TR-383

An Indexing Scheme for Terms Using Structural Superimposed Code Words

by
Y. Morita, A. Nakase(Toshiba) and H. Itoh

May, 1988

An Indexing Scheme for Terms using Structural Superimposed Code Words.

Yukihiro Morita, Akihiko Nakase[*] and Hidenori Itoh

ICOT Research Center,

21F, Mita Kokusai Building,

1-4-28, Mita, Minato-ku, Tokyo, 108, JAPAN

May 1988

Abstract

This paper describes an indexing scheme for terms and rules. It proposes a *structural superimposed code word* (*SSCW*) scheme, and discusses several characteristics of the scheme.

In knowledge information processing systems with a large amount of knowledge, a technique for fast knowledge retrieval is very important. There are some efficient techniques to retrieve simple data (atomic values). However, knowledge should be represented by a more complex data structure. One such basic data structure is a term. A term is a logical structure with variables. Fast retrieval is very important for a term which is unifiable to a specific term (query) from a set of terms.

This paper describes an indexing scheme for terms. This scheme is an application of the *superimposed code word* (*SCW*) method of data retrieval. This paper also describes the code design for the scheme.

[*]Toshiba Corporation

1 Introduction

Fast knowledge retrieval is essential for the knowledge information processing systems with large amounts of knowledge. There are some efficient techniques to retrieve simple data (atomic values). However, knowledge should be represented by a more complex data structure. One such basic data structure is a term. A term is a very powerful data structure in symbolic processing. We are developing a knowledge base machine in which the basic element is a term [MMISS88]. For conditions of term retrieval, we need unifiable relations more than equality relations. Techniques to retrieve a set of terms unifiable with specific terms is very important.

We proposed an indexing scheme for terms, called the ***structural superimposed code word*** (*SSCW*) method [MWI86][MMNS87]. This method is an application of the superimposed code word (*SCW*) method for partial match data retrieval [Ros79] to fast term retrieval. The SSCW method is a very efficient way of retrieving an object that has unifiable terms with specific (query) terms.

This paper describes an indexing scheme for term retrieval based on the SCW method and discusses some properties of the method.

2 Structural Superimposed Code Words

We proposed an indexing scheme using superimposed code words, called the structural superimposed code word scheme. This section describes this scheme, first looking at the superimposed code word method for data retrieval [Ros79].

2.1 SCW Method for Data Retrieval

The SCW method is a method for retrieving a data or record that has several key words specified as a query. In the SCW method, each record, R_i, has several key words, $K_{i,1}, \ldots K_{i,n_i}$, and one code word, S_i, for it. Each key word, $K_{i,j}$, is mapped by the hashing function to a binary code word (BCW), $b_{i,j}$. (This binary code word is sometimes called a *descriptor*.) A binary code word is a fixed length bit string. For the i-th record, a superimposed code word, S_i, is computed by logical OR (superimpose) of all BCWs of the record's key words. That is,

$$S_i = \bigvee_{j=1}^{n} b_{i,j} \tag{1}$$

Here, $\vee$ means a bitwise logical OR operator. To retrieve a record for several key word, first, we make a code word (called a *query mask*), Q, by superimposing each BCW for key words. For example, for the query that retrieves the record with key word key_1 and key_2, the query mask is a logical OR of a BCW of key_1 and key_2 ($bcw(key_1) \vee bcw(key_2)$, where bcw is a hashing function).

If the i-th record satisfies the query, then the following equation holds:

$$Q = Q \wedge S_i. \tag{2}$$

Here, $\wedge$ means a bitwise logical AND operator. Of course, there are some records that satisfy equation (2), but do not satisfy the query. Such misselections are called *false drops* [Ros79]. However, checking equation (2) can reduce the search space for records. Since checking equation (2) performs a very simple operation, logical AND, the SCW method can retrieve a set of records that satisfy the query effectively, especially for partial match retrieval.

2.2 SSCW Method for Term Retrieval

We extend this method to apply to term retrieval, which retrieves possibly unifiable terms to the query term from the set of terms. A term is a data structure with variable, and is defined as follows.

Definition 1 Let VAR be a set of variables and FUN_i be a set of i-place functors. FUN_0 means a set of constants. Then, terms are defines recursively as follows:

1. $X \in VAR$ (variable) is a term
2. $f \in FUN_0$ is a term
3. $f(X_1, \ldots, X_n)$ is a term, if $f \in FUN_n$ and $X_i (i = 1, \ldots, n)$ are terms

To apply the SCW method of data retrieval to term retrieval, we extend the SCW method the in following two points:

- Use a different hashing function between the code word (descriptor) and query mask (use special mapping for variables)
- Encode all functors and variables in a term to a partial BCW, and superimpose them according to the term's structure

To encode a term structure, we introduce a *partial BCW*.

Definition 2 A *partial binary code word* on a set of integer, X, is a BCW in which the i-th bit position is 0, if $i \notin X$. Here, $X \subset \{1, \ldots, l\}$ and l is the width of the BCW.

The hashing function from $FUN \cup VAR$ to a set of partial BCWs on X is denoted $H_X(f)$, where f is a member of $FUN \cup VAR$. X is called the *hash bit field* (*HBF*) of the hashing function.

Then we define two hashing functions for terms, one is to make index (descripter) HTs, and the other is to make query mask QTs.

Let $HF_X(f)$ be an arbitrary hashing function on X for the functor and $Bit_X(b)$ ($b = 0$ or 1) be a bit string which the bit at the i-th position is

$$\begin{cases} b & \text{if } i \in X \\ 0 & \text{if } i \notin X \end{cases}$$

That is, $Bit_X(0)$ means a code word in which all bits are 0 for any X. Let $subrange(X, f, n, i)$ be a mapping to a set of integers, where X is a set of integers, $f \in FUN_n$ and n and i are integers, $1 \leq i \leq n$ and $subrange(X, f, n, i) \in X$. Then the hashing function of the superimposed code word method is defined as follows:

Definition 3 *A hashing function on X of the SSCW descriptor*, $HT_X(T)$, and *query mask*, $QT_X(T)$, for term T, are defined as follows:

1. If v a is variable: $HT_X(v) = Bit_X(1)$, $QT_X(v) = Bit_X(0)$
2. If f is a constant (0-place functor): $HT_X(f) = QT_X(f) = HF_X(f)$
3. If f is an n-place functor, where $X_i = subrange(X, f, n, i)$ ($X_i \subset X$) for all i and $n > 1$

$$HT_X(f(t_1, \ldots, t_n)) = HF_X(f) \vee \bigvee_{i=1}^{n} HT_{X_i}(t_i)$$

$$HQ_X(f(t_1, \ldots, t_n)) = HF_X(f) \vee \bigvee_{i=1}^{n} QT_{X_i}(t_i)$$

Example 1 Let term T_1 be $f(g(a), Y)$; here, $f \in FUN_2$, $g \in FUN_1$, $a \in FUN_0$ and $Y \in VAR$. Let the width of the code word be 16 bits, and $X_0 = \{1, \ldots, 16\}$.

To compute $HT_{X_0}(f(g(a), Y))$ and $QT_{X_0}(f(g(a), Y))$, we must compute the hashing function, $HF_{X_0}(f)$, $HT_{X_1}(g(a))$, $QT_{X_1}(g(a))$, $HT_{X_2}(Y)$ and $QT_{X_2}(Y)$. Here, X_1 and X_2 must be a subset of X_0.

In this example, let X_1 be $\{5, \ldots, 10\}$ and X_2 be $\{11, \ldots, 16\}$.

Since $HT_{X_1}(g(a)) = HF_{X_1}(g) \vee HF_{X_3}(a)$ and $X_3 \subset X_1$, let X_3 be $\{7, 8, 9, 10\}$.

Then we assume the hashing values are:

$HF_{X_0}(f) =$ ‘$\underline{0010010000010101}$’

$HF_{X_1}(g) =$ ‘0000$\underline{100010}$000000’

$HF_{X_3}(a) =$ ‘000000$\underline{1000}$000000’

Where the underline indicates the hash bit field of each function.

$HT_{X_2}(Y) =$‘0000000000$\underline{111111}$’ and $QT_{X_2}(Y) =$‘0000000000$\underline{000000}$’, then the SSCW for term $HT_{X_0}(f(g(a), Y))$ is:

$$\begin{array}{rl}
HF_{X_0}(f) = & \text{`}\underline{0010010000010101}\text{'} \\
HF_{X_1}(g) = & \text{`}0000\underline{100010}000000\text{'} \\
HT_{X_2}(Y) = & \text{`}0000000000\underline{111111}\text{'} \\
HF_{X_3}(a) = & \text{`}000000\underline{1000}000000\text{'} \\
\hline
HT_{X_0}(f(g(a),Y)) = & \text{`}0010111010111111\text{'}
\end{array}$$

The query mask for term $QT_{X_0}(f(g(a),Y))$ is:

$$\begin{array}{rl}
HF_{X_0}(f) = & \text{`}\underline{0010010000010101}\text{'} \\
HF_{X_1}(g) = & \text{`}0000\underline{100010}000000\text{'} \\
QT_{X_2}(Y) = & \text{`}0000000000\underline{000000}\text{'} \\
HF_{X_3}(a) = & \text{`}000000\underline{1000}000000\text{'} \\
\hline
QT_{X_0}(f(g(a),Y)) = & \text{`}0010111010010101\text{'}
\end{array}$$

Proposition 1 *Let T_1 and T_2 be terms, and let HT_X and QH_X be instances of the hashing function defined as Definition 3 for some set of integers, X. Then, if T_1 and T_2 are unifiable, the following equation holds:*

$$HT_X(T_1) \wedge QT_X(T_2) = QT_X(T_2) \tag{3}$$

Of course, there are some terms that satisfy equation (3), but are not unifiable with the query term (T_2). Such mis-selection are called false drops. In the following section, the performance of the SSCW is measured by counting the false drops.

Example 2 Let term T_2 be $f(g(V),b)$; here, $f \in FUN_2$, $g \in FUN_1$, $b \in FUN_0$ and $V \in VAR$. Let X_0, X_1, X_2 and X_3 be the same values of example 1.

Then we assume the hashing value is $HF_{X_2}(b)$ = `$0000000000\underline{100100}$'. Then the SSCW and the query mask for T_2 are:

$$\begin{array}{rl}
HF_{X_0}(f) = & \text{`}\underline{\text{0010010000010101}}\text{'} \\
HF_{X_1}(g) = & \text{`0000}\underline{\text{100010}}\text{000000'} \\
HF_{X_2}(b) = & \text{`0000000000}\underline{\text{100100}}\text{'} \\
HT_{X_3}(V) = & \text{`000000}\underline{\text{1111}}\text{000000'} \\
\hline
HT_{X_0}(f(g(V),b)) = & \text{`0010111111111010'}
\end{array}$$

$$\begin{array}{rl}
HF_{X_0}(f) = & \text{`}\underline{\text{0010010000010101}}\text{'} \\
HF_{X_1}(g) = & \text{`0000}\underline{\text{100010}}\text{000000'} \\
HF_{X_2}(b) = & \text{`0000000000}\underline{\text{100100}}\text{'} \\
QT_{X_3}(V) = & \text{`000000}\underline{\text{0000}}\text{000000'} \\
\hline
QT_{X_0}(f(g(V),b)) = & \text{`0010110010110101'}
\end{array}$$

T_1 of example 1 and T_2 are unifiable. $HT_{X_0}(T_1) \wedge QT_{X_0}(T_2) = QT_{X_0}(T_2)$, and also $HT_{X_0}(T_2) \wedge QT_{X_0}(T_1) = QT_{X_0}(T_1)$. That is,

$$\begin{array}{rl}
HT_{X_0}(f(g(V),b)) = & \text{`0010111111110101'} \\
QT_{X_0}(f(g(a),Y)) = & \text{`0010111010010101'} \\
\hline
QT_{X_0}(f(g(a),Y)) = & \text{`0010111010010101'}
\end{array}$$

$$\begin{array}{rl}
HT_{X_0}(f(g(a),Y)) = & \text{`0010111010111111'} \\
QT_{X_0}(f(g(V),b)) = & \text{`0010110010110101'} \\
\hline
QT_{X_0}(f(g(V),b)) = & \text{`0010110010110101'}
\end{array}$$

We can extend a query that specifies several query terms to retrieve a unifiable term with query terms. That is, the SSCW method can be applied to the query of searching for a term unifiable with term t_1 and t_2. In this case, we can use $h(T_1) \vee h(t_2)$ as a query mask for the query; here, h is a query hashing function. In the following section, we discuss only single terms for queries.

3 Code Design for SSCWs

When we design a hashing function for SSCWs, we can select the functions, $subrange(X, f, n, i)$ and HF_X.

An important factor for the hashing function for functor HF_X is the *weight* of a code word, that is, the bit positions with binary value 1. In this paper, the ratio of the bit positions with binary value 1 to the cardinality of X ($| X |$) is called the *bit setting ratio* (BSR).

A binary code word for the functor (without a 0-place functor) can be divide into two fields; one is a set of bit positions in the hash bit field of the hashing function of its argument, another is a set of bit positions that does not use any arguments. The former field is called the *superimposed field* (SF) of the hash bit field of the functor, and the latter field is called the *non-superimposed field* (NSF) of the hash bit field of the functor. The ratio of the superimposed field to the hash bit field is called the*superimposed ratio.*

For the function, $subrange(X, f, n, i)$, the following function is used in the following section to observe an effect of the superimposed ratio:

$$subrange(X, f, n, i) = divide(X, w_1, n, i) \tag{4}$$

Here, we regard X as an arbitrary ordered set, and $divide(X, w_1, n, i)$ means a set of integers that contains from the j-th element of X such that

$$\frac{i-1}{n} \times w_1 \times \mid X \mid < j \leq \frac{i}{n} \times w_1 \times \mid X \mid$$

w_1 is a positive rational number where $0 \leq w_1 \leq 1$. w_1 is the superimposed ratio.

3.1 Bit Setting Ratio

[Ros79] gives a code design of SCWs for data retrieval. This section also discusses a code design of SSCWs for term retrieval. However our design is too complicated to analysis in the same way as [Ros79], because we assume terms and unification operation, and we use partial BCW. Therefore, we give a simple analysis.

Let p_i be the probability that binary value 1 occurs in the i-th bit of the descriptor and q_j be the probability that binary value 1 occurs in the j-th bit of query mask. The width of the SSCW is denoted b. We assume that p_i and p_j, and q_i and q_j are independent of each other for approximation. The probability of that equation (3) holds is:

$$\prod_{i=1}^{b}(p_i q_i + (1 - q_i)) \tag{5}$$

Let β_i be the probability of the i-th bit in the hash bit field corresponding to a variable, and let α_i be q_i. Then $p_i = (1 - \beta_i)\alpha_i + \beta_i$, and:

$$\prod_{i=1}^{b}((\beta_i - 1)\alpha_i^2 - (\beta_i - 1)\alpha_i + 1) \tag{6}$$

When $\alpha_i = 1/2$ (for all i), (6) is minimal and the minimal value is:

$$\prod_{i=1}^{b}(\frac{1}{2} + \beta_i) \tag{7}$$

This means that in the best hashing function, binary values 0 and 1 occur with equal probability in each position of the query mask (or descriptor without a variable).

The optimal values of the bit setting ratio depend on the set of terms to be retrieved. Since there are too complicated to be analyzed mathematically, we have done some experiments using small sample data.

To evaluate the SSCW performance, we used several term sets, each with randomly generated 100 terms. These 100 terms are used both as 100 data terms and as 100 query terms. That is, we use the *unification join* [MYNI86] for the evaluation that can be regarded as 100 queries to a set of 100 terms.

To select the bit setting ratio of hashing function HF_X for functors, we note that each hash bit field is divided into two: a superimposed field (SF) and a non-superimposed field (NSF). Therefore, we can divide the hash bit field of a functor into the SF and NSF, and use a different bit setting ratio to reduce a false drop ratio. This hashing function is called *field separated hash* (FSH). The bit setting ratio of the superimposed field is denoted BSR_{SF}, and the non-superimposed field is denoted BSR_{NSF}. The hashing function of a functor without dividing any field is called *uniformly distributed hash* (UDH)

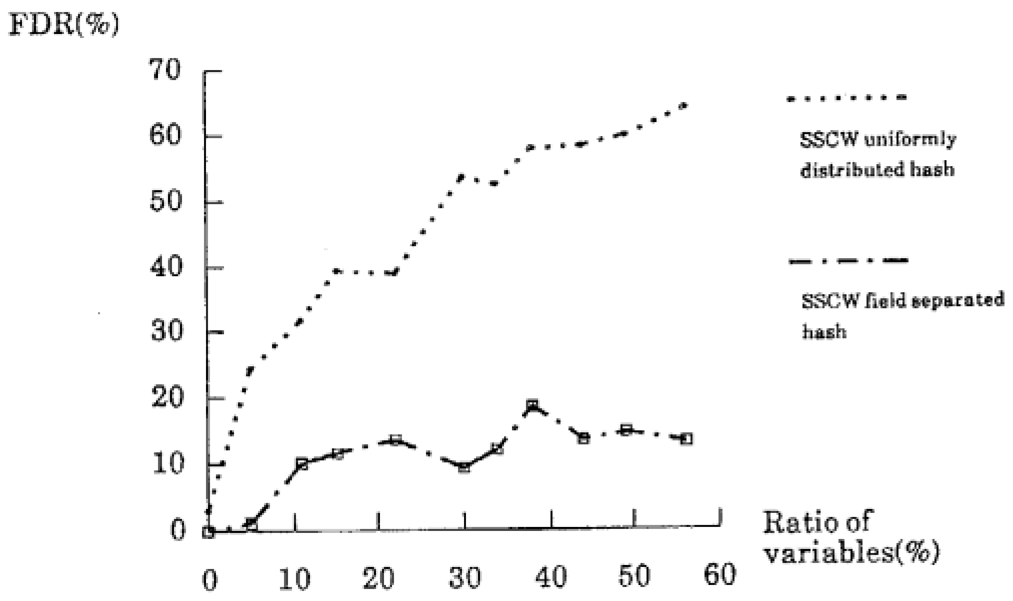


Figure 1 Field separated hash and Uniformly distributed hash

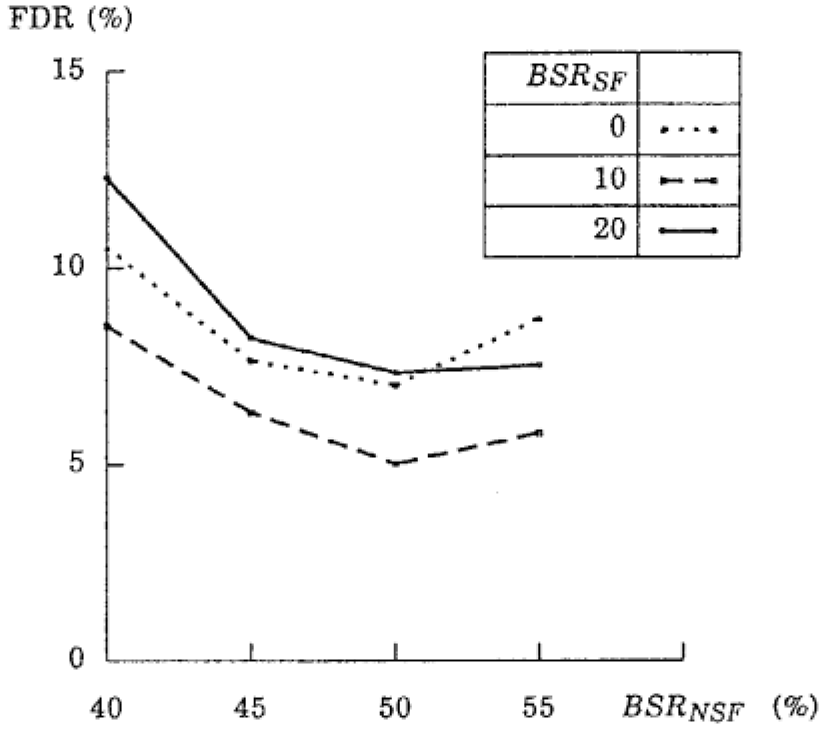


Figure 2 SSCW Bit Setting Ratios

Figure 1 shows the effect of the field separated hash. This figure shows the relationship between the variables in a term and false drop ratio of two hasing method for the sample data. The uniformly distributed hash method is weak in terms with many variable because of the effect of variable in data terms. Suppose the descriptor for an SSCW of $f(V, W)$ whose superimposed ratio is 70%, 70% of the hashed value of the functor f is masked by the hashed value of V and W. (Hashed value of variable for descriptor has bit 1 at all bit position in its hash bit field.) In uniformly distributed hash, the information about f is distributed uniformly in the superimposed field and non-superimposed field. However, in field separated hash, most of the information on f is concentrated in the non-superimposed field. Thus even for terms with many variable, information on a functor in the SSCW is not masked signigicantly. Therefore, we use the field separated hash in the following section.

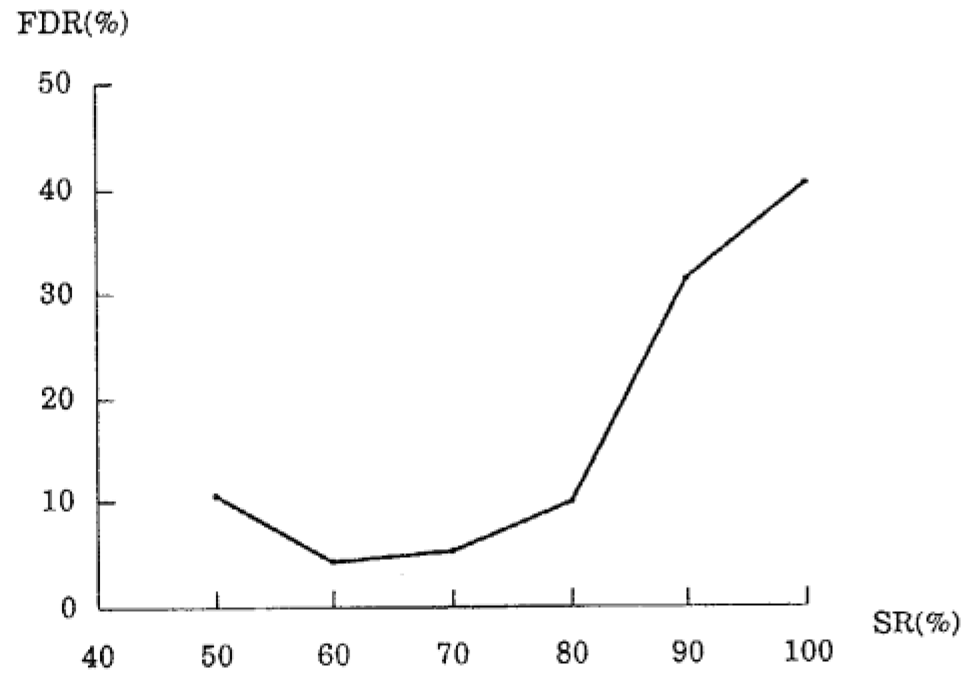


Figure 3 SSCW Superimposed Ratio

Figure 2 shows the relationship between the bit setting ratio and false drop ratio for the sample data. In the test data of this experiment, the average depth of terms is 2 to 3, and 1.5 to 2.5 hashed values of the functor (without variables) are superimposed in average. It seems that a BSR_{SF} of 50% in the non-superimposed field and a BSR_{SF} of 10 in the superimposed field is good for this sample data.

3.2 Superimposed Ratio

Figure 3 shows the relationship between the superimposed ratio and the false drop ratio. In this data, the average number of arguments of a functor is 2 or 3. Hence, when the superimposed ratio is about 70% (2/3 to 3/4), the performance of the SSCW is good. When the superimposed ratio becomes too large, the false drop ratio becomes very large. Because there are some terms in which arguments are variable, such as $f(V, W)$, the false drop ratio becomes large. Too small a superimposed ratio also generates a bad false

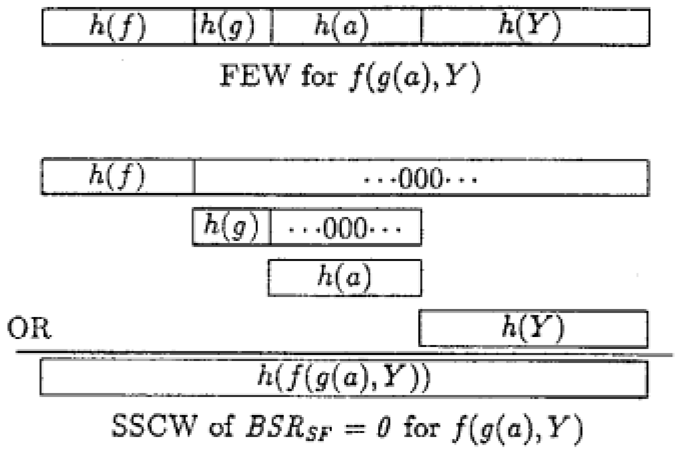


Figure 4 Example of the FEW and SSCW when $BSR_{SF} = 0$

drop ratio, because of the lack of information for arguments.

4 Indexing Schema for Terms and Structures

Several indexing scheme for terms and rules have been proposed. This section describes these schema.

Wise and Powers proposed the field encoded word (FEW) in [WP87] for Prolog's clause index. In this method, the code word is divided into several fields for each hashed value of symbols in a term, as illustrated in Figure 4. The hashed value of variables is all binary values 1 in the bit field for descriptors and all 0 for query masks, the same as for SSCWs. The FEW retrieval method is also $S_i \wedge Q = Q$, where S_i is a descriptor for the i-th term and Q is a query mask for a query.

The FEW method can be regarded as a special case of the SSCW method. (See Figure 4.) The FEW hashing function can be regarded as a kind of field separated hash where the bit setting ratio of the superimposed field is 0.

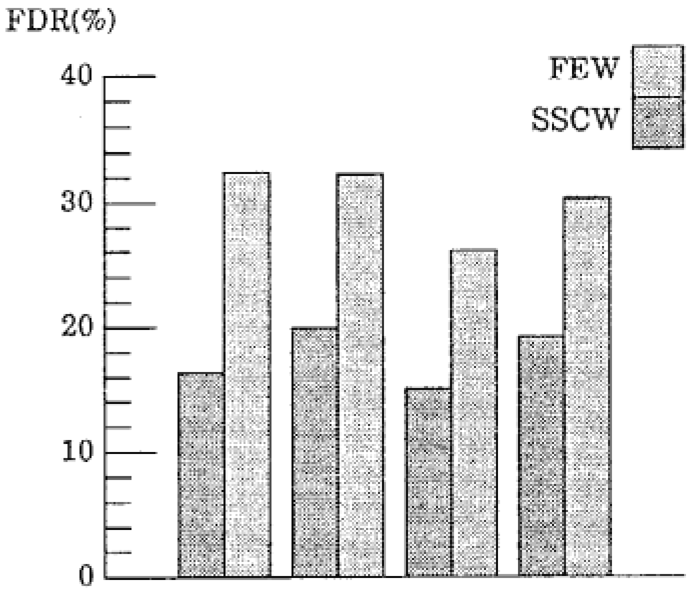
Since no hashed value of the symbol in a term is superimposed in FEW, the optimal value of the bit setting ratio of the hashing function of functors is 1/2. Since some parts of the hash bit field of each symbol are superimposed, the SSCW of section 3 has a wider hash bit field than that of the FEW. However, this causes some interference between overlapped hash bit fields for SSCWs of section 3. Then, if there are many variables in a term to be retrieved, this interference increase the number of false drops.

Figure 5 shows the performance of the SSCW method and FEW method. In this figure, the sample data is divide into two groups; Figure 5 (a) shows a situation where there are many variables in terms to be retrieved, and few variables in query terms. Figure 5 (b) shows a situation where there are few variables in terms to be retrieved and many variables in a query terms. In (a) the SSCW of section 3 has a small false drop ratio, because of the wider hash bit field. In (b) the FEW method shows good performance, because there is no interference.

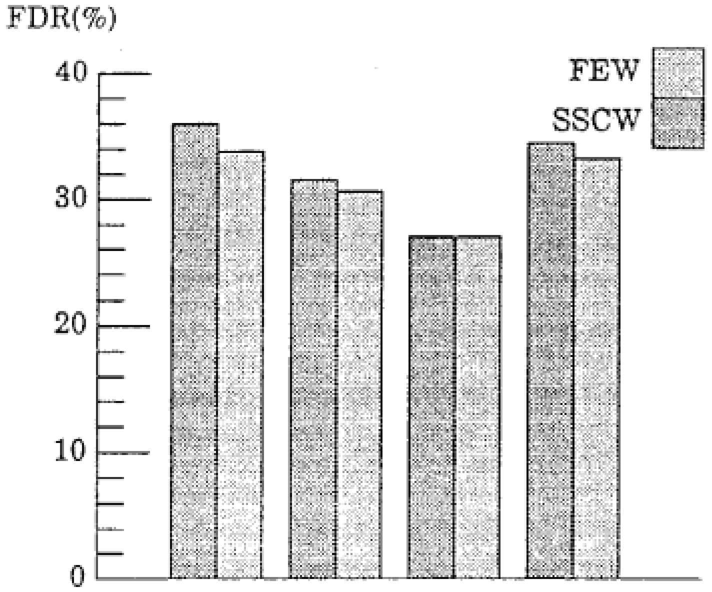
Because the FEW can be regarded as one of the SSCW methods, we can tune up (select) the hashing function for each application using the bit setting ratio of the field separated hash and *subrange*.

Other indexing scheme using a superimposed code word scheme have been proposed. Ramamohanarao and Shepherd [RS87] proposed an indexing scheme using a superimposed code word scheme. In this scheme, the descriptor is divided into two. One part is a code word that is computed by superimposing the hashed values of all functors in a terms. There is no information about the position of each functor in a term. The other part is used to indicate whether each position of a template term is a variable or functor (or nil). To retrieve a term, every descriptor is checked to see whether each functor of the query term occurs in the term or a variable occurs in the term in the same position as the functor.

There is another schema that concatenates each hashed value instead of superimposing them. Ohmori [OT88] has proposed an indexing scheme called



(a) Few variables in query terms



(b) Many variables in query terms

Figure 5 Effect of variables in SSCWs and FEWs

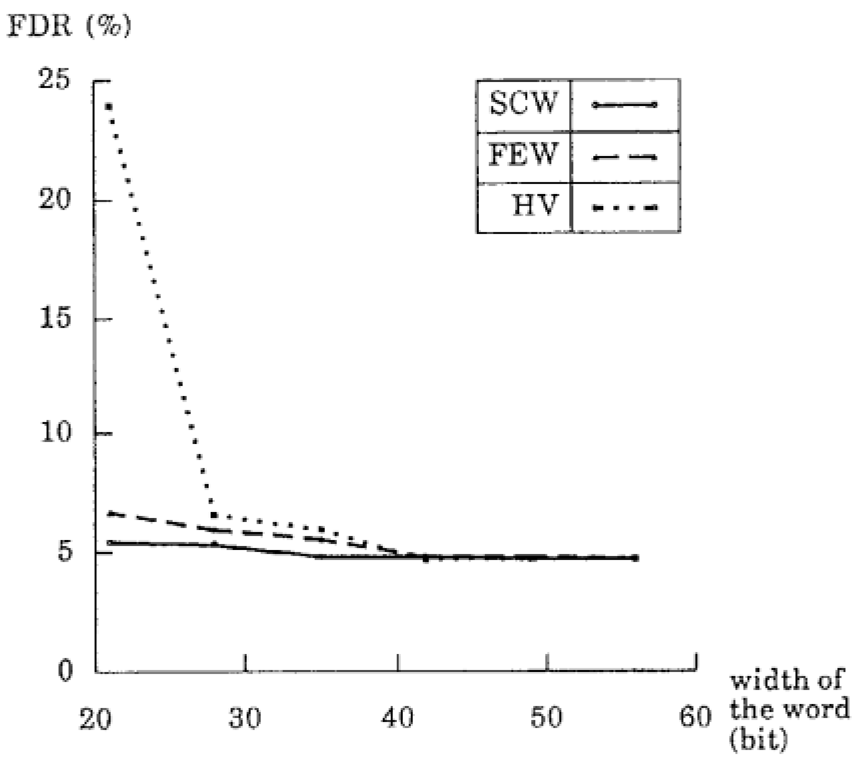


Figure 6 Code width and false drop ratio

the *hash vector* method. In this scheme, terms to be indexed are expanded first, according to a provided template term, and the hashed values of each functor and variable are concatenated. Henschen and Naqvi [HN81] have proposed an indexing scheme of terms for a theorem prover. Berra [Bea87] also described an indexing scheme.

Advantages of the above methods are efficient resolution capability of an index, and no interference of the hashed values of variables in the hashed values of functors.

There are two disadvantages. One is that their retrieval is rather complicated, that is, they need a logical operation with the same number of nodes as the template term The other is that the false drop ratio of such indexing methods is dependent on the figure of the template term, that is, if a functor is out of the template, the functor information is ignored.

An advantage of SSCWs and FEWs is simple retrieval. In an SSCW, a unifiable pair of terms can be checked by one logical operation which is provided in almost all micro-processors.

The SSCWs and FEWs can also flexibly determine the hash bit field of all nodes in the term, preventing nodes in a term set from being ignored as much as possible. Figure 3 shows the relationship between the code width and the false drop ratio of the hash vector (HV), FEWs and SSCWs for a sample case. In this case, there are many figures of a term that do not suit one template. Since code words are used efficiently in the FEW and the SSCW methods, the false drop ratio is not very large when the code width is small.

5 Summary

This paper described the SSCW method. This scheme provides fast term retrieval that retrieves an object (knowledge) which has a term unifiable with a specific query term. In this method, each term is encoded to code word efficiently and the object to be retrieved is pre-checked using its code word (descriptor) with a very simple operation.

This paper also briefly described the design of the code word, and proposed field separated hash to reduce the false drop ratio.

In future, we will use the SSCW method in an experimental knowledge base machine [MMISS88], and measure the performance of the index to terms.

Acknowledgments

We wish to thank Mr. N. Miyazaki and Mr. M. Wada of Oki Electric Industry Co., Ltd., and Mr. H. Sakai and Mr. S. Shibayama of Toshiba Corporation for many useful discussions. We also extend our thanks to the members of the KBM working group at ICOT for their helpful suggestions.

References

[Bea87] P.B. Berra et al., Computer architecture for a surrogate file to very large data/knowledge bases, *IEEE COMPUTER*, pp.25–32, March 1987

[HN81] L. Henschen and S. Naqvi,

A fast literal indexing scheme, in *Proceedings of the Seventh International Conference on Artificial Intelligence*, pp.528–529, Vancouver, British Columbia, August 1981.

[MMISS88] Monoi., H. et al, Parallel Control Technique and Performance of An MPPM Knowledge Base Machine, in *The Proceedings of the Fourth International Conference on Data Engineering*, Los Angeles, CA, pp.210–217, February 1–5, 1988

[MMNS87] Y. Morita, H. Monoi, A. Nakase, and S. Shibayama, A knowledge base machine with an MPPM (3) – an indexing scheme for terms –, in *Proceedings of 35th IPSJ Conference, 2C-7*, 1987 (in Japanese)

[MYNI86] Y. Morita, H. Yokota, K. Nishida and H. Itoh, Retrieval-By-Unification Operations on a Relational Knowledge Base, in *The Proceedings of the Twelfth International Conference on Very Large Data Bases*, Kyoto, Japan, pp.52–59, August 1986.

[MWI86] Y. Morita, M. Wada, and H. Itoh, Structure retrieval via the method of superimposed codes, in *Proceedings of 33th IPSJ Conference, 6L-8*, pp.1277–1278, 1986. (in Japanese)

[OT88] T. Ohmori and H. Tanaka, An Algebraic Deductive Database Managing a Mass of Rule Clauses, pp.660–673, *Database Ma-*

chines and Knowledge Base Machines, Kluwer Acdemic Publishers, 1988

[RS87] K. Ramamohanarao and J. Shepherd, *Answering Queries in Deductive Database Systems*, pp.1014–1033, Volume 2 of *Logic Programming*, The MIT Press, 1987

[Ros79] C. S. Roberts, Partial-Match Retrieval via the Method of Superimposed Codes, in *Proceedings of the IEEE*, Vol. 67 No. 12, pp.1624–1642, December 1979.

[WP87] M. J. Wise and D. M. W. Powers, Indexing Prolog clauses via superimposed code words and field encoded words, in *Proceedings of the IEEE Conference on Logic Programming*, pp.203–210, January 1987