RUBRIC
An Environment for Full Text Information Retrieval

Richard M. Tong
Victor N. Askman, James F. Cunningham, Carl J. Tollander

Advanced Information & Decision Systems
201 San Antonio Circle, Suite 286, Mountain View, CA 94040.

1. INTRODUCTION

This paper describes an ongoing investigation into the application of ideas from Artificial Intelligence (AI) in the development of a computer-based aid for Information Retrieval (IR). The prototype system, called RUBRIC, is designed to help IR professionals gain easy access to large unformatted full text databases. Knowledge about retrieval requests is encoded in RUBRIC as a collection of rules with attached uncertainty values. This representation provides an appropriately expressive query language that can represent partial relevance and which is easily understood and modified. When coupled with an effective user interface, the rule-based approach can, we believe, give significant improvements over commercially available Boolean keyword systems such as DIALOG, LEXIS, and MEDLARS. At the same time, it avoids the theoretical and computational problems associated with full scale natural language processing of documents (e.g., as proposed by Lebowitz [1]), and the difficulties users have in understanding the mechanisms used in statistical approaches (e.g., Salton's SMART system [2]).

RUBRIC differs in several important ways from traditional approaches, namely: (1) matching is performed over the whole document, (2) documents are given relevance values in the range [0,1], (3) queries are expressed in a language of rules that allows the user to develop hierarchical knowledge structures of retrieval concepts, and (4) users are provided with a collection of tools to help develop and modify queries, and to analyze the retrieval results.

In providing these characteristics RUBRIC makes use of several key ideas from AI. In particular, RUBRIC is an example of a production system that can perform evidential reasoning. In this view, the text of the document is the "evidence" on which the system determines the relevance of that document to the retrieval request. The knowledge on which this judgement rests is embodied in the rules which link retrieval concepts. In contrast to conventional expert systems, the knowledge is entered directly by the user of the system who thereby acts as his or her own expert. In addition, RUBRIC makes use of an object oriented presentation system to provide the necessary flexibility at the user interface.

2. THE RETRIEVAL MODEL

In developing our model we start from the premise that the function of a retrieval system is to select a sub-set of the documents in the database as defined by their *relevance* to the user's query. The inherent imprecision in the concept of relevance requires that this be a *fuzzy* sub-set [3] rather than a classical one. Suppose, then, that the user has a finite set of retrieval concepts, C, of interest:

$$C \triangleq \{c_1, c_2, \cdots, c_M\}$$

and that the database, S, contains a finite number of documents:

$$S \triangleq \{s_1, s_2, \cdots, s_N\}$$

then there is a fuzzy relevance relation, R, from C to S such that:

$$R(m,n) = \textit{the relevance of document } s_n \textit{ to concept } c_m$$

If we now assert that relevance can be quantified as a real number in the interval [0,1], then for any particular concept, c, which is an element of C, we can extract from R a row-tuple, $R^*(c)$, which then defines a fuzzy sub-set of S. This sub-set is the ground truth against

which we wish to measure the performance of our retrieval system. Our goal therefore is to build a system that can generate an R(c) which accurately "estimates" R*(c); with an ideal retrieval system giving R(c) = R*(c) for every c in C

To make our fuzzy set theoretic interpretation of the IR problem operational we need a calculus for the representation and propagation of relevance values. Since we assume that relevance can be represented as a numerical value in the interval [0,1], and that there is an obvious fuzzy set theoretic interpretation of these values, we can draw upon work on many-valued logics [4], and on the use T-norms as models of fuzzy set intersection [5], to help us construct a calculus of relevance values.

The first task is to define a set of operators for conjunction (the *and* connective), and disjunction (the *or* connective). There are many we could choose, but we shall consider four pairs as shown in Table 1. Here $v[A]$ and $v[B]$ denote the relevance values of the primary propositions, with $v[A \textit{ and } B]$ and $v[A \textit{ or } B]$ denoting the relevance value of their conjunction and disjunction respectively. The conjunction operators are T-norms, the disjunction operators are T-conorms, and the negation (the unary operator *not*) is defined by $v[\textit{not } A] = 1 - v[A]$.

The second task is to define a mechanism for performing rule-based inference. In two-valued logic the *modus ponens* syllogism allows B to be inferred from A and $A \Rightarrow B$. In an infinitely-valued logic, we need to extend this idea so that the relevance of B, denoted $v[B]$, can be computed from any given $v[A]$ and $v[A \Rightarrow B]$, where "$\Rightarrow$" is some infinitely-valued implication. Functions that allow us to compute $v[B]$ are called detachment operators (and are denoted *). It is usual to define them so that for a given definition of $\Rightarrow$, $v[A] * v[A \Rightarrow B]$ is a lower bound on the value of $v[B]$. Five of these are shown in Table 2, together with the corresponding implications.

Let us denote a particular calculus by L(i,j) where "i" is an index over the conjunct-disjunct operator pairs and "j" is an index over the detachment operators. Then we see that some of the L(i,j) are well known; in particular, L(3,3) is Lukasiewicz's nondenumerably infinite logic [6], and L(3,0) is a logic proposed by Zadeh [7]. Another calculus of interest is L(2,2), which we can view as a "pseudo-probability" logic in which A and B are independent events.

The way in which RUBRIC generates R(c) is to interpret the rules as a hierarchy of retrieval concepts and sub-concepts. Thus by naming a single concept, the user automatically invokes a goal oriented search of the tree defined by all of the sub-concepts that are used to define that concept. The lowest-level sub-concepts

Table 1 Conjunct-Disjunct Operators

	v(A and B)	v(A or B)
0	T[v(A),v(B)]★	S[v(A),v(B)]★
1	max[0,v(A)+v(B)-1]	min[1,v(A)+v(B)]
2	v(A).v(B)	v(A)+v(B)-v(A).v(B)
3	min[v(A),v(B)]	max[v(A),v(B)]

★T[1,1] = 1, T[x,1] = T[1,x] = x,
T[x,y] = 0 ∀x,y ∈ [0,1)

★S[0,0] = 0, S[x,0] = S[0,x] = x,
S[x,y] = 1 ∀x,y ∈ (0,1]

Table 2 Detachment and Implication Operators

	Detachment (*)	Implication (=>)
0	v(B) = min[v(A),v(A=>B)]	min[v(A),v(B)]
1	v(B) = min[v(A),v(A=>B)] if v(A)+v(A=>B) > 1 = 0 otherwise	max[1-v(A),v(B)]
2	v(B) = v(A).v(A=>B)	min[1,v(B)/v(A)]
3	v(B) = max[0,v(A)+v(A=>B)-1)]	min[1,1-v(A)+v(B)]
4	v(B) = max[0,(v(A)+v(A=>B)-1)/v(A))]	1-v(A)+v(A).v(B)

are themselves further defined in terms of pattern expressions in a text reference language which allows keywords, positional contexts, and simple syntactic and semantic notions. The relevance values attached to each rule then provide, together with an appropriate calculus of relevance values, a mechanism for determining the overall relevance of a given document as a function of those patterns which it contains.

3. THE RULE LANGUAGE

RUBRIC is a system that uses a rule-based reasoning process and in this section we describe the nature of a rule and its constituent parts. In Figure 1 we indicate the most general form of rule that can exist within the system. Its two parts consist of the primary inference which links the primary antecedent to the consequent concept, and the secondary inference which describes how the auxiliary antecedent modifies the primary inference. The motivating idea behind the secondary inference is that there are cases in which the existence of additional evidence would cause us to

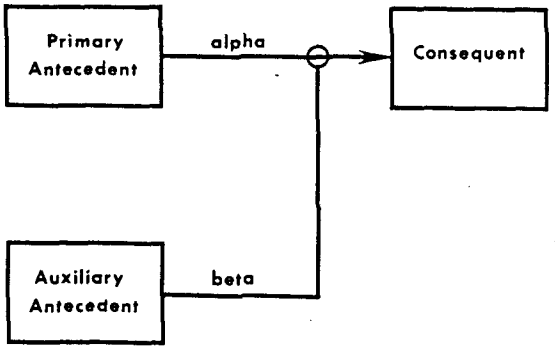

Figure 1 The General RUBRIC Rule

modify our original inference, with the proviso that this auxiliary evidence is by itself of no direct interest. Our new rule form models the effect of such evidence by changing the weight attached to the primary inference.

Formally we model this as:

$$v[consequent] = v[primary\_antecedent] * v[rule]$$

with

$$v[rule] = \{\alpha + (\beta - \alpha) \times v[auxiliary\_antecedent]\}$$

where alpha and beta are the relevance values associated with the primary antecedent and the auxiliary antecedent respectively, * denotes an appropriate detachment operator, and $v[\cdot]$ denotes the relevance value of a variable. Notice that we have chosen to model the effect of the auxiliary antecedent by a simple linear interpolation function. Given our current understanding of the impact of this rule form it seems to be an appropriate choice. Notice too that we allow at most one secondary antecedent. In the future may want to allow multiple secondary antecedents and will then have to define mechanisms to deal with conflict resolution.

3.1. Rule Types in RUBRIC

We provide a variety of rule types in RUBRIC. They have similar syntactic and functional forms but their semantics are intended to capture the different types of inferential relations that can exist between retrieval concepts. That is, we provide a rule language that allows the user of RUBRIC to express the required relationships between the concepts of interest. Since the rules carry semantic information they can be used to help perform more efficient searches of rule trees.

We briefly discuss the five rule types and then consider the elements used in constructing antecedent expressions:

The IMPLIES rule. This is the principal rule type implemented in RUBRIC. It is intended to link retrieval concepts and then be invoked in a generalized *modus ponens* inference. That is:

$$v[b] = v[a] * v[IMPLIES(a,b)]$$

where * is the appropriate detachment operator. Note that $v[IMPLIES(a,b)]$ is given as part of the rule definition (i.e., it depends on the values of alpha and beta and is given by the expression in braces in the definition above) whereas $v[a]$ is derived by the system from the application of other rules.

The EVIDENCE rule. This rule type is used to link text references to concepts. It is intended to capture the notion that text expressions are used as direct "evidence" in determining the relevance of the document to the retrieval topic. Functionally, this rule is similar to IMPLIES but we want to distinguish between inferences made using EVIDENCE and IMPLIES so as to provide better control of search. We have:

$$v[b] = v["a"] * v[EVIDENCE("a", b)]$$

where * is a detachment operator, and "*a*" is a text reference expression.

The SUBSET rule. This rule type allows us to express the relationship between a sub-set of a set and the set itself. It seems that these rules perform no modification of relevance values, but the reason we introduce them is to allow ourselves to take account of the length of the reasoning chain used to establish the relevance of a document.

The INSTANCE rule. A rule that allows us to express the relationship between an element of a set and the set itself. As with the SUBSET rule, there seems to be no need to modify relevance values.

The ATTRIBUTE rule. This rule is intended to capture the idea that concepts have components (or attributes), and that knowledge of these components may be used to help establish the presence of the concept itself.

3.2. Antecedent Operators in RUBRIC

These operators are the primitives used in developing the primary and secondary antecedent expressions. There are three main classes: (1) those which take concepts and text as arguments (i.e., the "logical" operators), (2) those which take only text (i.e., the "distance" and "boolean" operators), and (3) those which perform miscellaneous functions (i.e., the "scope" operators, the "proximity" operator and the "macro" function).

The Logical Operators. These operators take either concepts or text, or both, as their arguments, which themselves can be arbitrarily complex expressions using other antecedent operators. We allow generalized (i.e., multi-valued) forms of AND, OR, NOT, as well as two non-traditional operators BEST-OF and WEIGHT-OF which capture the idea that (1) any one of the arguments would be appropriate so we might as well take the best, and (2) the more arguments that are true the better.

The Distance Operators. These operators take a pair of text arguments and return a value which represents the distance between them. Currently we have implemented three fuzzy operators, and two boolean operators that also double as scope operators. The NEAR_W, NEAR_S and NEAR_P operators all return a value in the interval [0,1] which is a normalized measure of the distance in words, sentences or paragraphs between its arguments. The SENTENCE and PARAGRAPH operators each take a pair of keywords as arguments and tests to see if they occur within the same sentence or paragraph in the document.

The Boolean Operators. These operators take only text as their arguments and return a value from the set {0,1}. The PRECEDES operator takes two keyword arguments and tests whether one occurs before the other. The WITHIN operator takes two keyword arguments and tests whether they are within some distance (in words) of one another. The PHRASE operator takes multiple text arguments and tests whether the phrase defined by concatenating the keywords occurs within the document.

The Scope Operators. In their most general forms these operators are somewhat problematic. Conceptually they are straightforward, but their implementation is complicated. The SENTENCE and PARAGRAPH operators mentioned above are degenerate examples and are more conveniently thought of as distance operators with discontinuous functional forms. Scope operators take only one argument and their intended effect is to reduce the scope of the pattern matching to the scope unit indicated. Notice that there is an implied default scope unit of "document" if no scope operator appears. Obviously, there are some constraints on the way these operators can be nested. We allow scoping using two functions. The *SENTENCE* operator reduces the scope of the pattern matching to a single sentence. The argument can be any expression of antecedent operators and concepts and text. Similarly, the *PARAGRAPH* operator reduces the scope of the pattern matching to a single paragraph.

The Proximity Operator. This operator allows the user to take account of the "nearness" of concepts within a document. This is still an experimental feature and we are exploring the semantics of concept location together with appropriate distance measures.

The Macro Function. This is a feature that allows the rule writer to enter a special synonym symbol in any place where a text string could appear. When RUBRIC encounters such a symbol it recognizes it as a "place holder" for a set of synonymous text strings and expands the language expression accordingly.

3.3. Aggregation Functions

These functions determine how we will combine the inferred relevance values from multiple rules having the same retrieval concept as their consequent. In the current implementation we have an implied OR (i.e., a disjunction of the evidence), although the AGGREGATION function can be specified independently of any choices we make for the other operators. We also provide for alternative AGGREGATION functions, to be implemented as we consider the effects on nodes with multiple types of rules.

4. THE USER ENVIRONMENT

The target machine for RUBRIC is a professional workstation, such as a SUN, with high resolution bit-mapped display capabilities. In order to exploit the graphics facilities on such machines we have designed the user interface for RUBRIC around a multi-purpose interface system called MPS [8]. MPS utilizes object oriented descriptions of the information to be exchanged between the user and RUBRIC, thus allowing a clean demarcation between the functions of RUBRIC and those of the user interface. In addition to data about specific objects to be displayed, MPS maintains generic descriptions of the contents of presentation surfaces, so employing a high-level semantic model of the objects to be displayed. The MPS interface uses a relational database as the medium of communication, and since the semantic model is stored directly in this relational database, it is available to both RUBRIC and MPS. An important benefit of the cleanly defined interactions between RUBRIC and MPS is that RUBRIC is freed from the details of handling numerous user interface devices.

The current version of the interface supports a menu driven style of interaction on conventional alphanumeric terminals. Selection of menu items is done by positioning a cursor via function keys, input of information is done through displayed forms, and presentation of data is done by means of non-overlapping windows. An example of a RUBRIC information entry form is shown in Figure 2. This is the new rule template form. The user can call it from a menu and is then expected to enter the appropriate

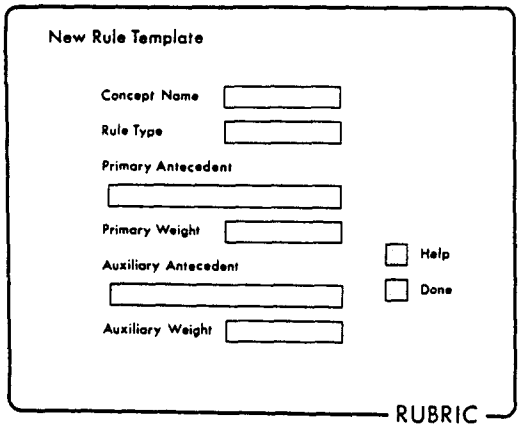

Figure 2 New Rule Template Form

information in the outlined areas. A screen editor with built-in syntax checking is provided. Notice the provision of a sub-menu within this form; the user can ask for help if necessary, and is provided with a mechanism for exiting when he or she is satisfied with the new rule.

Using MPS allows us to provide effective interaction mechanisms so that the iterative and incremental nature of query building can be properly supported. Our view is that the user first constructs an initial query by drawing upon existing knowledge in the rule-base and perhaps adding small amounts of additional knowledge. Having done that, he or she checks over the query for obvious flaws and errors, and then applies it to a document database. Most probably, the initial test of the new query will be on a small database of documents with which the user is familiar. Using the results of the test as a guide the user will make appropriate changes to the query and repeat the cycle. When the performance is satisfactory, the query will be applied to the main document database(s). If the query represents a retrieval concept that the user is likely to want to use again, then it can be entered into the permanent rule-base.

Given that query building passes through these stages then we can see the need for several categories of tools to support query development. These tools are really not tools for knowledge elicitation *per se*, but rather tools that will provide an "environment" in which the user can develop queries easily, get useful feedback about their performance and quickly make changes if this performance is not satisfactory. By analogy, we think of this "toolbox" as a collection of tools that either singly or in combination can help the user perform some activity.

Let us consider the tools that we provide in a little more detail:

Query Construction Tools. These tools are needed to help the user develop and edit new queries. They include mechanisms for describing the retrieval concept in terms of the knowledge in RUBRIC's existing rule-base(s), an editor and a syntax checker.

Rule-Base Access Tools. The purpose of these tools is to allow the user to explore the rule-base. For example, a user who is about to develop a new query many want to browse the rule-base to see if similar retrieval concepts already exist or to examine sub-concepts that might form a basis for the new extended query.

Static Check Tools. These tools might be invoked either by the user or by the system to check that a newly create query is "consistent." For example, such tools check that there are no circular paths in the reasoning, that the query can indeed return a relevance value significantly different from zero, etc.

Performance Analysis Tools. The user needs to be able to examine the results of the retrieval request in a variety of ways. A minimum requirement is for the results to be displayed in graphical as well tabular form. The user will also certainly want to be able to compute a variety of performance measures based on our fuzzified notions of precision and recall. He or she will also need some mechanism for storing performance results so that subsequent modifications to the query can be compared.

Diagnosis Tools. We expect that a large part of the knowledge elicitation process will be concerned with trying to understand *why* the query performed the way it did. To support this type of activity we provide a class of tools that allow the user to do things such as single-step the query, explore the sensitivity of the query to changes in its structure, generate traces of rule invocation, create artificial documents for checking sub-parts of the query and observe bottom-up propagation of user induced triggering of rules.

Help. Finally, we provide a generalized help facility within RUBRIC. At any stage in the process the user should be able to ask for on-line help that would explain the general features of the RUBRIC system, the purpose of a tool, or the nature of the response required at a decision point.

5. SYSTEM IMPLEMENTATION

RUBRIC is currently implemented in a Berkeley UNIX[1]/VAX 11-780 environment. The implementation is divided into two major modules: the preprocessor module and the system module. The preprocessor module is written in the C Language and takes as input the free format text of a collection of documents and builds the RUBRIC-readable database for that collection of documents. The primary component of this database is an inverted structure on the words (actually, word stems) occurring in the collection of documents. Each word has one entry in the structure and is accompanied by various contextual information (such as in which document(s) and in which position(s) it occurs). The system module, which includes the user interface, the toolbox and the retrieval sub-systems, is currently implemented in both Franz LISP and C. In general, the lower level word matching/database access functions are implemented in C, while the higher level query expansion/tree traversal functions are implemented in Franz LISP.

6. RETRIEVAL PERFORMANCE

We have performed a variety of experiments with RUBRIC to assess its effectiveness. These include tests of the impact of using different uncertainty calculi and restrictions on the use of the rule language [9], as well as timing and sizing tests. However, to illustrate our methods we will describe some experiments that were designed to assess the improvements that can be achieved over a conventional Boolean keyword approach.

As an experimental database for testing the retrieval properties of RUBRIC, we have used a selection of thirty documents taken from the Reuters News Service. Our basic experimental procedure is to rate the documents in the database by inspection (i.e., define the relevance relation row-tuple $R^*(c)$), construct a rule-based representation of a typical query, apply the query to the database, and then compare the rating, $R(c)$, produced by RUBRIC with the *a priori* rating $R^*(c)$.

Given our fuzzy set interpretation of the IR problem, there are a large number of possible measures of performance that we could employ. For this presentation we concentrate on just two. Both of these are based on the idea of using a selection threshold to partition the ordered documents so that those above it are "relevant" (either fully or marginally) and those below it are "non-relevant." In the first we lower the threshold until we include all those deemed *a priori* relevant, and then count the number of unwanted documents that are also selected (denoted N_F). In the second we raise the threshold until we exclude all irrelevant documents, and then count the number of relevant ones that are not selected (denoted N_M). The first definition therefore gives us an insight into the system's ability to reject unwanted documents (precision), whereas the second gives us insight into the system's ability to select relevant documents (recall).

We selected as a retrieval concept "Violent Acts of Terrorism," and then constructed an appropriate rule-based query. This is summarized in tree form in Figure 3, where we make extensive use of the extended rule form described above. An auxiliary_antecedent is shown linked to a primary inference by a horizontal directed arc. Application of this query to the document database with calculus L(3,2) (i.e., one that models conjunction/disjunction as min/max and detachment as product), results in the document profile shown in Figure 4. (Notice that for presentation purposes the relevance scores have been re-normalized and the documents ordered such that those determined to be *a priori* relevant are to the left in Figure 4.) This is excellent performance of course; the relevant and non-relevant documents being correctly partioned into two disjoint sets. (e.g., setting the selection threshold at 0.3 would make N_F and N_M simultaneously zero.)

To compare RUBRIC against a more conventional approach, we constructed two Boolean queries by using the rule-based paradigm and setting all rule weights to 1.0. (Thus showing, incidentally, that our method subsumes Boolean retrieval as a special case.) One of these queries is shown in Figure 5 as an AND/OR tree of sub-concepts. The only difference between the two Boolean queries is that in the first we insist on the conjunction of ACTOR and TERRORIST-EVENT (as shown), whereas in the second we require the disjunction of these concepts. Running each of these queries against the thirty document Reuters database produces a non-fuzzy subset of documents. Performance is then assessed in the conventional way with *recall* computed as the ratio of the number of relevant documents retrieved to the total number of relevant documents in the database, and *precision* computed as the ratio of the number of relevant documents retrieved to the total number retrieved. To get an equivalent RUBRIC score we construct a non-fuzzy set from $R(c)$ by setting the relevance threshold and then marking as " retrieved" all those documents with higher relevance values, and "not-retrieved" all those with lower values. The conjunctive form of the Boolean query misses five relevant documents and selects one non-relevant document, giving:

$$Precision = .89 \qquad Recall = .62$$

[1] UNIX is a Trademark of AT&T Laboratories

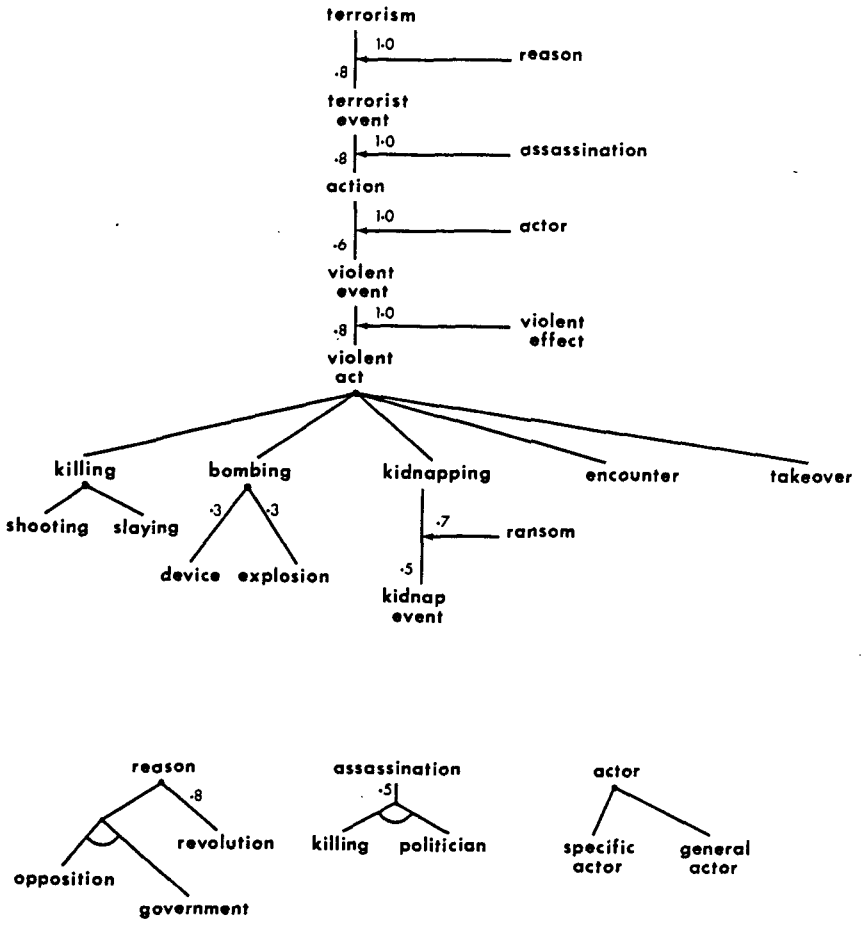

Figure 3 Example RUBRIC Query Tree

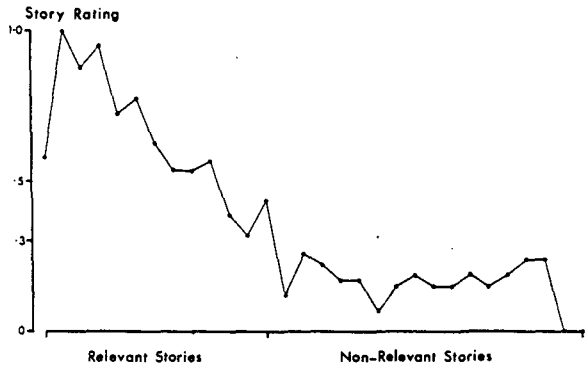

Figure 4 RUBRIC Retrieval Profile

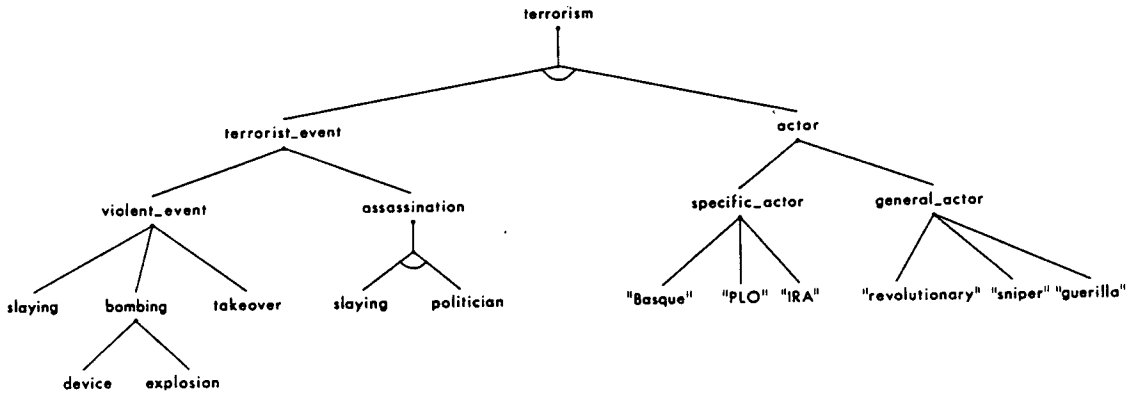

Figure 5 Comparative Boolean Query Tree

The disjunctive form selects all the relevant documents, but at the cost of also selecting seven of the non-relevant ones, giving:

$$Precision = .65 \qquad Recall = 1.0$$

However, if we select the relevance threshold to be 0.3, then the RUBRIC retrieval gives:

$$Precision = 1.0 \qquad Recall = 1.0$$

While these results represent only a partial test, we believe that they indicate that the RUBRIC approach allows the user to be more flexible in the specification of his or her query, thereby increasing both precision and recall. A traditional Boolean query tends either to over-constrain or under-constrain the search procedure, giving poor recall or poor precision. We feel that, given equal amounts of effort, RUBRIC allows better models of human retrieval judgment than can be achieved with traditional Boolean mechanisms.

7. SUMMARY

In this paper we have attempted to give an overview description of RUBRIC and the ideas on which it is based. Although it is still a research prototype we believe it shows considerable promise as an advanced IR system.

We believe that the major contributions of RUBRIC are that it encourages proper structuring of queries leading to more effective and better understood retrievals. Given equal amounts of effort, RUBRIC can give improved precision and recall when compared to conventional systems. Further, the provision of an advanced interface and toolbox gives the user an environment in which the IR task can be performed quickly and effectively. Finally, because of its inherent modularity, RUBRIC is an excellent vehicle for exploring a wide range of related research issues such as the problem of the representation and manipulation of uncertainty, the development of user models based on training and performance experiments, and the adequacy of various presentation and input formats.

8. ACKNOWLEDGEMENTS

As with all systems developed in a co-operative environment, RUBRIC has benefitted from extended discussions with other members of the AI&DS technical staff. We would like to acknowledge recent contributions from Sonia Schwartzberg, Dan Shapiro, Brian McCune and Gerry Wilson.

9. REFERENCES

[1] Lebowitz, M. (1983) Intelligent Information Systems. *Proc. 6th Int. ACM-SIGIR Conf. on R&D in Information Retrieval.* Bethesda, MD.

[2] Salton, G. (1971) *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice-Hall Inc., Englewood Cliffs, NJ.

[3] Zadeh, L.A. (1965) Fuzzy Sets. *Information and Control*, 8:338-353.

[4] Rescher, N. (1969) *Many Valued Logic*, McGraw-Hill, New York.

[5] Dubois, H., Prade, H. (1982) A Class of Fuzzy Measures Based on Triangular Norms. *Int. J. General Systems*, 8:43-61.

[6] Lukasiewicz, J. (1930) Many-valued Systems of Propositional Logic. In S. McCall, *Polish Logic*, O.U.P, 1957.

[7] Zadeh L.A. (1973) Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Trans. Systems, Man and Cybernetics*, SMC-3:28-44.

[8] Wilson, G.A., Domeshek, E.A., Drascher, E.L., Dean, J.S. (1983) The Multipurpose Presentation System. *Proc. 9th Int. Conf. on Very Large Data Bases*. Florence, Italy.

[9] Tong, R.M., Shapiro, D.G. (1985) Experimental Investigations of Uncertainty in a Rule-Based System for Information Retrieval. *Int. J. Man-Machine Studies*. (to appear).