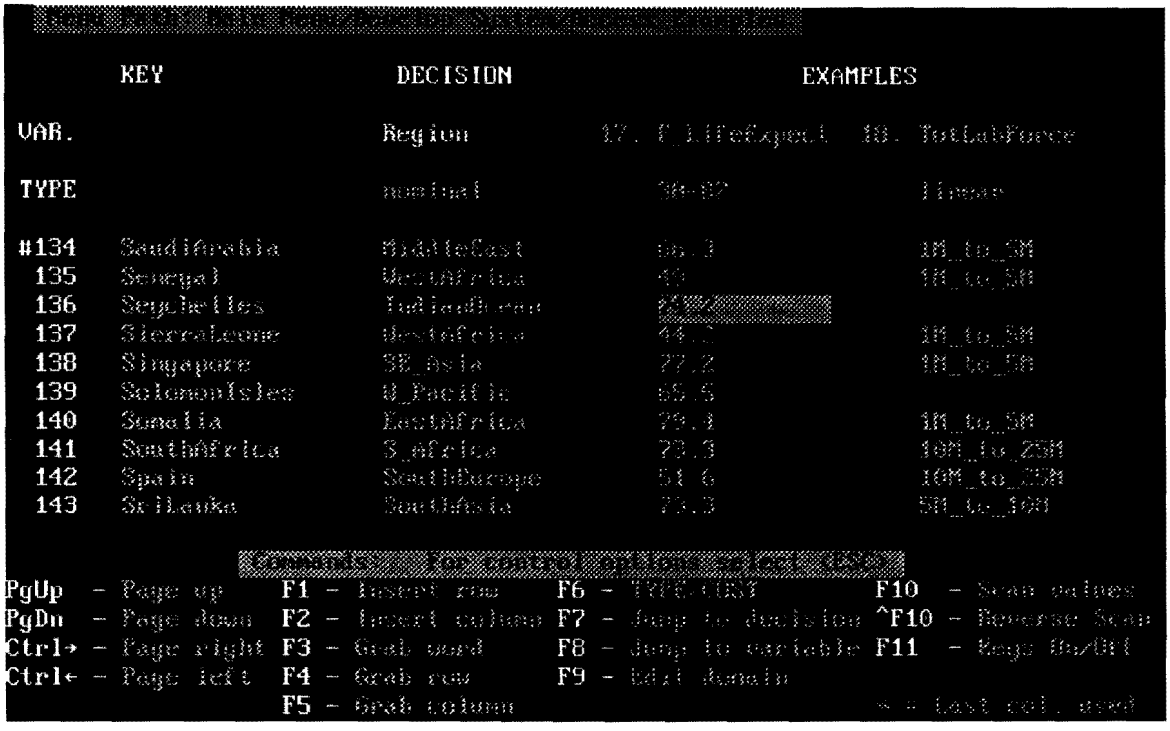


Figure 9. Editing examples

The user may scroll through the Examples table using the arrow keys, the PgUp and PgDn keys, and the Ctrl- right and left arrow keys, the latter to move right or left one full screen at a time. Because substructures are relevant only if their parent node was selected in the parent structure (see Appendix A for an explanation of substructures), cells in substructure columns will be skipped over unless the parent value was selected.

After having scrolled to a desired cell in the table, the user may type in or alter the value in that cell. The value entered must be a legal value for that variable. If it is not, an error message will appear, and the user will have to either correct the error or add the value to the legal list of values for the variable. The user locks in any change by typing Return (which behaves as a down arrow key), the space bar (which behaves as a right arrow key), one of the movement keys, or one of the Examples table function keys. These include:

F1 (Insert Row) – Insert a blank row at the current location. Move the row that was there and all below it one row lower in order to accommodate it. Note that this is not permitted when the current cell contains a variable name, type or cost.

F2 (Insert Column) – Insert a blank column at the current location. Move the column that was there and all other columns to its right one column further to the right in order to accommodate it. The variable table and testing/training example table are automatically modified to maintain database consistency.

F3 (Grab Word) – Call upon a menu of operations that can be performed upon a value. The user may copy a value to the row below (overwriting whatever was there), copy the value to all examples of that class (as determined by the value in the leftmost column), or delete a value, making the cell blank. Note that this is not permitted when the current cell contains a variable name, type or cost.

F4 (Grab Row) – Call upon a menu of operations that can be performed upon an entire example (row) in the table. The user may insert a duplicate copy of the row at the current location, moving the original row and all below it one column down; copy a row to another location, overwriting whatever was there previously; move a row to another location, again overwriting whatever was there; switch two rows in the table; or delete a row, as all rows that were to the right of it move one row up. Note that this is not permitted when the current row is one of the header rows.

F5 (Grab Column) – Call upon a menu of operations that can be performed upon an entire column in the table. The user may insert a duplicate copy of the column at the current location, moving the original column and all to its right one column to the right; copy a column to another location, overwriting whatever was there previously; move a column to another location, again overwriting whatever was there; switch two columns in the table; or delete a column, as all columns that were to the right of it move one column leftward. Note that the decision column may be involved in any of these operations; it is regarded as column 0. The variable table is automatically modified to maintain database consistency.

F6 (Type/Cost) – If the variable types are being displayed on the screen, show the costs instead. If the costs are being displayed, show the types instead.

F7 (Jump to Decision) – Immediately moves the cursor to the leftmost (decision) column.

F8 (Jump to Variable) – Immediately moves the cursor to the top of the current column.

F9 (Edit Domain) – During the process of entering examples, the user will often recognize the need to alter the Variables table. This function allows the user to go immediately into the Variables editor (Section 5.3), make changes there, and return to the Examples editor without leaving the current environment.

F10 (Scan Values) – As was emphasized earlier, the Examples editor will only accept values that have been defined as legal values for the current attribute. By repeatedly hitting the F10 key, the user may quickly scroll through the list of legal values for the variable, stopping when the correct one is reached. The scan is cyclical; after the last value has been shown, the first one will be shown again. Note that the use of the F10 key overwrites whatever was in the current cell.

The user can scan the values backwards instead of forwards by using **Shift-F10**, **Ctrl-F10** or **Alt-F10**.

F11 (Keys On/Off) – Move the display in or out of data key viewing mode.

Esc (Escape) – A menu offering options for transfer of control (e.g., Save Changes, Return to Parent Menu, Proceed to Learning Menu, Quit) appears, allowing the user to leave the Examples editor. If changes have not been saved, the user will be prompted to ensure that any desired save is performed before exiting.

Two other options are available in the Esc menu - to import or append a file. Importing a file means reading another file into the example table, overwriting the current database. The file should be in a flat 12 characters per field, one record per line format, or it can be a Dbase .DBF file. After reading the file in, it is a good idea to F9 to the Variables editor, enter the variable names and types, and save immediately; INLEN will automatically collect the values in the database and log them as legal values. You may have to reorder the values afterward if they belong to a linear or structured domain.

To append a file, INLEN adds the examples from another application system into the current Examples table, putting those records at the end of the table. Needless to say, the two systems must have compatible variable definitions.

5.5 Rules

Although the user may access many of INLEN's rule manipulation features through the Create/Modify and Browse modules, the rule base is more closely associated with the Learning and Discovery module, described in Section 5. Hence, this section will not discuss the details of the rule structure and the programs that access them; instead it will summarize the features available through the Create/Modify and Browse modules.

When the Create Rules operation is selected from the Develop System menu, the user has two choices - to learn them from the examples entered by the user (Section 5.1) or to enter them directly via an editor (Section 5.3). In the case of the latter option, the user has to be very careful to adhere to the syntax that INLEN will recognize (see Section 6). The user may build the file through a regular system editor, returning to the main menu when done. In example learning mode, the contents of the Examples table serves as the input for AQ15.

If Access Rules is chosen, there are options to edit rules directly (analogous to the direct entry described above and in Section 5.3), improve them with new examples stored in the Examples table, optimize them according to a new set of learning parameters (Section 5.1), test their consistency and completeness against testing examples (Section 5.2), append rules from another rule base, or compile the rules into a form readable by the advisory module (see Section 6).

If View Rules is selected, the user will be able to see a *compiled* rule set for the selected application system. The user is first prompted for the decision class whose rules should be shown. A "?" in response will bring up a menu of possible choices. The user may then scroll up and down through the text of the rules for that class. Note that if the application system's rule set has not yet been compiled, no rules will be displayed, and if a more recent

set of rules was entered or learned after the last compilation through rule learning or editing, the older, compiled rule set will still be shown. To the rule browser, the more recent, uncompiled rule sets are considered to be "under development".

5.6 Whole System

The Access, Copy and Delete commands may apply to an entire application system at once if Whole System is selected in the rightmost column of the Develop System menu. Access causes a summary to be displayed on the screen that indicates the system's level of development by showing which component files do or do not exist. Copy copies all system component files to another application system. Delete erases all system component files and removes the System Description from the active systems file. Because this is a drastic step, it is recommended that Delete System Description be used instead if there is any chance that the system files might be of use at a later date.

Whole System operations are not available within the Browse module.

6 Learning and Discovery

This section describes INLEN's Learning and Discovery module. While the Develop System modules primarily handle the maintenance of a database, this portion of the system is involved in the development of the knowledge base. Implemented Learning and Discovery functions include:

- Rule learning or optimization
- Rule testing
- Rule editing

A user may also access the Training and Testing Example Editors (Section 5.4) via the Learning Menu, since the set of active examples is critical to the learning process.

Table 1 shows the differences between rule learning (batch or incremental), optimization and testing in terms of input and output. For any of these functions, the user will be shown a screen (Figure 10 shows the screen for rule learning) asking for specification of the application systems containing the input examples and/or rules as well as (in the case of learning or optimization) the knowledge base to which the learned rules should be written. If the output rule base already exists, the user will be prompted to ensure that an overwrite is acceptable.

In the case of rule optimization, a ruleset is being modified without the use of any new training examples, and hence no decision attribute will have been specified. As a result, during the problem specification process, the user will be prompted for a decision attribute from among the variables for which rules exist.

If learning or optimizing, the user may select <Esc> to alter the Learning Parameters for the current task (Section 6.1.1). These parameters have no effect on rule testing. Additionally, when learning, the user may choose to learn a single concept rather than all decision classes. If a "?" is typed at the "concepts to learn" prompt, a menu will appear showing the possible decision classes.

Operation	Input	Output
Rule Learning (Batch Learning)	Training example set	Rule set for selected classes
Rule Improvement (Incremental Learning)	Initial rule set New training example set	Improved rule set consistent with new examples
Rule Optimization	Initial rule set New learning Parameters	Optimized rule set based on new parameters
Rule Testing	Rule set Testing example set	Analysis of rules' consistency and completeness with respect to testing examples

Table 1. Learning and Discovery Operators

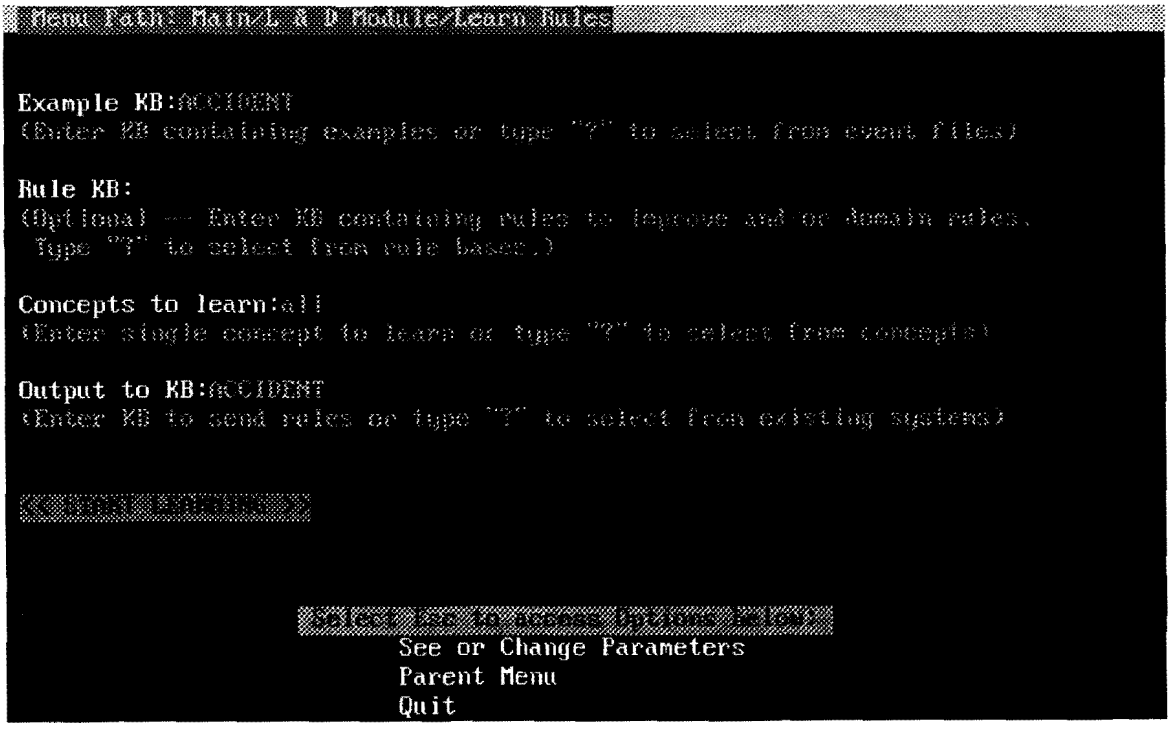


Figure 10. Preparing for rule learning

Users may scroll through the menu, accepting defaults or changing earlier entries, by using the arrow keys. The item labeled START LEARNING activates the learning engine when it is selected.

6.1 Rule Learning and Optimization

AQ15 (Michalski et al, 1986) is the primary mechanism for rule learning in INLEN. Given the input data definitions, parameters, examples and/or input rules, AQ15 generates rules for the specified decision classes that are as complete and consistent with respect to the

examples as possible. From the set of possible rules meeting the criteria, AQ15 chooses one based on the user's learning parameters. It should be noted that searching for *the* optimal rule is an NP-complete problem; instead, AQ15 discovers a rule guaranteed to be close to that optimal level in polynomial time.

While the AQ15 process is essentially the same, the difference between learning and optimization is the presence or absence of examples. The only inputs for optimization are a rule set and a set of learning parameters. Typical uses for optimization are the generation of discriminant rules from characteristic ones (these rules will likely be better than discriminant ones learned directly from examples (Cuneo, 1975) or fine-tuning of rules entered directly by a user. Rule learning when a rule set is present (incremental learning) only modifies the rule set to resolve inconsistencies between the rules and the new examples.

During learning, the display on the screen informs the user of AQ15's progress. The concept being learned is displayed, and a bar extends across the screen once for every rule being learned. It should be noted that there will likely be multiple rules for any particular decision class, and the progress of the bar does not reflect how many more rules will have to be learned for the class. Also, redundant rules may be removed during postprocessing, so that occasionally fewer rules for a class will be output than were initially generated.

6.1.1 Learning Parameters

The learning parameters are of critical importance in shaping the output from the learning engine. This section describes each parameter's function in the parameter set.. The screen to set the parameters is shown in Figure 11.

Mode for nominal output value – The rules for each decision class will cover the examples in that class without covering any of the examples from other classes. However, it will likely occur that there are parts of the event space not covered by examples of any class, and AQ15 may generate multiple rules that cover the same portion of these unassigned areas. By selecting *intersecting covers* mode (the default), the user indicates that this is not a matter of concern, that rules for different classes may intersect in the unassigned example space. If *disjoint covers* are selected instead, rules for different classes will not intersect anywhere in the event space.

In disjoint cover mode, a rule is learned for the first class by the normal technique of generating maximal descriptions that cover the positive examples of the class and none of the negative examples. However for learning rules for subsequent classes, the examples of classes already learned are removed from the negative example set, and are replaced by the entire space covered by the rules learned for those classes. Hence when learning in disjoint cover mode, the order in which the classes are learned (which is determined by order of first occurrence in the examples table) is significant. To demonstrate that, consider the following data set:

Class	Color	Size
1	Red	Small
1	Blue	Small
2	Yellow	Medium
2	Yellow	Large

In intersecting covers mode, typical learned rules might be:

Class = 1 if Size = Small Class = 2 if Color = Yellow

Note that a small, yellow object would satisfy the rules for both classes, and a large, red object would satisfy neither. In disjoint covers mode with class 2 learned first, AQ15 would learn something like:

Class = 1 if Size = Small Class = 2 if Size ≥ Medium

Class = 2 if Color = Yellow Class = 1 if Color ≠ Yellow

Either way, small, yellow objects would be classified as matching the first rule learned, and large, red objects would match the second one.

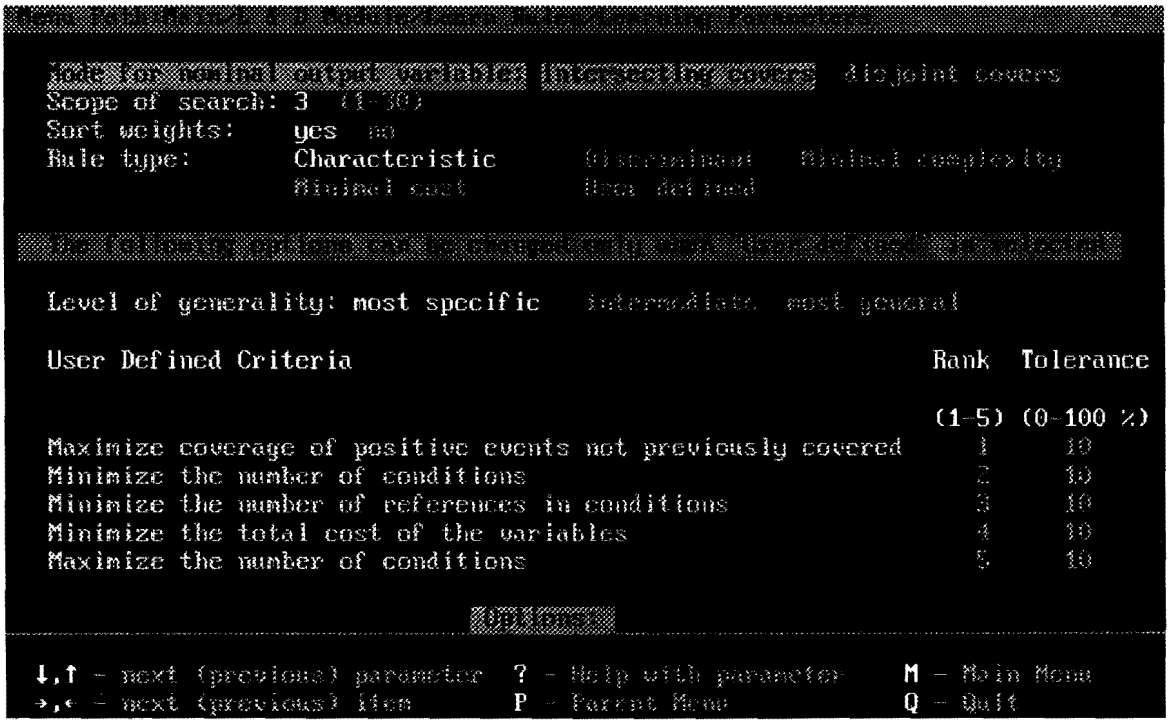


Figure 11. Selecting the Learning Parameters

Scope of Search – This parameter controls the depth of the search AQ15 will make. Legal values are 1 to 30, with the default set at 1. As this number becomes higher, the program will retain more intermediate hypotheses and is likely to return better results, but the learning process will take noticeably longer.

Sort Weights – When rules are produced for a decision class, numerical weights are assigned to them indicating how many examples of the class they cover. Rules with higher weights can be considered to be typical of the class, while "lightweight" rules likely

represent exceptional cases. INLEN will typically sort the output rules based on number of examples covered; this feature can, however, be turned off.

Rule Type – There are four different rule types (Table 2) made available by INLEN; each with its own set of optimality and rule generalization criteria for AQ15 to consult. In addition, a fifth type, User Defined, is available if the user wants to engineer a set of preference criteria other than one of the defaults.

Rule Type	Level of Generality	Primary Preference Criterion
Characteristic	Most Specific	Maximize coverage of positive examples
Discriminant	Intermediate	Maximize coverage of positive examples
Minimum Complexity	Intermediate	Minimize number of conditions in rules and number of values in conditions
Minimum Cost	Intermediate	Minimize total attribute cost in the generated rules

Table 2. System-defined Rule Preference Parameters

The level of generality in a rule determines how much of the event space the rules will cover, given that they cover a particular set of positive examples of a class and none of the negative examples. When Most Specific is selected, the rules will have as many conditions as possible, each adhering closely to the values found in the input examples. When Intermediate is selected, the individual conditions will remain as specific as possible, but there will be fewer conditions – conditions are removed whenever their absence will not cause the rule to cover examples of other classes. When Most General is chosen, the rules will have as few conditions as possible, with each one mapping an attribute to a maximal value set that does not cover any negative examples.

A user who selects User Defined rules rather than one of the four types shown in Table 2, may choose any of the three levels of generality, and also define a Lexical Evaluation Formula (LEF) for determining preferences among possible rules. To set up a LEF, the user orders these five selection criteria based upon their importance:

- Maximize the number of examples covered by the rule
- Minimize the number of conditions in the rule
- Minimize the number of attribute values in the rule's conditions
- Minimize the total cost of the rule
- Maximize the number of conditions in the rule

Each of these criteria is assigned a tolerance of 0 to 1 (with a default of 10%). During the learning process, when the number of hypotheses in consideration has to be reduced due to its exceeding the predefined scope of search, the hypotheses will be ranked based on the first (most important) criterion. The best scoring one will be retained, along with all others that are within the tolerance range of the best value. If too many hypotheses still remain,

the second criterion will be used to choose among them, and so forth. When learning is complete, the LEF will be used to determine which completed rule is the best one.

6.1.2 Postprocessing

After learning is complete, the user may choose to look at and evaluate the rules that were learned by AQ15 (Figure 12). Rules for a selected decision class will appear on the screen labeled with successive letters starting with A. The conditions for each rule will be labeled with successive numbers starting with 1. Each condition will have three weights associated with it - support level, number of positive examples of the class covered by the condition, and number of negative events covered by the condition. The support level is simply the percentage of examples covered by the condition that are positive examples of the class. Hence, it is an indicator of the informativeness of the condition, showing how strongly knowing that the condition is true supports the hypothesis that the decision is the proper one. Conditions in a rule are sorted by support level, from highest to lowest.

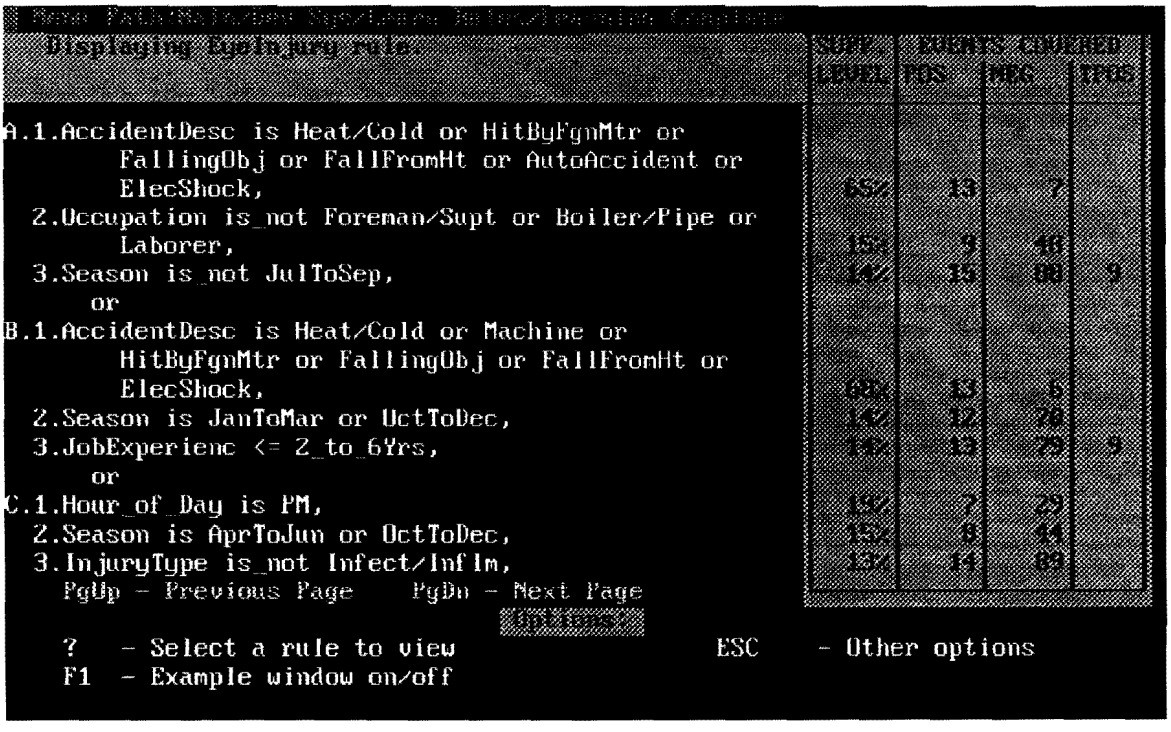


Figure 12. Displaying learned rules

Each rule as a whole has a weight associated with it, listed at the end of the rule. This is simply the number of examples of the class covered by the rule as a whole. When the sort weights parameter is activated, the rules for a class will be displayed in decreasing order of this value.

When in display mode, the user can compare the rules with the input examples by toggling on or off a window showing the examples table using the **F1** key. When this window is active, the user can transfer cursor control between the rule and example window by using the **F2** key. Arrow keys allow for scrolling through the example table.

When finished with viewing the rules, the user may opt to save them to the application system's knowledge base. **NOTE: This will overwrite the *entire* rule base for the given decision attribute, even if rules have been learned for only one class.** If the rules are saved, they will be written to two files - one with an .RLE extension that will be accessed by various portions of INLEN, and one with an .RLP extension for off-line printing and reading. The .RLE file will be located in INLEN's knowledge base (subdirectory KB\<name of application system>) and have a file name based on the first 8 characters of the name of the decision attribute. **It is therefore important that different attributes are assigned different names within the first eight characters.** Another file in this directory called SESSION will record the timestamp of the last update to the knowledge base for this application system.

For each decision class, the .RLP file indicates how many training examples belonged to the class. In the display of rules, there are five columns of weights - support level, positive examples covered, negative examples covered, commonality and positive examples covered by the entire rule. Commonality is an indicator of how representative a condition is; it shows the percentage of examples of the class that are covered by the condition. After each rule, the keyfields of the examples covered by the entire rule will be shown.

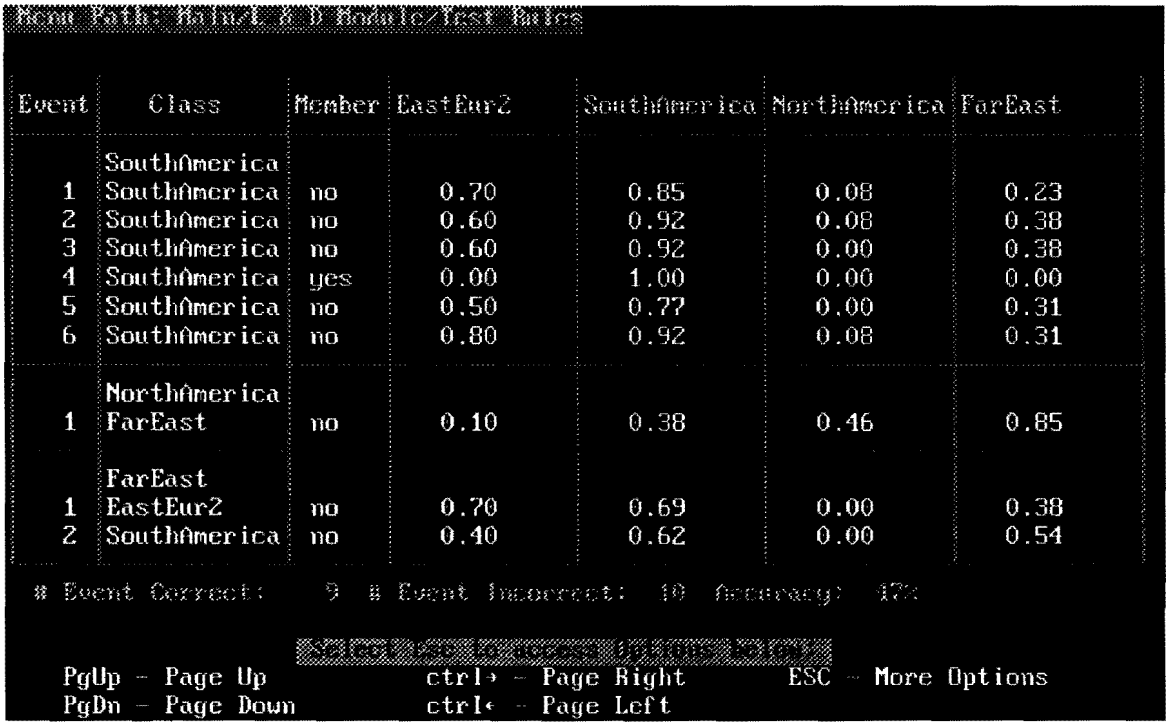
If the rules are to be used by the advisory module, they will need to be compiled. The user should select the Save and Compile option after learning and viewing the rules.

6.2 Rule Testing

ATEST (Reinke, 1984) is the program used by INLEN for testing rules against a set of testing examples. For each example, ATEST determines whether the example is covered by all of the conditions in any of the rules in the rule base. If so, the example is considered to match the class that that rule describes exactly, and a confidence value is assigned to that class based on the probabilistic sum of the confidence values for the individual rules for the class that cover the example. (The probabilistic sum of percentages a and b is defined as a + b - ab.) These individual rule confidence values are based on the rules' coverage of testing examples of that class. If there is no exact match, the confidence value for a class is equal to the probabilistic sum of the confidence values for the individual rules for the class, which in this case are the percentages of the conditions in each of the rules that the example satisfies. Examples are assigned to the class with the highest confidence value.

ATEST produces a table summarizing the performance of the rule base against the testing examples (Figure 13). In the lower part of the screen the overall summary is displayed, showing the number of examples correctly classified, the number of incorrectly classified examples, and the overall percentage of correct classifications. Above this, a table showing performance for each example is displayed. Each row represents an example, and the examples are sorted by class, with each class separated by horizontal lines. For each class, the name of the class is given first, followed by the information for its testing examples. Each example is identified by a number based on its position among the members of its class in the Examples table. Next to this number is the name of the class ATEST determined to be the best fitting for the example, followed by an indicator of whether the

example was exactly matched by any rules. The remaining columns show the confidence values generated for the various classes.



Event	Class	Member	EastEur2	SouthAmerica	NorthAmerica	FarEast
	SouthAmerica					
1	SouthAmerica	no	0.70	0.85	0.08	0.23
2	SouthAmerica	no	0.60	0.92	0.08	0.38
3	SouthAmerica	no	0.60	0.92	0.00	0.38
4	SouthAmerica	yes	0.00	1.00	0.00	0.00
5	SouthAmerica	no	0.50	0.77	0.00	0.31
6	SouthAmerica	no	0.80	0.92	0.08	0.31
	NorthAmerica					
1	FarEast	no	0.10	0.38	0.46	0.85
	FarEast					
1	EastEur2	no	0.70	0.69	0.00	0.38
2	SouthAmerica	no	0.40	0.62	0.00	0.54

Event Correct: 9 # Event Incorrect: 10 Accuracy: 47%

PgUp - Page Up ctrl→ - Page Right ESC - More Options
PgDn - Page Down ctrl← - Page Left

Figure 13. Displaying results of Rule Testing

If there are more than four classes, the user may bring other class columns into the display using the right and left arrow keys (which scroll across one column at a time) or the Ctrl-right and left arrow keys (which scroll across a full page at a time). To move up and down the example listing, use the up and down arrow keys or, for one-page-at-a-time movement, the PgUp and PgDn keys. Hitting the Escape key brings up a menu for transfer of control from ATEST, and also provides an option to save the results of the rule testing to a file.

6.3 Rule Editing

There are times when it may be desirable to edit a rule file manually, whether for fine-tuning an existing rule set or entering domain knowledge provided by an expert. INLEN provides an interface to on-line editors for this purpose. When the option to edit rules is chosen, control is transferred to the editor specified in file INLEN.EDT, or to the default system editor if that file is present. Needless to say, your system must have sufficient memory to be able to load the selected editor on top of the INLEN process, or an error message will appear.

Users editing an INLEN-learned rule file will notice that each condition is followed by a pair of numbers. These are *distinctiveness* and *commonality* weights of the conditions, and can range from 0 to 100 (a value of -1 in the distinctiveness column indicates that the value will be calculated during rule compilation). The distinctiveness weight is an indicator of how much information the condition carries with respect to determining that a decision

belongs in its class. When calculated dynamically by the rule compiler, it is the percentage ((T / C) - 1) / (T - 1), where $T \geq 2$ is the total number of decision classes, and C is the number of classes that contain rules in which that condition is met. For T = 1, distinctiveness = 100%. If a condition has multiple values (e.g., color = red or yellow), the distinctiveness for the entire condition will be the average of the distinctivenesses of the individual values.

The commonality of a condition indicates how representative that condition is of all the members of the class. A commonality of 75 means that we can expect 3/4 of the members of the class to meet the particular condition.

When editing the rule file, users should ensure that there are distinctiveness and commonality values for each condition, even if all or some of the former are -1. The other thing to notice is that the first line of an INLEN-learned file begins with "learn". This tells the system that these rules were learned by the system, and can be counted on to follow a strict format with respect to the columns in which the weights are found. It is suggested that users editing a rule file remove this entire line and the learning parameters lines that follow it.

7. Rule Compilation

The primary purpose of rule compilation is to generate the cross-reference tables that are required by the Advisory Module. In doing so, the compiler calculates distinctiveness values where necessary (Section 6.3), and checks the rule file for errors, both fatal and non-fatal. The rules for each decision class must adhere to syntactic structure described below.

Each ruleset begins with the line:

<decision attribute> is <decision value> if:

This line is then followed by the first of the rules for this class. These rules are labeled with successive letters starting with A. They are separated from one another by lines that simply read "or".

Conditions in each rule take the form:

<condition #>. <variable name> <relation> <value set> <punctuation>

where conditions in each rule are numbered successively from 1. Legal relations are is, is_not, <, >, <= and >=. Value set consists of a single legal value for the variable, although if the relation is "is" or "is_not", multiple values separated by "or"s are permissible. Additionally, in the case of linear, integer, or numeric variables, when the relation is "is" or "is_not", individual values may be replaced by ranges, such as low to high or 10 to 25. The punctuation mark at the end is always a comma, except in the case of the last condition of the last rule for the decision class, in which case, a period is used.

Each condition is followed on the same line by distinctiveness and commonality weights.

An example of a simple rule base is shown in Figure 14.

Class is yes if:		
A.1.Color is Red or Yellow or Blue,	75	100
2.Price <= Moderate,	40	85
3.Length is 10 to 25,	25	100
or		
B.1.Price is Very_high.	100	15
Class is maybe if:		
A.1. Length is 0 to 9 or 26 to 40.	100	100
Class is no if:		
A.1.Color is Green,	100	65
or		
B.1.Price is High,	70	92
2.Length > 40.	63	95

Figure 14. Rule Format for INLEN

Users may wish to insert comments into a rule base. Text delineated by { and } will be ignored by the compiler. Comments can appear anywhere, including across multiple lines, provided that they are separated from the actual text of the rules by at least one character of whitespace.

Fatal errors occur when the compiler finds an item such as a variable name, value or keyword missing, "to" is used with non-linear variables, an undefined value is used in a linear or structured domain, etc. Non-fatal errors include a word being too long (it is then truncated) or a variable's type being unknown (it is assumed to be nominal.) In these cases, warnings appear on the screen.

8. The Advisory Module

In this module, INLEN draws from its knowledge base in order to provide users with advice about various problems they define for the system. Section 8.1 provides an overview of the screens and choices presented in this module, Section 8.2 takes a closer look at the inference parameters that shape the module's performance, and Section 8.3 discusses the algorithms used by the Advisory Module.

8.1 Description of an Advisory Session

A user can enter the Advisory Module in one of two ways – either by selecting it through the Main Menu or, if the sole purpose of the INLEN session is to get advice from a particular application system, by typing INLEN followed by the name of the application system at the DOS prompt. The latter method will bypass the screens involved in the selection of an advisory system.

When the module is entered from the Main Menu, the user will be able to select from a list of advisory systems. If there is no Long Description File (Section 5.2) associated with the system, the advisory session will begin immediately after system selection. If such a file exists, it will be displayed on the screen; this offers users the opportunity to find out a bit

more about the domain they will be looking at, and to ensure that they have selected the proper advisory system. If not, options for going back to the system selection menu or leaving the Advisory Module altogether are available. Once the advisory system has been selected, if rule bases exist for more than one decision attribute, the user will be prompted for the decision attribute to be determined during the advisory session.

An advisory session is a three stage process, consisting of *reduction*, *discrimination* and *confirmation*. During reduction, the user is asked for the values of variables whose importances have been defined as 10 (in its Annotation). A selection is made by using the arrow keys to highlight the proper response and then hitting Return or, in the case of integer or numeric variables, typing in a value within the variable's legal range. The purpose of this phase is to remove immediately from consideration any hypotheses that contradict the user's input. (As an example, consider the process of choosing a restaurant at which to eat. If you decide you want Chinese food, you won't even consider a nearby Mexican restaurant, no matter how well it meets your needs in terms of price, convenience, etc.) If more than one hypothesis remains after reduction, INLEN continues to the discrimination phase, in which the effort is to single out one outstanding candidate hypothesis from the set of hypotheses still under consideration. The purpose of the final phase (confirmation) is to confirm or disconfirm that particular hypothesis; if it is disconfirmed, a new leading candidate hypothesis is selected, and confirmation continues. During all three phases, INLEN keeps the user notified of its progress and its confidences in the leading hypotheses in the screen's upper windows (Figure 15).

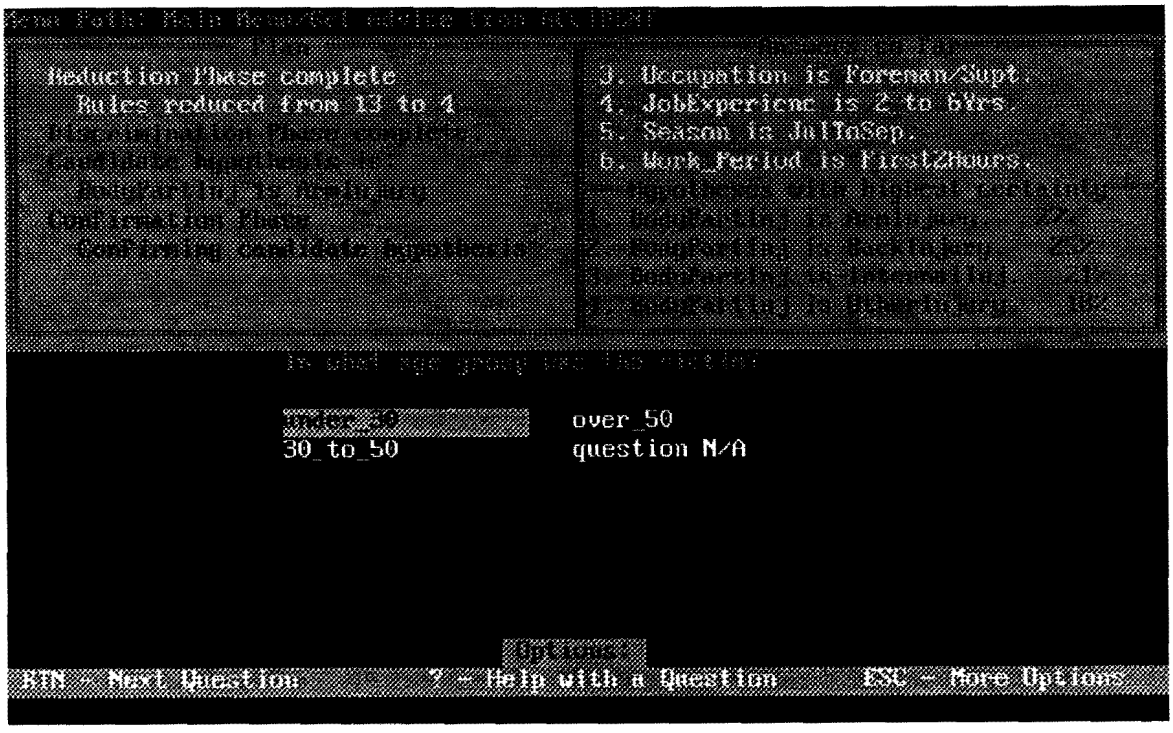


Figure 15. Example of an Advisory session

At any point, the user may interrupt the questioning process by hitting Esc. Options will then appear allowing the user to exit or resume the advisory process. Two other utilities are available:

- **View Alternative Hypotheses**, in which all of the hypotheses still in consideration are listed along with a bar chart display of their confidence levels. This option is only applicable after the reduction phase has been completed.
- **Access Rule Base**, in which the user may inspect the rules for a decision class during the inference process (Figure 16). A rule's confidence level (or notification that the decision was eliminated during reduction phase) appears at the top of the screen. Highlighted in the rule display are the values selected by the user that satisfy the listed conditions, and the values in those conditions that match the user's responses. Note that the two may not always coincide, particularly in the case of is_not conditions.

A user may also interrupt the questioning by hitting '?'. In this case, if there is a helpfile available for the highlighted value (named during the entry of value annotations described in Section 5.3.1), that file will be displayed on the screen.

The confirmed candidate hypothesis, or (if none was confirmed) the one with the highest confidence value when all questions were exhausted, will be the one INLEN suggests to the user. If a decision file was specified in the decision value's annotations, its contents will be displayed on the screen. Otherwise, a simple statement of the decision will appear.

Figure 16. Displaying rules under consideration

The user then has several options. In addition to leaving the Advisory Module and returning to the main menu or quitting INLEN, one may:

- **View Alternative Hypotheses** (as described above)
- **Access Rule Base** (as described above)
- **Execute Advice**, in which a program specified in the decision value's annotations is executed. INLEN execution continues after that program has finished.
- **Change an Answer**, where a user may correct a mistaken entry, alter conditions somewhat, or consider "what if" scenarios. INLEN prompts the user for the identity of the answer to change, followed by the new value. When all necessary answers have been changed, the user selects "Run Updated Session". INLEN remembers previous unchanged answers, and only asks the user in cases where further information is necessary.
- **Record Session**, in which the user may save the details of an advisory session, including decision, confidence values, values of independent variables, and a short descriptive note, to a file.
- **Start Another Advisory Session**, which brings the user back to the system selection menu.
- **Save Example**, in which the example, as defined by the responses given by the user during the advisory session and a decision defined by the user rather than by INLEN, is added to the application system's training example table, potentially to be part of further learning.

8.2 Inference Parameters

The inference parameters guide the progress of INLEN's advisory module. In particular, they manage the updating of confidence values and the progress from one phase of the advisory session to the next. The user can modify the inference parameters for an advisory system through a screen (Figure 17) accessed via the Develop System menu (Section 5.1). This section describes each parameter's function in the parameter set..

Multiple complex evaluation – There will often be two or more rules pertaining to a particular decision in consideration, even after irrelevant rules have been eliminated during the reduction phase. This parameter instructs INLEN in how to calculate the *degree of match* for a decision, given the degrees of match of its various relevant rules. If **Maximum** (the default) is chosen, the decision's degree of match will simply be the maximum of its rules' degrees of match, while if **Probabilistic Sum** is chosen, the degree of match for the decision will be equal to the probabilistic sum (sum minus product) of its rules' degrees of match. For example, if a decision has two relevant rules with degrees of match 50% and 75%, the Maximum method would give a 75% degree of match for the rule, while the Probabilistic Sum would be (.5 + .75) - (.5 * .75) = 87.5%.

Commonality constant – During the confirmation phase, the degree of match of each rule under consideration is adjusted based upon responses to questions about various features of the data. If the answer satisfies a condition in the rule, the degree of match is increased, and if it contradicts the condition, the degree of match decreases. The amount by which the degree of match changes is the product of the commonality of the condition and the commonality constant (default = 15%). For example, with a commonality constant of 15% and a condition's commonality of 80%, an answer relevant to that condition would adjust its rule's degree of match by 12%. Legal values for this parameter are 0 - 100%.

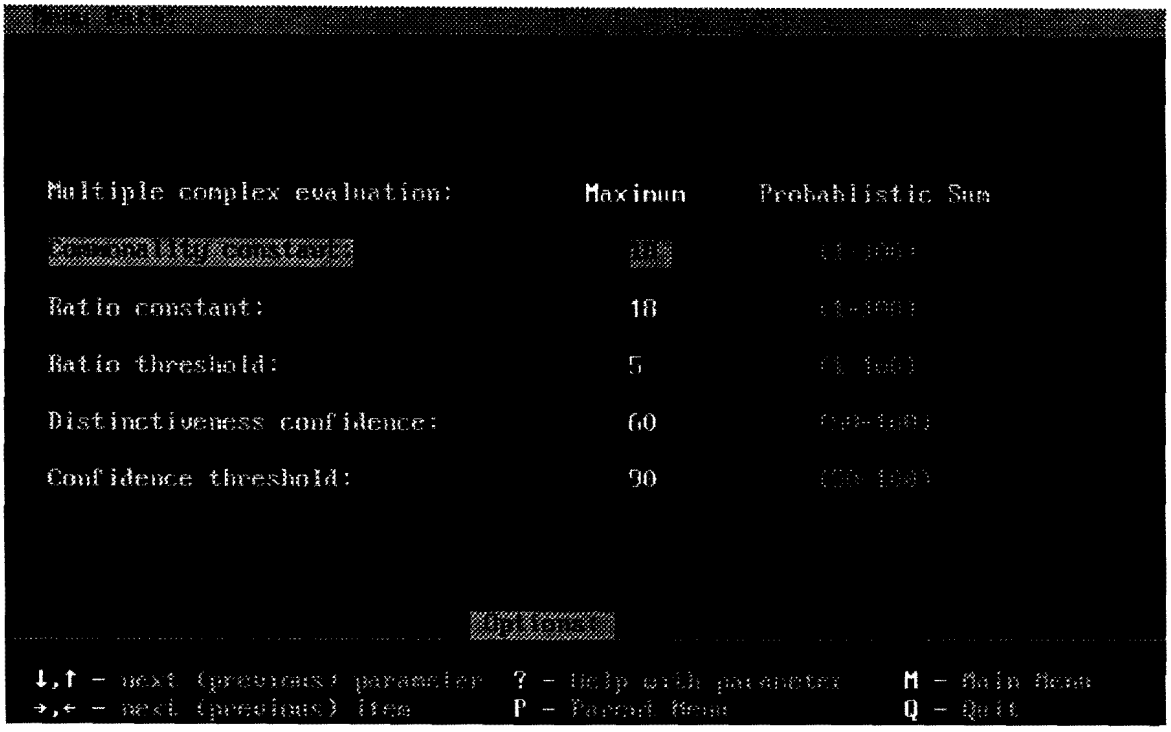


Figure 17. Setting the Inference Parameters

Ratio constant – During the confirmation phase, the difference between the degrees of match of the leading candidate hypothesis and its closest competitor is a strong indicator of our confidence in the leading hypothesis. INLEN calculates its degree of confidence for the leading candidate hypothesis as $(DM_1 + RC) * (DM_1 - DM_2) / DM_1$, where RC is the ratio constant, DM_1 is the degree of match for the leading candidate hypothesis, and DM_2 is the degree of match for the next best hypothesis. For example, with a ratio constant of 15% (the default) and the two leading candidate hypotheses having degrees of match of 50% and 20%, the degree of confidence in the leading hypothesis would be 39%. Legal values for this parameter are 0 - 100%.

Ratio threshold – The discrimination phase continues until all questions are exhausted or until one hypothesis stands out sufficiently far above the others, as determined in part by this parameter. The discrimination phase will end when $(DM_1 - DM_2) / DM_1$ exceeds this threshold, where DM_1 is the degree of match for the leading candidate hypothesis and DM_2 is the degree of match for the next best hypothesis. Legal values for this parameter are 0 - 100%, with a default of 15%.

Distinctiveness confidence – During the reduction and discrimination phases, a rule's degree of match is adjusted based on the distinctiveness of the answers given, i.e., a more unique answer will provide stronger support of a rule satisfied by it than a more commonplace answer will. The distinctiveness confidence is the user's level of confidence in the distinctiveness values given to or generated by the program, and should increase with the number of examples used in learning. When the distinctiveness confidence is 100%, a rule's degree of match is increased (or decreased) by taking the probabilistic sum

(difference) between the current degree of match and the distinctiveness of the new condition that was (not) satisfied. For lower values, the amount of increase or decrease is multiplied by the distinctiveness confidence Legal values for this parameter are 50 - 100%, with a default value of 90%.

Confidence threshold – Confirmation continues until all questions have been exhausted or the leading candidate's degree of confidence exceeds this threshold. Legal values for this parameter are 50 - 100%, with a default of 90%.

8.3 Summary of the Inference Process

During the reduction phase, values are sought for all attributes with an importance of 10. For each rule, INLEN keeps track of whether it is still in consideration due to no contradictions with the user's answers. As values are given, the degree of match for rules still under consideration with conditions satisfied by the values are increased based on the conditions' distinctivenesses.

At the end of reduction, decisions with rules which have not been eliminated, and their degrees of match are calculated by taking the maximum or the probabilistic sum of their active rules' degrees of match. Meanwhile, distinctiveness values for the attributes without an importance of 10 are recalculated based on the rules that are still in consideration. The order of questioning during the discrimination phase will be based half upon these distinctiveness values, and half upon the importance values for the attributes provided during definition of the domain.

If more than one candidate decision remain, the discrimination phase finds a leading candidate hypothesis through further questioning. Each answer adjusts the degrees of match of the active rules whose conditions refer to the feature in question based on the distinctiveness of the answer and the distinctiveness confidence parameter. Eventually, the ratio threshold is satisfied, and INLEN can move on to confirmation.

During the confirmation phase, this process continues (one possibility for future implementations of INLEN is to make commonality a factor in the ordering of questions during the confirmation phase, but at present, it simply picks up where discrimination left off), only now degrees of match are updated based upon commonality rather than distinctiveness. The rationale for this is that the goal of this phase is not to distinguish between candidate hypotheses, but rather to confirm or disconfirm the leading candidate; in doing so, answers that are more representative of the hypothesis will more strongly support the confirmation. Rules' degrees of match are updated based on the commonality constant and the leading candidate's confidence value is based on the ratio constant. Confirmation continues until all questions are exhausted, the degree of confidence of the leading candidate hypothesis exceeds the confidence threshold, or the candidate is disconfirmed (i.e., its degree of match falls below that of another hypothesis.)

9 Future work

INLEN-3 is the first in a series of systems for knowledge discovery in databases. Each of its successors will have new capabilities and operators for more versatility in its knowledge discovery tasks. There are plans for a more portable system that can run under MS-Windows or on a Sun or Macintosh machine, and work is underway to develop an interface between INLEN and an Oracle database. Among the new operators and capabilities that will soon be added to INLEN are:

- User-driven selection of examples from large data sets
- Conceptual clustering of records in the database
- Constructive induction for rule learning
- Decision tree-based knowledge representation and inference
- A package for statistical analysis of data
- Multiple answer capability during advisory sessions

References

Baim, P.W., "The PROMISE Method for Selecting Most Relevant Attributes for Inductive Learning Systems," Report No. UIUCDCS-F-82-898, Department of Computer Science, University of Illinois, Urbana IL, Sept. 1982.

Cuneo, R.P., "Selected Problems of Minimization of Variable-Valued Logic Formulas," Masters Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1975.

INIS (International Intelligent Systems, Inc.) "User's Guide to AURORA 2.0: A Personal Inference and Discovery System for Automated Rule Acquisition," Fairfax, VA, 1988.

Kaufman, K. "Comparing International Development Patterns Using Multi-Operator Learning and Discovery Tools," *AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle WA, August, 1994, pp. 431-440.

Kaufman, K., Michalski, R.S. and Kerschberg, L., "Mining for Knowledge in Data: Goals and General Description of the INLEN System," Piatetsky-Shapiro, G. and Frawley, W.J. (Eds.), *Knowledge Discovery in Databases*, AAAI Press, Menlo Park CA, 1991, pp. 449-462.

Michalski, R.S., Kerschberg, L., Kaufman, K. and Ribeiro, J., "Searching for Knowledge in Large Databases," *Proc. First International Conference on Expert Systems and Development*, Cairo, Egypt, April 1992.

Michalski, R.S., Kerschberg, L., Kaufman, K. and Ribeiro, J., "Mining for Knowledge in Databases: The INLEN Architecture, Initial Implementation and First Results," *Journal of Intelligent Information Systems: Integrating AI and Database Technologies*, Vol. 1, No. 1, August 1992, pp. 85-113.

Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N., "The AQ15 Inductive Learning System: An Overview and Experiments," Report No. UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

Reinke, R.E., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta-Expert System," Master's Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1984.

Wnek, J., Kaufman, K., Bloedorn, E. and Michalski, R.S., "Selective Induction Learning System AQ15c: The Method and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 95-4, George Mason University, 1995.

Appendix A Representaion of Structured Domains

A brief introduction to structured variables was given in Section 3.5, and their basic use was described in secion 5.3. To reiterate, they can be used for hierarchical representations of inheritance patterns in the domain. In simple structures, the user can represent these patterns by ordering the values and using the Tab key to specify depth in the hierarchy.

INLEN only allows structures to be entered five levels deep, and only represents simple, acyclic hierarchies. However, facilities exist to allow the user to go beyond these limitations, by defining two data types to work in conjunction with the structured data type: *substructure* and *sview*. Additionally, a user may use these facilities to instruct INLEN that certain nodes in the hierarchy are more significant to the user than either their parents or children, and learning should therefore focus on those *anchor nodes*.

The basic idea is as follows: When INLEN detects a column in the Variables table of type structured, followed immediately by one of more columns of type substructure and/or sview, the program will regard those columns as representing *different portions of a single attribute*. Each column is integrated into the framework of the structure as a whole, rather than being regarded as its own separate entity.

As its name suggests, a column in the Variables table of type *sview* represents an alternative view of the same structure. One may envision a variable AnimalType with such top-level values in its hierarchy as Mammal, Fish, Insect, etc. and leaves with values such as Dog, Piranha, Butterfly, etc. However, biological classification is not the only way of grouping animals. For certain tasks it may be more useful to organize them by habitat, i.e., a structure with the same leaves organized differently and with top-level nodes such as Temperate, Desert and RainForest.

An INLEN user can represent these different views by setting up a structure called Animal organized by biological classification followed immediately by a column of sview type, called Habitat. Names that are the same as ones in the previous column represent the same node in the structure, but the new links and nodes are added into the structure as provided. Hence Butterfly can be a descendant of both Temperate and Insect. The learning operators will consider both views and automatically select the representation best fitting the problem.

The *substructure* variable type can be used to extend structures beyond their natural maximum depth or to designate certain nodes in the hierarchy as being especially significant. Like the sview columns, the substructure columns of a structure are defined in the columns immediately to the right of the main structure. The variable name of the substructure **must be the name of a value of one of the nodes** in the structure defined in one of the columns to its left. Top-level values in the substructure are considered to be direct descendants of that node. For example, the structure described above could have a substructure named Dog with top-level values of Terrier, Retriever, Beagle, etc.

In addition, the designation of a node as the head of a substructure adds an extra degree of importance to the node; it is designated an anchor node of the structure. In the example above, since Dog is the head of a substructure, learning will tend to focus on that node. All things being equal, a rule will have the condition *Animal is Dog* rather than *Animal is Mammal* or *Animal is Beagle*. By designating Dog as the head of a substructure, the user will have informed the system of the node's importance. This corresponds to everyday cognition where in many endeavors, a person is likely to think of a beagle as a *dog* rather than as either a mammal (more general) or a beagle (more specific).

Appendix B System parameters and limits under INLEN 3.0

Maxinum Number of Application Systems:		30
Maximum Number of Application Domains:		20
Maximum Number of Variables:		95
Maximum Number of Values per Variable:		200
Maximum Number of Examples:		650
Maximum Depth for Single Structured Variable Column:		5
Range of Integer Variable		0-50
Minimum Number of Discretized Intervals (numeric variables):		number of classes / 2
Maximum Value of *Maxstar* Learning Parameter:		30
Maximum Number of Values Displayed in Advisory Module:		50
Cutoff Value for Confirmation:		70
Default Learning Parameters:		
	Scope of Search (Maxstar):	1
	Intersecting Covers Permitted?	Yes
	Show Rule Weights?	Yes
	Rule Type:	Characteristic
Default Inference Parameters:		
	Ratio Constant:	0.15
	Commonality Constant:	0.15
	Multiple Complex Evaluation:	Maximum
	Ratio Threshold:	15
	Distinctiveness Confidence:	0.9
	Confidence Threshold:	90

Appendix C Executable modules in INLEN

ADVV.EXE	Advisory System
AQ15.EXE	Rule Learning, Optimization and Testing
AURMAIN.EXE	Main Menu
COMP_DIS.EXE	Display Status of Application System
CREATE.EXE	Compile Rules
DATA.EXE	Examples Editor
DRAWAUR.EXE	Logo Graphics (demo mode)
GEM_COMP.EXE	Postprocessing after Learning/Optimization
INFPAR.EXE	Inference Parameter Editor
INLEN.EXE	Controls Flow, Calls Other Modules
INTRO.EXE	Introductory Description (demo mode)
MKB.EXE	Develop System and System Description Menus
PARAM.EXE	Learning Parameter Editor
REV.EXE	Browse a System
TUT.EXE	Tutorial on How to Use INLEN
VARB.EXE	Variables Editor
VARSEL.EXE	Select Most Relevant Attributes

Appendix D Files used by INLEN

System Files

AUR.HLP	Advisory System Tutorial
DEV.HLP	Developing Application System Tutorial
EXPBOOL.FIL	Booleans Global to the "exp" Modules
EXPINT.FIL	Integers Global to the "exp" Modules
EXPSTR.FIL	Strings Global to the "exp" Modules
GEM.HLP	Helpscreens for Learning Parameters
GLOBBYTE.FIL	Global "byte" Storage
GLOBINT.FIL	Global Integer Storage
INF.HLP	Inference Parameter Helpscreens
INLEN.EDT	Editor to be used for Rule Editing
INSTR.HLP	Instructions for Creating an Application System
LDMENU.HLP	Learning and Discovery Module Tutorial
LEARN.AUR	Default Learning Parameters
MKB.HLP	Helpscreens for Develop System Module
MKBINT.FIL	Integers Global to the "mkb" Modules
MKBSTR.FIL	Strings Global to the "mkb" Modules
SYSTEMS.AUR	Active Application Systems
VARSEL.HLP	Helpscreens for Attribute Selection Module

Application System File Extensions

Note: <as> represents the name of the application system; <da> represents the name of the decision attribute.

KB\<as>\<da>.ADV	Cross-Reference Tables for Advisory System
KB\<as>\<da>.CIN	Index for .CRL File
KB\<as>\<da>.CRL	Rules that have been Compiled
<as>.DTA	Examples Table
<as>.DTM	Temporary Copy of .DTA File
<as>.GEM	Examples for Input to AQ
<as>.INF	Inference Parameters
<as>.LRN	Learning Parameters
KB\<as>\<da>.RLE	Rules
<as>.RLP	Last set of learned rules, formatted for printout
<as>.TST	Output from ATEST
<as>.TTA	Testing Events Table
<as>.TTM	Temporary Copy of .TTA File