Flexible Semantic Networks for Knowledge Representation

KEN'ICHI HANDA*, TETSUYA HIGUCHI*, AKIO KOKUBU* and TATSUMI FURUYA*

IXL is a knowledge representation language based on semantic networks. IXL is developed in project IX as a language to describe all kinds of semantic network applications such as natural language processing and knowledge base system.

Since the abstraction level of the semantic networks in IXL is rather low compared with other networks, semantics of relations between concepts can be flexibly described. Two basic hierarchical relations (is_a and instance_of) are provided for inheritance. Other general relations are divided into two categories (assertion and property) and are expressed by a newly proposed network structure. This structure enables us to describe semantics of relations exactly and to treat negative knowledge in a natural manner.

In IXL, the consistency of knowledge throughout the network is checked by determing whether two concepts are exclusive of each other, and we can easily write knowledge bases which answer "true", "false", or "no idea" for inquiries.

For representing procedural knowledge, we adopted a logic-like expression which seems close to human thinking.

IXL is now implemented on DEC-2060 with Prolog, but will run at a high speed on the parallel hardware SMU which is also being developped in project IX.

1. Introduction

The semantic network model [Quillian 66] in knowledge representation has been widely utilized in such areas of artificial intelligence as natural language processing, knowledge base, and human memory models. As knowledge information processing comes into practical use, applications using semantic network representation are required to treat larger sizes of semantic networks.

In order to describe and execute upscaled programs of semantic network applications, two goals should be satisfied.

The first is to provide a programming language which can describe various kinds of semantic network applications. A knowledge base system, for instance, requires as its subsystem a natural language interface, inference engine, and knowledge acquisition system. A natural language processing system also requires a background knowledge base. Therefore providing a single programming language which describe all these types of knowledge processing is desirable for constructing integrated applications.

The second is to design parallel hardware which can utilize the concurrency peculiar to such semantic network processing as marker propagation [Fahlman 80, Hillis 81].

In order to meet these goals, we are developing the IX system (the pronunciation of "IX" is [iks] which stands for the Japanese term for "semantic memory system"). The IX system comprises two subsystems. The first is the programming language, IXL, and the second is the massively parallel hardware, SMU (Semantic Memory Unit). The objective of the IX system is to support all the processes of semantic network information processing in an integrated fashion: from the modelling and description of the semantic network application with IXL, to parallel execution of the IXL program with the SMU.

IXL plays an important role in the development of the IX system because the SMU is being designed as a language-oriented system. Therefore, the architecture of SMU includes some dedicated functions to execute the IXL programs efficiently.

This paper describes the features of IXL in section 2, detailed explanations in section 3 through 5, and a typical example showing the descriptive power of IXL in section 6. Further discussion and the current state of the IX project are given in section 7.

2. The Features of IXL

There are many methods available to represent knowledge in computer, such as production systems, first order logic, frame structures, and semantic networks.

Besides its good suitability for the massively parallel hardware [Higuchi 85], we chose a semantic network to represent declarative knowledge for the following

*Electrotechnical Laboratory 1-1-4, Umezono, Sakura-mura, Niihari-gun, Ibaraki, Japan 305.

reasons.

At first, the semantic network can be a common basic expression of all kinds of declarative (structural) knowledge. For instance, since a working memory cell in a production system can be regarded as a triplet of two nodes and a link between them, the knowledge represented by the production system can also be represented by the semantic network.

The frame structure is the most widely used method to represent structural knowledge because of its tractability and readability. But when we want to express more detailed knowledge such as a meaning of a slot itself or a backward relation from a slot value to its belonging frame, we get into difficulty. To overcome this problem, we must consider the frame, its slot, and the slot value equally as concepts whose semantics must be represented. In other words, we must decompose the frame structure into more basic knowledge units. This is an approach we intend to use with the semantic network. From the standpoint of the semantic network, the frame structure is just a syntax sugar to express several knowledge compactly.

In addition, it appears that the structure of the semantic network may even correspond to human intuition.

There also exists another kind of knowledge, procedural knowledge. To represent such knowledge, we have adopted logic-like expressions. Although other methods such as production rule, Pascal-like expressions, and functional expressions have advantages, we think logic is the most powerful and basic method to represent all such knowledge as rules, procedures, and functions.

We combine these two methods smoothly as described in the following section.

With semantic networks, however, some problems remain to be solved. The first is ambiguity in the semantics of links. In early works on semantic networks, various meanings were attached to links based only on the intuition of system developers. It is rather difficult for the user of such a system to program the meanings of links themselves. Woods provided important suggestions about the semantics of links [Woods 75].

The second problem is the confusion about the levels of concepts represented by the nodes in the networks [Brachman 78]. Since humans generally have several abstraction levels of concepts, it is difficult to define which level of concepts systems should support for programmers wanting to describe many levels of knowledge.

The third problem is the difficulty in describing procedural knowledge with semantic networks. With PSN [Levesque 77], a procedure is assigned to a node and the procedure is activated when the nodes are accessed. However, there exists a gap between human knowledge and representation using PSN because the procedure itself is written in a conventional Pascal-like program. With KRYPTON [Brachman 83], a logic-like expression of procedural knowledge is adopted. KRYPTON is divided into network and procedue areas, so the connection between both areas is not clear, though the expression seems much closer to human thinking than PSN.

The first two problems originate from the gap between the simple structure of networks and the semantics mapped on it. The difficulty is in defining how to map semantics onto the simple data structure which consists of nodes and links to represent concepts possessed by humans.

With IXL, a small set of basic links are provided whose level is semantically rather low compared with other networks, and the network is considered only as a data structure with some inheritance rules. This idea results in the ability to organize the network more freely and to describe the relation between concepts more exactly.

The major features of IXL are:

(1) Two basic hierarchical relations ("is_a" and "instance_of"), with simple inheritance rules, are supported in IXL. Other general relations are divided into two categories (assertion and property) corresponding to these hierarchical relations.

(2) Links are used as very primitive relations (including "is_a" and "instance_of") whose semantics are precisely defined in IXL. The other relational concepts (such as "husband_of") are represented with nodes, and the semantics of those concepts are defined by the programmer in the network structure and procedures. Negative relations can also be treated naturally in the network.

(3) A logic-like expression (using Prolog) can be attached to each node, which enables us to have a description of procedural knowledge related to each concept.

(4) The consistency of knowledge throughout the network is kept in IXL by checking whether two concepts are exclusive of each other.

(5) IXL is implemented with Prolog. All IXL commands in Fig. 1 are executed in the Prolog environment.

Although the treatment of the network at a low level results in a strict semantics of networks, it causes a

```
To connect nodes by a link:
 link(is_a,X,Y).
 link(instance_of,X,Y).
 link(a_kind_of,(X,Y,..),Z).
 link(source,R,X).
 link(destination,R,Y).
 link(rule,X,((
   asst(R,X,Y(,M(,L))):-....
   prop(R,X,Y(,M(,L))):-....
   isa(X,Y(,M,(L))):-....
   instance(X,Y(,M(,L))):-....)).

To construct a relation:
 assertion(R,X,Y(,M)).
 property(R,X,Y(,M)).

To inquire about a link
 isa(X,Y(,M(,L))).
 instance(X,Y(,M(,L))).
 ako(X,Y(,M(,L))).
 source(R,X(,M(,L))).
 destination(R,Y(,M(,L))).

To inquire about a relation.
 asst(R,X,Y(,M(,L))).
 prop(R,X,Y(,M(,L))).
```

```
Meanings of the variables
X,Y:concept node
R: relational node
M(true/false):
   the relation is true/false
L: list of used rules
```

Fig. 1 Command list of IXL.

semantic gap between the knowledge represented in IXL and human thinking. For instance, such notions as "A man has hands" and "John has hands" have the same structure in natural language. But their precise meanings are that every instance of a concept "man" has an instance of "hand" and that a concept "John" itself has an instance of "hand". The network of IXL must have the latter kind of the two different structures.

In our approach, however, this gap should be reduced by application programs written in IXL, not by the network itself. IXL is developped just as a language to describe such applications. The details are given in the following sections.

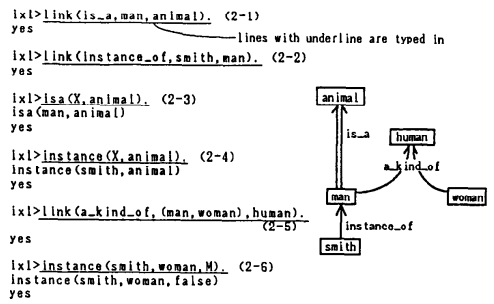


Fig. 2 Hierarchical relation.

3. Hierarchical Relation

One of the advantages of semantic networks is that they can directly represent hierarchical relationships between concepts. This leads to the notion of inheritance, which means that the property of a concept (class) is inherited by its subconcept (subclass).

However, it is impossible for a system to support all hierarchy types and corresponding inheritance rules because of their infinite number in the real world.

In the IXL system, only "instance_of" and "is_a" are supported as primitive links for hierarchy description. Other complex inheritance are to be described in the form of logical rules and attached to relational concepts. This mechanism of procedure addition is described in section 5.

"Instance_of" is a link to connect two concepts in the case that one (i.e. "X") is instanciation or realization of the other (i.e. "Y"). In other words, X is in a lower abstraction level than Y. We call X the "instance" in regard to Y, and Y the "class" concerning X.

There also exist a hierarchy in the same abstraction level, and "is_a" is used to connect two concepts in the case that one (i.e. "XX") is a specialization of the other (i.e. "YY") or the latter is a generalization of the former. Concepts such as XX/YY are called subclass/superclass. But this definition is rather ambiguous. The strict definition of "is_a" is:

"XX is_a YY" ↔

"every instance of XX is also an instance of YY"

This leads to:

"XX is_a YY" and "YY is_a ZZ"
→"XX is_a ZZ"

Fig. 2 shows an example of these hierarchies. (2-1) and (2-2) are declaration of hierarchy. (2-3) asks what is a subclass of "animal", and the system replays "man" by binding it with variable X. (2-4) asks a instance of "animal" and the system replies "smith".

IXL has also a facility to show the exclusiveness of concepts. For example, to show that "man" and "human" are both subconcept of "human" but exclusive of each other, we can say:

link (a_kind_of, [man, woman], human).

The exact meaning of 'exclusiveness' in IXL is that if several concepts are exclusive of each other, every instance of one of them can never be an instance of the other ones. (2-6) of Fig. 2 shows that the system detects the inconsistency with exclusiveness and says "Mr. Smith is not an instance of woman".

All IXL commands for query have an optional argument to show whether a reply is positive (true) or negative (false), and the failure of commands shows that the system knows nothing, indicating an answer of neither true nor false, about the question.

4. Relation

4.1 Assertion and Property

General relations (A concept X has a relationshp R with Y in the direction X→Y) are expressed by the network structure, which consist of the three nodes X, R, and Y combined by links. In this case, X is called the source of this relation, Y the destination, and R the relational concept. It is characteristic of IXL that these general relations are divided into the two categories of "assertion" and "property".

An assertion is a relation in which the source and destination themselves are concerned. A property is a relation in which instances of the source and the destination are concerned.

For instance, the notion "An airplane is equipped with wings" means that every instance of airplane has an instance of wings. So this relation is a property "airplane equipped_with wings". On the other hand, "The airplane was invented by the Wright brothers" means that the very concept "airplane", not an instance of it, was invented by the Wright brothers, and this is an assertion "airplane invented_by wright_brothers". A lack of such a distinction resulted in misinterpretations in early networks.

A difference between assertion and property also appears in inheritance rules. An assertion is never in-

```
ixl>assertion(invented_by,airplane,wright_brothers). (3-1)
yes
ixl>property(equipped_with,airplane,wings). (3-2)
yes
ixl>link(is_a,propeller_plane,airplane). (3-3)
yes
ixl>link(instance_of,the_spirit_of_st_louis,propeller_plane). (3-4)
yes
ixl>assertion(possessed_by,the_spirit_of_st_louis,lindbergh). (3-5)
yes
ixl>prop(equipped_with,propeller_plane,wings). (3-6)
yes>
ixl>prop(equipped_with,the_spirit_of_st_louis,wings). (3-7)
no
ixl>asst(equipped_with,the_spirit_of_st_louis,wings). (3-8)
yes
```

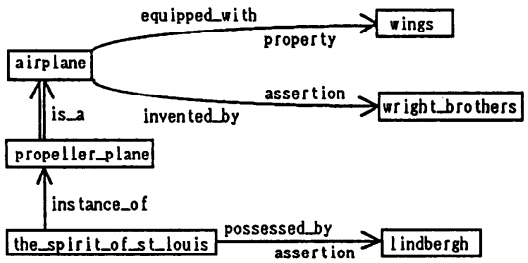


Fig. 3 Assertion and property.

herited unless a procedure forces it. A property is inherited through an "is_a" link, which means some properties true with one concept also hold true with its subclass. A property is inherited through an "instance_of" link in a different way. If a class has some properties with destinations, every instance of the class has the same assertions with those destinations. So we can say that properties are inherited by its instance as assertions.

Fig. 3 is an example to show the distinction of assertion and property. A network is constructed by (3-1) through (3-5). (3-6) asks if instances of propeller plane, a subclass of airplane, has wings. On the other hand, (3-7) asks the same question about "the_spirit_of_st_louis" and the answer 'fail' means that the system knows nothing about that. It is because there's no such knowledge, but that "the_spirit_of_st_louis" itself has instance of wings (3-8).

In some cases, we want to say that something has an assertion with an instance of something (i.e. Robin, which is a specific instance of a bird, has wings). Of course,

property (equipped-with, bird, wings).

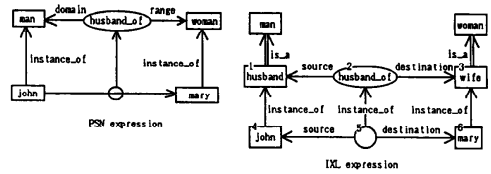
link (instance_of, robin, bird).

shows the information, but we can do that without mentioning bird as follows:

assertion (equipped_with, robin, X),

link (instance_of, X, wings).

In this case, variable X becomes an internal node without name.




```
ixl>assertion(husband_of,john,mary). (4-1)
yes
ixl>link(source,husband_of,husband). (4-2)
yes
ixl>link(is_a,husband,man). (4-3)
yes
ixl>link(destination,husband_of,wife). (4-4)
yes
ixl>link(is_a,wife,woman). (4-5)
yes
```

Fig. 4 Structure of relation (assertion).

When we describe a knowledge in IXL, we must consider the deep semantics of the knowledge. Namely, the difference of hierarchy ("is_a" or "instance_of"), and the difference of relation (assertion or property). These differences do not appear explicitly in natural languages.

4.2 Network Structure of Relation

To describe the semantics of a relational concept itself, the concept must be represented by nodes (not by links as in Fig. 3). Therefore some structure is required to combine concepts which concern a relation. Though the structure proposed here is similar to that in PSN, its meaning is more strict.

IXL introduces "source" and "destination" links, which connect a relational concept with concepts which necessarily concern the relation. Fig. 4 is an example of an assertion which also shows the difference between IXL and PSN. (4-1) creates the structure which consists of 6 numbered nodes. At this time, node 1, 3, and 5 have no names. (4-2) and (4-4) name the nodes 1 and 3 "husband" and "wife" respectively. Node 5 is the key node of this structure because the system knows the relation between John and Mary through this node.

The structure of the IXL network shows that every instance of "husband" requires a relation "husband_of" to some instance of "wife". So, only if Mr. Smith is an instance of husband, we know that Mr. Smith surely has assertional relationship "husband_of" with some instance of "wife". Such knowledge is difficult to represent by the "domain" and "range" links of PSN.

Property can be represented in a similar structure by using "is_a" link instead of "instance_of" link.

These structures also make it possible to represent negative relations in a simple way. To treat negative relations, we introduce "not_isa" and "not_instanceof" and use them as shown in Fig. 5. (5-3) declares that penguin doesn't have relation "can" with "fly". In other words, a penguin can't fly. This declaration brings on the warning from the system that (5-3) goes against the relation inferred from (5-1) and (5-2). ";" forces the system to accept the new knowledge. This

```
ixl>property(can,bird,fly).          (5-1)
yes

ixl>link(is_a,penguin,bird).         (5-2)
yes

ixl>link(instance_of,pen_A,penguin).
yes

ixl>property(can,penguin,fly,false). (5-3)
It's inconsistent because,
  [isa(penguin,bird),
   prop(can,bird,fly,true)] ;
accepted                    forcing

ixl>asst(can,pen_A,fly,M).
asst(can,pen_A,fly,false).
```

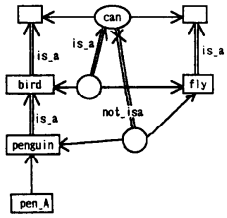


Fig. 5 Structure of property and negation.

type of knowledge is very useful in knowledge base description and inquiry.

5. Procedure Addition

A procedure in knowledge representation can be assumed to be a rule to infer the existence of a relation between some concepts. From this point of view, it is natural to express the procedure in a logic-like form. In IXL, Prolog is adopted to express procedural knowledge.

It is not known how procedural knowledge is stored in the human brain, but at least it can be said that procedural knowledge stored without structure is not practical. If procedural knowledge is described monotonously, we can't tell when to use it to infer some relation. Fortunately, since concepts are represented by nodes in a structured network, we only have to add a procedure to a node which specifically concerns the relation.

The method for the procedure addition is shown below. A "rule" link from a concept node indicates a procedural node which consists of several Prolog Horn clauses such as:

asst (R, X, Y, M, L):- . . . ,

prop (R, X, Y, M, L):- . . . ,

isa (X, Y, M, L):- . . . , and

instance (X, Y, M, L):- . . . ,

where R is the relational node, X is the source of the relation, Y is the destination of the relation, M is a true/false flag to indicate whether the inference is correct/incorrect, and L is a list of inference rules used in this procedure. The body of those clauses consists of any prolog term at all and IXL commands. Each Horn clause is activated when the corresponding inquiry occurs.

The priority in determining which inferred relation is true is as follows:

1) A relation described directly in the network.
2) A relation inferred from an inheritance rule.
3) A relation inferred from a procedure.

3.1) Inferred from a procedure added to the source of a relation.

3.2) Inferred from a procedure added to a relational node.

3.3) Inferred from a procedure added to the destination of a relation.

If the inference fails at a priority level, it advances to the next lower priority level.

This method makes it possible to represent inference rules accompanying a relational concept. Fig. 6 is an example of relational concept "brother_of". (6-1) is a step to declare that A is a brother of C if A is a brother of B and B is a brother of C. The query (6-2) sequentially returns answers one by one by using this procedural knowledge.

The procedure in Fig. 6 indicates a recursive call with no terminate conditions (i.e. asst(. .) appears in the clause whose head is asst(. .)). Although, in general, such a Prolog program doesn't stop when no solution is found, IXL can check this kind of infinite loop and stop and fail such a predicate call. IXL owes this to the fact that all variables in IXL commands can be bound only with names of nodes on semantic network.

6. Description of Semantics of Relational Concept

Although knowledge representation languages proposed so far have respective advantages in expressing semantics of such concepts as bird, man, or desk, they do not have sufficient power to describe semantics of such relational concepts as color_is, husband_of, etc.

The advantage of IXL is that all the procedural knowledge to infer some relation is activated only when required by adding such knowledge to a proper concept. We don't have to describe the knowledge either too specifically or too generally.

In this section, how the inference rule of IXL works with procedural knowledge is described along with an example about the concept "husband_of". The goal of the program in Fig. 7 is to infer that Mary is a wife of John only from the knowledge that John is a husband of Mary.

Required knowledge for this inference is:

(1) "husband_of" is an inverse relation of "wife_of".

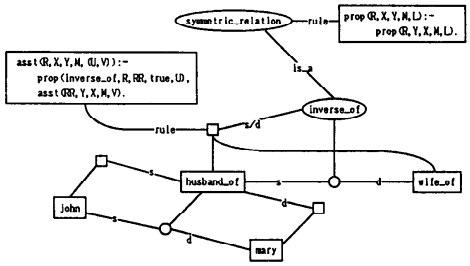
(2) If R is an inverse of RR and Y has a relation RR with X, X has a relation R with Y.

(3) The relational concept "inverse_of" is a subclass of "symmetric_relation".

(4) If R is a symmetric relation and Y has a relation R

```
ixl>link(rule,brother_of,((
    asst(R,X,Y,true,(L1,L2)):-
    asst(R,X,Z,true,L1),asst(R,Z,Y,true,L2))). (6-1)
yes
ixl>assertion(brother_of,paul,john).
ixl>assertion(brother_of,john,kenny).
ixl>assertion(brother_of,kenny,tom).
ixl>asst(brother_of,paul,Y,M,L).    (6-2)
asst(brother_of,paul,john,true,
     ((brother_of,paul,john))) ;
asst(brother_of,paul,kenny,true,
     ((brother_of,paul,john),
      (brother_of,john,kenny))) ;
asst(brother_of,paul,tom,true,
     ((brother_of,paul,john),
      (brother_of,john,kenny),
       (brother_of,kenny,tom)))) ;
no
```

Fig. 6 Procedure addition.




```
ixl>property(inverse_of,husband_of,wife_of).  (7-1)
ixl>source(inverse_of,S),
       link(destination, inverse_of,S),
       link(rule,S,((asst(R,X,Y,M,(U,V)):-
            prop(inverse_of,R,RR,true,U),asst(RR,Y,X,M,V)))).  (7-2)
ixl>link(is_a,inverse_of,symmetric_relation).  (7-3)
ixl>link(rule,symmetric_relation,((
       prop(R,X,Y,M,L):- prop(R,Y,X,M,L)))).  (7-4)
ixl>assertion(husband_of,john,mary).  (7-5)
ixl>asst(wife_of,mary,John,M,L).  (7-6)
asst(wife_of,mary,John,true,
   (prop(inverse_of,husband_of,wife_of),
    isa(inverse_of,symmetric_relation),
    prop(inverse_of,wife_of,husband_of),
    asst(husband_of,john,mary))
ixl>property(inverse_of,right_of,left_of).  (7-7)
ixl>assertion(right_of,block_A,block_B).  (7-8)
ixl>asst(left_of,block_B,X).
asst(left_of,block_B,block_A)  (7-9)
```

Fig. 7 Semantics of relational concepts.

with X, X also has the relation R with Y.

This knowledge corresponds to each statement of the IXL program. (1) is regarded as an assertion because every instance relation of "husband_of" has an inverse relation which is an instance of "wife_of", and is declared as (7-1).

The procedure representing (2) must be activated when a query about a relation, an inverse relation of which exists, occured. So the procedure is added to the node indicated from "inverse_of" by source and destination link so as to be inherited to every subclass of the node which surely have some inverse relation of it.

(3) is a simple subclass/superclass hierarchy, and (4) is expressed as a procedure to be activated on a query about "symmetric_relation".

After this background knowledge is entered and constructed as a semantic network within the system, the declaration of (7-5) induces the objective inference of (7-6). First the system searches the network and fails. Next it searches for a procedure added to "john", "mary", or "husband_of", and finds the rule of (7-2) inherited to "husband_of". The procedure is called and the body predicates are executed as follows:

prop (inverse_of, wife_of, RR, true, U),

asst (RR, john, mary, true, V).

On the call of the first predicate, the system searches the network and fails again because there's only property (inverse_of, husband_of, wife_of). Then a procedure about "wife_of" or "inverse_of" is searched and (7-4) is found. The call for this procedure terminate successfully and return "husband_of", the inverse relation "wife_of", to variable RR. The next predicate of the body of (7-2) is then executed with instanciated RR and terminated successfully, and the initial call of (7-6) also terminates successfully with L which has a list of used rules.

The remarkable thing with this example is that all the knowledge is represented at an appropriately general level. For instance, it is of course possible to have the same inference only through the below knowledge added to "wife_of":

(5) If Y is a husband of X, X is a wife of Y.

But, there are many other relations which have similar natures as (5), and it is too troublesome to add such a rule to all of them. In the above example, however, the simple declaration of (7-7) about "right_of" and "left_of" makes it possible for the system to reply as (7-9) from the assertion (7-8).

7. Discussion and Conclusion

In this paper, a new approach to the usage of semantic networks was described and the knowledge representation language IXL was proposed.

The network itself does not attempt to force a reduction in the semantic gap between the simple structure of the network and natural language in human thinking, but rather is organized at a low abstraction level. Since programmers can operate the network freely just as a data structure, they can describe semantics of relational concepts themselves in detail. This is because IXL is designed to describe semantic network applications. The applications are what reduce the gap mentioned above.

The multiple use of the knowledge once added is an especially important faculty of knowledge representation languages. It can be said that the number of different responses induced from one input is the criterion of such languages or systems. This requirement depends on how generally programmers can construct knowledge.

In order to do so, we must, first, consider what is the basic element of the knowledge. Before entering some knowledge, it must be sufficiently considered whether or not the knowledge can be easily inferred from other elemental knowledge, and whether or not a more general concept holds such a knowledge.

We believe that this method to construct a knowledge base gains in importance hereafter in knowledge representation, and we hope IXL will be a useful tool to meet the needs for this consideration.

All of the IX projects are now proceeding in parallel, on the hardware of SMU [Higuchi 85], on IXL, on the graphical user interface of IXL, and on the application program written in IXL. This program is a knowledge base for French wines.

Since IXL is now implemented, temporarily, with Prolog on DEC-2060, the memory size and the execu-

tion speed are not sufficient for large practical application programs. This lower speed is mainly due to the process that checks the exclusiveness of concepts. However, this process has high and simple parallelism potentially, and the SMU will surely solve this problem.

The approach of SMU is to map the semantic network directly onto the collections of processing elements. Execution of all IXL query commands (i.e. isa, prop, and so on) may be performed within the instruction cycles proportional to the depth of the network (i.e. the link counts from the top-most class to its instance) even in the case of large networks [Higuchi 85]. This means that we do not have to care about a size of a network and a sequence of creating a network any more for efficiency. So, we can concentrate on the crucial thing, describing precise knowledge.

Problems still exist with IXL such as how to treat sets, quantification, and grouping of knowledge. These are problems not only with IXL but with any other knowledge representing languages. Further research is required on these subjects.

References

[Brachman 78] BRACHMAN, R. J. On the Epistemological Status of Semantic Networks, In "Associative Networks" (Findler ed), *Academic Press*, New York, 3-50.

[Brachman 83] BRACHMAN, R. J., FIKES, R. E. and LEVESQUE, H. J. Krypton: A Functional Approach to Knowledge Representation, *Computer* **16**, 10 (1983), 67-74.

[Fahlman 80] FAHLMAN, S. E. Design sketch for a million-element NETL machine, *Proc. of First Annual National Conf.* on AI (August 1980).

[Higuchi 85] HIGUCHI, T. A Semantic Network Language Machine, *Proc. of Euromicro* '85.

[Hillis 81] HILLIS, G. E. "The connection machine," TR-646, Cambridge, Mass. MIT AI Lab.

[Levesque 77] LEVESQUE, H. J. "A Procedural Approach to Semantic Networks," Technical Report, 105, *Department of Computer Science*, Univ. of Toronto.

[Quillian 66] QUILLIAN, M. R. "Semantic Memory," Report AF-CRL-66-189, BBN, Cambridge, Mass., (1966).

[Woods 75] WOODS, W. A. What's in a link? Foundations for semantic networks, In Representation and Understanding (Bobrow and Collins eds), *Academic Press*, New York 35-82.

(Received December 13, 1985; revised December 11, 1986)