ICOT Technical Report: TR-293

TR-293

A Superimposed Code Scheme for Deductive Databases

by
M. Wada, H. Yamazaki, S. Yamashita,
N. Miyazaki (Oki), Y. Morita and H. Itoh

August, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Superimposed Code Scheme for Deductive Databases

Mitsunori Wada[†], Yukihiro Morita[‡], Haruaki Yamazaki[†], Shouji Yamashita[†], Nobuyoshi Miyazaki[†] and Hidenori Itoh[‡]

[†] Oki Electric Industry Co., Ltd., Tokyo, Japan
[‡] Institute for New Generation Computer Technology, Tokyo, Japan

ABSTRACT

An experimental distributed knowledge base system, KBMS PHI, is being developed as a part of the knowledge base research in the Fifth Generation Computer Systems project. A query expressed in Horn clause form is combined with related rules and compiled to relational operations to realize efficient processing in PHI.

A superimposed code scheme is being developed to speed up the processing. This paper describes the superimposed code scheme for compiled relational operations and analyzes its performance. An extension of the scheme for the processing of terms and rules is also discussed.

INTRODUCTION

The management of large shared knowledge bases is one of the most important research topics in realizing knowledge information processing systems. An experimental distributed knowledge base system, KBMS PHI, is being developed as part of knowledge base research in the Fifth Generation Computer Systems (FGCS) project. Two principal knowledge base models, the combined model and the integrated model, are being investigated in the FGCS project[1]. PHI is based on a variation of the combined model which is essentially a deductive database system. Horn clause queries are combined with related rules in the intensional database (IDB) and compiled to relational operations for the processing the extensional database (EDB) to utilize the database technology effectively. A superimposed code scheme is being investigated to support relational operations as well as the proc-

essing of rules. The major topic of this paper is the use of superimposed codes for relational operations in deductive database systems. The extension of the method for terms and rules is also briefly discussed.

KBMS PHI

KBMS PHI physically consists of a number of personal sequential inference machines (PSIs) linked by the ICOT LAN. The PSI is a personal computer system developed by ICOT that executes a logic programming language[2]. The ICOT LAN is an ethernet like local area network whose broadcast communication capacity reduces the communication overhead[3]. Some PSIs act as knowledge base machines which serve the requests from other PSIs used as host computers on which user programs run. This configuration represents one of the approximate models of the combination of the inference machine and the knowledge base machine for the FGCS. PSIs which have the role of knowledge base machines are called PHi machines in this paper.

Logical Configuration of PHI

KBMS PHI logically consists of global knowledge base managers and local knowledge base managers, as shown in Figure 1. One global knowledge base manager is dynamically assigned as a coordinator for each user program that accesses the knowledge bases. Local knowledge base managers that manage the related knowledge bases cooperate to answer requests from the user program. This configuration is a distributed and extended version of the model proposed in [4]. The major features of the previous model are as follows.

(1) A relational database management system (RDBMS) manages the EDB. The use of set oriented relational operations for large knowledge bases is essential to improve the overall performance.

(2) The Horn clause interface is used between the logic programs and the extended RDBMS. The extension is for recursive queries.

The first feature represented the commonly accepted view of the relationship between the inference and the relational databases when ICOT was inaugurated[5]. The second is the major issue of this proposal.

The model was extended for PHI to include the following additions.

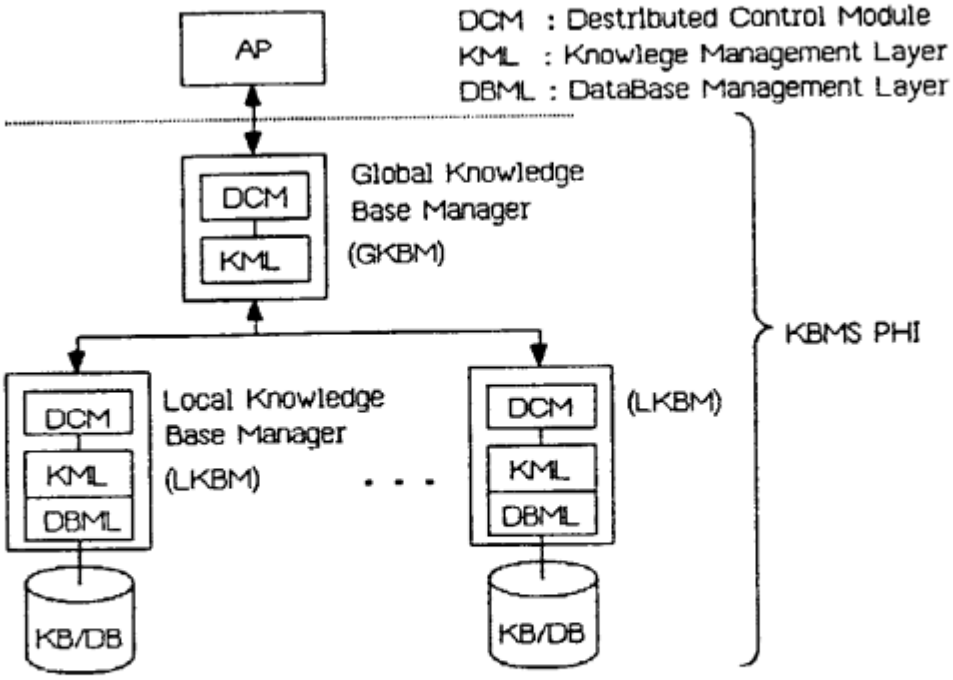

Figure 1 Logical Configuration of KBMS PHI

(3) The extended part is separated from the RDBMS as a knowledge management layer to broaden the scope of IDB management.

(4) A distributed control module has been added to cope with the distributed environment.

Thus, the kernel of KBMS PHI is a distributed deductive database system.

Query Processing in KBMS PHI

Query processing strategy in KBMS PHI was investigated by dividing the problem into two sub-problems: query processing of the distributed relational database and query processing of the deductive database. An integrated strategy is being developed based on strategies for these sub-problems. Query processing of the deductive database is summarized in this section.

The size of the IDB is assumed to be relatively small compared to the size of the EDB. The relational database operation is an attractive alternative in such a case to realize the deductive inference based on the well known one-to-one correspondence between a fact of the logic program and a tuple of a relation. A query is first combined with related rules in the

IDB. The resultant rule set can be regarded as a Horn clause query. A Horn clause query without functions (structures) can be easily compiled to a relational query if there are no recursive expressions in it. Thus, the central issue of query processing is how to deal with recursion. There have been many strategies proposed for recursive query processing. They can be classified based on their main characteristics, i.e. interpretation versus compilation and top down versus bottom up[6]. The compiled approach enables techniques developed in the database field to be applied in order to improve performance. Therefore, a strategy based on the compiled and bottom up approach was proposed for PHI[7]. It uses a procedure called Horn clause transformation to simplify queries for bottom up processing. Part of this strategy was implemented to study the behavior[8].

It is easy to process simple queries efficiently using the bottom up approach[7]. However, some kind of the binding (condition) propagation mechanism is necessary to improve the performance of a bottom up strategy for complex queries. The principle of restricted least fixed points that reduces the size of virtual relations was proposed for this purpose[9]. It introduce a rule set called restrictor rules, which are similar to the magic sets summarized in [6]. With this improvement, most queries including "not" predicates, and mutual recursions, can be processed effectively by relational operations in PHI.

Application of Superimposed Codes in KBMS PHI

The query processing of PHI consists of two phases. The first is the processing of the IDB related to the query. This phase includes the extraction of related rules from the IDB and the compilation of the query. The second phase is the execution of the compiled relational operations on the EDB to compute the answer. The system is being designed and implemented on PSIs, and part of it, the database management layer (RDBMS), is currently operational. The hardware support of the processing is also being investigated to improve the performance.

The use of superimposed codes possibly provides a unified approach to realize the efficient processing of deductive databases that consist of the IDB and the EDB. Therefore, a superimposed code scheme is being investigated as an alternative way of realizing the deductive database processing. Relational operations that frequently appear in compiled queries are

selections, joins, set operations and set comparisons. The frequent use of set operations and set comparisons is the major difference between deductive and relational databases. The use of superimposed codes in these operations is an effective way of improving the performance of the overall processing. PHI was designed under the assumption that the size of the EDB would be much larger than that of the IDB. Therefore, processing of these relational operations is more critical than IDB processing. An experimental knowledge base engine (KBE) is being designed based on a superimposed code scheme to process relational operations. The KBE is hardware attached to the PSI. The logical structure of the KBE is shown in Figure 2. The accelerator is simple dedicated hardware to process indices that consist of superimposed codes. The method to be used in the KBE is discussed in the following sections.

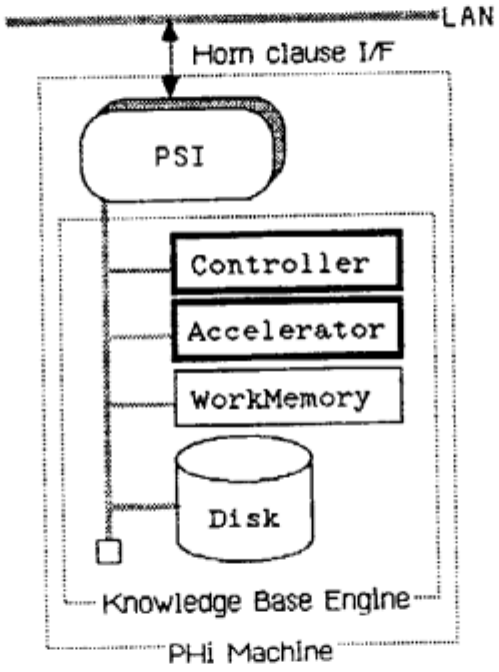

Figure 2 Knowledge Base Engine

RETRIEVAL USING A SUPERIMPOSED CODE

This section explains how to make an index, demonstrates retrieval with it, and discusses the optimum index parameters.

Index Creation

To retrieve large amounts of text data, a method using the superimposed code was proposed[10]. This method assigns index records with superimposed code words (SCWs) to a record file that has multiple key words. This method provides powerful partial match retrieval.

For quick tuple retrieval, an index with SCWs is introduced. An index value is derived as follows:

Suppose a relation, R, consists of N tuples; $R=\{T_1,T_2, \ldots ,T_N\}$. The relation, R, has r^R key attributes. To compute an index value, a hashing function is defined to map attribute values to codes called binary code words (BCWs) according to data type. For a given tuple, T_i, the index value is produced as follows:

(1) The value of each key's attributes of T_i is transformed to the BCW.

(2) All BCWs derived are ORed together.

The result of the OR operation is a SCW, an index of T_i. The index table of R is produced by pairing index values and a pointer to the corresponding tuple for all tuples of R. Figure 3 is an example of the index table.

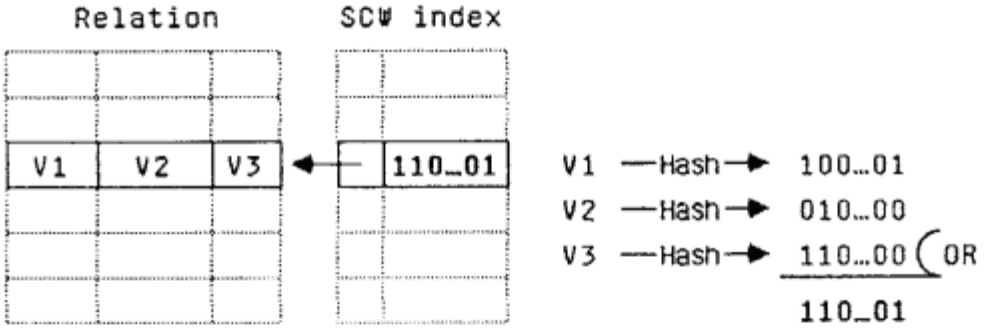

Figure 3 Index Creation

Retrieval Using an Index

As explained above , an index table is used for retrieval. In retrieval query q, r^q key attribute values are specified as a retrieval condition. The first phase, PHASE1, extracts candidate tuples that satisfy the retrieval condition comparing index values with a binary value, called a query mask. PHASE2 examines the contents of each candidate to extract the tuples satisfying the retrieval condition.

In more detail, query mask Q is computed from the set, Sq, of attribute values specified in the retrieval condition, analogous to the way in which an

index value is derived. First, the hashing function maps each element in Sq into a BCW. The query mask is derived from ORing together the r^q BCWs.

If Q and any index value, S_i, do not satisfy equation 1, the tuple corresponding to the index record dose not satisfy the query.

$$Q \wedge S_i = Q \quad \cdots\cdots (1)$$

As a result of PHASE1, a collection, C, of tuples whose index value, S_i, satisfies equation 1 is extracted. PHASE2 examines C so that the tuples which satisfy the retrieval condition can be picked up.

Set operations and set comparisons are frequently executed in PHI, as discussed in previous sections. Each tuple in a relation should be compared to every tuple in another relation in these operations. This tuple-wise comparison is very time consuming. The comparison time is $O(M \times N)$ for relations having M and N tuples.

It is difficult to use index methods such as B^+ or a hash table to reduce the processing time of these operations. The SCW index can reduce the time for these operations. First, relations are divided into tuple groups based on their index values. Next, each group of a relation is paired with a group of another relation that has the same index value. Then, comparisons can be made within these paired groups. The comparison time can be greatly reduced by pairing groups. The time for grouping and pairing is in the linear order of the size of relations. The details of these operations are discussed in [11].

Design of Index Parameters

An index value is assumed to have uniform distribution. Then $p(drops)$ is defined by equation 2:

$$p(drops) = (\textit{the number of } C \textit{ elements} / \textit{the number of all tuples in } R\ )$$

$$= \sum_{x=0}^{b} \phi(b,k,r^q,x)\ p(b,k,r^R,x) \qquad (2)$$

$$\phi(b,k,r^q,x) = (-1)^x {}_bC_i \sum_{i=0}^{x} (-1)^i {}_xC_i \left(\frac{{}_iC_k}{{}_bC_k} \right)^{r^q}$$

$$p(b,k,r^R,x) = \sum_{i=0}^{x} (-1)^i {}_xC_i \left(\frac{{}_{b-i}C_k}{{}_bC_k} \right)^{r^R}.$$

$p(drops)$ can be computed using parameters b, k, r^R, r^q[10], where r^R is the number of key attributes in the relation, r^q is key attributes specified at

retrieval condition, b is the length of BCW and k is the weight of BCW (number of '1' in BCW). b and k are parameters that specify the nature of the index. This section discusses the optimal value of k.

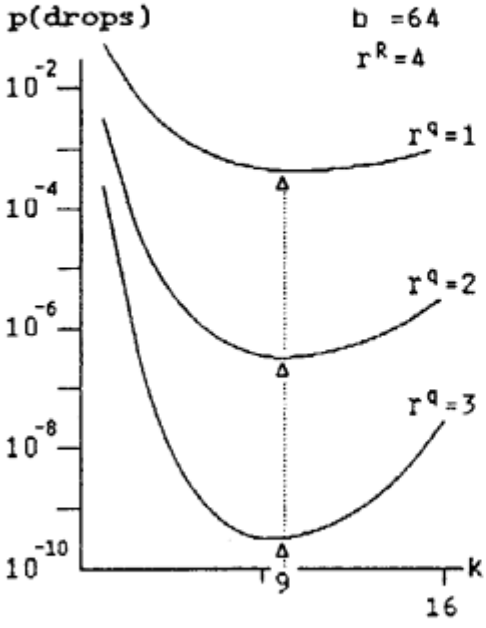

Figure 4 Transition of Selectivity (1)

As shown in Figure 4, when b and r^R are fixed, the transition of $p(drops)$ according to k's value is examined. The results is that the more r^q increases, the more $p(drops)$ decreases. Hence, the more key attributes specified in the query, the fewer tuples satisfy the query. The number of tuples satisfying the query is maximal when $r^q=1$. Thus, the optimization of value k is the most important design criterion. As can be seen in Figure 4, this optimal value of k does not change in other curves.

Next, as shown in Figure 5, the transition of $p(drops)$ in the case of $r^q=1$ according to k's value when b is fixed is examined. The result is that $p(drops)$ increases in proportion to r^R. Hence, if the number of key attributes in the relation increases, the number of tuples which do not satisfy queries is expected to increase.

Next, the value of b is set in proportion to r^R. Figure 6 illustrates the transition of $p(drops)$ in the case of $b=r^R\times 16,24,32$ and $r^R=2,8$. The value of k that minimizes $p(drops)$ is in this range. It is clear that if r^q is a constant, then the transition is very small, within

$$\frac{1}{2}\times\left(\frac{b}{r^R}\right) \leqq k \leqq \frac{2}{3}\times\left(\frac{b}{r^R}\right).$$

Therefore, k should be set in this range.

ESTIMATED PROCESSING TIME

This section estimates the cost of retrieval processing with SCWs, and compares it with the costs using the hashing table or B$^+$tree. The retrieval operation is the operation executed most often. The relational algebraic

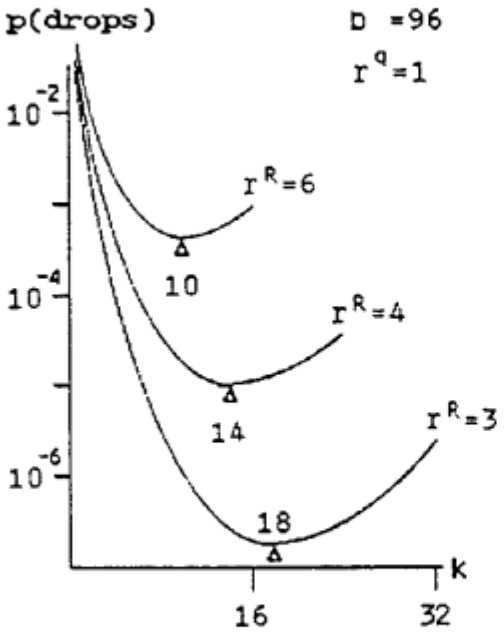

Figure 5 Transition of Selectivity (2)

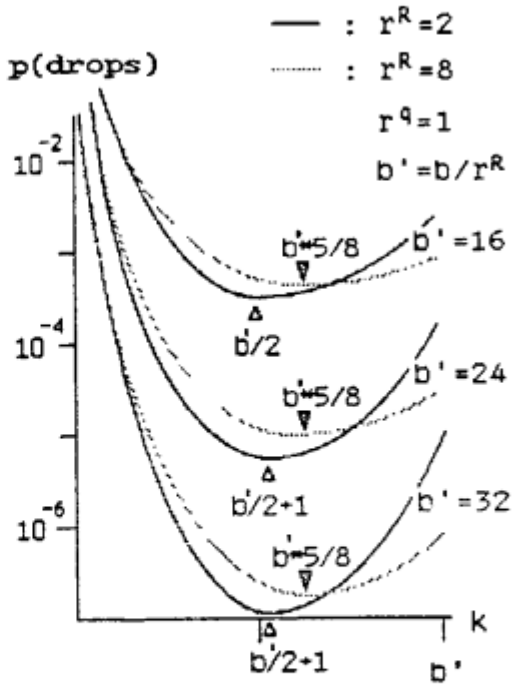

Figure 6 Transition of Selectivity (3)

command sequence is optimized by the selection-first strategy in the PHi machine. Therefore, the execution cost of retrieval command affects the entire execution cost of the PHi machine.

Here, a retrieval is executed to select all tuples satisfying a retrieval condition, and in the condition, some attribute values are specified and ANDed. When a retrieval command is applied to a relation that has a hashing table or B$^+$tree index of key attributes, it is applied in two phases as in the SCW method. The first phase, PHASE1, selects tuples that match one key value specified in the retrieval condition, called candidate tuples. The second phase, PHASE2, tests the candidates and determines tuples satisfying the retrieval condition completely.

Comparison Time

This section estimates the comparison time to select candidate tuples, and evaluates the effect on retrieval. All index records must be tested in the SCW method regardless of the number of tuples in a relation.

When retrieval is performed for an N tuple relation, PHASE1 should test index records N times. The number of comparison steps to select all candidates is N. When retrieval is performed for a relation that has hash

ing tables, only one access after hashing is required to select all candidates. One comparison step is required to select all candidates. For the B$^+$tree, the number of comparison steps to select all candidates is (Constant$\times\log_B N$) B:number of branches.

Therefore, the number of comparison steps of each method is estimated.

1) SCW index method : N

2) Hashing table method: 1

3) B$^+$tree index method : Constant$\times\log_B N$

Obviously, the SCW method gives the largest number of steps. The following values are assumed to estimate the order of comparison time.

Parameters

Number of tuples (N) : 2^{16}

Frequency of comparing one index value (f) : 1~10

Comparison time : $10\sim10^2$ nsec

Here, f is set to be 1 to 10, since the limitations of the register size of the accelerator mean that index values must sometimes be compared several times. The order of each method is:

O(SCW index method) = $10^{-1}\sim10$ msec

O(Hash table method) = $10\sim10^3$ nsec

O(B$^+$tree index method) = $10^{-1}\sim10$ μsec

The total comparison time with the hashing table or B$^+$tree is negligible compared to the disk access time, which is in the order of milliseconds. Because total comparison time with the SCW index is greater than in other methods, an accelerator is being designed to realize high-speed index processing in the KBE.

Estimated Disk Access Time

This section estimates the time to read candidate tuples from disk hardware in PHASE2. The following parameters are used:

s : Average seek time (msec)

r : Rotational latency (msec)

d : Disk transfer time (msec/byte)

P : Page size (bytes)

N: Number of tuples in a relation

n : Number of tuples satisfying a query

In the SCW method, when $p(drops) \ll 1$, the same number of pages as candidate tuples will be read. Candidate reading time T_{tuple} for disk access is estimated as follows:

$$T_{tuple} = N \times p(drops) \times (s + r + P/d)$$

In this expression, $N \times p(drops) = N_d$ is the expected number of candidates in PHASE1.

Figure 7 shows the disk read time. In this figure, the axis of the abscissa is the rate of n to N_d. Here, the following values are estimated:

1) Disk

$s + r$ = 25 msec

d = 1 Kbytes/msec

P = 4 Kbytes

2) Number of tuples in a relation : 216

3) Index value length : 16, 24, or 32 bits per attribute

For $b/r^R = 24,32,\ldots$, the disk access time to read candidates is represented by curve L_0. For $b/r^R = 16$ and $r^q = 1$, disk access time is constant up to a certain point as shown in L_1 and L_2. For $r^q \geqq 2$, disk read time is represented by curve L_0.

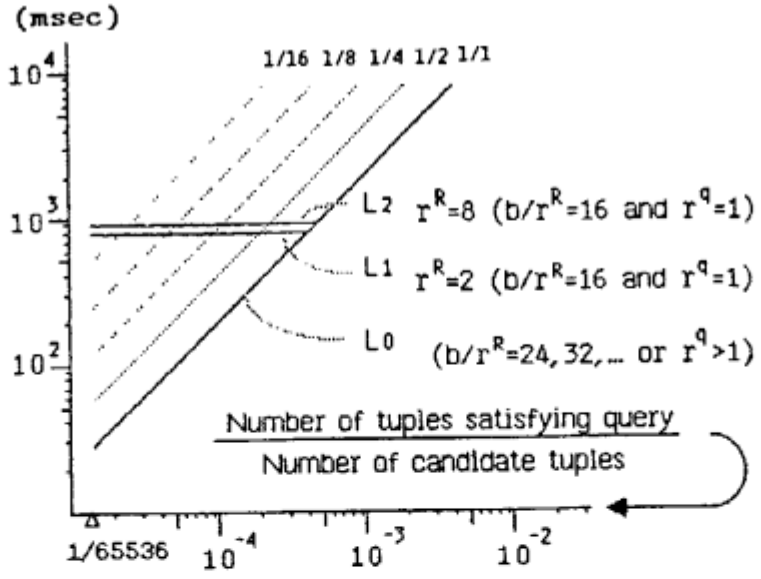

Figure 7 disk access time

When tuples are retrieved with the hashing table or the B$^+$tree index, candidate tuples are selected by using them for a single attribute. Next, the candidates are tested to see whether they satisfy the query. Candidate hit ration $P(HIT)$ is defined as follows:

$$P(HIT) = (\textit{Number of tuples satisfying query} / \textit{Number of candidate tuples}).$$

With $P(HIT)$, the expected reading time, T_{tuple}, for candidates can be calculated as follows:

$$T_{tuple} = n \times (s + r + P/d) / P(HIT).$$

According to this expression, Figure 7 explains the disk access time for $P(HIT) = 1/1, 1/2, 1/4, 1/8, 1/16$.

In the SCW method, for $r^q \geqq 2$ the disk access time follows L_0. In other methods, $P(HIT)$ is expected to decrease as r^q increases. If the number of the tuples satisfying a query exceeds a certain threshold, then the SCW method is better than the other methods in terms of disk access time to read candidates. As $P(HIT)$ decreases, the threshold value decreases. In other words, the performance of retrieval using the SCW method is expected to be better than other methods as the number of key attributes specified in the retrieval condition increases.

EXTENDING THE SUPERIMPOSED CODE SCHEME FOR TERMS AND RULES

PHI deals with not only the EDB but also the IDB. Therefore, a fast retrieve mechanism for rules is necessary if the size of the IDB becomes large. Another problem is processing structures in the IDB and EDB. These two problems can be handled by the superimposed code scheme for terms.

The main problem in processing terms is expressing variables and composite terms by superimposed codes. A scheme called the structural superimposed code word (SSCW) method is being investigated.

First, a term is represented by a tree. If the term is an atom or a variable, it is represented by a simple tree that consists of only a root node. If the term is composite, the functor of the term is represented by the root and its arguments become its children. If some arguments are again composite, they are represented by subtrees. The hashing function for the BCW is decided under the following conditions.

(1) Each functor, atom, or variable that is a component of a term is mapped to a BCW shorter than the intended SSCW.

(2) Variables are mapped to the codes whose components are all '1' for indices of terms in the IDB or the EDB. They are mapped to all '0' for query masks.

(3) The range of the BCW for a parent node covers those of its children.

An SSCW for a term is computed by superimposing BCWs of its components according to its structure, as shown in Figure 8. If a term, t_1, is unifiable with another term t_2, then the index value (SSCW), S, for t_1 and the query mask, Q, for t_2 satisfy equation 1 in the previous section.

There have been several indexing schemes proposed for terms and rules [12] to [15]. The relationship of these methods are briefly analyzed in [16] and [17], and it is shown that the selectivity of the SSCW scheme is better than that of methods discussed in [12] and [13].

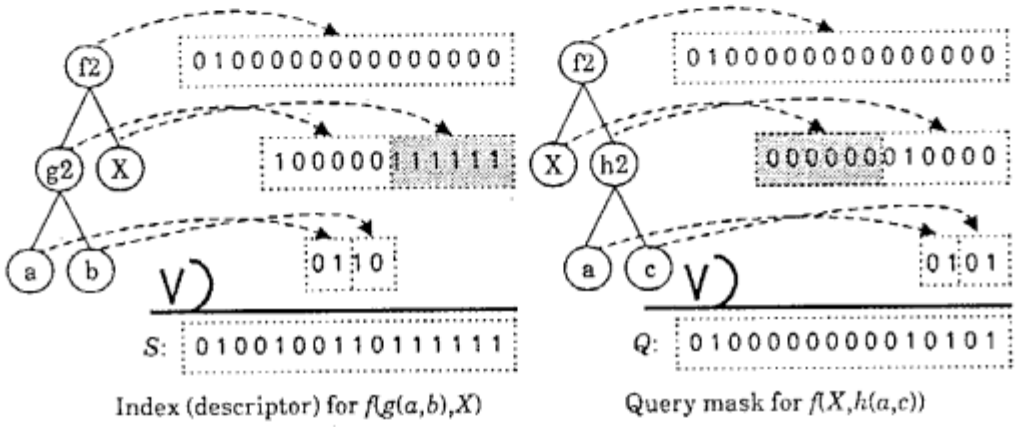

Figure 8 Example of SSCW

CONCLUSIONS

This paper described a superimposed code scheme planned to be used in KBMS PHI. PHI compiles a deductive query to relational operations to effectively process the query. The superimposed code scheme is used for indices of the EDB, i.e. the relations, to improve the performance of the execution of the compiled query. The advantages and disadvantages of the scheme were analyzed. The scheme is expected to be more effective than other methods if the number of key attributes specified in the retrieval condition is larger than one. This scheme is also very effective for the set operations and comparisons frequently used in deductive database processing. An experimental knowledge base engine is being designed based on this scheme.

The extension of the scheme for the processing of the IDB, i.e. rules, were also briefly discussed. The superimposed code scheme is suitable for a parallel architecture because it is structurally simple. The investigation of the parallel architecture is considered to be one of the directions of future research.

REFERENCES

[1] Itoh, H., Research and Development on Knowledge Base Systems at ICOT, *Proc. of 12th VLDB*, pp.437-445, 1986

[2] Uchida, S. and Yokoi, T., Sequential Inference Machine: SIM, *Proc. of FGCS*, pp.58-69, Tokyo 1984

[3] Taguchi et al., INI: Internal Network on the ICOT Programming Laboratory and Its Future, *ICCC*, Sydney, Australia, Oct. 1984

[4] Miyazaki, N., A Data Sublanguage Approach to Interfacing Predicate Logic Languages and Relational Databases, ICOT Technical Memorandum, 1982

[5] Fuchi, K., Aiming for Knowledge Information Processing Systems, *Proc. of FGCS*, p.2-15 to p.2-28, Tokyo, 1981

[6] Bancilhon, F. and Ramakrishnan R., An Amateur's Introduction to Recursive Query Processing Strategies, *SIGMOD '86 Proc.*, pp.16-52, 1986

[7] Miyazaki, N., Yokota, H. and Itoh, H., Compiling Horn Clause Queries in Deductive Databases: A Horn Clause Transformation Approach, ICOT Technical Report, 1986

[8] Abiru, Y., Haniuda, H., Miyazaki N. and Morita, Y., KBMS PHI: An Experimental Deductive Database System PHI/K2, *Proc. 34th Annual Convention IPS Japan*, pp.1491-1492, 1987 (in Japanese)

[9] Miyazaki, N. and Itoh, H., Restricted Least Fixed Points and Recursive Query Processing Strategies, ICOT Technical Report,1987

[10] Roberts, C.S., Partial-Match Retrieval via Method of Superimposed Codes, *Proc. of IEEE*, 67(12), pp.1624-1642, 1979

[11] Wada, M., et al., KBMS PHI: Superimposed Codes for Relational Algebra Operations, *Proc. 34th Annual Convention IPS Japan*, 3K-7, pp.1489-1490, 1987, (in Japanese)

[12] Wise, M. J., and Powers, D. M. W., Indexing PROLOG Clauses via Superimposed Code Words and Field Encoded Words, *Proc. IEEE Conf. Logic Programming*, Atlantic City, NJ, January 1984, pp.203-210.

[13] Morita, Y., et al., Retrieval-By-Unification Operation on a Relational Knowledge Base, *Proc. of the 12th VLDB*, 1986

[14] Ramamohanarao, K., and Shepherd J., Answering Queries in Deductive Database Systems, *Logic Programming: Proc. Fourth Int'l Conf.* Vol.2, pp.1014-1033, 1987

[15] Berra, P.B. et al., Computer Architecture for a Surrogate File to a Very Large Data/Knowledge Bases, *IEEE COMPUTER*, March 1987

[16] Morita, Y., Wada, M. and Itoh, H., Structure Retrieval via the Method of Superimposed Codes., *Proc. 33rd Annual Convention IPS Japan*, 6L-8, 1986, (in Japanese)

[17] Morita, Y., et al., A Knowledge Base Machine with an MPPM (3) - An Indexing Scheme for Terms -, *Proc. 35th Annual Convention IPS Japan*, 2C-7, 1987, (in Japanese), (to appear)