ICOT Technical Report: TR-240

TR-240

Performance Evaluation of a Unification Engine for a Knowledge Base Machine

by

Y. Morita, M. Oguro (NTT), H. Sakai,

S. Shibayama (TOSHIBA Co.),

H. Itoh and Y. Morita

March, 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

Performance Evaluation of a Unification Engine for a Knowledge Base Machine

Yukihiro Morita, Masami Oguro[†], Hiroshi Sakai[‡],
Shigeki Shibayama[‡] and Hidenori Itoh
ICOT Research Center
Mita Kokusai Building, 21F,
4-28, Mita 1-Chome, Minato-ku, TOKYO 108, JAPAN

January 30, 1987

Abstract

In this paper we describe the performance evaluation of a *unification engine* (*UE*) for a knowledge base machine.

The UE is dedicated hardware that performs the *retrieval-by-unification operations* (*RBU* operations) associated with a relational knowledge base model. It is a conceptual model for a knowledge base in which the knowledge is represented in relations consisting of terms, and unification on terms is used as the retrieval mechanism. The performance of the UE mainly depends on the amount of data passing each component.

We have developed two kinds of pipeline unification methods and performed a quantative comparison of the two methods. The MAM method which applies the most general unifier at the end of a unification is proven to outperform the SAM method.

[†]NTT Electrical Communications Laboratories

[‡]Toshiba Corporation

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. We are developing a knowledge base machine which is an extension of a relational database machine [Itoh 86]. This knowledge base machine is based on the relational knowledge base model in which knowledge is represented by *terms*. This machine stores the knowledge in the secondary storage in the form of a *term relation* in which elements of each domain of each attribute of the relation are extended to terms from atomic values. Operations which retrieve these terms (tuple) in the term relation are also extended to using unification and called *retrieval-by-unification* (*RBU*) operations[Yokota 86]. We use the *unification engine* (*UE*), dedicated hardware to RBU operations, to perform fast knowledge retrieval [Morita 86a].

This paper describes the performance evaluation of the UE. Section 2 describes the configuration of the UE and its simulator. Section 3 shows some results of the performance evaluation of the UE for sample data. Finally, Section 4 proposes a new unification method that improves the performance of the UE.

2 Configuration of the UE

A relational knowledge base system architecture was proposed in [Yokota 86] and [Monoi 86]. Fast unification is the key to the performance of the computation on this model. We propose the UE, dedicated hardware for performing retrieval-by-unification operations at a high-speed[Morita 86a].

Figure 1 shows the UE configuration. A UE uses three channels, two for input data streams to it and one for output data streams from it. A stream is a line of contiguous data items that flows through each units. It processes data streams by a pipeline fashion; it gets the data stream from input channels, processes it and put the results on the output channel.

The unification engine consists of the following units:

tuple memory(TM): The tuple memory stores input tuples.

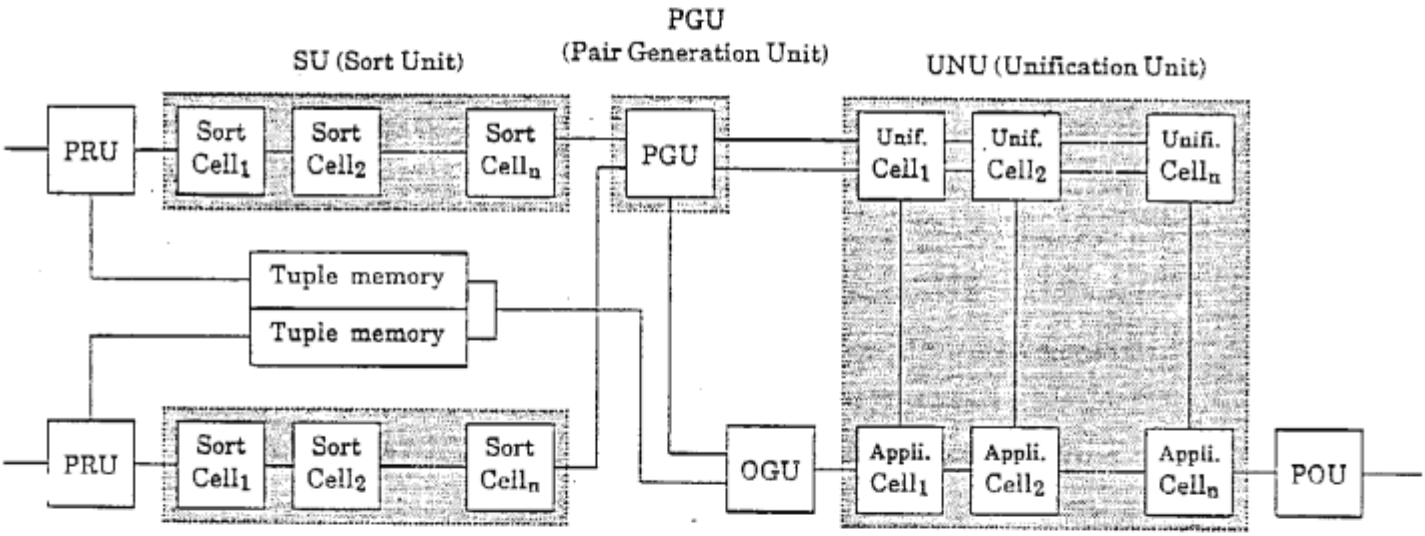


Figure 1. Unification engine configuration.

preprocess unit(PRU): This unit extracts an object item (term) from a tuple and sends out only that item to the sort unit. For example, in the case of join between term relations $kb_1(A, B)$ and $kb_2(C, D)$ on attributes B and C, this unit sends to the sort unit only the values (terms) of attributes B and C.

sort unit (SU): The sort unit sorts sets of terms according to the order of generality using a two-way merge-sort algorithm. This unit contains several sort cells. The input data is a variable-length character strings represented by TRIE representation[Aho 81] that reduces the amount of the data of the later cell.

pair generation unit (PGU): This unit accepts two streams of sorted terms, then generates pairs of possibly unifiable terms and send them to the unification unit. At the same time, pointers to each original tuple in the tuple memory corresponding to the pair is sent to the output tuple generation unit.

output tuple generation unit(OGU): This unit refers to the tuple memory and generates the output tuple according to the information of the PGU.

unification unit(UNU): The unification unit obtains the most general unifier (mgu) of generated term pairs and applies it to the output tuple received from the OGU. The UNU contains several unification cells and application cells.

- **unification cell:** The unification cell finds the disagreement of the input data pair and finds a substitution, if possible, for a variable. Then this cell sends the substitution information to the corresponding application cell.
- **application cell:** The application cell applies the substitution for one variable to the original tuples.

postprocess unit (POU): The postprocess unit reforms the output data.

The entire UE operates in a pipeline fashion among these units. The pipeline flow, however, becomes discontinuous because of the dynamically generated intermediate results.

We have evaluated the performance of the UE using a simulation program written in C. This simulator simulates the behavior of the unification engine at a clock-cycle level, according to a hardware model. The hardware model is set up to be feasible using current technologies. In this paper, basically, one clock cycle is defined as the time required for one read/write RAM operation of 1 word (4 bytes) of data.

Name	Level of the Node			Tuple No.	Data Length	Result Tuple No.
	1	2	3			
Para1	1 binary functor	3 constants		18	352	156
Para1	2 binary functors	3 constants		35	692	311
Para1	3 binary functors	3 constants		52	1032	466
Para1	5 binary functors	3 constants		86	1712	776
Para2	1 binary functor	1 unary functor	3 constants	14	196	178
Para2	1 binary functor	2 unary functors	3 constants	35	871	565
Para2	1 binary functor	3 unary functors	3 constants	66	2340	1169
Para2	1 binary functor	4 unary functors	3 constants	107	4909	1993
Para3	1 3ary functor	1 constants		16	256	256
Para3	1 3ary functor	2 constants		38	1200	824
Para3	1 3ary functor	3 constants		78	4008	1914
Para3	1 3ary functor	4 constants		142	10660	3700

Table 1. Parameters of sample data

3 Performance evaluation of UE

Since the unification-join is the heaviest among RBU operations, as join operations in normal relational operations, because of its $O(N^2)$ computation complexity with a naive implementation, we evaluated the performance of the unification-join operation of the UE.

3.1 Sample data

The first evaluation data we used is a set of artificial terms in which the functor that appears at each level is restricted to an element of the specified set of functors for each level. We used these sets of terms as unary term relations and evaluated the processing time of unification-join between the identical term relations and output tuples that contain attribute values of both term relations (the result relation is a binary relation). Table 1 lists the parameters of the sample data. By using these artificial sets of terms, we think that the results are not biased to a particular

application. This we thought important as the initial stage of this kind of simulation.

The second evaluation data we use is the DCKR sample data. We expected, by contrast to the former artificial sample data, a more application-oriented behavior. The DCKR (Definite Clause Knowledge Representation)[Koyama 85] is a way to represent the knowledge of structured objects by means of definite clauses. The DCKR sample data is a term relation in which a piece of structured knowledge is stored them into a term relation. [Murakami 86] gives a method to store knowledge represented by DCKR into term relation together with an inference mechanism using RBU operations. The sample DCKR term relations are show in Appendix A.

In order to evaluate the UNU separately with a former research result [Yasuura 85], we used another artificial sample data, which is given in Table 3.

3.2 Processing time of the UE

The main components of the UE are the sort unit (SU), the pair generation unit (PGU) and the unification unit (UNU). Figure 2 shows the total processing time of UE and the processing time of each unit for the data Para1. The processing time of the UE is almost equal to the time of the UNU in Figure 2, and it is true in the cases of Para2 and Para3. In these cases the selectivity (result tuple count relative to the product of each relation's cardinality) of the unification-join is poor, that is, PGU generates a lot of pairs. The amount of processing of the UNU depends on the amount of input data, and the amount of the input data to the UNU depends on the filtering efficiency of the PGU. Therefore, the processing time of the UE depends on the selectivity of the PGU and the processing speed of the UNU.

On the other hand, in the case of DCKR sample data, processing times of the SU, PGU and UNU are a 60%, 90% and 20 to 50% of the processing time of the entire UE, respectively. In these cases, (1) selectivity of the unification-join is good, and (2) PGU works effectively to reduce the amount of pairs to be checked at the UNU considerably, so the processing time ratio of the UNU to that of the entire UE is smaller than the above cases.

We describes the a performance evaluation of each unit in the following subsections.

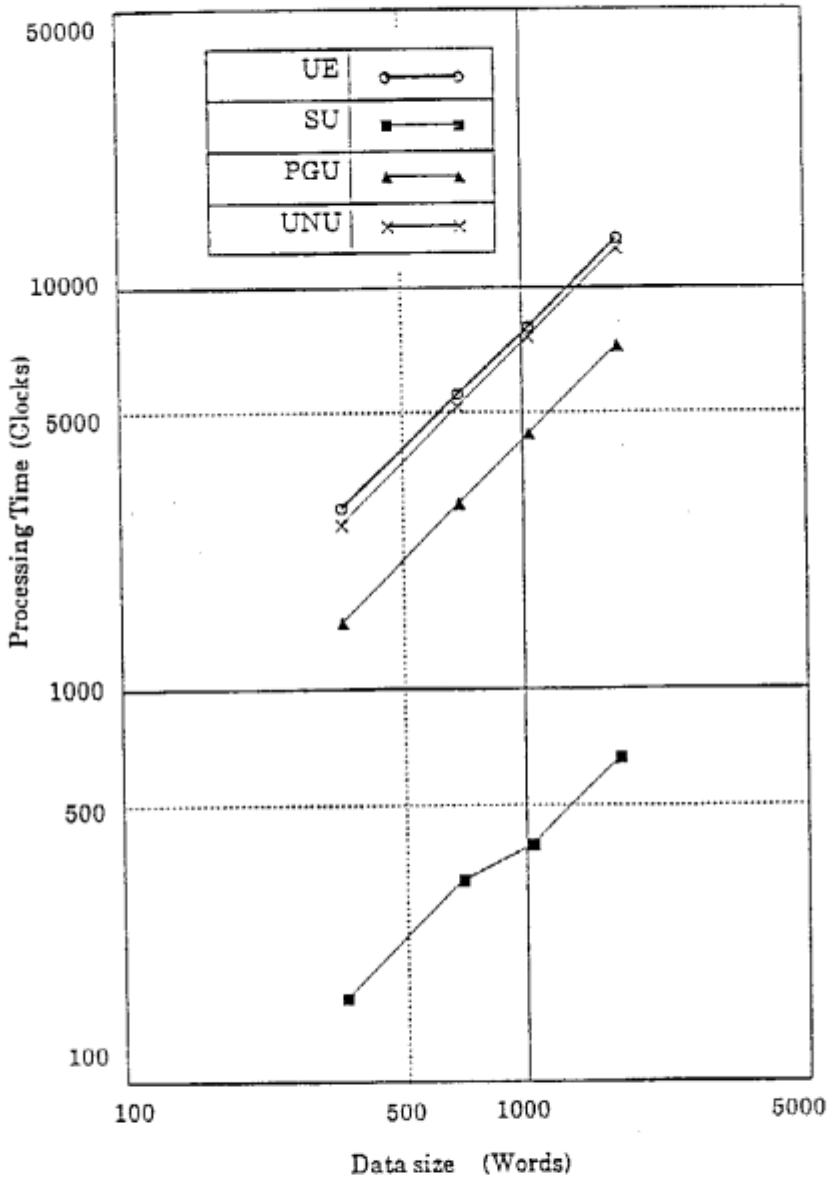


Figure 2. Processing time of each unit

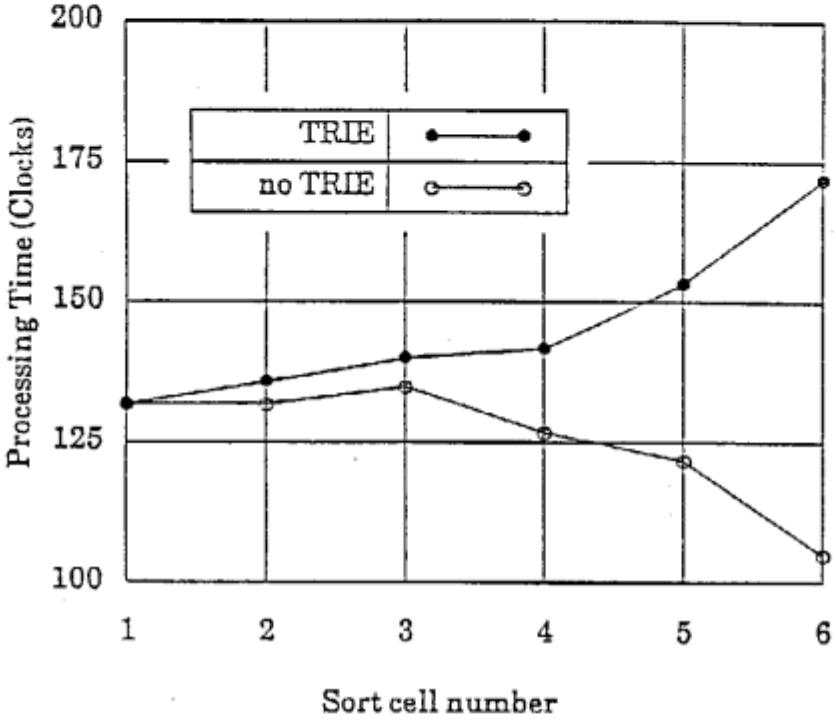


Figure 3. Effect of the TRIE representation

3.3 Processing time of the SU

To consider the processing time of SU, we were interested in the performance improvement caused by incorporating the TRIE representation. It is an effect of the TRIE representation that the processing time of the SU is better than the time proportional to the amount of input data for SU in Figure 2. If we used normal flat representation instead of TRIE, the SU performance would be proportional, at the best, to the amount of input data. Each sort cell reduces the common substring among contiguous input data items in a data stream. Data used in Figure 2 and Figure 3 show the effect of data reduction by using TRIE. Since we use the two-way merge-sort algorithm, the nth sort cell of SU requires the memory for 2^n tuples and there is a fill-up delay for the first 2^n tuples. So if there are no common substrings

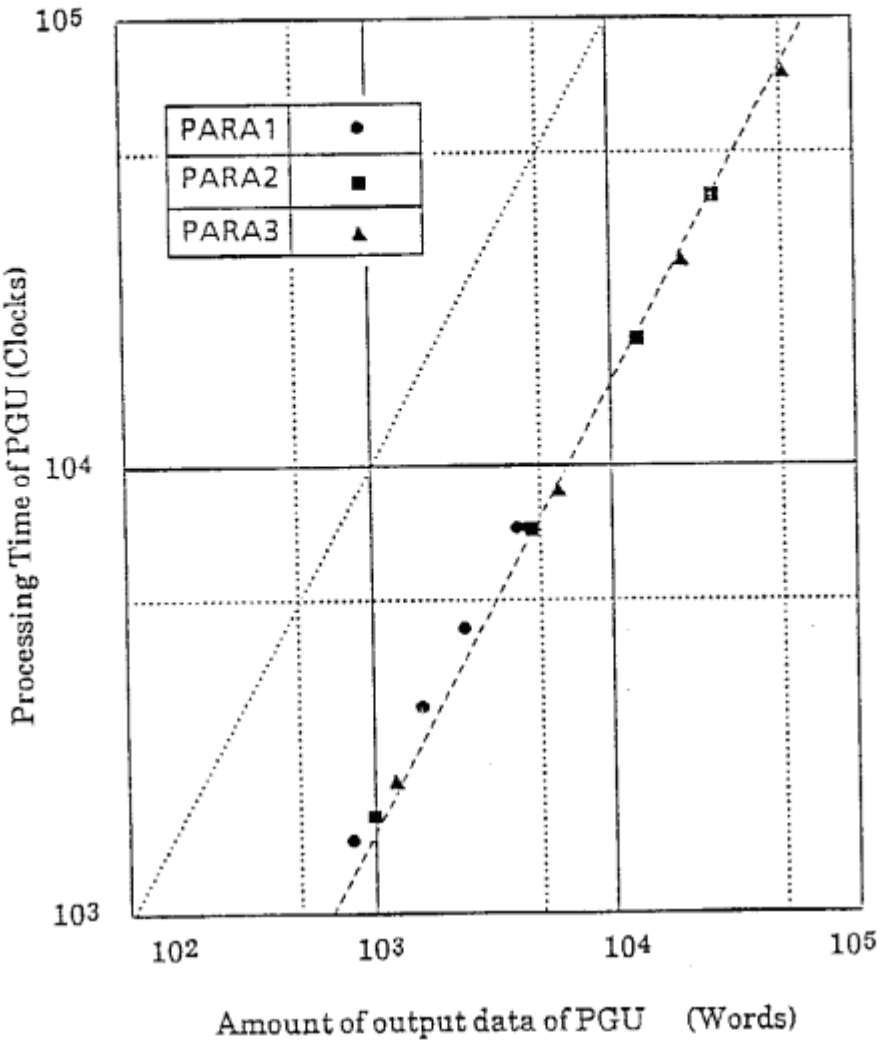


Figure 4. A relationship between the processing time and the amount of the output data of the PGU

in the tuples, then the processing time of the succeeding cells is greater than that of the preceding cells because of fill-up delay (Figure 3).

3.4 Processing time of the PGU

In the case of Para1, 2 and 3, the amount of output data for the PGU is larger than the amount of input data. Figure 4 shows the relationship between the amount of output data of the PGU and the processing time of the PGU. In Figure 4, it is shown that the processing time of the PGU is in proprotion to the amount of its output data. Therefore, if the amount of output data of the PGU is larger than that of input data, the processing time of the PGU is bound to the amount of the output

Level of Node	*Variety of Functor*		
	1	2	3
Root	1.08	1.08	1.08
Intermediate	1.08	1.86	2.63
Leaf	1.08	1.69	2.73

Table 2. A generated pair/unifiable pair ratio

data and the processing time of the pair generation checking is smaller than the time to send a pair to UNU. In other words, the next pair can be generated while the currently-generated pair is output to the UNU.

The ratio of input data and output data of the PGU is an important factor for the performance of the entire UE. The performance of the entire UE depends on the selectivity (of the unification-join) and the pair generation algorithm. Table 2 shows the generated pair/unifiable pair ratio of the pair generation algorithm to each data (Para1, 2 and 3). We call this ratio the *filtering factor* of PGU in this paper. The filtering factor get worse if the variety of functors that appear at the intermediate or leaf node is rich. Put another way, the performance of the UE drops when terms in the term relation have complex structure. This is because the pair generation algorithm has to produce numerous pairs when they have many arcs, which corresponds to the structure complexity.

[Henschen 81] and [Morita 86b] proposed the term index scheme and [Ohmori 86] proposed the hash-sort method for the unification-join. These methods, generaly speaking, have better filtering factor than our pair generation algorithm, because these method check more nodes than our method. The algorithm to generate possibly-unifiable pairs is a tradeoff between pair generation check calculation load and selectivity. These methods, we think, are more suitable to perform the unification-join operation on relations which are classified according to hashing (or indexing). So we plan to employ these techniques to decompose unification-join operation to be done parallelly by multiple UE's.

Unification between $t_{1,i} = f(a_1, a_2, \ldots, a_k)$ and $t_{2,j} = f(X_1, X_2, \ldots, X_k)$, where $i = 1, \ldots, n$.

	Number of Tuples				
	UNU				*
k	1:1	5:5	20:20	50:50	1:1
1	18	9	8	6	10
2	21	12	9	8	13
10	56	37	27	26	37
25	131	76	56		82

unit:clock/pair of tuple

*:[Yasuura 85]

Table 3. Effect of pipeline

3.5 Processing time of the UNU

The UNU processes data in the pipeline fashion. Table 3 show the effect of the pipeline. In this table, the processing time of the UNU per pair of tuples is compared to the hardware algorithm of unification proposed in [Yasuura 85]. The processing time of the UNU is not better than Yasuura's algorithm in the unification between a single pair of terms, but the UNU performs better in the case of unification between multiple pairs of terms. This tendency is clearer when the multiplicity increases. This clearly shows the pipelining effect and thus proves the effectiveness of the pipeline hardware configuration of UNU.

The UNU has two kinds of input data streams; one consists of attribute-value pairs for unifiability checking which comes from PGU, and the other one which comes from the OGU consists of output tuples the most general unifer (mgu) is applied to. The relationship between the amount of input data from the tuple memory to the application cell and the processing time of the UNU are shown in Figure 5. This Figure shows that the processing time of the UNU is in proportion to the amount of the input data from OGU to the application cell.

In the sample data case, two unary relations are unification-joined to form a binary relation each attribute of which consists of the original unary relation. So

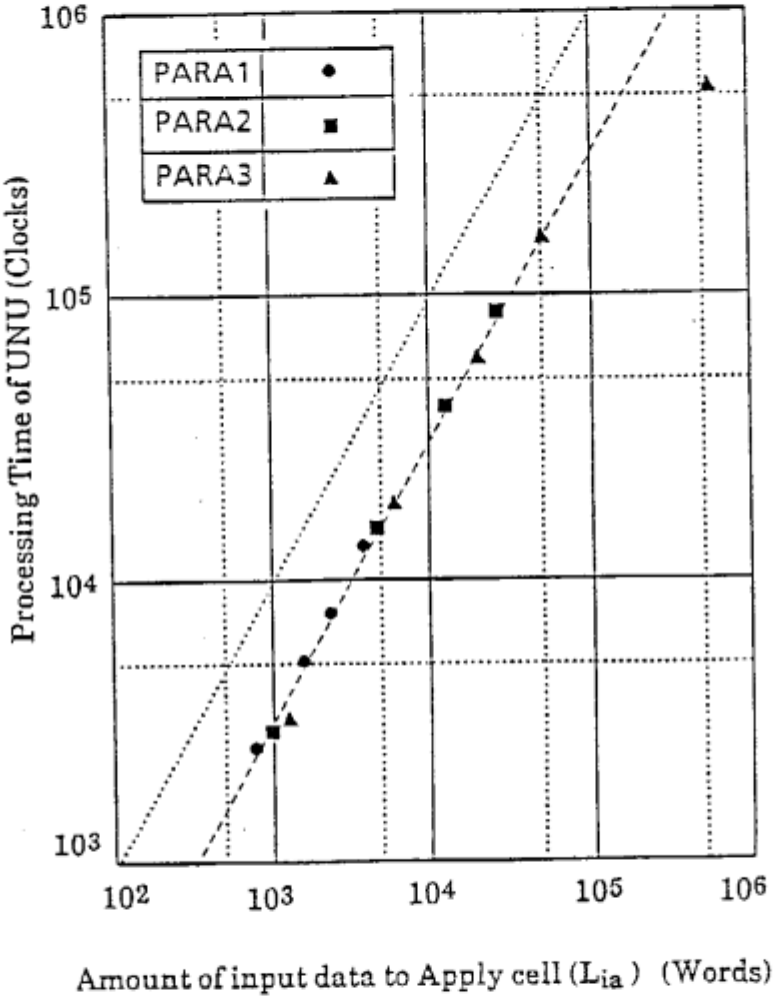


Figure 5. A relationship between the processing time and the amount of the input data form OGU of the UNU

the amount of input data from tuple memory, which becomes output tuples by the application of the mgu, is about twice larger than that of input data from PGU. Since the amount of the input from tuple memory is this large, naturally the processing time for computing the most general unifier of the pair of terms is smaller than the processing time of applying the mgu to the input data from the tuple memory to form the output tuple.

3.6 Discussion

The performance of UE is basically an accumulation of each component. However, each component works in a pipeline fashion, mere pipe travel time (a performance measure of separate components) does not affect the performance when large amount of input data is supplied. In this case, more influential to the performance is the dynamic variance of data streams. Especially, the behavior of SU, PGU and UNU greatly affects the performance since in each unit such dynamic variance takes place. The SU incorporates a hardware variable-length two-way merge-sort algorithm. Unlike the fixed-length hardware two-way merge-sort, the behavior of the sort cells depends heavily on the variance of the input data length. So, generally speaking, the processing time of the SU depends on the amount of input data, a necessity, and how it flows down the cells without busy interlocks caused by the variance of data items. In our results, the processing time of SU depends almost only on the amount of input data. This is because 1) the sets of sample data we used did not have much variance in their data length among tuples and 2) the TRIE representation could, in average, lessen the amount of data transfers between cells, which prevents inter-cell locks caused by long tuples transfers.

The PGU is performing a kind of filtering, that is, it is trying to reduce the amount of data (pairs) to be exactly checked for unification in the succeeding UNU. As is given in 3.4, there are algorithms for this filtering. The feature of our filtering algorithm is that it can receive two input streams dynamically at the same time. So if separate channels are provided to each input port, as in our configuration, the total time for input can be reduced to nearly a half of the single channel case.

The UNU's algorithm is based on performing a one-argument unification at each unification cell. Although we gave a better method in Section 4 in this category, the

processing time is bound to the maximum of the amount of input data for unification checking or output data which is the successful result of unification. In that sense, since we cannot control the amount of successful unification (the output), the PGU is the key to the overall performance. (A naive PGU could generate Cartesian product of two input relations into UNU.)

4 Refinement of the UNU

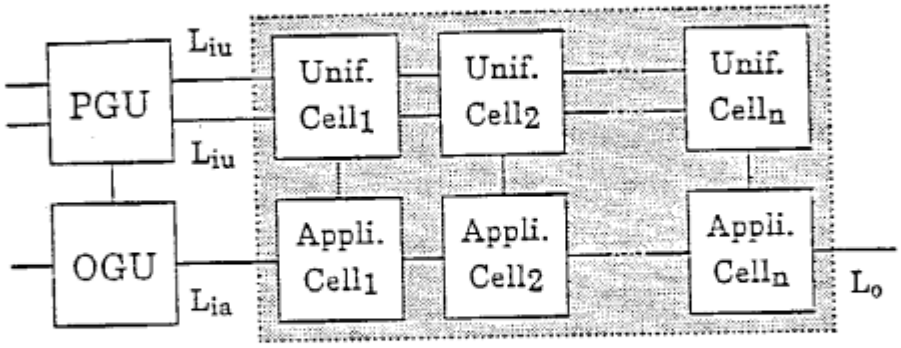
4.1 Modified UNU configuration

Since the original method applies each substitution locally at each unification cell to each variable, we call this the *local substitution application method* (*SAM*). SAM has the following features(Figure 6(a)).

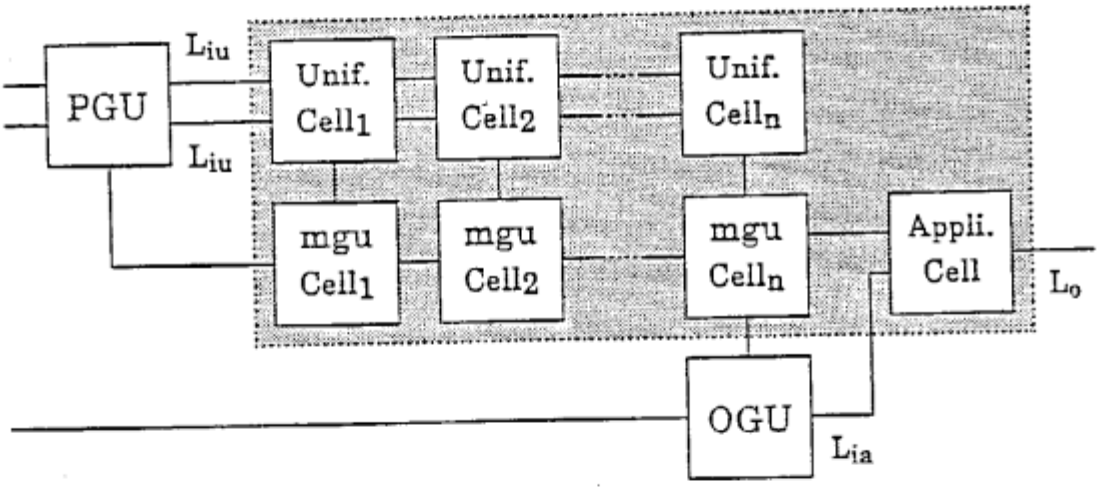
1. Each unification cell processes a pair of terms and obtains a substitution for one variable. Each cell compares two input streams until a difference is found, and obtains the substitution, if one of two input items is a variable. After that, the unification cell sends the substitution information to the corresponding application cell, at the same time, the application cell applies the substitution to the remaining sections of the two streams and sends them to the next unification cell. Therefore, genernry speaking, the terms get longer in size as they go through the unification cells. This causes the processing time of the succeeding cells to become longer than that of the preceding cells.
2. Each application cell processes the output tuple from OGU and applies a substitution for one variable. Since each cell applies the substitution, the amount of data passing through the preceding cell is smaller than that of the succeeding cell.

The SAM has the following problem.

When the mth unification cell finds that a pair of terms is not unifiable at its place, the application cells prior to the mth application cell have already begun to read the corresponding output tuple and to process it. Since the processing time of UNU depends on the amount of data that passes through the application cell, as mentioned in Section 3, this behavior degrades performance.



(a) Local Substitution Application Method (SAM)



(b) mgu Application Method (MAM)

Figure 6. UNU configuration corresponding SAM and MAM.

To solve this problem, we propose the *Most general unifier Application Method* (*MAM*) shown in Figure 6(b). The MAM has the following features.

1. The unification cells work just as in SAM.
2. We use mgu cells to compute most general unifier of the pair of terms from substitutions for each variable. The first mgu cell sends the substitution obtained by the corresponding unification cell to the second mgu cell. Each cell applies the substitution obtained by the corresponding unification unit to the data from the preceding mgu cell and sends them with the substitution to the succeeding cell. For example, when an mgu cell accepts a set of substitutions $\theta_{i-1} = \{f(a)/X, Z/Y\}$ and the corresponding unification cell obtains a substitution $\theta_i' = \{a/Z\}$, the cell applies θ_i' to the θ_{i-1} and outputs $\theta_i = \{f(a)/X, a/Y, a/Z\}$. The output of the last mgu cell is a most general unifier of the pair of terms.
3. In this method, mgu is applied to the output tuple after the mgu cells complete the mgu. Thus the application cell applies the mgu only to the output tuple corresponding to the successfully unifiable pair of terms. Therefore, we can eliminate the time required at the application cells in using SAM.

At first sight, if there are many pairs of terms not unifiable in the input data of the UNU, MAM is faster than SAM. The discussion of the two cases is given in the following subsection.

4.2 Comparison

Figure 7 shows the relationship between the generated pair/unifiable pair ratio (filtering factor) and processing time of both methods. We control the filtering factor by changing the constants in the relation. In Figure 7 the input data to the first unification cell is the same in all cases, and the size (L_{iu}) is a 2×1728 words. That means that the number of generated pair is also constant. Therefore, the filtering factor depends on the amount of output data that depens on the number of unifiable pairs in this case.

The amount of the input data to the first application cell (L_{ia}) is 3265 words in the case of SAM. In the case of MAM, it is variable depending on the filtering factor.

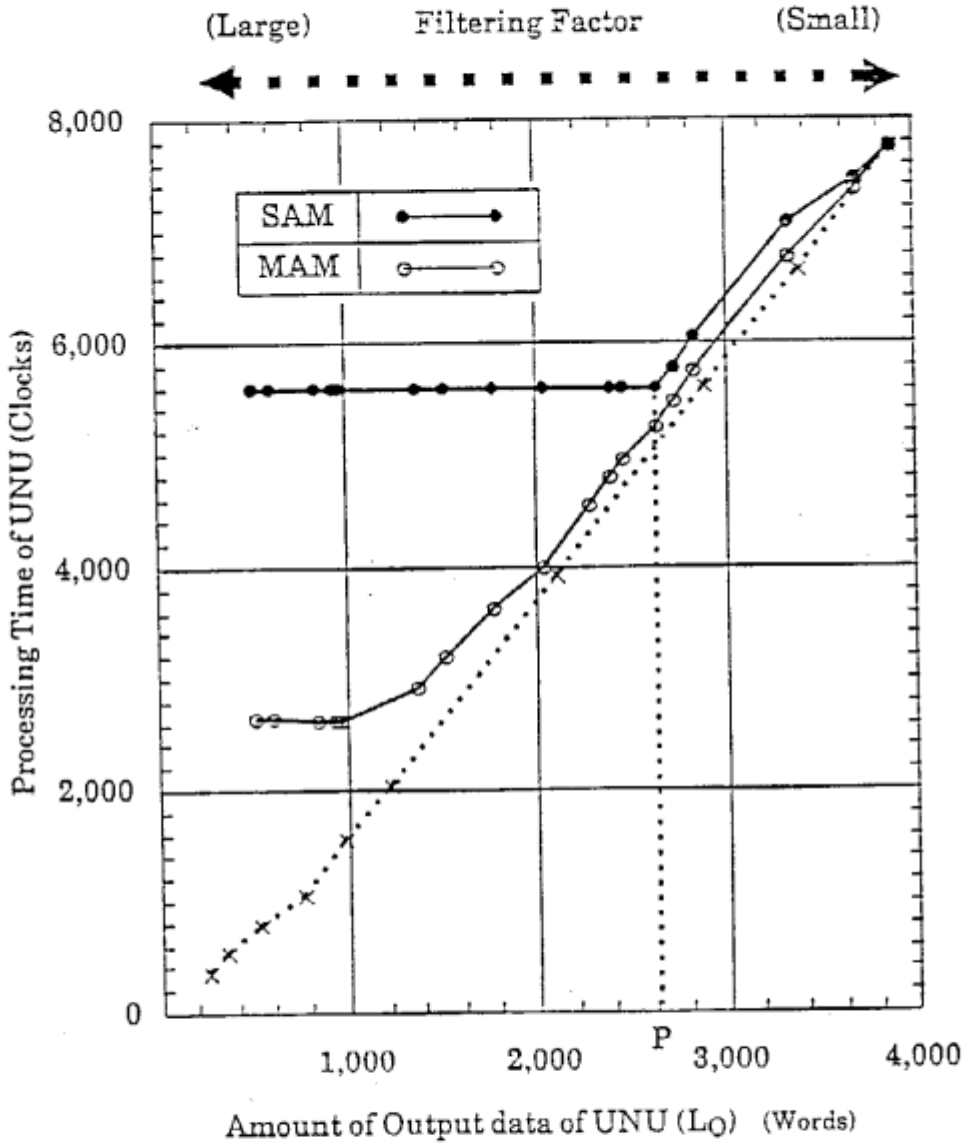


Figure 7. Relationship between filtering factor and processing time of UNU

The broken line in the figure is the processing time of SAM by changing the amount of the all-unifiable input pairs. This line is given as a reference. Figure 7 show the following facts.

1. If the amount of the output data (L_o) is larger than P (filtering factor of about 1.50 in this case), the processing time depends on L_o in both methods.
2. If L_o is smaller than P, the processing time by SAM does not depend on the L_o and the processing time of the MAM is always smaller than that of the SAM.

The first fact means that L_o is larger than any of the amount of data passing any application or unification cells in this case. The second fact means that the amount of data passing the last application cell in the SAM and the last mgu cell in the MAM are both greater than the L_o, and the former is larger than the latter.

Since there are 40 to 70% unifiable terms in the input data to the UNU (filtering factor 1.4 to 2.5) in the sample data cases (Para1, 2 and 3), MAM is faster than SAM.

Next, we compare two methods by DCKR sample data. The filtering factor is 1 to 7.14 in the DCKR sample data. So as we mentioned above, it seems that MAM is faster than SAM in small filtering factor range. But SAM is 1 clock faster than MAM, for example, in the case that filtering facor is 3.57, the processing time of SAM is 1628 clock and that of MAM is 1629.

The reason is that when the L_{iu} is larger than L_{ia} as in these cases, then the processing time of computing mgu is larger than that of applying the mgu. In these cases, the length of each output tuple is smaller than that of stream which represents each pair of terms (the join attribute is not specified to be output). Therefore, SAM is faster than MAM which requires one more cell over SAM.

On the other hand, if the processing time of computing mgu is larger than that of applying the mgu, then MAM is faster than SAM. Figure 8 indicates this. In this figure, the axis of abscissa indicates the mean length of output tuples that go through the application cell(s) (L_o/Number of output tuple).

The mean size of the pairs of attribute values that is checked to be unifiable or not by the unification cells (L_{iu}/Number of tuple) is 13.7 words in this figure. Therefore, MAM is faster than SAM if the filtering factor is high and many attributes

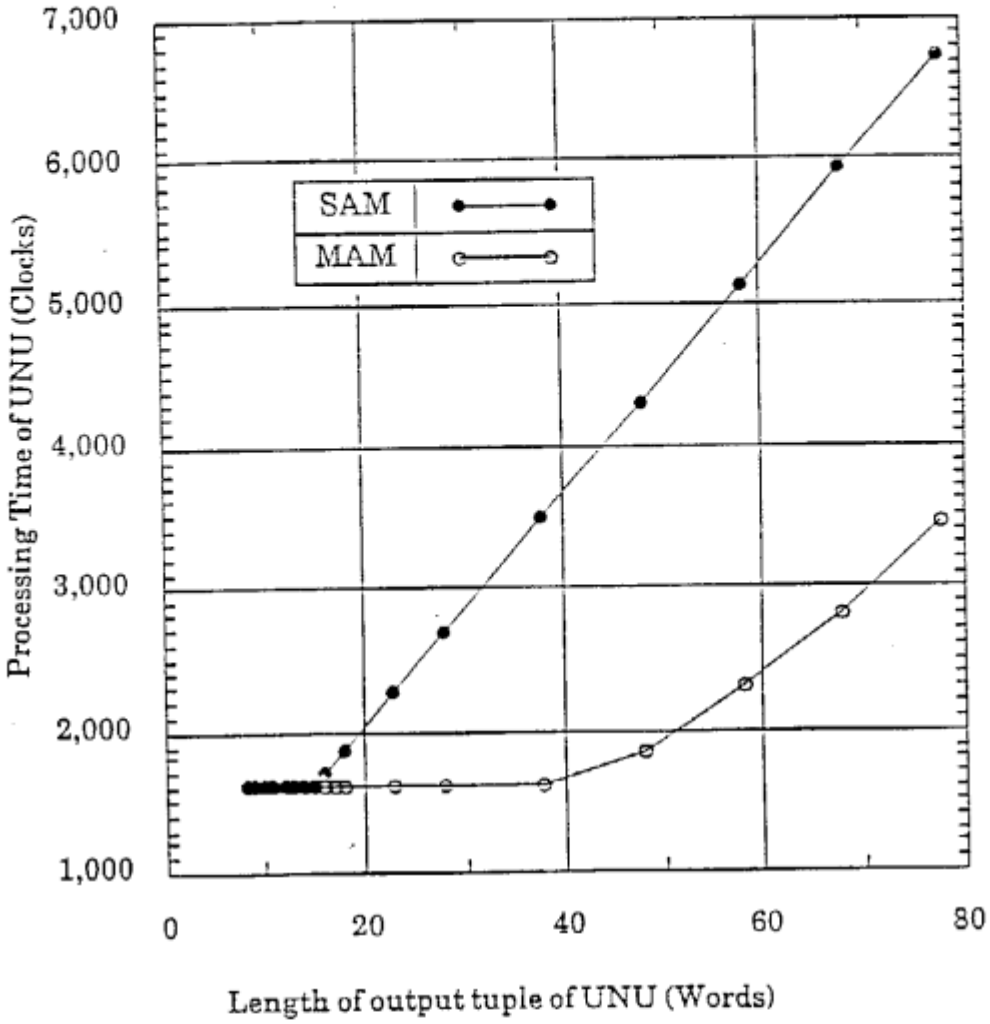


Figure 8. Relationship between size of output tuple and processing time of UNU

are specified for the output.

5 Summary

In this paper we described a performance evaluation of UE. The performance of UE depens on the amount of data that passes each component. The filtering factor of PGU drops if terms of input relation have complex structures and the performance of UE also degrades. We proposed the MAM to compensate the degradation. We also explained the effect and effective area of MAM against SAM. In the future, we plan to study the parallel control techniques of UE's.

ACKNOWLEDGEMENTS

The authors thank Mr. I. Yamazaki of Toshiba Corporation and the members of KBM working group at ICOT for many useful discussions. The authors also thank Mr. K. Shiranami and Mr. M. Takahashi of JUSE Computer Art for their coding the simulation program.

References

[Aho 81] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *Data Structures And Algorithms*, Addison-Wesley, 1981.

[Henschen 81] Henschen, L., and Naqvi, S., "A Fast Literal Indexing Scheme", *ibid*, pp. 528-529.

[Itoh 86] Itoh, H. "Research and Development on Knowledge Base Systems at ICOT", *Proceedings of the 12th International Conference on Very Large Databases* , August 1986.

[Koyama 85] Koyama, H., and Tanaka., H. "Definite Clause Knowledge Representation", *Proceedings of Logic Programing Conference'85*, ICOT, pp.95-106, 1986.

[Monoi 86] Monoi, H., Yokota, H., Murakami, M., and Itoh, H., "A Large-Scale Knowledge Base Machine Control Technique Using Multi-Port Page-Memory", *ICOT Technical Report* TR-156., February 1986.

[Morita 86a] Morita, Y., Yokota, H., Nishida, K., and Itoh, H., "Retrieval-By-Unification Operation on a Relational Knowledge Base", *Proceeding of the 12th International Conference on Very Large Databases* , August 1986.

[Morita 86b] Morita, Y., Wada, M. and Itoh, H., "Structure Retrieval via the Method of Superimposed Codes" *Proceeding of the 33th IPSJ Conference*, 6L-8, In Japanese, 1986.

[Murakami 85] Murakami, M., Yokota, H., Itoh, H., "Formal Semantics of a Relational Knowledge Base", *ICOT Technical Report* TR-149., December

1985.

[Murakami 86] Murakami, M., Yokota, H., and Itoh, H., "Description of Knowledge Base Software Retrieval-By-Unification Language", *Proceeding of the 32th IPSJ Conference*, 1M-9, 1986, In Japanese.

[Ohmori 86] Ohmori, T., Yoshida, A., and Tanaka, H., "An Approach to a Relational Database System Integrated with the Inference Power", AI86-21 *Technical Report of IECE*, AI86-21, pp.33-40, July 1986, in Japanese.

[Robinson 65] Robinson, J.A., "A Machine-Oriented Logic Based on the Resolution Principle", *J.ACM* **12**, pp.23-41, January 1965.

[Yasuura 85] Yasuura, H., Ohkubo, M., and Yajima, S., "A Hardware Algorithm for Unification in Logic Programming Language", *Technical Report of IECE*, EC84-67, pp.9-20, March 1985, in Japanese.

[Yokota 86] Yokota, H., and Itoh, H., "A Model and Architecture for a Relational Knowledge Base", *ICOT Technical Report* TR-144, *Proceedings of the 13th International Symposium on Computer Architecture*, pp.2-9, June 1986.

Appendix

A DCKR sample data

```
[DCKR]

    [sem(cl,A),B]     [sem(ele,A),B]
    [sem(ele,A),B]     [sem(mam,A),B]
    [sem(jewel,A),B]     [sem(stone,A),B]
    [sem(jewel,A),B]     [sem(accessory,A),B]
    [sem(jewel,A),B]     [sem(fortune,A),B]
    [sem(diamond,A),B]     [sem(jewel,A),B]
    [sem(sapphire,A),B]     [sem(jewel,A),B]
    [sem(ruby,A),B]     [sem(jewel,A),B]
    [sem(A,sink_in(water)),B]     [sem(A,density(heavy)),B]
```

```
[sem(hare,color(white)),A]    [sem(current_season,state(winter)),A]
[sem(hare,color(brown)),A]    [sem(current_season,state(summer)),A]
[sem(man,A),B]    [sem(human,A),B]
[sem(bird,has_a(A)),B]    [sem(wing,has_a(A)),B]
[sem(mike,A),B]    [sem(cat,A),B]
[sem(gonbe,A),B]    [sem(cat,A),B]
[sem(hideyoshi,A),B]    [sem(cat,A),B]
[sem(giovanni,A),B]    [sem(cat,A),B]
[sem(giovanni,A),B]    [sem(cat,A),B]
[sem(companella,A),B]    [sem(cat,A),B]
[sem(cat,A),B]    [sem(animal,A),B]
[sem(car,A),B]    [sem(traffic,A),B]
[sem(bmw,A),B]    [sem(car,A),B]
[sem(four_st_engine,A),B]    [sem(engine,A),B]
[sem(psi,A),B]    [sem(computer,A),B]
[sem(bit_map_terminal,A),B]    [sem(terminal,A),B]
[sem(computer,has_a(A)),B]    [sem(terminal,has_a(A)),B]
[sem(computer,has_a(A)),B]    [sem(terminal,is_a(A)),B]
[sem(psi,has_a(A)),B]    [sem(bit_map_terminal,has_a(A)),B]
[sem(psi,has_a(A)),B]    [sem(bit_map_terminal,is_a(A)),B]
[sem(bit_map_terminal,has_a(A)),B]    [sem(bit_map_display,has_a(A)),B]
[sem(bit_map_terminal,has_a(A)),B]    [sem(bit_map_display,is_a(A)),B]
[sem(keyboard,A),B]    [sem(input_device,A),B]
[sem(terminal,has_a(A)),B]    [sem(keyboard,has_a(A)),B]
[sem(terminal,has_a(A)),B]    [sem(keyboard,is_a(A)),B]
[sem(keyboard,has_a(A)),B]    [sem(key,has_a(A)),B]
[sem(keyboard,has_a(A)),B]    [sem(key,is_a(A)),B]
[sem(ele,color(gra)),A]    A
[sem(mam,bt(wam)),A]    A
[sem(accessory,looks(beautiful)),A]    A
[sem(accessory,on(ring)),A]    A
[sem(accessory,on(necklace)),A]    A
```

```
[sem(stone,density(heavy)),A]    A
[sem(stone,hardness(high)),A]    A
[sem(fortune,price(expensive)),A]    A
[sem(fortune,in_the(safe)),A]    A
[sem(diamond,color(clear)),A]    A
[sem(ruby,color(red)),A]    A
[sem(sapphire,color(blue)),A]    A
[sem(pearl,color(white)),A]    A
[sem(current_moon,state(not_full_moon)),A]    A
[sem(wolf,face(shaggy)),A]    A
[sem(wolf,attacks(lady)),A]    A
[sem(wolf,attacks(children)),A]    A
[sem(human,face(not_shaggy)),A]    A
[sem(man,attacks(lady)),A]    A
[sem(birds,can_fly(yes)),A]    A
[sem(bat,can_fly(yes)),A]    A
[sem(tobiuo,can_fly(yes)),A]    A
[sem(airplane,can_fly(yes)),A]    A
[sem(penguin,can_fly(no)),A]    A
[sem(bird,can_fly(yes)),A]    A
[sem(bird,has_a(wing)),A]    A
[sem(cat,likes(milk)),A]    A
[sem(cat,likes(fish)),A]    A
[sem(cat,likes(kotatsu)),A]    A
[sem(giovanni,color(blown)),A]    A
[sem(gonbe,color(black)),A]    A
[sem(hideyoshi,color(white)),A]    A
[sem(giovanni,color(stripe)),A]    A
[sem(car,has_a(wheel)),A]    A
[sem(wheel,has_a(tred)),A]    A
[sem(car,has_a(four_st_engine)),A]    A
[sem(four_st_engine,has_a(valve)),A]    A
```