Modeling Semantic Networks on The Connection Machine [1]

Technical Report No. CENG 89-13

Sang-Hwa Chung, Dan Moldovan and Yu-Wen Tung [2]

Department of Electrical Engineering-Systems

University of Southern California

Los Angeles, CA 90089-0781

October 10, 1989

[1]This research has been partially supported by the National Science Foundation Grant No. MIP-89/02426.

[2]Tung is currently at USC - Information Science Institute. He has been partially supported by the NASA Cooperative Agreement NCC-2-539 and RADC contract F30602-88-C-0135.

Abstract

In this paper, it is shown how the Connection Machine can be used to process knowledge represented by semantic networks. The software data structure for semantic networks, marker propagation rules, and the semantic network instruction set were defined and implemented on the Connection Machine. Five examples of semantic network processing were programmed and their timing measured. From these examples, we observed that the execution time is proportional to the length of critical path, the number of marker propagations, and branching factor.

Contents

1 Introduction

A *semantic network* representation was introduced by Quillian [6] in 1968 to model human memory. Since then, semantic networks have played a significant role in knowledge representation research. Semantic networks can represent a broad range of knowledge and support various reasoning mechanisms.

In spite of operating with a large knowledge base, human agents take a few hundred milliseconds to perform a broad range of cognitive tasks [4], while artificial intelligent systems are still far from achieving this performance. To meet this time constraint, it seems that intelligent systems have to resort to parallelism. A possible solution to this problem is to process semantic networks on a massive parallel processor. The Connection Machine, a massive parallel processor commercially available, could be an effective tool for this purpose.

In this paper, we show how semantic network processing can be done on the Connection Machine, and report on the performance achieved. Sections 2 and 3 introduce semantic network concepts and the Connection Machine. Section 4 describes the mapping and programming of a semantic network model on the Connection Machine. Five examples are given in Section 5.

2 Semantic Networks

2.1 Representation of Semantic Networks

Semantic networks express knowledge in terms of concepts, their properties, and the hierarchical relationship between concepts. In other words, concepts are connected to their subsuming concepts (super concepts) by *is-a* or *subsumption* links, and connected to their properties by appropriately labeled links.

Concept nodes are organized hierarchically, that is, more abstract, or general concepts are placed higher in the hierarchical representation. Properties of more abstract concepts are inherited by all concepts subsumed by higher concepts, except when specially declined.

As an illustration, consider a semantic network in Figure 1. In this example, `Dog` is `Fido`'s subsuming concept, and `Fido` has the properties: (`has-hair-type Long`) and (`has-color Brown`), where `Fido` is an instantiation of `Dog` in the conceptual hierarchy. `Fido` inherits `Dog`'s property: (`has-part Legs`).

The term *Semantic Networks* has been defined and used in several different ways. Most notable systems using semantic networks are FCL [7], KRL [1], NETL [3], and KL-ONE [2]. In spite of some differences, the basic description of semantic networks given above is shared by all these systems.

Next, a semantic network representation of a few English sentences is shown. The English sentences are as follows:

```
Clyde is circus-elephant.  Circus-elephants are elephants.
```

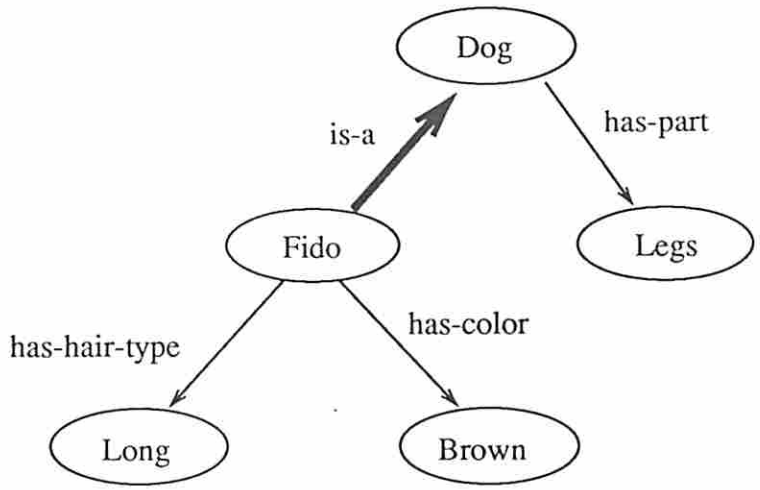

Figure 1: An Example of Semantic Network

```
Circus-elephants are performers.  Performers have part
costumes.  Costumes are clothes.  Elephants are mammals.
Elephants have part elephant's head.  Elephants have color
gray.  Elephants hate lions.  Elephant's head has part
trunk.  Elephant's head has part elephant's mouth.  Elephant's
mouth has part tusk.  Tusk are teeth.  Lions are mammals.
Lions have part teeth.  Mammals are things.  Mammals have
part legs.  Legs are used for movement.
```

Several semantic network representations may be derived for the above English sentences. One is drawn in Figure 2, where each ellipse represents a concept, `is-a` links are depicted by wide arrows, and user-defined links by narrow arrows. This example will be used later to illustrate a query processing.

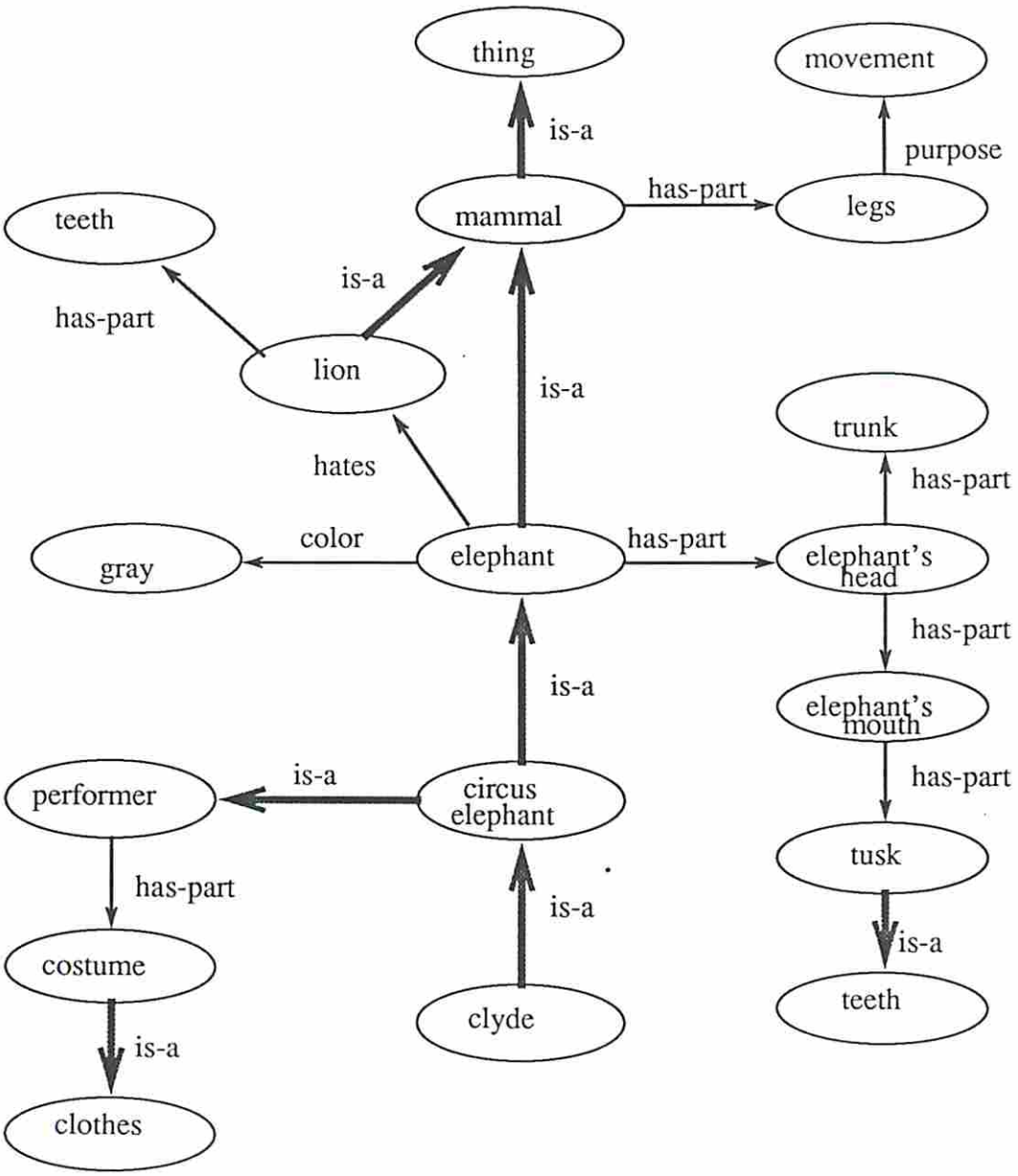

Figure 2: Semantic Network for Clyde

2.2 Inferences on Semantic Networks

Once a semantic network is constructed, we can use the network for reasoning. Concept recognition and property inheritance are two important types of inferences, and are discussed in this paper. These two forms of inferences lie at the core of intelligent behavior and act as precursors to more complex inferences.

Inheritance is the form of reasoning that leads a system to infer properties of a concept based on the properties of ancestors. Inheritance may be defined as the process of determining properties of a concept X, by looking up properties locally attached to X, and if such local information is not available, by looking up properties attached to concepts that lie above X in the conceptual hierarchy.

In Figure 2, the query: `Does Clyde have teeth?` is an example of inheritance problem. Let us examine this query in detail. In this paper, we use *markers* [3] to answer the query. Markers are assigned to certain nodes in the network and propagated to other nodes by way of messages. First, the node `Clyde` may be marked, and markers are propagated to all `Clyde`'s ancestors. Then markers are propagated from `Clyde` and its ancestors to other nodes through the relation `has-part`. Similarly, node `teeth` may be marked, and markers are propagated to all of the descendants of `teeth`. As a result of marker propagations, originated from the extreme nodes, some intermediate nodes may be reached by both markers (e.g. `tusk` and `teeth`). This indicates that there is a path linking `Clyde` and `teeth`, thus the query is positive.

Recognition is the dual of inheritance. While inheritance seeks a property value of a given concept, recognition seeks a concept that has some specified property values. The recognition problem may be defined as: given a description of a set of properties, find a concept or a pattern of concepts that best matches the description.

Again, in Figure 2, the query: `Who has tusk and hates lion?` is an example of recognition problem. In this case, recognition process is straightforward because there exists an exact match in the semantic network. The answer is `elephant`, the most general concept with these properties. Let us consider a non-trivial example of recognition problem. The network is intended to depict the following information [8]:

> `Apple and grapes are fruits. 70% of apples are sweet while 30% of apples are sour. 60% of apples are red while 40% of apples are green. 60% of grapes are sweet while 40% of grapes are sour. 10% of grapes are red while 90% of grapes are green.`

The semantic network for the above English sentences is shown in Figure 3.

In this example, the two user-defined-relations, `has-taste` and `has-color` have weights to indicate the statistical information given from the input. Now, consider the following recognition query: `Is a red sweet object an apple or a grape?`. To solve the above problem, the markers must have the capability of handling weighted information. That is, when markers are propagated through `has-taste` links from the concept node `sweet`, they ac-

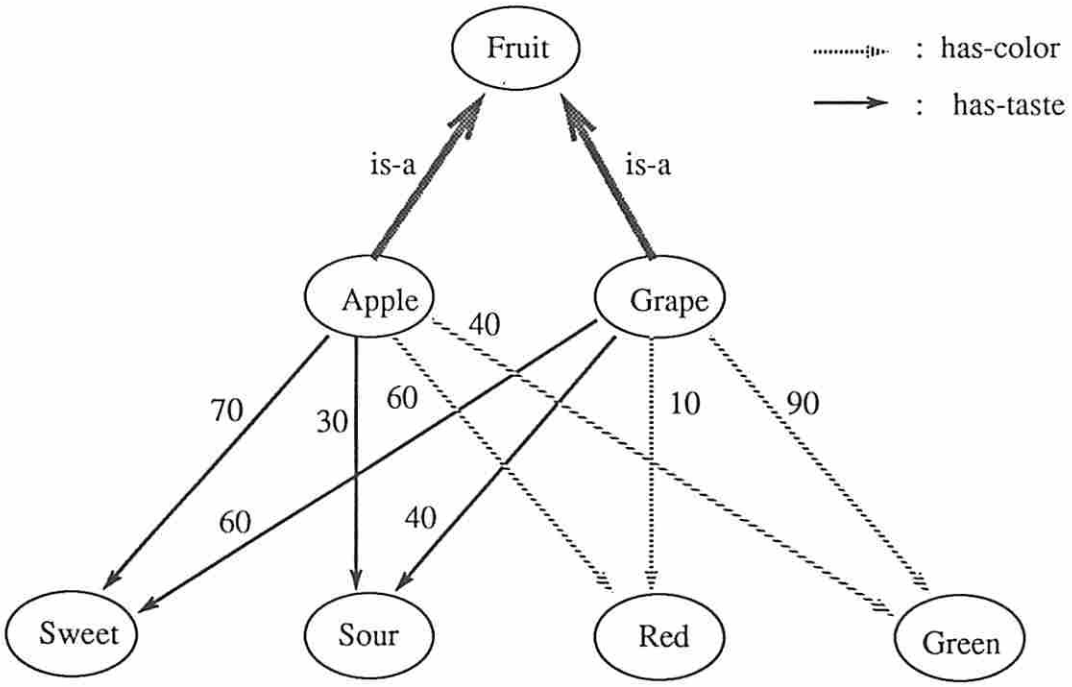

Figure 3: Semantic Network for `Fruit` Example

quire the weights from the corresponding `has-taste` links. As a result of marker propagations from the concept nodes `red` and `sweet`, the concept node `apple` contains two markers with weights 70 and 60 respectively while the concept node `grape` contains two markers with weights 60 and 10 respectively. According to the best estimate rule from Shastri [8], the node which has the greater multiplication results of two markers is the winner. So the answer is `apple`.

In this section, we introduced two important forms of inferences, inheritance and recognition. We found out that in order to implement these inferences the following *operations* are frequently used:

- Search for concept nodes in semantic networks
- Propagate markers through the conceptual hierarchy in semantic networks
- Perform set operations such as intersection and union to identify concepts which share two or more properties, and respectively to identify properties common to several concepts.
- Collect the concept nodes activated through search, marker propagation, and set operations.

In Section 4, Some semantic network instructions are defined which are further used to implement these operations. Search, marker propagation, and set manipulations are highly parallel operations. As discussed in the previous section, we need massive parallel processors to exploit the inherent

parallelism and handle the huge amounts of data seen in real world semantic networks.

The goal of this paper is to show how the Connection Machine can be used to process knowledge represented by semantic networks. Some relevant features of the Connection Machine are discussed in the following section.

3 The Connection Machine

The Connection Machine was originally developed by Hillis [5] to implement Fahlman's NETL [3] , the marker propagation programs for retrieving data from semantic networks. Since then, due to its massive parallelism and good numeric capabilities it has been used as a much more general purpose computer.

Currently, we use the Connection Machine Model CM-2 [9] available at USC-ISI, hereafter CM-2, which is a fine grained massive parallel processor operating in SIMD fashion. Each processor has 64K bits of bit-addressable local memory and a serial ALU. A fully configured CM-2 has 64K data processors (ISI's CM-2 consists of 16K data processors) and 512 megabytes of memory that can be read or written at about 300 gigabits per second. Parallel data structures are spread across the data processors, with a single element stored in each processor's memory. When parallel data structures have more than 64K data elements, the hardware operates in virtual processor mode, presenting the user with a large number of processors, each with a correspondingly smaller memory.

One of the most important requirements of general purpose data parallel

computing is the ability of the data elements to communicate information among themselves in patterns that vary according to the problem and with time. The CM-2 system provides two forms of communication within the parallel processing unit. The more general mechanism is known as the *router*, which allows any processor to communicate with any other processor. The messages may be of any length. Each CM-2 processor chip contains one router, which serves 16 data processors on the chip. For a fully configured CM-2 system, the network is a 12-cube connecting 4096 processor chips. The CM-2 parallel processing unit also has a somewhat faster communication mechanism called the *NEWS grid* which is a programmable grid of arbitrary dimensions. The advantage of this mechanism over the router is merely that the overhead of explicitly specifying destination addresses is eliminated; for many applications this is a worthwhile optimization.

The computation style of the Connection Machine matches the natural computational parallelism inherent in many data-intensive problems. Particularly, in the case of semantic networks, each node in a semantic network can be directly mapped into each data processor regardless of the actual number of nodes in the semantic network. This natural mapping of the problem to the machine coupled with the rich interprocessor connections offered by CM-2 leads to massive parallel solution for knowledge processing. For example, marker propagation times are linear functions of the *critical path* or maximum depth of semantic networks, as will be verified through a couple of example programs in Section 5.

4 Semantic Network Programming on the Connection Machine

4.1 Representation of Semantic Network on the Connection Machine

For our purpose, a semantic network is a colored directed graph. As mentioned above, a semantic network can be mapped directly into the Connection Machine, that is, each node is stored in one data processor and a data processor has full information about the node and its connections. Thus, the data structure is a doubly linked list which allows processing of data items in the list in either forward or backward direction. This mapping is illustrated in Figure 4.

Based on the above representation scheme, we define the detailed software data structure which will be used later for our semantic network programming. Due to the ability of the Connection Machine to handle bit-addressable local memories, the space of each slot in the data structure is allocated at bit-level. The data structure for each node is shown in Figure 5 and has the following slots:

name (10 bits) : the name of the concept node.

flag (16 X 1 bits) : each node has 16 general purpose flags (these flags can be used to identify the concept nodes activated through search, marker propagation, and set operations).

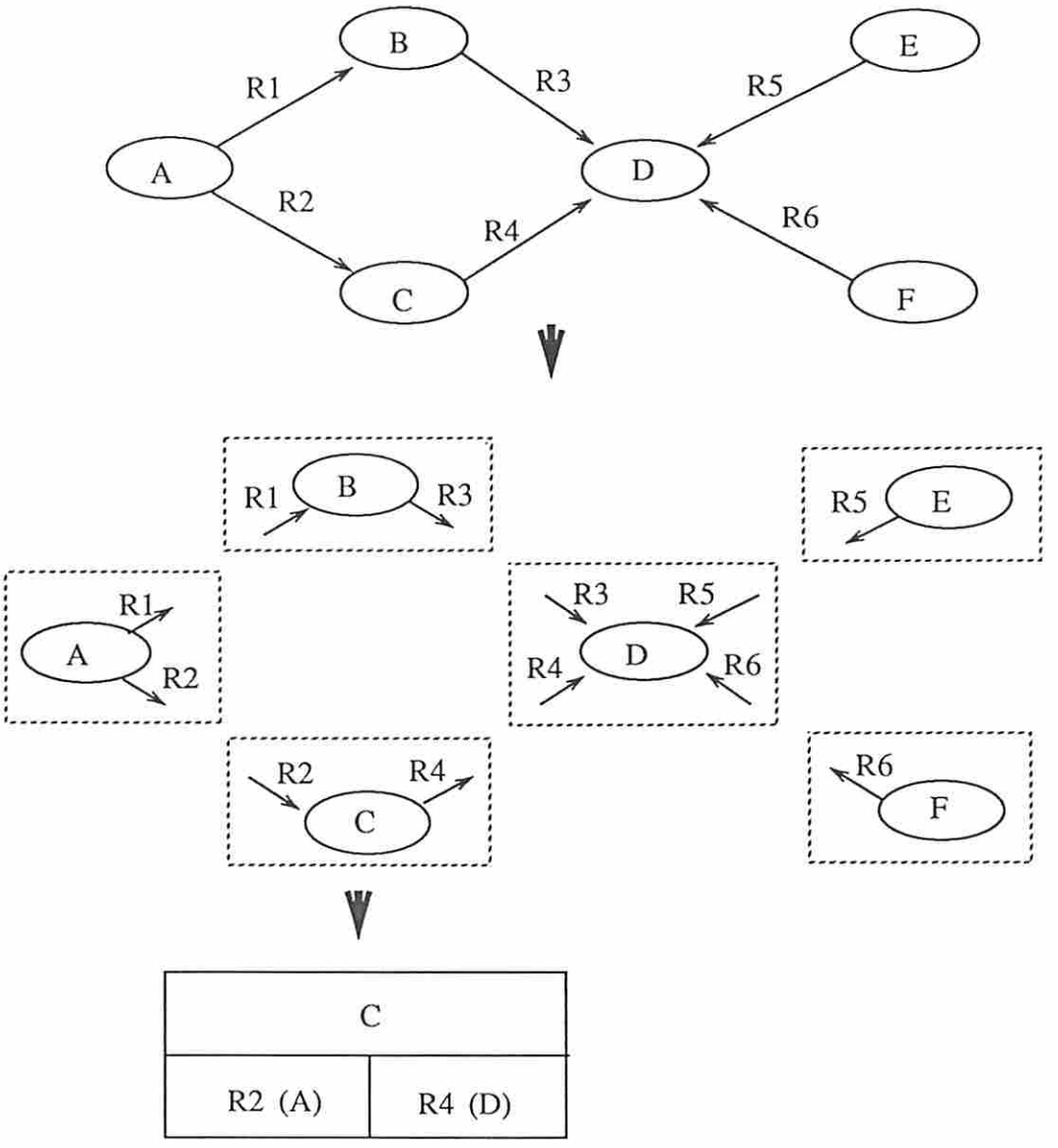

Figure 4: A Mapping between a Semantic Network and The Connection Machine

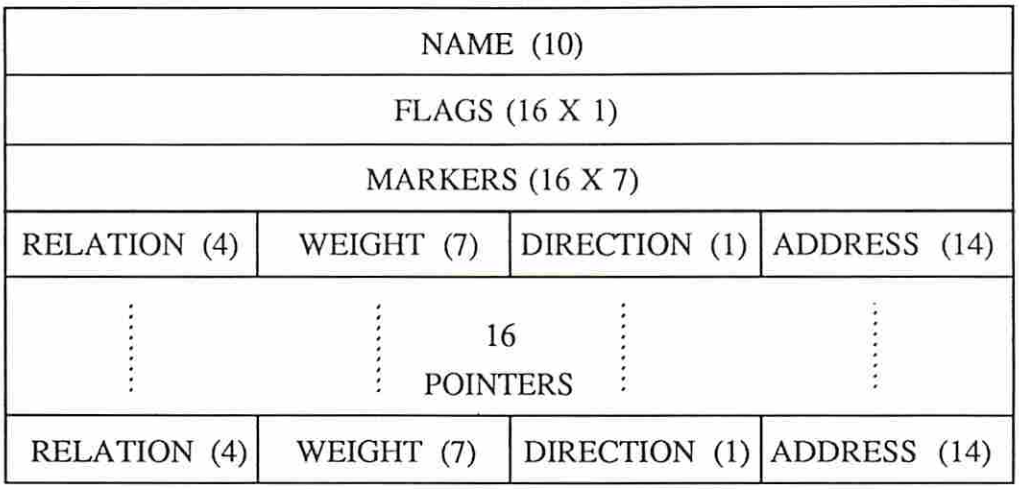

Figure 5: Data Structure for Each Node

marker (16 X 7 bits) : each node has 16 markers which are used to spread information through the conceptual hierarchy (for best match, each marker can hold a weight between 0 and 100).

pointer (16 X 26 bits)

relation (4 bits) : the name of the relation connecting this node with another node.

weight (7 bits) : the weight of a user-defined-relation.

direction (1 bit) : the direction of the relation.

address (14 bits) : the address of the related node.

According to the above data structure, concept node C in Figure 4 is actually stored as shown in Figure 6.

C			
(flags and markers will be used			
during the inferencing process)			
R2	100	nil	address of A
R4	100	t	address of D
⋮	⋮	⋮	⋮

Figure 6: An Example Data Structure

In the above data structure, the number of bits can be expanded according to the need of specific semantic networks. To represent the above data structure in each data processor, about 554 bits are needed. In CM-2 systems, each data processor has 64K bits. Therefore, we can provide enough memory to represent our semantic network model.

The slot *marker* plays an important role in providing inferencing capability for the semantic network. Markers can be assigned to certain concept nodes activated by search operations. These markers are then propagated to other nodes via pointers. Every time a marker is propagated, the corresponding marker bit is set. For best match, as mentioned in Section 2, a marker can handle the weighted information from the corresponding user-defined-relation.

Each node has 16 *pointer* slots and each slot contains *relation*, *weight*, *direction*, and *address* sub-slots. The sub-slot *relation* is another important

component of the data structure. Currently, the following relations are defined:

is-a : an inheritance relation between concept nodes

user-defined-relation : a property relation between concept nodes, like has-part, corresponding to the notion of *roles* in KL-ONE [2]. In this case, the distinction between transitive relation and non-transitive relation must be specified.

cancel : a relation to be used when an exception exits in the property of a special type under a supertype.

split : a relation to separate exclusive types.

generic : a relation to group up tokens with same characteristics.

In this section, we represented semantic networks on the Connection Machine in the form of a software data structure. In the following sections, based on this representation, we define the marker propagation rules and the semantic network instruction set.

4.2 Marker Propagation Rules

When markers propagate from one node to other nodes via relations, the particular relations needed to carry out the given inference must be specified. Otherwise, a marker placed on a node would propagate to all other nodes connected to it by relations. The marker propagation rules govern how

markers are passed in the network. To implement our semantic network examples, the following marker propagation rules are necessary (one can define new propagation rules for more complex inferences, not considered in this paper):

subsumption : markers propagate to all nodes that subsume the current node.

individuation : markers propagates to all nodes that are subsumed by the current node.

user-defined-relation : markers propagate to all nodes that are linked by user-defined-relation from the current node.

4.3 Semantic Network Instruction Set

Semantic network instructions are written in *Lisp, a parallel extension to Common Lisp [10]. Each instruction is processed as a combination of sequential mode in the host machine and parallel mode in the Connection Machine. That is, for each instruction control portions are executed in the host machine and parallel computations are done in the Connection Machine. There are 11 primitive instructions: create, delete, search, propagate, collect, best_match, and, or, not, and_marker, and or_marker. Create and delete are used to maintain semantic networks while others are used for inferences. In Figure 7, the instructions are classified into several groups according to their functions.

Instruction	Function
CREATE DELETE	Node Maintanence
SEARCH	Search
PROPAGATE	Marker Propagation
COLLECT BEST_MATCH	Data Retrieval
AND OR NOT AND_MARKER OR_MARKER	Logical

Figure 7: A Classification of The Semantic Network Instruction Set

In more details, the operation performed by each instruction is described below:

- CREATE <node1> <relation> <node2>
  CREATE is a function that creates new nodes or new relations between existing nodes.

- DELETE <node1> <relation> <node2>
  DELETE is a function that deletes a relation between <node1> and <node2>.

- SEARCH <node> <relation> <flag>
  SEARCH is a function that searches <node> with <relation> and sets <flag> on.

- PROPAGATE <flag> <marker> <propagation-rule-1> <propagation-rule-2>
  All cells with <flag> on, will be given <marker>. <marker> will then propagate through <propagation-rule-1> (if <propagation-rule-2> is provided, then both rules are used for propagation).

- COLLECT <flag>
  COLLECT is a function that collects the names of the nodes that have <flag> on.

- BEST_MATCH <marker1> <marker2>
  BEST_MATCH is a function that collects the name of the node that best matches the given description.

- AND <flag1> <flag2> <flag3>

  AND is a function that performs a logical-and of <flag1> and <flag2>. <flag3> is set or reset based on the result of the logical-and.

- OR <flag1> <flag2> <flag3>

  OR is a function that performs a logical-or of <flag1> and <flag2>. <flag3> is set or reset based on the result of the logical-or.

- NOT <flag>

  NOT is a function that negates the status of <flag>.

- AND_MARKER <marker1> <marker2> <flag>

  AND_MARKER is a function that performs a logical-and of <marker1> and <marker2>. <flag> is set or reset based on the result of the logical-and.

- OR_MARKER <marker1> <marker2> <flag>

  OR_MARKER is a function that performs a logical-or of <marker1> and <marker2>. <flag> is set or reset based on the result of the logical-or.

5 Example Programs and Their Performance

In the previous sections, the data structure, the marker propagation rules, and the semantic network instruction set were defined. Now, some example programs on the Connection Machine are presented.

The execution time can be measured by using the CM-2 *Paris* primitive called (cm: time) [9]. It provides total elapsed time and actual CM time, where CM clock speed is 6.47443 MHz. The total elapsed time includes the time spent in the front-end machine, in our case it is VAX 6210 which is a slow time-sharing machine that hardly gives us accurate timing. In order to measure the total elapsed time accurately, every example was executed more than 10 times and the total elapsed time was averaged.

5.1 Example 1: Inheritance (`Clyde, the circus elephant`)

The semantic network for `Clyde, the circus elephant` was shown in Section 2. The semantic network is loaded on the Connection Machine using the instruction: (create node1 relation node2). The query is: `Does Clyde have teeth?`. A program implementing this query and its time measurements are shown in Figure 8.

In this example, the total cm-time is about 0.202 seconds, and most of this cm-time is spent in the *propagate* instructions. Therefore, marker propagation operations are the most time consuming operations and consequently dominate the performance.

Let us focus on the major marker propagation instruction: (propagate F0 M0 sub has-part). In this instruction, the marker M0 is propagated through subsumption link or has-part link starting from the node `clyde`. Considering Figure 2, the longest path for the marker propagation is: `clyde, circus elephant, elephant, elephant's head, elephant's mouth, tusk, teeth`. In this case, the length of the critical path, or maximum depth is 6. The

Instruction	actual cm-time (second)	total elapsed time (second)	percentage of cm-time
(SEARCH Clyde nil F0)	0.022367	0.07	31.95%
(PROPAGATE F0 M0 sub has-part)	0.119885	2.29	5.23%
(SEARCH Teeth nil F1)	0.022394	0.07	31.99%
(PROPAGATE F1 M1 ind nil)	0.032734	0.61	5.36%
(AND_MARKER M0 M1 F2)	0.000000	0.01	0.00%
(COLLECT F2)	0.005047	1.19	0.00%

Figure 8: Instructions and Time Measurements for Example 1

critical path must be followed sequentially during the parallel propagation of marker in the Connection Machine. Therefore, the performance of semantic networks is a function of the length of the critical paths.

5.2 Example 2: Inheritance (An imaginary two-dimensional network of size 10 X 10)

In example 1, we tested a semantic network of small size. Now, we create an imaginary semantic network of much larger data size, which will lead to more realistic performance measurements. This imaginary semantic network is shown in Figure 9.

In this imaginary 10 X 10 mesh network, we ask the following query: `Does node 0 have node 99?`. As observed in example 1, the performance is dominated by marker propagation operations. The length of the critical path for the major marker propagation is 18. The marker propagation occurs in the semantic network instruction: (propagate F0 M0 sub has-part). A complete program and its time measurements are shown in Figure 10.

In this example, the cm-time of the major marker propagation is about 0.333 seconds while the total cm-time is about 0.519 seconds. As discussed above, the major marker propagation dominates the performance.

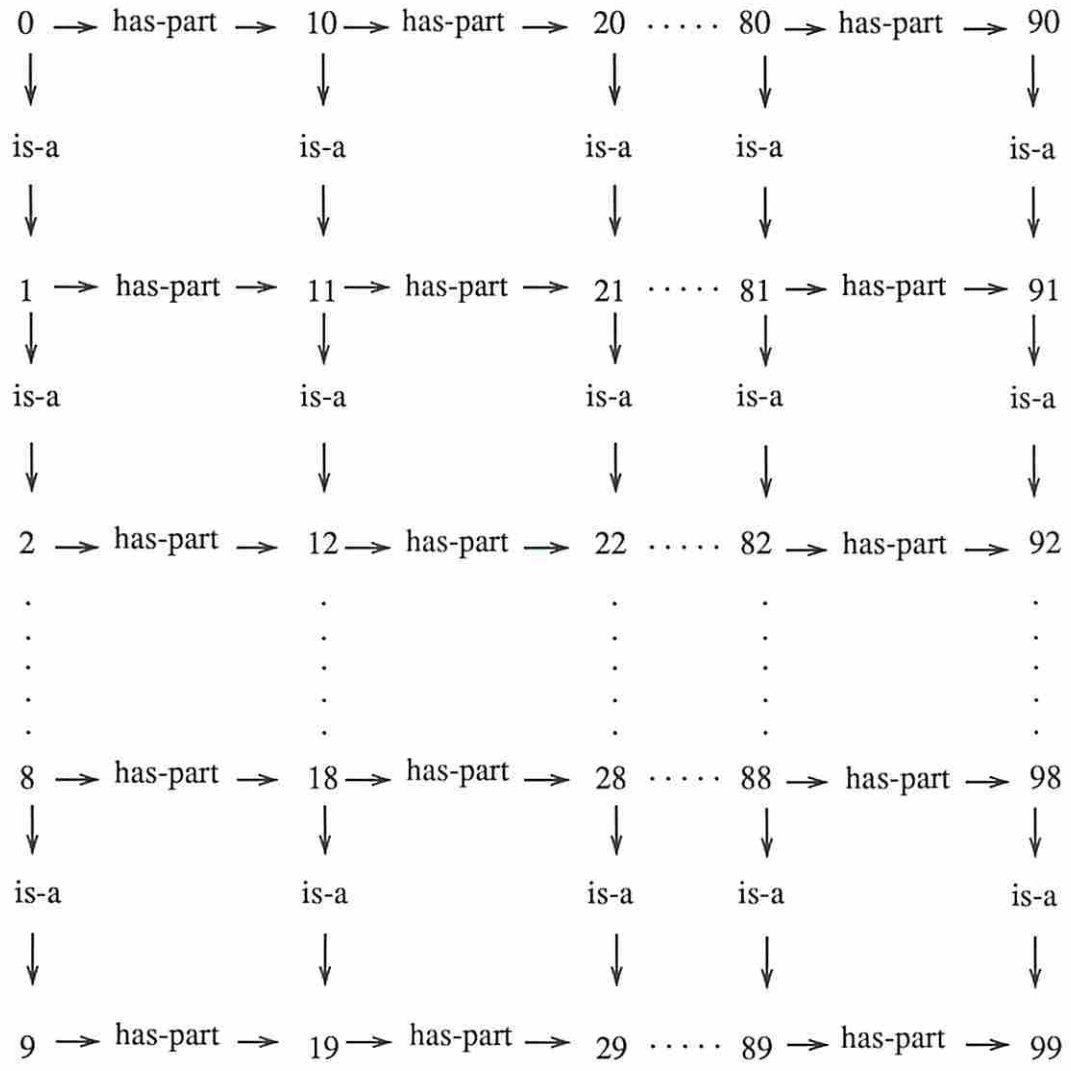

Figure 9: An Imaginary Semantic Network

Instruction	actual cm-time (second)	total elapsed time (second)	percentage of cm-time
(SEARCH Node0 nil F0)	0.002368	0.01	23.68%
(PROPAGATE F0 M0 sub has-part)	0.332931	6.31	5.27%
(SEARCH Node99 nil F1)	0.002445	0.01	24.45%
(PROPAGATE F1 M1 ind nil)	0.174663	3.28	5.32%
(AND_MARKER M0 M1 F2)	0.000000	0.01	0.00%
(COLLECT F2)	0.006739	1.22	0.01%

Figure 10: Instructions and Time Measurements for Example 2

Instruction	actual cm-time (second)	total elapsed time (second)	percentage of cm-time
(SEARCH Node0 nil F0)	0.002646	0.01	26.46%
(PROPAGATE F0 M0 sub has-part)	3.464537	59.59	5.81%
(SEARCH Node9999 nil F1)	0.002081	0.01	20.81%
(PROPAGATE F1 M1 ind nil)	1.692098	34.14	4.95%
(AND_MARKER M0 M1 F2)	0.000000	0.01	0.00%
(COLLECT F2)	0.015172	2.18	0.69%

Figure 11: Instructions and Time Measurements for Example 3

5.3 Example 3: Inheritance (An imaginary two-dimensional network of size 100 X 100)

In order to study the effect of semantic network size on the performance, we have increased the size of the semantic network from the previous example to 100 X 100. The query is: `Does node 0 have node 9999?`. The major marker propagation from `node 0` to `node 9999` has the critical path of length 198. A complete program and its time measurements are shown in Figure 11.

In this example, the cm-time of the major marker propagation is about

3.465 seconds while the total cm-time is about 5.177 seconds. As in the previous examples, the marker propagation time dominates the performance. When comparing this example with the previous examples, we observe that the execution time increases proportional to the length of the critical paths, but not proportional to the semantic network size.

5.4 Example 4: Best Match Recognition

In Section 2, we considered an example of best match recognition problem shown in Figure 3. The query was: `Is a red sweet object an apple or a grape?`. A program implementing this query and its time measurements are shown in Figure 12.

This example is different from the previous inheritance examples. No major marker propagation exists, that is, the marker propagation operations encounter critical paths of length 1.

5.5 Example 5: Recognition with Multiple Properties

In the previous recognition example, only two marker propagations were needed to carry out the given query. Now, we consider an example which requires multiple marker propagations. A semantic network example is shown in Figure 13. The query is: `Among the graduate students, who meet the following description: single, male Ph.D. student who lives on-campus, has a RAship, and has a car?` This inference is a recognition problem with multiple properties which requires multiple marker prop-

Instruction	actual cm-time (second)	total elapsed time (second)	percentage of cm-time
(SEARCH Red nil F0)	0.023651	0.07	33.78%
(PROPAGATE F0 M0 has-color nil)	0.030625	0.49	6.25%
(SEARCH Sweet nil F1)	0.023452	0.07	33.50%
(PROPAGATE F1 M1 has-taste nil)	0.032628	0.49	6.65%
(BEST_MATCH M0 M1)	0.009413	1.05	0.89%

Figure 12: Instructions and Time Measurements for Example 4

agations. A complete program and its time measurements are shown in Figure 14.

As in Example 4, the length of the critical path is 1, and no major marker propagation exists. But, the query contains 6 marker propagations. In this type of query, the execution time is influenced on how many marker propagations are needed to carry out the given query.

5.6 Performance Comparison between Examples

Five different semantic network examples were presented in the previous section. In this section, we analyze the performance by comparing the time measurements of these examples. The table shown in Figure 15 summarizes the results.

Consider the first three examples which are inheritance problems. We observe that major marker propagations dominate the performance, that is, the execution time is proportional to the length of the critical path.

From Example 4 and 5, one can make a different observation. The execution time is not proportional to the length of the critical path because in those examples, the length of the critical path is trivial. However, in spite of the trivial critical path, Example 5 takes more execution time than Example 1 because Example 5 requires 3 times more marker propagations than Example 1. Therefore the execution time is also proportional to the number of marker propagations in case the given inference contains multiple properties.

The recognition problems considered in Example 4 and Example 5 have the critical paths of length 1, which means the target concepts are locally con-

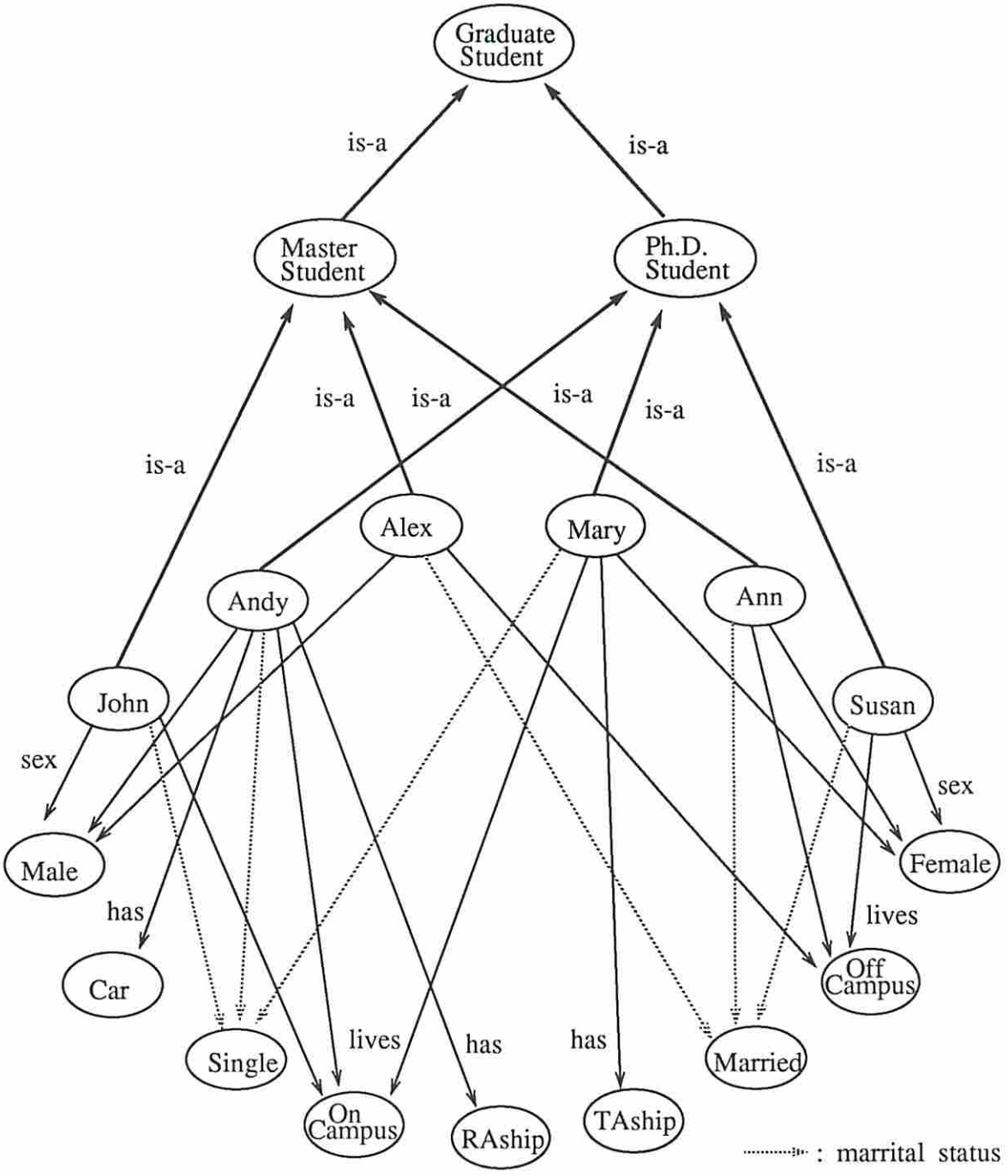

Figure 13: An Example for Recognition with Multiple Properties

Instruction	actual cm-time (second)	total elapsed time (second)	percentage of cm-time
(SEARCH Ph.D. Student nil F0)	0.024090	0.07	34.41%
(PROPAGATE F0 M0 sub nil)	0.039896	0.62	6.43%
(SEARCH Male nil F1)	0.024020	0.07	34.31%
(PROPAGATE F1 M1 sex nil)	0.034798	0.61	5.70%
(SEARCH Single nil F2)	0.024191	0.07	34.55%
(PROPAGATE F2 M2 marrital-status nil)	0.037874	0.61	6.20%
(SEARCH On-Campus nil F3)	0.024084	0.07	34.40%
(PROPAGATE F3 M3 lives nil)	0.039476	0.61	6.47%
(SEARCH Raship nil F4)	0.023745	0.07	33.92%
(PROPAGATE F4 M4 has nil)	0.039825	0.61	6.52%
(SEARCH Car nil F5)	0.024182	0.07	34.54%
(PROPAGATE F5 M5 has nil)	0.034549	0.62	5.57%
(AND_MARKER M0 M1 F6)	0.000000	0.01	0.00%
(AND_MARKER M2 M3 F7)	0.000000	0.01	0.00%
(AND_MARKER M4 M5 F8)	0.000000	0.01	0.00%
(AND_FLAG F6 F7 F9)	0.000000	0.01	0.00%
(AND_FLAG F7 F8 F10)	0.000000	0.01	0.00%
(AND_FLAG F9 F10 F11)	0.000000	0.01	0.00%
(COLLECT F11)	0.008118	0.21	3.86%

Figure 14: Instructions and Time Measurements for Example 5

	length of critical path	number of marker propa-gations	maximum number of branches	size of semantic network	major marker propa-gation time	actual cm-time	total elapsed time
Example 1 (inheritance)	6	2	2	17	0.120 sec	0.202 sec	4.24 sec
Example 2 (inheritance)	18	2	2	100	0.333 sec	0.519 sec	10.84 sec
Example 3 (inheritance)	198	2	2	10000	3.465 sec	5.177 sec	95.94 sec
Example 4 (recognition)	1	2	2	7	0.033 sec	0.120 sec	2.17 sec
Example 5 (recognition)	1	6	3	18	0.040 sec	0.379 sec	4.37 sec

Figure 15: Summary of Performance Measurements

nected to the properties involved in the given query. A complex recognition problem may require searching through conceptual hierarchy to get target concepts. In that case, the length of the critical path will be increased.

Note that from Example 1 to Example 5, branching factors are at most 3. In general, the branching factors influence the performance because the marker propagations have to be done sequentially when the corresponding nodes have multiple branches.

From Example 1 to Example 5, we observe that the size of semantic networks does not affect the performance directly. This is mainly due to simultaneous marker propagations in the network.

By combining the above discussions, we conclude that the performance is proportional to the length of critical path, the number of marker propagations, and branching factor.

When comparing total elapsed time with cm-time, we realize that considerable execution time is spent on the host computer, mainly because of the following three reasons:

1. The Connection Machine is an SIMD machine, where the front-end has to control the sequence of program execution, in other words, a lot of interactions between the front-end and the Connection machine take place during program execution.

2. The semantic network programs are written in *Lisp, a high level language, in which the Connection Machine primitives can only be interpreted or macro-expanded at run time. To avoid this problem, one could directly use Paris, the low level language for the Connection Ma-

chine.

3. The front-end, VAX 6210, is a slow time-sharing machine.

6 Conclusions

In the previous sections, we introduced semantic network concepts and the Connection Machine, and showed how a semantic network model may be mapped and programmed on the Connection Machine. According to the example programs shown in Section 5, the execution times are proportional to the length of the critical path in the semantic networks, and also proportional to the number of marker propagations in case the inference contains multiple properties. Branching factor also affects the performance.

The Connection Machine does not support multiple marker propagations efficiently, because the Connection Machine operates on SIMD mode, that is, each node in the Connection Machine can send one marker at a time. As a result, the parallelism existing in multiple marker propagations cannot be exploited on the Connection Machine, in other words, different marker propagations cannot be overlapped. As mentioned in the previous section, the marker propagation takes most of the cm-time during the program execution. Therefore, this limitation might be a bottleneck in real world applications with huge amounts of data.

Although the Connection Machine is a viable tool for semantic network processing, further improvements are possible if some key operations such as marker propagations were implemented directly in hardware so multiple

marker propagations became possible. Another direction for improvement is to design a specialized controller to better control the operation of the machine.

References

[1] Bobrow, D. G. and Winograd, Terry "An Overview of KRL: A Knowledge Representation Language," *Cognitive Science* 1, 3-46, 1977.

[2] Brachman, Ronald J. and Schmolze, James G. "An Overview of The KL-ONE Knowledge Representation System," *Cognitive Science* 9, 171-216, 1985.

[3] Fahlman, S. E. *NETL: A System for Representing and Using Real-World Knowledge.*, The MIT Press, Cambridge, MA, 1979.

[4] Feldman, J. A. and Ballard, D. H. "Connectionist Models and Their Properties," *Cognitive Science* 6, 205-254, 1982.

[5] Hillis, W. Daniel *The Connection Machine.*, The MIT Press, Cambridge, MA, 1985.

[6] Quillian, M. R. "Semantic Memory," *Semantic Information Processing*, M. Minsky(Ed.), 216-270, The MIT press, Cambridge, MA, 1968.

[7] Roberts, B. and Goldstein, I. *The FRL Manual MIT AI Memo 409*, 1977.

[8] Shastri, Lokendra *Semantic Networks.*, Pitman Publishing, London, Great Britain, 1988.

[9] Thinking Machine Corp. *Connection Machine Technical Summary*, Thinking Machine Corp., Cambridge, MA, 1987.

[10] Thinking Machines Corp. **Lisp Release Notes*, V 5.0, Thinking Machine Corp., Cambridge, MA, 1988.