TR-158

Retrieval-By-Unification Operation On a Relational Knowledge Base Model

by

Yukihiro Morita, Haruo Yokota
Kenji Nishida and Hidenori Itoh

February, 1986

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

Retrieval-By-Unification Operation on a Relational Knowledge Base Model

Yukihiro Morita, Haruo Yokota, Kenji Nishida and Hidenori Itoh

Institute for New Generation Computer Technology

Mita Kokusai Building,21F

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

February, 1986

Abstract

This paper describes a method for *retrieval-by-unification* (*RBU*) operations, especially *unification-join*, on a relational knowledge base model. The relational knowledge base model is a conceptual model for a knowledge base. In this model, knowledge is represented by *term relations*. Terms in the term relations are retrieved with operation called RBUs (i.e., *unification-join* and *unification-restriction*). To perform unification-join in the simplest manner, all possible pairs of tuples in term relations should be checked to see if each pair of terms in the tuples is unifiable or not. This would result in an extremely heavy processing load. We propose a method which involves ordering terms and, as result, omitting some pairs from this processing. The paper also describes a method for implementing the *unification engine* (*UE*), that is, hardware dedicated to the RBU operations.

1 Introduction

The Fifth Generation Computer Systems (FGCS) project in Japan aims to develop inference and knowledge base mechanisms to implement a knowledge information processing system. To create a large-scale system for knowledge information processing it is necessary to make a subsystem which efficiently manages and shares knowledge, like the database management system in data processing. In this paper, the machine that efficiently realizes the above subsystem is called a knowledge base machine. Development of a knowledge base machine is one of the goals of the four-year intermediate stage (1985 to 1988) of the project.

The knowledge base machine will be used by a variety of users and host computers, so a flexible conceptual schema is desirable. The relational knowledge base model suggested in [Yokota 85] is an extremely flexible conceptual model of a knowledge base. Knowledge is represented by term relations, which can include a set of Horn clauses or of semantic networks. However, the amount of processing by the machine becomes enormous when the RBU operation proposed in [Yokota 85] is performed in a simple manner.

This paper describes how to process the RBU operations. Section 2 provides necessary information on the relational knowledge base model and the RBU operations. Section 3 proposes an efficient method for processing the RBU operations. Finally, Section 4 introduces a method for implementing the unification engine(UE), that is, dedicated hardware for performing the RBU operation.

2 A Relational Knowledge Base Model

One reason why database systems have prospered is that sets of data can be shared by a number of applications as a result of the establishment of data independence based on data models. It is important for a knowledge base system to supply a number of applications with more complex structures than the data stored in databases. Thus, we must set up a knowledge model for uniformly treating knowledge among suppliers and users of the knowledge. We proposed a relational knowledge base model in [Yokota 85] as such a common model.

2.1 Basic Concept

The relational data model is suitable for treating sets of data mathematically. Let $U = \{A_1, A_2, \ldots, A_n\}$ be a set of attributes, and a domain $\mathbf{D}_i = dom(A_i)$ $(i = 1, \ldots, n)$. Formally, relation $\mathbf{R}(A_1, A_2, \ldots, A_n)$ on U is defined as follows:

$$\mathbf{R} \subseteq \mathbf{D}_1 \times \mathbf{D}_2 \times \ldots \times \mathbf{D}_n.$$

$\mu \in \mathbf{R}(A_1, A_2, \ldots, A_n)$ is called a tuple. If it is necessary to distinguish among the attributes the disjoint sets of attributes X and Y, we use the notation $\mathbf{R}(X, Y, \ldots)$. For example if $X = \{A_1, A_2\}$ and $Y = \{A_3, A_4, A_5\}$, the tuple $(x, y, \ldots)$ stands for $(a_1, a_2, a_3, a_4, a_5, \ldots)$.

However in the relational data model, domains are restricted to sets consisting contains of nothing but constants. In the relational knowledge base model, on the other hand, domains are expanded to sets of terms. A term is a kind of structure capable of containing a number of constants and variables. A subset of the Cartesian product of term domains $\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_n$ is called a *term relation*[Yokota 85] on U.

$$\mathbf{T} \subseteq \mathbf{K}_1 \times \mathbf{K}_2 \times \ldots \times \mathbf{K}_n$$

where $\mathbf{K}_i$ is a set of terms. $\mu \in \mathbf{T}(U)$ is also called a tuple (over U).

Assume that **Var** is a set of variables and **Fun** $_i$ $(i = 0, 1, 2, \ldots)$ is a set of i-place function symbols. $\cup_{i=0,1,2,\ldots}$**Fun** $_i$ is denoted by **Fun** . Elements of **Fun** $_0$ are called constants. We assume that **Fun** $\cap$ **Var** $= \phi$. Now, terms on **Fun** $\cup$ **Var** are recursively defined as follows:

1. Any constant $a \in$ **Fun** $_0$ and any variable $x \in$ **Var** are terms.

2. If $t_1, t_2, \ldots, t_n$ are terms and

   $f \in$ **Fun** $_n$ is an n-place function symbol, then $f(t_1, t_2, \ldots, t_n)$ is also a term.

3. All terms are generated by applying the above rules.

Let Term be a set of terms on Fun ∪ Var. A *substitution* θ : Var → Term is represented by a finite set of ordered pairs of terms and variables

$$\{ \{t_i/x_i\} \mid t_i \in \text{Term}, x_i \in \text{Var} \quad and \quad if \quad i \neq j \quad then \quad x_i \neq x_j \}$$

Applying a substitution θ to term t, we represent the resulting term by $t\theta$. $t\theta$ is called an instance of t.

A substitution θ is called a *unifier* for t_1 and t_2, if and only if $t_1\theta = t_2\theta$. We also say that t_1 and t_2 are *unifiable* when there is a unifier for them.

A unifier θ is said to be the *most general unifier* (*mgu*), if and only if for any unifier θ' of the set there is a substitution θ'' such that $\theta' = \theta \circ \theta''$, where $\circ$ is composition of substitutions. We write the mgu of t_1 and t_2 as $mgu(t_1, t_2)$.

A substitution θ is called a *simultaneous unifier* for the set of pairs of terms $\{(t_{i,1}, t_{i,2})\}_{i=1,\ldots,n}$ if and only if there is a substitution θ such that $t_{i,1}\theta = t_{i,2}\theta$ for all i.

A simultaneous unifier θ is said to be the *most general simultaneous unifier* (*mgsu*), if and only if for any simultaneous unifier θ' of the set there is a substitution θ'' such that $\theta' = \theta \circ \theta''$. The mgsu for the set of pairs of terms $\{(t_{i,1}, t_{i,2})\}_{i=1,2,\ldots,n}$ is denoted by $mgsu(\{(t_{i,1}, t_{i,2})\}_{i=1,2,\ldots,n})$.

2.2 RBU Operations

Data manipulation languages for relational databases are basically grouped into two types: relational algebraic languages and relational calculus-based languages. Relational calculus is non-procedural while relational algebra is procedural. Thus, it is easy for us to model real operations on data using relational algebra.

In the process of extending the relational data model to the relational knowledge base model, operations of conventional relational algebra, such as join and restriction, are extended to operations based on unification. In other words, equality-check operations between constants are enhanced to unification operations between terms. Thus (equi)join and restriction are extended to ***unification-join*** and ***unification-restriction***, respectively.

The projection of a (term) relation $\mathbf{T}(X, Y)$ over a set of attributes X is defined by $\mathbf{T}[X] = \{ x \mid \exists y\ (x, y) \in \mathbf{T}\}$.

Let w_1 and w_2 be attributes or terms, and μ be a tuple of a term relation. Let $tm(w_1, \mu)$ be defined as

$$tm(w_1, \mu) = \begin{cases} \mu[w_1], & \text{if } w_1 \text{ is an attribute;} \\ w_1, & \text{if } w_1 \text{ is a term.} \end{cases}$$

$w1 \diamond w2$ represents the condition for the tuple μ such that $tm(w_1, \mu)$ and $tm(w_2, \mu)$ are unifiable. Let F be a formula $\vee_{k=1,\ldots,n}(\wedge_{j=1,\ldots,m_k}(A_{i_{j,k}} \diamond w_{j,k}))$, where $A_{i_{j,k}}$ is an attribute and $w_{j,k}$ is a term or an attribute. The unification-restriction of term relation T, written $\sigma_F \mathbf{T}$, is defined as

$$\sigma_F \mathbf{T} = \{ \mu\theta \mid \exists k, \exists \mu \in \mathbf{T}, \theta = mgsu(\{(\mu[A_{i_{j,k}}], tm(w_{j,k}, \mu))\}_{j=1,\ldots,m_k})\}.$$

Let X, Y, Z, W be sets of attributes. The unification-join of $\mathbf{T}_1$ and $\mathbf{T}_2$, written $\mathbf{T}_1(X, Y) \underset{Y \diamond Z}{\bowtie} \mathbf{T}_2(Z, W)$, is defined as the term relation on $X \cup Y \cup W$:

$$\begin{aligned} \mathbf{T}_1(X, Y) \underset{Y \diamond Z}{\bowtie} \mathbf{T}_2(Z, W) = \{ \mu \mid \exists \mu_1 \in T_1, \exists \mu_2 \in T_2, \theta =\ & mgu(\mu_1[Y], \mu_2[Z]), \\ \mu[X'] =\ & \mu_1[X]\theta, \\ \mu[Y'] =\ & \mu_1[Y]\theta = \mu_2[Z]\theta, \\ \mu[W'] =\ & \mu_2[W]\theta \quad \}. \end{aligned}$$

Where $W \cap X = \phi$. We rename attributes, if needed. We call these operations *retrieval-by-unification* (*RBU* for short) operations[Yokota 85].

In the relational knowledge base model, knowledge is represented as term relations. *Term relations* stored in a certain format may be regarded as a set of Horn clauses (see Figure 1). [Yokota 85] showed that input resolution can be performed using RBU operations.

The relational knowledge base model is also expected to be capable of other types of knowledge representation such as frames and semantic networks. A common model to handle various types of knowledge representation is necessary to create a shared knowledge base. The relational knowledge base model is a promising candidate for such a model.

A mathematical foundation for formal semantics of relational knowledge bases was studied in [Murakami 85].

Figure 1 shows an example of term relations and RBU operations. Here X, Y, Z, W and $S \in$ **Var**, *ancestor* and *parent* $\in \mathrm{Fun}_2$ and *smith*, *clark* and *turner* $\in \mathrm{Fun}_0$. $\mathrm{KB1}\,(A_1, A_2)$ is an example of a term relation. $\mathrm{KB2} = \sigma_{A_1 \diamond [ancestor(smith,W)]}\mathrm{KB1}$, and $\mathrm{KB3} = \mathrm{KB2} \underset{A_2 \diamond A_1}{\bowtie} \mathrm{KB1}$.

3 A Processing Method for RBU Operations

The relational model provides users with a flexible data model, but it requires a large amount of processing. In particular, performing join operations with large relations requires a tremendous amount of computation. Several algorithms for implementing join have been proposed and studied [King 80][Merrett 83]. The Delta machine [Kakuta 85] employed dedicated hardware relational algebra engines to improve efficiency.

In the relational knowledge base model, unification-join processing is likely to generate very large computation loads. In this section, we propose a method to process RBU operations, especially unification-join, on the assumption that dedicated hardware will be used. In Section 4, we propose a method to realize this hardware.

3.1 Ordering of Terms

We assume that a very large amount of knowledge will be stored in the knowledge base machine. Thus, we assume these terms are stored in secondary storage

$ancestor(X,Y) \vee \neg parent(X,Y)$

$ancestor(X,Y) \vee \neg parent(X,Z) \vee \neg ancestor(Z,Y)$

$parent(smith, clark)$

$parent(clark, turner)$

$\vdots$

KB1

$[ancestor(X,Y)\|S]$	$[parent(X,Y)\|S]$
$[ancestor(X,Y)\|S]$	$[parent(X,Z), ancestor(Z,Y)\|S]$
$[parent(smith, clark)\|S]$	S
$[parent(clark, turner)\|S]$	S
$\vdots$	$\vdots$

KB2

$[ancestor(smith, Y)]$	$[parent(smith, Y)]$
$[ancestor(smith, Y)]$	$[parent(smith, Z), ancestor(Z,Y)]$

KB3

$[parent(smith, clark)]$	$[parent(smith, clark)]$	$[]$
$[ancestor(smith, Y)]$	$[parent(smith, clark), ancestor(clark, Y)]$	$[ancestor(clark, Y)]$

Figure 1. Example of term relations [Yokota 85].

```
      f
     / \
    g   a
   / \
  x   b
```

family order: $(f2)$ $(g2)$ (x) $(b0)$ $(a0)$

level order: $(f2)$ $(g2)$ $(a0)$ (x) $(b0)$

Figure 2. A tree and character strings representing term $f(g(x, b), a)$.

(e.g. moving head disk).

To perform unification-join in the simplest manner, all possible pairs of tuples in term-relations should be checked to see if each pair of terms in the tuples is unifiable or not. Generating all possible pairs, however, would result in extremely heavy processing loads. One way of preventing it involves ordering terms and, as a result, omitting some pairs.

To arrange terms in order, we introduced the concept of generality as follows[Yokota 85]:

Suppose t_1 and t_2 are terms. If t_2 is an instance of t_1, then t_1 is more general than t_2. That is,

$$t_1 \sqsupseteq t_2 \qquad iff \qquad \exists\theta \quad t_2 = t_1\theta \qquad (\theta \text{ is substitution.}) \ .$$

This generality order, however, is a partial order. All terms should be ordered thoroughly keeping the order of generality. Terms can be represented by trees (Figure 2), which can then be linearized to character strings. Since trees can be linearized in various ways, such as family-order and level-order methods[Knuth 73a], there are many character string representations. The character string representation of method m of term t is denoted by $rep_m(t)$. Note that each character corresponds to elements of **Var** $\cup$ **Fun**. The corresponding elements of **Var** $\cup$ **Fun** of the character c is denoted by $node(c)$.

The length of string s is denoted by $length(s)$. We write the substring from the i-th character through the j-th character of the character string s by $s[i;j]$, where $i \leq j$, and especially when $i > j$ it denotes the *null-string* ($length(s[1;0]) = 0$). The position of a first variable in character string c is denoted by $posv(s)$ and the position in which variable x appears first is denoted by $posv(s,x)$. If there are no variables in s then we define $posv(s) = length(s)$, and if there are no variables x in s then we also define $posv(s,x) = length(s)$. $c[1;posv(s)-1]$ is denoted by $prefv(s)$. We define $difpos(s_1,s_2)$ as

$$difpos(s_1,s_2) = \begin{cases} 1, & if\ s_1[1;1] \neq s_2[1;1] \text{ or,} \\ & s_1 \text{ or } s_2 \text{ is } \textit{null-string}, \\ n, & if\ s_1[1;n] = s_2[1;n], s_1[n;n] \neq s_2[n;n]\ (n \geq 2). \end{cases}$$

We define a lexicographic order of character string representations of terms as follows:

$s_1 > s_2$ if and only if $n = difpos(s_1,s_2)$, and

1. $node(s_1[n;n]), node(s_2[n;n]) \in$ **Var**,
   (a) $posv(s_1, node(s_1[n;n])) > posv(s_2, node(s_2[n;n]))$.
   (b) $posv(s_1, node(s_1[n;n])) = posv(s_2, node(s_2[n;n]))$,
   $s_1[n+1; length(s_1)] > s_2[n+1; length(s_2)]$.
2. $node(s_1[n;n]) \in$ **Var**, $node(s_2[n;n]) \in$ **Fun**.
3. $node(s_1[n;n]), node(s_2[n;n]) \in$ **Fun**,
   $node(s_1[n;n]) > node(s_2[n;n])$ in arbitrary order in **Fun**

We use this lexicographic order (of method m) to order the terms. That is,

$$t_1 \succ_m t_2 \qquad iff \qquad rep_m(t_1) > rep_m(t_2)$$

In this paper, the method of ordering of terms which corresponds to the family-order representation is called the *left-most* method. Similarly, the method of ordering of terms corresponding to the level-order representation, is called the *outer-most* method.

Note that, in both of the linearized methods, family-order and level-order, the 'father' node appears before it's 'son' nodes. So for substitution $\theta_1 = \{\{t_1/x_1\}\}$ and term t, let s_1, s_2 and k be $rep_m(t), rep_m(t\theta_1)$ and $posv(s_1, x_1)$, respectively. Then s_1 and s_2 have a common identical substring $s_1[1;k]$ and $node(s_1[k;k]) \in$ **Var**, $node(s_2[k;k]) \in$ **Fun** $\cup$ **Var**. So $rep_m(t) \geq rep_m(t\theta_1)$. (Note that when $node(s_2[k;k]) \in$ **Var**, $posv(s_1, x_1) = k$ and $posv(s_2, t_1) \leq k$.)

Therefore, for term t and substitution $\theta = \{\{t_i/x_i\}_{i=1,\ldots,n}\}$

$$rep_m(t) \geq rep_m(t\theta).$$

Where $rep_m(t) = rep_m(t\theta)$ hold when θ is a renaming substitution (i.e., $\theta = \{\{t_i/x_i\}\}$ then $t_i \in$ **Var** for all i and if $i \neq j$ then $t_i \neq t_j$). Thus both of these ordering methods maintain the order of generality (i.e., if $t_1 \sqsupseteq t_2$ then $t_1 \succeq t_2$). The order introduced in [Yokota 85] is an instance of the left-most method.

3.2 A Processing Method for Unification-Join

In a set of terms ordered in the above way (i.e., left-most method or outer-most method), character strings of terms which can be unified should have a common identical substring preceding a variable.

Let us consider a pair of terms t_1 and t_2. Suppose $s_1 = rep_m(t_1)$, $s_2 = rep_m(t_2)$, $k = \min(posv(s_1), posv(s_2))$, $p_1 = prefv(s_1)$ and $p_2 = prefv(s_2)$. If t_1 and t_2 are unifiable, then there exists $\theta = mgu(t_1, t_2)$ and $t_1\theta = t_2\theta$, so, $rep_m(t_1\theta) = rep_m(t_2\theta)$, and

$$rep_m(t_1\theta)[1;k] = s_1[1;k] = rep_m(t_2\theta)[1;k] = s_2[1;k].$$

That is,

$$s_1[1;k] = s_2[1;k] \qquad \text{where} \quad k = \min(posv(s_1), posv(s_2)).$$

INPUT : Two sets of terms T_1 and T_2.

OUTPUT: All the possible pairs of terms.

Step 1: Set $fs \leftarrow$ *null-string*, $class_i(k) \leftarrow \phi$ for $i = 1$ *or* 2 and $k = 0, 1, 2, \ldots$.

Step 2: Take a term $t \in T_i$, ($i = 1$ *or* 2), such that $t \succeq t'$ for all $t' \in T_1 \cup T_2$.

And let $T_i \leftarrow T_i - \{t\}$.

If there are no terms in $T_1 \cup T_2$, then Stop.

Step 3: Set $p \leftarrow prefv(rep_m(t))$, and $n \leftarrow difpos(p, fs)$.

Step 4: Let $class_1(k) \leftarrow \phi$, and $class_2(k) \leftarrow \phi$ for $k \geq n$,

Let $fs \leftarrow p$.

Step 5: Output all pairs (t, t'), where $t' \in class_j(k), 1 \leq k \leq n-1, j \neq i, (j = 1$ *or* $2)$.

Step 6: Let $class_i(length(p)) \leftarrow class_i(length(p)) \cup \{t\}$ and go to step 2.

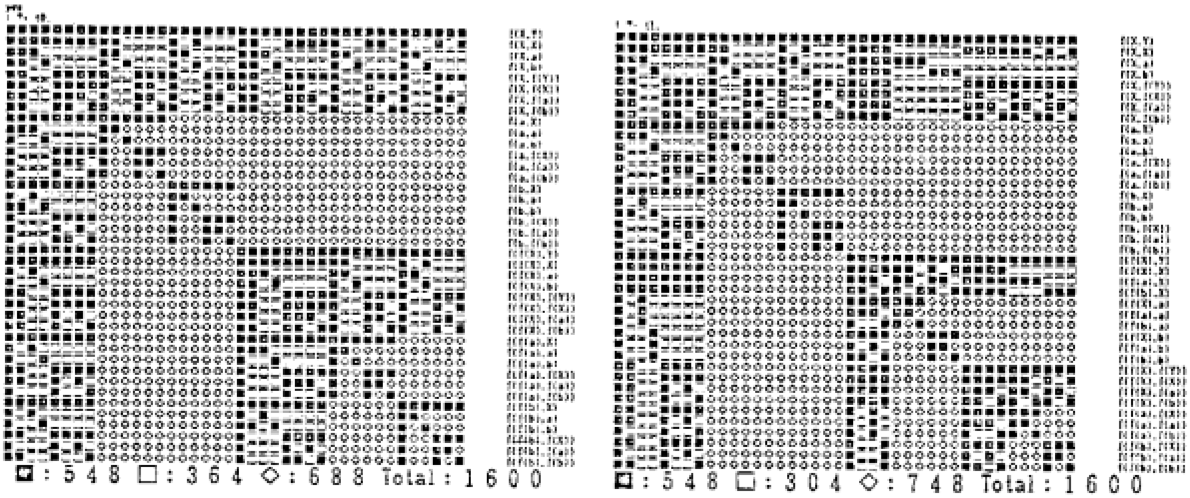
Figure 3. The pair generation algorithm.

Therefore, only such terms should be paired and checked as unifiable or not. When character strings are sorted, it is easy to select such pairs.

Figure 3 shows a pair generation algorithm. Here we maintain the set $class_i(k)$ as a set of terms such that $prefv(t)$ is $fs[1;k]$.

Samples of the generated pairs in each case (left-most method and outer-most method) are given in Figure 4. This algorithm generates term pairs indicated by the squares, and the pairs indicated by the diamond symbol are omitted. Black squares indicate unifiable pairs of terms.

In tree representation, variables appear only at leaves of tree. Since the level-order method lists the nodes from left to right, one level at a time, leaves appear later in $rep_{level\text{-}order}(t)$ than those in $rep_{family\text{-}order}(t)$. Since we check only $prefv(rep_m(t))$, the outer-most (level-order) method omits more pairs of terms than the left-most (family-order) method in general.



(left-most method) (outer-most method)

Figure 4. Example of combination of terms.

Unification-restriction can be achieved by using unification-join. For example, suppose $\mathbf{T}(A_1, A_2, A_3, A_4)$ is a term relation, $f = (A_1 \diamond t_1 \wedge A_2 \diamond A_3) \vee (A_3 \diamond a_1)$ and $X = \{A_1, A_2, A_3, A_4\}$, where t_1 and a_1 are terms and $x_{i,j}$ are variables. Let a term relation $\mathbf{T}'(A_1, A_2, A_3, A_4)$ be

$$\{(t_1, x_{1,2}, x_{1,2}, x_{1,4}), (x_{2,1}, x_{2,2}, a_1, x_{2,4})\}, \text{ then}$$

$$\sigma_f \mathbf{T} = \mathbf{T} \underset{X \diamond X}{\bowtie} \mathbf{T}'.$$

4 Design of the UE

In the initial stage of the FGCS project, we developed a relational database engine to be used in the relational database machine Delta[Kakuta 85][Sakai 84]. In the knowledge base machine, we also aim at improving efficiency by creating hardware dedicated to the RBU operation.

A relational knowledge base system architecture was proposed in [Yokota 85][Monoi 86]. Here we propose dedicated hardware called a *unification engine* (*UE* for short) for performing retrieval-by-unification operations as fast as possible.

The following describes a method to realize this dedicated hardware UE which is based upon the RBU processing method proposed in Section 3.

4.1 Unification Engine Configuration

The unification engine is dedicated hardware for retrieving terms from term relations. It processes data streams by pipeline processing.

Figure 5 shows a unification engine configuration. A unification engine uses three channels, two for input data streams to it and one for output data streams from it.

The unification engine consists of the following five units:

preprocess unit: This unit extracts an object item (term) from a tuple and sends out only that item to the sort unit.

sort unit: The sort unit sorts sets of terms into order.

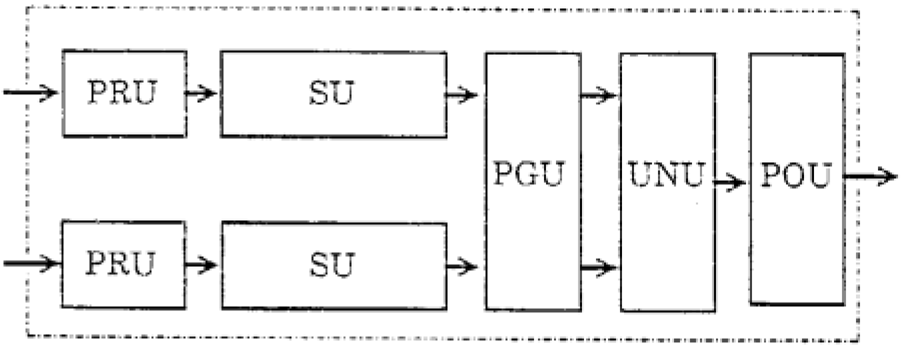
pair generation unit: This unit accepts two strings of sorted terms, then generates pairs of possibly unifiable terms.

unification unit: The unification unit obtains the most general unifier (mgu) of generated term pairs.

postprocess unit: The postprocess unit applies the mgu to the original tuples.

4.2 Sort Unit

Several sorting algorithms have been proposed and studied[Knuth 73b]. The relational database engine of Delta employed a sorter which adopted the two-way merge sort algorithm[Todd 77]. Since we number variables left-to-right in character strings of terms, we can obtain the lexicographic order by variable length character sort. So we adopt the variable-length two-way merge sort method. We use a TRIE representation of variable-length character strings to avoid readjusting



PRU :Preprocess Unit
SU : Sort Unit
PGU :Pair Generation Unit
UNU :Unification Unit
POU :PostProcess Unit

Figure 5. Unification engine configuration.

comparison starting points. This representation is used in a pipelined heap sorter proposed in [Tanaka 85].

4.3 Pair Generation Unit

The pair generation unit puts terms in the stack until all possibilities for unification are exhausted. Then, comparing these terms with the input term, the unit outputs all pairs of terms excluding irrelevant pairs of terms. This unit accepts the TRIE representation, so it is easy to compute the $difpos(p, fs)$ in Figure 3. Figure 6 shows the configuration of the pair generation unit.

The pair generation unit consists of five components, a comparator, two term-stacks, a functor-stack, a selector and an output buffer. The comparator sends out streams of two terms to be input one by one in order. Term-stacks store terms while possibilities of unification remain and the functor-stack indicates the order of current processing. The selector and the output buffer control send out streams

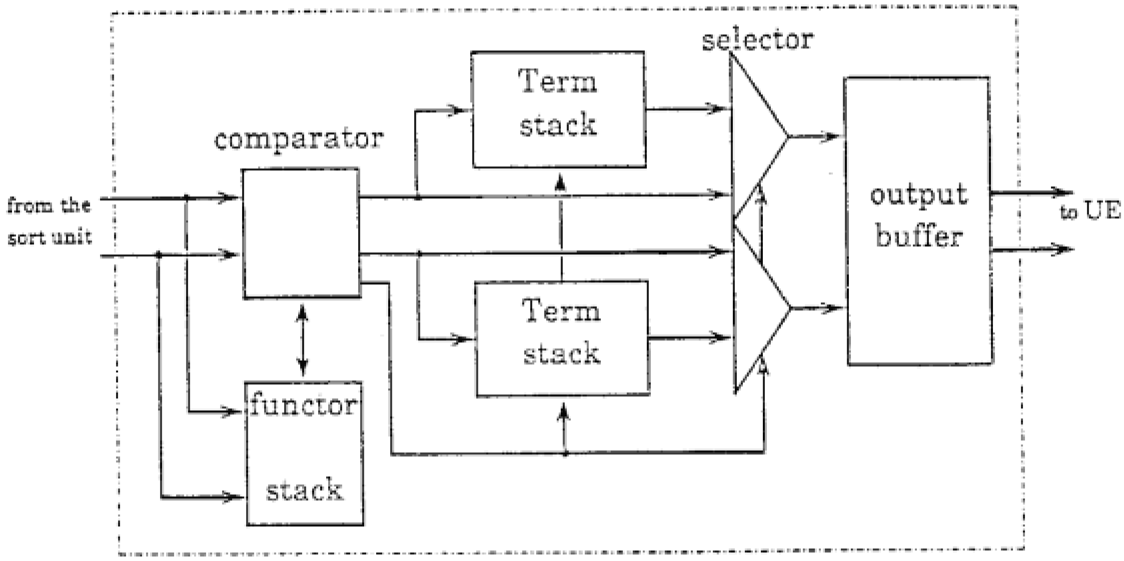


Figure 6. Pair generation unit configuration.

of pairs of terms to be output to the unification unit.

4.4 Unification Unit

Unification was first introduced by Robinson as a the basic operation of resolution. Several unification algorithms have been studied[Yasuura 85]. Most unification algorithms, structure shared methods or structure copy methods, use pointers to bind variables. However, pointers are not appropriate to data stream processing.

Figure 7 shows one of the basic unification algorithms[Chang 73]. Here W is a set of terms.

Let us consider the hardware for executing the repetition part of Figure 7 (step 2 → step 4) (we call this hardware a *unification element.*). Collecting unification elements in series (see Figure 8) allows pipeline processing of pairs of terms from the pair generation unit.

Figure 9 shows a configuration of the unification elements. The blocks corresponding to each 'step' in the algorithm processes character streams in the pipeline manner.

Step 1: Set $k = 0, W_k = W$, and $\theta_k = \varepsilon$.

Step 2: If W_k is a singleton, stop; θ_k is most general unifier for W.

Otherwise, find the disagreement set D_k of W_k.

Step 3: If there exist elements v_k and t_k in D_k

such that v_k is a variable that does not occur in t_k, go to step 4.

Otherwise, stop; W is not unifiable.

Step 4: Let $\theta_{k+1} = \theta_k\{t_k/v_k\}$ and $W_{k+1} = W_k\{t_k/v_k\}$. (Note that $W_{k+1} = W\theta_{k+1}$.)

Step 5: Set $k = k + 1$ and go to Step 2.

Figure 7. Unification algorithm [Chang 73]

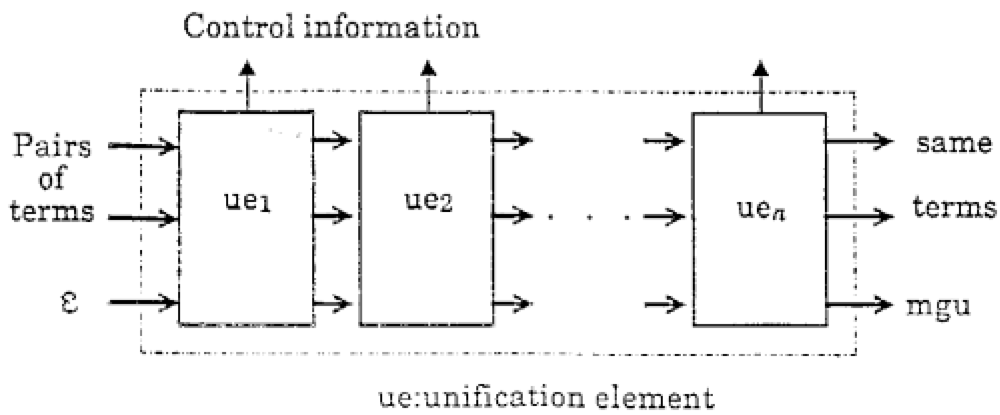
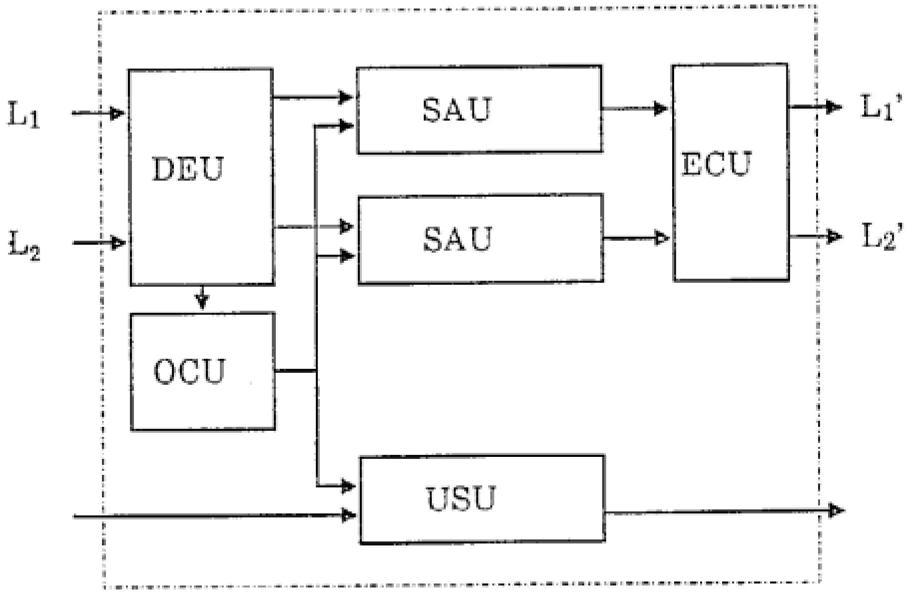
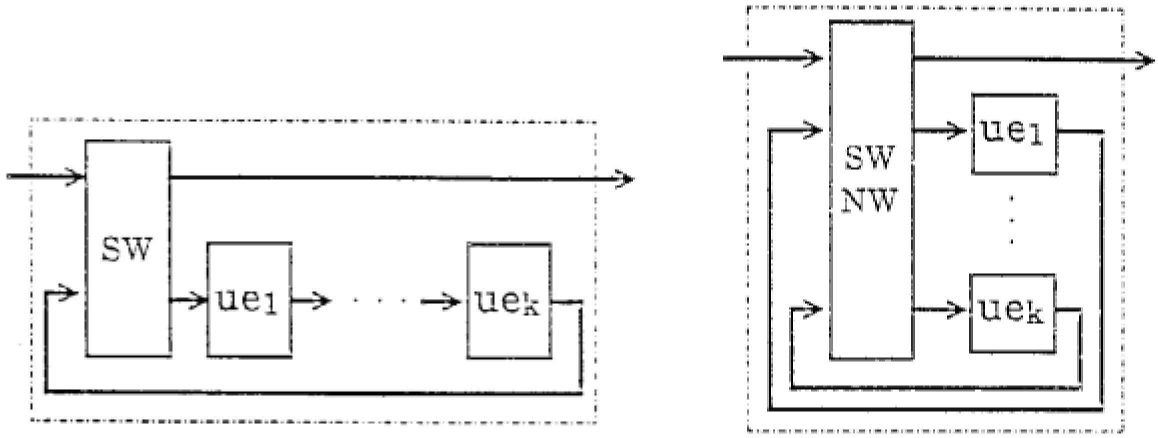


Figure 8. Hardware for unification algorithm.



DEU: Disagreement Extract Unit
OCU: Occurrence Check Unit
SAU: Substitution Apply Unit
USU: Unifier Synthesize Unit
ECU: Equalty Check Unit

Figure 9. Unification element configuration.



SW :circuit changing switches
SWNW :switching network
ue_i :unification element

Figure 10. Unification unit configurations.

If there are no limits set on the number of variables in terms, then an infinite number of unification elements would be required. This problem is easily solved by a modification of the configuration using circuit changing switches or a switching network as shown in Figure 10.

5 Summary

In this paper, we proposed an RBU operation processing method and a method for implementing it. Our unification engine applies not only to knowledge base machines, but also to other knowledge information processing systems. The ordering of terms proposed in Section 3 also can be used for a disk clustering method, a kind of page indexing method, to narrow the search space. In the future, we plan to evaluate the proposed algorithm and engine by means of simulation.

ACKNOWLEDGEMENTS

The authors thank Mr. K. Yokota of ICOT research center, Mr. H. Sakai of Toshiba Corporation and the members of KBM (Knowledge Base Machine) working group at ICOT for many useful discussions.

References

[Chang 73] Chang, C. L., Lee, R. C., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[Kakuta 85] Kakuta, T., Miyazaki, N., Shibayama, S., Yokota, H., and Murakami, K. "The Design and Implementation of Relational Database Machine Delta", *Proceedings of the International Workshop on Database Machines'85*, March 1985.

[King 80] King, W. F., "Relational Database Systems: Where We Stand Today", *IFIP*,pp.368-381,1980.

[Knuth 73a] Knuth, D. E., *The Art of Computer Programming*, Vol. 1, *Fundamental Algorithms*, second edition, Addison-Wesley, 1973.

[Knuth 73b] Knuth, D. E., *The Art of Computer Programing*, Vol. 3, *Sorting and Searching*, Addison-Wesley, 1973.

[Merrett 83] Merrett, H. T., "Why Sort/Merge Gives the Best Implementation of the Natural Join.", *ACM SIGMOD Record* **13**, No **2**, January 1983.

[Monoi 86] Monoi, H., Yokota, H., Murakami, M., and Itoh, H. "A Large-Scale Knowledge Base Machine Control Technique Using Multi-Port Page-Memory", Submitted to *the 12th International Conference on VLDB* August 1986.

[Murakami 85] Murakami, M., Yokota, H., Itoh, H. "Formal Semantics of a Relational Knowledge Base", *ICOT Technical Report* No. TR-149., December 1985.

[Sakai 84] Sakai, H., et. al., "Design and Implementation of the Relational Database Engine." *ICOT Technical Report* No. TR-063., April 1984, *Proceedings of International Conference on Fifth Generation Computer Systems*, November 1984.

[Tanaka 85] Tanaka, Y. "A VLSI Algorithm for Sorting Variable-Length Character Strings", *New Generation Computing* 3, No. 3, pp. 273-306, 1985.

[Todd 77] Todd, S., "Algorithm and Hardware for a Merge Sort Using Multiple Processors", *IBM Journal of Research and Development*, 22 ,1977.

[Yasuura 84] Yasuura, H., "On Parallel Computational Complexity of Unification", *Proceedings of International Conference on Fifth Generation Computer Systems* 1984.

[Yasuura 85] Yasuura, H., Ohkubo, M., and Yajima, S., "A Hardware Algorithm for Unification in Logic Programming Language", *Technical Report of IECE*, EC84-67, pp9-20, March 1985, in Japanese.

[Yokota 85] Yokota, H., and Itoh, H. "A Model and Architecture for a Relational Knowledge Base", *ICOT Technical Report* No. TR-144, To appear in *Proceeding of the 13th International Symposium on Computer Architecture,* November 1985.