Text Understanding and Question Answering with SNAP

Technical Report No. CENG 89-12

Ig-Tae Um, Ronald Demara, Juntae Kim
Dan Moldovan

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-0781

October 2, 1989

[0]This research has been funded by the National Science Foundation Grant No. MIP-89/02426

Abstract

This paper describes the issues in the design of the SNAP/NLP natural language processing system. This prototype system utilizes the architectural features and inherent parallelism of the Semantic Network Array Processor (SNAP). A predetermined set of domain background knowledge and a lexicon for a specific application domain were implemented for SNAP. The lexical entries are represented by a graph with syntactic categories and word sense patterns. SNAP/NLP uses this knowledge to function in two modes of operation: a text understanding mode and a question answering mode. The text understanding mode consists of a syntactic parser, a semantic interpreter, a discourse analyzer and a classifier. The Augmented Transition Network (ATN) based parser generates a parse tree with deep structure information. The parse tree guides the construction of a kind of conceptual graph for each sentence by the graph unification based semantic interpreter. The discourse analyzer constructs a theme for each paragraph and performs anaphoric reference. The classifier locates the proper place in the semantic network knowledge base for the new graph. In the question answering mode, the semantic network knowledge is utilized to generate responses to user queries. Algorithms utilizing SNAP's parallelism have been designed for semantic interpretation, anaphoric reference, classification and query processing.

Contents

1 Introduction

Natural language understanding is an intricate and complex process. The difficult aspects of natural language analysis range from resolution of lexical and structural ambiguities to performance of loosely defined inferences and management of the knowledge base. Ambiguities can lead to incorrect interpretation and increased time complexity of processing. Similarly, many idiomatic and metaphoric usages of words, phrases, and sentences make the correct analysis nearly impossible in certain circumstances [8]. Regarding the machine processing, the major reasons for this problem are insufficient syntactic, semantic, and domain knowledge, inflexible or weakly structured knowledge representation, and a lack of powerful hardware support [22].

In prior research, effort was concentrated on finding the most useful language features and developing the knowledge representation system with powerful expressiveness. This resulted in several significant approaches such as Quillian's *semantic network* [15], Fillmore's *case grammar* [9], and Woods's *ATN parser* [25]. These approaches became the most popular starting points of future natural language processing research.

However, most natural language processing systems have not gotten good results in correct processing because they lack a sufficiently powerful knowledge representation scheme. Even in a limited domain, a great deal of knowledge must be kept in the system's knowledge base. In recent research, the breadth and depth of the system's capability are pursued on the basis of the common natural language processing paradigms. Some systems emphasize breadth, and some emphasize depth, while others seek a balance of both.

Several recent techniques have appeared that are furthering the fundamental concepts in new directions. Approaches such as the *KL-ONE family representation language* [5] [4], Schank's *conceptual dependency theory* [6], Wilks' *preference semantics* [23] and Sowa's *conceptual graph theory* [18] are setting the direction for the current state of the art natural language systems. These techniques helped resolve some of the above ambiguity problems [2] [7] [16] [12] [20] [21], however, limitations still exist with respect to the overall processing paradigm. Since this paradigm assumes a sequential hardware architecture, the problems of the limitation of the knowledge representation and the inherent complexity of search and match algorithms have been left unsolved.

This paper describes an integrated approach that extends the above techniques with parallel algorithms and dedicated hardware. The proposed approach involves a knowledge processing architecture called the *Semantic Network Array Processor* (SNAP) [13] and a Natural Language Processing application system SNAP/NLP. The SNAP architecture provides the capability of versatile graph representation. In addition to the semantic networks and a powerful reasoning mechanism (marker propagation), the proposed approach has more expressive and manipulative power for knowledge representation to greatly reduce the processing complexity.

Due to this dedicated hardware support, the knowledge representation can be more expressive and its manipulation more responsive to the demands of the AI operations that occur during natural language processing. This expressive power helps to establish the viability of a method to deal with pronoun and definite noun phrase reference problems in addition to lexical and structural ambiguities. Furthermore, SNAP's associative search and match can considerably reduce the overall processing time.

Architectural Overview of SNAP

SNAP(Semantic Network Array Processor) is a highly parallel architecture targeted to AI applications. SNAP consists of 256 custom-made chips, and each chip has 64 nodes with their associated pointers. Concepts in the semantic network are mapped into the nodes, and each link is implemented through the use of hardware pointers.

The knowledge base is distributed over the SNAP array, and the processing is done locally using *marker propagation*. The marker can be set by the SNAP controller to any arbitrary node in the array, and it can be propagated to other nodes according to the propagation rule. The propagation rule can also be given by the controller using SNAP instructions. In SNAP, several markers can be propagated simultaneously giving rise to parallelism. Reasoning mechanisms, such as inheritance inferencing, can be performed in parallel by using these capabilities. We will describe briefly how we can make use of these capabilities for our system.

2 Overview of SNAP/NLP

The primary functions of the SNAP/NLP system are *text understanding* and *query answering*. Each function is performed exclusively in different operating modes. The complete system consists of *processing modules*, *knowledge base modules* and a *system operator*, which cooperate to perform the tasks of each mode. (see Figure 1).

Text understanding is a process of constructing a knowledge base from the symbols in the input text. The process is decomposed into steps of syntactic parsing, semantic interpretation, discourse analysis, and hierarchical classification. The syntactic parsing process constructs a parse tree along with surface and deep structures for each sentence based on an ATN grammar. The semantic interpretation step generates meaning representation for each parse tree based on a graph unification process with background knowledge. The discourse analysis relates the current sentence to the previous sentences. The hierarchical classification procedure populates the knowledge base by storing the correct meaning representation at the proper location in the hierarchy in order to maintain consistency in the knowledge base. To promote modularity and improve maintainability of the system, each stage of the process is carried out in a corresponding processing module. The related processing modules are called the syntactic parser, semantic interpreter, discourse analyzer and classifier.

Question answering is the dual of text understanding. It consists of retrieving knowledge from the knowledge base. The question answering process consists of query parsing, query interpretation, query processing, and answer generation. First, a query is converted into a interrogative meaning representation through parsing and interpretation. The query meaning representation is matched against the knowledge base after the query processing stage. The answer generation process shapes the result of the query processing into a natural language answer. The related processing modules include the query parser, query interpreter, query processor, and answer generator.

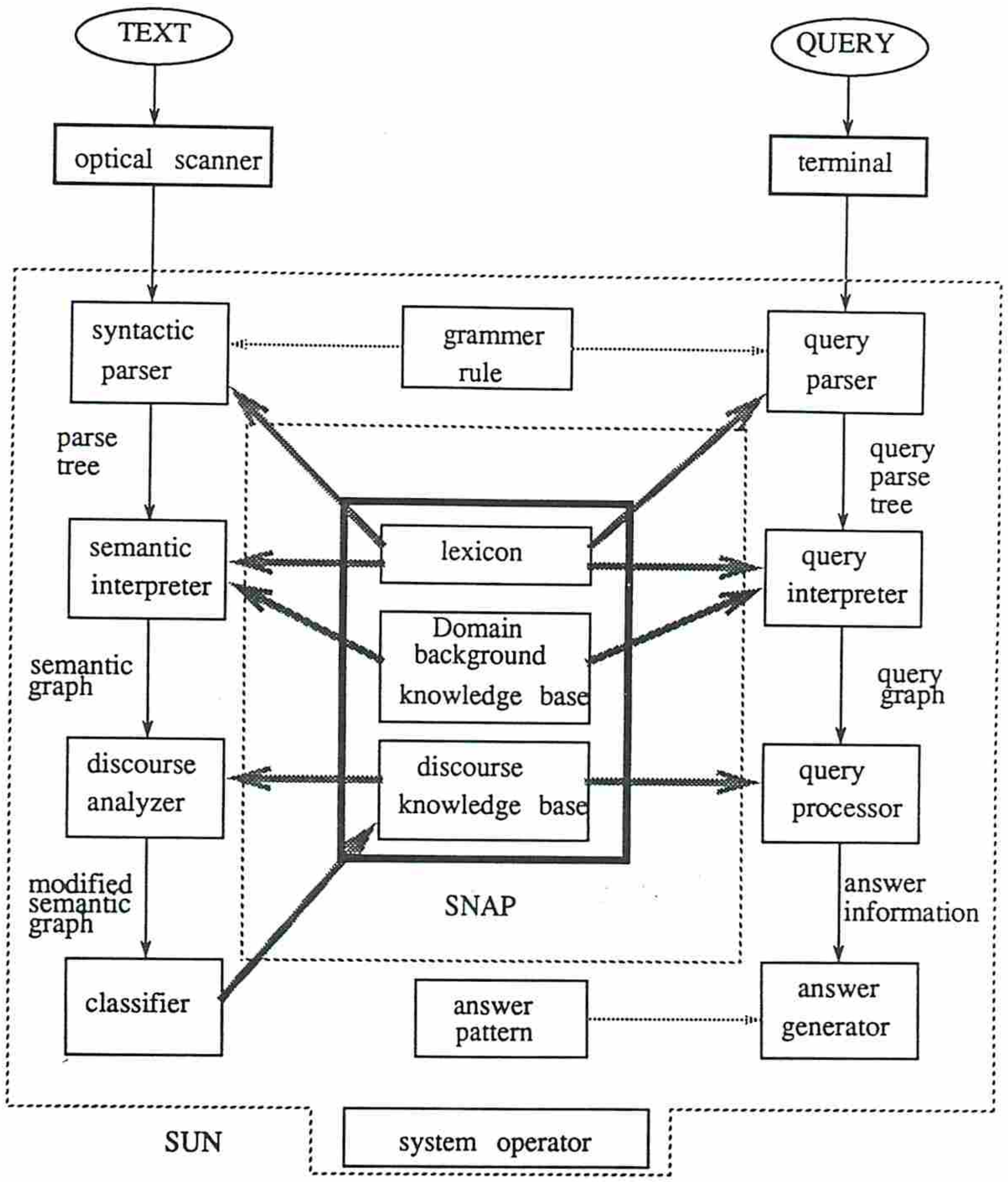


Figure 1: System Block Diagram

An additional module, the system operator module, uses metarules to decide which primary procedures to execute at any given time. If there is any exception such as an unknown word, the processing module requests the system operator for the exception handling. If the request is not within the capability of a callable procedure from the system operator, it will request the intervention of the user for more information of the unknown word or phrase.

The knowledge base for the system is partitioned into a background knowledge base and a discourse knowledge base. The background knowledge base consists of a lexicon, a domain background semantic network, and a grammar rule base. The discourse knowledge base is a semantic network which is actually constructed on the basis of the domain background semantic network. Also, additional processing knowledge is contained in each processing module both explicitly and implicitly.

3 Knowledge Representation on SNAP

Artificial intelligence systems are typically developed under the assumption of an application domain. A natural language processing system needs a considerable amount of linguistic knowledge in addition to the domain knowledge [22]. These two types of knowledge are the background knowledge which should be stored in the form of a knowledge base in a system.

In this paper, the basic representation method is based on the use of colored graphs, especially semantic network. This knowledge consists of concepts, conceptual relations, and assertions. An assertion corresponds to a sentence composed of concepts and relations [18]. With the colored graph, concepts and relations are differentiated by the color of concept nodes and relation links.

3.1 Types of Task Domain Knowledge

Task domain knowledge consists of *background and discourse knowledge*. The background knowledge is built-in knowledge while the discourse knowledge is obtained during text understanding process.

Background Knowledge

Computer architecture has been chosen as the domain for developmental purposes. The general background knowledge of the domain is obtained from a corpus representing the essential features of the organization and operation of a digital computer [10]. In SNAP/NLP, this knowledge is currently used to deal only with this domain and a limited set of language features. Since the emphasis of purpose is on investigation and utilization of SNAP's mechanisms for enhancing AI processing, we are dealing primarily with simple sentences at this point. Sentences may consist of various arrangements of noun, verb, and prepositional phrases.

Discourse Knowledge

Discourse knowledge is defined as the knowledge from the progression of assertions in the text. This knowledge thus consists of specific concepts and conceptual relations, propositions for each sentence or subordinate clauses, and themes for the paragraphs. As an example of the discourse knowledge base, consider the network depicted in Figure 2 which shows the representation of the following sentences:

```
SNAP is a processor.
SNAP has a controller.
SNAP has an array.
```

In this figure, each node has its own concept label. Every pair of nodes has a relation type. Among concept nodes, PROPOSITION and THEME nodes are not derived directly from a text. These two concepts are generated as part of the text understanding process. A proposition represents concepts and relations in a sentence as a whole [18]. For representing causality and event sequence, this type of representation technique is extremely useful. The overall theme consists of the set of propositions in a paragraph. A paragraph represents a topic or a closely related description in the discourse of a text. The discourse knowledge is then overlayed and represented in conjunction with the domain background knowledge semantic network so that the overall domain world knowledge can be maintained in one place.

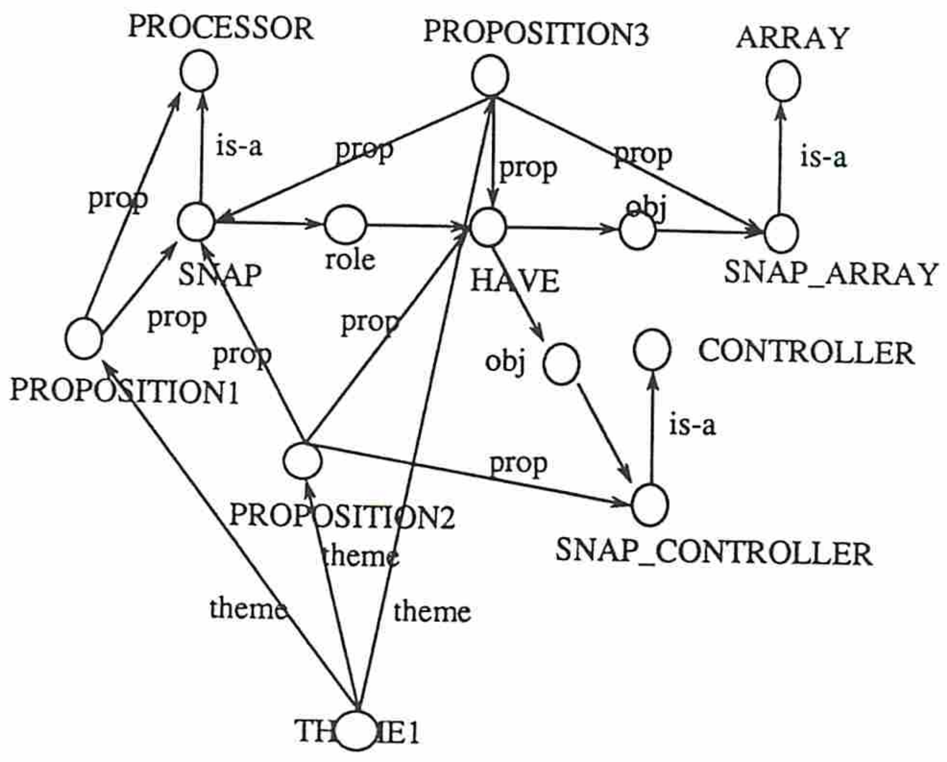


Figure 2: Example Discourse Representation

3.2 Processing Knowledge Bases

The logical view or functional view of natural processing centers around the manipulation of background knowledge of lexical, syntactic, semantic, and thematic varieties. Lexical knowledge contains word's syntactic category and word sense. This knowledge is usually stored in a special data structure called a lexicon. Similarly, syntactic knowledge determines the grammar which defines rules which govern the structure of a sentence. The grammar is usually implemented in some form of a rule base. Semantic knowledge consists of concepts and conceptual relations in terms of a specific domain area. The semantic knowledge helps build the meaning representation of a sentence. Thematic knowledge can be viewed as a focus of interest, topic, or meaning cluster in a domain. Until now, there has been no general agreement on the definition and form of representation for this type of knowledge. The domain knowledge is used to restrict possible word and phrase senses on this case.

The physical view of natural language understanding deals with the components composing the text such as words, sentences, and paragraphs. Words are delimited by a blank between their adjacent occurrences. Sentences are recognized by any one of several punctuation marks, typically periods. In written text there is the additional attribute of paragraph structure which does not occur in spoken dialogue. Each paragraph is delimited by starting and ending blank lines that can be used for recognition during analysis. This information is available since the text file contains these special characters. The domain knowledge provides a default knowledge and restricts the possible word sense and phrase sense.

Lexicon

A sample lexicon entry is shown in Figure 3 for the word *send*. Note that morphological variations are enumerated separately so that *send*, *sent*, *sends*, and *sending* are listed as individual words. Since the word in this example is a verb, it also has associated dimensions such as *Tense* and *Root* which specify the tense of the verb and its principal form, respectively. Since SNAP's relations are user-definable, additional relations can created as necessary to permit the identification of mass and count nouns, demonstrative, quantifiers, and so on. Ideally, each element would have only one syntactic category so that a word that can function in multiple syntactic categories are represented by multiple lexicon entries.

In the semantic definition, a word definition consists of a *semantic pattern*, *synonym* and *antonym*. The semantic pattern shows how a concept is related to other concepts. There are two types of word definition, that is, *conceptual word* and *relational word*. A concept denotes a referent on the world which may be physical or abstract. Usually, nouns, verbs and adjectives have an associated concept while prepositions have only relational meaning. Each conceptual word is represented by its own concept type in conjunction with, relations to other concepts and its *most specific supertype*. Relational words are represented by a relation type. Furthermore, synonyms and antonyms are listed to answer a query like `What does SNAP array receive?`. Even

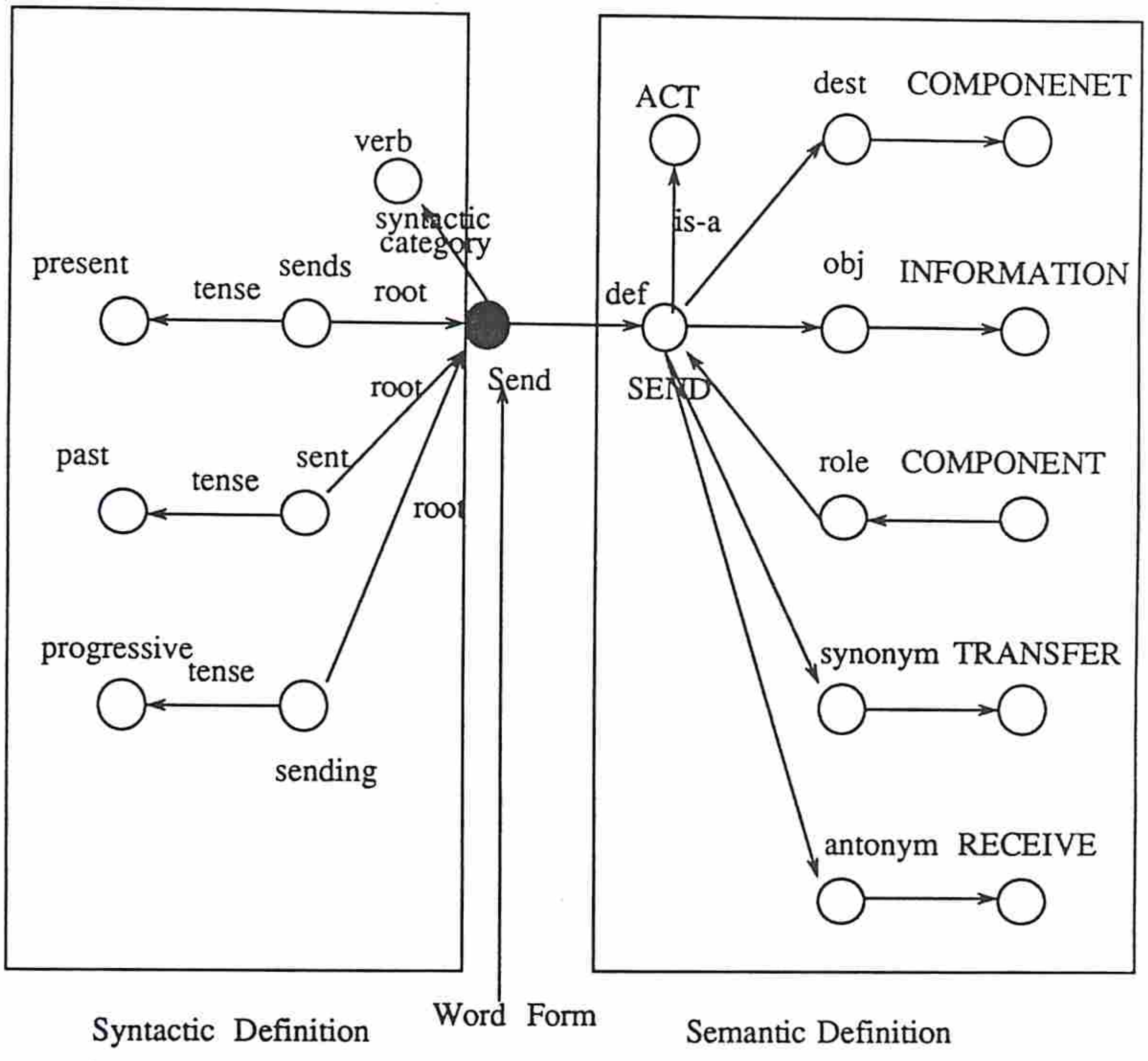


Figure 3: A Lexicon Entry

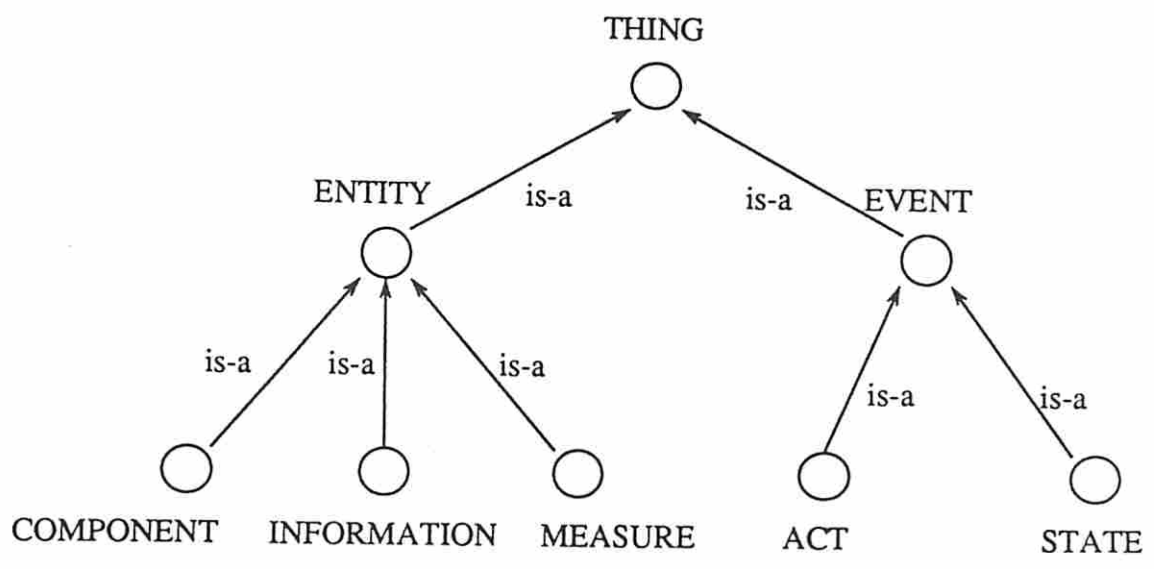


Figure 4: Example Concept Type Hierarchy

if there is no concept RECEIVE in the discourse network, the system still responds to the query due to the antonym information.

Domain Concept Type Hierarchy

Domain background knowledge is the default knowledge associated with the domain. The default knowledge consists of typical concept types and properties which can be obtained interactively from human experts or from existing typewritten materials in the domain. In the SNAP/NLP approach, the knowledge is organized into a semantic network implementing a concept type hierarchy. Figure 4 shows part of the type hierarchy. In this figure, the highest node is THING and all other nodes become the children of this node.

Implementation Considerations on SNAP

When the lexicon is implemented on SNAP, each word is represented by a node in the lexicon network and is thus stored in the content addressable memory of a unique array cell the SNAP array. Relation links corresponding to the attributes of the word are implemented with by SNAP's relational pointers. The pointers express relations to indicate all the pertinent information required to describe each word. This implementation of the lexicon provides several advantages. First, it provides a means for fast associative search of lexical entries. Since the words are stored as cells in the SNAP array, they can be looked up quickly by using the *associative search* capability of SNAP. This lookup can be performed by the issuance of the search instruction which causes each *content addressable memory* to attempt to match the input word. Furthermore, the dimensions can be stored and manipulated with little software overhead because of SNAP's extensive support for relation pointers. The merits of this approach are apparent by contrasting the capability for improved performance and ease of implementation with that of lexicons in conventional text understanding systems.

The current design of SNAP supports only a single semantic network. This implies that the *semantic network for domain background knowledge* and the semantic network for discourse knowledge would need to be implemented on the same physical network. However, the lexicon should be separated from the semantic network because its initial definition should not be affected during the text understanding process. This problem is solved by *tagging* every node in the lexicon. The tag is a special character string which would not otherwise appear in the corpus of text. Before each knowledge processing step, the comparison is made with tag to determine which network the node belongs to.

4 Text Understanding with SNAP

Generally, text understanding is thought as a task to transform the embedded knowledge in a text into an explicit knowledge structure. The process of text understanding is carried out sentence by sentence because a text has a sequential nature in information transfer and its basic unit is a sentence. In this paper, each sentence is assumed to be well-formed so that any kinds of ill-formed sentences are beyond our immediate concern.

4.1 Syntactic Parsing

Syntactic parsing is an important first step in the text understanding process. It is performed in order to convert the input text into a set of formal structures that are suitable for the following processing step.

Creation of the Input File

The body of input text must first be prepared and entered into the Sun workstation. It may be typed-in interactively by the user or it may be scanned-in from a written document using a commercially available Optical Character Reader (OCR). If an OCR is used, a spelling verification program should be applied to the file to remove any errors that may have been introduced during the scanning process. The input file should be structured as a standard flat-formatted ASCII data file. This data file can then be processed to form the knowledge base in SNAP/NLP.

Structure of Syntactic Side of Lexicon

Once the text is resident on the Sun workstation, it is ready for parsing using the information that is already contained in the lexicon and the grammar rule base. The syntactic part of the lexicon catalogues information on each word in a pre-defined language for the domain (see Figure 3). The purpose of the lexicon at this point is to provide information useful for determining the syntactic role of the each word, rather than evaluating its contribution to sentence meaning. Example syntactic relations include the lexical category or part of speech for along with a corresponding set of dimensions which describe the range of properties that each word possesses.

Surface Structure Analysis

The surface structure analysis is obtained by utilizing the syntactic information contained in the lexicon along with a set of grammar rules which specify the composition of an acceptable sentence. The sentences in the input text are parsed one at a time starting from the beginning of the file until the entire body of text has been processed. First, a pre-processing step is performed to condition the input text such that it will be acceptable to the parser. Pre-processing is necessary to deal with constructs that are difficult to parse directly in their native form such as enumerated lists, possessive, and punctuation marks. The pre-processing step identifies these items and restructures the sentence as necessary.

Upon completion of preprocessing, the result is forwarded to the syntactic processing module in order to perform the surface structure analysis. The surface structure analysis forms a parse tree for the input sentence. It depicts the syntactic role of each word and the organization of the sentence as constituents consisting of different types of acceptable phrases. An Augmented Transition Network (ATN) is used to realize a simple grammar with register and hold mechanisms. The ATN grammar used in SNAP/NLP is a variation of the ATN grammar made popular by Woods [25] with extensions described by Allen [1] and Winograd [24]. The current SNAP/NLP design emphasizes simplicity and extensibility over generality and broad grammar coverage. This is in agreement with the interests of the development of a prototype implementation on the SNAP machine.

A top-down, depth-first parsing strategy is employed to facilitate straightforward and efficient execution. The transition list is maintained as a LIFO stack and updated with each word accepted. Backtracking is used as necessary to obtain the correct parse. At each state in the execution of the parse, a data structure containing the current parse location, current state, and a list of return nodes is maintained so that if the attempted parse is unsuccessful, it can be resumed at an appropriate point. Furthermore, a syntactically correct parse may be created that does not have an acceptable semantic interpretation. In this case, the semantic interpreter will issue a retry message to the parser to generate the next most likely parse tree. To illustrate surface syntactic analysis, consider the operation of the parser on a portion of the sample input text. The input sentence, **`The controller sends instructions to the array.`**, would be parsed by first stripping out the terminating period and then initiating an associative lookup of the first word in the sentence.

The parsing process is accomplished by realizing a sequence of acceptable state transitions in the network representation of the ATN grammar. This representation includes a set of four transition networks that process inputs

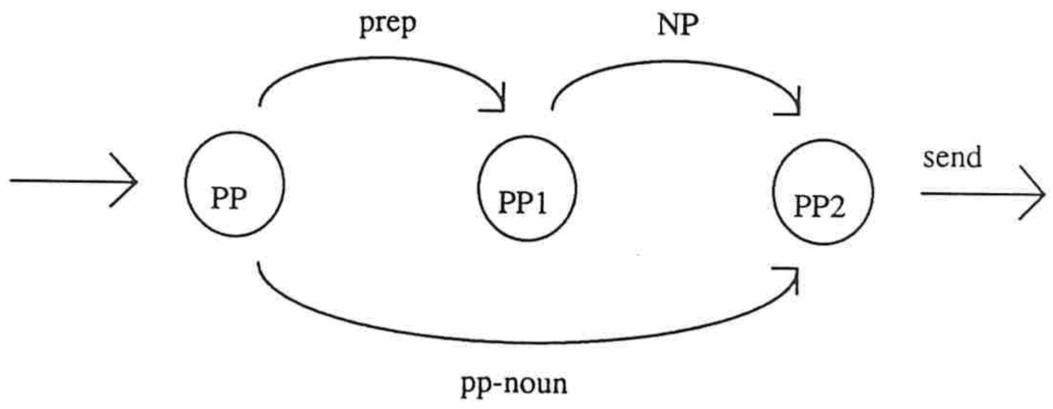


Figure 5: A Prepositional Phrase Network

as finite state accepting automaton. There is one network for each major type of constituent (see Figure 5). Prepositional phrases, noun phrases, verb phrases and their overall arrangement into acceptable sentences are thus described by each network. The networks consist of nodes which represent various internal states of the machine and arcs which define allowable transitions between states based on the current input word. There are several types of arcs used, including lexical categories to allow processing of a single word type and calls to other networks to allow more complex structures and recursion. Each arc is given a precedence number such that the most likely parse can be obtained from a set of multiple valid transitions on a per constituent and per sentence basis.

For example, consider the graph representation of the prepositional phrase network. It is the simplest of four networks used in SNAP/NLP. It can recognize prepositional phrases consisting of prepositions, noun phrases, and prepositional phrase nouns (e.g. yesterday, now, here, home). The initial state is indicated by the short incoming arrow. The graph accepts the input sequence as a phrase of this type only if it is possible to reach a send arc during the course of parsing.

For example, the surface structure for the input sentence is represented by the parse tree shown in Figure 6.

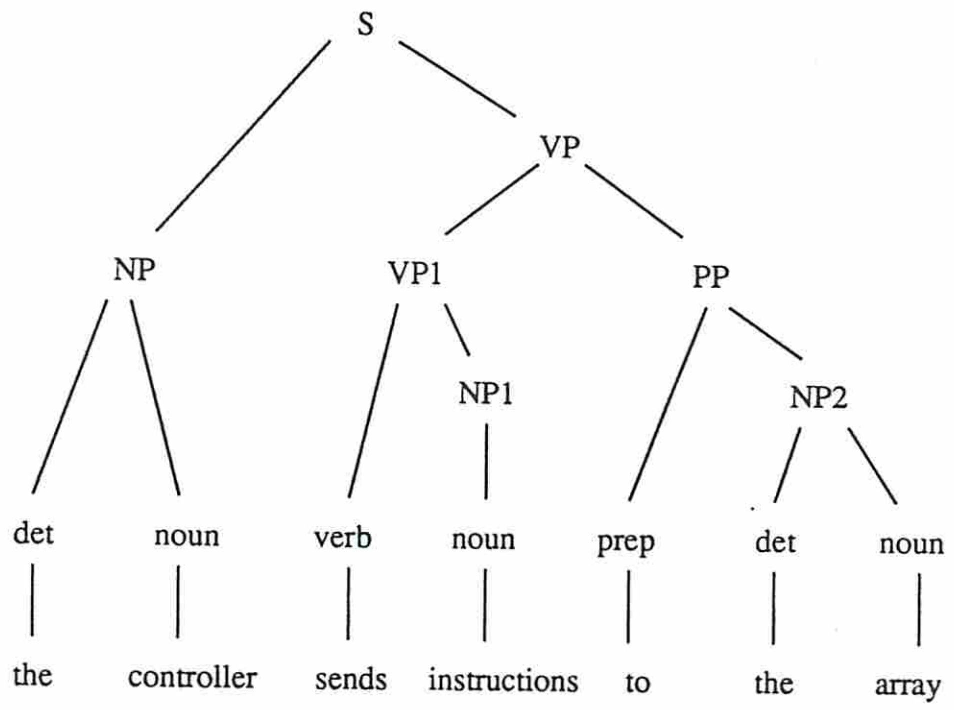


Figure 6: A Sample Parse Tree

Deep Structure Analysis

The deep structure is derived as an augmentation to the parsing process in order to expose the functional role of each sentence element. It supplements the results of the surface structure analysis with an understanding of the dependencies between constituents. It is obtained by testing and setting a set of data structures called *feature registers.* These feature registers allow the delineation of deep structure by presenting a set of tests and actions for each word or constituent accepted by the transition network.

For each network there is an associated set of rules which governs assignment of the feature registers. A group of assignment rules which govern the occurrence of particular lexical categories in specific locations are applied to determine subject-verb agreement, auxiliary-verb agreement, verb complements. Registers are set to reflect any deviation of the deep structure from a set of default values based on the particular sequence of arcs traversed during the parse. In each rule, the asterisk (*) is used to represents the current word being parsed. For the PP network, PREP denotes a preposition, HEAD denotes the head word, and POBJ denotes the prepositional object. Thus the rules for the PP network are:

Arc Test Action

PP/1 — PREP¡-*
PP/2 — HEAD¡-*
PP1/1 — POBJ¡-*

As will be shown, this identification of functional role is an essential prerequisite to semantic analysis. For instance, the deep structure analysis will identify a noun phrase (NP) as the subject of a simple sentence if it occurs before the verb while, a noun phrase at the end the sentence will be identified as a direct object (DOBJ). Especially significant is the determination of the principal noun or *head* of the noun phrase because it identifies the focus of the phrase. A number of techniques can be employed during deep structure analysis to maintain register integrity such as variations on logical unification that preserve the capability for intra-parse backtracking. The output of the syntactic processor for the sample sentence above is the following deep

```
( S      SUBJ     ( NP     DET       ( The )
                           HEAD      ( Controller )
                           NUM       ( S )

         VP       ( VP     VERB      ( sends )
                           NUM       ( S )
                           TENSE     ( present )
                           PERSON    ( 3 )
                           ROOT      (send )
                           NOUN      ( instructions )
                           NUM       ( P )

         PP       (        PREP      ( to )
                           DET       ( the )
                           NOUN      ( array )
                           NUM       ( S ) ) )
```

Figure 7: A Sentence Deep Structure

structure(see Figure 7).

4.2 Semantic Interpretation

In this section, the method, by which the knowledge contained in a sentence is captured into an explicit formation of concepts and conceptual relations, is desçribed. Regarding the level of interpretation, only *surface semantics* is considered. Since a sentence can be interpreted in more than one perspective depending on contexts, the explicit representation of only surface semantics is likely to be the most reasonable choice due to its *neutrality* [21]. Further inference can then be done on the basis of explicit surface semantics.

Interpretation by Graph Unification

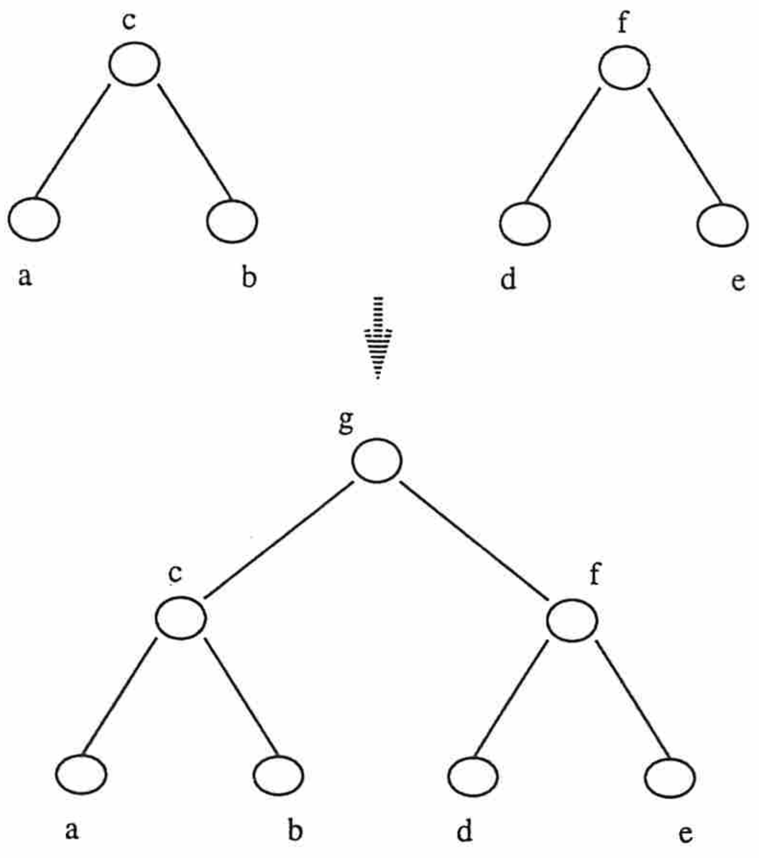


Figure 8: Graph Unification

The semantic interpretation process begins with the parse tree which is output from the syntactic parser. The parse tree shows a partial order in which the words in a sentence are combined. The partial order guides the sequence of surface semantic construction in bottom-up manner. Since the representation of word definition and the target surface semantic representation can be viewed as a type of graph, advantage is taken of graph unification in the semantic interpretation method [19] [20].

Graph unification is the construction of a new graph from two separate graphs [3]. The new graph should subsume two the old graphs in a certain set of graph characteristics, i.e., structure of nodes and links (see Figure 8). The basic graph unification steps are shown in the following:

(S: (NP: the control) (VP: (VP!: send (NP! instruction)) (PP: to (NP the array))))

Figure 9: A List for a Parse Tree

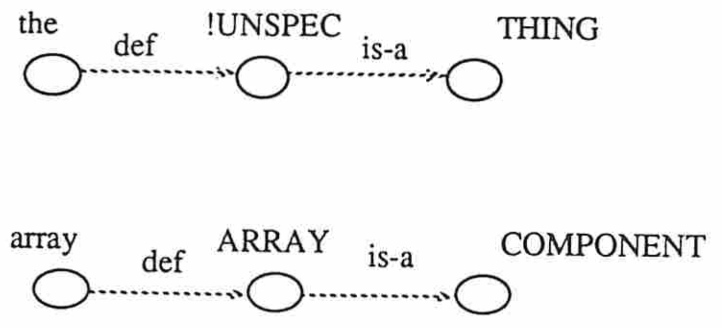
1. Copy each old graph: `G(a,b,c)` and `G(d,e,f)`.
2. Select the head node in each copied graph: node `c` and node `e`.
3. Join the two head nodes: `G(a,b,c,d,e,f,g)`.

The parse tree in Figure 6 is stored in the host computer with the format in Figure 9. The innermost lists should have two words. Single word phrase is stripped the parenthesis. Whenever a join operation is performed, the related word or phrase pair is replaced by the corresponding phrase or sentence label. If there are more than two words in a phrase of a parse tree, this phrase is represented by a nested list. For example, a noun phrase `SEAVER SCIENCE LIBRARY` is represented by `(NP1 SEAVER (NP1.1 SCIENCE LIBRARY))`. The graph unification method is modified for semantic interpretation. Figure 10 shows an example *semantic graph unification* performing the following steps.

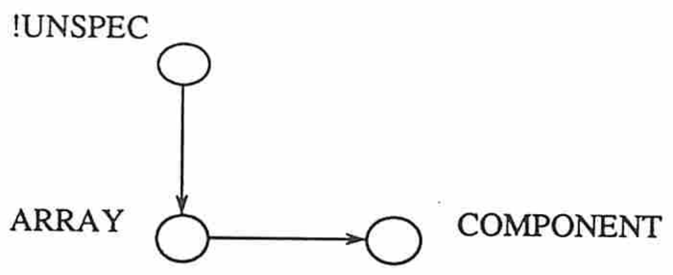
1. Associate each word in a sentence with the word definition graph: `the` and `controller`.
2. Test the conformity between the head nodes in each graph. Selection of head node is not strict. If the test fails for the current head node selection, then try another combination.
3. If the test succeeds, then join the two graph with respect to the conformed nodes: `the controller`.

Here, the conformity is decided on whether the concepts associated with the testing nodes have the *same* concept type, or on whether the two concepts have a *subsumption* relation. This test is done with the concept type hierarchy in Figure 4. In either case, the join of two graphs are permissible. Otherwise, the two graphs cannot be joined so that the parse tree is rejected. This subsumer is used for the actual conformity test.

The join operation is used to combine two graphs. Unlike any typical graph join operation, however, one of two nodes tested for conformity is



(a) Copy



(b) Join

Figure 10: Semantic Graph Unification

removed by the following rule: a node in one graph should replace the subsumer node in other graph. All links involved with the subsumer node are redirected to the subsumer node. The joined graph is a new graph bearing the semantics of the two old graphs. Since the parse tree has a recursive nature, the same procedures are used for the interpretation of the entire sentence.

Algorithm with SNAP

The semantic graph unification in Figure 10 can be done using SNAP's *marker propagation* technique. The basic steps are shown in the following:

1. Select a innermost list for semantic unification.
2. Each word in a selected pair is tagged to access the lexicon.
3. Mark the concept for each tagged word.
4. Find the superconcept for each tagged word. This is done by propagating marker along *is-a* link.
5. Collect the superconcepts and delete the tag. If the collected superconcepts are the same type, then go to step 9.
6. Mark the superconcepts in the concept type hierarchy, respectively.
7. Propagate markers along *is-a* link.
8. Collect the nodes containing both markers. One concept containing both markers may be a subsumer of the other concept. If there is no such concept, then the joining is blocked. The current parse tree is rejected.
9. The word concept in the subsumee graph has *is-a* link from the subsumer link if the concept in the subsumer graph is !UNSPEC. Here, !UNSPEC means the existence of the unspecified concept. If two comparand concepts are the same type, then a concept with *is-a* link to this concept will receive the relation pointer from other graph. The relation pointer is not actual pointer, but it is a virtual pointer putting markers in the order of source and sink.

After the final join operation, the semantically unified graph is retrieved back to the host computer. The retrieval operation is performed while maintaining the denoted semantics. Figure 11 shows an retrieved semantic graph. This figure is read as `The controller send instruction to the array.` There are two definite anaphora in the Figure, which will be handled in the discourse analyzer.

Resolution of Ambiguities

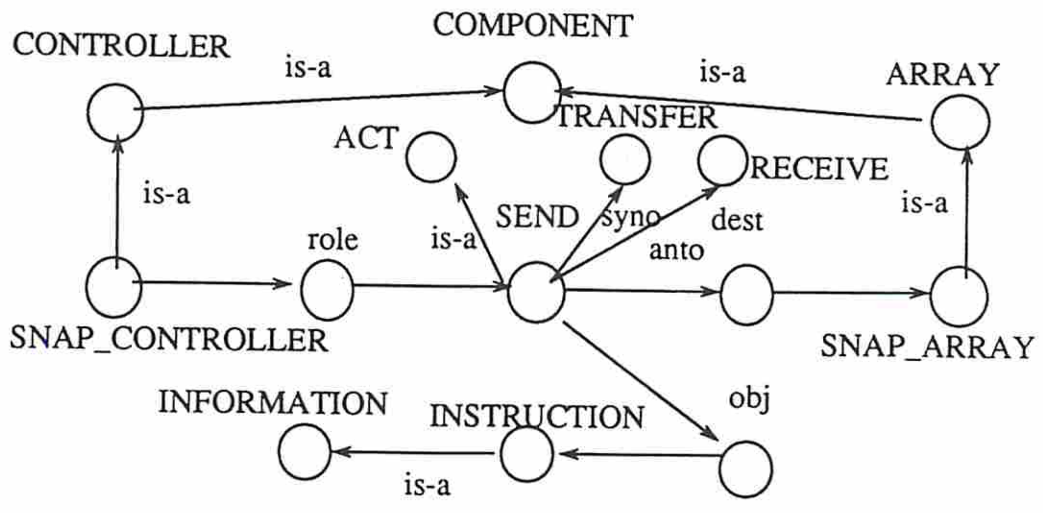


Figure 11: Semantic graph for a sentence

The semantic interpretation by graph unification can handle the lexical ambiguity and some prepositional phrase ambiguity by the conformity test. In this paper, only simple sentences in syntactic or semantic sense are processed. First, the syntactic parser limits the versatility of input sentences due to the capability of the parser. Any type of ill-formed sentences are beyond the scope of the current paper. Second, the semantic interpreter and the discourse analyzer limits the scope of language features.

Since the semantic graph unification is basically based on the case grammar and the selectional restriction, this method can resolve lexical ambiguity. Usually the word category ambiguity can be easily resolved because the wrong choice of a word's category leads to blocking in the unification. The word sense ambiguity is resolved by the main verb graph's selectional restriction nodes. The structural ambiguity mainly comes from the prepositional phrases. Some of the cases can be resolved, but not all. The discourse analyzer helps resolve this type of ambiguities.

4.3 Discourse Analysis

Discourse analysis may encompass a broad range of operation. In this paper, the discourse analyzer performs proposition and theme building, anaphoric reference and establishment of both causality and sequence.

Proposition and Theme

After semantic interpretation, the concepts in the current surface semantic graph are linked to the new proposition node. Each proposition node has different identification number showing the order of assertion. An associated register indicates the current theme identification number. The proposition node is linked to the theme node with the value contained in this register.

Anaphoric Reference

Anaphoric reference is defined as the reference of a noun phrase to an object mentioned earlier in the proposition or in a theme [1] [17]. Two types of anaphoric reference are handled. One is a reference by a *definite noun phrase.* The other is a reference by a *pronoun.* In Figure 11, **`the controller`** and **`the array`** are definite noun phrases. !UNSPEC in each noun phrase graph represents an unknown referent for each noun phrase.

It is observed that a referent for a pronoun reference is found in the current theme including the current proposition. Also, most references by a definite noun phrase can be resolved in the current theme. Besides, the referent is constrained for its concept type semantic interpretation. This observation helps develop an efficient algorithm:

1. Find the referent in the current proposition.

2. If it fails, then find the referent in the current theme.

3. If it fails again, try the previous themes.

We expect that most referents are found in the step 1 or step 2. Since the resolving reference problem needs the access of discourse knowledge base, we develop the following marker propagation algorithm for step 2 and step 3. The step 3 uses the same algorithm of step 2 with other themes.

1. Mark all concepts in the current theme. (Marker 1)

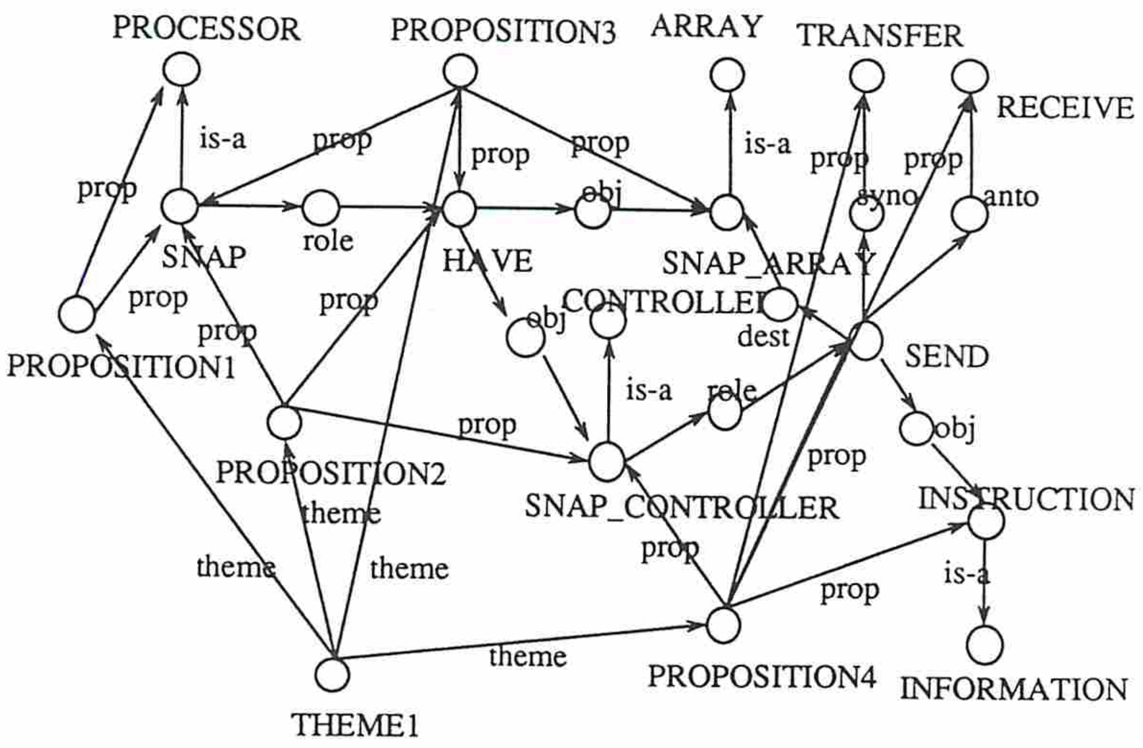


Figure 12: Updated Discourse Network

2. Mark the concept type. This concept type is the proper concept type for the referent or its subsumee. Propagate the marker to subsumers. (Marker 2)

3. Collect the most specific subsumee with marker 1 and marker 2.

If there are more than one candidates, a certain decision mechanism is used. Currently, we assume to have only one referent. After solving the anaphora reference problems, the expected discourse network looks like Figure 12.

4.4 Classification of concepts

Classification is a process of constructing a concept taxonomy for the knowledge representation system which represents terminological knowledge in a hierarchical manner. Here its basic functions and its relation to other modules are described and the implementation of the efficient classifier using the

SNAP architecture is shown.

Terminological Reasoning

In the knowledge representation system described in the previous section, the generalization relation between concept and superconcept forms an inheritance hierarchy or taxonomy called a *Subsumption relation* The property of inheritance from the superconcept to the subconcept along this subsumption relation is essential in inferencing [11].
One important characteristic of these systems is that there exist not only explicit or given relations but also implicit relations between concepts.

Therefore, in order to construct and maintain correct taxonomy, some type of reasoning needs to exist with which the hidden relations can be found.[14]. This is called *Terminological reasoning* and there are two types of basic functions for such reasoning.

- Decision for subsumption
  Subsumption is defined as follows. Concept A subsumes concept B only if the set it denotes necessarily includes the set denoted by B [5]. Whenever a new concept develops, it must be compared with other concepts in the taxonomy in order to determine the subsumption relation.

- Classification
  After the subsumption relations are tested, the Most Specific Subsumer (MSS) must be found from among the subsumers in order to put the new concept into the right position in the existing taxonomy.

In general, classification means both deciding the subsumption relations between concepts and finding the MSS.

Classifier as a Software Module

A software system which performs the above functions of subsumption and classification is implemented in the host machine. This machine makes use of SNAP instructions in order to perform those functions.
It is located at the end of the SNAP/NLP system, and directly handles the knowledge base in the SNAP array. The input to the classifier is the concept

description from the semantic interpreter. Whenever a new concept arises, the classifier finds the MSS of that concept and then puts that concept in the correct place in the knowledge base.
The basic steps of classification are as follows.

1. By comparing the properties of new concept with those of the other concepts in taxonomy, find out all the superconcepts (subsumers).

2. Search for MSS among those subsumers.

3. Make subsumption link between the MSS and new concepts.

The above steps can be performed very efficiently in SNAP through use of parallel marker propagation. The classifier will also work on any change in the knowledge base to maintain the correct hierarchy.

Efficient Classification using SNAP

SNAP architecture supports simultaneous marker propagation, which is extremely useful for many functions used in Artificial Intelligence [13]. The classifier algorithm takes advantage of SNAP instructions which are executed directly in hardware. Subsumers (superconcepts) of a new concept can be found, not by sequential comparison but by parallel searching and MSS in parallel can also be found by using the marker propagation. The simplified procedure for the classification algorithm on SNAP can be described by the following steps.

1. Mark all subsumers of new concepts with a marker and propagate that marker through the subsumption link.

2. Mark all subsumers of roles with a marker and propagate that marker through the role-relation link.

3. Mark all concepts which have at least one marker.

4. Unmark all concepts which have roles different from the new concept. After these steps, all marked concepts will be subsumers of the new concepts.

5. Find out the MSS (which can be done in constant time on SNAP).

For every concept in the taxonomy each step can be done in parallel. The detailed algorithm and time analysis will not be described here. Simulations of real-life examples are in progress.

5 Question Answering with SNAP

To extract information from the processed text, a query is issued. This query is processed using procedures analogous to those in text understanding. A query graph is constructed. The query is answered by matching its graph against the discourse semantic network.

There are two types of queries: a query asking *yes* or *no* (called a yes-no query) and a query starting with *when, where, how, what, why and which* (called wh-query). Among wh-query, a query asking a reason or explanation is not expected. It is assumed in this paper that only simple concept retrieval queries are to be issued. Also, it is not sought to provide natural language generation capability.

5.1 Query Parsing

Query parsing creates a formal syntactic structure of the natural language query. It identifies the type of query as yes-no or wh-word and determines the category of item sought by the query.

Query Entry

Queries from the user are entered into the Sun host computer and are processed interactively. Upon receipt of a query, it must first be parsed using a grammar that can accept additional constituent sequences (such as the placement of auxiliaries as the initial word). Further rules and techniques are added to deal with the aspects of *movement* introduced in questions. These methods include hold mechanisms which provide the ability to use the constituents themselves as variables to fill gaps created by the missing information in the sentence.

Yes-No Query

Yes-no questions are handled using the *subject-aux inversion* technique. This allows conversion of the query to an equivalent declarative sentence that can then be confirmed or denied during query processing. Thus, this inversion provides an efficient method for converting the input query into a form that can be matched directly against the knowledge base. It is accomplished by setting an additional register called MOOD which indicates whether a given sentence is declarative or interrogative. Similarities of other facets of the query and the sentence, such as subject-verb agreement and transitivity, are typically maintained [1]. The query parser takes advantage of this similarity between declarative sentences and interrogative questions. The primary difference between them is the insertion of an auxiliary as the first word and inversion of the sentence subject. Consider the following declarative sentence:

```
The controller sends instructions to the SNAP array.
```

A corresponding yes-no query might be:

```
Does the controller send instructions to the SNAP array?
```

This query can be parsed by supplementing the syntactic analysis procedure for declarative sentences with identification of the auxiliary `Does` and the inverted subject `the controller`. It then becomes possible to match the newly formed declarative hypothesis against the existing knowledge base to generate the 'Yes' or 'No' response. This is accomplished by forwarding the deep structure to the query interpretation module.

Wh-Query

The technique for handling queries starting with *What, Who, When, Which*, etc. is similar to that for yes-no queries. It also takes advantage of the fact that a wh-query is a form of rearrangement of sentence constituents. However, this type of query is more complicated since the item of interest is missing. The technique for handling this *unbounded movement* in SNAP/NLP is illustrated by the following example.

```
( S     WH-QUERY     ( NP1     NOUN       What )
        MOOD           WH-QUERY
        AUX                              ( Does )
        SUBJ         ( NP      DET        ( the )
                               NOUN       ( SNAP )
                               NUM        ( S )
                               HEAD       ( array )
                               NUM        ( S )
        VP           ( VP      VERB       ( receive )
                               NUM        ( S )
                               TENSE      ( present )
                               PERSON     ( 1, 2 )
                               ROOT       ( receive ) )
        OBJ                               ( ? what ) )
```

Figure 13: Query Deep Structure

`What does the SNAP array receive?`

This query is handled as an instance of a typical pattern in Wh-queries [1]:

1. *Identification* : Wh-word as initial word.
2. *Arrangement* : Subject-aux inversion.
3. *Additions* : A *do* auxiliary inserted.

The parser thus derives the deep structure for the query as shown in Figure 13. Here the missing quantity is identified as a *?What* variable which can then be instantiated by processing the deep structure against the existing semantic network already in the knowledge base.

5.2 Query Interpretation

Yes-no and WH-word queries are interpreted in a similar way. The resulting graph has a form of declarative sentence interpretation. Since the query parsing process detects the functional role of WH-word in WH-query, the proper referent concept type such as COMPONENT, INFORMATION, can be decided on the basis of the WH-word form and its functional role.

Next, the WH-word is replaced by a variable with its type restriction. The rest of the interpretation is the same as the yes/no query. Since the query interpretation procedure is similar, the same semantic graph unification algorithm can be used for the query interpretation. Figure 14 (a) shows the query graph for the above example sentence. In this figure, ?WHAT represents a variable to be filled with the retrieved concept from the discourse network.

5.3 Query Processing

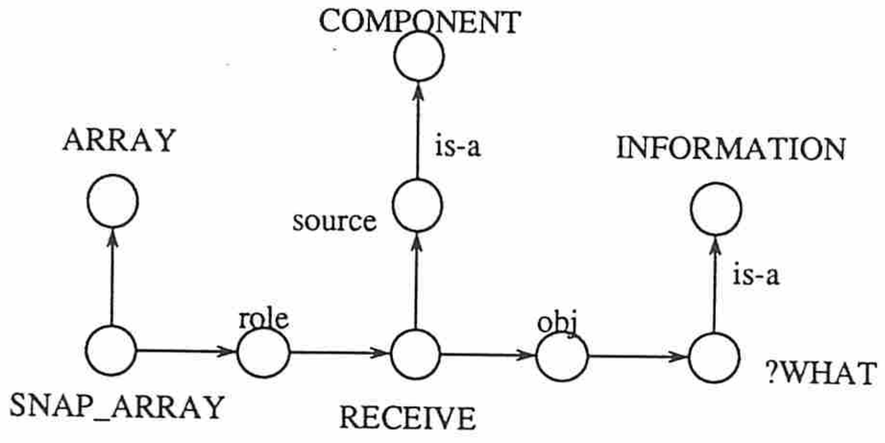
Depending on the type of a query, the appropriate query processing method is selected. For yes/no queries, the query processor performs *pattern matching* against the discourse knowledge semantic network with the query graph. For this purpose, the following method has been developed:

1. Due to the *head analysis* in the query parsing, a head concept to start the matching is available.

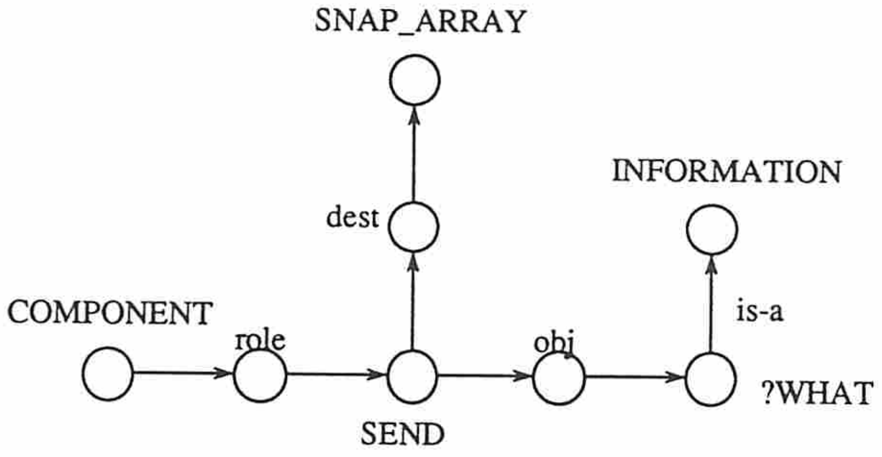
2. From the head concept, test the relations and the concepts of the query graph against the discourse knowledge semantic network.

3. When a certain relational concept such as consist and connect is considered as transitive, the two patterns in Figure 15 are considered to be virtually identical.

This method decides whether there is a pattern equivalent to the query graph in the semantic network. For WH-query, a head concept is selected except the concept variable from the query graph. Then, the query graph without the concept variable node and its concept type node is matched against the semantic network. If the match succeeds, then the corresponding concept satisfying the type restriction for the concept variable is found and



(a) Original Query Graph



(b) Modified Query Graph

Figure 14: Query Graphs

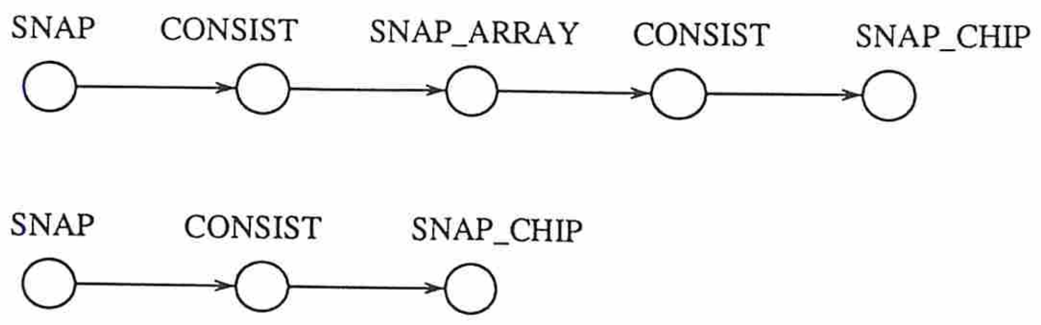


Figure 15: Semantically Equivalent Graphs

filled into the variable. A corresponding algorithm for pattern matching has been designed with SNAP instructions [13].

For the example query graph in Figure 14 (a), there is no successful match in the discourse network. However, the fact that RECEIVE is an antonym of SEND can be detected during the query processing. The *antonym* relation suggests to modify the current query graph into the semantically equivalent query graph in Figure 14 (b). The modified query graph is read as the following sentence:

```
What does a component send to SNAP array?
```

By using the pattern matching algorithm, the proper concept for the variable is found (INSTRUCTION).

6 Conclusion

In this paper, an approach to the natural language understanding and related knowledge processing using the SNAP array has been described. Each of the software modules mentioned in the previous sections are being now developed on the base of the prototypes we designed and tested. In conjunction with this, it is attempted to perform the simulation of some functions which will be done on SNAP array, by using the SNAP simulator.

The overall structure of our system is generally applicable to natural language processing. The research is focused on the development of very efficient and fully automated knowledge processing systems by using the special purpose array processor, called SNAP, which is designed for AI applications, especially for knowledge processing and has the unique feature of marker propagation.

While designing the systems and developing each algorithm, it was found that there are many possibilities of performance increase in natural language processing which arises from the intrinsic nature of parallelism of SNAP. It was attempted to realize those possibilities.

At this time, SNAP/NLP has the features of completely automated knowledge acquisition and makes full use of the capabilities of SNAP to increase performance. There are, however, more functions which remain to be implemented on this machine, parallel bottom up parsing being one such capability. Knowledge processing on the system will be another topic of future investigations.

References

[1] James Allen. *Natural Language Understanding.* The Benjamin Cummings Publishing Company, Inc., Menlo Park, California, 1987.

[2] Yigal Arens. *CLUSTER: An Approach to Contextual Language Understanding.* PhD thesis, University of California, Berkeley, 1986.

[3] Gosse Bouma, Esther Konig, and Hans Uszkoreit. A flexible graph-unification formalism and its application to natural-language processing. In *IBM Journal of Research and Development*, pages 170–184, March 1988.

[4] Ronald J. Brachman, Richard E. Fikes, and Levesque Hector J. Krypton: A Functional Approach to Knowledge Representation. In *Computer*, pages 67–73. IEEE, October 1983.

[5] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. In *Cognitive Science 9*, pages 171–216, 1985.

[6] Schank R. C. *Conceptual Information Processing.* North-Holland/American Elsevier, 1975.

[7] Gerald DeJong. Prediction and Substantiation: A New Approach to Natural Language Processing. In *Cognitive Science 3*, pages 251–273, 1979.

[8] Michael G. Dyer. Knowledge Interactions and Integrated Parsing for Narrative Comprehension. In David L. Waltz, editor, *Semantic Structures: advances in natural language processing*, pages 1–56. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[9] C. J. Fillmore. The case for case. In E. Bach and R. Harms, editors, *Universals in Linguistic Theory*, pages 1–90. Holt, Rinehart, and Winston, New York, 1968.

[10] John J. Jr. Granacki, Alice C. Parker, and Yigal Arens. Understanding System Specifications Written in Natural Language. In *Proceedings of the tenth international joint conference on artificial intelligence Vol.2*, pages 688–691, 1987.

[11] Thomas A. Lipkis and James G. Schmolze. Classification in the KL-ONE knowledge representation system. In *Proceedings of the eighth international joint conference on artificial intelligence Vol.1*, pages 330–332, 1983.

[12] C. S. Mellish. *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood Limited, 1985.

[13] Dan Moldovan, Wing Lee, and Lin Changhwa. SNAP: A Marker-Propagation Architecture for Knowledge Processing. Technical Report CENG89-10, University of Southern California - Department of Electrical Engineering- Systems, 1989.

[14] Bermhard Nebel. Computational Complexity of Terminological Reasoning in BACK. In *Artificial Intelligence 34:3*, 1988.

[15] M. R. Quillian. Semantic Memory. In M. Minsky, editor, *Semantic Information Processing.* MIT Press, MA, 1968.

[16] Roger C. Schank and Riesbeck Christopher K. *Inside Computer Understanding.* Lawrence Erlbaum Associates, 1981.

[17] C. Sidner. Focusing in the Comprehension of Definite Anarphora. In M. Brady and Berwick R. C., editors, *Computational Models of Discourse*, pages 267–330. MIT Press, MA, 1983.

[18] John F Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley Publishing Company, 1984.

[19] John F. Sowa. Using a lexicon of canonical graphs in a semantic interpreter. In Martha Walton Evens, editor, *Relational models of the lexicon*, pages 113–137. Cambridge University Press, 1988.

[20] John F. Sowa and Eileen C. Way. Implementing a semantic interpreter using conceptual graphs. In *IBM Journal of Research and Development Vol.30 No.1*, pages 57–69, January 1986.

[21] Paola Velardi, Maria Teresa Pazienza, and Mario DeGiovanetti. Conceptual graphs for the analysis and generation of sciences. In *IBM Journal of Research and Development Vol.32 No.2*, pages 251–267, March 1988.

[22] Ralph M. Weischedel. Knowledge Representation and Natural Language Processing. In *Proceedings of the IEEE Vol.74 No.7*, pages 905–920, July 1986.

[23] Yorick Wilks. An Intelligent Analyzer and Understanding of English. In *Communications of the ACM 18:5*, pages 264–274, 1975. reprinted in *Readings in Natural Language Processing.*

[24] T. Winograd. *Language as a Cognitive Process.* Addison-Wesley Publishing Company, 1983.

[25] W. A. Woods. Transition Network Grammars for Natural Language Analysis. In *Communications of the ACM Vol.13*, pages 591–606, 1970.