SYNTEL™: KNOWLEDGE PROGRAMMING USING FUNCTIONAL REPRESENTATIONS

Rene Reboh and Tore Risch

Syntelligence, Inc.
1000 Hamlin Court, PO Box 3620
Sunnyvale, CA 94088

ABSTRACT

SYNTEL™ is a novel knowledge representation language that provides traditional features of expert system shells within a pure functional programming paradigm. However, it differs sharply from existing functional languages in many ways, ranging from its ability to deal with uncertainty to its evaluation procedures. A very flexible user-interface definition facility is tightly integrated with the SYNTEL interpreter, giving the knowledge engineer full control over both form and content of the end-user system. SYNTEL is fully implemented and has been successfully used to develop large knowledge bases dealing with problems of risk assessment.

I INTRODUCTION

A large class of expert system applications concerns judging risks under conditions of uncertainty. Problems of financial risk analysis--with which we have been particularly concerned--have this characteristic, but it arises in many other fields, such as public health and safety.

An effective knowledge representation language addressing such domains must satisfy a number of requirements. The assessment of financial risk involves a combination of quantitative analysis and qualitative judgment. Both quantitative and qualitative factors might have either high or low confidence associated with them, imposing a need to manage uncertainty uniformly across different types of variables. Financial risk analysis often requires consideration of sets of similar objects (such as geographical sites or years), suggesting that the representation and manipulation of large sets requires efficient support.

For the end user, the analysis of financial risk may require identifying and analyzing a large amount of (possibly conflicting) data. Accordingly, data must be presented (and requested) in a compact, familiar form; equally important, the user must be spared the need even to look at system displays that are not immediately relevant .

For the knowledge engineer, the knowledge representation must afford a natural means to structure the assessment criteria used by experts, and to do so on a large scale. This goes beyond the availability of a well-developed knowledge-acquisition tool set and the design of a convenient syntax, although surely both are important. It requires the architecture of the language to mirror the structure of explicit domain expertise and to hide all inessentials from the knowledge engineer.

These and other requirements were sufficiently demanding that we were persuaded to design and implement a new language. Our philosophical point of view is that a purely non-procedural representation of knowledge most naturally reflects the way experts appear to think in our domains of interest: Experts are very much concerned with relations among factors and subfactors, but appear far less concerned with the order in which factors are considered--provided, of course, that all potentially relevant items are considered before a recommendation is made. SYNTEL is thus at one end of the "what-to-how" spectrum and bears certain similarities to other non-procedural languages such as LUCID [Wadge & Ashcroft, 1985], BATTLE [Slagle & Hamburger, 1985] and that described by [Lucas & Risch, 1982].

Among the more widely used expert system languages, SYNTEL might be compared with a production language like OPS5 [Brownston, et al., 1985]. However, SYNTEL differs from OPS5 in using an alternative to a recognize/act architecture, in the high level of primitives it supports, in its treatment of inexact reasoning, its runtime features, and its integration with a user interface-definition facility.

The following sections discuss SYNTEL's use of a functional representation of knowledge, its inference procedure, the user interface facility, and some run-time features of the language. A final section comments on some lessons learned.

II KNOWLEDGE REPRESENTATION

The basic entities in SYNTEL are *variables* and *functions*. Variables can represent input data, intermediate levels of assessments, or final (i.e., output) assessments. Functions define mappings between variables. A SYNTEL program (i.e., a knowledge base) consists in large part of a collection of variable and function definitions.

A. Variables

A SYNTEL variable does not hold a single value but instead represents a set of probability distributions indexed by *formal parameters*. For example, a variable *EconomicOutlook*, indexed by the single parameter *Region*, might contain the following information:

Region	Weak	Avg	Strong
NE	.3	.6	.1
SE	.4	.5	.1
NW	.1	.4	.5
SW	.1	.2	.7

The column headings indicate that *EconomicOutlook* can take on the values *Weak*, *Avg*, and *Strong*. Each row of the table holds a probability distribution over these values for an *instance* of the variable, each instance corresponding to a value of the *Region* parameter. In general, variables can be indexed by any number of formal parameters.

The variable *EconomicOutlook* is ordinal-valued and the probability distributions describing its instances are discrete. Variables in SYNTEL can also have logical, nominal (e.g., a city name), string or real values. The probability distribution of a real-valued variable is represented by a mean and a variance for each variable instance.

Because uncertainty, in various guises, plays such an important role in financial assessment problems, we have provided additional mechanisms for its representation. For example, variables and parameters can both have *Unknown* as an explicit value, with consequences that are context dependent. An unknown parameter causes the creation of a partially-indexed set, an unknown logical variable is treated using a 3-valued logic, and so forth. Other mechanisms for managing uncertainty include support of prior and default probability distributions over variables.

Our interest in the efficiency of the knowledge engineering process, together with our desire for extensive error checking facilities, has led us to develop a powerful type system. The language has a handful of built-in variable types, but a typical knowledge base contains many times that number of additional variable types defined by the knowledge engineer. This extensive use of typing leads to code-sharing (further enhanced by an inheritance hierarchy of types), and to improved consistency of large knowledge bases. Control of errors is facilitated by compile-time and run-time type-checking.

B. Functional Representations

We have made a strong commitment to knowledge representation based on functional mappings between variables. To describe this approach by example, suppose a real-valued variable *Population*, indexed by *Region*, is to be combined with our previously-defined *EconomicOutlook* to estimate the size of a target market of interest. This would be represented by

TargetMarket(Region) <=
f[EconomicOutlook(Region),
Population(Region)].

The knowledge engineer is not free to define the function *f* arbitrarily. Rather, he or she selects the computational form of *f* from among the several dozen supported in SYNTEL. Accordingly, this selection process--and, for some functions, specification of additional information like weighting factors--plays a central role in the knowledge engineering process.

The functional forms supported by SYNTEL, called *ComputationTypes*, fall naturally into several families. Beyond the expected arithmetic and logical functions, there are important families of functions for performing a variety of tests and manipulations on sets, for combining or weighting variables in several ways, for testing the current state of the computation, and for accessing databases.

Certain *ComputationTypes* have the important ability to combine arguments of different modalities probabilistically. The simplest such, *Table*, illustrates the principle. To use it to compute *TargetMarket*, a table such as the following would be defined, which would apply to each *Region*.

	EconomicOutlook		
TargetMarket	Weak	Avg	Strong
Population			
0 - 40	10	20	25
40 - 50	15	25	30
50 - +INF	15	35	45

The table extensionally represents the expert's judgment of how an ordinal and a real-valued variable are combined to estimate the real variable *TargetMarket*.

Suppose there were uncertainty in the arguments because each is an estimate of the value, say, in the year 1995. Then each argument would have an associated probability distribution, discrete for *EconomicOutlook* and normal for *Population*. Integration of the normal over the indicated ranges, together with an assumption of independence, allows the mean and variance of *TargetMarket* to be computed.

While simple tables are useful, judgments of this sort are usually combined using a more complex *ComputationType*, called *Weight*, that implements a non-linear, probabilistic voting scheme. *Weight* uses a mechanism analogous to *Table*, in combination with summation and a "soft" threshold, to map between variables of arbitrary modalities.

With the intense current interest in methods for managing inexactness in expert systems [e.g., Pearl, 1985; Heckerman, 1985; Zadeh, 1985; Gordon & Shortliffe, 1985], it may seem foolhardy to have designed and implemented an alternative representation for experts' subjective beliefs. Nonetheless, the "rate and weight" style of reasoning we found prevalent in financial institutions led us to develop the *Weight* mechanism which, extending earlier work [Duda and Reboh, 1984; Reboh, Reiter and Gashnig, 1982], eases the problem of representing the "apples to oranges" comparisons that are an essential part of financial risk assessment.

Even a cursory description of the remaining *ComputationTypes* would far exceed space limitations, but we make a few general comments. Every function works with probability distributions as inputs; independence of arguments is assumed as needed. Distributions are typed; e.g., normal, discrete, exact, or undefined. However, unlike the strong typing of variables, distribution types can be determined only at run-time. All probabilistic calculations converge smoothly to deterministic calculations when a variable is known with certainty.

Each argument to a function can be a set (of distributions), so it is important to support set manipulations within argument definitions. We have therefore provided means for selecting, aggregating, disaggregating, and defaulting variable instances. A simple example of this is the selection of the *MaximumInstance* of a variable. In the illustration above, this could be used to identify the *Region* having the largest *TargetMarket*.

Continuing the illustration, we can easily imagine an expert specifying *EconomicOutlook* to depend on a variety of additional quantitative and qualitative variables. One or more additional equations would therefore have to be written to define *EconomicOutlook* and its arguments until a primitive input level was reached. Mathematically, the principal operation for articulating a knowledge base is thus function composition. The computational correlate of this is function evaluation, which will be further described below.

Before moving on, though, we should re-emphasize the purely declarative nature of this representation. The "<=" signifies function definition, not assignment; the right hand side of the equation contains no assignments, deletions, or other side effects. Thus, the order of function evaluation is controlled entirely by the SYNTEL interpreter, not by the knowledge engineer. This separation is more than a theoretical nicety. In practice, the knowledge engineer is not concerned with control issues. (See [Schor, 1986] for a further discussion of this point.)

III INFERENCE

A single inference step in SYNTEL consists of the side-effect-free evaluation of a function, given the value of its arguments. This step, called *propagation*, takes place whenever the value of an argument to any function is changed. If the function being evaluated is itself an argument of a second function, a subsequent evaluation will be performed. The basic evaluation strategy is thus forward chaining, and the result of an inference cycle is to maintain the values of functions consistent with the (new) values of their arguments.* Propagation is the central operation of SYNTEL's inference engine.

A. Inference Networks

The nesting of function compositions is compiled into an explicit data structure called the *static knowledge base*. The static knowledge base can be represented as a directed acyclic graph in which nodes correspond to functions and arcs point to a function from its arguments. We call these graphs *inference nets*, following the terminology established in the PROSPECTOR system for analogous but simpler structures. Nodes with no incoming arcs represent special elementary functions that accept input data from the user or from external data sources. Nodes with no outgoing arcs represent "top-level" functions whose values are among those displayed to the user.

A typical cycle of operation begins with the user providing a new value for any node in the inference net to which he or she has access. (Access mechanisms are described in the following section.) Subsequent function evaluations can be visualized as a dataflow originating at the changed node and propagating upwards along the arcs of the inference net.

Changing the value of a node cannot lead to the creation of new variables-- i.e., of new nodes. However, it can lead to the creation or deletion of instances of existing variables. When this occurs, a truth maintenance procedure (adapted to our functional representation) is invoked to assure global consistency over all instances of all variables.

B. Efficiency Issues

A typical knowledge base written in SYNTEL contains thousands of nodes and, since functions in general compute sets of distributions, many thousands of instances. We have therefore been motivated to develop several techniques to enhance propagation efficiency. Together, they implement the strategy "Compute only what you must, and that only once."

Updated instances: Function values are computed only for those instances of its arguments that have changed significantly. Referring to our earlier example, if our economist tells us that the *EconomicOutlook* for the NorthEast region has improved, we would compute a new value for only the variable instance *TargetMarket(NE)*.

Limited Propagation: New function values are computed only if they affect the display seen by the user. A change in the value of any node may, in general, affect the values of many output nodes, only some of which can be displayed at any one time. By limiting propagation to these visible nodes, execution time is proportional to the number of outputs displayed rather than to the total size of the knowledge base. Potential changes to other nodes are recorded in a queue if needed.

*But see Section V for a description of a backward chaining sub-process.

Non-redundant computation: Because an inference net is a graph rather than a tree, because instances of a node can be selected by other nodes, and because propagation can be initiated from several nodes simultaneously, a straightforward propagation algorithm will perform unnecessary recomputation of functions.

Limited propagation and non-redundant computation are supported by an extensive compile-time analysis of the static knowledge base. The analysis identifies the connections between each node in the inference net and each screen that the user may see. It also produces an optimal partial ordering for function evaluation. In these and similar optimizations we have favored run-time efficiency at the expense of compilation cost.

IV USER INTERFACE

It is widely recognized that the user interface consumes a large proportion of application development resources, yet its design is so critical to system acceptance that its development cannot be slighted. Our approach to this ubiquitous problem is to separate the description of the interface cleanly into two parts: one governing the appearance and interactive properties of the display itself, and the other defining the logical relations between display objects and the underlying inference net.

A. Display Description

Our application domains make it natural to adopt the *business form* as the underlying display metaphor, much as ONCOCIN [Hickam, et al., 1985] uses medical forms for its domain. Examples of business forms are insurance applications or sets of financial statements. Figure 1 shows a form concerning the assessment of a building's fire risk.

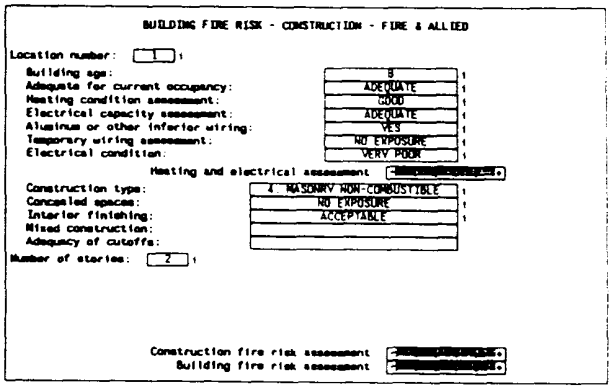

Figure 1: Fire risk at Location 1

Forms are represented as structured objects constructed of rectangular primitive regions called *boxes* and non-primitives called *groups*. Boxes (which need not have visible outlines) are used to accept input data, display several kinds of output assessments, and allocate regions for displaying text and annotations of various kinds. Groups, which can be nested to any depth, are typically used to define larger regions of the display, to define a single full display screen (a "form") and to define sets of forms.

A high-level form description language provides the means for defining and positioning boxes, for composing them into higher-level groups, and for associating various properties with them. These descriptions can be compiled to run on desired target displays.

B. Inference Net Integration

An important design feature of SYNTEL is the close relation between the user interface and the control of propagation in the inference net. Control is accomplished through explicit *links* between variables held in nodes of the inference net and display objects. Among the more important links are the following:

Input data: A link can specify a node which is to receive a user-supplied value. When a user enters a value for a variable (typically by menu-selection), the forward-propagation process is initiated. Since any variable can be selected, the underlying flow of control is effectively in the hands of the user*.

Output and Limited Propagation: A node value can be displayed (in any of several formats) in a linked object, and is updated whenever a change occurs. As mentioned earlier, propagation does not proceed "upward" beyond currently-displayed objects.

Conditional display: In our domains of interest, the complete assessment of a single case can in principle require the user to supply a very large number of individual data items. In practice, fortunately, most cases can be resolved after supplying only a small subset of the possibly-relevant data. A forward-chaining system, however, cannot easily take advantage of this opportunity to spare the user from the need to supply values for, or even to scan, unnecessary items.

Our design approach to this difficulty has been to hide from the user all information or requests for data whose present relevance is problematic. We accomplish this by allowing the display of any object to be conditional on the value of any logical variable. Figure 2 shows the Fire Risk form as it applies to Location 2. The input data requested for this building is rather different from that shown in Figure 1, because the two buildings differ in key aspects such as age, height and construction-type. Thus, while the user controls the flow of inferences, this control can be exercised only within currently-relevant portions of the knowledge base.

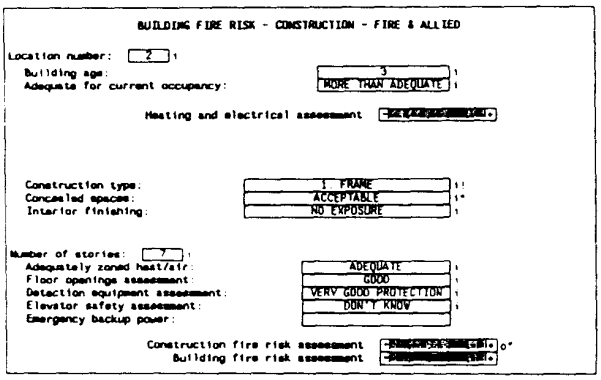

Figure 2: Fire risk at Location 2

* This is analogous to the use of active values in access-oriented programming systems [Stefik, et al., 1986]. In SYNTEL, however, function evaluation is invoked uniformly by the system, rather than by programmer-provided explicit triggers.

The user-interface definition-facility has allowed us to factor knowledge base design into two independent parts. The inference net is based on decision criteria supplied by experts, while the interface is governed by the need to provide a smooth, natural and efficient environment for the end user.

V RUN-TIME CAPABILITIES

The SYNTEL run-time environment contains a number of features to increase the efficiency and confidence of the end user. For reasons of space, we describe just two of them: the backward chaining facility and the explanation facility.

A. Backward Chaining

While we believe that the user should control the interaction, we have nonetheless found it important to indicate which missing data items are especially relevant in the current context. To this end, we have designed a best-first search algorithm based on sensitivity analysis. Working backwards from an identified goal node, the algorithm uses the probability distribution at each daughter node to estimate the amount by which that daughter is likely to change the distribution of the parent. The process continues until input nodes are reached. The input node having the largest likelihood of changing the goal node is indicated to the user, who is free to accept the advice or to supply some other piece of data.

B. Explanations

The ability to explain reasoning is an important feature of expert systems, improving both the efficiency of knowledge engineering and the acceptability by users. However, as representational power has grown beyond that of the earliest systems, it has become increasingly difficult to generate cogent explanations.

Because we wish to minimize the need for knowledge engineers to define significant auxiliary structures, we have had to rule out approaches like those described by [Neches, et al., 1985] and [Smith, et al., 1985]. Instead, we adopted a simpler approach based on refining the standard notion of the *support* for an inference.

In a functional representation such as ours, the support for a variable is simply the arguments of the function that computes its value. We extend this slightly and define *explainable support* to be a subset of the support so designated by the knowledge engineer. Explanations can be then restricted to include only those variables whose meanings are judged to be intuitive and useful to the end user.** A recursive procedure allows the user to explore explainable supports to any desired depth.

VI STATUS AND CONCLUSIONS

SYNTEL has been implemented on a Xerox 11xx in Interlisp-D and in a distributed IBM mainframe/workstation environment in PL/I and C. Several large knowledge bases have been built, with the aid of a well-developed knowledge engineering environment, to address risk assessment problems in commercial insurance and banking.

What has been learned?

**Even though a SYNTEL knowledge base contains no control information--a standard problem for explanation systems--the expert may judge some intermediate variables to represent unintuitive concepts.

The salient feature of SYNTEL is its functional representation of knowledge. Even experienced computer scientists and software engineers require a little time before they are fully comfortable with this purely non-procedural representation. Significantly, however, computer-naive domain experts find it very natural to describe their expertise in these terms. This suggests that we have at least partially achieved our goal of matching the representation to the structure of domain knowledge. This conclusion is further supported by the fact that no knowledge base built to date has had to escape to the underlying implementation language.

We have come to appreciate the benefits to the user of a "soft" mixed-initiative system. Forward chaining provides the basis for high-bandwidth, user-controlled interactions. The features for computing advice and conditionally controlling the display prevent the interactions from becoming unfocused and time-wasting.

A final point: We have tenaciously maintained the purity of the non-procedural architecture, with consequences extending beyond philosophical gratification. It has allowed the system to undergo major enhancements with minimum re-coding.

ACKNOWLEDGEMENTS

SYNTEL has benefited greatly from the work and ideas of many people. We especially wish to acknowledge the contributions of A. Barzilay, R. O. Duda, S. Gower, J. Harris, P. E. Hart, M. Ljungberg, J. Reiter, J. F. Rulifson, and A. White. We also thank D. Bobrow, A. Schiffman and J. Aikens for their constructive comments on earlier versions of this paper.

REFERENCES

Brownston, L., R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley, 1985.

Duda, R. O. and R. Reboh, "AI and Decision Making: The PROSPECTOR Experience," in W. Reitman, Ed., *Artificial Intelligence Applications for Business*, pp 111-147,(Ablex Publishing, Norwood, NJ, 1984).

Gordon, J., and E. H. Shortliffe, "A Method of Managing Evidential Reasoning in a Hierarchical Hypothesis Space," *Artificial Intelligence, 26 (3)*, pp 323-358, (July, 1985).

Heckerman, D., "Probabilistic Interpretations of MYCIN's Certainty Factors," *Proc. Workshop on Uncertainty and Probability in Artificial Intelligence*, (UCLA, 14-16 August, 1985).

Hickam, D.H., E. H. Shortliffe, M. B. Bischoff, A. Carlisle Scott, and C. D. Jacobs, "The Treatment Advice of a Computer-Based Cancer Chemotherapy Protocol Advisor," *Annals of Internal Medicine 103 (6)*, (December 1985).

Lucas, P. and T. Risch, "Representation of Factual Information by Equations and Their Evaluation," *Sixth International Conference on Software Engineering*, (Tokyo, 13-16 September 1982).

Neches, R., W.R. Swartout, and Moore, J., "Explainable (and Maintainable) Expert Systems," *Proc. Ninth International Joint Conference on Artificial Intelligence*, (Los Angeles, 18-23 August 1985).

Pearl, J., "How to Do with Probabilities What People Say You Can't," *Proc. Second Conference on Artificial Intelligence Applications*, IEEE Computer Society, (Miami Beach, 11-13 December, 1985).

Reboh, R., J. Reiter and J. Gashnig, "Development of a Knowledge-Based Interface to a Hydrological Simulation Program", Final Report, SRI Project 3477, (May 1982).

Schor, M.I., "Declarative Knowledge Programming: Better than Procedural?," *IEEE Expert, 1 (1)*, (Spring 1986).

Slagle, J. R., and H. Hamburger, "An Expert System for a Resource Allocation Problem," *Comm. ACM, 28 (9)*, (September, 1985).

Smith, R., H.A. Winston, T.M. Mitchell, and B.G. Buchanan, "Representation and Use of Explicit Justifications for Knowledge Base Refinement," *Proc. Ninth International Joint Conference on Artificial Intelligence*, (Los Angeles, 18-23 August 1985).

Stefik, M. J., D. G. Bobrow, and K. M. Kahn, "Integrating Access-Oriented Programming into a Multiparadigm Environment," *IEEE Software, 3 (10)*, (January, 1986).

Wadge, W. W. and E. A. Ashcroft, *Lucid, the Dataflow Programming Language*, Academic Press, 1985.

Zadeh, L., "Syllogistic Reasoning as a Basis for Combination of Evidence in Expert Systems," *Proc. Ninth International Joint Conference on Artificial Intelligence*, (Los Angeles, 18-23 August 1985).