# TOWARDS AN EXPERT SYSTEM FOR BIBLIOGRAPHIC RETRIEVAL: A PROLOG PROTOTYPE

C.R. Watters and M.A. Shepherd
Dept. of Mathematics, Statistics, and Computing Science
Dalhousie University
Halifax, Nova Scotia, Canada B3H 3J5

W. Robertson
School of Computer Science
Technical University of Nova Scotia
Halifax, Nova Scotia, Canada B3J 2X4

## ABSTRACT

A prototype Prolog system has been developed for online bibliographic retrieval. Most online bibliographic retrieval systems may be characterized by queries based on the occurrence of keywords and by databases consisting of possibly millions of records. Such systems have very fast response times but generally lack any deductive reasoning capability.

An expert system for online bibliographic retrieval, developed in Prolog, would provide enhanced retrieval capabilities through the application of deductive reasoning. Such a system would permit knowledge-type queries to be asked in addition to the traditional keyword-type of queries.

A concern with using Prolog to perform an online search of a million-record data base is that the response time would be unacceptable. In order to overcome this drawback two alternatives are examined: a special-purpose hardware device and an extended Prolog capability.

## 1. INTRODUCTION

A project has been undertaken to develop a bibliographic retrieval system based on predicate calculus using Prolog as both programming language and data language. A Prolog-based system would provide a deductive reasoning capability not normally available in bibliographic retrieval systems. The project is planned as four stages: prototype retrieval natural language interface, techniques to handle large databases, and study of faceted indexing schemes as the basic construct of the knowledge base for such a system.



Traditional online bibliographic systems tend to be based on either the Boolean or the vector-space model of retrieval (Salton and McGill, 1983) and may access databases containing millions of documents. In both of these models, the documents are retrieved by matching terms of the query with keywords occurring in the records of the database.

In order to provide fast response time, bibliographic retrieval systems are usually based on inverted files, although they can be implemented through a DBMS (Crawford, 1981; Macleod and Crawford, 1983; Shepherd and Watters, 1985a). As these systems are based on the occurrence of keywords and not on the relationships between concepts, the systems lack any deductive reasoning capability.

A Prolog-based system may provide the deductive reasoning capability required of an expert bibliographic retrieval system. An example of such a system is TUGA (Coelho, 1982) in which the database consists of documents in the area of Artificial Intelligence and each document is classified according to an Artificial Intelligence system of categories. TUGA permits the user to retrieve information about the document collection and about the classification system. The database, however, consisted of only 46 documents each indexed according to an a priori classification scheme.

The applicability of predicate calculus for the implementation or modelling of the domain of information retrieval has been discussed elsewhere (Watters et al, 1986). Although it may be an appropriate medium for the manipulation of bibliographic information, two difficulties arise: the definition of the knowledge base for deduction, and online response with large bibliographic databases.

The knowledge base for use in a deductive system must combine both the data structures for containing the concepts covered in a given database and the rules for manipulation of that knowledge (Rich, 1983). A knowledge base must reflect the contents of a given database and the structure of each knowledge base must be able to expand and accept new relationships as the database expands.

A concern with a predicate calculus approach to retrieval is the speed of the depth-first search strategy used in Prolog implementations. It is likely that solutions for providing adequate access times can be found using either hardware or software techniques. First a special-purpose hardware device is proposed that, given an initial query, scans the disk-based Prolog database and returns to the main computer a subset of predicates relevant to the query. This set of predicates may then be used to resolve the user query in the normal Prolog manner. A software solution is also proposed to reduce the size of the problem space of any given Prolog search by introducing a function that produces a reduced problem space (set of predicates) by using variables stored in predicates to select records from a direct access file where the records of such a direct access file contain predicates.

The final part of the current study is to study the applicability of indexing or classification schemes for encapsulating the concept information required in such a knowledge base.

## 2. PHASE ONE - DEDUCTIVE MODEL FOR RETRIEVAL

A prototype expert retrieval system has been developed in Prolog to illustrate the feasibility and range of capabilities that can be provided in a system based on predicate calculus. An expert bibliographic retrieval system should be able to expand both the query formats available to the user and the capabilities of responses from the system to the user. In addition to Boolean combinations of keywords retrieving copies of database records, the prototype system provides responses to a variety of information needs that include the following:

(a) What journals should I be reviewing?

(b) In what journals should I publish this article?

(c) Who is active in my research area?

(d) What areas are currently active in programming languages?

(e) What are the sub-areas of research in A.I.?

The prototype, PROBIB-2, was written entirely in Prolog, which is an implementation of predicate calculus. Prolog has been used both as the programming language and as the data storage language.

### 2.1. Prototype Database.

The database contains bibliographic descriptions of items that are represented by document predicates. The knowledge base contains two subcomponents: subject contents of items and concept inter-relationships, and user profiles. These subcomponents are represented by the subject and user predicates.

The documents in the database are represented by the predicates defined as:

Doc(<docnum>, <title>, <author>, <source>, <date>, <subject>)

where

- <docnum> is a constant that uniquely identifies the document
- <title> is a list of terms representing the title
- <author> is a list of terms representing the author(s)
- <source> is a constant that identifies the journal/proceedings/monograph where the item can be located
- <date> is a constant that identifies the month/year of publication
- <subject> is a list of terms and relationships representing the subject content of the item.

The subject predicates define the terms and the relationships between the terms used in classifying or describing the subject content of the documents represented in the database. The subject predicates, taken all together, make up a faceted classification that represents the content of the database. In the prototype only one relationship, the hierarchical relationship, was used. The subject predicate, Subject, has been defined as:

Subject(<major concept>, <subconcept>)

where

- <major concept> is a term that identifies a single subject concept
- <subconcept> is a list of terms that identify those subject facets occurring in the database that are directly subordinate to the major concept.

Each user profile is represented by one or more predicates of the form:

User(<user-name>, <subject>)

where

- <user-name> is a constant that uniquely identifies a user
- <subject> is a list of terms and relationships that represents the area(s) of subject interest of a user.

### 2.2. Prototype Retrieval

Retrieval is performed by application of deduction rules to the database components. Prolog searches for solutions to query statements by means of a mechanical theorem proving technique called resolution. The predicates used in Prolog are all Horn clauses, where a Horn clause has the form

A :- B,C,D

which can be read as

IF (B and C and D) THEN A.

A query is a Horn clause with an empty left side of the form

? :- E,F,G

which can be read as

Find x such that E and F and G are all true.

The retrieval functions are defined as Horn clauses and instantiated with values from the query and database to process a user's query. An example of a deduction rule for retrieval is

```
Interest(_user, _item) :- User(_user, _about),
                          Doc(_item, _about).
```

where _x indicates an uninstantiated variable. This rule can be read as

IF (a user is interested in topic x)
AND (an item is about topic x)
THEN (the user is interested in the item).

The prototype identifies the user through a password and accesses a previously established user profile of interests of that user. Any additional subject areas may be introduced by the user for the current session.

The user interacts with the system using a domain-specific natural language interface, described below.

For example a user may ask:

> What articles were printed after 1984?
....
> Where can I get them?

The retrieval process at all times uses information related to the user's search interest and to the history of the current search session. In this way requests can be nested and dialog can be less repetitive. As an example of nested searching consider the following search segment for user called 'John' whose user profile indicates an interest in 'information retrieval':

```
Probib >John are you still interested in
        information retrieval?
User   >yes
User   >find me references on logic.
       ...
User   >get me those on prolog
       ...
User   >which of these are before 1982.
```

In this search segment each request narrows the reference space starting with the space defined by the user's profile as follows:

```
information retrieval
information retrieval - logic
information retrieval - logic - prolog
information retrieval - logic - prolog - before 1982.
```

In order to allow the user to follow an iterative (i.e., nested) search dialog, sets of temporary predicates satisfying the current goal are created. These temporary sets of predicates can then be used in satisfying further goals as requested by the user.

## 3. PHASE TWO - NATURAL LANGUAGE INTERFACE

The second phase of the prototype development was to incorporate a natural language interface between the user and the retrieval system. The natural language used is a subset of English restricted to the domain of bibliographic querying.

The parser, acting on a context-free grammar, was written in Prolog following the outline described by Clocksin and Mellish (1984). The parser searches for a complete sentence from the input, where acceptable sentences are defined by the grammar shown in Figure 1. Figure 2 presents sample sentences defined by this grammar. Each of these sentence structure definitions forms a path that the parser follows in trying to match the input words. During execution the parser follows one path at a time, with backtracking, trying to exhaust the list of input words. Figure 3 is an example of a parse tree for an acceptable sentence. The range of sentence structures and breadth of vocabulary can both be easily expanded.

The parser returns a set of goals to the retrieval module, as shown in Figure 3. The parser instantiates these goal predicates that are used by Prolog to process the query.

## 4. PHASE THREE - HANDLING LARGE DATABASES

The prototype retrieval system described in this report has a very small database of 40 references and 10 user profiles. As long as the predicates can be kept in memory response time is satisfactory. When predicates are stored in a file, however, and accessed only as needed, the response time degenerates greatly. Since bibliographic databases can be expected to be very large other access methods must be considered to provide acceptable response times.

### 4.1. Hardware Approach

As an alternative to building a Prolog machine, a special-purpose hardware device is proposed that, given an initial query, scans the disk-based Prolog database and returns to the main computer a subset of predicates relevant to the query and/or the user profile.

The Special-Purpose Prolog Controller (SPPC) has been designed and simulated to operate at the word serial transfer rate of a disk controller. The device is synchronized by the Direct Memory Access (DMA) Controller of the host computer. This requires a specific format, described in Appendix A, for the records on the disk. It is assumed that the Prolog interpreter (or compiler) conforms to this format. The matching discussed in this paper is for the presence of instantiated variables in specific field positions of the predicates.

The SPPC scans the entire database and returns a set of predicates related to the query. This set of predicates may then be used to resolve the user query in the normal Prolog manner. This model is similar to that developed by Watters (1986) to perform a fast sequential scan of very large but unindexed bibliographic databases. The result of the fast hardware scan was an index to the set of potentially relevant documents upon which traditional online retrieval could then be performed.

Figure 1. Partial Grammar Definition

| Rule No. | Rule Name | Definition |
|---|---|---|
| 1 | sentence | :- single-word commands. |
| 2 | sentence | :- verbphrase,nounphrase. |
| 3 | sentence | :- verbphrase. |
| 4 | sentence | :- nounphrase,verbphrase. |
| 5 | sentence | :- catch-all,error. (illegal sentences) |
| 6 | verbphrase | :- $verb,$adverb,nounphrase,verbphrase. |
| 7 | verbphrase | :- $verb,$adverb,$noun,nounphrase,verbphrase. |
| 8 | verbphrase | :- $verb,$adverb,$noun,nounphrase. |
| 9 | verbphrase | :- $verb,$adverb,nounphrase. |
| 10 | verbphrase | :- $verb,$adverb. |
| 11 | nounphrase | :- $adjective,$noun,preposition-phrase. |
| 12 | nounphrase | :- $determiner,$adjective,$noun,preposition-phrase. |
| 13 | nounphrase | :- preposition-phrase. |
| 14 | nounphrase | :- $determiner,$adjective,$noun. |
| 15 | nounphrase | :- $adjective,$noun. |
| 16 | preposition-phrase | :- prep-and-sub. |
| 17 | prep-and-sub | :- $preposition,$subject,prep-and-sub. |
| 18 | prep-and-sub | :- $preposition,$subject. |
| 19 | prep-and-sub | :- $preposition,special-phrase. |
| | Terminals (preceeded by $ above) | |
| 20 | special-phrase | :- [...phrase...]. |
| 21 | noun | :- journals,proceedings, etc. |
| 22 | verb | :- have,wrote,published,etc. |
| 23 | preposition | :- after,on,before,during,etc. |
| 24 | subject | :- logic, bibliographic,full-text,recursion,etc. |
| 25 | adjective | :- full,complete,etc. |
| 26 | determiner | :- the,this,a,etc. |
| 27 | adverb | :- again,etc. |

4.1.1. Hardware Design

The SPPC has been designed to operate at the word serial transfer rate of a disk controller. The device is synchronized by the Direct Memory Access (DMA) Controller of the computer with which it is designed to co-operate. This approach requires that the memory address normally supplied by the DMA Controller be superceded by the SPPC to eliminate unwanted records by overwriting them in the main memory buffer. It is assumed that a complete file is normally transferred to the memory buffer under DMA control, and that a 16 bit word is available from the disk when the DMA controller makes a request for a CPU cycle. The SPCC operation is synchronized to the DMA cycle request and picks up each word transferred to memory to check it against the appropriate word in the SPPC buffer. A block schematic of the SPPC is shown in Figure 4.

The SPPC operation can be described by the following "hardware" pseudocode, which can be converted into a circuit implementation. In this pseudocode, each bracketed number is associated with a state of the algorithmic machine (ASM). As the ASM must check each word before the next word becomes available the clock of the ASM is synchronized to the disk bit rate. The variables upon which decisions are made in the pseudocode

remain stable during a state. An Immediate action (I) is completed before the end of the state in which it is invoked; any new data indirectly controlled by such an action, is therefore, available for decision making in the subsequent state.

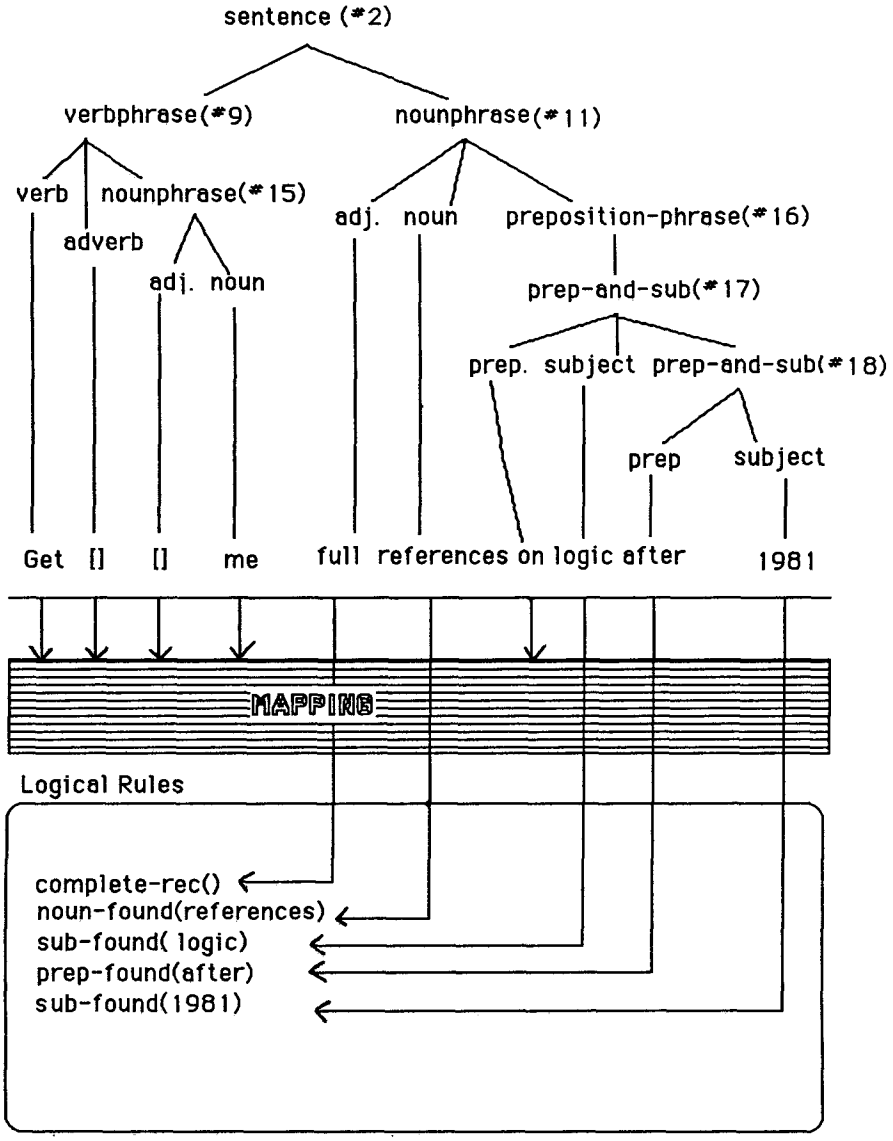
```
(1)   IF NOT start THEN (1)
      ELSE  Reset Buffer-Pointer(I) : (2);
(2)   IF DMA-Request THEN
           Latch-Word {Word consists of Hi-Byte,Lo-Byte}:
           Latch Memory-Address(I):
           (3) {To check SOR/EOD }
      ELSE (2);
(3)   IF (Lo-Byte=SOR) OR (Hi-Byte=SOR) THEN
           (4) {To check Word tokens against Buffer-Token}
      ELSE
           IF (Lo-Byte=EOD) OR (Hi-Byte=EOD) THEN
                (1) {To wait for next initiation}
           ELSE (2);
(4)   {Check Word tokens against Buffer-Token}
      IF (Buffer-Token=IGNF) THEN
           INC Buffer-pointer(I):
           (5) {Look at Word for EOF to match IGNF EOF}
      ELSE
           IF (Lo-Byte=SOR) THEN
                (6) {To look for match or mismatch at Lo-Byte}
           ELSE
                IF (Hi-Byte=SOR) THEN
                     (11) {To check for match or mismatch at Hi-Byte}
                ELSE
                     (6);
(5)   {Look at Word for EOF to match IGNF EOF}
      IF NOT (Lo-Byte=EOF) THEN
           IF NOT (Hi-Byte=EOF) THEN
                (7) {To continue search for EOF at next word}
           ELSE {Hi-Byte was EOF: Continue scan at next word}
                INC Buffer-Pointer(I):
                (8) {To go to next field at next word}
      ELSE
           {Lo-Byte was EOF: Next byte to check is in Hi-Byte}
           INC Buffer-Pointer(I):
           (9) {Check token in Hi-Byte after IGNF EOF};
(6)   {Look for match or mismatch at Lo-Byte}
      IF NOT (buffer-token=Lo-Byte) THEN {Mismatch: look for EOR}
           Reset Buffer-Pointer(I):
           (13) {Check Hi-Byte for an EOR}
      ELSE
           {Match at Lo-Byte must be match to date}
           INC Buffer-Pointer(I):
           (11) {Check Hi-Byte for a match};
(7)   {Search Word for EOF to match IGNF EOF}
      IF DMA-Request THEN
           Latch-Word:
           (5) {Check Lo-Byte and Hi-Byte for EOF}
      ELSE (7);
(8)   {Continue scan at next field starting at next Word}
      IF DMA-Request THEN
           Latch-Word:
           (4) {To start check of a new field}
      ELSE
           (8);
(9)   {Check token in Hi-Byte after IGNF EOF}
      IF (Buffer-token=IGNF) THEN
           INC Buffer-Pointer(I):
           (12) {Check Hi-Byte for EOF to match IGNF EOF}
                ELSE
           (11) {Check Hi-Byte for a match};
(10)  {Look for an EOR,after a mismatch, starting at next Word}
      IF DMA-Request THEN
           Latch-Word:
           (13) {Check Hi-Byte for an EOR}
      ELSE
```

```
          (10);
(11)   {Check token in Hi-Byte for a match}
       IF NOT(buffer-token=Hi-Byte) THEN {Mismatch at Hi-Byte}
            Reset Buffer-Pointer(I):
            (13) {Check Hi-Byte for an EOR}
       ELSE
            IF (Buffer-token=EOR) AND (Hi-Byte=EOR) THEN
                 Pass-Record-Along: Reset Buffer-Pointer(I):
                 (2) {Wait for new record}
            ELSE {Match to date}
                 INC Buffer-Pointer:
                 (14) {Continue scanning present record};
(12)   {Check Hi-Byte for EOF to match IGNF EOF}
       IF NOT (Hi-Byte=EOF) THEN
            (7) {Look for EOF, to match IGNF EOF, from next word}
       ELSE
            INC Buffer-Pointer(I):
            (8) {Wait for next field};
(13)   {Check Hi-Byte for an EOR}
       IF NOT (Hi-Byte=EOR) THEN
            IF (Lo-Byte=EOR) THEN
                 Error-Interrupt {EOR's are in Hi-Byte only}: (1)
            ELSE
                 (10) {Look for EOR, after mismatch, from next word}
       ELSE {Overwrite the mismatching record}
            Reload (I) DMA's next address with (Memory-Address)-1:
            (2);
(14)   {Match to date at Hi-Byte; continue scanning in present record}
       IF DMA-Request THEN
            Latch-Word:
            (6) {Look for match or mismatch at Lo-Byte}
       ELSE
            (14);
```

Figure 2. Sample Input Sentences

*Get me references on logic.*
*What are the journals on logic.*
*Which proceedings are after 1980?*
*I need journals during 1980.*
*Which ones of these are published after 1980?*
*Let me have journals on bibliographic retrieval after 1980.*
*What proceedings are published on logic after 1980?*
*Let me have all journals on logic and prolog published during 1982.*
*Get me the complete records.*
*I need those references on logic for information retrieval.*
*Which ones of these are on prolog?*
*Where can I find them?*
*Who wrote these articles?*
*Tell me who wrote them.*
*Where can I find references after 1980 on logic?*

Figure 3. Mapping of Parser Output to Logic Rules




4.2. Software Approach

Until special-purpose hardware is a reality, database designs must be considered that will be able to process inferences on large databases quickly. A software alternative to special-purpose hardware is suggested wherein the database records are indexed, as suggested by Warren (1981), but only the higher level of indices are kept in memory as predicates. The rest of the predicates can be stored as direct access records in a disk file, as shown in Figure 5. The indexing predicates contain the direct access record numbers as variables and can thereby be used to generate subsets of predicates for Prolog search as needed. In this way, problem spaces can be kept small for searching in the Prolog depth-first manner.

5. PHASE FOUR - DESIGN OF KNOWLEDGE STRUCTURE FOR BIBLIOGRAPHIC DATABASES

In the prototype, a simple hierarchical set of relationships is defined between the various components of the subject of each document. The same relationships are user to define the users' interests.

The fourth phase of this project is to investigate structures for the knowledge base that are perhaps better suited to this problem space. Faceted schemes appear to have promise and may provide those tools needed to define the concepts

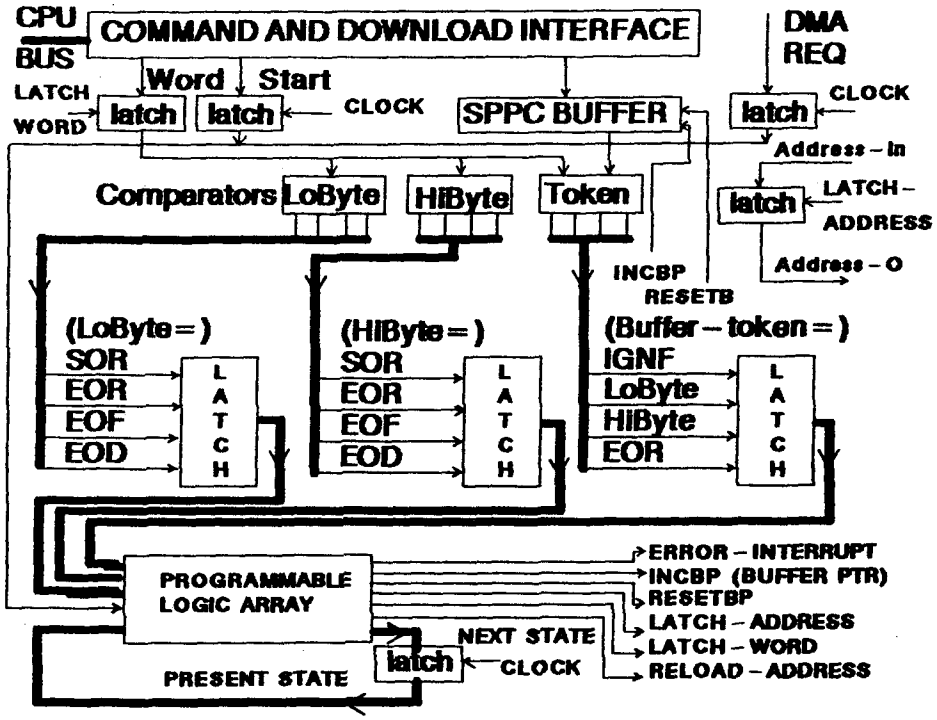
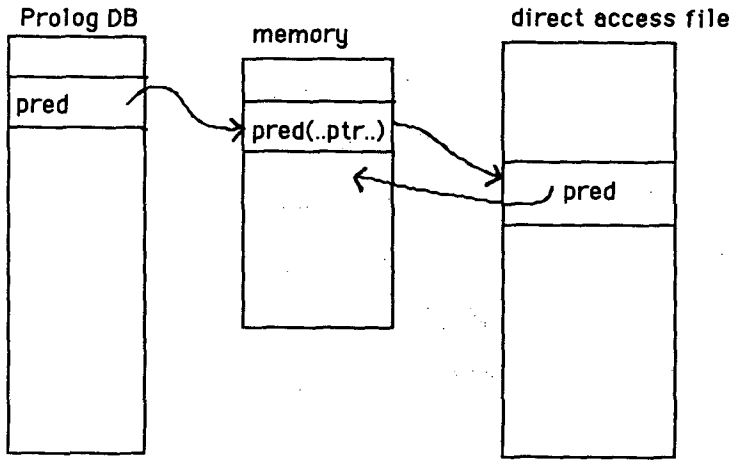



Figure 4 A Block Schematic of the SPPC

Figure 5. Use of Direct Access File for Indexing



and relationships that are actually present in a given database.

The relationships between components of a subject may be expressed through a fully faceted indexing scheme (Ranganathan, 1967). Shepherd and Watters (1985b) demonstrated that a document database, where each document is fully described by a faceted scheme, can be mounted in a relational DBMS and retrieval performed using the facets in addition to keyword occurrences. In a system that is essentially a faceted system, Smith et al (1984), were able to map domain-specific surface structures found in the indexing of the Chemical Abstracts into a set of deep structures or information classes. From these mappings it was possible to develop a relational database where the attribute fields were based on the information classes. In addition, the relationship between logic and relational DBMS has been described by Gallaire (1981) and Dahl (1982).

## 6. SUMMARY

A four-phased project exploring the application of predicate logic to information retrieval has been described.

A Prolog-based prototype system has been introduced that has the potential of providing a deductive reasoning capability for large bibliographic databases. Continuing research is directed towards the integration of a full set of faceted relationships that will increase the richness of the subject statement of a database entry. Coupled with this is the research and development of a more flexible hardware device and/or a software approach to provide response times suitable for online retrieval.

## 7. REFERENCES


Coelho, H. 1982. Man-machine communication in Portuguese: A friendly library service system. Information Systems. 7(2), 163-181.

Dahl, V. 1982. On database systems development through logic. ACM Transactions on Database Systems. 7(1), 102-123.

Crawford, R.G. 1981. The relational model in information retrieval. Journal of the American Society for Information Science. 32(1), 51-64.

Callaire, H. 1981. Impacts of logic on data bases. Proc. of the Seventh International Conf. on Very Large Data Bases. Cannes, France. 9-11 Sept. New York, IEEE. 248-259.

Macleod, I.A. and R.G. Crawford. 1983. Document retrieval as a database application. Information Technology: Research and Development. 2, 43-60.

Ranganathan, S.R. 1967. Prolegomena to Library Classification. 3rd ed. New York. Asia Publishing House.

Salton, G. and M. McGill. 1983. Introduction to Modern Information Retrieval. New York, McGraw-Hill Book Company.

Shepherd, M.A. and C. Watters. 1985a. A common interface for accessing document retrieval systems and DBMS for retrieval of bibliographic data. Information Processing & Management. 21(2), 127-138.

Shepherd, M.A. and C. Watters, 1985b. Implementation of facet-based retrieval using a relational database management system. Proc. of the International Conf. on Ranganathan's Philosophy: Assessment, Impact and Relevance. Edited by T.S. Rajagopalan. 11-14 Nov., New Delhi. New Delhi, Vikas Publishing House Pvt Ltd., 649-658.

Smith, P.J., Chignell, M. and D.A. Krawczak. 1984. Development of a knowledge-based bibliographic information retrieval system. Proceedings of the 1984 IEEE International Conference on Systems, Man and Cybernetics. 10-12 October, Halifax, Canada. 222-225.

Warren, D.H.D. 1981. Efficient processing of interactive relational database queries expressed in logic. Proc. of the Seventh International Conf. on Very Large Data Bases. Cannes, France. 9-11 Sept. New York, IEEE. 248-259.

Watters, C.R. 1986. Extended Sequential Search Model for Hardware Based Information Retrieval. Doctoral Thesis. School of Computer Science. Technical University of Nova Scotia.

Watters, C.R., Robertson, W., and Shepherd, M.A. "Prolog for bibliographic retrieval." 3rd International Conference on Systems Research, Informatics, and Cybernetics. Baden-Baden. August 19-24, 1986.


## APPENDIX A

### Predicate Formats

For the present application, each predicate has a name field and up to five other constants whre each constant is a term or a nested term (i.e., a list). A nested term may be nested to a depth of 3 per term. The proposed implementation relies upon control codes to delineate the terms and nested terms, therefore the record buffer length is the limiting factor on both the number of fields and the nesting depth. The predicate structure is:

name {optional fields}
where a field x is:
$string_x$, or
($string_{x1}$ $string_{x2}$ {$string_{x3}$..$string_{xm}$}).

For example, the following are valid predicate formats:

name;
name $c_1$ ;
name $c_1$ $c_2$ .. $c_m$;
name ($c_{11}$ ($c_{12}$ .. $c_{1n}$) $c_2$ .. $c_m$;
name ($c_{11}$ ($c_{121}$ .. $c_{12n}$)) $c_2$ .. $c_m$;
name ($c_{11}$ ($c_{121}$($c_{1211}$ .. $c_{121q}$) .. $c_{12n}$) $c_{13}$) $c_2$ .. $c_m$;

Predicate Formats on Disk

Each record of the disk file contains a single predicate. Upon making a request for the transfer of a data base segment, a buffer in the SPPC must be loaded with a predicate image according to a defined format. The control codes used to designate specific term and list boundaries are:

(BBB) Blank as required to align EOR or EOD to Hi-Byte of a word

(EBC) End of Bracketed Constant

(EOD) End of Data

(EOF) End of Field

(EONF) End of Name Field

(EOR) End of Record

(EOS) End of String

(SBC) Start of Bracketed Constant

(SOD) Start of Data

(SOR) Start of Record .

Using these control codes, the six valid predicate formats given above would be stored on the disk as follows (spaces and new lines are used for reasons of clarity only):

SOD
SOR name EONF EOR
SOR name EONF $string_1$ $EOS_1$ {BBB} EOR
SOR name EONF $string_1$ $EOS_1$ $EOF_1$
$string_2 EOS_2 EOF_2$ ..
{BBB} EOR
SOR name EONF $SBC_1$ $string_{11}$ $EOS_{11}$
$string_{12}$ $EOS_{12}$ ..
$string_{1n}$ $EOS_{1n}$
$EBC_1$ $EOF_1$
$string_2$ $EOS_2$ $EOF_2$ ..
$string_m$ $EOS_m$ $EOF_m$
{BBB} EOR
SOR name EONF $SBC_1$ $string_{11}$ $EOS_{11}$
$SBC_{11}$ $string_{121}$ $EOS_{121}$..
$string_{12n}$ $EOS_{12n}$
$EBC_{11}$
$EBC_1$ $EOF_1$
$string_2$ $EOS_2$ $EOF_2$
$string_m$ $EOS_m$ $EOF_m$
{BBB} EOR
SOR name EONF $SBC_1$ $string_{11}$ $EOS_{11}$
$SBC_{11}$ $string_{121}$ $EOS_{121}$
$SBC_{12}$ $string_{1211}$ $EOS_{1211}$..
$string_{121q}$ $EOS_{121q}$
$EBC_{12}$ ..
$string_{12n}$ $EOS_{12n}$
$EBC_{11}$
$string_{13}$ $EOS_{13}$
$EBC_1$ $EOF_1$
$string_2$ $EOS_2$ $EOF_2$
$string_m$ $EOS_m$ $EOF_m$
{BBB} EOR
{BBB} EOD

Upon making a request for the transfer of a data base segment, a buffer in the SPPC must be loaded with a query predicate in the above format. However, as there may be fields of which the contents are of no interest an additional control code, Ignore Next Field (IGNF), is required. For example, using a "-" to represent a field that is of no interest to the current query, the following is a valid query:

(name $V_1$ - $V_3$ -- )

This query predicate would be stored in the buffer in the following format:

SOR name EONF $vstring_1$ $EOS_1$ $EOF_1$
IGNF $EOF_2$
$vstring_3$ $EOS_3$ $EOF_3$
IGNF $EOF_4$
IGNF $EOF_5$
EOR.

SOD and EOD are not inserted in the SPPC record image buffer.