ICOT Technical Report: TR-141

TR-141

A Model and an Architecture for a Relational Knowledge Base

by
Haruo Yokota and Hidenori Itoh

November, 1985

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 Model and an Architecture for a Relational Knowledge Base

Haruo Yokota, Hidenori Itoh

ICOT Research Center
Institute for New Generation Computer Technology
Tokyo, Japan

November 1985

ABSTRACT

A relational knowledge base model and an architecture which manipulates the model are presented. An item stored in the relational knowledge base is a term, and it is retrieved by unification operation between the terms. The relational knowledge base architecture we propose consists of a number of unification engines, several disk systems, a control processor, and a multiport page-memory. The system has a knowledge compiler to support a variety of knowledge representations.

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. In the first three-year (1982-84) stage of the project, we developed a relational database machine *Delta* [Kakuta et al. 84] and several sequential inference machines to investigate techniques for implementing these mechanisms. We are now in the intermediate four-year (1985-88) stage of the project, and plan to develop prototypes of knowledge base machines and parallel inference machines. In this paper, we propose a knowledge model and an experimental knowledge base architecture which manipulates the model.

The strongest motivation for knowledge base systems seems to be faster retrieval from a large amount of knowledge. The objects retrieved as knowledge items may take a lot of different forms. Knowledge base systems have to be able to treat this variety of knowledge objects uniformly. We expect we can represent these objects using a set of terms, well-defined structures capable of containing a number of variables. Thus, the system described here can quickly retrieve these terms. From the database point of view, if we assume the knowledge base system to be an enhanced database system, the improvement is in the structure of items

in each system. We think an atom, an item treated in conventional database systems, is not sufficient for representing object worlds. We need more flexible structures. We choose the term as the item to be treated in the knowledge base system.

We propose to use a unification operation to retrieve the terms and to develop unification engines as dedicated hardware for that operation. Since the number of terms manipulated in knowledge base systems seems to become very large, we assume these terms are stored in secondary storages (i.e. moving head disks.) When we make a system handle a large amount of data stored in secondary storage systems, the problem arises that there is a data processing bottleneck between processing elements and the secondary storage systems. We present an architecture for providing a wide data stream path between the unification engines and the secondary storage systems. We expect we can use many technologies obtained through the development of *Delta*. We have developed relational engines to manipulate data streams. Techniques for handling data streams and execution controls for the multi engines that manipulate data streams may be used in the knowledge base system. Moreover, there are several methods for efficient data retrieval in databases, e.g., hashing, indexing, clustering. These methods can be applied to the knowledge base system. Our adoption of a clustering method for *Delta* particularly disposes us to use similar techniques.

There are several other approaches to implementing a knowledge base system. A deductive database system consisting of a Prolog processor and a relational database management system [Gallaire et al. 84, Yokota et al. 85] is one approach. A Prolog machine manipulating a large amount of data [Sabbatel et al. 84] is another approach. However, we think our approach is the most flexible and can do what the other approaches can do. We show, in this paper, that resolutions can be performed if we store Horn clauses in the knowledge base as a collection of terms. Note that the resolution is merely one of the applications using the knowledge base operations. The main objective of the architecture is to retrieve terms from large scale knowledge bases as fast as possible.

This paper is organized as follows. In Section 2 we propose a knowledge model for a flexible knowledge base system. We show that resolutions can be performed in the knowledge base system as an application and consider the adaptability for other knowledge representations. Section 3 presents an architecture which manipulates the model. We briefly describe data structure, each basic hardware component and the software configuration of the system. The observation and discussion are summarized in Section 4.

2. Knowledge Model

The reason why database systems have prospered is not only that the amount of data stored in them has increased, but also that sets of data have become useable by a number of applications as a result of establishment of data models. It is also significant 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 a knowledge model for uniformly treating knowledge between suppliers and users of the knowledge.

2.1 Basic Concept

The relational model is suitable for treating sets of data mathematically. However, each item in relational databases only represents an element of a relation defined on a finite domain which consists of nothing but constants. An item is retrieved merely by using equality-check operations between constants. The item is never recognized as structured data.

We propose to introduce terms as objects of relational operations to treat knowledge. A term is a kind of structure capable of containing a number of variables. We introduce a unification operation between terms as a basic operation to retrieve terms, instead of the equality-check operations. Since a term represents a relation defined on an infinite domain containing structures, complex structures of knowledge can be handled. We call sets of relations containing terms *relational knowledge bases*.

Formally, an object treated in relational databases is represented as follows:

$$d \in R \subset D_1 \times D_2 \times \ldots \times D_n \qquad (D_i \text{ is a set of constants}).$$

R is called a relation. An object treated in relational knowledge bases is

$$k \in T \subset K_1 \times K_2 \times \ldots \times K_n \qquad (K_i \text{ is a set of terms}).$$

We call T a *term relation*. As a relation has its name, so a term relation has its name. Terms are defined recursively as follows:

i. Constants and variables are terms.

ii. If f is an n-place function symbol, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

iii. Terms are generated merely by applying the above rules.

Data manipulation languages for relational databases are basically grouped into two types: relational algebraic languages and relational calculus languages. Relational algebra is a procedural system, while relational calculus is non-procedural one. Therefore, it is easy for us to

image real operations for data using relational algebra. We enhance relational algebra as a knowledge manipulation language to treat term relations. Note that the relational knowledge model is a collection of extensions of terms, i.e., a set of ground instances of the terms, while enhanced relational algebra for relational knowledge base is a language manipulating the term relations which represent the model, i.e., the enhanced algebra treats even non-ground terms.

The basic operations of relational algebra are the projection, join, and restriction operations. We now consider an enhancement for these operations. Since the projection operation does not treat each element of sets, the operation needs no change. For the join and restriction operations, equality-check operations between the elements are enhanced to unification operations between them. In the strict sense, a unification operation searches for a term unifiable with the search condition and derives the most general unifier between the condition and term and applies the unifier to both the condition and term. Thus, we introduce two new operations: *unification-join* and *unification-restriction*.

From the operational point of view, these operations retrieve terms from the knowledge represented by term relation using unification. We call these operations *retrieval-by-unification* operations.

2.2 Resolution using Retrieval by Unification

As an application of relational knowledge base operations, we show that resolutions are performed by storing a set of definite clauses† into a term relation which has two attributes and iterating the unification-join and unification-restriction operations. A tuple in the term relation corresponds to a definite clause. The first attribute of the term relation corresponds to the head portion of the clauses and the second one corresponds to the body portion††.

Both head and body portions are represented by binary trees to proceed with resolution only iterating retrieval-by-unification operations. The leaves of the tree correspond to literals contained in each portion. A common variable is put at the last leaf of both the head and body trees. If the clause is a unit clause, the body tree is just the variable put at the last leaf of the head tree.

A definite clause

$$P \vee \neg Q_1 \vee \neg Q_2 \vee \ldots \vee \neg Q_n$$

is represented using the two binary trees illustrated in Figure 1. In the Figure, S indicates the

† Horn clauses excluding negative clauses (queries) are called definite clauses.

†† We call a positive literal in a definite clause the *head portion* and a set of negative literals in the clause the *body portion*.

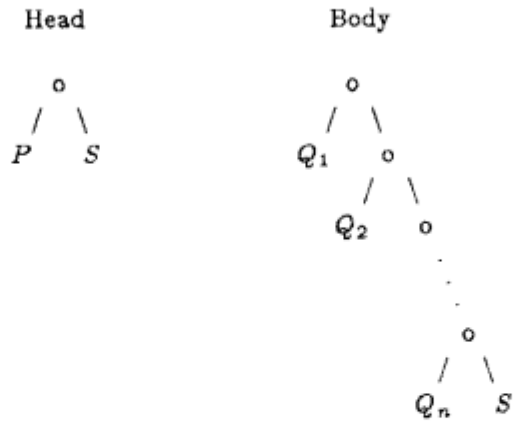


Figure 1. Horn clause representation in term relation

common variable between two portions.

First of all we show the resolution process using an example. As an example of knowledge, there are rules which indicate that one's parent is one's ancestor and that the ancestors of the parent are also one's ancestors. And there are facts which indicate each instance of the relation between children and their parents. The following definite clauses represent the knowledge.

$$ancestor(X,Y) \vee \neg parent(X,Y)$$
$$ancestor(X,Y) \vee \neg parent(X,Z) \vee \neg ancestor(Z,Y)$$
$$parent(smith, clark)$$
$$parent(clark, turner)$$
$$\vdots$$

These rules and facts are stored in the term relation illustrated in Figure 2. We use LISTs of

kb_1

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

Figure 2. An example of a term relation

DEC-10 Prolog to denote the trees; $[a|b]$ stands for $.(a, b)$ and $[a, b, c]$ stands for $.(a, .(b, .(c, [\,])))$, here $[\,]$ means nil.

When we search the relational knowledge base for ancestors of *smith*, we first invoke the unification-restriction operation for kb_1 by the condition

$$\textit{the first attribute} \diamond [ancestor(smith, A)]$$

and make a new term relation $temp_1$ (Figure 3.) Throughout this paper we use $A \diamond B$ for denoting a unification between A and B. If there is a most general unifier between A and B, the value of this expression is true and the unifier is applied to A and B.

Next, we invoke the unification-join operation between the second attribute of $temp_1$ and the first attribute of kb_1, and invoke the projection operation for the result relation of the unification-join to derive the first attribute of $temp_1$ and the second attribute of kb_1 and make the new term relation $temp_2$ (Figure 4.)

The first attributes of temporary term relations generated during resolution processes are used for leaving variable substitutions for a goal clause. The second attributes of tuples in these term relations correspond to resolvents derived by input resolutions between a goal clause or former resolvents and applicable input clauses. The top of the list stored in the second attribute of a tuple corresponds to one of the literals resolved upon. If no tuples unifiable with the tuple exist in the given term relation, it indicates a failure of resolution for the tuple. If the list is an empty list, it indicates the empty clause is derived, i.e., a refutation is derived by the variable substitutions indicated by the first attribute.†

$temp_1$	
$[ancestor(smith, A)]$	$[parent(smith, A)]$
$[ancestor(smith, A)]$	$[parent(smith, Z), ancestor(Z, A)]$

Figure 3. The result of the unification-restriction

$temp_2$	
$[ancestor(smith, clark)]$	$[\,]$
$[ancestor(smith, A)]$	$[ancestor(clark, A)]$

Figure 4. The result of the unification-join and projection

† The terminology for logic used here, e.g., resolvent, input resolution, literals resolved upon, refutation, is in [Chang and Lee 73].

$R \leftarrow \emptyset$

$T_0 \leftarrow \sigma_{head \diamond goal}(T)$

$i = 0$

while $T_i \neq \emptyset$ **do**

begin

$R \leftarrow \sigma_{body=[]}(T_i) \bigcup R$

$T_{i+1} \leftarrow \pi_{T_i.head, T.body}(T_i \underset{body \diamond head}{\rhd\lhd} T)$

$i \leftarrow i + 1$

end

Figure 5. Resolution process using retrieval-by-unification

In the above example, the first tuple in $temp_2$ is an answer, since it has an empty clause as a resolvent. This tuple indicates that *clark* is an ancestor of *smith*. The second attribute of the second tuple in the $temp_2$ is a subgoal to search for ancestors of *clark*, and it indicates that ancestors of *clark* are ancestors of *smith*.

Continuing the coupled operations, the unification-join and projection, we can derive all answers for the query, since the input resolution is complete for Horn clauses. If all items in the second attribute of a temporary term relation are empty lists or lists which cannot unify with any tuples in the given term relation, the next term relation generated by the coupled operations has no tuples. We can see the termination point of the resolution process by this an absence of tuples. If the resolution process does not generate infinite terms, the process must be terminated. Needless to say, if the set of Horn clause is function-free, it is terminated. Since resolvents are necessary to derive answers, absence of resolvents indicates all answers are derived already. We can collect the answers by searching tuples whose second attributes are empty lists for all temporary relations generated during the resolution process.

We show the resolution process in Figure 5. R is a term relation to eliminate the answers. T is a given term relation and T_is are temporary term relations. In the above example, $temp_1$ corresponds to T_0 and $temp_2$ corresponds to T_1. Symbols, σ, $\bigcup$, π, and $\rhd\lhd$ respectively denote restriction, union, projection, and join operation.

Since this method derives all resolvents in a level of search tree at once, it implements a breadth-first search. Variables put in the last leaf of binary trees implement stuck operations for each search tree node. Since this method replaces the literals of subgoals, it also seems to implement a string reduction. When we focus on functions, the retrieval-by-unification

operations provide a set-oriented Prolog meta-predicate, such that *clause* or *call* predicates.

2.3 For Other Knowledge Representations

First-order logic is one choice for representing knowledge, but there are several other knowledge representations, e.g., production systems, semantic network systems, frame systems. We think the relational knowledge base model is applicable not only to first-order logic but also to other knowledge representations.

A production system consists of a short term memory representing a state and a long term memory collecting a set of production rules. A production rule is composed of an if-condition portion and a then-action portion and is used for rewriting short term memory. The production rules are regarded as a collection of terms. The contents of short term memory can be written with terms. Thus, the production system can be implemented with term relations.

A semantic network system consists of nodes and arcs. Operations on the network are basically to search for desired nodes. These nodes and arcs are represented using terms [Kowalski 79, Koyama 85]. Retrieval by unification is an operation to search for terms.

A frame system consists of a set of tables. Each row of the tables are either structures or pointers. The relational knowledge base system is regarded as a collection of tables, and the pointer operation can be performed with inter-relation operation between the term relaions.

3. Architecture Overview

We now consider an architecture which manipulates the above relational knowledge base model. It is important for the system to retrieve a large amount of terms fast. Disk systems have to be used for storing such huge set of terms. We propose dedicated hardware for the retrieval-by-unification operation to derive a desired set of terms as fast as possible. We call the hardware a *unification engine*. Since we assume disk systems, the unification engine manipulates a data stream from and to the disk systems. We propose to use a *multiport page-memory* [Tanaka 84] for supplying buffer and working area between a number of unification engines and disk systems to transfer the data without access conflict or suspension.

We use a kind of clustering method for storing sets of terms into the disk systems to reduce the amount of data stream processed by the unification engines. We introduce an ordering of terms to adopt the clustering method. The ordering is also used for the efficient operation in the unification engines.

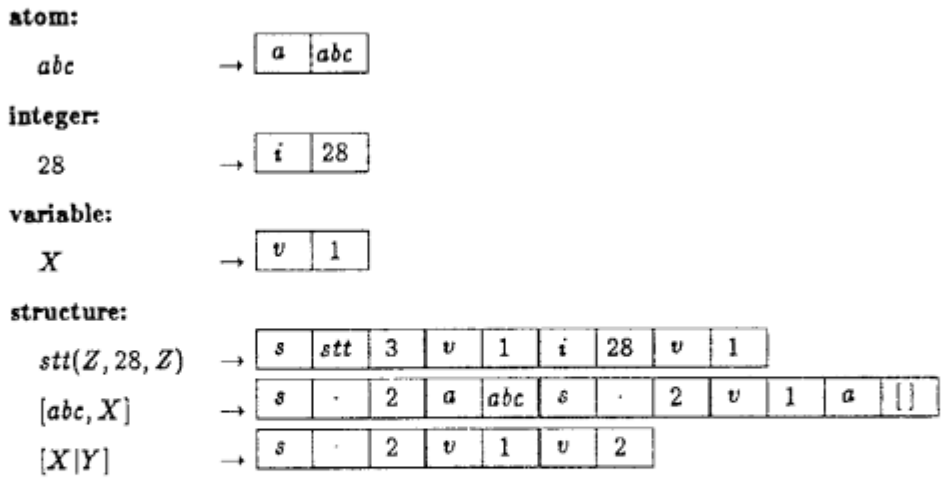


Figure 6. Argument Forms

It is important for the knowledge base architecture to support a variety of knowledge representations. We choose term relations as a base of these knowledge representations. The knowledge base system has a *knowledge compiler* to translate this variety of knowledge representations to the term relations.

3.1 Data Structure

We first briefly describe data structure manipulated in the experimental knowledge base system.

3.1.1 Argument Forms

The knowledge base system must handle four data types depending on the term's definition: atoms, integers, variables and structures. There are other data types, e.g., real numbers, boolean. We use only the previous four data types in the system because it is an experimental system. Both atoms and integers are constants in the terms definition and can be treated the same way, but we distinguish between them for introducing arithmetic operations into the system.

The system uses tags for distinguishing each data type. An atom is merely a character string with a tag. An integer consists of a tag and a fixed-length binary string. A variable can be represented using a sequential number with a tag, since there are no global variables in term relations and a variable is used to relate the argument to other arguments in one tuple. A structure consists of a tag, a function symbol, an arity (number of arguments), and a list of

arguments. Structures are recursively represented in this way. Besides using this representation, we can use pointers for the structures. However, since pointers are not appropriate to data streams processing, we bring the arguments into line. We illustrate the data structure in Figure 6.

3.1.2 Ordering of Terms

We introduce an ordering of terms to manipulate terms efficiently. Since the system retreives terms with unification operations, it is better for the system to let the term be ordered based on generality. Let both T_1 and T_2 be terms. If T_1 is an instance of T_2, T_2 is greater than T_1. That is,

$$T_1 \leq T_2 \qquad iff \qquad \exists \theta \quad T_2\theta = T_1$$

Since this ordering does not generate a total order, we introduce the following rules to line terms. Here, S^n stands for n-tuple terms $s_1, \ldots, s_n$.

(a) $f(S) < g(T) \quad \Leftrightarrow f < g$

(b) $f(S^n) < f(T^m) \quad \Leftrightarrow n < m$

(c) $f(S^n) < f(T^n) \quad \Leftrightarrow (s_1, \ldots, s_{i-1}) \approx (t_1, \ldots, t_{i-1}), s_i < t_i \quad (1 \leq i \leq n)$

(d) $f(S) < X \quad \Leftrightarrow X$ is variable

(e) $X < Y \quad \Leftrightarrow$ or $\begin{cases} X \text{ has already appeared but } Y \text{ has not.} \\ X \text{ and } Y \text{ have appeared but } X \text{ has appeared before } Y. \end{cases}$

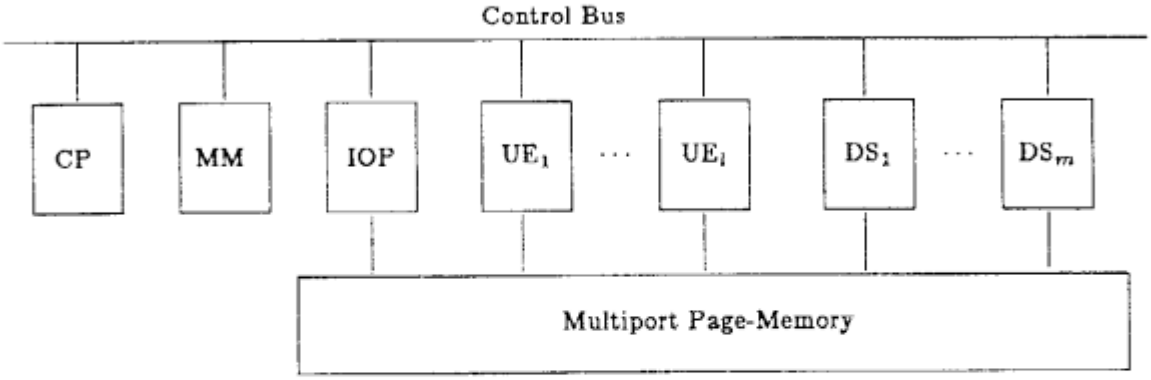
(f) $X \approx Y \quad \Leftrightarrow$ or $\begin{cases} X \text{ and } Y \text{ have not appeared yet.} \\ X \text{ and } Y \text{ have appeared in corresponding positions.} \end{cases}$

(g) $f(S^n) \approx f(T^n) \quad \Leftrightarrow (s_1, \ldots, s_n) \approx (t_1, \ldots, t_n)$

Since we number variables left-to-right in one term, we can obtain this ordering by scanning characters in terms left-to-right. There are other rules for ordering terms, e.g., those based on the nested levels.

3.1.3 Clustering

When the system searches objects with a wide search space, some search methods are necessary to obtain these objects rapidly. Conventional database management systems are adopting indexing and hashing techniques. Most of these database management systems are implemented on general-purpose computers and indexing and hashing techniques are appropriate to their architectures which retrieve tuples one by one. The knowledge base system is not implemented on general-purpose computers and it uses unification engines handling data streams. The unification engine can be seen as a filter for data streams from secondary storage. However, it is not efficient to let all data streams flow. We propose to use a clustering method, a kind of



CP: Control Processor
MM: Main Memory of CP
IOP: I/O Processor for CP
UE_i: Unification Engine
DS_i: Disk System (Consists of a Disk Controller and Disk Units)

Figure 7. Hardware Configuration

page indexing method, to narrow the search space.

The clustering method chooses physical pages which contain terms unifiable with a search condition. Then the data streams for unification engines can exclude irrelevant pages. The system sorts all terms depending on the above ordering method and stores them into pages while making page indices to select relevant pages.

3.2 Hardware Configuration

The main components of the experimental relational knowledge base system are a set of unification engines, a set of disk systems, a control processor and a multiport page-memory. These unification engines and disk systems are connected with the multiport page-memory (Figure 7.) The main focuses of this architecture are the wide bandwidth between the unification engines and disk systems and the parallel execution control among these components. The disk systems generate data streams and send them to the multiport page-memory. The unification engines get the data streams from the multiport page-memory, process them using a kind of pipeline method, and send the result data streams to the multiport page-memory, again. The results are stored in the disk systems, or used by the unification engines again, or output to the user.

The control processor controls these data flows and the parallel execution environment

among these components. It is possible to send control commands to the unification engines and disk systems and receive responses from them with multiport page-memory. However, since multiport page-memory is designed to transfer data several pages at a time, it is not suitable for transferring short information quickly. We use a control bus to send the commands or to receive the responses. The main memory and I/O processors of the control processor and the like are connected to the control bus. One of the I/O processors is directly connected with the multiport page-memory. When the control processor needs to access the data which is in the multiport page-memory, the control processor handles it via the I/O processor. Another I/O processor is used for interface host machines. We omit such unimportant components from Figure 7.

3.2.1 Unification Engine

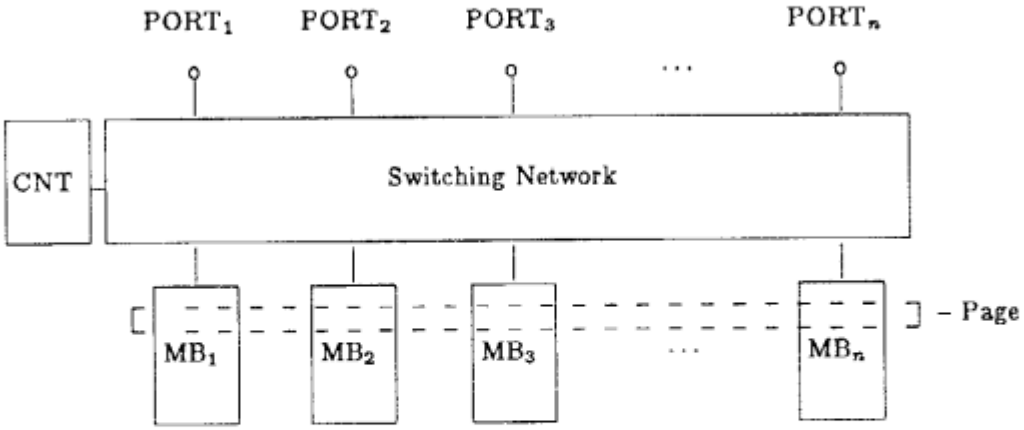
The unification engine is dedicated hardware for retrieving terms from term relations. It processes data streams in the pipeline way, while it gets the data stream from the disk systems and puts it into the disk systems. A unification engine uses three channels to connect the multiport page-memory. Two of them are used for reading data from the multiport page-memory. The remaining one is used for writing results into the multiport page-memory.

To sort the data stream before unification-join or unification-restriction operation renders the operation efficient. We introduced the ordering of terms depending on generality. If the engine sorts the data stream on the above ordering, the engine omits the irrelevant combination of terms.

3.2.2 Multiport Page-Memory

The multiport page-memory has a set of I/O ports, a set of memory banks and a switching network for connecting these ports and memory banks (Figure 8.) A logical page must be horizontally allocated to all memory banks to make the page simultaneously accessible from a number of ports. The connection between a port and a memory bank is fixed in one-to-one for a time period and changed by the system clock to inhibit access conflicts between the memory banks. If we assume that the access unit is a page and the page can be started accessing from its middle position, no ports have to wait to access the same page [Tanaka 84].

When we implement the multiport page-memory, it becomes a problem of implementation that the bus width between the ports and memory banks will be wide. We propose to use the data bus for transporting the address to make it as narrow as possible. Since it is ineffective to transport the address for each data, however, we set the access unit a page or a set of page



CNT: Controller

MB_i: Memory Bank

Figure 8. Multiport Page-Memory Configuration

and the address calculation is performed in the memory banks.

Since the logical page is divided into all memory banks, rearrangement of the switching network connection should be synchronized with the address transportation. Each memory bank has a shift register for transporting the page addresses. The shift registers in the neighbor memory banks are connected. The rearrangement of the switching network connection is to shift the connection for the neighbor memory bank in the same direction as the shift register.

Each memory bank has a latch to hold the page address while the memory bank transports the page address. Then the memory banks transport the page address independent of memory accesses from the port. We must fix an accesss unit for the memory banks to access without the rearrangement while at least the addresses are transported.

3.2.3 Control Processor

The main role of the control processor is general control of the whole knowledge base system, especially parallel execution control among the unification engines and disk systems. The important feature of the control portion of a functionally distributed system, like this system, is rapid responses. The control processor must have a wide control bus and real-time operating system. A general-purpose processor may be used for the control processer, while a kind of inference machine can be used. Since the control processor has to have a knowledge compiler, it seems to be better for it to use the inference machine.

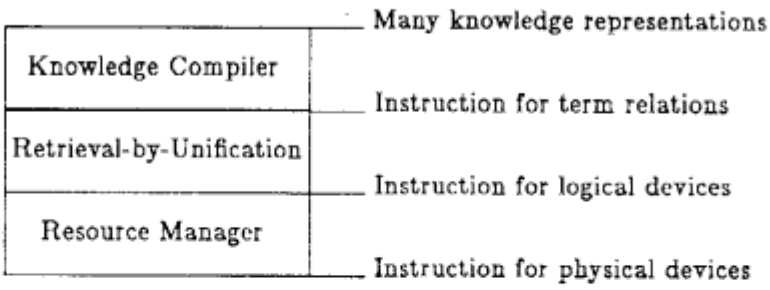


Figure 9. Software Layers

3.3 Software Layers

A software system on the control processor can be constructed hierarchically. Basically, the software system has three layers: a knowledge compiler unit, a retrieval-by-unification unit and a resource manager unit (Figure 9.)

Knowledge base systems must handle a variety of knowledge representations. However, it is expensive to provide different systems for all knowledge representations and it is difficult to use different knowledge representations from different applications. To provide a kernel knowledge representation to which all knowledge representations can be translated is appropriate to knowledge base systems. We choose the relational knowledge model as a kernel representation. Then, we must develop a system which translates each knowledge representation to the relational knowledge model. The knowledge compiler unit is the system for the operations. The knowledge compiler translates operations for one of the knowledge representations into operations for the term relations. We expect the unification engines can be used for these translations.

The retrieval-by-unification unit accepts the instructions for term relations, such as unification-join, unification-restriction, or iteration control for these operations. The unit translates these instructions into the instructions for logical devices. The retrieval-by-unification unit can also be seen as a kind of compiler. It does not directly check the branch or iteration conditions, but only generates instructions for the resource manager.

The resource manager unit allocates the real devices. Parallel executions of the unification engines and disk systems are controlled by this unit. The unit also handles the clustering method described above.

There are other functions used for database management systems. Concurrency control for supplying multiple users, database recovery, security control are examples. Since the knowledge base system is an experimental system, we omit the recovery and security control functions.

We only plan to implement concurrency control for the system.

4. Conclusion

We introduced a new concept the 'relational knowledge base'. Its object is a collection of terms. Basic operations on it are unifications. It enables arithmetic treatment for knowledge bases to formalize the collection as a relation. It enables flexible retrieval for knowledge bases to introduce unifications as operation searching for terms. Since the relational knowledge bases are enhanced relational databases, relational databases are contained in the relational knowledge bases as subsets.

We proposed an architecture for implementing a system which manipulates the relational knowledge bases. We proposed to develop unification engines for applying unification operations to terms in data streams and use a multiport page-memory for providing a wide data path between the unification engines and secondary storage systems. Though several unification engines have been presented already [Yasuura et al. 85, Woo 85], we plan to develop a more efficient unification engine for streamed knowledge using the presented ordering. The multiport page-memory consists of a set of memory banks and a switching network. There have been same architectures for database machines [Boral et al. 82], Prolog machines [Sabbatel et al. 84], etc. We think this kind of construction is a common method for highly parallel architecture.

This paper only describe a basic idea for constructing a knowledge base system. There still remain many problems we must solve The function of the knowledge base management system is the most important problem. We must consider the update of the knowledge base under the multi-user environment. It is related to the knowledge assimilation system [Miyachi et al. 84] and the knowledge acquisition system [Kitakami et al. 84].

ACKNOWLEDGMENTS

The authors thank Dr. M. Yoshida of Kyoto University and Dr. Y. Tanaka of Hokkaido University for their useful discussions.

REFERENCES

[Boral et al. 82] Boral, H., DeWitt, D. J., Friedland, D., Jarrell, N. F., Wilkinson, W. K., Implementation of the Database Machine DIRECT, *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 6, November 1982.

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

[Gallaire et al. 84] Gallaire, H., Minker, J., and Nicolas J.-M. Logic and Databases: A Deductive Approach. *Computer Surveys*, Vol. 16, No. 2, June 1984.

[Kakuta, et al. 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.

[Kitakami et al. 84] Kitakami, H., Kunifuji, S., Miyachi, T., Furukawa, K., A Methodology for Implementation of A Knowledge Acquisition system, *Proceedings of the 1984 International Symposium on Logic Programming*, February 1984.

[Kowalski 79] Kowalski R. A, Logic and Semantic Networks, *Communication of the ACM*, Vol. 22, No. 3, pp.185-192, March 1979.

[Koyama et al. 85] Koyama, H., Tanaka, H., Definite Clause Knowledge Representation, Proceedings of the Logic Programming Conference '85, pp.95-106, July 1985, in Japanese.

[Miyachi et al. 84] Miyachi, T., Kunifuji, S., Kitakami, H., Furukawa, K., Takeuchi, A., Yokota, H., A Knowledge Assimilation Method for Logic Database, *Proceedings of the 1984 International Symposium on Logic Programming*, February 1984.

[Sabbatel et al. 84] Sabbatel, G. B., Dang, W., Ianeselli, J. C. Nguyen, G. T., Unification for a Prolog Data Base Machine, *Proceedings of the Second International Logic Programming Conference*, pp207-217 July 1984.

[Tanaka 84] Tanaka, Y., A Multiport Page-Memory Architecture and A Multiport Disk-Cache System, *New Generation Computing 2*, pp241-260, 1984.

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

[Yokota et al. 84] Yokota, H., Sakai, K., Itoh, H. Deductive Database System based on Unit Resolution, to appear *Proceedings of the Second International Conference on Data Engineering*, February 1986.