

INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.
2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University
Microfilms
International
300 N. Zeeb Road
Ann Arbor, MI 48106

McDonald, David Blair

UNDERSTANDING NOUN COMPOUNDS

Carnegie-Mellon University

PH.D. 1982

**University
Microfilms
International** 300 N. Zeeb Road, Ann Arbor, MI 48106

Copyright 1982

by

McDonald, David Blair

All Rights Reserved

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ☒.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages ☒
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Other _____

University
Microfilms
International

Understanding Noun Compounds

David B. McDonald

January 1982

**Department of Computer Science,
Carnegie-Mellon University,
Pittsburgh, Pennsylvania, 15213.**

Submitted to Carnegie-Mellon University in
partial fulfillment of the requirements for the
degree of Doctor of Philosophy.

Copyright © 1982 David B. McDonald

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551, and in part by the National Research Council of Canada.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Carnegie-Mellon University

MELLON INSTITUTE OF SCIENCE

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy

TITLE UNDERSTANDING NOUN COMPOUNDS

PRESENTED BY David Blair McDonald

ACCEPTED BY THE DEPARTMENT OF Computer Science

Scott E. Fahlman 10/15/81
MAJOR PROFESSOR DATE

Jeff Abernethy 1/18/82
DEPARTMENT HEAD DATE

APPROVED BY

Guy Berry 28 Jan 82
DEAN DATE

Abstract

Understanding noun compounds is one aspect of natural language understanding where syntactic information is not very helpful. Noun compounds consist of a sequence of two or more nouns that modify one another. The problem is to determine the meaning of a compound as a whole. This includes finding how the nouns in a compound should be grouped and what the underlying meanings of the groups are. The meaning for a compound is often determined by relationships between two or sometimes three nouns. Examples of noun compounds include: "golf club", "boy king", "car assembly plant", "glass wine glass", "woman coal miner", and "water meter cover adjustment screw".

This thesis describes a computer program that attempts to understand noun compounds. The program includes a data base of real-world knowledge containing information about the concepts represented by nouns. The information in this data base is used to find possible relationships between the nouns in a compound as it is processed from left to right. Possible interpretations for a compound are formed based on these relationships. This approach is different from most previous work dealing with noun compounds, since it does not rely on a fixed set of relationships. When the end of the compound is reached, there may be zero, one, or several possible interpretations. If no interpretation is found, nothing further is done. If one interpretation is found, information is added to the data base to represent this meaning for the compound. When there is more than one possible interpretation for a compound, some heuristics are used in an attempt to determine the best interpretation. Several heuristics have been developed that attempt to select the best interpretation from the set of possible interpretations produced when the system first processes the compound left to right.

Understanding Noun Compounds

Thesis Summary

David B. McDonald

September 1981

**Department of Computer Science,
Carnegie-Mellon University,
Pittsburgh, Pennsylvania, 15213.**

**Summary of a thesis submitted to Carnegie-Mellon
University in partial fulfillment of the requirements for the
degree of Doctor of Philosophy.**

Copyright © 1981 David B. McDonald

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551, and in part by the National Research Council of Canada.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Table of Contents

1. Introduction	1
1.1. What is a Noun Compound?	1
1.2. Examples of Problems	2
1.3. Previous Work	4
2. The Knowledge Representation System	5
3. Finding Possible Interpretations	6
4. Deciding on Which Interpretation is Best	7
4.1. Context	8
4.2. Cognates	9
4.3. Embedded Cognates	9
4.4. Instances of an Interpretation	9
4.5. Embedded Instances of an Interpretation	10
4.6. Verifying Slots	10
4.7. Similar Compounds	10
4.8. Number of Times an Interpretation has been Encountered	11
4.9. Combining the Results of the Heuristics	11
5. Interfacing the System to a Parser	12
6. Experiment to Determine How Well the System Performs on Real-World Examples	12
7. Results	13
8. Conclusions	14

Preface

A brief summary of my thesis is given in this document. The thesis describes a computer program that can understand noun compounds. Chapter one is an introduction to the problem of understanding noun compounds including previous work in the area and a brief description of the approach used in the thesis to understand noun compounds. Chapter two describes the NETL knowledge representation system which is used to store information about concepts, other real-world knowledge, and the meanings of compounds created by the system. Chapter three and four describe the two distinct phases that the system currently uses. Chapter three describes the first phase which finds all the possible interpretations for a compound. Chapter four describes the second phase which consists of several heuristics used to determine the best meaning for the compound if more than one possible interpretation is found during the first phase. Chapter five describes how the components of the system could be interfaced to a parser along with many of the issues that this would involve. Chapter six describes an experiment where the algorithms described in chapters three and four were hand simulated on over six hundred noun compounds occurring in text from real-world examples. Chapter seven presents the results of the research from several different views. Finally, chapter eight presents several conclusions from this work along with some directions for future work. The following sections give a brief summary of the important aspects of the corresponding chapters.

1. Introduction

A computer program has been written that attempts to find the meaning for a noun compound using real-world knowledge. This program was designed with the possibility of eventually incorporating it into a more complete natural language understanding system. It expects information that only such a system can provide to disambiguate some of the noun compounds encountered. The following subsection introduces the problem of determining the meaning of a noun compound.

1.1. What is a Noun Compound?

A noun compound is a sequence of two or more nouns (e.g., "computer science", "car assembly plant", "glass wine glass", "water meter cover adjustment screw", etc.) in which the *head noun* or last noun in the sequence is usually modified by the other nouns in some manner. Noun compounds are used frequently by people to describe things. To successfully communicate, a person must have a model of the subject being discussed similar to that of the person with whom he is communicating. When a person uses a noun compound, he is usually using a shorthand notation to save time in communication. The reader must reconstruct the information deleted by the writer, if the reader is to correctly comprehend the phrase.

The problem addressed here is how to produce one interpretation from such a sequence of nouns. Or, in other words, what is the knowledge necessary to reconstruct the information deleted when the compound was

formed. Syntactic knowledge is not sufficient to understand most compounds. Rather, it is necessary to access semantic knowledge. The semantic knowledge needed to understand a compound depends on the compound under consideration. An interpretation for some compounds can be produced just using the meanings of the nouns, without any further knowledge. For example, the most likely meaning of the compound "dog owner" can be found using the definitions of the words "dog" and "owner". However, other compounds require more knowledge to be available to successfully produce an interpretation. For example, the compound "bee sting", has at least two meanings (i.e., a wound caused by a bee, and the part of bee that can cause such a wound), both of which are reasonable interpretations. The method to distinguish between them seems to rely on contextual information that would normally surround the compound.

1.2. Examples of Problems

There are several problems that arise when an attempt is made to understand a noun compound. Meanings for many compounds can be determined from a relationship between the nouns in the compound. However, there are some compounds whose meaning is not determined by such a relationship. Rather, their meaning is associated with the sequence of words in the compound. For example, consider the compound "noun phrase". Most people would recognize this compound as a phrase that contains words that are nouns as well as other parts of speech because it has a well defined meaning in linguistics. It does not mean a phrase that consists of just nouns which is the meaning the program would find. Most people would probably find the same meaning if they did not know the linguistic meaning for the compound. Any system to understand noun compounds must deal with ones having a meaning distinct from the meaning provided by a underlying relationship between the nouns. Such compounds must be stored in the system's data base so that if the correct sequence of nouns appears, the built-in meaning becomes a possible interpretation. This does not mean that a built-in meaning is instantly used as the best one, only that it is a very good candidate. For example, in some contexts the built-in meaning for "noun phrase" could be the wrong meaning (e.g., if the person speaking the phrase did not know the linguistic meaning). In such a case, the built-in meaning should not be used, and another meaning should be considered. Other examples of such idiomatic compounds include: "chain gang" and "crown prince".

Another problem a theory to understand noun compounds must deal with is multiple meanings of words. This is a problem that any natural language system must cope with, but is no less important here. Some words in a compound can be disambiguated from the context provided by the compound itself. For example, consider the compound "glass wine glass". The word "glass" has at least the two meanings:

1. A container from which people drink certain liquids.
2. An amorphous material (silicon dioxide) from which certain things are made (e.g., a glass as above).

In the compound "glass wine glass", the first "glass" has meaning (2), and the second meaning (1). This can be established fairly easily, since the first "glass" has no underlying relationship with "wine" that makes sense given the order of the words. The second "glass" fits best if meaning (1) is used, since wine is a liquid that people drink. Then the first "glass" fits nicely by being the material from which the second "glass" is made. In this example, the multiple meanings of the word "glass" do not cause any difficulty because the compound itself provides sufficient structure to determine the correct meaning of "glass".

In the example above, the compound contains sufficient constraints to determine which meaning of "glass" was meant. There are compounds for which this is not the case. For example, consider the compound "coal bill" which has at least two meanings (one as a bill requesting payment for coal and the other as a bill before congress or another governing body concerning coal). The compound "coal bill" does not by itself contain sufficient information to decide which of the meanings is the correct one. To make this decision, it is necessary to provide more context. The context necessary to disambiguate the compound must come from the surrounding phrases. In this case, context can be provided by a template for a verb with some of its slots filled. The system must determine from this context the only reasonable meaning for the compound "coal bill".

Another problem is that more than one relation can be used to form a meaning for a compound. This is likely to happen fairly often as any reasonable size data base of knowledge will probably have several relationships between some nouns. This means any system to understand noun compounds must select the relation producing the best meaning. For example, consider the compound "computer evaluation". Assume the concept "evaluation" has two slots: one is for the thing doing the evaluation, and the other is for the thing evaluated. There are other slots available (e.g., the time when the evaluation action took place and the location where it occurred) to use as the basis for an interpretation, but these will be ignored in this discussion for simplicity. Since in some contexts a computer can be the thing doing the evaluation (e.g., computer evaluation of an election), as well as the thing evaluated, both slots lead to a meaning for the compound "computer evaluation". Normally, a compound of the form "x evaluation" has the meaning where x is the thing evaluated (e.g. "car evaluation", "stereo evaluation", etc.). Assume the data base contains several compounds of the form "x evaluation", where x is the thing maintained, and a few (if any) where x is the thing doing the evaluation. One source of information to decide which relationship is the best one to use in forming a meaning, is to look for compounds that are similar to "computer evaluation" and use a similar structure for this compound. This is likely to lead to selecting computer as the thing evaluated, rather than the thing that is doing the evaluation.

Of course, this cannot always happen, otherwise in the context of "computer evaluation of an election", the wrong interpretation for "computer evaluation" will be chosen. It is necessary to provide context to enable

the system to produce the correct interpretation. The best way to provide this context is to specify that one of the possible relationships between "computer" and "evaluation" is already used. In this case, the relationship specifying the thing evaluated must be filled with "election". This leaves only one possible relationship between "computer" and "evaluation", i.e., the thing doing the evaluation.

So far, the ambiguities pointed out above are not caused by lack of knowledge, but by the inherent ambiguity local to noun compounds. Another problem arises because the data base may not contain all the necessary information to obtain only those relationships that can possibly exist between nouns. For example, with the compound "blood evaluation", if the data base does not contain the information that "blood" cannot be the thing that performs an "evaluation", two possible relationships will be found (i.e., where "blood" is the object being evaluated and the spurious one). It is unreasonable to expect the methods described above to obtain the correct interpretation, since they deal with ambiguity inherent in natural language, not ambiguity created by the data base. One method to cope with this problem satisfactorily is to check the relationships to make sure they are the ones expected. In the "blood evaluation" example, this means checking if "blood" is expected to be the thing doing the evaluation, or the object evaluated. In this case, it is expected to be the object evaluated, and thus this interpretation is more likely to be correct. This check can also help disambiguate a word having multiple meanings.

Any system to produce the one best interpretation for a compound must address the problems presented above. One of the goals of the thesis is to determine the knowledge necessary to solve these problems in such a way that one interpretation can be produced.

1.3. Previous Work

At least two groups of researchers have worked in the area of noun compounds. Linguists became interested in noun compounds first. Artificial Intelligence researchers, especially those interested in natural language understanding, have also become interested in them. There is a fair amount of information in the literature about how linguists have approached the problem. However, for computational models, there are only a handful of efforts directly related to noun compounds.

Most of the work by linguists has taken the approach of trying to create a categorization scheme for noun compounds. They have tried to find a fixed list of relationships that can be used as the basis for forming the meaning of a noun compound. The major problem with this approach, as far as the current research is concerned, is that it does not address the problem of how to understand noun compounds. Downing's article takes a more general approach to the problem in that she claims that no fixed set of relationships between nouns will be sufficient to understand all compounds. However, she does not propose any concrete method

to understand them.

Artificial Intelligence researchers, especially those interested in natural language understanding, have also performed work in the area of noun compounds. Some of the researchers have followed the most prevalent linguistic approach, i.e., they have used a fixed set of relationships between nouns to either generate compounds from phrases, or to produce interpretations for a compound. Most of these efforts have restricted themselves to compounds containing only two nouns, rather than the more general problem of compounds with an arbitrary number of nouns.

Finin's approach in his thesis does not rely on a fixed set of relationships that can exist between two nouns. Rather, he has a data base of real-world knowledge which is searched to find the possible relationships between the nouns of a compound. The current work goes beyond Finin's by bringing more knowledge to bear, such as context and other heuristics used to disambiguate the meaning of a noun compound.

2. The Knowledge Representation System

The organization of the current system is based on a data base containing the meanings of all the concepts necessary to understand noun compounds. The program uses the semantic network representation system, NETL, developed by Fahlman as the underlying knowledge representation system. NETL can store definitions of words, as well as other real world knowledge necessary to understand noun compounds. The knowledge stored within NETL is easily accessed, and searches can be performed to find the information pertinent to what the system is currently doing. NETL is designed to run on a parallel architecture so that certain types of searches can be executed very quickly. This ability to do parallel searching is used to great advantage by the system to understand noun compounds. The current implementation uses a NETL simulator that runs under MacLisp on a PDP 10 or 20, since no NETL machine currently exists.

Information in NETL is stored as a hierarchy of concepts. Many of the concepts have words associated with them, but not necessarily all. Concepts can inherit information from other concepts higher in the hierarchy. They cannot inherit information from concepts lower in the hierarchy. A very important idea in NETL, and one which is heavily used in the program, is the split set. A split set is a set of concepts in NETL, each of which is distinct from the other. For example, a split set is formed between the two concepts "living things" and "non-living things". That is, a "living thing" is different than a "non-living thing", and vice-versa. This allows the system to distinguish between a "living thing" and a "non-living thing" easily.

As well as having some concepts as members of split sets, any concept in NETL may have what are called *roles* or *slots*. Roles allow specific information about a concept to be associated with the concept. For example, the NETL node representing the concept "elephant" has a slot representing an elephant's trunk.

among others. A concept and its slots can be thought of as a frame in the Minsky sense. Thus, all the information about a concept is accessible from one node in the network.

3. Finding Possible Interpretations

The basic approach taken to find the meaning of a noun compound is generate and test. The system is currently split into two phases. The first phase generates a list of possible interpretations and the second phase tries to find the best interpretation for the compound from the ones generated. This section gives a brief description of how the possible interpretations for a compound are found. The next section briefly describes how the best interpretation is selected.

Once definitions of some nouns and other information is placed into the network, a mechanism must be provided to produce an interpretation for a compound. The basic assumption of the system is an interpretation for a two noun compound can be produced by filling a slot in the head noun by the modifying noun. For example, assume the definition of "exploration" has a slot specifying the thing explored for as a mineral, among others. The compound "gold exploration" is interpreted by filling the "thing explored for" slot of "exploration" with the concept representing "gold". Thus, the basic approach to finding interpretations for a compound of the form " $N_1 N_2$ ", where N_1 and N_2 are nouns, is to find all the slots in N_2 into which N_1 can be placed. For compounds containing only two nouns, this strategy seems to work well. However, when there are compounds containing more than two nouns, there are more patterns than just the simple one above.

Some compounds with more than two nouns have the form $[\dots [[N_1 N_2] N_3] \dots N_n]$. That is, they are formed in a left to right fashion and can be processed easily left to right. The noun group immediately to the left of a noun fills a slot in the noun. Some examples of this type of compound are: coal mine supervisor, blood donor recruiter, and sugar cane plantation owner.

Some compounds with more than two nouns have the form $[N_1 [N_2 [N_3 \dots N_n]]]$. That is, they are formed in a right to left fashion. An example of such a compound is glass wine glass.

Some compounds contain nominalized verbs in the middle of the compound. A nominalized verb is a noun that is derived from a verb (e.g., the nominalized verb "evaluation" describes a "to evaluate" act). These compounds form ternary relations with two slots of the nominalized verb filled by the surrounding nouns (or sub-compounds possibly). Some examples of such compounds are: car assembly plant, food distribution centre, watch repair person, and water meter cover adjustment screw. In the last example, the meaning for "water meter cover" is created in a left to right parse as described above.

Some compounds contain a nominalized verb at the end of the compound. These compounds also form ternary relations with the two previous nouns filling slots in the nominalized verb. Some examples of such compounds are: student course evaluation, government price support, executive stock purchase, and city garbage collection.

Another class of compounds occurs sometimes when a noun is used very much like an adjective. These nouns seem to be distinguished by overriding a default assumption about the noun it modifies. The noun "boy" in the compound "boy king" is an example. "Boy" does not fit well into any slot of "king" and thus no meaning would normally be found. However, "boy" is overriding the default assumption that a "king" is an adult. That is, "boy" is behaving very much like an adjective. The nouns that have this property have been named *pseudo-adjectives*. It is necessary to perform special searches to find if a default assumption is being overruled. Note that the method to understand these types of compounds requires that the data base has these default assumptions stored in it. Two other examples of such compounds are: woman coal miner and toy gun.

The above patterns seem to cover a large proportion of noun compounds, but not all, as shown in chapter five of the thesis. The current system can be extended without too much difficulty to cover the patterns not covered above. In the original design of the system, fewer examples were examined than in the experiment described in chapter five and thus some of the patterns found there were missed since these new patterns occur fairly infrequently. There may still be other patterns that are missing. As can be seen from some of the examples, the classes of compounds described above are not disjoint. It is quite possible for a compound to be interpreted using two or more of the patterns.

With several different patterns, the possibility of multiple meanings for words, and the possibility of more than one relationship as the basis for the meaning of a compound, there is a good chance that there will be more than one possible meaning after all the possible interpretations have been found. At this point, a decision must be made as to which of the interpretations is the best one for the compound with the information available. This leads into the major part of thesis, i.e., the heuristics and knowledge necessary to make this decision.

4. Deciding on Which Interpretation is Best

The decision of which is the best interpretation for a compound is made by several heuristics. These heuristics attempt to use the kinds of knowledge people seem to use to make a similar decision. Some of the knowledge necessary to make this decision comes from information stored in the data base, i.e., information that has been stored in long term memory. Other information comes from the context in which the

compound occurred. A list of the heuristics that are used to find the best interpretation follows.

- **Heuristics based on Context:** use the context in which a compound occurs to disambiguate it. Context comes from the sentence in which the compound occurred as well as from the topic under discussion.
- **Cognate Heuristic:** selects the interpretation for a compound that already has a meaning stored in the data base. There are three ways for a cognate to occur. The first occurs when a compound has an idiomatic or "conventionalized" meaning associated with a particular sequence of nouns. Another occurs when an interpretation stored in the data base without being associated with any sequence of nouns. The last occurs when an interpretation is stored in the data base by the program to represent the meaning of a previously seen compound.
- **Embedded Cognate Heuristic:** selects the interpretation with the most embedded cognates. This heuristic is only effective for compounds containing more than two nouns.
- **Instances of an Interpretation Heuristic:** selects the interpretation that has the most instances stored in the data base.
- **Embedded Instances of an Interpretation Heuristic:** selects the interpretations with the most sub-compounds that have instances stored in the data base. This heuristic is only effective for compounds containing more than two nouns.
- **Slot Verification Heuristic:** selects the interpretation whose filler of a slot is one of the concepts that best fills the slot.
- **Similar Compounds Heuristic:** selects the interpretation with the most similar interpretations stored in the network.
- **Occurrence Heuristic:** selects the interpretation that has been seen most often during one session.

A more detailed description of the above heuristics follows.

4.1. Context

The heuristics based on context use the context in which the compound occurred to determine the best interpretation. There are two ways that context can be helpful. The first, and the way that applies to other areas of natural language, is to disambiguate a word having multiple meanings. For the heuristics based on context to be used, a system to understand noun compounds must be provided with extra information specifying the context in which the compound occurred. For example, the word "club" in the compound "golf club" has at least two meanings that are relevant to understanding the meaning of the compound (i.e., a club as used in games, and as an organization that has a specific purpose). The meaning of the compound "golf club" can be determined in many cases by embedding the compound in a sentence. For example, the two sentences below provide sufficient context to allow a person to disambiguate the compound correctly.

1. The golf club hit the ball.

2. The golf club fired its manager.

The context from sentence (1) is presented to the system by providing a structure where "hit" is a verb with its slot for the object being hit filled with a particular ball. This will leave several slots available in this instance of a "hit" action that may be filled by "golf club". However, the meaning of "golf club" as a piece of equipment for playing golf can only fit into the slot for the thing used to do the hitting. The organization meaning of "golf club" cannot fill any of the slots of hit given the syntactic constraints of the sentence. Likewise, sentence (2) can provide context to get the organization meaning of "golf club".

The other way context becomes important is when more than one relationship can lead to an interpretation for a compound. For example, the compound "computer maintenance" has two relationships that are reasonable. One is where "computer" is the thing performing the maintenance, the other is where "computer" is the thing maintained. The expected interpretation is probably where "computer" is the thing maintained. However, consider the phrase "computer maintenance of a data base". In this case the relationship for the thing maintained should be taken up with "a data base". Thus, the only relationship that can exist is for "computer" to be the thing doing the maintenance. The only way this knowledge can be used is to provide external context with a structure specifying one or more relationships fulfilled by some concept.

4.2. Cognates

Cognates are compounds that the system already knows about for some reason. There are built-in compounds (e.g., chain gang and crown prince) that have an idiomatic or "conventionalized" meaning. There are also compounds recognized by the system and stored in its memory for later use. If the system encounters a compound having more than one interpretation, and only one of these interpretations has been seen before, then it is the most likely interpretation for the compound if no other available information is contradictory.

4.3. Embedded Cognates

Another time cognates are useful is when there is a compound with more than two nouns. Some of the sub-compounds within an interpretation for the compound can have cognates. If there is one compound with several internal cognates, it is likely to be a better interpretation than one having none or only a few. There is likely to be zero or one in most compounds, unless the compound is very long.

4.4. Instances of an Interpretation

Rather than having a concept that represents the compound, there may be an instance of the compound stored in memory. For example, assume the compound "coal truck" has two possible relationships that can be used to form a meaning. One is the truck transports coal, and the other is the truck is made of coal. Also, assume there is an instance (or maybe several) in memory where a truck was seen transporting coal. It is more

likely the interpretation where the truck transports coal is the correct one, rather than the one where the truck is made of coal. This heuristic depends on what knowledge is stored in memory.

4.5. Embedded Instances of an Interpretation

The "embedded instances of an interpretation" heuristic selects the interpretation whose sub-compounds have the most instances. This heuristic is a generalization of the "instances of an interpretation" heuristic and is based on the same premise. This heuristic is only effective when a compound contains more than two nouns.

4.6. Verifying Slots

The method used to find possible interpretations for a compound allows slots to be found which may not be the best. It does not allow slots to be found for which an interpretation is clearly wrong, unless there is a major error in the knowledge used. Verifying the slot underlying a compound means the system makes sure the concept that is to fill a slot is in the set of concepts expected to fill the slot. An interpretation in which this is true should be better than one where the concept is not expected to fill the slot.

4.7. Similar Compounds

The similar compounds heuristic is based on the assumption that if there are interpretations similar to a particular interpretation under consideration, such an interpretation is better than one that does not have this property. An interpretation of a compound is similar to another if the concepts representing the meaning of the head nouns of the two compounds are similar, the concepts filling the slots are similar, and the slots are similar. Two concepts are similar if they exist in the same type hierarchy in NETL. Two slots are similar if they have a common parent which will often be the defining occurrence of the slot.

For example, assume the compound "computer maintenance" has been interpreted using the two possible relationships between "computer" and "maintenance" (i.e., where "computer" does the maintenance, and where it is the thing maintained). There are likely to be several other compounds where things that are similar (in a fairly broad sense) to a computer are the things maintained. For example, assume such compounds as "car maintenance", "aircraft maintenance", "engine maintenance", etc. have been seen by the system and have been recognized as compounds where the first noun is the thing maintained. If no other information is available to the system (such as context), it is better to choose the meaning where computer is the thing maintained.

4.8. Number of Times an Interpretation has been Encountered

This heuristic keeps a count of the number of times a particular interpretation for a compound has been selected as the best interpretation. An interpretation that has been selected several times should be better than one selected only a few times.

4.9. Combining the Results of the Heuristics

The information gathered by the heuristics can be used in several ways. One is to have each heuristic act on its own. If only one interpretation is selected by the heuristic, it is chosen as the best one. This clearly does not allow the constraints that may occur between heuristics to come into play. One simple way to combine the results of the heuristics is to associate a weight with each heuristic according to how much effect it should have on the final decision of which interpretation is best. Some heuristics are likely to have a large weight and are likely to overbear many of the other heuristics. Also, some heuristics (e.g., context) should be sufficient on their own to rule out some interpretations completely.

The scheme used in the current system to combine the results of the heuristics is a combination scheme. Some of the heuristics are allowed to rule interpretations out as impossible based on the current information. For example, the heuristics based on context should be allowed to prune interpretations from the list of possibilities. If the context does not allow for a particular interpretation, then it should be removed from consideration. Similarly, the "cognate" heuristic should be able to prune the list of interpretations in most cases. The idea is to allow those heuristics that are very accurate in their assessment of interpretations to remove from consideration any interpretations that reject. This reduces the number of possibilities the less accurate heuristics must consider.

The other less accurate heuristics can show a preference for a particular compound. If all these heuristics show a preference for one particular interpretation, this interpretation is probably better than any of the others. If no consensus exists, a weighted sum of the results of the heuristics is used to make a determination of which is the best interpretation. The "instances of interpretation" heuristic, the "slot verification" heuristic, and the "similar compounds" heuristic should be weighted relatively highly compared to the rest of the heuristics. The heuristics based on context can also provide a preference for a particular interpretation, much in the same way as the others, since context can support one interpretation more than another. Note that this assumes context supports more than one (or no) interpretation from the beginning, otherwise the one interpretation selected by context is selected. The heuristics based on context are weighted heavier than any other single heuristic. The "cognate" heuristic is a binary one, since either there is a cognate or there isn't for each interpretation. Thus, it only rules out possible interpretations as long as there is at least one cognate, and plays no further role.

5. Interfacing the System to a Parser

Chapter five of the thesis provides a partial solution to the problem of interfacing the ideas presented in the preceding chapters of the thesis to a parser that currently exists. The result is that a significant amount of semantic processing must be done to make available the information the current system needs to process the heuristics based on context. One thing that must be done before or in conjunction with noun compound processing is prepositional phrase attachment. The "slot already filled heuristic" can use the information about where prepositional phrases are attached to remove some possible interpretations from consideration. Also, to use the "general topic under discussion" and "selectional restriction" heuristics it is necessary for a far more complete semantic system to generate the kind of knowledge that is necessary for these heuristics to work.

The system can be interfaced to a parser, although the most effective way to do this is to use the ideas presented in the thesis rather than the current implementation. This allows for the incremental processing of the compound which in turn allows the noun compound processing component to help guide the parser make decisions as the parse is constructed.

6. Experiment to Determine How Well the System Performs on Real-World Examples

Chapter six describes an experiment in which over six hundred compounds were collected from various sources including: articles from the New York Times, articles from The Communications of the Association for Computing Machinery, an article from Newsweek, an article from a book containing a collection of papers, and an article from Scientific American. The algorithm for determining a meaning for a compound was hand simulated on all these examples. Just over sixty percent of the compounds would be processed correctly by the system using the techniques described in the thesis. Another thirty percent could probably be processed by the system with a few modifications to the current system or by making some assumptions. The modifications would involve adding new patterns to the current system and processing elliptical phrases. This seems to extend the coverage of the system by about six percent. Several assumptions account for the other twenty-four percent. One assumption would make some compounds cognates, i.e., the compounds have idiomatic meanings. The other assumption involves the existence of some slots that may be of a questionable nature. In many cases these slots should exist.

The remaining ten percent of compounds examined can not be correctly processed by the current system and can not be considered as cognates. The main problem seems to be that these ten percent of compounds require a much deeper semantic or pragmatic processing than the current system can perform. There seem to be several types of reasoning that must be considered to process correctly many of these compounds. In some

compounds it is necessary to understand that a process is involved between the nouns of the compound. For example, the compound "diesel particulates" is best understood by knowing that a "burning" process involving diesel oil can produce particulates. Some of the compounds have a metaphorical meaning that require a deeper semantic understanding than the current program has. The current system has a very strict view of what relationships can exist between concepts, a more relaxed view is necessary to understand some compounds.

7. Results

Chapter 7 discusses the results of the thesis in several different areas. It discusses the results produced by the program and presents several examples of the interpretations formed by the system. A description of the output produced by the program that describes the meaning for a compound stored in the data base is given. It also discusses the effectiveness of the various heuristics described in the chapter on finding the best interpretations. Some of the heuristics are extremely effective at disambiguating noun compounds, others are not so effective. For example, the heuristics based on context are extremely effective. The "cognate" heuristic is also effective while it can distinguish between two meanings of a compound. However, once all the meanings of a compound have been formed, the "cognate" heuristic is not able to help disambiguate the compound. The "similar compounds" heuristic is also fairly effective, though it sometimes produces the incorrect meaning when the most common one is not the one wanted. The "occurrence" heuristic is extremely weak. The "instances of an interpretation" heuristic is also fairly weak, especially since it is hard to test without biasing the experiment in its favour.

Finally, this chapter discusses how long it takes the system as it is currently implemented to process some of the compounds. It also gives some estimates of how long it is likely to take on one version of a NETL machine. On the simulated version the amount of CPU time varies widely according to the compound. Compounds with two words and only one meaning are processed in about one second of CPU time. Compounds containing several nouns take up to about five seconds if only one meaning is found. Compounds that are ambiguous can take up to thirty seconds of CPU time. Using a count of the number of cycles used by the simulated NETL machine it is possible to give estimates of how long it would take a NETL machine to process the same compounds. Assuming that the NETL machine has a cycle time of one milli-second produces the following timing information. Compounds of two words with one meaning take about half a second of real time to process. Compounds with several nouns take much longer, up to about twelve seconds of real time. Ambiguous compounds with two nouns take about one to two seconds to process if no context is provided. With context they take about a second of real time. Note that the timings for the NETL machine should remain fairly stable as the size of the data base increases because they depend on the length of the longest path between concepts. The time for the simulated version will increase at about the same rate

as the size of the data base, since it depends on the number of the nodes in the data base.

8. Conclusions

The last chapter presents several conclusions from the research described in the thesis. The answers to the following questions form the basis for the conclusions from this work:

- How hard is it to extend the program to process new compounds? Since much of the knowledge used to process compounds is contained in NETL's data base, in most cases it should only be necessary to add new knowledge to the data base to have the program process new compounds. No new procedures should have to be written to handle new compounds. It will be necessary to run the program on any new compounds because the data base is likely to contain errors.
- Is a real-world knowledge base required? One of the problems with understanding noun compounds is to reconstruct information that is omitted when the compound was formed. To reconstruct this information it is sometimes necessary to perform complex inferences about the concepts involved. To do this effectively requires a data base containing real-world knowledge.
- What features of NETL are necessary to process noun compounds? NETL provides several facilities the program to understand noun compounds uses to its advantage. Other knowledge representation systems could be used as the underlying representation scheme, but some searches would have to be done differently.
- How well does the program model the way people process noun compounds? It probably does not model the way people process compounds very well in the current implementation. It does things that people do not seem to.
- Does the program learn from experience? The "similar compounds" heuristic, the "cognate" heuristics, and the "instances of an interpretation" heuristics provide a simple but effective means of learning. Since the meanings of new compounds are added to the data base, these heuristics use this information to help disambiguate subsequent compounds.
- How well does the program deal with ambiguity? The program seems to be able to cope with some forms of ambiguity that any natural language system must cope with.
- How hard would it be to interface the program to a parser? Interfacing the program as it currently stands to a parser is probably the wrong way to proceed. As mentioned earlier, it should be possible to use the ideas presented in this thesis to help a parser disambiguate noun compounds.

Finally, there are several directions in which future research can be carried out. One is to increase the size of the data base to several hundred words from about two hundred as it currently stands. This would test how easy it is to extend the system by only adding new knowledge. This is hard to do with the current implementation since the address space of a PDP20 is too small to increase the size of the data base by more than about one hundred words. Another direction for future research is to try to use the ideas presented in the thesis to process other kinds of phrases. It may be possible to use similar techniques to process other types of phrases, especially prepositional phrases and verb phrases. Another possible direction for research is to use

a more flexible method of restricting the values of the slots of a concept. Currently, there are three possibilities for a concept trying to fill a slot: it can not fill a slot because of a split statement, it can fill the slot but is not the expected value, or it is the expected value for the slot. This can be extended by having slots restrictions having more possibilities, such as: a default value for the slot, a strong preference for a value of a slot, or must satisfy the restriction on the slot. This should allow the system to distinguish between some ambiguous compounds better. Finally, the current system has two phases. The second consists of a set of heuristics that are supposed to select the best meaning for a compound. It should be possible to move some or all of these heuristics back into the first phase. This has the advantage that more constraint can be placed on the generation of the possible candidates.

Table of Contents

1. Introduction	5
1.1. Understanding Noun Compounds	5
1.1.1. What is a Noun Compound?	5
1.1.2. Sample Output	6
1.1.3. Examples of Problems	8
1.1.4. Why Study Noun Compounds?	11
1.2. Related Work	12
1.2.1. Work by Linguists	12
1.2.2. Work by Artificial Intelligence Researchers	14
1.3. Analyzing a Noun Compound	16
1.3.1. The Representational System	16
1.3.2. Finding Possible Interpretations	18
1.3.3. Deciding on Which Interpretation is Best	20
1.3.3.1. Context	21
1.3.3.2. Cognates	22
1.3.3.3. Embedded Cognates	22
1.3.3.4. Instances of an Interpretation	22
1.3.3.5. Embedded Instances of an Interpretation	22
1.3.3.6. Verifying Slots	23
1.3.3.7. Similar Compounds	23
1.3.3.8. Number of Times an Interpretation has been Encountered	23
1.3.3.9. Combining the Results of the Heuristics	24
1.3.4. Implementation	25
1.4. Overview of the Thesis	25
2. The NETL Representation System	27
2.1. Type Hierarchies	29
2.2. Split Statements	32
2.3. Roles and Mapping	34
2.4. Statements	37
2.5. Representation of a Word	38
2.6. Representation of Nominalized Verbs	40
3. The Possible Interpretations for a Compound	43
3.1. The Two Noun Problem	44
3.1.1. Investigating One Meaning of a Noun	46
3.1.2. Finding Candidate Meanings	50
3.2. Compounds with Three or More Nouns	55
3.3. The Algorithm to Find Possible Interpretations	62

3.4. Results of Finding Possible Interpretations	66
3.4.1. Creating Structure to Represent an Interpretation in NETL	66
4. The Best Interpretation for a Compound	71
4.1. Context	72
4.1.1. Selectional Restrictions	73
4.1.2. Slot Already Filled	77
4.1.3. General Topic under Discussion	78
4.1.4. Using the Results of Context	79
4.2. Cognates	80
4.3. Embedded Cognates	86
4.4. Instances of an Interpretation	87
4.5. Embedded Instances of an Interpretation	88
4.6. Slot Verification	89
4.7. Similar Compounds	90
4.8. Number of Times an Interpretation has been Encountered	94
4.9. Combining the Results of the Heuristics	95
5. Interfacing the Compound System to a Parser	99
5.1. The Structure of Parsifal	99
5.2. Interfacing the Noun Compound System to Parsifal	104
5.3. Examples of How the Interfacing Should Work	112
5.3.1. Cargo Bay Doors	112
5.3.2. Text Processing Systems	115
5.3.3. Police Search	117
5.3.4. Money Supply	118
5.4. Finding the End of a Noun Compound	119
5.5. Summary	122
6. An Experiment using Real-World Examples	125
6.1. Correct Left to Right Parse of Noun Compounds	129
6.2. Compounds Formed from the Middle Ternary Pattern	129
6.3. Compounds formed from the End Ternary Pattern	130
6.4. Multiple Interpretations due to Multiple Meanings of Nouns	130
6.5. Multiple Interpretations due to More than One Slot	131
6.6. No Meaning is Generated for the Compound	133
6.7. The Wrong Meaning is Generated for the Compound	133
6.7.1. Ellipsis	134
6.8. A Questionable Slot is used to Generate the Correct Meaning	137
6.9. A Questionable Slot Assignment is used to Generate the Correct Meaning	139
6.10. The Correct Meaning as far as it Goes	139
6.11. The First Noun Provides a Slot for the Second One	140
6.12. The Meaning of the Compound is Provided in the Text	141
6.13. A Group of Concepts Fill the Same Slot of the Head Noun	142
6.14. Multiple Word Senses	142
6.15. The Possible Relationships Between Two Nouns	144
6.16. Summary	145

7. Results	149
7.1. Results of the Program	149
7.2. Effectiveness of the Heuristics	155
7.2.1. Context	155
7.2.1.1. Selectional Restrictions	155
7.2.1.2. Slot Already Filled	155
7.2.1.3. General Topic	156
7.2.2. Cognate and Embedded Cognate Heuristics	156
7.2.3. Instances and Embedded Instances of Interpretations	157
7.2.4. Slot Verification Heuristic	157
7.2.5. Similar Compounds	158
7.2.6. Occurrence Heuristic	158
7.3. Completeness	158
7.4. Program Timings	159
8. Conclusions	163
8.1. Extending the Program to Process New Noun Compounds	164
8.2. Need for Real-World Knowledge to Understand Noun Compounds	165
8.3. Features used in NETL and their Necessity	165
8.4. Modelling the Way People Process Noun Compounds	167
8.5. Learning	167
8.6. Dealing with Ambiguity	168
8.7. Interfacing the Program to a Parser	169
8.8. Directions for Future Research	169
8.8.1. Extending the Coverage of the System	169
8.8.2. Increasing the Size of the Data Base	170
8.8.3. Extending the Paradigm to Other Phrases	170
8.8.4. Using Default Values for Slots	171
8.8.5. Moving the Heuristics Back into the Generator	173
Bibliography	175
Appendix A. Example Run of the Program	179
Appendix B. Examples of Noun Compounds	191
B.1. Examples from an article in Scientific American	192
B.2. Examples from the Communications of the Association for Computing Machinery	203
B.3. Examples from an Article in a Book	219
B.4. Examples from the April 27, 1981 Newsweek Article on the Space Shuttle	226
B.5. Examples from Articles in the New York Times	232

List of Figures

Figure 1-1:	The components and information flow for the program.	17
Figure 2-1:	A non-tangled hierarchy as viewed from the interior node X.	29
Figure 2-2:	A tangled hierarchy as viewed from the interior node X.	30
Figure 2-3:	A simple type hierarchy with Clyde (who is an elephant) represented as an individual node.	31
Figure 2-4:	Representation of a Split Statement with the ancestors of the node Clyde marked with marker M1.	33
Figure 2-5:	NETL detects a clash between Clyde and cabbage after the ancestors of "cabbage" are marked with marker M2.	34
Figure 2-6:	Definitions for the "heart" in-role and the "weight" of-role for the typical "elephant".	36
Figure 2-7:	Mapping the nose role of mammal down to elephant and then down to Clyde.	37
Figure 2-8:	The typical elephant's trunk is a cylinder of length 1.3 metres.	37
Figure 2-9:	Definition of the hates statement, with in-roles for the hater and the thing hated.	38
Figure 2-10:	Creating an individual statement of the hates statement to represent "Clyde hates snakes".	39
Figure 2-11:	Representation of the word plant and associated concepts.	40
Figure 2-12:	Representation of the nominalized verb "evaluation".	41
Figure 3-1:	Simplified NETL Structure Necessary to Understand the Compound "Car Maintenance"	45
Figure 3-2:	First Step of the Search to Find the Possible Slots for One Meaning of "Glass"	47
Figure 3-3:	Marking the Slots of One Meaning of the Word "Glass"	47
Figure 3-4:	Mark all the Ancestors of "&Car" to Find the Slots that "&Car" can not Fill	48
Figure 3-5:	Mark all the Split Statements that Point at a Node Marked During the First Step	49
Figure 3-6:	Mark all the Nodes Pointed at by the Split Statement, Except Those Already Marked by M1	49
Figure 3-7:	Perform a Downward Scan from all Nodes Marked with M3	50
Figure 3-8:	NETL Network Showing How Possible Interpretations are Found	52
Figure 3-9:	Find Interpretations for the Compound "Woman Miner" Using Pseudo-Adjectives	53
Figure 3-10:	Network after default assumptions are marked for "&miner" with marker M2	55
Figure 3-11:	Network after the superiors of defaults for "&miner" are marked with marker M3	55
Figure 3-12:	Network after the "&woman" and its superiors are marked with marker M4	56
Figure 3-13:	Network to Find Possible Meanings for the Compound "Coal Mine Supervisor"	58
Figure 3-14:	Network to Find Possible Meanings of the Compound "Glass Wine Glass"	59

Figure 3-15: Network to Find the Possible Meanings of the Compound "Watch Repair Person"	61
Figure 3-16: Network to Find the Possible Meanings of the Compound "Government Price Support"	62
Figure 3-17: Description of the compound "car maintenance".	67
Figure 3-18: The Network Structure Built to Represent the Meaning of the Compound "Car Maintenance"	68
Figure 3-19: The Network Structure Built to Represent the Meaning of the Compound "Watch Repair Person"	69
Figure 3-20: The Network Structure Built to Represent the Meaning of the Compound "Woman Coal Miner"	70
Figure 4-1: Example Network for Use by the Selectional Restrictions Heuristic.	76
Figure 4-2: The Network to Find Cognates of "Wine Glass".	82
Figure 4-3: The Network After the Descendants of the "Contain!Containee" Slot is Marked.	82
Figure 4-4: The Network After the Slot Filler "Wine" is Marked.	83
Figure 4-5: The Network After Slots not Marked by the Slot Filler Marker are UnMarked.	83
Figure 4-6: The Network Showing Possible Cognates (nodes marked with M_1 and M_4).	84
Figure 4-7: Network Structure for Ternary Compounds with a Middle Nominalized Verb.	85
Figure 4-8: Example Network for Verifying Slots of "wood production".	89
Figure 4-9: Network to Find Similar Compounds of "steel truck" after initialization.	91
Figure 4-10: The Network After Stepping Up One VC Link for the Compound "steel truck".	93
Figure 4-11: The Network After Several Steps Upward have been Processed.	93
Figure 4-12: The Network Showing Possible Similar Compounds for "steel truck".	94
Figure 5-1: State of Parsifal's data structures during a parse.	100
Figure 5-2: Some sample grammar rules for Parsifal	101
Figure 5-3: Example of an attention shifting rule	103
Figure 5-4: Attention shifting rule that starts noun compound processing	106
Figure 5-5: Rule for parsing middle ternary compounds	107
Figure 5-6: Rule to process a compound fitting the end ternary pattern	108
Figure 5-7: Rule to parse a compound left to right	108
Figure 5-8: Two rules to parse a compound from right to left	109
Figure 5-9: Rule that finishes the processing of a noun compound	109
Figure 5-10: Rules for handling the case when there is a collapsed clause	110
Figure 5-11: State of the parser just before rule Start-NC is run.	112
Figure 5-12: State of the parser after rule Start-NC has run.	113
Figure 5-13: State of the parser after the first sub-compound is formed.	114
Figure 5-14: State of the parser after the "door" has been processed.	115
Figure 5-15: Parser state for "text processing systems" after rule Start-NC is run.	116
Figure 5-16: Parser state for "text processing systems" after rule Parse-MT is run.	117
Figure 5-17: Parser state for "police search" at then end of the parse.	118
Figure 5-18: Parser state for "money supply" after its parse is complete.	119

List of Tables

Table 4-1:	Table of Weights used in the Final Rating of an Interpretation	96
Table 6-1:	Number of compounds of each length found and the average number of sentences between finding a new compound.	125
Table 6-2:	Results of using the proposed model on real-world examples	128
Table 7-1:	Table of Comparative Timings for Simulator and NETL Machine	162

Acknowledgements

I am deeply indebted to my thesis adviser, Scott Fahlman, for his advice and guidance throughout the research and writing that produced this thesis. I am also grateful for the advice and suggestions made by the other members of my thesis committee: Allen Newell, Jaime Carbonell, Phil Hayes, and Bonnie Webber. Their suggestions have greatly improved the thesis. I would also like to thank the NETL group -- Dave Touretzky, Walter van Roggen, and Leonard Zubkoff -- for their help.

I would also like to thank all the rest of the people of the Computer Science Department at Carnegie-Mellon for making it such a wonderful place to do research.

Finally, I would like to thank my parents, my brother, and my sisters for moral support throughout my career as a graduate student.

PLEASE NOTE:

Page 4 is lacking in number
only. No text is missing.
Filmed as received.

UNIVERSITY MICROFILMS INTERNATIONAL.

Chapter 1

Introduction

1.1. Understanding Noun Compounds

Noun compounds are a phenomenon that occur frequently in real-world text. This thesis explores the problem of how a computer can understand noun compounds and describes a program that can understand many noun compounds. It is necessary to approach the problem using semantic and real-world knowledge, since syntactic information provides little help in deciding the correct syntactic structure, let alone the meaning of a noun compounds. The following sub-sections describe the problem of determining the meaning of a noun compound more fully.

1.1.1. What is a Noun Compound?

A noun compound is a sequence of two or more nouns (e.g., "computer science", "car assembly plant", "glass wine glass", "water meter cover adjustment screw", etc.) in which the *head noun* or last noun in the sequence is usually modified by the other nouns in some manner. Noun compounds are used frequently by people to describe things. To successfully communicate, a person must have a model of the subject being discussed similar to that of the person with whom he is communicating. When a person uses a noun compound, he is usually using a shorthand notation to save time in communication. The reader must reconstruct the information deleted by the writer, if the reader is to correctly comprehend the phrase.

The problem addressed here is how to produce one interpretation from such a sequence of nouns. Or, in other words, what is the knowledge necessary to reconstruct the information deleted when the compound was formed. Syntactic knowledge is not sufficient to understand most compounds. Rather, it is necessary to access semantic knowledge. The semantic knowledge needed to understand a compound depends on the compound under consideration. An interpretation for some compounds can be produced just using the meanings of the nouns, without any further knowledge. For example,

the most likely meaning of the compound "dog owner" can be found using the definitions of the words "dog" and "owner". However, other compounds require more knowledge to successfully produce an interpretation. For example, the compound "bee sting", has at least two meanings (i.e., a wound caused by a bee, and the part of a bee that can cause such a wound), both of which are reasonable interpretations. The method to distinguish between them relies on contextual information that normally surrounds the compound.

1.1.2. Sample Output

This sub-section presents a few brief examples of the meanings created by the system (the end of chapter 3 gives more details on the creation of meanings for compounds). The first example is fairly simple:

```
(meaning '(dog owner) nil)
There is 1 compound after finding all possibilities.
```

```
Creating compound: DOG OWNER
DOG OWNER[&DOG OWNER-A] is an OWNER[&OWNER].
OWNER OWNS: DOG[&DOG]
```

The first line of the example invokes the program with the compound "dog owner". The second line specifies that the system found only one possible interpretation for the compound. The third line specifies that the system is about to create a meaning for the compound in its data base. The fourth line specifies the word sequence followed by the name of the concept that represents the meaning of the compound "dog owner" in square brackets and then specifies its ancestor after the phrase "is an". In this case it uses the word sequence "owner" followed by the name of the concept representing the meaning of "owner" as a something that can own things (i.e., &owner). The last line indicates the relationship that exists between the two nouns. In this case, a "dog" is owned by an "owner".

The next example is the compound "water meter cover adjustment screw". This compound is quite long, but the system is able to create the only reasonable meaning of the compound. This is due to the fact that the compound itself provides sufficient constraint on how the meaning of the compound can be constructed. Note that the meaning "water meter cover" is used as the filler for one of the slots in the ternary relation based on "adjustment". The system is able to use two different patterns to form the meaning of a compound without any problem.

(meaning '(water meter cover adjustment screw) nil)
There is 1 compound after finding all possibilities.

Creating compound: WATER METER COVER ADJUSTMENT SCREW
WATER METER[&WATER METER-A] is a METER[&METER].
METER|MEASURES: WATER[&WATER]
WATER METER COVER[&WATER METER COVER-A] is a COVER[&COVER].
COVER|COVERS: WATER METER[&WATER METER-A]
WATER METER COVER ADJUSTMENT SCREW
[&WATER METER COVER ADJUSTMENT SCREW-A]
is a SCREW[&SCREW].
&ADJUST-1
ADJUST|TOOL-OF-&ADJUST-1: SCREW[&SCREW]
ADJUST|ADJUSTEE-OF-&ADJUST-1:
WATER METER COVER[&WATER METER COVER-A]

The next example shows what happens when an ambiguous compound is presented to the system. The compound "coal bill" has two meanings. One as a bill of law regarding some aspect of coal, and the other as a bill presented to someone who purchased coal. The system finds both meanings for this compound and creates both in the data base, as shown below:

(meaning '(coal bill) nil)
There are 2 compounds after finding all possibilities.

Creating compound: COAL BILL
COAL BILL[&COAL BILL-A] is a BILL[&BILL-B].
BILL-B|BILL-FOR: COAL[&COAL]

Creating compound: COAL BILL
COAL BILL[&COAL BILL-B] is a BILL[&BILL-A].
BILL-A|ABOUT: COAL[&COAL]

To distinguish between these two meanings it is necessary to provide context to the system. Since this is not a full language understanding system, it is not possible to use directly a sentence or phrase to provide the context. Rather, it is necessary to present the equivalent information to the system. This is done by presenting the program with a list representing the part of a sentence that provides the context. For example, the following two sentences provide sufficient context to allow the system to disambiguate the compound "coal bill".

1. Congress voted on the coal bill.
2. The farmer paid the coal bill.

When the program is presented with the context these two sentences provide, it produces the following results.

(meaning '(coal bill) '(((&vote-a () (vote-alvoter &Congress))))))
 There are 2 compounds after finding all possibilities.

Creating compound: COAL BILL
 Using cognate: &COAL BILL-B
 COAL BILL[&COAL BILL-B] is a BILL[&BILL-A].
 BILL-A|ABOUT: COAL[&COAL]

(meaning '(coal bill) '(((&pay-a () (pay-alpayer &farmer))))))
 There are 2 compounds after finding all possibilities.

Creating compound: COAL BILL
 Using cognate: &COAL BILL-A
 COAL BILL[&COAL BILL-A] is a BILL[&BILL-B].
 BILL-B|BILL-FOR: COAL[&COAL]

In the first example, the context equivalent to sentence (1) above is presented to the system by the list (&vote-a () (vote-alvoter &Congress)) which specifies that the compound occurred in the context of a vote action where Congress does the voting. The system finds that a "coal bill" as a bill of law can be voted on. The other meaning is not supportable by this context, and thus the system selects the meaning "a bill of law about coal". Similarly, context equivalent to sentence (2) above can be provided to the system, and it selects the other meaning.

These three examples show some of the capabilities of the system. It can disambiguate other compounds that require other kinds of knowledge.

1.1.3. Examples of Problems

There are several problems that arise when an attempt is made to understand a noun compound. Meanings for many compounds can be determined from a relationship between the nouns in the compound. However, there are some compounds whose meaning is not determined by such a relationship. Rather, their meaning is associated with the sequence of words in the compound. For example, consider the compound "noun phrase". Most people would recognize this compound as a phrase that contains nouns as well as other parts of speech because it has a well defined meaning in linguistics. It does not mean a phrase that consists of just nouns which is the meaning the program would find for it. Most people would probably find the same meaning if they did not know the linguistic meaning for the compound. Any system to understand noun compounds must deal with ones having a meaning distinct from the meaning provided by an underlying relationship between the nouns. Such a compound must be stored in the system's data base so that if the correct sequence of nouns appears, the built-in meaning becomes a possible interpretation. This does not mean that a

built-in meaning is instantly used as the best one, only that it is a very good candidate. For example, in some contexts the built-in meaning for "noun phrase" could be the wrong meaning (e.g., if the person speaking the phrase did not know the linguistic meaning). In such a case, the built-in meaning should not be used, and another meaning should be considered. Other examples of such idiomatic compounds include: "chain gang" and "crown prince".

Another problem a theory to understand noun compounds must deal with is multiple meanings of words. This is a problem that any natural language system must cope with, but is no less important here. Some words in a compound can be disambiguated from the context provided by the compound itself. For example, consider the compound "glass wine glass". The word "glass" has at least the two meanings:

1. A container from which people drink certain liquids.
2. An amorphous material (silicon dioxide) from which certain things are made (e.g., a glass as above).

In the compound "glass wine glass", the first "glass" has meaning (2), and the second meaning (1). This can be established fairly easily, since the first "glass" has no underlying relationship with "wine" that makes sense given the order of the words. The second "glass" fits best if meaning (1) is used, since wine is a liquid that people drink. Then the first "glass" fits nicely by being the material from which the second "glass" is made. In this example, the multiple meanings of the word "glass" do not cause any difficulty because the compound itself provides sufficient structure to determine the correct meaning of "glass".

In the example above, the compound contains sufficient constraints to determine which meaning of "glass" was meant. There are compounds for which this is not the case. For example, consider the compound "coal bill" that was used as an example earlier. This compound has at least two meanings. The compound "coal bill" does not by itself contain sufficient information to decide which of the meanings is the correct one. To make this decision, it is necessary to provide more context than that provided by the compound itself. The context necessary to disambiguate the compound must come from the surrounding phrases. In this case, context can be provided by a template for a verb with some of its slots filled. The system must determine from this context the only reasonable meaning for the compound "coal bill".

Another problem is that more than one relation can be used to form a meaning for a compound. This is likely to happen fairly often as any good sized data base of knowledge will have several

relationships between some nouns. This means any system to understand noun compounds must select the relation producing the best meaning. For example, consider the compound "computer evaluation". Assume the concept "evaluation" has two slots: one for the thing doing the evaluation, and the other for the thing evaluated. There are other slots available (e.g., the time when the evaluation took place and the location where it occurred) to use as the basis for an interpretation, but these will be ignored in this discussion for simplicity. Since in some contexts a computer can be the thing doing the evaluation (e.g., computer evaluation of an election), and in others it is the thing evaluated, both slots lead to a meaning for the compound "computer evaluation". Normally, a compound of the form "x evaluation" has the meaning where x is the thing evaluated (e.g., "car evaluation", "stereo evaluation", etc.). Assume the data base contains several compounds of the form "x evaluation", where x is the thing evaluated, and a few (if any) where x is the thing doing the evaluation. One source of information to decide which relationship is the best one to use in forming a meaning, is to look for compounds that are similar to "computer evaluation" and use a similar structure for this compound. This is likely to lead to selecting computer as the thing evaluated, rather than the thing that is doing the evaluation.

Of course, this cannot always happen, otherwise in the context of "computer evaluation of an election", the wrong interpretation for "computer evaluation" will be chosen. It is necessary to provide context to enable the system to produce the correct interpretation. The best way to provide this context is to specify that one of the possible relationships between "computer" and "evaluation" is already used. In this case, the relationship that specifies the thing evaluated is filled with "election". This leaves only one possible relationship between "computer" and "evaluation", i.e., the thing doing the evaluation. Chapter 5 on how to interface the system to a parser addresses how this information can be obtained in more detail.

So far, the ambiguities pointed out above are not caused by lack of knowledge, but by the inherent ambiguity local to noun compounds. Another problem arises because the data base may not contain all the necessary information to obtain only those relationships that should exist between nouns. For example, consider the compound "blood evaluation". If the data base does not contain the information that "blood" cannot be something that can perform an "evaluation", two possible relationships will be found (i.e., where "blood" is the object being evaluated and the spurious one). People understand this because they "know" that "blood" can not perform an "evaluation"; if the computer doesn't, it must consider such an interpretation. It is unreasonable to expect the methods described above to obtain the correct interpretation, since they deal with ambiguity inherent in natural language, not ambiguity created by the data base. One method to cope with this problem

satisfactorily is to check the relationships to make sure they are the ones expected. In the "blood evaluation" example, this means checking if "blood" is expected to be the thing doing the evaluation, or the object evaluated. In this case, it is expected to be the object evaluated, and thus this interpretation is more likely to be correct. This check can also help disambiguate a word having multiple meanings.

Any system to produce the one best interpretation for a compound must address the problems presented above. One of the goals of the thesis is to determine the knowledge necessary to solve these problems in such a way that one interpretation can be produced.

1.1.4. Why Study Noun Compounds?

There are several reasons why the study of noun compounds is interesting. A major reason is if a computer is ever to understand natural language in domains other than so called "toy" ones, it will have to understand noun compounds. People often use noun compounds to describe things, and a computer must be able to determine what is being described.

A further reason is the problem of understanding noun compounds encompasses a large number of problems occurring in natural language understanding. Natural language understanding systems must disambiguate phrases they come across. Many words have multiple meanings and a natural language understanding system must determine which meaning the writer meant. Also, some words have more than one part of speech. Determining which part of speech a word is may help disambiguate a phrase. Similar problems arise with noun compounds, since some of the words in a compound may have multiple meanings. To understand these compounds, a decision must be made to determine the best meaning. Most compounds are formed originally because some relationship exists between the nouns in the compound. When a system tries to find an interpretation of compounds, by looking for possible relationships between nouns, several relations may be found. The system must disambiguate compounds based on the relations found, as well as on the meanings of the words involved.

Another reason is to explore what kinds of knowledge are necessary to understand noun compounds. Several heuristics are proposed in this thesis to help make the decision of which is the best interpretation. The heuristics to disambiguate noun compounds may not all be relevant to disambiguating other types of phrases, but they may give some insight into the kinds of knowledge necessary to disambiguate other phrases occurring in natural language.

Many of the techniques developed in the thesis could possibly be used to process other types of phrases. For example, the problem of prepositional attachment might be able to be processed to some extent using similar techniques. Any problem that involves deriving semantic relationships between concepts should be able to be approached using similar techniques.

1.2. Related Work

Both linguists and AI researchers have studied noun compounds. There is a fair amount of information in the literature about how linguists have approached the problem. However, for computational models, there are only a handful of efforts directly related to noun compounds.

1.2.1. Work by Linguists

Linguists first became interested in noun compounds in the middle part of this century with the work by Jespersen [Jespersen 46]. Noun compounds are a linguistic phenomenon for which linguists want to develop theories to explain how they are formed. Linguists have traditionally been more interested in the generation of noun compounds than in how people understand them. Thus, their work tends to look at noun compounds from a different perspective than an artificial intelligence researcher developing a computer model to understand natural language.

Most of the linguists who have studied this problem have attempted to come up with a set of relationships between nouns to describe how noun compounds are formed. This approach tends to cover a large number of possible compounds. However, there will always be a few compounds that do not behave according to the expected set of relationships and cause exceptions to these general rules. Examples of this approach are the theories developed by Jespersen [Jespersen 46], Hatcher [Hatcher 65], and Marchand [Marchand 69]. Another approach that linguists have used is to create a grammar that transforms an underlying phrase or sentence into a noun compound. The work by Lees [Lees 60, Lees 70] is an example of this approach. As part of the grammar, Lees also developed a categorization scheme that many noun compounds fall into. An example of a categorization scheme is the one developed by Lees based on his grammar to nominalize sentences:

1. **Subject-Predicate:** The subject and the predicate adjective of the underlying sentence are used to form a compound (e.g., girl friend, fighter plane).
2. **Subject-Middle Object:** The subject and a possessive genitive construct in the underlying sentence are used to form the compound (e.g., doctor's office, arrowhead, rattle snake).

3. **Subject-Verb:** The subject and the main verb of the underlying sentence are used to form the compound (e.g., payload, population growth, farm production).
4. **Subject-Object:** The subject and the object of the underlying sentence are used to form the compound (e.g., steamboat, car thief, water spot, automobile plant).
5. **Verb-Object:** The main verb and the object of the underlying sentence are used to form the compound (e.g., setscrew, pickpocket, cost reduction, book review).
6. **Subject-Prepositional Object:** The subject and a case of the main verb introduced by a preposition in the underlying sentence are used to form the compound (e.g., gun powder, garden party, egg plant, cell block).
7. **Verb-Prepositional Object:** Similar to Subject-Prepositional Object, except the main verb of the underlying sentence is used (e.g., grind stone, recovery time, boat ride, sleep walker).
8. **Object-Prepositional Object:** Similar to Subject-Prepositional Object, except the Object of the underlying sentence is used (e.g., bull ring, station wagon, wood alcohol, block house).

The other categorization schemes are similar in nature though they differ in particulars.

One of the latest approaches taken is by Downing [Downing 77] in which she does not rely on a fixed set of relationships as the basis for generating (or understanding) noun compounds. Rather, she claims there is no fixed set of relationships and that some relationships depend on the intended meaning of the compound. Sometimes, the relation must be determined by the appropriate context. She also sets forth a set of relationships, similar to those of other linguists before her, that seem to capture a large number of compounds, but states that they can not capture all the possible relationships that occur. Downing has no proposals for dealing with relations that are not on a fixed list, she just notes that they occur.

Linguists have developed several schemes for showing how noun compounds are formed. In general, they do not address the problem of how to understand a noun compound. That is, given only the compound, they provide no procedure to determine the category to which a compound belongs. There are often many inferences necessary to understand a compound for which the generative models developed by linguists provide no help. The basic problem is the categorization schemes seem to need the meaning of a compound before they can be used effectively. Once a meaning for a compound has been determined, it should be fairly easy to classify the compound according to any of the various categorization schemes.

1.2.2. Work by Artificial Intelligence Researchers

Artificial Intelligence researchers, especially those interested in natural language understanding, have also performed work in the area of noun compounds. Some of the researchers have followed the most prevalent linguistic approach, i.e., they have used a fixed set of relationships between nouns to either generate compounds from phrases, or to produce interpretations for a compound. Most of these efforts have restricted themselves to compounds containing only two nouns, rather than the more general problem of compounds with an arbitrary number of nouns.

An example of the generation of noun compounds from phrases is the work by Rhyne [Rhyne 76]. In his thesis, Rhyne describes a program that takes a relative clause such as "a thief who steals cars", and produces a noun compound from it (i.e., "car thief"). Rhyne used a number of classes for compounds, each class has specific surface level cases [Fillmore 68, Simmons 73] that could be associated with any verb for which that case is permissible. The class that a phrase fits into is used to determine the form of the compound generated. The input to the program is a relative clause already having the correct cases assigned to the nouns involved. This work does not easily extend to understanding noun compounds. Although Rhyne mentions the possibility that the rules can be used in reverse with some sort of inference scheme, it is not immediately clear that this is as easy as Rhyne makes it sound. When a compound is formed, a large amount of information may be deleted that may be very difficult to reconstruct. The burden in reconstructing this knowledge is placed fully on the inference system.

An example of understanding compounds from a fixed set of relationships is the work done by Russell [Russell 75]. In her thesis, Russell uses the conceptual dependency representation of Schank [Schank 73] as the basic meaning representation for the words in her dictionary and the compounds she forms. The definitions of nouns are augmented with feature information used by a set of rules to decide what relationship must exist between two nouns to form a particular meaning for a compound. The problem with this system is that some of the best ranked meanings would not be ranked best by a person. For example, consider the two compounds "garage sale" and "house sale". Most people would expect "garage sale" to mean a sale that takes place in a garage, whereas "house sale" is the sale of a house. Russell's system would take the sale of a house or garage as the best meaning for both, since both have the same semantic features. A system to understand noun compounds must be able to cope with idiosyncratic knowledge of this nature. Also, the rules used by her system have been designed to work with only a subset of nouns. To handle the nouns ignored, new semantic features may be necessary. As new semantic features are added to the system, the rules that form

meanings of compounds must be changed to account for the new features. New rules may also have to be added. Also, Russell makes no claim about the need for a lexicon of exceptions, even though there is clear evidence that such a lexicon is necessary.

The latest work in the area of noun compounds by an artificial intelligence researcher is the thesis by Finin [Finin 80]. Rather, than rely on a fixed set of relationships that can exist between two nouns, Finin uses a data base of real world knowledge. The relationships used to form interpretations for a compound are derived from the relationships that exist in the data base. Thus, Finin is not locked into a fixed set of relationships which must be augmented if a compound is encountered that does not fit the normal set of relationships. Rather, new relationships may be added freely by adding new information to the data base. The main area where his work differs from mine is the method for selecting the best interpretation for a compound.

Finin selects the best interpretation for a compound by using scoring functions based on how well concepts fill slots. There are several scoring functions that can be associated with a slot. Each scoring function computes a value according to how well it believes a concept fills a slot. These values are added together to obtain the final value for a concept filling a slot. The interpretation with the best fit for concepts filling slots is selected as the meaning for the compound. This is a fairly easy and simple way to select the interpretation that a person is likely to think of first given no context for a compound. However, in some contexts Finin's system will select the wrong meaning or will have two equally good meanings.

The current system attempts to get around this problem by bringing more knowledge to bear. Some of the heuristics used in the current system have a similar purpose to Finin's scoring functions. However, there are several other heuristics that go beyond these scoring functions. In particular, the heuristics based on context are very effective in selecting the correct meaning for a compound when there are several likely candidates. Also, some of the heuristics such as the "instances of an interpretation" heuristic and the "similar compounds" heuristic attempt to use general, non-linguistic knowledge that people might have. These heuristics are not directly related to how well a value fills a slot. Thus, the current system attempts to use more knowledge to solve the problem of understanding noun compounds than Finin's system does.

1.3. Analyzing a Noun Compound

A system to understand compounds using real world knowledge must have several components. Figure 1-1 shows the various components in the system as well as the flow of information. The following components exist in the current program:

- A knowledge representation system that is capable of storing a large range of information, including definitions of words, other static knowledge about concepts, and dynamic knowledge such as contextual information and script or frame-like information [Minsky 75, Schank 75].
- A method for finding the possible interpretations for a compound. It must be able to interface flexibly to the knowledge representation system, so that the information stored there can be accessed when necessary.
- A mechanism for deciding which interpretation is the best one if more than one interpretation is possible.

The double arrows show information that is input to the system. In particular, the sequence of nouns to be understood is one input, and the context (if any) in which the compound occurred is another. The solid single arrows shows the direction that information about a particular compound flows. The dashed arrows show the direction that information flows with regard to the data base. The following sub-sections describe the above components in more detail.

1.3.1. The Representational System

The organization of the current system is based on a data base containing the meanings of all the concepts necessary to understand noun compounds. The program uses the semantic network representation system, NETL, developed by Fahlman [Fahlman 79] as the underlying knowledge representation system. Chapter 2 gives a more detailed description of the features in NETL that are used to understand noun compound in the current system. NETL can store definitions of words, as well as other real world knowledge necessary to understand noun compounds. The knowledge stored within NETL is easily accessed, and searches can be performed to find the information pertinent to what the system is currently doing. NETL is designed to run on a parallel architecture so that certain types of searches can be executed very quickly. This ability to do parallel searching is used to great advantage by the system to understand noun compounds. The current implementation uses a NETL simulator that runs under MacLisp on a PDP 10 or 20, since no NETL machine currently exists.

Information in NETL is stored as a hierarchy of concepts. Many of the concepts have words

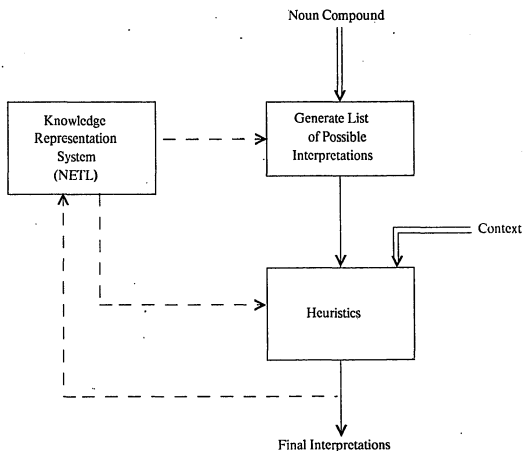


Figure 1-1: The components and information flow for the program.

associated with them, but not necessarily all. Concepts can inherit information from other concepts higher in the hierarchy. They cannot inherit information from concepts lower in the hierarchy. A very important idea in NETL, and one which is heavily used in the program, is the split set. A split set is a set of concepts in NETL, each of which is distinct from the other. For example, a split set is formed between the two concepts "living things" and "non-living things". That is, no entity can be a "living thing" and a "non-living thing" at the same time. This allows the system to distinguish between a "living thing" and a "non-living thing" easily.

As well as having some concepts as members of split sets, any concept in NETL may have what are called *rules* or *slots*. Roles allow specific information about a concept to be associated with the concept. For example, the NETL node representing the concept "elephant" has many slots, one of which is a slot representing an elephant's trunk. A concept and its slots can be thought of as a frame

in the Minsky sense [Minsky 75]. Thus, all the information about a concept is accessible from one node in the network.

1.3.2. Finding Possible Interpretations

The basic approach taken to find the meaning of a noun compound is generate and test. The system is currently split into two phases. The first phase generates a list of possible interpretations and the second phase tries to find the best interpretation for the compound from the ones generated. This sub-section gives a brief description of how the possible interpretations for a compound are found. Chapter 3 gives complete details of how the current system finds possible interpretations for a compound. The next sub-section briefly describes how the best interpretation is selected.

Once definitions of some nouns and other information is placed into the network, a mechanism must be provided to produce an interpretation for a compound. The basic assumption of the system is an interpretation for a two noun compound can be produced by filling a slot in the head noun by the modifying noun. For example, assume the definition of "exploration" has a slot specifying the thing explored for as a mineral, among others. The compound "gold exploration" is interpreted by filling the "thing explored for" slot of "exploration" with the concept representing "gold". Thus, the first step to finding interpretations for a compound of the form " $N_1 N_2$ ", where N_1 and N_2 are nouns, is to find all the slots of N_2 that can be filled by N_1 . For compounds containing only two nouns, this strategy seems to work well. However, when there are compounds containing more than two nouns, there are more patterns than just the simple one above.

Some compounds with more than two nouns have the form $[\dots [[N_1 N_2] N_3] \dots N_n]$. That is, they are formed in a left to right fashion and can be processed easily left to right. The noun group immediately to the left of a noun fills a slot in the noun. Some examples of this type of compound are: "coal mine supervisor", "blood donor recruiter", and "sugar cane plantation owner".

Some compound with more than two nouns have the form $[N_1 [N_2 [N_3 \dots N_n]]]$. That is, they are formed in a right to left fashion. An example of such a compound is "glass wine glass".

Some compounds contain nominalized verbs in the middle of the compound. A nominalized verb is a noun that is derived from a verb (e.g., the nominalized verb "evaluation" describes a "to evaluate" act). These compounds form ternary relations with two slots of the nominalized verb filled by the surrounding nouns (or sub-compounds possibly). Some examples of such compounds are:

"car assembly plant", "food distribution centre", "watch repair person", and "water meter cover adjustment screw". In the last example, the meaning for "water meter cover" is created in a left to right parse as described above.

Some compounds contain a nominalized verb at the end of the compound. These compounds also form ternary relations with the two previous nouns filling slots in the nominalized verb. Some examples of such compounds are: "student course evaluation", "government price support", "executive stock purchase", and "city garbage collection".

Another class of compounds occurs sometimes when a noun is used very much like an adjective. These nouns seem to be distinguished by overriding a default assumption about the noun it modifies. The noun "boy" in the compound "boy king" is an example. "Boy" does not fit well into any slot of "king" and thus no meaning would normally be found. However, "boy" is overriding the default assumption that a "king" is an adult. That is, "boy" is behaving very much like an adjective. The nouns that have this property have been named *pseudo-adjectives*. It is necessary to perform special searches to find if a default assumption is being overruled. Note that the method used to understand these kinds of compounds requires the data base to contain these default assumptions. Two other examples of such compounds are: "woman coal miner" and "toy gun".

The above patterns seem to cover a large proportion of noun compounds. As shown in chapter 6, some compounds encountered in English do not follow any of these patterns. However, the current system can be extended without too much difficulty to cover the patterns not covered above. In the original design of the system, fewer examples were examined than in chapter 6 and thus some of the patterns found there were missed since these new patterns occur fairly infrequently. There may still be other patterns that are missing. As can be seen from some of the examples, the classes of compounds described above are not disjoint. It is quite possible for a compound to be interpreted using two or more of the patterns.

With several different patterns, the possibility of multiple meanings for words, and the possibility of more than one relationship as the basis for the meaning of a compound, there is a good chance that there will be more than one possible meaning after all the possible interpretations have been found. At this point, a decision must be made as to which of the interpretations is the best one for the compound with the information available. This leads into the major part of thesis, i.e., the heuristics and knowledge necessary to make this decision.

1.3.3. Deciding on Which Interpretation is Best

The decision of which is the best interpretation for a compound is made by several heuristics. These heuristics attempt to use the kinds of knowledge people seem to use to make a similar decision. Some of the knowledge necessary to make this decision comes from information stored in the data base, i.e., information that has been stored in long term memory. Other information comes from the context in which the compound occurred. Note that many of the functions of these heuristics will be subsumed by the mechanisms in a complete natural language system. A list of the heuristics that are used to find the best interpretation follows.

- **Heuristics based on Context:** uses the context in which a compound occurs to disambiguate it. Context comes from the sentence in which the compound occurred as well as from the topic under discussion.
- **Cognate Heuristic:** Selects the interpretation for a compound that already has a meaning stored in the data base. There are three ways for a cognate to occur. The first is because it has an idiomatic or "conventionalized" meaning associated with a particular sequence of nouns. Another is there could be an interpretation stored in the data base without being associated with any sequence of nouns. The last is an interpretation stored in the data base by the program to represent the meaning of a previously seen compound.
- **Embedded Cognate Heuristic:** Selects the interpretation with the most embedded cognates. This heuristic is only effective for compounds containing more than two nouns.
- **Instances of an Interpretation Heuristic:** Selects the interpretation that has the most instances stored in the data base.
- **Embedded Instances of an Interpretation Heuristic:** Selects the interpretations with the most sub-compounds that have instances stored in the data base. This heuristic is only effective for compounds containing more than two nouns.
- **Slot Verification Heuristic:** Selects the interpretation whose filler of a slot is one of the concepts that best fills the slot.
- **Similar Compound Heuristic:** Selects the interpretation with the most similar interpretation stored in the network.
- **Occurrence Heuristic:** Selects the interpretation that has been seen most often during one session.

A somewhat more detailed description of the above heuristics follows. Chapter 4 contains a complete description of all the heuristics used to disambiguate a noun compound.

1.3.3.1. Context

The heuristics based on context use the context in which the compound occurred to determine the best interpretation from several possibilities. There are two ways that context can be helpful. The first, and the way that applies to other areas of natural language, is to disambiguate a word having multiple meanings. For heuristics based on context to be used, a system to understand noun compounds must be provided with extra information specifying the context in which the compound occurred. The "selectional restrictions" heuristic attempts to use the role that the compound plays in a sentence to disambiguate the meaning of the compound. For example, the word "club" in the compound "golf club" has at least two meanings that are relevant to understanding the meaning of the compound (i.e., a club as used in games, and as an organization that has a specific purpose). The meaning of the compound "golf club" can be determined in many cases by embedding the compound in a sentence. For example, the two sentences below provide sufficient context to allow a person to disambiguate the compound correctly.

1. The golf club hit the ball.
2. The golf club fired its manager.

The context from sentence (1) is presented to the system by providing a structure where "hit" is a verb with its slot for the object being hit filled with a particular ball. This will leave several slots available in this instance of a "hit" action that may be filled by "golf club". However, the meaning of "golf club" as a piece of equipment for playing golf can only fit into the slot for the thing used to do the hitting. The organization meaning of "golf club" cannot fill any of the slots of hit given the syntactic constraints of the sentence. Likewise, sentence (2) can provide context to get the organization meaning of "golf club".

The other way context becomes important is when there is more than one relationship that can lead to an interpretation for a compound. The "slot already filled" heuristic attempts to reduce the number of possible relationships by using the context surrounding a compound. For example, the compound "computer maintenance" has two relationships that are reasonable. One is where "computer" is the thing performing the maintenance, the other is where "computer" is the thing maintained. The expected interpretation is probably where "computer" is the thing maintained. However, consider the phrase "computer maintenance of a data base". In this case the relationship for the thing maintained should be taken up with "a data base". Thus, the only relationship that can exist is for "computer" to be the thing doing the maintenance. The only way this knowledge can be used is to provide external context with a structure specifying one or more relationships filled by some concept.

1.3.3.2. Cognates

Cognates are compounds that the system already knows about for some reason. There are built-in compounds (e.g., "chain gang" and "crown prince"), that have an idiomatic or "conventionalized" meaning. There may be compounds stored in the data base without a sequence of nouns associated with them. There may also be compounds recognized by the system and stored in its memory for later use. If the system encounters a compound having more than one interpretation, and only one of these interpretations has been seen before, then it is the most likely interpretation for the compound if no other contradictory information is available.

1.3.3.3. Embedded Cognates

Another time cognates are useful is when there is a compound with more than two nouns. Some of the sub-compounds within an interpretation for the compound can have cognates. If there is one compound with several internal cognates, it is likely to be a better interpretation than one having none or only a few. There is likely to be zero or one in most compounds, unless the compound is very long.

1.3.3.4. Instances of an Interpretation

Rather than having a concept that represents the compound, there may be an instance of the compound stored in memory. For example, assume the compound "coal truck" has two possible relationships that can be used to form a meaning. One is a truck that transports coal, and the other is a truck that is made of coal. Also, assume there is an instance (or maybe several) in memory where a truck was seen transporting coal. It is more likely that the interpretation where a truck transports coal is the correct one, rather than the one where a truck is made of coal. This heuristic depends on what knowledge is stored in memory.

1.3.3.5. Embedded Instances of an Interpretation

The "embedded instances of an interpretation" heuristic selects the interpretation whose sub-compounds have the most instances. This heuristic is a generalization of the "instances of an interpretation" heuristic and is based on the same premise. This heuristic is only effective when a compound contains more than two nouns.

1.3.3.6. Verifying Slots

The method used to find possible interpretations for a compound allows slots to be found which may not be the best (Chapter 3 gives the details of this method). It does not allow slots to be found for which an interpretation is clearly wrong, unless there is a major error in the knowledge used. Verifying the slot underlying a compound means the system makes sure the concept meant to fill a slot is in the set of concepts expected to fill the slot. An interpretation in which this is true should be better than one where the concept is not expected to fill the slot.

1.3.3.7. Similar Compounds

The similar compounds heuristic is based on the assumption that if there are compounds similar to a particular interpretation under consideration, such an interpretation is better than one that does not have this property. An interpretation of a compound is similar to another if the concepts representing the meaning of the head nouns of the two compounds are similar, the concepts filling the slots are similar, and the slots are similar. Two concepts are similar if they exist in the same type hierarchy in NETL. Two slots are similar if they have a common parent which will often be the defining occurrence of the slot.

For example, assume the compound "computer maintenance" has been interpreted using the two possible relationships between "computer" and "maintenance" (i.e., where "computer" does the maintenance, and where it is the thing maintained). There are likely to be several other compounds where things that are similar (in a fairly broad sense) to a computer are the things maintained. For example, assume such compounds as "car maintenance", "aircraft maintenance", "engine maintenance", etc. have been seen by the system and have been recognized as compounds where the first noun is the thing maintained. If no other information is available to the system (such as context), it is better to choose the meaning where computer is the thing maintained.

1.3.3.8. Number of Times an Interpretation has been Encountered

This heuristic keeps a count of the number of times a particular interpretation for a compound has been selected as the best interpretation. An interpretation that has been selected several times should be better than one selected only a few times.

1.3.3.9. Combining the Results of the Heuristics

The information gathered by the heuristics can be used in several ways. One is to have each heuristic act on its own. If only one interpretation is selected by the heuristic, it is chosen as the best one. This clearly does not allow the constraints that may occur between heuristics to come into play. One simple way to combine the results of the heuristics is to associate a weight with each heuristic according to how much effect it should have on the final decision of which interpretation is best. Some heuristics are likely to have a large weight and are likely to overbear many of the other heuristics. Also, some heuristics (e.g., context) should be sufficient on their own to rule out some interpretations completely.

The scheme used in the current system to combine the results of the heuristics is a mixed one. Some of the heuristics are allowed to rule interpretations out completely based on the current information. For example, the heuristics based on context are allowed to prune interpretations from the list of possibilities. If the context does not allow for a particular interpretation, then it is removed from consideration. Similarly, the "cognate" heuristic is able to prune the list of interpretations in most cases. The idea is to allow those heuristics that are very accurate in their assessment of interpretations to remove from consideration any that are rejected. This reduces the number of possibilities the less accurate heuristics must consider.

The less accurate heuristics can show a preference for a particular compound. If all these heuristics show a preference for one particular interpretation, this interpretation is probably better than any of the others. If no consensus exists, a weighted sum of the results of the heuristics is used to make a determination of which is the best interpretation. The "instances of an interpretation", the "slot verification", and the "similar compounds" heuristics are weighted relatively highly compared to the rest of the heuristics. The heuristics based on context can also provide a preference for a particular interpretation, in the same way as the others do, since context can support one interpretation more than another. Note that this assumes context supports more than one interpretation from the beginning, otherwise the one interpretation selected by context is selected. The heuristics based on context are weighted heavier than any other single heuristic. The "cognate" heuristic is a binary one, since either there is a cognate or there isn't for each interpretation. Thus, it only rules out possible interpretations as long as there is at least one cognate, and plays no further role.

1.3.4. Implementation

A program has been written in MacLisp to both find possible interpretation of noun compounds, and when necessary to apply the above heuristics to decide which is the best interpretation. The program interfaces to a simulator for NETL [Fahlman 79] which is also written in MacLisp. The program currently contains full and partial definitions for about two hundred words, most of which are nouns. It runs on a PDP 2060 and uses about 215K words of storage when it is started up. This includes about 12K words of compiled code for the NETL simulator, another 14K words of compiled code for my program, about 35K for the definitions of words and other knowledge, and about 70K for working storage. It takes two to five seconds of CPU time to generate interpretations for simple compounds, such as "wine glass". Ambiguous compounds, such as "golf club", take about five to fifteen seconds of CPU time depending on what context, if any, is provided to the system. Long compounds, such as "water meter cover adjustment screw", take about five to ten seconds of CPU time, as long as they only have one interpretation after all the words have been processed. Long compounds that are ambiguous can take over 30 seconds of CPU time.

1.4. Overview of the Thesis

The rest of the thesis is divided into seven chapters. Chapters 2, 3, and 4 describe the techniques used to understand noun compounds in more detail. These three chapters contain the technical details of how the program interprets noun compounds. Chapter 2 describes those parts of the NETL knowledge representation system used to represent the knowledge necessary to understand noun compounds. If you have encountered NETL's parallelism, type hierarchy, split statements, and roles and role mapping before, this chapter can be skipped. Chapter 3 describes the algorithms used to generate possible interpretations for a compound. It describes the problem in terms of compounds containing only two nouns, and then proceeds to the more general case of compounds containing more than two nouns. The end of the chapter describes the structures created in NETL's data base to represent the meaning of a compound. Chapter 4 describes all the heuristics that are used to determine the best meaning for a compound. It also describes how the results of the heuristics are combined to select the best interpretation for a compound. Appendix A shows the results of running the program on several examples.

Chapter 5 briefly describes how it might be possible to interface the program to a parser. It raises issues that are not dealt with in the current implementation because the program is not part of a fuller natural language understanding system. This chapter is of interest if you want to consider applying any of the ideas in the thesis to an existing natural language parser.

Chapter 6 describes an experiment in which the algorithms described in chapters 3 and 4 were hand simulated on over six hundred noun compounds from text occurring in real-world situations. This chapter analyzes where the current program breaks down and how it might be improved to process some of the compound it would currently process incorrectly. Appendix B contains a complete list of the compounds used in this experiment.

Chapter 7 discusses the results obtained by the program. These include the meanings created by the program, the effectiveness of the heuristics, and the execution times of the program for the current implementation as well as an estimate of the amount of time it would take on one possible implementation for a NETL machine.

Chapter 8 presents several conclusions from this research, as well as some directions for future research.

Chapter 2

The NETL Representation System

A system to understand noun compounds requires access to a large data base of real-world knowledge. This knowledge must include the meanings of words and other knowledge about concepts. It tends to stay relatively static over fairly long periods of time. It is also necessary to be able to represent knowledge that is more dynamic in nature, such as contextual information surrounding the occurrence of a noun compound. The underlying knowledge representation mechanism that the current system uses is the NETL system developed by Fahlman [Fahlman 79]. NETL provides all the facilities that this system needs to understand noun compounds.

NETL is a fairly recent semantic network scheme that addresses many of the problems that earlier semantic networks failed to tackle. There are two parts to the system proposed by Fahlman. The first part is an underlying parallel architecture that allows certain types of inferences to be performed very quickly. The second is a set of knowledge representation conventions that provide a clean structure to represent real-world knowledge. The two parts can be looked at independently, since it is not necessary to have a parallel architecture to use the knowledge representation conventions. The knowledge representation conventions can be used on a serial architecture. In fact, the system to understand noun compounds uses a NETL simulator that runs in MacLisp on PDP10's and 20's, since a NETL machine does not currently exist. The searches necessary to perform many inferences will run very slowly compared to the speed a parallel architecture can provide, since on a serial architecture, search time is proportional to the amount of knowledge present.

The parallel architecture assumes that each node or link in the semantic network is a very simple hardware device, i.e., it can do some amount of simple processing. These devices are able to propagate a variety of *marker bits* from one node to others. As well as a few bits of memory to hold these marker bits, each device has a few bits of memory used to specify the type of the node or link that the device represents. Each link has wires to nodes that allow it to propagate marker bits from one node to other ones attached to the link. This is the only thing that the parallel architecture

provides, i.e., it has the ability for nodes or links to be marked by marker bits, and for marker bits to be propagated to surrounding nodes via links according to which marker bits are on or off. Hanging off this parallel architecture is a serial processor to issue commands to the processing elements representing nodes and links. This serial processor is the one that decides what marker propagation searches are to be performed, and what to do with the results once these searches finish. Thus, the serial processor is the guiding intelligence behind the parallel architecture. The parallel architecture just allows certain types of searches to be performed in parallel, and thus very quickly.

Marker propagation in NETL is controlled by a network controller that can issue commands through a bus that is common to all the nodes and links in the network, so that a command can be sent to all the nodes and links at the same time. The major operation of this parallel network is to propagate marker bits through the network according to some conditions specified by the network controller. For example, one of the most common operations is for a link to mark the node at its head with a marker according to the markers that are on the node at its tail (or vice versa, since it is often necessary to traverse links backwards). This is done by the network controller activating all the devices representing a particular type of link at one time with a command of the form: if the node at your tail is marked with marker M1, then mark the node at your head with marker M2 (M2 may be the same as M1, if the desire is to propagate this one marker through a chain of links). Note that this takes only one cycle to have a marker propagation go through one level of links, since all the links can perform the appropriate operation in parallel. The conditions can be more complex by specifying a set of marker bits that must be on, and another set that must be off. This allows various types of searches to be specified within NETL. These simple rules seem to be sufficient for specifying searches that people are very fast at, e.g., intersection searches or property inheritance searches.

The knowledge representation conventions of Fahlman's system define the structures used to represent the knowledge stored in the parallel network. The next few sections describe in more detail the structures needed to understand noun compounds. This is not a complete description of NETL's knowledge representation conventions, but only a description of those parts used by the system to understand noun compounds. For a more detailed description of NETL, see [Fahlman 79]. The last two sections (sections 2.5 and 2.6) describe structures developed for this particular application of NETL.

2.1. Type Hierarchies

The basic knowledge representation structure that NETL relies on is the type hierarchy. This type hierarchy forms the backbone of the knowledge used by the system to determine the meaning of a compound. A type hierarchy is a hierarchy of concepts: the higher the concept in the hierarchy, the more general it is. A type hierarchy in NETL is very similar to the IS-A hierarchy of other semantic network representation systems. In NETL there is one distinguished node (the "thing" node) that is the highest node in the hierarchy. This is the only node in the hierarchy that does not have a superior. All other nodes in the hierarchy must have a superior. There may be no cycles in the hierarchy, i.e., it is not legal to form a loop in the hierarchy. These conditions mean that every concept in the hierarchy must have the "thing" node as an ancestor, and that no node may have itself as an ancestor. The hierarchy may be tangled which means that a node may have more than one parent node. For example, figure 2-1 shows a hierarchy that is not tangled from the point of view of the interior node X. Figure 2-2 shows one that is tangled from the point of view of the interior node X. A tangled hierarchy allows a node to inherit attributes from more than its immediate superior node or nodes. This tangled hierarchy forms the back bone of the semantic network.

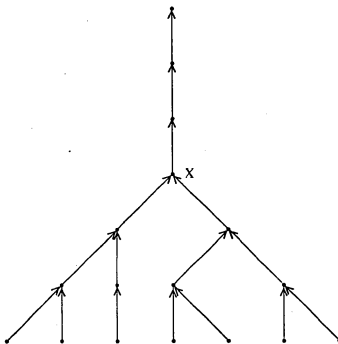


Figure 2-1: A non-tangled-hierarchy as viewed from the interior node X.

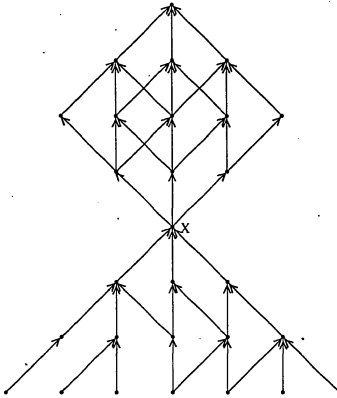


Figure 2-2: A tangled hierarchy as viewed from the interior node X.

Figure 2-3 shows a description of the concept "elephant", and some of its ancestors on the path to the "thing" node at the top of the hierarchy. There is a *virtual copy (VC)* link running from the node labelled "Clyde" to the node labelled "elephant". A VC link is similar to an IS-A link as used in many other semantic network representation systems. A VC link allows an inferior node to inherit information from its superiors much in the same way that an IS-A link does. It just makes explicit that the effect is one similar to making a copy of all the inherited information, even though it is not normally necessary to make a copy. Similarly, there is a VC link running from the node labelled "elephant" to the node labelled "mammal". A VC link means that the node at the tail of the link acts as if the complete description provided by the node at the head of the VC link (including all the descriptions above it in the hierarchy) is copied down to it. That is, it acts as if a copy of all the structure necessary to fully describe the meaning of the node at the tail of a VC link has been made. To actually make such a copy would be extremely expensive in terms of space, and also in terms of updating the information when information changed at higher levels in the hierarchy. Thus, the VC link provides a cheap way of specifying that all the information exists for a particular node without

actually copying it. The section on roles (section 2.3) describes how properties of a concept can be attached to the node representing the meaning of the concept. It also describes how to make copies of this information at lower levels of the hierarchy when it is necessary to represent additional information about a property at the lower levels. The section on statements (section 2.4) describes how statements can be made about a concept.

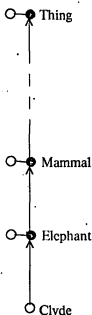


Figure 2-3: A simple type hierarchy with Clyde (who is an elephant) represented as an individual node.

The node labelled "Clyde" in figure 2-3 is an *individual* node, and represents the particular elephant named "Clyde". An individual node represents a particular individual and is the centre of all the information that pertains to that one individual. An individual node should almost always have a VC link from it to a type node. In particular, an individual node should not have a VC link to another individual node. An individual node under normal circumstances will not have any descendants. The solid black node labelled "elephant" in figure 2-3 is a *type* node and represents the typical elephant. A type node represents the information about a concept that is typical for the class. Thus the node marked "elephant" in figure 2-3 represents the typical elephant and is the base-node for the description of a typical elephant. The node off to the left of this node and joined to it by a short horizontal line is an individual node, and represents the set consisting of all elephants. Every type node in the NELL hierarchy of concepts has a set (represented by an individual node) associated

with it. Thus, the system can talk about the typical elephant and make statements about it, as well as talk about the set of all elephants and make statements about this set. The individual node "Clyde" is assumed to be a member of this set implicitly, because of the VC link from it to the node "elephant", i.e., NETL does not require that the node "Clyde" be explicitly a member of the set of all elephants.

2.2. Split Statements

In a knowledge representation system, there is often a need to specify that the sets of objects described by two concepts have no elements in common. In particular, it is often necessary to make sure an object in one set can not be placed into the second one. In NETL, a concept can be placed into two sets by creating a VC link to the type nodes describing the typical element of each set. For example, the node representing the meaning of "elephant" will probably have a VC link or chain of VC links pointing to the concept "mammal". Assume Clyde is an individual elephant, i.e., there is an individual node representing "Clyde" whose parent is the elephant type node. Assume the VC link from "Clyde" to "elephant" already exists, and the system wants to create a VC link from the individual node "Clyde" to the type node representing the class of all "male" things, i.e., assume the system has just found out that "Clyde" is male and wants to add this information to the data base. In this example, the VC link from "Clyde" to "male" should be created without any interference. What happens if for some reason the node representing "Clyde" who is an "elephant" is to be made VC of the node representing "cabbage"? Most people would almost immediately wonder what is happening, because people "know" that something can not be an elephant and a cabbage at the same time. How people know this is not clear. They are unlikely to store the information directly, so there must be a fairly fast inference mechanism that allows them to arrive at this conclusion. The split statement allows NETL to make this inference quickly. In NETL terminology, a *clash* occurs when the statement "Clyde is a cabbage" is added to the data base.

NETL can detect these types of clashes by using *split statements*. Figure 2-4 shows a portion of a NETL network where a split statement is used to say that the two concepts "animal" and "plant", which are both living physical objects (*physobs*), are distinct classes, and that an element of one class should not be an element of the other one. The circle with an "X" in it represents this split statement. The two links going from it to "plant" and "animal" are *split links* and specify that animal and plant are members of the split set defined by the split statement. In the example above, where a VC link is proposed from "Clyde" to "cabbage", this split statement would normally complain that an elephant cannot be a cabbage. Before putting in the new VC link, NETL searches the network to see if any split statements will complain. The search proceeds as follows:

1. Mark upward from Clyde with the marker M1, as shown in figure 2-4.
2. Mark upward from the node that Clyde is to be made VC of (i.e., "cabbage") with marker M2, as shown in figure 2-5.
3. Finally, if there is any split statement that has one split link pointing at a node with marker M1 on and marker M2 off and another split link pointing at a node with marker M2 on and marker M1 off, then this split statement would be violated if the proposed link is put in. In the example, the split statement between "animal" and "plant" detects a clash during the check to see if a VC link can be created between Clyde and "cabbage" (see figure 2-5). Note that all split statements can check for this in parallel.

Thus, NETL would detect a clash, and could ask the user what should be done. The user could tell NETL to create the VC link anyway and to ignore the conflict in future, or he could tell the system that he made a mistake and not to put in the VC link from "Clyde" to "cabbage". The second alternative is the most likely one, since most of the time, you do not want to violate a split statement. The searches that use split statements in this way are similar to the match-on-superordinate process described by Collins [Collins 75].

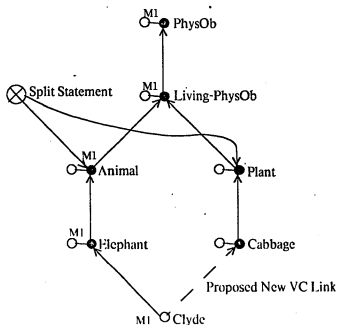


Figure 2-4: Representation of a Split Statement with the ancestors of the node Clyde marked with marker M1.

Split statements are one very important aspect of NETL as far as the system to understand noun compounds is concerned. The system wants to find slots that can contain concepts. The user can

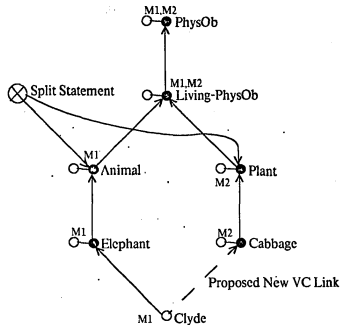


Figure 2-5: NETL detects a clash between Clyde and cabbage after the ancestors of "cabbage" are marked with marker M2.

restrict the filler of a slot to a certain class, as shown in the next section. Split statements prevent conflicting classes from filling a slot without NETL detecting a clash. Thus, split statements along with other mechanisms in NETL provide a simple way to check whether a concept can be the filler of a slot. For example, the word "analysis" has at least two slots (i.e., the thing analyzed, and the thing doing the analysis") that are available to form the meaning of any compound in which "analysis" is the head noun. For the compound "suffix analysis" the system will find both slots but immediately rule out the slot for the thing doing the analysis because the slot restrictions on this slot will not allow "suffix" to fill it. The restrictions on a slot are enforced by searching for split statements that would cause a clash if a concept is placed into a slot.

2.3. Roles and Mapping

So far, only the type hierarchy and the method to split classes of concepts into non-intersecting sets has been discussed. To be able to describe the properties of a concept, it is necessary to add more structure to the representation. The type nodes introduced earlier serve as the base-node for the description of a concept. To flesh out the description of a concept, NETL uses *roles*. In NETL, a description of a concept consists of a single base-node that represents the concept, and a set of role

nodes, connected to this base-node by what are called *existence links*. Roles represent the various things that every copy of the base-node has. That is, for each descendant of a base-node, there exists one copy (either explicit or implicit) of each role. This corresponds to a logic statement of the form: for all x such that x is a virtual copy of the base-node, there exists y such that y is a role. For example, for every x such that x is an elephant, there exists y such that y is the trunk of x . A role node is an individual node that represents a single entity or a set. For example, the left front leg of an elephant is a single entity that can be represented as a role node attached to the base-node representing the typical elephant (actually this role is likely to be defined higher in the hierarchy). Similarly, the set of four legs of an elephant is a set that can be represented as a role node. An existence link is run from the role node to the base-node of the description of which the role node is part. A role node may function as a base-node for its own description, with properties and statements linking it to other nodes. For example, the left front leg of a typical elephant would be represented by an individual node that is VC of the typical elephant leg and thus inherits information from this node. Similarly, the set of four elephant legs of a typical elephant would be represented by an individual node that is VC of the typical set. Attached to this node normally would be the information that it has four elements, and possibly even the four elements.

A role may be either an *in-role* or an *of-role*. An in-role specifies that the role exists within the area defined by the owner node (e.g., the engine of a car would be represented by an in-role, since it is within the area defined by car). An of-role specifies that somewhere there exists a map of the role that corresponds to each copy of the owner node (e.g., the weight of a car would be represented by an of-role, since it is not within the area defined by car). Note that an in-role makes a stronger statement about the relationship between the owner and the role than an of-role does. Figure 2-6 shows two roles for the typical "elephant". A typical "elephant" has a "heart" in-role. This role is represented by an individual node that is attached to the typical "elephant" node by an *exists in* (*exin*) link. A typical "elephant" also has a "weight" of-role that is represented by an individual node attached to the typical "elephant" node by an *exists for* (*exfor*) link. The two existence links are drawn with dashed lines. The stronger *exin* link is drawn with a solid arrowhead, the weaker *exfor* link is drawn with an open arrowhead.

A node that is lower in the hierarchy of concepts is able to inherit all the roles of its ancestors. Thus, Clyde has a heart and a weight by virtue of being a member of the class of elephants. For this inheritance to work, it is not necessary to make a copy of all the roles higher in the network. Instead, the VC links can be traversed upwards to find all the roles that a concept can inherit. However, occasions will arise where something must be said about a role of a concept that is a descendant of the

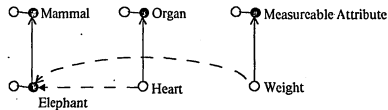


Figure 2-6: Definitions for the "heart" in-role and the "weight" of-role for the typical "elephant".

owner of the role. The defining occurrence of the role can not be used in the statement, since then it would apply to all the concepts that can inherit the role. NETL solves this problem by using a *map node* and *map link* to map a role down to a concept that wants to say something about the role. The statement can then be attached to the map node, and it will affect only the owner of this map node, and any of its descendants, rather than all the concepts that inherit this particular role. Note that this mapping process does not have to occur until the driving system wants to represent additional information about a particular role.

For example, figure 2-7 shows the "nose" role of "mammal" mapped down to "elephant". The node representing the map of the "nose" role is named the "trunk" role for elephant. The arrow drawn with two lines with an arrowhead only at the top represents a map link. The node that looks like an individual node at the tail of the map link is actually a map node. The link running horizontally from the "trunk" map node to the "elephant" type node is an *owner link* and specifies that the "elephant" type node is the owner of the map node. No distinction is made for owner links according to whether the defining role is *exin* or *exfor*. This information can be determined by looking at the defining occurrence if necessary. The map node "trunk" maps the role pointed at by the map link down for the typical "elephant". The "trunk" role can then be mapped down to an instance of an elephant, as for "Clyde" in figure 2-7. The system can now make statements about a typical elephant's trunk, or Clyde's trunk without affecting the nose role of the typical "mammal". For example, figure 2-8 shows the representation of the statement "an elephant's trunk is a cylinder, with a typical length of 1.3 metres". If nothing is said about "Clyde's trunk", then the system can use the length of 1.3 metres as an estimate of how long "Clyde's trunk" is. Note that this statement says nothing about what a typical mammal's nose looks like, or how long it is. In figure 2-8, the two way double arrow is an *EQ link* and means that the "trunk length" node and the "1.3 metre" node are equal to one another. This just means that any time the system looks at one, it will find the other. This is the method NETL uses to represent the value assigned to a role.

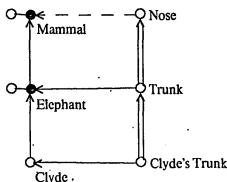


Figure 2-7: Mapping the nose role of mammal down to elephant and then down to Clyde.

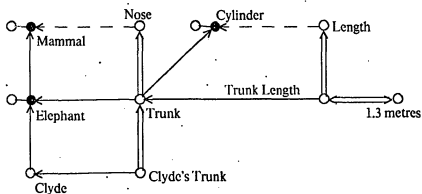


Figure 2-8: The typical elephant's trunk is a cylinder of length 1.3 metres.

2.4. Statements

Another part of NFTL that the system to understand noun compounds uses is the conventions used to represent statements. Statements are used in the representation for nouns derived from verbs (see section 2.6 for details). Statements in NFTL are used to say things about the concepts that nodes in the network represent. Some statements are represented by links, such as VC links, or HQ links. Other statements are represented by an *individual statement* node or *IST* node. An IST node is very similar to an individual node, in that it is used as the base-node for the structure describing the concepts that the statement says things about, as well as anything that can be said about the statement itself. A statement usually has in-roles that represent information about the concepts that can take part in the statement. A statement may also have of-roles representing information about the

statement itself. Statements exist in a hierarchy much the same as other concepts, and can inherit information from nodes higher in the hierarchy. This representation of statements is similar to the case-frame relations of Fillmore [Fillmore 68], although the implementation is quite different.

Figure 2-9 shows the definition of the "hates" statement with roles for the thing doing the hating ("hater"), and for the thing hated ("hatee"). Figure 2-10 shows an individual statement that says "Clyde hates snakes". The node labelled "Clyde hates snakes" is the IST node that represents the statement itself. The "hater" role is mapped down to the individual statement and set equal to "Clyde" with an EQ link. Similarly, the role for "hatee" (i.e., the thing that is hated) is mapped down to the individual statement, and set equal to the node representing a typical snake. Since the "hatee" role is mapped down and set equal to the type node for "snake", this is a statement about the typical snake, and thus all snakes will inherit it. The IST node that represents a statement is just another node, so it is possible to make statements about a statement.

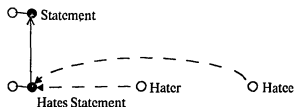


Figure 2-9: Definition of the hates statement, with in-roles for the hater and the thing hated.

2.5. Representation of a Word

The NETL simulator developed by Fahlman for his thesis research [Fahlman 79] only has a one to one mapping between a word and the node representing its meaning. The current research is interested in dealing with words with multiple meanings, and thus needs a many to many mapping between words and the nodes that represent their meanings. A simple scheme has been developed so that this many to many mapping between words and the nodes representing their meanings can be represented in NETL.

A word can be associated with many different concepts and also a few parts of speech. For example, the word plant can be either a noun or a verb. As a noun, plant, has at least two meanings (e.g., a plant as you find in gardens, and plants as factories in manufacturing operations). As a verb, it

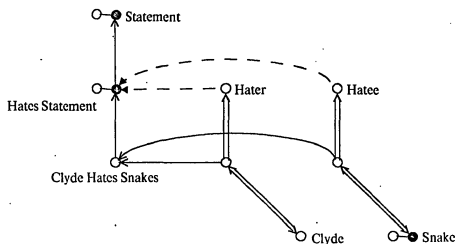


Figure 2-10: Creating an individual statement of the hates statement to represent "Clyde hates snakes".

has at least the meaning of placing a plant (as in a garden) into the ground. Thus, a word can be mapped to various concepts in the NETL hierarchy. Similarly, a concept in the NETL hierarchy can be mapped to more than one word. Any natural language understanding system must be able to cope with multiple meanings of words in a flexible way. In the current system, structures are created in NETL that allow a word to be associated with all the possible meanings for the word. The link between a word and a concept has associated with it the part of speech of the word when it is used to mean this particular concept.

Figure 2-11 shows how the various meanings of plant are associated with the word "plant". The word "plant" is represented as an individual node that is a member of the class of concepts defined by the meaning of the word "word". The sinusoidal lines are called *type-name* links, and state that the concept at the tail of such a link can be mapped to the word at the head of the link. These links may have statements made about them. This is the method used to represent the part of speech that a word is used as when it means the associated concept at the tail of the type-name link. The statements are tied to the larger circle around the middle of the type-name link, as shown in figure 2-11. In figure 2-11, a VC link runs from this circle to either of the nodes "*verb" or "*noun". The nodes "*verb" and "*noun" represent sub-types of the more general type-name statement, and are used to represent the part of speech that the word is used as when it means a particular concept. For example, when the word "plant" means the thing that is a living-physob, it is being use as a noun, and the type-name link from the concept "&plant-B" to the word "plant" reflects this by being VC of the

"*noun" sub-type. This is not the only way that this information could be stored in NETL's data base, it is just a convenient and flexible way to do it. NETL's role mechanism is powerful enough to allow this information to be represented. However, a large amount of structure would have to be built to do this properly. To distinguish between words and the nodes that represent their meanings, all nodes representing the meaning of a word are labelled by the word preceded by an ampersand (&). If there are several concepts associated with a word, they are distinguished by appending "-letter", where letter starts at "A", and proceeds alphabetically through the alphabet for each new meaning.

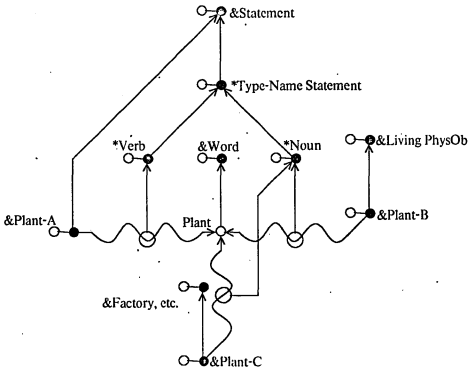


Figure 2-11: Representation of the word plant and associated concepts.

2.6. Representation of Nominalized Verbs

There is one set of nouns, i.e., nominalized verbs, that need more structure than provided by the normal type hierarchy with roles described earlier. For example, the noun "evaluation" is derived from the verb "evaluate", one meaning of the noun "assembly" is derived from the verb "assemble". These nouns are generally used to refer to actions and their slots are generally similar to those for the

corresponding verb. Figure 2-12 shows the NETL structure for the nominalized verb "evaluation". The slots in the description of the verb "evaluate" are connected to the slots for the noun "evaluation" by creating an instantiation of the evaluate statement (&evaluate-1) that has its slots filled by values equal to the slots in the definition of the noun "evaluation". This structure looks more complex than some other possible ones. The main reason this structure was chosen is so that the roles of the noun can be mapped down, and still retain the relationship with the roles of the verb without the need to make a copy of the statement and all of its roles for each map node.

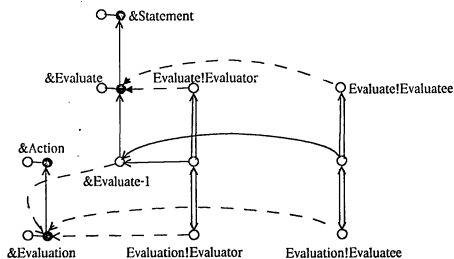


Figure 2-12: Representation of the nominalized verb "evaluation".

Chapter 3

The Possible Interpretations for a Compound

The system uses two phases to find the meaning of a noun compound. The first phase finds all of the possible interpretations for a compound. These may include interpretations a person would not normally consider. The second phase takes these possible interpretations and selects those interpretations (in most cases only one) most likely to be the best ones with the knowledge available to the system. This second phase is only activated when more than one interpretation for a compound is found during the first phase. When only one interpretation is found, it is considered as the best one and is used as the meaning for the compound. The division of the system into two phases was done so that when only one interpretation is found the second phase can be ignored. More constraint could be obtained in the first phase by moving some or all of the heuristics in the second phase back into the first phase. For example, the heuristics based on context could be used to provide constraint during the first phase. However, some of the contextual information may occur after a compound in a sentence. The control structure presented here is not meant to be a psychological model of how people might process noun compounds. This chapter presents the part of the system that finds the possible interpretations for a compound; the next chapter discusses how the best interpretation is selected.

The process of finding possible interpretations for a compound involves extracting information from the NETL data base. The algorithm to find the possible interpretations for a compound uses the parallelism assumed by NETL to good advantage. This does not mean that the algorithm is only useful for a parallel architecture like NETL. It can be adapted to serial architectures, as well, although with a large knowledge base it will run much more slowly than it would on a parallel architecture under the NETL paradigm.

The first section of this chapter looks at the problem of finding an interpretation for a compound containing only two nouns. The second section gives a description of how the system processes compounds containing more than two nouns. The third section presents the algorithm used to find

the possible interpretations for a compound and an example of its uses. Finally, the last section describes the results of finding the possible interpretations of a compound.

3.1. The Two Noun Problem

This section looks at the simpler problem of determining the meaning of a compound containing only two nouns of the form " $N_1 N_2$ ". The basic assumption made by the system is that either " $N_1 N_2$ " is idiomatic or there exists at least one relationship between the two nouns forming a compound. Another important assumption is that the *head noun* of the compound (N_2 in this case) is being modified by the previous one in some way. Thus, a compound of the form " $N_1 N_2$ " is an N_2 that is being modified in some way by N_1 . The problem is to find all the possible relationships that can exist between the two nouns in the compound. Note that in some cases there may be no possible interpretation, in which case the system reports this fact. A further assumption the system currently makes is that the relationships that can exist between two nouns is provided by the noun N_1 filling a slot in N_2 . That is, the head noun of the compound provides the relationship (via a slot), and the modifying noun is used to fill this slot. For example, consider the compound "car maintenance". The node representing the meaning of the word "maintenance" has a slot for the thing maintained (among others). A meaning for "car maintenance" can be formed by placing "car" into this slot, i.e., "car maintenance" is a "maintenance" action where a car is the thing maintained. Thus, to find all the possible relationships between two nouns, it is necessary to search the data base for all the slots of N_2 into which N_1 can be placed.

The simplest way to perform the search is to find all the possible slots that N_2 has, and try placing N_1 into them one at a time. If N_1 can fill the slot (i.e., NETL does not complain because a split statement says N_1 is something that can-not fill the slot), then this is a possible interpretation. If N_1 can not fill the slot, then this is not a possible interpretation. For example, consider the compound "car maintenance" and the associated structure about the two nodes "&car" and "&maintenance" that represent the meaning of the two nouns "car" and "maintenance" respectively, as in Figure 3-1. There are the two slots "maintenance!maintainer" and "maintenance!maintainee" of the node "&maintenance" representing the meaning of the word "maintenance" into which the node "&car" representing the meaning of "car" could be placed. However, if "&car" is placed into the slot "maintenance!maintainer", the split statement complains, because "&car" is a "&non-living-physob", rather than a "&living-physob". When "&car" is placed into the slot "maintenance!maintainee", NETL finds no split statements that complain that it can not fill the slot, and so this is a possible interpretation. This is using the idea of selectional restrictions [Katz 63] provided only by the context of the compound itself.

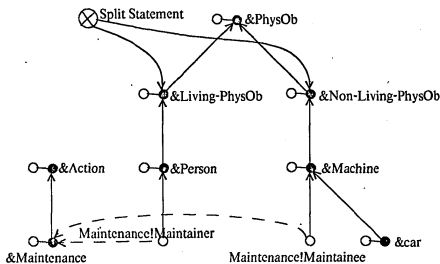


Figure 3-1: Simplified NETL Structure Necessary to Understand the Compound "Car Maintenance"

The above is a fairly simple view of the processing necessary to find possible interpretations for a compound. It assumes that none of the nouns have multiple meanings. Of course, this may not be the case, because words in English can have multiple meanings both in terms of parts of speech (e.g. "plant" has at least four meanings as a noun, and at least three as a verb) and also multiple meanings with the same parts of speech (e.g. "glass" as a container from which people drink liquids, and as the material from which such objects among others are made). It is possible to have a compound that has multiple meanings for either or both of the nouns in the compound (e.g., "plot outline" and "glass glass"). If the second noun of the compound has multiple meanings, each of the meanings must have their slots checked to see if the first noun can fill them. If the first noun has multiple meanings, each of these meanings must be checked to see which of the slots of the second compound they can fill. This can very quickly lead to a large number of tests to find the possible interpretations for a compound.

The discussion above has made no assumptions about what kind of representation scheme is being used to represent the meanings of concepts. Using the parallel nature of NETL, some of the work described above can be done much more efficiently. The following sub-section describes the searches performed in the NETL network to make the discovery of possible interpretations more efficient. Unfortunately, using NETL does not directly address the problems introduced by multiple meanings of words. It is still necessary to perform a search for each possible meaning of a word

individually, since if multiple meanings were all looked at in parallel, some very bizarre effects would occur (i.e., the search would be likely to confuse the meanings, and assume information not contained in the network).

3.1.1. Investigating One Meaning of a Noun

The discussion above described a serial method for finding all the possible interpretations (i.e., by placing the concept representing the meaning of the first compound in all the slots of the second). A similar effect can be achieved by using the parallel marker passing abilities of NETL. First, it is necessary to define the following two operations:

1. All the slots directly or indirectly attached to a node representing a meaning of a noun, are marked by a particular marker. The time for this search is proportional to the length of the longest VC chain in the network.
2. All the slots into which a node can **not** be placed are marked by a particular marker. Again, the time for this search is proportional to the length of the longest VC chain in the network.

The search to mark all the slots of a meaning of a noun is fairly straight forward. For example, assume the system needs to find all the slots of the concept "&glass-b", one of the meanings of glass. Figure 3-2 shows the structure of the network around this node, as well as, a few of the likely slots to be associated with it. The search to find all the slots of "&glass-B" proceeds as follows:

1. An upward scan from the node "&glass-b" is performed. An *upward scan* from a node marks that node and all of its ancestors in the type hierarchy. This marks all the nodes from which "&glass-b" could possibly inherit any slots with the marker M1. Figure 3-2 shows the concept "&glass-b" and all of its ancestors marked with the marker M1.
2. Next, it is necessary to mark backwards across the links that point from the slots to nodes marked by marker M1 with marker M2. Figure 3-3 shows the status of the network after this scan has been done.

At this point, the marker M2 marks all the possible slots associated with the concept "&glass-b".

The search to mark all the slots into which a concept can not fit is somewhat more complex. As noted above, in the simple approach to finding possible interpretations (i.e., placing the concept representing the meaning of the first noun into each of the slots of the second), the split statements existing in the network are how the system discovers whether a slot can be filled by the proposed concept. It is these same split statements that play a large role in the search to mark those slots that can not be filled by a concept. The search to mark these slots proceeds as follows:

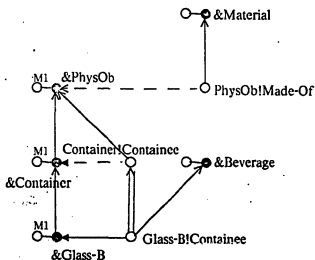


Figure 3-2: First Step of the Search to Find the Possible Slots for One Meaning of "Glass"

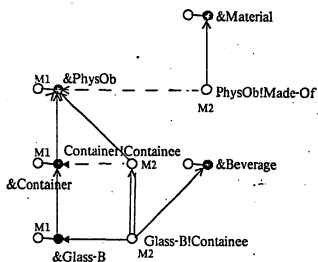


Figure 3-3: Marking the Slots of One Meaning of the Word "Glass"

1. Perform an upward scan from the concept marking it and all of its ancestors with marker M1. This is the same as the first step of the search above (and in fact the two searches are combined to the extent that they both use the one upward scan). Figure 3-4 shows the concept "&car" with all of its ancestors marked by marker M1.
2. Mark across all the split links that come into nodes marked by marker M1 with marker M2. This step marks all the split statements that could become active during a search if

the concept is placed into a slot. Figure 3-5 shows the status of the network around the concept "&car" after all the split statements having a split link pointing at a node marked with M1, are marked with marker M2. This marks any split statements that may be involved if the concept "&car" is placed into an arbitrary slot.

3. Next, all the split statements marked with marker M2 are processed. All the nodes not marked with marker M1, and pointed at by a split statement marked by M2, are marked with marker M3. Figure 3-6 shows the status of the network around the concept "&car" after this step is completed.
4. Finally, a downward scan from all the nodes marked by M3 is performed with marker M3. A *downward scan* marks a node and all of its descendants with a particular marker. It is possible to make this mass downward activation, since the purpose of the search is to mark all the concepts that could conflict with the concept "&car". Figure 3-7 shows the status of the network around the concept "&car" after this step of the search is complete.

At this point, the marker M3 marks all the nodes that definitely conflict with the original concept. The nodes marked include all the slots that the concept can not fill. For example, the slot "&maintenance!maintainer" in figure 3-7 has been marked with the marker M3. This means that if the node "&car" were to be placed into this slot, NETL would complain, because it violates the split statement that says a maintainer is a "living physob" and can not be a "non-living physob" which is what a car is.

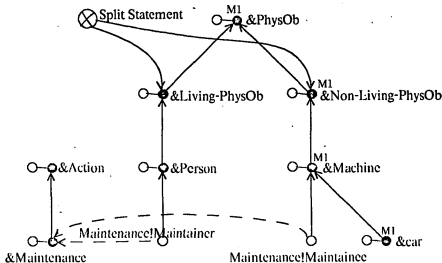


Figure 3-4: Mark all the Ancestors of "&Car" to Find the Slots that "&Car" can not Fill

Note that this algorithm takes one upward scan, two steps across split links, and a downward scan.

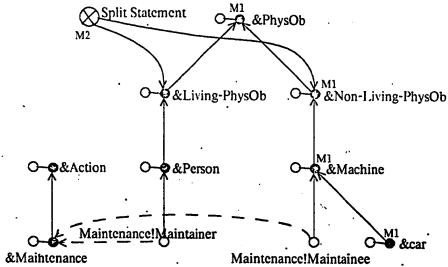


Figure 3-5: Mark all the Split Statements that Point at a Node Marked During the First Step

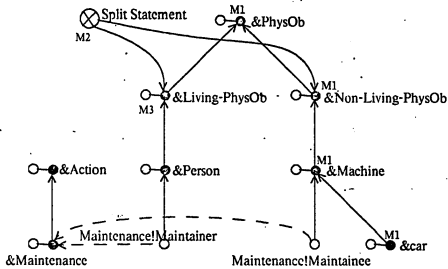


Figure 3-6: Mark all the Nodes Pointed at by the Split Statement, Except Those Already Marked by M1

These scans will probably mark more nodes than a single scan that checks if a concept can fit into a slot, but if a parallel architecture is assumed, as in NETL, the actual time spent will be on the order of two tests using the simple method described above. Thus, if more than two slots are involved, it is likely to be cheaper, at least in terms of time, to use this more complex search. Most concepts are

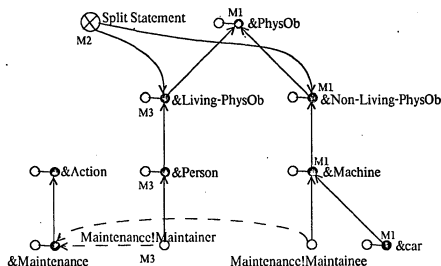


Figure 3-7: Perform a Downward Scan from all Nodes Marked with M3

likely to have more than two slots associated with them, because they can inherit slots from all the nodes that are above them in the NETL hierarchy.

Once these two searches have been performed, there is sufficient information for the system to find all the possible interpretations for a compound of two nouns. The next sub-section describes how all the possible interpretations for a compound are found using the information provided by the two activation scans described here.

3.1.2. Finding Candidate Meanings

The previous sub-section presented the main activation scans for concepts representing nouns. This sub-section shows how this information is used to form possible interpretations for a compound. Assume for the time being that there is only one meaning for each of the two nouns forming the compound. The system wants to place the concept representing the meaning of the first noun into a slot of the second. The marking process described above provides all the information necessary to find the slots of the second noun into which the first one can be placed. For a compound of the form " $N_1 N_2$ ", let M1 be the marker that marks all the concepts (including slots of N_2) that conflict with the meaning of N_1 , and S2 be the marker that marks all the slots of the meaning of N_2 . Note that this assumes N_1 and N_2 have only one meaning as nouns. For each slot marked with marker S2 (i.e., is a slot of the meaning of N_2), the system checks to see if the slot is marked with marker M1 (i.e., N_1

conflicts with the concepts placing restrictions on the slot). If it is, then the meaning of N_1 can not normally fill the slot, and the slot is rejected as the basis for a possible interpretation. If it is not marked with marker M1, the meaning of N_1 can fill the slot, and the slot provides the basis for an interpretation.

For example, consider the compound "car maintenance". Figure 3-8 shows a partial network around the concepts necessary to process the compound "car maintenance". Marker M1 marks all the concepts that are definitely not a "car", and marker S1 marks all the slots of the concept representing the meaning of "car". Similarly, there are corresponding markers M2 and S2 for the node representing the meaning of "maintenance". The markers M2 and S1 are not used in forming the meaning of the compound and neither are shown in figure 3-8. The slot "maintenance!maintainer" of the concept representing the meaning of "maintenance" is marked by both the S2 (i.e., is a slot of the concept representing the meaning of "maintenance") and M1 (i.e., the concept representing the meaning of "car" should not be put into this slot). This means that the slot "maintenance!maintainer" should not be used to interpret the meaning of the compound "car maintenance". However, the slot "maintenance!maintainee" only has the marker S2 on it, and so it can be used as a possible interpretation of the compound. Once the searches described above have been made, this is the only test needed to find a possible interpretation for the compound. Notice that the use of the NETL split statement is vital for this search to produce appropriate interpretations. If the split statement did not exist, two interpretations would be formed. Some of the heuristics can help select the better meaning when this occurs.

To find the interpretations for a compound containing only two nouns is relatively simple, as shown above. Note that both nouns are activated by marking their slots and also marking all the slots that they can't fill. This was done mainly for generality, in case an interpretation was ever formed by placing the second noun in a slot of the first. As shown in chapter 6 there are a few compounds whose meanings are formed by placing the second noun in a slot of the first one. When this happens, the first noun still modifies the second one. The second noun specifies the type of thing being referred to. The first noun makes this reference more specific by modifying the second noun. The searches performed in the network only find the possible interpretations, they do not decide which of the interpretations are best. That is left to the heuristics which are described in chapter 4.

The two noun case is also fairly simple when words have multiple meanings. The only difference, is that if the second noun has multiple meanings, the slots of all the different meanings must be checked one at a time, as described above. If the first noun has multiple meanings, then each of the

them, as characterized by a slot. There is one other form of compound that occurs, as in the compound "woman miner". There is unlikely to be any slot of the meaning of "miner" that can be filled by the meaning for "woman". What the compound seems to be talking about is a miner who also happens to be a woman. The distinguishing feature of such compounds seems to be that the first noun overrides a default type for the second noun. In the case of "miner", the default is that a miner is a man. Woman and man will be split by a split statement saying someone who is a woman can not also be a man. When a default assumption is overridden this way, the two concepts are in contrast with one another. NETL's split statement seems to be the best way to represent this contrast. The woman in "woman miner" is overriding the default assumption that a "miner" is a man. Nouns such as "woman" have been named *pseudo-adjectives* since the nouns in these cases behave very much like adjectives. Figure 3-9 shows the structure used to represent "miner" and "woman". Note that the node "&miner" has a VC link from it to "&person" and a default VC link from "&miner" to "&man". Other examples of such compounds include "boy king" and "girl queen".

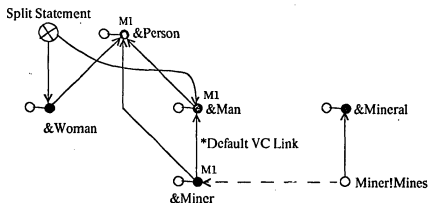


Figure 3-9: Find Interpretations for the Compound "Woman Miner" Using Pseudo-Adjectives

The algorithm necessary to process the compound " $N_1 N_2$ " where the first noun is a pseudo-adjective follows. The compound "woman miner" is used as an example to show how the various steps of the algorithm work.

1. Mark all default VC links as being inactive. Check to see if N_2 can be an N_1 . The following possibilities exist for this check:

- If N_2 is not an N_1 , then do nothing further since N_1 can not modify N_2 .
- If N_2 is an N_1 , there is no need to do any further searching. An interpretation for the phrase is added to the list of possible interpretations.

- If it is unknown whether N_2 is an N_1 , then proceed with the rest of the algorithm.

This step just checks to see if the rest of the steps are worth pursuing. For the compound "woman miner", the last case holds.

2. Perform an upward scan from N_2 with marker M1. All default VC links are left inactive during this upward scan. This marks all the superiors of N_2 . Figure 3-9 shows the state of the network for the compound "woman miner" after the concept "&miner" and its superiors have been marked with marker M1.
3. All nodes at the head of a default VC link whose tail is marked with marker M1 are marked with marker M2. This finds all the defaults for N_2 and all of its superiors. It is necessary to find all the default assumptions for a concepts ancestors because they may be the assumption that is overridden. For example, in the compound "boy king", the noun "boy" overrides the default assumption that a "king" is an "adult". It is likely that this assumption will not be stored with "king" but with its ancestor "man". The search must take this possibility into consideration. Figure 3-10 shows the state of the network for the compound "woman miner" after this step has been executed.
4. Mark all nodes marked by marker M2 with marker M3. Perform an upward scan from all nodes marked with marker M3 marking all superiors with M3. This marks all the superiors of an default nodes with marker M3. Figure 3-11 shows the state of the network for the compound "woman miner" after this step has been executed.
5. Perform an upward scan from N_1 with marker M4. Default VC links are made active during this upward scan. Figure 3-12 shows the state of the network for the compound "woman miner" after "&woman" and all of its superiors have been marked with marker M4.
6. Finally, if there is a split statement that points at one concept marked with marker M4 and not marker M3 and points at another concept marked with marker M3 and not M4, then N_1 overrides a default for N_2 and a meaning for the compound is an N_2 that is also an N_1 . In the "woman miner" example, the meaning is a miner who is also a woman.

The above algorithm is run to find the meanings of all compounds that can be created by assuming the first noun is a pseudo-adjective.

The above has introduced the basic paradigm used by the system to create possible interpretations for a noun compound. However, the discussion above has only handled compounds containing two nouns. There are compounds that contain more than two nouns (e.g., glass wine glass, car assembly plant, water meter cover adjustment screw, etc.). The next section describes how the method used for processing the case when there are two nouns can be extended to process compounds containing more than two nouns. Unfortunately, it is not just a simple extension; more structure has to be added to the processing.

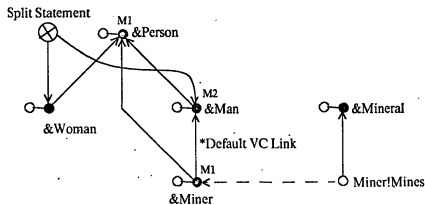


Figure 3-10: Network after default assumptions are marked for "&miner" with marker M2

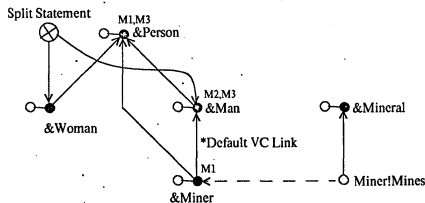


Figure 3-11: Network after the superiors of defaults for "&miner" are marked with marker M3

3.2. Compounds with Three or More Nouns

The obvious way to extend the system to compounds with three or more nouns is to parse the compound in a left to right manner. That is, the first noun fits into a slot of the second. This sub-compound fits into a slot of the third, and so on. This is in fact one pattern that a compound with three or more nouns may have. However, not all compounds with three or more nouns can be interpreted using this simple extension. Rather than the simple pattern mentioned above, compounds with three or more nouns seem to fall into at least the following four distinct patterns:

1. Compounds that can be parsed in a left to right manner, i.e., $[[[N_1 N_2] N_3] \dots N_n]$. Some examples of such compounds include: science teacher institute, coal mine supervisor, blood donor recruiter, and sugar cane plantation owner.

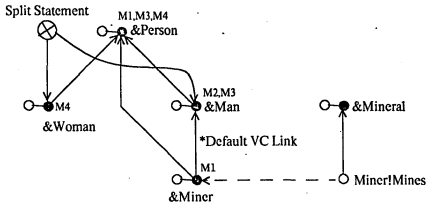


Figure 3-12: Network after the "&woman" and its superiors are marked with marker M4

2. Compounds that can be parsed in a right to left manner, i.e., $[N_1 [N_2 \dots [N_{n-1} N_n]]]$. An example of such a compound is: glass wine glass.
3. Compounds that have a nominalized verb in the middle of the compound, i.e., $[N_1 V_2 N_3]$. Where N_i are nouns, and V_2 is a nominalized verb. Some examples of such compounds are: car assembly plant, food distribution centre, watch repair person, and water meter cover adjustment screw. In the last example, the meaning of "water meter cover" is created in a left to right parse as described in (1) above.
4. Compounds that have a nominalized verb at the end of the compound, i.e., $[N_1 N_2 V_3]$. Where N_i are nouns, and V_3 is a nominalized verb. Some examples of such compounds are: student course evaluation, government price support, executive stock purchase, and city garbage collection.

An important assumption made in the processing of noun compounds containing more than two nouns is that the head noun of any sub-compounds provides all the information necessary to find out whether the sub-compound fills a slot of a succeeding one. The patterns described above were observed in several compounds and seem to handle all of the compounds with more than two nouns that have been encountered so far.

Since several patterns exist for compounds with three or more compounds, it is necessary to use a more sophisticated method to find the possible interpretations than the one used for finding the interpretations for compounds containing only two nouns. For example, consider the compound "coal mine supervisor". Figure 3-13 shows the partial network around the concepts necessary to process this compound. The algorithm that constructs possible interpretations proceeds in a left to right manner. The first step of the algorithm is to activate the concepts representing the meaning of the first word of the compound as described in section 3.1. Since coal only has one meaning (the

node "&coal" in figure 3-13), its slots are marked with the marker SC, and the slots it can not fill are marked with the marker NC. The "supervisor!supervises" slot is marked with NC because there will be a split statement between living and non-living physical objects. At this point, nothing else can be done, and the system starts to process the next word in the input.

All the meanings of the noun "mine" are activated in a similar manner. Again, there is only one possible meaning (the node "&mine" in figure 3-13), and its slots are marked with the marker SM, and the slots it can not fill are marked with the marker NM. At this point an attempt is made to place any sub-compounds, constructed with the previous noun as the head noun, into the slots of "&mine". There is a single previous sub-compound "&coal", since to make the system work more flexibly, the first noun of a compound is considered a sub-compound at this point in the processing. The slot "mine!produces" can be filled by "&coal", i.e., coal is a mineral that can be mined by a mine. A data structure is created to represent this interpretation for the sub-compound "coal mine", and is stored with the other sub-compounds with head noun "mine". In this example, there will only be the one sub-compound with head noun "mine". The system finds nothing else to do, and so goes onto the next word.

All the meanings of the word "supervisor" are activated as described in section 3.1. Again, there is only one possible meaning (the node "&supervisor" in figure 3-13), and its slots are marked with marker SS, and the slots it can not fill are marked with the marker NS. An attempt is made to place the sub-compounds, constructed with any meaning of "mine" as the head noun, into the slots of "&supervisor". The only slot of "&supervisor" that can be filled by "&mine" is "employee!employer", and a data structure that represents this interpretation, where the sub-compound "coal mine" is used to fill the "employee!employer" slot of the only meaning of "supervisor". This interpretation is stored with all the sub-compounds that have any meaning of "supervisor" as the head noun. The system accepts as the possible interpretations all the interpretations that are stored under the last noun of the compound and that span the whole input. In this case there will just be the one interpretation where a "supervisor" is employed by a "mine" that produces "coal". There is a second meaning that many people think of where the supervisor supervises a coal mine and hence its employees. This meaning is obtained by a deeper semantic process than the current system can perform. The current system only looks at one level of slots in concepts. To obtain the meaning "supervisor who supervises employees of a coal mine", the system has to retrieve information from a second level of slots, in particular the "employs" slot of "coal mine". It currently does not use information at this level of detail. This paragraph has described the basic method the system uses to find the possible interpretations of compounds. It must be extended to handle the other three patterns that can be used to form an interpretation of a compound.

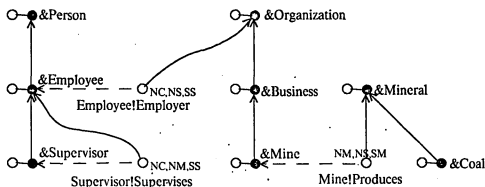


Figure 3-13: Network to Find Possible Meanings for the Compound "Coal Mine Supervisor"

The second pattern that a compound with three or more nouns may match is the one specified by a right to left parse. For example, consider the compound "glass wine glass". Figure 3-14 shows the NETL network around the concepts needed to find the meaning of this compound. There are two meanings for the word "glass", one is a container from which people drink liquids, and the other is a clear material made from amorphous silicon dioxide. The processing for this compound will proceed as described above. That is, the first and second word will be activated, by marking their slots, and the slots they can not fill. However, when an attempt is made to find a slot of "wine" in which "glass" can be placed, the system finds no candidates. This makes the implicit assumption that a liquid does not have a "contains" role. Actually, there is such a slot, it is just that neither meaning of "glass" should be able to fill it. Since "wine" contains "alcohol", a liquid must have a "contains" slot, but note that this use really means that wine is made up of several substances, one of which is alcohol. Glass as a material or a container does not fill this role correctly. Note that a containment relation could exist in which a container contains a liquid and either type of glass, but then the liquid is not really containing either type of glass.

Rather, than aborting the process because no slot is found, and claiming there are no possible interpretations for the compound, the system proceeds to the next word. The second "glass" is activated, but since it has already been seen before for this particular compound, the system does nothing, because all the information the system needs is already available. The system starts by trying to fit any sub-compounds of "glass wine" into slots of "glass". However, there are no such sub-compounds, and nothing can be done. The system then proceeds to see if a meaning of the previous word (i.e., "wine") can fill any of the slots of any of the meanings of the second "glass".

Since "wine" is a liquid that people drink, it can fill the "glass-B!container" slot, and an interpretation for the sub-compound "wine glass" is formed. The next step is to see if any meanings of the first "glass" can fill a slot of the second "glass". Note it is assumed that the sub-compound "wine glass" does not have any additional slots, since it is a type of "glass-B" and thus inherits only the slots found through "glass-B". In fact, it has one less slot available since the "glass-B!container" slot is filled with a value and can not be filled with another one. Since the second "glass" has already been disambiguated to have the meaning of a container from which people drink liquids, this meaning is checked to see if there are any other slots that can be filled by any meaning of "glass". Since a "glass" that people drink from can be made from a material, the first "glass" is disambiguated to mean the clear material made from silicon dioxide, and fills the "container!made-of" slot. Note that the context provided by the compound itself is sufficient to allow the system to disambiguate which meaning of "glass" was appropriate for each occurrence of the word "glass".

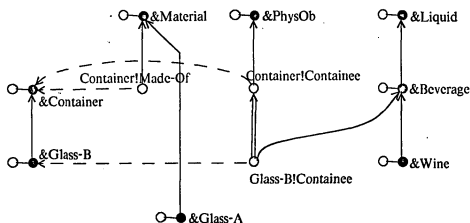


Figure 3-14: Network to Find Possible Meanings of the Compound "Glass Wine Glass"

The phenomenon of the context of a compound disambiguating the meaning for the compound occurs only when one or more of the nouns in a compound have multiple meanings. If only one meaning for such a noun can fill one or more slots of the following noun, then the meaning for the noun is disambiguated to that one particular meaning since the other meaning (or meanings) can not be used as the basis for the meaning of the compound as a whole. Also, if a noun with multiple meanings provides slots that the previous noun can fill, and if only one meaning of the noun with multiple meanings provides one or more slots that can be filled by the preceding noun, then the noun with multiple meanings is disambiguated. The context provided by the compound itself can only

disambiguate nouns with multiple meanings, it will not help disambiguate the cases when one noun has more than one slot that can be filled. Note that even when this occurs, it is sometimes possible to disambiguate a noun with multiple meaning from just the context provided by the compound.

The processing to form the possible interpretations of a compound in which a nominalized verb occurs somewhere in the middle of the compound starts off the same as the two above. That is, the first two nouns have their slots marked, and also the slots that they can not fill marked. For example, consider the compound "watch repair person". Figure 3-15 shows the network around the concepts necessary to find the meanings of this compound. Since the above two methods are being processed in parallel, the sub-compound "watch repair" will be formed after the word "repair" (as a nominalized verb, i.e., describing the action of repairing something) has been processed. However, this sub-compound plays no further role in the processing the system performs. After the word "person" has been activated, the system will try to form sub-compounds based on the first method above, and will fail, since there are no slots of the meaning of "person" that can be filled by the meaning of "watch repair". If a person has an "occupation" slot, then it could be allowed to be filled by "repair", but should it? A problem with this is what restrictions should be placed on the slot? "Repair" describes an action that can be performed by a person. Are all such actions to be allowed to fill this slot? This will allow a large number of incorrect meanings for phrases of the form "action person". "Repairer" on the other should be allowed to fill such an occupation slot.

The system does not stop its processing after finding no slot of "person" that "watch repair" can fill. If the previous word (i.e., "repair") has any meanings that are nominalized verbs, the system attempts to use the slots in it to find possible interpretations for the compound. The first step is to find if the meaning of "person" can be placed into a slot of the meaning of "repair". The "repair|repairer" slot (i.e., the slot for the thing that performs the repair action) is available and can be filled by the meaning of "person". Similarly, all the sub-compounds that have the head noun from two words back, are checked to see if any of them fit into a slot of "repair". In this case, the meaning of the word "watch" can be placed in the "repair|repairer" slot (i.e., the slot for the thing that is being repaired) of repair. The system creates a data structure to represent this particular interpretation of the compound. Since this is the only interpretation found, the system determines the meaning of the compound "watch repair person" to be "a person who repairs watches".

The system begins processing compounds that contain a nominalized verb at the end of the compound in the same way as described above. For example, consider the compound "government price support". Figure 3-16 shows the NETL structures necessary to find the meaning of this

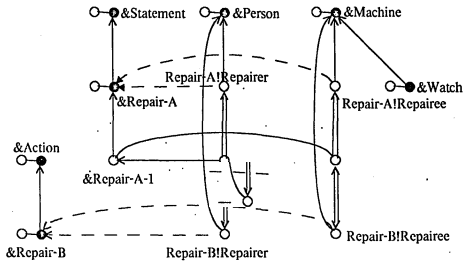


Figure 3-15: Network to Find the Possible Meanings of the Compound "Watch Repair Person"

compound. The system marks the slots of the first two nouns, and also marks the slots that they can not fill. Finally, the word "support" is processed by the system. Since the word "support" is a nominalized verb, the system checks to see if the meaning of the previous word can be placed into a slot of the meaning of "support". Since a "price" can be supported, the system finds that the slot "support!supporttee" (i.e., the thing that is supported) can be filled by the meaning of "price". Note that the restrictions placed on the "support!supporttee" slot must be fairly loose because many things can be supported literally or metaphorically. The system then looks at the sub-compounds whose head noun is two words back from "support". The meaning of the word "government" is found, and the system attempts to place its meaning into any of the slots of the meaning of "support" except "support!supporttee" which has already been filled by the meaning of the word "price". The system finds that a "government" can support things, i.e., it can fill the "support!supporter" slot of "support". The system creates a data structure that represents the interpretation of "government price support" where a "government" supports a "price". Thus, this will be one of the possible interpretations that the system must consider for determining the meaning of "government price support".

Nouns that are pseudo-adjectives may also be involved in compounds with three or more nouns. Pseudo-adjectives tend to be the first noun of the compound. They can modify any head noun of a sub-compound, but the system prefers to have them modify the head noun of the compound if at all possible. If there are two interpretations, one in which a pseudo-adjective modifies a sub-compound,

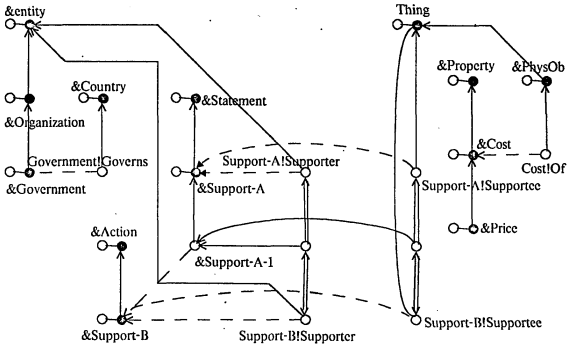


Figure 3-16: Network to Find the Possible Meanings of the Compound
"Government Price Support"

and another in which the pseudo-adjective modifies the head noun of the whole compound, then the system selects the second interpretation as the better one.

3.3. The Algorithm to Find Possible Interpretations

The above descriptions have given examples of how the system finds possible interpretations. This section gives a more complete description of the algorithm to find possible interpretations for a compound. To make the description of the algorithm easier, the following notational conventions are used:

- n - the number of nouns in the compound to be processed.
- W_i - the i^{th} noun of the compound.
- N_i - the number of concepts of W_i that have meanings as nouns.
- C_i - the set of concepts associated with the i^{th} noun, i.e., the set of all possible meanings of the i^{th} noun.

- P_i - all the possible sub-phrases containing W_i as the last noun.
- S_i - the set of possible meanings of noun compounds with head noun W_i .

The full description of the algorithm is as follows:

1. Activate all the meanings W_1 as described in section 3.1.1, i.e., mark all the slots of each meaning of W_1 , and all the slots that each meaning can not fill.
2. Set S_1 to the set of concepts that are meanings of W_1 used as a noun.
3. Set P_1 to the set consisting of the one element W_1 .
4. Set i to 2.
5. Activate all the meanings W_i as in section 3.1.1, i.e., mark all the slots of each meaning of W_i , and all the slots that each meaning can not fill.
6. Set P_i to the set consisting of the set with element W_i union the set constructed by appending W_i to the end of all elements of P_{i-1} . This just constructs the set of all possible sub-phrases ending with W_i .
7. For each x in P_i , add any built-in compounds associated with x to the set S_i . This just takes care of the case when a compound has been built into the data base. Examples of such compounds are "crown prince" and "chain gang". This mechanism could also be used to pick up compounds that have been added to the data base, but currently does not.
8. For each x in C_i :
 - a. If x is a noun, set S_i to S_i union the set of compounds formed by finding a sub-compound in S_{i-1} that fits into a slot of x . Here the phrase is processed left to right.
 - b. If x is a noun and $i > 2$ and S_{i-1} is empty, then first form a compound y by filling a slot in x by a concept from C_{i-1} . Finally set S_i to S_i union the set of compounds formed by filling a slot in y by a sub-compound from S_{i-2} . This allows the system to parse a compound from right to left, if the need ever arises.
 - c. If x is a noun and $i > 2$ and $N_{i-2} > 0$ and y in C_{i-1} is an nominalized verb, set S_i to S_i union the set of compounds formed by filling one slot of y by x and another slot of y by a sub-compound from S_{i-2} . This does the processing to detect noun compounds that contain nominalized verbs in the middle.
 - d. If x is a noun and $i > 2$ and $i = n$ and $N_{i-2} > 0$ and x is an nominalized verb, set S_i to S_i union the set of compounds formed by filling one slot of y by x and another slot of x by a sub-compound from S_{i-2} . This does the processing to detect noun compounds with nominalized verbs at the end.
 - e. If x is a noun and the number of concepts in C_i is not equal to N_i , add x to S_i . This just makes sure that if the previous word could be used as a pseudo-adjective, that x can start a noun compound.

9. For each x in S_i , if x does not start at the beginning of the phrase, try and construct a phrase that starts at the beginning by checking for pseudo-adjectives before the start of x and adding them to the phrase. This is accomplished by the following algorithm:

- a. Set j to the position at which x starts minus 1.
- b. Set B to the set of compounds consisting of the one element x initially.
- c. Set A to B , and B to NIL.
- d. For each y in A and each z in C_j such that z is a pseudo-adjective and z can modify the head noun of y , set B to B union the pseudo-adjective phrase formed by adding z to the front of y .
- e. Set j to $j-1$.
- f. If $j > -1$, then go to c.
- g. Set S_{i+1} to S_i union B .

10. Set i to $i+1$.

11. If $i \leq n$, go to 5.

12. S_n should contain the possible interpretations of the phrase.

At this point, the possible meanings for the phrase should have been found. Since the description of the algorithm above may not be particularly clear, a brief example is presented here to give a better idea of how this algorithm works. The compound "blood donor recruiter" will be used for this example.

A. Step 1 of the algorithm activates the one meaning of "blood" as described in section 3.1.1.

This will mark all the slots that the meaning of "blood" can not fill with marker NB, and also all the slots of "blood" are marked with the marker SB.

B. Step 2 of the algorithm sets the set S_1 to the concept representing the meaning of the word "blood". Note that there is only one meaning of "blood" in this example.

C. Step 3 sets the set P_1 to the set {"blood"}, i.e., the single element "blood". The various P 's are used to find built-in meanings for compounds or sub-compounds.

D. Step 4 sets the variable i to 2. At this point the system has been initialized and is ready to start processing the rest of the compound.

E. Step 5 is the start of the main loop of the algorithm. Since the variable i is currently 2, the meaning of the word "donor" is activated as in section 3.1.1. This will mark all the slots that the meaning of "donor" can not fill with marker ND, and mark all the slots of "donor" with the marker SD. At this point, the algorithm is almost ready to try to find possible interpretations for the sub-compound "blood donor".

F. Step 6 sets P_2 to {"blood donor", "donor"}, i.e., all the possible sub-compounds that end with the word "donor".

G. Step 7 looks at each compound in the set P_2 to see if they have a built-in compound stored in NETL's data base. No built-in compounds will be found for any of the phrases in P_2 , and the processing continues.

H. Step 8 looks at the single meaning for "donor" and does the following:

- Step 8a tries to fit the meaning of "blood" into a slot of "donor". It will find that the slot "donates" of donor can be filled by "blood". Thus S_2 will be set to a description of the compound "blood donor" in which a donor donates blood.
- Steps 8b, 8c, 8d, and 8e do nothing more.

At this point, there will be one sub-compound contained in the variable S_2 .

I. Step 9 does nothing, since the only sub-compound in S_2 spans all of the input that has been seen so far.

J. Step 10 increments the variable i to 3.

K. Step 11 checks to see if i is less than or equal to the length of the compound. Since the compound is of length three, the algorithm will jump back to step 5, and process the next word of the compound.

L. Steps 5 through 10 are executed again with similar results, except that the meaning sub-compound "blood donor" can fill the "thing recruited" slot of the meaning of "recruiter" to obtain the final meaning of the compound as a recruiter who recruits a donor who donates blood. The set variable S_3 will be set to this single meaning for the compound. The variable i will be set to 4.

M. Step 11 will fail to jump back to step 5, since all the words of the compound have been processed.

N. Step 12 will find the single meaning of "blood donor recruiter" in the variable S_3 which will be used as the meaning for the compound.

The next section describes what happens after this algorithm has finished executing.

3.4. Results of Finding Possible Interpretations

Once the system comes to the end of a compound, there are three possible outcomes. The system may not have found any possible interpretations for the compound. In this case, the system reports this fact to the user and does nothing further. This may not be the best thing to do. In fact this is likely to be one of the most important cues for the system to use deeper semantic and pragmatic processing that seems to be necessary to process some compounds, as suggested in chapter 6. In humans it seldom happens that no interpretations are found, but with NETL's current rather limited knowledge base it is not infrequent. The second case occurs when only one interpretation is found by the system. In this case, the system proceeds to construct structures in NETL's data base to represent the meaning of the compound. This meaning can be used later by the system to disambiguate other compounds. If the system comes across the same compound again, it will use the structure it created already as the meaning of the compound, rather than creating a duplicate structure. The last possibility is that the system will discover more than one interpretation for a compound. If this occurs, the system will pass all the possible interpretations to the heuristics section. The heuristics attempt to find the best meaning for a compound by using several different heuristics. Chapter 4 describes in detail all the heuristics that are currently available to the system. When the heuristics have selected the best meanings for a compound, or when there is only one interpretation, the system creates structures in NETL's data base to represent these meanings. The following sub-section describes in more detail the structures the system creates to represent the meanings of compounds.

3.4.1. Creating Structure to Represent an Interpretation in NETL

As the system builds the possible interpretations, it does not immediately alter the NETL data base, because some of the interpretations may not be valid. Rather, it builds a fairly complex data structure that describes exactly what should be built in the network. When the system builds structure in NETL's data base to represent the meaning of a compound, it scans this data structure recursively. It builds all the structure necessary to describe the sub-compounds at each level, before building the structure for the main compound. Since the system looks for slots that can be filled by other concepts, the representation of a compound involves filling a slot with the appropriate value. The concept whose slots are to be filled, the slot to be filled, and the value to fill the slot are all stored in the description of an interpretation. This description provides all the information necessary for the system to create the correct structure to represent the meaning of an interpretation. Figure 3-17 shows an abbreviated description for the compound "car maintenance".

Head Concept: &Maintenance
 Type of Compound: LeftRight
 Word sequence: Car Maintenance
 Concept providing the slots: &Maintenance
 Cognate (if any): &Car Maintenance-A
 Name of slot used to form the meaning: Maintenance!Maintainee
 Concept to be placed into the slot: &Car
 Number of instances in the data base: 0

Figure 3-17: . Description of the compound "car maintenance".

Figure 3-18 shows the structure that would be built to represent the meaning of the compound "car maintenance". This is the structure that would be built in NETL's data base to represent the interpretation of the compound. The single node "&car maintenance" is used to represent the meaning, with other structure built around it to fill in the details of the interpretation. This other structure involves mapping the slot to be filled down to the node "&car maintenance", and setting the node representing the map of the slot equal to the node representing the meaning of the word "car". For the compound "car maintenance" this involves mapping the slot "maintenance!maintainee" from the concept "&maintenance" and making it equal to the node "&car". Thus, the node "&car maintenance" describes a typical maintenance of a typical car.

The structure described above is the basic one used for all the interpretations except for those having a nominalized verb in the middle of the compound. In the case of a nominalized verb in the middle of a compound, the head noun of the compound does not provide the slots that are filled. Instead, the nominalized verb provides these slots. This makes the representation of such a compound somewhat more complex. Again, there is only one node that represents the meaning of the interpretation with structure built around it to fill in the details. For example, consider the compound "watch repair person". Figure 3-19 shows the structures that would be built in NETL's data base to represent the meaning of this compound. The node "&watch repair person" is the single node that represents the meaning of the compound. The node "&repair-A" represents the meaning of the word "repair" as a verb. The node "&repair-B" represents the meaning of the word "repair" as a noun. The individual statement node "&repair-A-1" connects the slots of the two concepts that represent the meaning of "repair" as a verb and as a noun. This structure is used so that less structure has to be built when a compound is created in NETL's data base. The individual statement node "&repair-a-1-1" is created that exists for the node "watch repair person", and maps the correct slots down from the verb associated with the nominalized verb. The slots mapped down are set equal to the nodes representing the meaning of the sub-compounds surrounding the nominalized verb. In this case, the person who does the repairing is mapped down and set equal to the node representing

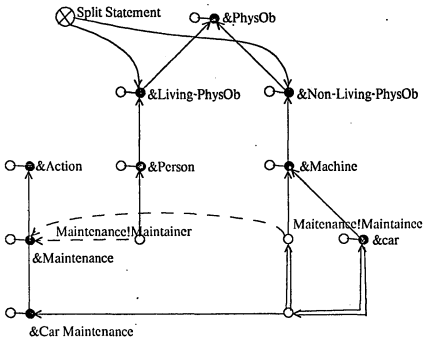
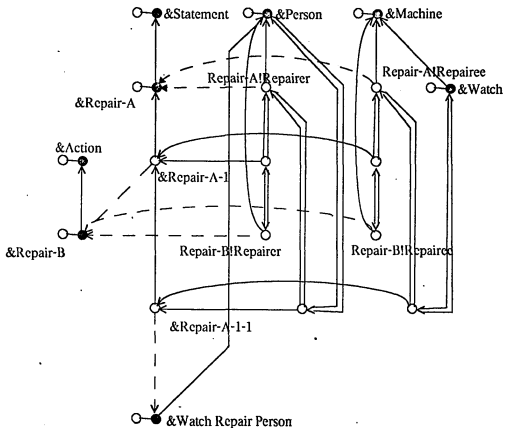


Figure 3-18: The Network Structure Built to Represent the Meaning of the Compound "Car Maintenance"

the meaning of "person", and the thing that is repaired is mapped down and set equal the node representing the meaning of "watch".

When the system creates a compound consisting of a pseudo-adjective, the node representing the meaning of the compound has a VC link created that points from it to the node representing the meaning of the pseudo-adjective as well as a VC link to the node representing the meaning of the head noun of the compound. For example, figure 3-20 shows the structure that would be used to represent the meaning of the compound "woman miner". The node "&woman miner" is created to represent the meaning of the compound "woman miner". This node points at the node representing the meaning of the noun "miner", and also at the node that represents the meaning of the word "woman". There is also a *cancellation link* going from "&woman miner" to "&man". This in effect cancels the default VC link from "&miner" to "&man", so that the split statement won't complain every time "&woman miner" is marked. This structure specifies that a "woman miner" is a "miner" who is also a "woman" and not a "man".

If there is only one interpretation, then the system creates the structure to represent the meaning of



the compound in NETL's data base as described above. If there is more than one interpretation, the heuristics are invoked to attempt to find the best meaning for the compound. After the heuristics have finished processing all the interpretations, structures are created to represent the selected interpretations in the NETL data base, as described above. The next chapter discusses several heuristics that are used to attempt to make this decision.

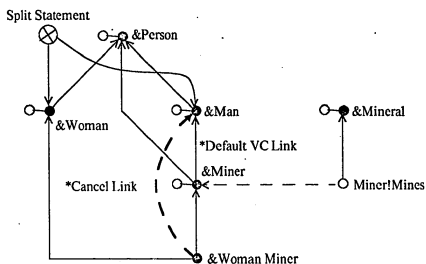


Figure 3-20: The Network Structure Built to Represent the Meaning of the Compound "Woman Coal Miner"

Chapter 4

The Best Interpretation for a Compound

Once all the possible interpretations have been found for a compound, there are three possibilities:

1. There may be no interpretations.
2. There may be one interpretation.
3. There may be more than one interpretation.

Possibilities (1) and (2) are straight forward. In case (1), the system finds no meaning for the compound, and reports this to the user. In case (2), only one meaning is found, and the system creates a structure in the NETL network to represent this meaning of the compound. Possibility (3) is the interesting case. The system must decide which of the possible interpretations is the one that best represents the meaning of the compound. Several heuristics have been developed that attempt to make this decision. Note that these heuristics have been developed outside of a complete natural language understanding system. Many of these heuristics will be subsumed by mechanisms provided by a complete natural language system. The functions provided by many of these heuristics would have to exist in a complete system. The use of focus as proposed by Sidner [Sidner 78] or Grosz [Grosz 78] should provide significant constraint and thus some of the least reliable heuristics will be used infrequently if at all. These focus mechanisms would also replace the "general topic under discussion" heuristic. The "selectional restrictions" and "slot already filled" heuristics will be subsumed by more general mechanism for determining relationships between phrases of a sentence.

There are two classes of heuristics used by the system. One class can remove possible interpretations from consideration; the other only shows a preference for particular interpretations. The boundary between these two classes of heuristics is not well defined, since it is possible for a heuristic to rule out interpretations, as well as provide an indication of how well it supports the remaining ones. For example, heuristics based on context can rule out interpretations, but if more than one remains, they may prefer one of the remaining interpretations over the others. The assumption is the interpretation with the most support is more likely to be the correct than one with less support.

The heuristics that rule out interpretations prune these interpretations from the list of possibilities as soon as possible. The heuristics that can provide a preference for an interpretation calculate a rating in the range zero to one for the compound. This number represents how well the criteria for the heuristic are met by the interpretation. Higher numbers show a higher preference for a particular interpretation. The heuristics providing preference of this nature are executed after the heuristics that can prune the list of possible interpretations. Thus, they are only executed for those interpretations that remain, rather than for all of them.

The next several sections describe in detail all the heuristics that are currently available to enable the system to select the best interpretation for a compound. After all the heuristics have been fully described, a section is devoted to the problem of selecting the best interpretation using the information provided by the heuristics.

4.1. Context

One of the design goals of the system was that it could be integrated with a larger language understanding system. The context used by the system is intended as the connection to such a system. Contextual information comes from the phrases, sentences, and paragraphs around the compound, rather than the compound itself. The current work uses three distinct forms of context to disambiguate the meaning of a compound, as follows:

1. Context based on selectional restrictions [Katz 63] for the head noun of the compound.
2. Context provided by an already filled slot in the head noun of the compound.
3. Context provided by the general topic in which the compound occurred.

Context is provided to the system by a list of the following form:

(Level₁ Level₂ ... Level_n).

Where level_i represents context at some level of the conversation. Level₁ context represents very local context, i.e., probably context provided by the phrase or sentence in which a compound occurs. Level_n represents very global contextual information, i.e., the general topic of the conversation, or information provided by previous paragraphs. The more specific contextual information is given a higher preference than the more general information. As the system currently stands information provided by the first level of context (i.e. at the level of the phrase or sentence in which the compound occurred) is sufficient to specify the context for the "selectional restriction" and "slot already filled" heuristics. The second level is used to specify the context for the "general topic under discussion" heuristic in most cases. Each level contains a list of the following form:

$$(Item_1 \ Item_2 \ \dots \ Item_n).$$

Where $item_i$ is one particular item of contextual information. An item may be a single concept that provides information about the general topic under discussion, or it can be a list of the following form:

$$\begin{aligned} & (NodeName \ SlotList \\ & \quad (Slot_1 \ Value_1) \\ & \quad (Slot_2 \ Value_2) \\ & \quad \dots \\ & \quad (Slot_n \ Value_n)) \end{aligned}$$

Where *NodeName* is the concept providing the slots that can be filled. *SlotList* is a list specifying the slots that the compound could fill due to syntactic information. The position of the compound in a sentence can often restrict the set of available slots of a verb for the compound to fill. $Slot_i$ is the name of a slot of node and $value_i$ is the node that fills it. This structure for presenting context to the system provides a fairly flexible way of using context. This is how the system currently accepts context to disambiguate a compound. Chapter 5 describes how the current system could be interfaced to a parser and gives a more detailed description of where contextual information must come from.

The following three sub-sections describe the three kinds of heuristics based on context in greater detail. The last sub-section describes how the results of these three sub-heuristics are used by the system.

4.1.1. Selectional Restrictions

Selectional restrictions [Katz 63] are a well known method of using the context in which a word with multiple meanings occurs to disambiguate its meaning. Context based on selectional restrictions generally arises from the sentence containing the compound. The selectional restriction paradigm assumes that an ambiguous word must fill a particular role in the sentence. This may involve filling a slot in a verb or noun, or may involve some other aspect of the sentence. For example, the word "club" in the compound "golf club" has at least two meanings that are relevant to understanding the meaning of the compound (i.e., a club as used in games, and as an organization that has the purpose of providing golfing facilities to golfers). In many cases, the meaning for the compound "golf club" can be determined by embedding it in a sentence. For example, the two sentences below provide sufficient context to disambiguate the compound correctly.

1. The golf club hit the ball.
2. The golf club fired its manager.

The context from sentence (1) is presented to the system by providing a structure where "hit" is a verb with its slot for the object being hit is filled with a particular ball. This will leave several slots available in this instance of a "hit" action that may be filled by "golf club". However, the meaning of "golf club" as a piece of equipment for playing golf can only fit into the slot for the thing used to do the hitting. The organizational meaning of "golf club" cannot fill any of the slots of "hit" given the syntactic constraints of the sentence. Similarly, sentence (2) provides the context necessary to obtain the organizational meaning of "golf club", since a piece of equipment can not fire someone, whereas an organization can.

The current system is dealing mainly with ambiguous nouns. Ambiguous nouns internal to a compound can not be disambiguated by the context provided by selectional restriction information, since the structure of a compound involves the ambiguous internal noun filling a slot in one of the preceding or following nouns. However, the head noun of a compound may have multiple meanings and may not be disambiguated by the context provided by the compound. One way to disambiguate such a noun (and in the process, the compound) is to provide some context to the system. To use selectional restrictions, it is necessary to provide the system with the concept that provides the roles the compound is meant to fill in a sentence. In many cases, this will be a verb with some of its slots already filled (i.e., can not be filled by the compound). This leaves some slots available for the compound to fill, and this is the information this heuristic uses to prefer one interpretation over another. In some cases, it may be another type of phrase that provides the selectional restriction necessary to disambiguate a compound.

The information necessary to use the selectional restrictions as mentioned above is provided to the system by a list with the following format:

```
(NodeName SlotList
  (Slot1 Value1)
  (Slot2 Value2)
  ...
  (Slotn Valuen))
```

Where node specifies the concept that provides the slots available for use by the selectional restriction heuristic. SlotList is a list specifying the slots that the compound could fill due to syntactic constraints. Slot_i specifies a slot that belongs to node, and value_i specifies the value that should fill slot_i. For example, in the "golf club" example above, the information provided by the two sentences is presented to the system by the following two lists respectively.

1. (&hit (hit!hitter hit!with)
(hit!hittee &ball-1))
2. (&fire (fire!firer)
(fire!firee &manager-1))

List (1) says that the context is a "hit" action in which the thing hit is &ball-1 and that "golf club" can be placed into one of the slots "hit!hitter" or "hit!with" due to syntactic constraints. Similarly, list (2) says that the context is a "fire" action in which &manager-1 is the person who was fired and that "golf club" can only be placed in the slot "fire!firer" due to syntactic constraints.

The problem the system must address is to find those slots that can be filled by a particular meaning for the compound. If such a slot can be found, then this interpretation of the compound is said to be supportable by context. The system first finds all the slots that belong to the concept that has the slots to be filled. If a list of possible slots is provided, the system uses the intersection of this list and all the slots found, otherwise it uses the list of all slots found. Those slots that are already filled by another concept are not considered. For each of the possible slots, the head noun of each interpretation is checked to see if it can possibly fill the slot. If the NETL network contains no information specifying that the head noun of the compound can not fill the slot, then the interpretation is said to be supportable by context. This is done by searching the type hierarchy to see if the node representing the head noun of the interpretation would cause NETL to complain if it is put into the slot. If NETL returns with the result that the head noun can not fit into the slot, the interpretation is not supportable by context. For example, in figure 4-1, the head node "&club-B" of the compound "golf club" as a piece of golf equipment (i.e., club as a stick) causes NETL to complain when it is placed in the "hit!hitter" slot of the "hit" action, because of type split statement A between &living-physob and &non-living-physob. However, NETL does not complain when it is placed into the "hit!with" slot, since there is no information saying that it can not fit into this slot. The interpretation of "golf club" as a stick is supportable by the context provided by the sentence "The golf club hit the ball". When the concept "&club-A" is placed into either of the slots of hit shown in figure 4-1, NETL will complain. In the first case, split statement B says that an organization is not a "&physob" and thus can not fill a slot that requires a living "&physob". The same split statement comes into play when the "hit!with" slot is considered. Thus the organization interpretation for "golf club" using the "hit" context is not supportable. Note that there are contexts in which the "selectional restriction" heuristic has equal preference for two or more interpretations. For example, the sentence "the golf club belonged to the man" will not allow the "selectional restriction" heuristic to select one meaning of "golf club" over the other.

Selectional restrictions are one method using the context that can be presented to the system. However, this is not the only way that the context in which a compound occurs can be used to disambiguate the meaning of a compound. The following sub-section discusses another way that context can be used to disambiguate the meaning of a compound.

4.1.2. Slot Already Filled

Context from an already filled slot in the head noun of the compound usually arises from a prepositional phrase adjacent to the head noun of the compound. For example, the compound "computer maintenance" can have two meanings. One meaning is when the computer is the thing maintained, and the second is when it is the thing doing the maintaining. The phrase "computer maintenance of the data base" provides context where a slot is already filled for the compound "computer maintenance". This heuristic relies on prepositional phrase attachment to be done before the compound is fully processed. This not an unrealistic assumption, since the phrase "maintenance of a data base" must also have prepositional phrase attachment, but now there is no compound that has to be processed at the same time. It is probably necessary in a more complete natural language understanding system to allow interaction between the prepositional phrase attachment and the noun compound component. This should provide more constraint for phrases like this one. As the system currently stands this would require an extensive amount of work. The noun "maintenance" has at least two slots: one for the thing being maintained, and one for the thing actually doing the maintaining. There will also be slots for the time at which the maintenance action occurred as well as the location, among others. With the phrase above, the slot for the thing being maintained must be filled with "the data base" due to the construction of the phrase. This leaves only the slot for the thing doing the maintenance to be filled by "computer".

The context for this heuristic is supplied by a list with exactly the same structure as for the selectional restriction heuristic above, i.e.:

```
(NodeName SlotList
  (Slot1 Value1)
  (Slot2 Value2)
  ...
  (Slotn Valuen))
```

Where NodeName is the concept that owns all the slots. SlotList in the case for the "slot already filled" heuristic will be the null list, since syntax can place no constraints on what slots are available. Slot_i is the name of a slot associated with the concept NodeName, and value_i is the value with which the slot is filled. The concept NodeName must be a concept that occurs in some interpretation of the

compound, and at least one other concept representing a noun must fill one of its slots. If node is not such a concept, then no interpretation for the compound can be removed from consideration, since node does not occur in an interpretation of the compound, and therefore any of its slots that are filled will not matter. For example, the context to obtain the meaning for "computer maintenance" where computer is the thing doing the maintenance is provided by the list:

```
(&maintenance ()
  (maintenance!maintainnee &data-base-1)).
```

In the absence of such context, it is probably better if the system chose the interpretation where a computer is the thing being maintained. Heuristics other than context ones are available to do this if the network has sufficient knowledge in it.

The system checks for this type of context by looking at all the possible sub-compounds in an interpretation. If the node representing the meaning of the head noun of a sub-compound is equal to the concept specified by NodeName in the list providing the context, the slots filled in the interpretation are compared with the slots of the slot-value pairs in the list providing context. If a slot in the interpretation is successfully matched against one of the slots of a slot-value pair in the list providing context, the slot is already filled by another concept, and can not be filled by a second to form an interpretation for the compound. This means that in such a context, the interpretation is not available and is pruned from the list of possible interpretations. Thus, this heuristic, when it comes into play, is extremely powerful, because it can immediately rule out an interpretation and prune it from the list of possibilities. The other two types of context can only give a preference when more than one interpretation is supported by them.

4.1.3. General Topic under Discussion

Context may also be provided by the general topic under discussion. Such information normally comes from what has gone on in a conversation before the occurrence of a compound. This would normally require the input of several sentences worth of information, which is awkward in the current system. Instead, the system can be presented with a list of concepts that provide the general context of the discussion. Thus, as part of the context information that can be provided to the system, a concept or several concepts can be used to represent the general topic of the discourse.

Returning to the "golf club" example, if the general topic of discussion is organizations of one kind or another, then the organizational meaning of "golf club" is supportable by this context. The "golf club" as used in the game of golf is not supportable. The context for this example can be provided by the single node "&organization" representing the meaning of the word "organization". Similarly, if

the conversation is about sports equipment used in various games, then the meaning of "golf club" as used in the game of golf is supportable, rather than the organizational interpretation.

The system processes such context by looking for a relationship between the general concept provided as context, and any of the concepts that occur in a particular interpretation for a compound. If any such relationship exists, then the system assumes that the interpretation is supportable by the context. The system checks for these relationships by searching for various possible relationships that may occur between the general topic and the concepts in an interpretation. The first search is to see if any of the concepts in a particular interpretation are descendants of the general topic. If any are, then the interpretation is supportable by context. For example, if the general topic of conversation is organizations, then the organizational meaning of "golf club" is supportable by this context, since "&club-A" is a descendant of "&organization" (see figure 4-1). The second check sees if any slots of the general topic can be filled by the concept representing the meaning of the head noun in an interpretation for a compound. If such a slot exists, the system assumes that this particular interpretation is supportable by context.

The problem with this heuristic is it may be difficult to find the topic under discussion in a piece of text. The focus mechanisms of either Sidner [Sidner 78] or Grosz [Grosz 78] should be able to determine the topic of a discussion. The "general topic under discussion" heuristic could use the focus of a conversation as the concept in the search described above. It may be necessary to loosen the search to allow for other relationships to occur. One important relationship is how closely related a concept is to the focus. It is better to use a concept closely related to the focus in an interpretation for a compound than one that is not as closely related.

4.1.4. Using the Results of Context

The results of the three heuristics based on context are used by the system in two ways. The first is to rule out any interpretation not supportable by context, as long as at least one interpretation is supportable by context. If no interpretation is supportable by context, all the interpretations are retained, rather than pruning them all, which would be senseless. If only one interpretation remains after the context routines have finished their processing, the system uses this single interpretation as the meaning for the compound. Since the list of possible interpretations is pruned as soon as these heuristics are finished their processing, the rest of the heuristics should not have to look at as many possible interpretations. The reason the context heuristics are allowed to prune the list of possibilities in this way is based on the assumption that a compound supportable by context is overwhelmingly

better than one that is not. If one interpretation is expected, but is ruled out by context, it is likely the context provided to the system is not really what the user wanted.

4.2. Cognates

The "cognate" heuristic looks for interpretations of a compound that already exist in the NETL data base. There are three distinct types of cognates. The first type of cognate has an idiomatic or "conventionalized" meaning that can not usually be determined by the techniques in this thesis. Examples of such compounds include "chain gang" and "crown prince". The second kind of cognate has a representation already stored in the NETL network, but the sequence of words has not been seen by the system previously. The third kind of cognate is created by the system to represent the meaning of a compound it has processed. These cognates have a well defined structure as described in section 3.4.1, and also have associated with the structure the sequence of words that caused the system to build the structure. The technique for finding these three classes of cognates is different. In the first case, the meaning must be associated with a particular sequence of words. In the second case, the meaning must be searched for in the NETL network. The third case can be handled by either of these two methods, since they have a particular sequence of words associated with them, and they have the structure expected by the routine that searches for cognates in the second case. The system currently searches for such interpretations, and does not use the sequence of words to index the possible cognates that have been seen before and created by the system. The decision to do it this way was made because the search has to be made for the second type of cognate anyway because other information may have been added to the data base since the last time this compound was encountered. It is also a more general solution to the problem. Following is a discussion of what actually is done to retrieve all the possible cognates for a compound of any length.

As each possible interpretation for a compound or sub-compound is found, a search is made to see if that exact interpretation already exists in the NETL network. If it does, then the single node that represents this interpretation is saved to be used when an interpretation for a longer sub-sequence of words is found, and when it is necessary to use heuristics to decide on the best interpretation for a compound. There are actually two distinct searches that are necessary. One is for interpretations that involve filling slots in the concept representing the meaning of the head noun of a sub-compound. This search is used for most of the possible patterns that seem to be formed by compounds. Another search is necessary for the pattern when the concept that has its slots filled is not the head noun of the compound. This arises when a nominalized verb occurs between two other nouns (e.g., food distribution centre, watch repair person, etc.). Following is a description of these two searches.

The first search is straight-forward. Figure 4-2 shows the NETL structure that is necessary to find the cognate ("&wine-glass-A") for the compound "wine glass". The search to find this cognate proceeds as follows:

1. All the nodes below the concept representing the head noun of the compound (i.e., &glass-B in figure 4-2) are marked with marker M1. Figure 4-2 shows the head node "&glass-B" and all of its descendants marked with marker M1.
2. The slot and all the map nodes used to map this slot down to a lower level are marked with marker M2. Figure 4-3 shows the network with all the instances of the slot "glass-B:containce" marked with marker M2.
3. The node "&wine", all the nodes equal to it (via an active EQ link), and all of its descendants are marked with marker M3 as shown in figure 4-4.
4. Any node marked by marker M2 (the marker for the slots) and not marked by the marker M3 (the marker for the value to fill the slot), has its M2 marker turned off as in figure 4-5. At this point, M2 marks all the nodes that could be slots for the cognate representing the meaning of the compound "wine glass".
5. The last step is to mark across the owner wire of all the map nodes marked by marker M2 with marker M4. This marks all the owners of slots that have been filled with the correct value. See figure 4-6 for the results after this has been done.

The possible cognates are all those nodes marked with markers M1 and M4, as shown in figure 4-6. This only gives a list of possible cognates, it is possible that these nodes are not actually cognates. Some further checks must be made to make sure that a particular node is a cognate.

To make sure that a candidate for being a cognate actually is a cognate, three further tests must be made. The first test checks to see if each node that could be a cognate does not have any other slots except those filled by the expected values. If a candidate does have extra slots used, then it is rejected as a cognate. The nodes "&wine-glass-A" and "&wine-glass-1" in figure 4-5 both satisfy this condition. This test is necessary because it is possible to have two or more slots mapped down. This is a more specific meaning than the interpretation should be and thus must be rejected as a possible cognate. The next test is to make sure that the cognate found has the concept representing the head noun of the compound as a direct ancestor. If the candidate does not fulfil this condition, then the candidate is not a legal cognate and is rejected. Again both "&wine-glass-A" and "wine-glass-1" satisfy this condition. This test is necessary because it is possible to have a chain of VC links from the concept representing the head node of a compound, down several levels and then have the right slot filled with the correct value. Again, this is likely to be a more specific interpretation than should be used as a cognate. The last test is to make sure that the node representing the supposed cognate is a type node. The node "&wine-glass-a" is the only node that satisfies this condition. Note that

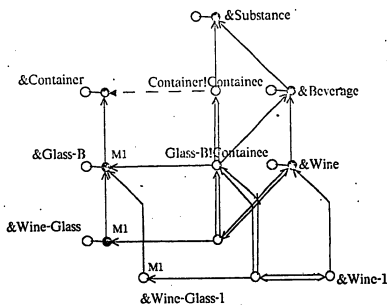


Figure 4-2: The Network to Find Cognates of "Wine Glass".

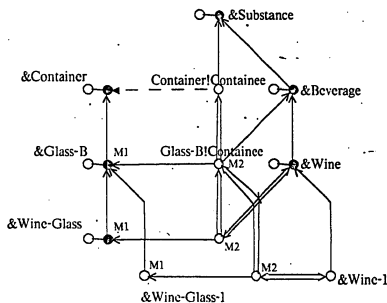


Figure 4-3: The Network After the Descendants of the "Contain!Containee" Slot is Marked.

"&wine-glass-1" is actually an instance where a particular glass was seen with wine in it. This would

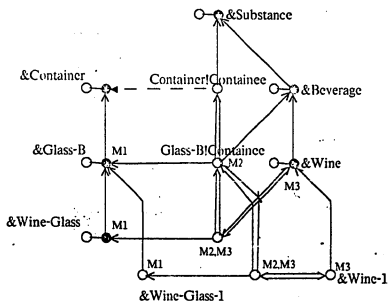


Figure 4-4: The Network After the Slot Filler "Wine" is Marked.

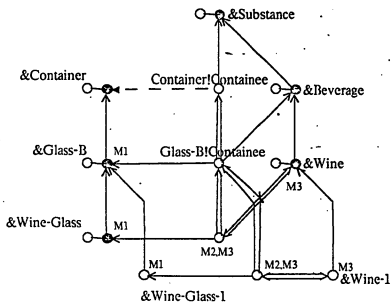


Figure 4-5: The Network After Slots not Marked by the Slot Filler Marker are UnMarked.

likely have been seen before the more general class of wine glasses had been discovered. At some

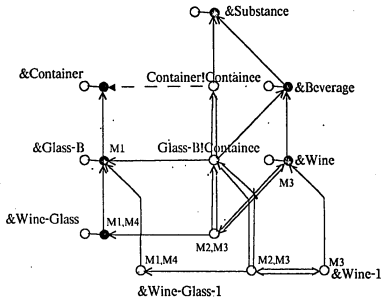


Figure 4-6: The Network Showing Possible Cognates (nodes marked with M_1 and M_4).

point, it should be made a descendant of the node representing the typical wine glass ("&wine-glass-a"). At this point, the cognate for this particular interpretation of a the compound will have been discovered. This cognate will only be one of the ones previously created by the system, or one that does not have a word sequence associated with it.

The search for cognates of compounds that have a nominalized verb in the middle of the compound (e.g., food distribution centre, watch repair person) is more difficult, because the structure to represent such an interpretation is more complex. Figure 4-7 shows the representation of a meaning of the compound "watch repair person". Note that slots in the definition of a typical person (&person) are not used in the representation of the meaning of the compound. Rather, since "repair" is a nominalized verb, there is a verb that is associated with it. This verb is used to create a statement about the relation between the three nouns when the cognate is first formed. The problem is that the slots in the meaning of the noun "repair" are not used, but the slots of the associated verb are. Thus, the problem is to find the verb and its slots, and then find the individual statement that is used in the representation of the meaning of the compound.

The first step in the search is to find the statement that defines the associated verb for the noun "repair". This can be done without too much difficulty by looking at links coming into the node

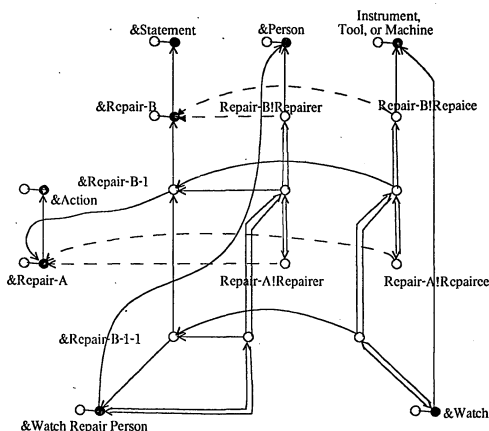


Figure 4-7: Network Structure for Ternary Compounds with a Middle Nominalized Verb.

"&repair-A" that represents the noun meaning of the word "repair". Once the statement has been found, it is possible to find the slots in the statement that correspond to the slots in the noun definition by traversing the H:Q links that tie the two together. The rest of the search is very similar to the search for normal cognates. Rather than scanning downward from the head node for a compound, a downward scan from the IST node (&repair-B-1 in figure 4-7) is performed. When candidate nodes (e.g., &repair-B-1-1 in figure 4-7) are found, a check is performed to make sure that only the slots involved exist, otherwise the node is rejected. The exfor link from it to the actual node representing the meaning of the compound must be traversed to get the node that represents the meaning. This node must be checked for two conditions. The first is it must be a direct descendant of the head node of the compound. The second is it must be a type node. If all these conditions are met, a cognate for this particular meaning has been found. If the node is an individual node, it is counted as an instance of the compound.

The last case to look at is when there is a built-in meaning for a compound. The main problem here is the compounds that must be handled by the system may be of arbitrary length. Since, it is desirable to find the cognates for any sub-sequence of words (since the search for cognates described above uses the cognates for sub-sequences of words to build up cognates for larger sub-sequences), it is not just a simple matter to look up all the cognates stored under the sequence of words in the compound. What must be done is to look at all the possible sub-sequences of words and check each for possible cognates. To construct all the possible sub-sequences of words is not difficult. This is done by putting the first word of a compound in a list. Each time a new word is processed, it is appended to all the sequences of words that exist in the list, and finally added to the list itself. Thus the next word will be joined to the previous word to form a sub-sequence, as well as all the previous sub-sequences. Once all the sub-sequences at a particular point are found, it is necessary to access the NETL network to find all the cognates that are associated with a sub-sequence of words. If any such cognate exists in the network, there will be a link from it to the sequence of words. It is an easy matter to traverse this link backwards to retrieve all the possible cognates for any word sequence. The word sequence itself is represented as a NETL node. This means if the word sequence does not exist in the network, there is no cognate associated with it.

The paragraphs above explain how cognates for a compound are found. However, they do not explain how cognates are used to help make the decision of which is the best interpretation for the compound. The cognate heuristic is one of the heuristics that prunes the list of possibilities. If there is at least one interpretation for which a cognate exists, then all those interpretations for which no cognate exists are removed from the list of possibilities. If there is only one interpretation left in the list of possibilities, it is returned as the best meaning for the compound. Since this heuristic provides only a yes or no answer, its results are used only to prune the list of possibilities, and not to show a preference for particular meanings.

4.3. Embedded Cognates

The embedded cognate heuristic is one of the heuristics that only provides a preference for one of the possible interpretations, and does not prune the list of possibilities. The cognate heuristic above produces all the information that this heuristic needs, since the results of the cognate heuristic are saved with each description of an interpretation. The cognates for sub-compounds of the main compound are also saved. It is these cognates for the sub-compounds that are used by the embedded cognate heuristic. The idea behind the embedded cognate heuristic is that an interpretation having several sub-compounds with associated cognates is better than one that does not. This heuristic is

only effective when there are three or more nouns in a compound and one (or more) of the sub-compounds has multiple interpretations. It does not help make a decision when the top level compound is the cause of the ambiguity.

For example, consider the compound "investment program operation". Assume that the system has previously seen the compound "investment program", and has stored in its data base only one meaning for this compound, i.e., a plan to follow in investing money in securities or some commodity. Note that this compound has a second possible meaning where a computer program has been written that computes things about investments. Thus, when the system processes the compound "investment program operation" it will find at least two meanings for the compound. One is the operation of a plan to invest money and the other is the operation of a computer program for investments. Assume the only meaning for the sub-compound "investment program" that is stored in the data base is the one where a plan exists for investing capital (this could happen with the use of the following heuristic for instance). The "embedded cognate" heuristic specifies that the meaning for the compound "investment program operation" having an embedded cognate for "investment program" is better than one that does not have an embedded cognate. Thus, in this example, this heuristic prefers the meaning where a plan exists for investing, rather than the meaning where a computer program has been written for investing. This heuristic depends heavily on what has happened previously to the system. If all the possible meanings for a compound have been formed and stored in NETL's data base, then this heuristic can not distinguish between any of them, because as far as it is concerned, they are all equally good.

4.4. Instances of an Interpretation

The idea behind the "instances of an interpretation" heuristic is that people go around seeing or hearing things that they remember. That is, they remember instances of concepts, some of which can be named by noun compounds in a straight-forward way. People will build up some structure in memory to represent what they have seen or heard, i.e., people will remember instances or examples of concepts without necessarily attaching a name to them. If later they hear a compound that describes such an instance or example, they could use this information to help disambiguate the compound if it has an ambiguous meaning. This heuristic assumes that the program has had experiences similar to those that people have as they go about life and tries to model this. This heuristic uses a non-monotonic inference since it is reasoning from incomplete knowledge [McDermott 79, Doyle 79, Collins 75].

The "instances of an interpretation" heuristic is similar to the cognate heuristic. Rather, than looking for cognates for a particular interpretation of a compound, this heuristic looks for instances of the interpretation. The search for instances of an interpretation is basically the same as that for cognates. The one difference is that instead of looking for type nodes with the correct structure, it looks for individual nodes. Thus, the search and all the preliminary calculations are the same for both heuristics (and in fact these heuristics are processed at the same time). Instead of a candidate node having to be a type node, as for cognates, it must be an individual node. The important information for this heuristic is the number of instances that are found, rather than the actual nodes that represent the instances.

For example, consider the compound "egg box". This compound has at least two meanings, one where a box contains eggs, and the other where the box is made from eggs. This heuristic assumes the system has picked up knowledge about the world around it, such as having seen instances of compounds without having a particular word sequence associated with them. Thus, for this heuristic to work for the above example, it is necessary to assume the system has seen some instances of boxes that have contained eggs, and that it has not seen as many (if any) instances where a box has been made from eggs. In such a situation the best interpretation to choose is the one where the box contains eggs, since a name can now be associated with structures built up in the data base through experience. This heuristic tries to model the way that people might behave in such a situation. It depends heavily on what information has been stored in the data base.

4.5. Embedded Instances of an Interpretation

The "embedded instances of an interpretation" heuristic has a similar function as the "embedded cognate" heuristic and is a generalization of the "instances of an interpretation" heuristic. In compounds consisting of more than two nouns, there is the possibility that one of the sub-compounds of an interpretation may have some instances stored in the network. An interpretation that has some instances is more likely to be correct more often than one that does not have such instances. This heuristic can only have a preference for a compound consisting of three or more nouns. It is used to disambiguate sub-compounds contained in these longer compounds, since the heuristics based on context usually can not disambiguate these sub-compounds. The results of this heuristic depend heavily on what has happened to the system before it sees the compound.

For example, consider the compound "consumer safeguard plan". Assume the system has not seen the compound "consumer safeguard" before, but does have several instances where a consumer

has been the thing being safeguarded (such information could come from articles in newspapers, etc.) Also, assume the other meanings for consumer safeguard (a safeguard against consumers or a safeguard where consumers are doing the safeguarding) do not have as many (if any) instances stored in NETL's data base. The system will find at least three meanings for "consumer safeguard plan"; one for each possible meaning of "consumer safeguard". The embedded instance heuristic gives a preference for the meaning that has more instances in the data base, and thus will prefer the meaning of "consumer safeguard plan" where there is a plan to safeguard consumers.

4.6. Slot Verification

The slot verification heuristic checks a value that is to be placed in a slot to see how well it fills the slot. The system knows that the value can fill the slot; it is just a matter of determining how well. The definition of a slot has a description of the concepts that are expected to fill the slot. This heuristic checks to see if the value concept is a descendant of this expected concept. The method for finding possible interpretations of a compound allows a concept that is not one of the typical members of the set of concepts that can fill a slot to be placed into the slot. The only time that a concept is definitely ruled out from filling a slot is if the NETL clash detection mechanism would detect a clash if the concept was placed into the slot. The description of a slot may have several VC links running from it to concepts that describe the possible fillers of the slot. The more of these concepts that a value has as an ancestor the better it fills the slot. The problem then is to find out how many VC links from the node representing the slot point to nodes that are ancestors of the proposed filler of the slot.

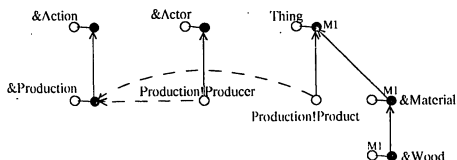


Figure 4-8: Example Network for Verifying Slots of "wood production".

The solution to this problem requires a search of the NETL network. Figure 4-8 shows the structure for two interpretations of the compound "wood production". One occurs when wood is the

product, and the other when wood is the producer. Assume that the slot "production!product" is to be filled by the concept "&wood". The problem then is to see how well the concept "&wood" fits into this slot. The first step is to mark the concept "&wood" and all of its ancestors with the marker M1 as in figure 4-8. The next step is to go through all the VC links coming from the node describing the slot. For each link the node at the end of the link is checked to see if it has the marker M1 on it. The number of links that point at a node with marker M1 on is divided by the total number of links coming from the slot producing a number between zero and one. A value of zero means that the value can be placed into the slot, but that it is not supported by what the slot expects to contain. A value of one means that the value fits the slot as well as possible. In this case, this test is successful, and the slot is verified, producing a value of one for this interpretation. When the slot "production!producer" is checked, the test is unsuccessful and the slot is not verified, producing a value of zero. Thus this heuristic has a preference for the interpretation where "wood" is the thing that is produced.

4.7. Similar Compounds

The "similar compounds" heuristic looks in the NETL data base for any previously defined compounds that are similar to the one under consideration. The basic assumption of this heuristic is that an interpretation having several similar compounds in the network is better than one that has none or only a few. An interpretation is similar to another if the head concept, the slot and the value used to fill the slot of the first interpretation are similar to the corresponding head concept, slot, and value concept in the second one. Two concepts are similar if they both exist in the same hierarchy of type nodes in the NETL network. Two slots are similar if they both exist in the same hierarchy of map nodes. The next few paragraphs describe in more detail how similar compounds are discovered for a particular interpretation.

The "similar compounds" heuristic can be a very expensive heuristic to use, since it may activate large portions of the network. However, its results are usually more reliable than some of the other heuristics. For example, consider the compound "steel truck". The problem with "steel truck" is that two slots in the concept representing "truck" ("vehicle!transports" and "vehicle!made-of") can be filled by "steel". Since there are more compounds in the network in which a concept similar to "steel" is placed in the "vehicle!transports" slot, it is probably the better interpretation as far as the system is concerned. To find these similar compounds, a fairly complex search must be made in the NETL network. The first step is to initialize the search. This involves assigning three markers to each distinct concept that occurs in any of the interpretations. The first marker is the *step marker*

which is used to step up the VC hierarchy one step at a time. Initially, this marker is just placed on all the concepts that exist in any of the interpretations. Figure 4-9 shows the initial set up of the step markers for finding similar compounds of "steel truck". Step Marker S1 is associated with the concept for "&truck" and S2 is associated with the concept for "&steel". The second marker or *downward scan marker* marks all the nodes that can be reached from any node marked by the corresponding step marker in a downward scan of the network. All nodes below a node marked with a particular step marker will be marked by the corresponding downward scan marker. The third marker is used to follow cancel links during the downward scan, so that nodes that should not be reached from the step marker are not marked by the downward scan marker. This third marker will be omitted from this discussion, since it only adds complexity to something that is already very complex. Figure 4-9 shows the results of the downward scan from the nodes originally marked by step markers S1 and S2. The downward scan marker D1 corresponds to the step marker S1, and D2 to S2. The last thing done during the initialization phase is to mark all the instances of the slots involved in all the interpretations. In the "steel truck" example, there are two different slots involved "vehicle!transports" and "vehicle!made-of". A marker is reserved for each slot. Figure 4-9 shows a portion of the network after all the instances of the "vehicle!transports" slot is marked with marker R1, and all the instances of the "vehicle!made-of" slot is marked with marker R2 (no nodes in figure 4-9 are marked with this marker). Note that the defining instances of these slots are not marked, only the nodes that map the slots. At this point, the first attempt is made to find similar compounds.

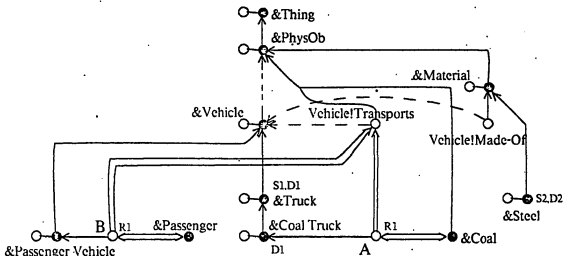


Figure 4-9: Network to Find Similar Compounds of "steel truck" after initialization.

All the slots marked by both the markers D2 and R1 are found. In this case, no such slots exist. Similarly, there are no slots marked by D2 and R2, and thus no similar compounds for "steel truck" where truck is made of steel are possible. Since no similar compounds have been found, the system steps up one VC link. Figure 4-10 shows the results of the search after the stepping up process and downward scan from the newly marked nodes has occurred. Again, no slots exist that have been marked by R1 and D2 or by R2 and D2, and so the stepping up process continues. Eventually, the system will step up to the node "&PhysOb" with both the S1 and S2 markers. Figure 4-11 shows the status of the markers when this happens. At this point there are two nodes that are marked by markers R1 and D2 (nodes A and B in figure 4-11). The nodes owning these slots are marked by marker O1 as in figure 4-12. The marker O1 marks all the concepts that could be a meaning for a compound that is similar to the compound "steel truck". All the nodes marked with marker O1 and D1 are possible similar interpretations for "steel truck" where "steel" is the thing that is being transported. Similarly, the nodes owning any slot marked by both D2 and R2 are marked with marker O2. In this case, there are no such nodes. When a possible similar interpretation is found, a check is made to make sure the slot used in the similar interpretation is the only one defined for the candidate. In this case, it is, and the system has found two similar interpretations ("&coal truck" and "&passenger vehicle" as in figure 4-12). A similar search is made for the interpretation where the truck is made of steel. However, no similar compounds are found. The system currently stops looking for similar compounds as soon as at least one is found for any interpretation, or when there is no node to which the system can step upwards. Thus, in this case, the similar compounds heuristics gives a preference for the interpretation of "steel truck" where the truck is transporting steel, rather than having the truck be made of steel.

The "similar compounds" heuristic has the problem that it will tend to select the meaning for a compound that has the largest number of similar compounds. This is not always correct even when the compound is encountered with no context. For example, this heuristic would find the meaning "a truck that transports fiberglass" for the compound "fiberglass truck". Most people would probably assume that the truck is made of fiberglass. This kind of pragmatic processing is not done by the current system.

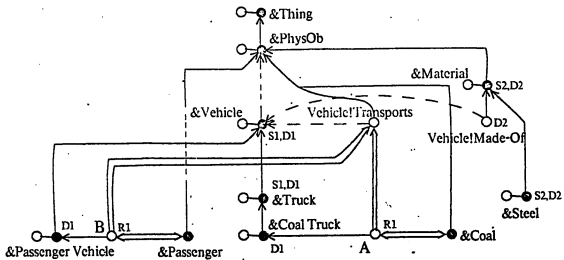


Figure 4-10: The Network After Stepping Up One VG Link for the Compound "steel truck".

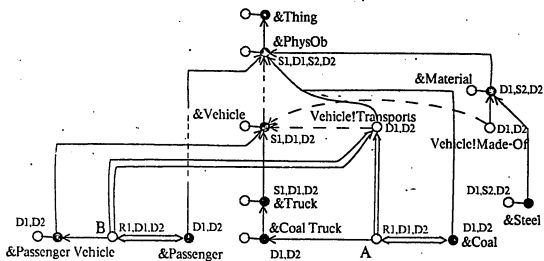


Figure 4-11: The Network After Several Steps Upward have been Processed.

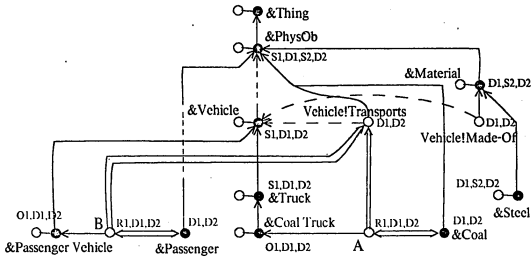


Figure 4-12: The Network Showing Possible Similar Compounds for "steel truck".

4.8. Number of Times an Interpretation has been Encountered

The last heuristic looks at the number of times an interpretation has been encountered. This heuristic only comes into play if the list of possible interpretations has ones with cognates. The system keeps a record of the number of times each interpretation is used over the time that the system is run. This is not trying to simulate the number of occurrences in one piece of text, but tries to simulate the experience that a person might build up over a lifetime of dealing with compounds. The interpretation used the most often is assumed to be better than one that is not used as much. Each time a new interpretation is created in the network, a count of the number of times it has been seen is initialized to one. Each time it is selected as the best interpretation, this count is incremented by one. This is a very weak heuristic that tries to break ties by assuming that a compound that has been seen more often during one session is better than one that has not been seen as often. This is done over only one session since the system does not currently save the compounds created during one session for the next one.

4.9. Combining the Results of the Heuristics

The heuristics specified above are executed in the order specified. This is not important for most of the heuristics, but it is for the first two. The heuristics based on context and the cognate heuristic may prune the list of possible interpretations. The heuristics based on context will prune any interpretation that is not supportable by context, as long as there is at least one interpretation that is supportable by context (i.e., if there isn't an interpretation that is supportable by context, it does not remove any of the possible interpretations from consideration). If after this pruning there is only one possible interpretation left, it is chosen as the best meaning for the compound. Similarly, the "cognate" heuristic removes from consideration all those interpretations that do not have a cognate, if there is at least one interpretation having a cognate. If there is only one interpretation left after the "cognate" heuristic has pruned the list of possibilities, then this single interpretation is chosen as the best meaning for the compound. Currently these are the only two heuristics that have the ability to choose immediately a particular interpretation as the best meaning for a compound. The other heuristics can only show a preference for a certain interpretation, and not select some to the exclusion of all others. Note that the heuristics based on context are executed before the "cognate" heuristic. This is an attempt to reflect the fact that the context in which a compound occurs is a more reliable source of knowledge to select a particular interpretation than having seen the compound before. Context is allowed to overrule the existence of one cognate for a particular compound, if the context supports a different interpretation. The "cognate" heuristic plays no further role in the selection of the best compound, since each interpretation either has a cognate or it doesn't. This is not true for context, since more than one interpretation can be supportable by the same context. Thus, the results of the heuristics based on context are saved and used with the results of all the other heuristics. After these two heuristics have pruned the list of possible interpretations, the other heuristics are executed before any further decisions are made.

All the heuristics except the "cognate" heuristic can support an interpretation to a higher or lesser degree. If a heuristic's conditions are fully satisfied then it has a very high preference for the meaning. If no conditions are met, the heuristic has no preference for the compound. To try to reflect this all the heuristics, except the "cognate" heuristic, calculate a number in the range 0 to 1 that represents how high a preference a heuristic has for a particular interpretation. A value of 0 for an interpretation means the heuristic has no preference for it. A value of 1 for an interpretation means the heuristic has the highest preference of all the possible interpretations for that one interpretation.

Once all the heuristics except the "cognate" heuristic have returned a value representing how well each possible interpretation is supported by the heuristic, the system selects the one interpretation that is supported by all the heuristics, if such an interpretation exists. If a heuristic is unable to support any interpretation, then it is ignored for this calculation. That is, only those heuristics that have supported at least one of the interpretations are considered when the system attempts to use this method to select the best interpretation for a compound. If there is only one interpretation that is supported by all the heuristics showing a preference, then this interpretation is selected as the best one by the system.

If there are still two or more possible interpretations for the compound, the system resorts to a rating system to rate the various interpretations. Each heuristic calculates a value between zero and one. This value is multiplied by a weight that is meant to measure how likely the heuristic is to choose the correct interpretation. For example, the heuristics based on context are very likely to support the best interpretation, and thus have the highest weight associated with them. The number of times that an interpretation has been seen is not particularly indicative of which interpretation is the best one, and thus has one of the lowest weights associated with it. These weighted values are added together, and the interpretation or interpretations with the highest rating are selected as the best meanings for the compound. Table 4-1 gives a list of all the heuristics, and their associated weights. These weights were selected as an approximation to how much importance should be attached to a particular heuristic. The only time that these weights were changed was when a new heuristic was added to the system. They were changed then to give a wider range of values.

Heuristic	Weight
Heuristics Based on Context	16
Embedded Cognate Heuristic	8
Instances of an Interpretation	2
Embedded Instances of an Interpretation	1
Slot Verification	8
Similar Compounds	4
More Specific Slot Heuristic	1
Occurrence Heuristic	1

Table 4-1: Table of Weights used in the Final Rating of an Interpretation

Once the system has selected the best interpretation (or interpretations) for a compound, it creates structure in the NEIL network to represent the meaning of this instance of the compound. If sufficient information is provided to the system, it should come up with only one interpretation. However, there are situations, where more than one interpretation is possible, and in fact desired, if

not enough context is supplied to the system, for instance. The system currently has two modes of operation. The standard mode is to create the structure in the NETL network to represent the meaning of an interpretation if there is only one interpretation after all the processing has been finished. If there is more than one interpretation, the system asks the user to decide which of the interpretations should be created. The system will then create representations only for those interpretations selected by the user. The second mode is to create representations in the network for all the interpretations that are selected by the heuristics. This mode is useful for running tests on the system to see how well it performs. The interpretations created in the network can be used by the system to help disambiguate other compounds using the "similar compounds" heuristic.

Chapter 5

Interfacing the Compound System to a Parser

As mentioned earlier, one of the goals in the design of the current system was to use it with a parser for natural language. This chapter takes a fairly detailed look at how the current system could be interfaced to a parsing system. There were several candidates for this investigation including the Diamond parser [Robinson 80], Parsifal [Marcus 80], and the RUS parser [Bobrow 78, Woods 79, Bobrow 80]. All of these parsers have comparable power and they all allow semantic processing to take place during the parsing process. Rather than a cursory look at how these three parsers could be interfaced to the noun compound system, a fairly detailed description is given for one of them, in particular Parsifal. Parsifal was selected for this task because it has extremely detailed documentation of the parser's operation in the form of Marcus' thesis [Marcus 80]. Also, RUS uses at least some of the ideas in Marcus' thesis.

The first section of this chapter describes the structure of Parsifal, and gives a brief example of how it processes a simple phrase. The second section explores the problem of interfacing the current system for understanding noun compounds to this parser. The third section presents several examples of how the two systems would work together to process four examples of noun compounds. Finally, a summary of the results of this exploration of combining the system with Parsifal is presented.

5.1. The Structure of Parsifal

This section gives a brief overview of the operation of Parsifal. Marcus' parser is based on his determinism hypothesis: "that the syntax of any natural language can be parsed by a mechanism which operates strictly deterministically in that it does not simulate a nondeterministic machine". According to Marcus any deterministic grammar interpreter must have the following properties:

- It must be at least partially data driven.
- It must be partially hypothesis driven to reflect the expectations based on the partial grammatical structures built so far.

- It must have some sort of "look-ahead" facility, even if it is basically left to right.

Once a syntactic structure has been built it must be used in the final parse tree. This hypothesis has lead to a particular structure for Parsifal. His parser consists of two components: one is an underlying architecture; the other is a grammar for parsing natural language that runs on this underlying architecture.

Parsifal consists of two distinct data structures. One is a push down stack, called the *active node stack*, that contains incomplete constituents of the sentence or phrase that is being parsed. Marcus' notation for this stack has it growing downwards, thus the active node is at the bottom of the stack rather than the top. This notation is used so that the active node stack reflects the structure of the final parse tree. The second data structure is a small buffer that contains complete constituents whose grammatical role in the sentence or phrase is not yet known. This buffer can currently hold five constituents. However, the parser can only look at three buffer cells at any one time and thus has limited look-ahead. Words are automatically inserted into the buffer as they are required during the parsing process. Figure 5-1 shows the state of the parser's data structure during the parse of the sentence "John should have scheduled the meeting". At the bottom of the active node stack is an auxiliary node (Aux1) labelled with the features modal, past, among others. It has a daughter node that is the modal "should". Above the bottom of the stack is an S node with the NP "John" as a daughter. The first buffer cell contains the word "have" along with grammatical information about the verb "have". The second buffer cell contains the word "scheduled". The rest of the buffer cells contain nothing. The parser has not yet seen the words "the meeting".

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)
NP: (John)

Aux1 (Modal Past VSPI, Aux) / (Build-Aux)
Modal: (should)

The Buffer

- 1: Word3 (*have Verb TnsLess AuxVerb Pres V-3S): (have)
- 2: Word4 (*schedule Verb V-3S Part Past 1:3D): (scheduled)

Yet unseen words: the meeting.

Figure 5-1: State of Parsifal's data structures during a parse.

The two data structures described above are acted upon by a grammar which is made up of pattern/action rules. The patterns of these rules are matched against the constituents in the buffer and a specific subset of the partially completed constituents in the active node stack. The rules in the grammar are organized into packets. Also each rule is assigned a priority from 0 upwards. The lower this number, the higher the priority of the rule. The parser only matches those rules that are in active packets, and the rule with the lowest priority number that has not yet been run on the current data is selected as the rule to execute. Any rule can activate a packet by associating that packet with the bottom node of the active node stack. Only those rule packets associated with the node at the bottom of the active node stack are active while the parser performs the matching operation. Figure 5-2 shows several rules that occur in Marcus' version of the grammar that runs under Parsifal.

```
{Rule Start-Aux Priority: 10 in Parse-Aux
[= verb] -->
Create a new aux node.
Label C with the meet of the features of 1st and pres, past, future, tnsless.
Activate Build-Aux.}

{Rule To-Infinitive Priority: 10 in Parse-Aux
[= *to, auxverb] [= tnsless] -->
Create a new aux node.
Label C with inf.
Attach 1st to C as to.
Activate Build-Aux.}

{Rule Aux-Attach Priority: 10 in Parse-Aux
[= aux] -->
Attach 1st to C as aux.
Activate Parse-VP.
Deactivate Parse-Aux.}
```

Figure 5-2: Some sample grammar rules for Parsifal

The first line of a rule consists of the word "Rule" followed by the name of the rule. The name is followed by the optional keyword "priority" which is followed by a non-negative integer that indicates the priority of the rule. The default priority of a rule is 10. The last item on this first line is the keyword "in" followed by the name (or names) of the packet (or packets) in which the rule is to be placed. The next line of a rule is the pattern that is to be matched against the buffer. The first item in square brackets is used to match against the first buffer cell, the second item against the second buffer, the third item against the third buffer cell. Also, some rules may want to match against the partially completed constituent in the active node stack. This is done by using a pattern in square brackets as above, except that the first item in the pattern must be the sequence of characters

"C;".** The "C" is used to refer to the partially completed constituent at the bottom of the active node stack. The equal sign in a pattern means that the constituent must have that feature. For example in the second rule in figure 5-2, the pattern "[=*to, auxverb]" means that the item in the first buffer cell must be the word "to" and must be flagged with the feature "auxverb" for the pattern to be successfully matched. Similarly, the pattern "[=tnsless]" means that the constituent in the second buffer cell must be flagged with the feature "tnsless" for this pattern to be successfully matched. The rest of the lines of a rule describe the actions to be taken once the rule has been successfully matched and has been selected for execution. The actions that are important for later sections of this chapter are:

- **Create a new X node:** A new node with name X is created and placed at the bottom of the active node stack, i.e., the parser has decided that the parse tree should have such a constituent in it, and creates a new node of the correct type to represent the constituent. The constituent may be complete or incomplete.
- **Label C with X:** Labels the bottom node of the active node stack with the feature X. In the first rule in figure 5-2 the phrase "the meet of features of 1st and pres, past, future, tnsless" means that the intersection between the set of features of the constituent of the first buffer cell and the set "pres, past, future, tnsless" is taken, and the result is used to label the bottom node of the active node stack.
- **Activate X:** Associates the packet X with the bottom node of the active node stack. This means that while that node is at the bottom of the active node stack, the parser may use the rules in packet X during the pattern matching process.
- **De-Activate X:** If packet X is associated with the bottom node of the active node stack, packet X is removed from the list of associated packets. The parser will no longer be able to use the rules in packet X during the pattern matching process.
- **Attach B to C as L:** Attaches the constituent in buffer position B (one of 1st, 2nd, or 3rd) to the bottom node of the active node stack and labels the sub-constituent with the label L.
- **Drop C:** Once a constituent in the active node stack has been completed, it is necessary to place it in the buffer for the parser to use the complete constituent. The "drop C" command places the constituent at the bottom of the active node stack into the first cell of the buffer. The constituents already in the buffer are shifted-right by one place. If this shifting would cause a constituent to be shifted out of the buffer, an error condition is raised.
- **If B then S1 else S2:** The "if" statement allows conditional execution of actions. The symbol "B" represents a boolean expression that can be formed by tests on various states of the machine or even access semantic routines. The two symbols "S1" and "S2" represent a list of actions that must be separated by semi-colons.

The above description gives a brief overview of the action parts of the rules in the grammar written for Parsifal. There is one further rule that is important. This is the "AS Rule" or *attention shifting rule*. As mentioned earlier, Parsifal's buffer consists of 5 cells, but so far all the rules have only accessed at most three cells. The parser accesses the other two cells by means of an attention shifting rule. Each attention shifting rule has only one node description in its pattern. When the parser is about to start matching constituents in buffer cells against patterns of normal rules, it first checks to see if any active attention shifting rule matches the constituent in the buffer. If the match is successful, the logical start of the buffer is shifted to the buffer cell containing the node that was successfully matched by the attention shifting rule. After the attention shifting rule has been matched successfully, the parser does not see the buffer cells before the cell containing the successfully matched node. References to the first buffer cell will now access the cell containing the successfully matched node, references to the second buffer cell will now access the following cell, references to the third buffer cell will now access the cell two past the one containing the successfully matched node. Figure 5-3 shows the structure of an attention shifting rule. The structure of the rule is the same as before, except now it is labelled as an "AS Rule", and there is only one element in its pattern. Its actions can be any of the ones previously mentioned. The last thing that must be mentioned is how to get back to the previous state, i.e., before the attention shift took place. Once the parser has finished constructing the constituent that caused the attention shift to take place, it can drop the constituent into the buffer in the logical first location, and then execute the command "Restore the buffer". This will return the state of the buffer to what it was before the attention shift command was executed. Note that this leaves the newly constructed constituent in the correct location for the parser to continue from where it was interrupted.

```
{AS Rule Start-NP in CPool
 [= npstart] -->
 Create a new NP node.
 Activate NPool.}
```

Figure 5-3: Example of an attention shifting rule

The attention shifting rules are used fairly infrequently because most of the time three buffer cells are sufficient. However, in some cases it is necessary to parse a complete constituent before some particular rule can be executed. This happens most frequently when dealing with noun phrases. Most of the rules of the grammar treat a noun phrase as one constituent and thus the parse of the noun phrase must be completed before such a rule can be matched against or executed. There may already be constituents in the first or second buffer cell when the start of a noun phrase is detected. Nothing can be done to the constituents in the buffer until after the noun phrase has been parsed. To

parse the noun phrase requires access to three buffer cells and this what the attention shifting rule allows. Other cases when an attention shifting rule is used are: parsing numbers, and a few occurrences where information is set up for the parser and the initial state restored immediately.

This has been a brief description of the parser developed by Marcus. For a more complete description, see [Marcus 80]. The next section describes how the noun compound system could be interfaced to a parser written using the framework developed by Marcus.

5.2. Interfacing the Noun Compound System to Parsifal

This section gives the details of how the ideas in the current noun compound system could be used with Parsifal as described above. There are two basic methods that can be used to interface the noun compound system to Parsifal:

1. All of the nouns in a compound could be collected into a list. This list could then be passed directly to the noun compound system which could process the compound with little modification.
2. The ideas developed for the noun compound system could be used to parse the compound as the parser encounters each noun. The semantic processing for the compound would be done incrementally as each new noun is encountered in the input stream.

Each of these two methods of interfacing the noun compound understanding system to the parser has advantages and disadvantages.

Using the current noun compound system on the list of all the nouns in the compound would be fairly easy to implement. However, it has one major disadvantage. Consider the phrase "the gift John bought". The phrase "gift John" would be taken as a noun compound by a system that just collects all the nouns and passes them to the noun compound system. However, the phrase "John bought" is a relative clause that is modifying the noun "gift" and the phrase "gift John" should not be taken as a compound. If the noun compound system is called after collecting all the nouns it may be difficult to retrace the steps of the parser so that the correct parse is created. If the noun compound system is called incrementally, then this can be detected before the parser has made any irrevocable decisions about the structure of the sentence. Thus, the advantage of processing the compound incrementally is that it is possible to more directly influence the decisions that the parser makes. A similar problem arises when there is a noun followed by a word that could be a noun or another part of speech. For example, consider the phrase "the dog means business", where means

could be either a noun or a verb. The system should find no relation between dog and means and thus can immediately help the parser decide that the phrase "dog means" is not a compound, and "means" should be taken as a verb rather than a noun. The one problem with this suggestion is that there may be a weak relation between the concepts involved. What seems to be necessary is a more complete semantic processor that could evaluate the alternative interpretations and select the one that is more syntactically and semantically preferred. A check of this nature should definitely be made if the compound formed has a very low rating. One disadvantage is that the current noun compound system can not be plugged into the parser as easily. Also, since the compound is being processed incrementally, it will be necessary to move many of the heuristics back into the generator. This has the added benefit that the generator will generate fewer candidates and should reduce the amount of work needed to process a compound. Another disadvantage to incrementally processing a compound is when the compound is ambiguous it may be necessary to postpone creating the final structure for the compound until the sentence has been processed.

No matter which of these two methods for interfacing the system to the parser is used, the following issues must be addressed:

- Context needed to decide on the meaning of a compound may come after the compound is encountered. Thus in both methods, it may be necessary to wait until the sentence containing the compound is parsed before using some of the heuristics based on context.
- A semantic processor that handles more than just noun compounds is necessary. In particular, the "slot already filled" heuristic relies on the fact that prepositional phrase attachment has been done. One slot in the head noun of a compound may be ruled out as the basis for forming the meaning of the compound because a prepositional phrase fills it. The semantic processing necessary to do this comes from the sentence in which the compound occurred.
- A similar problem is the slot assignments for the main verb may have to be partially made if the "selectional restriction" heuristic is to provide context to disambiguate some compounds. The semantic processing necessary to do this comes from the sentence in which the compound occurred.
- The "general topic under discussion" heuristic must get its information from what has gone on in the previous sentences as well as in the current one. This kind of knowledge must come from a fairly sophisticated semantic processor that has access to what has gone on before.

None of these issues involving more complex semantic processing are addressed by the current noun compound system.

For the following discussion on how the current noun compound system could be interfaced to a

parser the method that incrementally processes the compound is used. The main reason this method is used in this discussion is that it is a more interesting approach to the problem since it can provide more information to the parser than the other approach. The method of passing all the nouns found together to the current noun compound system has to address many of the same issues, it just is not as flexible an approach to the problem.

To incrementally process the compound as the parser encounters the nouns in the input stream it is necessary to write some grammar rules for Parsifal that parse a noun compound by accessing the semantic routines necessary to process a compound. The following paragraphs present each grammar rule one (or in a group if the rules have a similar function) at a time. The places where the grammar rules are interfacing to the noun compound system are highlighted by using boldface.

Figure 5-4 shows the grammar rule that gets the noun compound processing started. Rule Start-NC is an attention shifting rule that is in the packet that processes noun phrases. When a noun is first entered into the buffer, this rule will be activated causing the logical start of the buffer to be shifted to the cell containing the noun. It creates a new NC node and attaches the noun to it as the head noun. An NC node represents the syntactic structure of a noun compound. The degenerate case of having only one noun is also stored as an NC node to make the parser rules easier to write. The last thing it does is activate the packet Parse-NC which contains all the rules that actually parse a noun compound. The rest of the rules described in this section are contained in this packet.

```
{AS Rule Start-NC in NPool
[= noun]-->
Create a new NC node.
Attach 1st to .C as HeadNoun.
Activate Parse-NC.}
```

Figure 5-4: Attention shifting rule that starts noun compound processing

Figure 5-5 shows the rule that processes compounds that contain a nominalized verb in the middle of the compound. Its pattern matches against the NC node that should exist at the bottom of the active node stack, a noun that must be flagged with the feature nominalized-verb, and finally any noun. This is the first rule that accesses the semantic routines to process the compound. It calls the routine *MiddleTernary* which checks to see if a middle ternary relationship, as described in chapter 3, exists between the sub-compound at the bottom of the active node stack and the two nouns in the first two positions of the buffer. If this check is successful, i.e., the compound can be interpreted by the middle ternary pattern, the parser executes the rest of the actions. The first action is to drop the NC node that is at the bottom of the active node stack into the buffer. A new NC node is created to

represent the structure of the new compound and is labelled as being a middle ternary compound. The semantic routines return the results of their processing in a variable whose name is constructed depending on where their arguments come from. In this case, since the arguments come from the bottom of the active node stack and the first two buffer positions, the variable name will be !V-C-1-2 (C for the active node stack, and 1 and 2 for the first and second buffer positions). The exclamation mark in front of the variable V-C-1-2 tells Parsifal to evaluate the variable in Lisp. It is also possible that there is a left to right parse of the sub-compound and the two following nouns. This condition is checked for and the results of the analysis are saved with the parse node for the compound. If a left to right parse is possible, the parse node is also labelled with the feature "LeftRight" and the corresponding semantics are attached to the parse so that later processes know that the compound has an ambiguous structure. The noun that is now in buffer cell 3 (used to be in buffer cell 2) is attached to the node as the head noun of the new sub-compound. Similarly, the sub-compound and noun in buffer cells 1 and 2 (because of the drop command) are attached to the new sub-compound node as modifiers. Finally, the noun compound parsing packet "Parse-NC" is made active for this new node, and the processing continues.

```
{Rule Parse-MT Priority: 5 in Parse-NC
[**C; = NC] [= noun, nominalized-verb] [= noun] -->
if MiddleTernary relation between C, 1st, and 2nd then
  Drop C;
  Create a new NC node;
  Label C with MiddleTernary;
  Attach !V-C-1-2 to C as MTSemantics.
  % The compound may also be parsable left to right. %
  (if LeftRight relation between 1st and 2nd and
    LeftRight relation between 2nd and 3rd then
    Label C with LeftRight;
    Attach !V-1-2 to C as L.R1Semantics;
    Attach !V-2-3 to C as L.R2Semantics.);
  Attach 3rd to NC as HeadNoun;
  Attach 2nd to NC as Modifier1;
  Attach 1st to NC as Modifier2;
  Activate Parse-NC.}
```

Figure 5-5: Rule for parsing middle ternary compounds

Figure 5-6 shows the rule used to process the end ternary patterns, as described in chapter 3. This rule follows the structure of the rule for the middle ternary pattern almost exactly. The major difference is that the parser must check to make sure the noun that fits the "nominalized-verb" pattern is the last noun of the compound. All words other than nouns will need to have the feature "end-nc" added to their feature list. This will allow this rule to behave correctly. There is one

problem with this rule. In the previous section it was stated that it is illegal to drop something into a buffer in which all three cells are full. The rule to process the end ternary pattern violates this condition, and thus it will be necessary to extend Parsifal to allow this to happen.

```
(Rule Parse-ET Priority: 5 in Parse-NC
[**C; = NC] [= noun] [= noun, nominalized-verb] [= end-nc] -->
if EndTernary relation between C, 1st, and 2nd then
  Drop C;
  Create a new NC node;
  Attach !V-C-1-2 to C as ETSemantics;
  Label C with EndTernary;
  % The compound may also be parsable left to right. %
  (if LeftRight relation between 1st and 2nd and
    LeftRight relation between 2nd and 3rd then
      Label C with LeftRight;
      Attach !V-1-2 to C as LR1Semantics;
      Attach !V-2-3 to C as LR2Semantics;);
  Attach 3rd to C as HeadNoun;
  Attach 2nd to C as Modifier1;
  Attach 1st to C as Modifier2;
  Activate Parse-NC.}
```

Figure 5-6: Rule to process a compound fitting the end ternary pattern

Figure 5-7 shows the rule that parses a compound from left to right in most cases (except in the two cases above). This rule follows a similar pattern as the two above. It is not as complex because it is looking at only two constituents. Note that its priority number is higher than for the two rules above. This means that this rule will be executed only after the two rules above have either failed to change the status of the machine or their patterns were not successfully matched in the first place. This rule creates a new NC node that contains the information about how the compound should be formed.

```
{Rule Parse-LR Priority: 10 in Parse-NC
[**C; = NC] [= noun] -->
if LeftRight relation between C and 1st then
  Drop C;
  Create a new NC node;
  Label C with LeftRight;
  Attach !V-C-1 to C as Semantics;
  Attach 2nd to C as HeadNoun;
  Attach 1st to C as Modifier;
  Activate Parse-NC.}
```

Figure 5-7: Rule to parse a compound left to right

Figure 5-8 shows the two rules that are necessary to handle compounds that are processed right to left. The first rule creates a sub-compound from the two nouns that must be in the first and second buffer positions. After it creates the NC node, there are two NC nodes on the active node stack. It then drops the first one into the first buffer position. This allows the pattern of the second rule to be successfully matched and its actions finish the process of creating a right to left parse of the compound.

```
{Rule Parse-RL Priority: 15 in Parse-NC
[**C; =NC][=noun][=noun] -->
if LeftRight relation between 1st and 2nd then
  Create a new NC node;
  Label C with LeftRight;
  Attach !V-1-2 to C as Semantics;
  Attach 2nd to C as HeadNoun;
  Attach 1st to C as Modifier;
  Activate Parse-NC;
  Drop C;}

{Rule Parse-RL2 Priority: 15 in Parse-NC
[**C; =NC][=NC] -->
if LeftRight relation between C and 1st then
  Drop C;
  Create a new NC node;
  Label C with RightLeft;
  Attach !V-C-1 to C as Semantics;
  Attach 2nd to C as HeadNoun;
  Attach 1st to C as Modifier;
  Activate Parse-NC;}
```

Figure 5-8: Two rules to parse a compound from right to left

Figure 5-9 shows the rule that finishes up after the compound has been completely parsed as far as it can with the information available. It deactivates the packet that contains these rules and drops the finished compound into the buffer so that the rest of the noun phrase rules can use it to build a noun phrase. Finally it restores the buffer to the state it was before the noun compound was encountered.

```
{Rule End-NC Priority: 15 in Parse-NC
[**C; =NC][=end-NC] -->
De-Activate Parse-NC;
Drop C;
Restore the buffer;}
```

Figure 5-9: Rule that finishes the processing of a noun compound

Figure 5-10 shows the rules that handle the situation when no relation is found between two

constituents that could form a compound. These rules handle cases such as "the gift John bought" or "the dog means business". Both these rules are similar except the first one takes care of the case when there is a single word that is the subject of a collapsed clause or is being used as a different part of speech than a noun. The second rule handles the case when a compound is the subject of the collapsed clause. These rules are insufficient for a more complete semantic component. The problem is that there may be some weak relation between two concepts that should not form a compound because a collapsed clause exists or a word has another part of speech besides being a noun. What is needed in these cases is some method of preferring one structure over the other. The rules to detect collapsed clauses and nouns that are also another part of speech perform the same actions as the rule above, except now there will be a compound and a word, or two compounds left in the first and second buffer positions when the state of the parser is reset to what it was before the first noun was encountered.

```
{Rule Collapsed-Clause-1 Priority: 20 in Parse-NC
[**C: = NC] [= noun] [= end-nc] -->
De-Activate Parse-NC.
Drop C.
Restore the buffer.}

{Rule Collapsed-Clause-2 Priority: 20 in Parse-NC
[**C: = NC] [= NC] [= end-nc] -->
De-Activate Parse-NC.
Drop C.
Restore the buffer.}
```

Figure 5-10: Rules for handling the case when there is a collapsed clause

The above grammar rules have mapped the patterns described in chapter 3 into the structure expected by Parsifal. Once these rules have parsed a noun compound, there is the possibility that there will be syntactic as well as semantic ambiguities. The syntactic ambiguities may arise because of the use of the two ternary patterns for the structure of a compound. In these cases, it is possible that a better meaning exists using a left to right parse of the compound. The semantic ambiguities arise from two sources: one is because some nouns have more than one meaning; the other is because two or more slots of one noun can be used as the basis for an interpretation. The problem remains to resolve these ambiguities in a general way. Since many of these ambiguities must be resolved by using context, it is not always possible to resolve the ambiguities at the time the compound is being processed since the context that provides the constraints to decide which interpretation is best often occurs after the compound itself. The final decision for the meaning of a compound with an ambiguous meaning must often wait until after the sentence in which the compound occurred has

been fully parsed. Most of the time it should be possible to determine the meaning for a compound as it is parsed. However, any time ambiguity can not be resolved as the compound is parsed, it will be necessary to save the results of the other heuristics, since the ones based on context must have precedence. This suggestion is not in keeping with Marcus' determinism hypothesis. He believes that the syntactic/semantic interactions must occur, but a "goodness" measure can be used to select the best parse with limited look-ahead. Marcus has no suggestion for how such a "goodness" measure can be calculated. Waiting until the end of the sentence should provide all the information necessary to disambiguate a compound.

It may be necessary to wait until the end of the sentence in which a compound occurs before being able to decide which interpretation is correct. For example, in the phrase "the police search for the boy", it is necessary to know that the boy fills the "thing searched for" slot of search. This can not happen until after prepositional phrase attachment has been done. In many cases this has to wait until the end of the sentence. There are other problems as well. The major one is that much of the information provided by context comes from having the meaning of at least part of the sentence correctly determined. This problem arises with regard to the "selectional restriction" and "slot already filled" heuristics. The "selectional restriction" heuristic tries to find the role in the sentence that the compound is playing. For this to work correctly, it may be necessary to find the roles that other parts of the sentence play also. This is especially true when the compound is filling a role in the main verb of a phrase, since it may be important to know what other roles of the main verb are filled. With the "slot already filled" heuristic there is a similar, but somewhat different, problem. There are occasions when a slot of the head noun can be filled by another phrase. This usually happens when the compound is followed by a prepositional phrase. Thus, it is necessary to do prepositional attachment before the "slot already filled" heuristic can work correctly. Prepositional attachment must be done because the prepositional phrase following a compound may be modifying the compound or it may be modifying the main verb of the phrase in which the compound occurs. The latter case can only happen when the compound occurs after the main verb. Thus, for the context heuristics to work satisfactorily it is necessary for the natural language system to find the roles that the various parts of the sentence play, as well as doing prepositional phrase attachment. This is an argument against the determinism hypothesis for syntactic and semantic processing at least for noun compounds.

5.3. Examples of How the Interfacing Should Work

This section presents several examples of how the parsing rules described above would work. Four different examples will be worked through. Each example shows a different aspect of the grammar rules and the semantic interaction that is necessary for the meaning of the compound to be determined. The first example goes into a fair amount of detail of how the syntactic processing is done to give an idea of how the grammar would work in practice. The second example presents a compound which has a nominalized verb in the middle of the compound. These two examples do not contain any problems for the semantic routines. The last two examples present the processing for compounds that have an ambiguous meaning after the syntactic processing (with the semantic processing that is done during the parsing of the compound) has finished. These examples show how the grammar can interface to a semantic component that deals with the context provided by the sentence itself. Each of these examples is explored in a separate sub-section below.

5.3.1. Cargo Bay Doors

The first example is the compound "cargo bay doors" as it occurs in the sentence "The cargo bay doors opened". This example is fairly straight-forward since the semantic processing can be done as the compound is parsed. Figure 5-11 shows the state of the parser when the noun "cargo" is first placed into the buffer. The parser has been parsing the noun phrase NP1 which means that the packet of rules containing the rule Start-NC is active. Start-NC is an attention shifting rule that will be executed at this point.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP) / (NPool)
Det: (The)

The Buffer

1: Word2 (*cargo Noun): (cargo)

Yet unseen words: bay doors opened.

Figure 5-11: State of the parser just before rule Start-NC is run.

Figure 5-12 shows the state of the parser after the rule Start-NC is run. The newly created NC

node will be at the bottom of the active node stack. It has the Parse-NC packet of rules associated with it, so this packet is the only active one. Note that the buffer at this point would likely contain no words that are visible to the parser. The words "bay", "doors", and "opened" will be put into the buffer as soon as the parser starts matching rules to find the next rule to execute.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP) / (NPool)
Det: (The)

NC1 (NC) / (Parse-NC)
HeadNoun: (cargo)

The Buffer

1: word3 (*bay Noun): (bay)
2: word4 (*door Noun PL): (doors)
3: word5 (*open Verb ED): (opened)

Figure 5-12: State of the parser after rule Start-NC has run.

Figure 5-12 shows the state of the parser just after it has started matching rules for the next cycle, but before any rules have been successfully matched. The first two rules that will be used to match against the buffer are Parse-MT and Parse-ET since they have the highest priority in the rule packet. Neither of these rules will be successful because none of the nouns in the buffer have the feature "nominalized-verb". The next rule that will be matched against is Parse-LR. This rule does successfully match the contents of the active node stack and the buffer. Thus its actions will be executed. The first thing it does is access the semantic routine "LeftRight" to see if there is a slot in "bay" that can be filled by "cargo". It will find the "contains" slot of "bay" as a compartment that can be filled by "cargo". It saves this relation in the variable V-C-1, so that the parser can access it. Since this test has been successful, the parser will execute the rest of the actions as follows:

1. It will drop the NC node containing "cargo" into the buffer. This makes it available to build a new NC node for the sub-compound "cargo bay".
2. It creates a new NC node at the bottom of the active node stack. This node will be the one that represents the parse and meaning of the sub-compound "cargo bay".
3. It attaches the noun "bay" to the new NC node as the head noun of the sub-compound.
4. Similarly it attaches the NC node for "cargo" as the modifier.
5. It also attaches the meaning for the compound returned by the semantic routine in the variable V-C-1 to this node as the semantics for the sub-compound.

6. Finally, it re-activates the packet Parse-NC since there is a new NC node at the bottom of the active node stack that would have no packets associated with it if this wasn't done.

Figure 5-13 shows the state of the parser after this rule has finished executing.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP) / (NPool)

Det: (The)

NC2 (NC LeftRight) / (Parse-NC)

HeadNoun: (bay)

Modifier: NC1

HeadNoun: (cargo)

Semantics: ((bay (contains cargo)))

The Buffer

1: Word4 (*door Noun PL): (doors)

2: Word5 (*open Verb ED): (opened)

Figure 5-13: State of the parser after the first sub-compound is formed.

The rest of the parse follows a similar pattern. At this point, the rules Parse-MT and Parse-ET will fail to match as before. However, the rule Parse-LR will succeed again. After it has finished executing, the state of the parser will be as shown in figure 5-14. The last thing that will happen is the parse node NC3 will be dropped into the logical first position of the buffer and the status of the buffer restored to what it was before the parsing of the compound was initiated.

This is the end of the first example. It has been fairly straight-forward, since all the semantic processing produces only one meaning, even though the word "bay" has several meanings. The parse tree will contain the semantic meaning for the compound as well as the syntactic structure for it. Note that no changes have been made to the data base. The changes to the data base would probably be made later as the sentence is passed onto the phase that deals with the meaning of the sentence as a whole.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP) / (NPool)

Det: (The)

NC3 (NC LeftRight) / (Parse-NC)

HeadNoun: (door)

Modifier: NC2

HeadNoun: (bay)

Modifier: NC1

HeadNoun: (cargo)

Semantics: ((bay (contains cargo)))

Semantics: ((door (door-of (bay (contains cargo)))))

The Buffer

1: Word5 (*open Verb ED): (opened)

Figure 5-14: State of the parser after the "door" has been processed.**5.3.2. Text Processing Systems**

This example shows how the middle ternary compound "text processing system" as seen in the sentence "This method of document preparation has created a market for **text processing systems**" would be processed by the grammar rules described earlier in this chapter. The processing for this compound would start out in the same way as the processing for the compound "cargo bay doors" as described in the previous sub-section. After the grammar rule Start-NC has run, the status of the parser will be as shown in figure 5-15.

At this point, the pattern for the rule Parse-MT will match the bottom of the active node stack and the current buffer contents. This causes the actions for the rule Parse-MT to be executed. These actions will do the following things:

1. The node at the bottom of the active node stack (i.e., NC1) is dropped into the buffer. This is so that it can be used to build the structure for the compound during the rest of the actions.
2. A new NC node is created to represent the parse of the middle ternary compound "text processing system". All the structure for the compound will be attached to this node.
3. The node is labelled as being middle ternary, so that the semantic routines will know how the compound was formed if necessary.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)
 NP1 (NP): (This method of document preparation)
 VP1 (VP): (has created)
 NP2 (NP): (a market)

PP1 (PP)
 Prep: (for)

NP3 (NP) / (NPool)

NC1 (NC) / (Parse-NC)
 HeadNoun: (text)

The Buffer

1: Word12 (*processing Noun Nominalized-Verb): (processing)
 2: Word13 (*system Noun PL): (systems)

Figure 5-15: Parser state for "text processing systems" after rule Start-NC is run.

4. The meaning for the compound saved when the call to the middle ternary semantic routine was made is stored with the node for later use.
5. If the three nouns can be parsed left to right, then the grammar will label the node as being a left right parse. It will save the meaning or meanings for the compound with a left to right parse so that semantic routines can use these meanings later. In this case, there is no left to right parse that is semantically acceptable, and thus nothing is done.
6. The three nouns are attached to the new node. The noun "system" is taken as the head noun of the compound, and the other two nouns are taken as modifiers for the compound.
7. Finally, the packet of rules Parse-NC are activated for this new NC node so that processing can be finished for the compound.

When the action part of the rule has finished executing, the status of the parser will be as shown in figure 5-16. The rest of the parse is completed as explained in the previous sub-section. Note that the head noun for this compound is different from the one that provides the slots used as the basis for the meaning of the compound.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP): (This method of document preparation)

VP1 (VP): (has created)

NP2 (NP): (a market)

PP1 (PP)

Prep: (for)

NP3 (NP) / (NPool)

NC2 (NC MiddleTernary) / (Parse-NC)

HeadNoun: (system)

Modifier1: (processing)

Modifier2: NC1

HeadNoun: (text)

MTSemantics: ((processing (processor system) (thing-processed text)))

The Buffer**Figure 5-16:** Parser state for "text processing systems" after rule Parse-MT is run.**5.3.3. Police Search**

The compound "police search" occurred in the sentence "Both officials had questioned the the veracity of the story when it was published last Sept. 28, setting off a fruitless police search for the boy". The compound "police search" has the problem that there are at least three meanings for it. All these meanings arise because the word "police" can fill any one of three slots of search (i.e., the thing searched for, the one who is searching, and the thing searched). The syntactic structure of this compound is straight-forward and can be constructed without having to disambiguate the meaning of the compound. The problems arise in actually deciding which of the slots should be the one used as the basis for creating an interpretation for the compound. Figure 5-17 shows an abbreviated structure of the parser just after the syntactic structure of the compound has been determined.

In the context of the phrase "setting off a fruitless police search for the boy", the parser would next parse the prepositional phrase "for the boy". After it had finished processing the prepositional phrase, a semantic component which is not part of the current system would need to find the place to which the prepositional phrase should be attached. This semantic component should attach the prepositional phrase to the compound "police search". This will fill the slot of "search" for the thing searched for. Thus one of the slots that could be used as the basis for an interpretation of the

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP): (Both officials)

VP1 (VP): (had questioned the veracity of the story when it was published
last Sept. 28, setting off)

NP2 (NP) / (NPool)

Det: (a)

Adjective: (fruitless)

The Buffer

1: NC2 (NC LeftRight): (police search)

NC2

HeadNoun: (search)

Modifier: NC1: (police)

Semantics: ((search (searcher police))

(search (search-for police))

(search (thing-searched police)))

2: Word23 (*for Prep): (for)

Figure 5-17: Parser state for "police search" at then end of the parse.

compound has been removed from consideration. This still leaves two slots that are available to form the meaning for the compound. The context available can not help decide which of these two slots should be used. Thus it will be necessary to revert to some of the other heuristics, such as the "similar compounds" heuristic or the "instances of an interpretation" heuristic. Note that it is necessary to wait until after context has been used to use the results of these two heuristics. Otherwise, it is quite likely that the meaning that was ruled out by context will be the one selected by one or both of these heuristics.

5.3.4. Money Supply

The last example in this section is the compound "money supply" which occurred in the sentence "The Morgan Stanley analyst noted that the apparent peaking of inflation coincided with the topping out of growth in the money supply". Here the problem arises because there are two meanings of supply each of which form a meaning for the compound (i.e., the amount of money available or the action of supplying money). The processing of the compound itself will be fairly straight-forward. The syntactic structure of the compound is the same no matter which meaning is meant. However, it is necessary to resort to context to decide which of the meanings is the appropriate one. Figure 5-18 shows the status of the parser after the compound has been processed.

The Active Node Stack

S1 (S Decl Major S) / (Parse-Aux CPool)

NP1 (NP): (The Morgan Stanley analyst)

- 2: VP1 (VP): (also noted that the apparent peaking of inflation coincided with the topping out two years earlier of the rate of growth)

The Buffer

1: Word4 (*of Prep): (of)

2: NC2 (NC LeftRight): (money supply)

HeadNoun: supply

Modifier: money

Semantics: ((supply (amount-of money))

(supply (thing-supplied money)))

Figure 5-18: Parser state for "money supply" after its parse is complete.

In the context of "the rate of growth of the money supply", the meaning for the compound should be the amount of money available. This meaning is determined through the action of prepositional phrase attachment and the "selectional restriction" heuristic. Some other semantic component needs to decide that "money supply" must modify the noun "growth" in the phrase "rate of growth". Once this has happened, the "selectional restriction" heuristic will select the meaning where the amount of money is being talked about since an amount of something can grow, whereas the action of supplying money can not grow. Again, the context in which the phrase occurred must be used to disambiguate the meaning of the compound. In this case it is probably possible for the system to determine this while it is processing the compound. However, if the compound occurred before the main verb of the phrase containing the compound, and the compound was to fill a slot in the main verb using the "selectional restriction" heuristic, it would be necessary to wait until, at least the main verb was discovered. In some cases, it might be necessary to wait until some of the other noun phrases have been placed into slots of the main verb because the compound might be able to fill more than one slot of the main verb. If one of these slots is filled by something after the main verb, this will allow the "selectional restriction" heuristic to discover the correct meaning for the compound.

5.4. Finding the End of a Noun Compound

A major problem with the approach described above is how to decide when a noun compound ends. This is actually part of a more general question of how to decide when a noun phrase ends. As shown in Gershman's thesis [Gershman 79], this is a very hard problem that he only partially solves. The following discussion will focus on the problem of determining when a compound ends. There

seem to be two cases when it is difficult to determine the end of a noun compound. The first occurs when one of the nouns has a meaning as a different part of speech, especially a verb. For example, the phrase "police search" can be a noun compound or it can be the noun "police" followed by the verb "search". The following two sentences show the distinction between these two ways to interpret the phrase:

- The police search for the boy was unsuccessful.
- The police search for the boy frequently.

Another problem arises when a collapsed clause is used. The following two sentences show how this problem arises:

- The boy girls like best is John.
- The metal cars are made of is produced in the east.

The problem with multiple parts of speech at least can be detected easily and the system can be aware that a problem may occur. With collapsed clauses, however, any sequence of nouns could be split to form a collapsed clause. The collapsed clause must be found before this type of problem can be processed correctly. Note that these sentences tend to be what Marcus calls *garden path* sentences since the reader may select the wrong syntactic structure at first; only realizing his mistake after reading more of the sentence.

There are at least three ways that this problem can be approached. One is to perform lookahead beyond the phrase causing the problem until it is possible to disambiguate the structure of the sentence. This lookahead may have to go at least as far as the main verb of the sentence. Note that this may involve lookahead to make sure that the right verb is selected as the main one. The problem with this approach is that this lookahead is going to frequently require more than the three buffer cells allowed in the current implementation of Parsifal. Another solution is to rely on a non-deterministic approach to parsing and allow the parser to back-up or to create more than one parse for a sentence. This approach would completely invalidate Marcus' determinism hypothesis and thus is not a viable alternative within Parsifal's framework. One further alternative is to use Marcus' solution to garden path sentences. Unfortunately, this solution is very sketchy and thus does not form a very strong basis to solve this problem.

To remain close to the paradigm proposed by Marcus, it is necessary to extend Parsifal to allow for a longer lookahead buffer. How long does this buffer have to be? Unfortunately it would have to be

long enough to contain all the constituents of a sentence and thus no fixed limit can be placed on it. If Parsifal was extended in such a way, would it be possible to parse correctly sentences such as the examples given above? The problem with these sentences should be solved once the main verb of the sentence is determined. In the case where a noun is also a verb, no main verb will be found if the word with multiple parts of speech is really the main verb of the sentence. When a collapsed clause occurs, once the main verb is found, the parser should know that a collapsed clause exists and proceed accordingly. A problem exists when a phrase contains three or more nouns. At what point in the sequence of nouns does the subject of the collapsed clause begin? For example, consider the sentence:

- The metal golf carts are made of is produced in the east.

In this sentence the collapsed clause begins at the word "golf". This is one problem that a noun compound processor should help solve. For example, the phrase "metal golf" is probably not going to have a viable meaning or at least a very poor one as a noun compound. The compound "golf cart", however, has a semantically acceptable meaning. This information should allow a natural language system to find the correct place to begin the collapsed clause. Note that this requires access to semantics to obtain this type of information because syntax does not have the necessary information.

The proposals above seriously violate Marcus' three buffer cell lookahead. Other parsing systems can solve this problem without violating their underlying philosophy. For example, the RUS parser [Bobrow 78, Woods 79, Bobrow 80] allows the system to back up. If RUS goes down a garden path as it may do in the examples above, it can back up to select a different path when the first one runs into a dead end. Similarly, the Diamond parser [Robinson 80] carries along all the possible parses for a sentence as it processes the sentence. If a parse goes down a garden path, it will fail and the rest of the parses will proceed. Note that this philosophy is very similar to the one used in the current system where all the possible interpretations for a compound are processed at the same time. Both RUS and Diamond will have to allow for collapsed clauses that can occur anywhere in a noun compound.

This section has pointed out a problem that the analysis in the previous sections of this chapter did not address. It has given some suggestions as to possible ways to approach the problem. As stated earlier in this section, the problem of where a noun compound, in particular and a noun phrase more generally, ends is a hard problem that still requires research.

5.5. Summary

This has been a relatively simple design for the interfacing of the system to one particular parser. How difficult would it be to interface the system to other parsing systems? There are other parsers, such as RUS [Bobrow 78, Woods 79, Bobrow 80] and Diamond [Robinson 80], that have comparable power to Parsifal. The following paragraphs give a very brief description of each parser and some suggestions of how the semantic processing could be done for noun compounds using the ideas developed in this thesis.

Diamond's rules consist of a phrase structure rule and properties that can be evaluated during three different processing phases. The procedures for each of these phases are:

- **Constructors:** These procedures are evaluated during the bottom up assembly of phrases. They are used to build the parse tree for a sentence. This is where the initial calls to the noun compound system should be placed. This allows the parser to reject paths that are not worth following. The semantic information from these calls can be attached to the parse tree for later use.
- **Translators:** These procedures are called after the parse tree has been generated. Their purpose is to translate the parse tree into a semantic representation of the sentence as a whole. This is where the "selectional restriction" heuristic and "slot already filled" heuristics should be placed. They would have access to the sentence as a whole.
- **Integrators:** These procedures take the semantic representation created by the translator functions and integrates this representation into the domain knowledge of the system. This is the best place for the "general topic under discussion" heuristic to be placed, since it needs access to the discourse knowledge that would be available to the integrator procedures.

It should be relatively easy to interface the current noun compound system to Diamond. However, no details have been worked out on how to do this correctly.

The three phases that Diamond has allows a flexible approach to the problem of interfacing semantic routines to a parser. The semantic routines can be placed in the phase that has access to the information that is needed for them to do their processing. The advantage that Diamond has over Parsifal is that it allows for multiple parses to be processed at the same time. Since Parsifal only keeps one parse, it is sometimes necessary to leave the decision about the syntactic structure of a compound to later in the parse. Also, Diamond uses a non-deterministic approach to parsing which allows an easier solution to the problem of finding where a noun compound ends. Diamond's approach to parsing is very similar to the philosophy of the current system.

The RUS [Bobrow 78, Woods 79, Bobrow 80] parser consists of two major components: an ATN parser [Woods 70] and a semantic component. The parser and the semantic component communicate with one another by sending messages back and forth. This communication takes place incrementally as a sentence or phrase is parsed left to right. When the parser has reached the end of a sentence, it sends a signal to the semantic component specifying that the end of the sentence has been reached. A similar paradigm to the above using Parsifal can be used to interface the noun compound system to RUS. The original processing to find the possible relations between nouns can be done incrementally in the RUS paradigm. In ambiguous cases, the semantic component can use the context of the sentence to disambiguate the compound after the end of the sentence. Many of the issues mentioned in this chapter would have to be addressed in interfacing the noun compound system to RUS.

The communication that goes on between the RUS parser and a semantic component is a flexible approach to the problem of processing semantics during the parsing of a sentence. RUS has the advantage over Parsifal of allowing for the possibility for multiple parses of a sentence. It does not carry all the plausible parses along at one time as Diamond does. A parse must be rejected before the next possible one is found. This approach should help with the problem of finding the end of a noun compound, as well as allowing for a collapsed clause to be processed more easily.

This chapter has addressed the issue of how the current system for understanding noun compounds could be interfaced to an existing parser. What types of interactions are necessary for this to be feasible? The following list presents several types of interactions that must go on.

- The syntactic structure that the parser builds as it processes words must be passed to the semantic routines. The semantic routines can look up the possible words in a phrase and check to see if the phrase is semantically viable. The semantic routines will need to keep track of all the information the parser has provided along with any information they have obtained, so that they can use this information while processing later phrases. Note that for noun compounds at least a phrase will consist of a single word in most cases.
- The syntactic structure of a compound is one of the most important things that the semantic routines can return to the parser since it can not directly produce the syntactic structure on its own.
- The semantic routines should also pass back information specifying that a noun compound is not very good semantically and that there seem to be two sub-phrases. The place that these sub-phrases split could be the start of a collapsed clause. The parser could use the information to guide it in looking for such a clause.

Other information must be available to the noun compound system if it is to determine the meaning of ambiguous compounds successfully. This information comes from a more complete semantic

system that is needed for a complete natural language system. The following information is necessary:

- Information provided by a focus mechanism such as proposed by Sidner [Sidner 78] or Grosz [Grosz 78]. This will often be necessary to constrain the possible meanings of ambiguous nouns in a compound.
- Some form of selectional restrictions [Katz 63] that try to find the role a compound plays in a sentence. This should provide constraint for the meaning of the head noun of compound if the head noun is ambiguous.
- Some form of prepositional phrase attachment is necessary. Some prepositional phrases may be attached to a noun compound. These attachments will rule out possible slots of the head noun which can not be filled by preceding nouns. This provides another form of constraint on the meaning of a compound.

These are the various areas where a complete natural language system can provide significant constraints to help select the correct meaning for a noun compound.

Chapter 6

An Experiment using Real-World Examples

This chapter describes an experiment that explores how well the model presented above processes noun compounds occurring in text from real-world examples. The experiment was to collect a large number of noun compounds occurring in text from various sources. The sources included articles from the New York Times, two articles from the Communications of the Association for Computing Machinery: one on spelling correction and the other on learning and reasoning, an article from Scientific American, an article about the properties of semantic networks, and an article from Newsweek on the space shuttle. A list of all the compounds that occurred in each piece of text was made. Appendix B contains the list of compounds found in the articles used for the experiment as well as pointers to the actual articles. Over six hundred compounds were found in this search. Table 6-1 shows how this data broke down in terms of how many compounds with two, three, etc. nouns occurred and approximately how often new compounds occurred in the text.

Article or Group of Articles	2	3	4	5	6	Sent/Comp
Scientific American	103	17	1	0	1	2
CACM: Spelling Correcting	98	11	0	0	0	3
CACM: Learning & Reasoning	65	7	3	0	0	5
Semantic Networks	49	18	0	0	0	7
Newsweek on Shuttle	59	9	1	0	0	2
New York Times	149	32	2	0	0	2

Table 6-1: Number of compounds of each length found and the average number of sentences between finding a new compound.

One interesting statistic is noun compounds occur more frequently than many people believe. Thus being able to understand noun compounds is necessary for a language understanding system to correctly interpret many sentences.

Once this list of more than six hundred compounds was created, each compound was examined to see how well the model presented above would process it. The program itself was **not** run on all these examples. Instead the processing for this set of compounds was done by hand using the algorithms

described above. There are two reasons why the program was not run on this set of examples for a more accurate determination of the results:

- To understand the approximately six hundred compounds would make it necessary to add a large amount of knowledge to the data base. All this knowledge would have to be added to the data base by hand. This means there is a very good chance errors will be made that will require a fair amount of time to correct. Even without this problem, it would still take several months to enter all the necessary data.
- The amount of memory available to store knowledge is very limited because the current system runs on a PDP 20, which has a severely limited address space. Currently the system contains partial definitions of just under two hundred words. With limitations, this vocabulary could be extended to no more than twice this size. That is, it might be possible to add definitions so the system could process about 100 of these compounds, but the amount of working memory would be severely restricted and the system would spend a large amount of its time garbage collecting and thus run very slowly.

Since the program could not be run on this large set of examples as explained above, it was necessary to process the noun compounds by hand. For each of the approximately six hundred compounds found, the following was done:

1. The possible meanings of the words involved and the likely slots involved in the formation of the compound were determined. If there was any doubt about the meaning of the word or what slots it had, a dictionary was consulted to see what it said about the word. An attempt was made not to bias the experiment based on the compound.
2. The noun compound was then processed according to the algorithm given in chapter 3. This would produce the possible interpretations that the program should find for the compound. The heuristics were not used at this point.
3. The meaning that I believed the author of the noun compound intended was determined.
4. The noun compound was then classified into one of several possible classifications by comparing the meaning that would be produced by the program with the one meant by the author. The following paragraph describes the classifications used.

The classification scheme divided the approximately six hundred noun compounds found during the experiment into the following categories:

- Correct left to right parse of the compound: The program would have produced the same meaning as the author intended. A compound was placed in this category if the following conditions held:
 - The program would have produced a left to right parse of the compound, i.e., a structure of the form $[[[N_1 N_2] N_3] \dots N_n]$.
 - The program would have produced only one meaning for the compound without

using any heuristics, i.e., there would only be one possible candidate for the meaning after all the possible interpretations were found as specified in chapter 3.

- The slot or slots selected by the program as the basis for an interpretation of the compound are the correct ones, i.e., these slots provide the underlying relationship or relationships between the nouns forming the compound. Since almost all words are going to have several slots, the program must select the correct slot to represent the meaning of the compound. A left to right parse without giving the correct relationship between the nouns involved is of little use in understanding the compound.
- If one or more nouns of the compound have multiple meanings, the program would select the correct meaning for them. In these cases the correct meaning of the noun should be determined by using the context provided by the compound itself without the need to resort to various heuristics.
- **Correct middle ternary meaning of the compound:** The program would produce the correct meaning of a compound containing more than two nouns using the middle ternary pattern as described in chapter 3.
- **Correct end ternary meaning of the compound:** The program would produce the correct meaning of a compound containing more than two nouns using the end ternary pattern as described in chapter 3.
- **Multiple interpretations produced due to ambiguous words:** The program would produce more than one meaning for the compound because one or more of the words in the compound had multiple meanings. The correct interpretation would be one of the ones found by the program. With the use of context, the program should be able to process these compounds correctly.
- **Multiple interpretations produced due to multiple slots:** The program would produce more than one meaning for the compound because the head noun of the compound or sub-compound had more than one slot that could be filled by the previous noun or sub-compound. The correct interpretation would be one of the ones found by the program. With the use of context, the program should be able to process these compounds correctly.
- **No meaning would be found by the program:** The program would find no possible interpretation for the compound, i.e., no slot would be found that could be used as the basis for an interpretation of the compound.
- **The wrong meaning would be found by the program:** The program would find one meaning that would be incorrect given the context in which the compound occurred.
- **A questionable slot is needed for the correct meaning to be produced:** The program would find the correct meaning for the compound only if a slot of a questionable nature was allowed. A questionable slot is one which may or may not be reasonable for the noun to have associated with it.

- **A questionable slot assignment is needed for the correct meaning to be produced:** The program would find the correct meaning for the compound only if a slot is filled with a questionable value. A questionable value is one which may violate the restrictions that are normally placed on the slot involved.
- **The meaning is correct as far as it goes:** The program produces a meaning which is correct as far as it goes. The compound usually has some connotation that is not covered by the simpler meaning that is likely to be created by the program.
- **The first noun provides the slot for the second:** The program would not produce any meaning for the compound. However, if the program was allowed to place the second noun of the compound into a slot of the first, the correct meaning would result.
- **The compound is defined in the text:** The program would not produce the correct meaning for the compound. However, the compound is defined in the text to have a particular and usually very specific meaning.
- **The slot filler is a union of concepts:** The program would not produce the correct meaning for the compound. The two or more nouns preceding the head noun seem to fill the same slot of the head noun as well as having some relationship among themselves.

Table 6-2 shows the break down of the approximately six hundred noun compounds examined according to the above classification scheme.

Category of Compound	Percent in the Category
Left to right parse	52%
Middle ternary	1%
End ternary	1%
Multiple interpretations due to multiple meanings of nouns	6%
Multiple interpretations due to multiple slots	2%
No meaning	10%
Wrong meaning	6%
Questionable slot	13%
Questionable slot assignment	3%
Correct meaning as far as it goes	2%
First noun provides a slot for the second	2%
Compound defined in the text	1%
Union of concepts filling same slot	1%

Table 6-2: Results of using the proposed model on real-world examples

From table 6-2, it can be seen that about fifty-four percent of the compounds from text occurring in real-world documents should be processed correctly by the program without using any heuristics. The following sub-sections explore each of these classifications in more detail. The examples used in the following sub-sections are taken from appendix B which contains a numbered list of all the compounds used in the experiment. The examples are keyed to the entries in appendix B with the

notation " $\#c$ page p ", where p is the number of the page on which the compound occurs and c is the number of the compound.

6.1. Correct Left to Right Parse of Noun Compounds

The largest category of compounds is the one for which the program would produce the correct left to right parse. Examples of such compounds include: "water flow" (#279 page 216; a flow where the thing flowing is water), "water pipe situation" (#282 page 216; a situation describing pipes that carry water), "learning ideas" (#303 page 218; ideas where the ideas are about learning), "oil consumption" (#24 page 194; consumption where the thing consumed is oil), "attribute name" (#340 page 222; a name where the thing named is an attribute). This category of compounds is fairly straight-forward and the program should have no trouble with them. The only interesting thing about these compounds is that the program must select from all the slots available the one that produces the correct meaning. In these examples the restrictions placed on the slots are sufficient to rule out all but one interpretation. A few of the compounds (about 5% of the compounds in this category) contained nouns with multiple meanings. Examples of these types of compounds include: "story plot" (#265 page 215; a plot that gives a brief description of a story; plot could also mean a plot of land), "cargo bay doors" (#404 page 228; doors to a bay (as a compartment) that holds cargo; bay could also mean a body of water among others), "capital expenditure" (#16 page 194; expenditure where the thing expended is capital (i.e., money); capital also has the meaning as a city where a country's or state's government is located). The interesting point with these compounds is that the program should be able to disambiguate the meaning of the ambiguous nouns using the context provided by the compound itself without needing to resort to more complex heuristics.

6.2. Compounds Formed from the Middle Ternary Pattern

A few compounds that contained three nouns would be processed using the middle ternary pattern described in chapter 3. Examples of such compounds are: "text formatting program" (#126 page 203; a program that formats text), "text processing system" (#131 page 203; a system that processes text), "main-tank attachment point" (#393 page 227; a point that is used to attach a main-tank), and "law enforcement official" (#566 page 242; an official who enforces the law). This category of compounds is also fairly straight-forward and the program should have no trouble processing them.

6.3. Compounds formed from the End Ternary Pattern

A few compounds that contained three nouns would be processed using the end ternary pattern described in chapter 3. Examples of such compounds are: "engine power output" (#57 page 197; output where power is output by an engine), "accessory power requirements" (#77 page 199; requirements that require power for accessories in the context of accessories for cars), "memory storage requirements" (#195 page 209; requirements that require memory for storage), and "higher-operator quantifier representation" (#373 page 225; a representation that represents quantifiers by higher-operators). This category of compounds is fairly-straight forward and the program should have no difficulty processing them.

6.4. Multiple Interpretations due to Multiple Meanings of Nouns

The first interesting category is the one where one or more of the nouns in the compound have multiple meanings and by using context, the system should produce the correct meaning. Note that there are many more compounds containing nouns with ambiguous meanings. However, the compounds with these other ambiguous nouns are placed into other categories, such as the correct left to right parse category if the context provided by the compound itself is sufficient to disambiguate the compound. There are approximately 6% of the compounds that fell into this category. For example, the compound "plant material" (#110 page 201) could mean material that comes from a living plant, or it could mean material in or around a plant that performs a manufacturing operation. The only possible method to disambiguate this compound properly is with the use of context. Another example, is the compound "water pipe law" (#239 page 213) which could mean a law passed by some governing body concerning water pipes, or it could be a physical law about the behaviour of water pipes under various circumstances. Context seems to be the only method to disambiguate this compound properly. Another example is the compound "money supply" (#473 page 234). This compound could mean the action of supplying money to someone, or the amount of money available for some purpose. Other examples of these kinds of compounds include: "table structure" (#202 page 209; the structure of a table used to eat from or a table as a data structure in computer programming), "table size" (#225 page 211; the size of a table from which people eat or a data structure in computer programming), "plot outline" (#240 page 213; outline of a plot for a story or a plot of land). Context seems to be the best way for the system to disambiguate these compounds. The following paragraphs give an indication of how the current system would use its heuristics based on context to disambiguate these kinds of compounds.

In the above examples, the problems arose because one word has more than one meaning. The "selectional restriction" or "general topic under discussion" heuristics are the heuristics most likely to come into play in these examples. The slot already filled heuristic, the other heuristic based on context, will not help disambiguate these compounds. The compound "plant material" (#110 page 201) arose in the context "methanol can be produced from plant material" in a discussion about fuels. This compound is one that may not work under the current implementation of context. The selectional restriction heuristic has the problem that it is likely to provide no constraint since either meaning of the compound could fill the "produced-from" slot in the verb "produce". The general topic under discussion heuristic has a similar problem. The way that people seem to disambiguate this compound is to know that methanol is normally made from organic sources, and thus "plant material" means material from the living thing. The current system might get the correct meaning by providing "methanol" as the topic of conversation, assuming that methanol has the information that it can be made from plants. If such an assumption is made, then the system will correctly disambiguate the compound.

The compound "water pipe law" (#239 page 213) occurs in a context in which physical laws are being discussed. Thus the general topic under discussion heuristic should produce the correct meaning for this compound. The problem seems to be to decide that physical laws are being discussed. The compound "water pipe law" occurs in the same sentence that the phrase "Ohm's Law" occurs. Ohm's law can only be interpreted as a physical law, and thus so should "water pipe law".

The example of "money supply" (#473 page 234) occurred in the context "rate of growth of the money supply". Thus "money supply" must be able to fill the thing that is growing slot in the noun "growth". The selectional restrictions heuristic can come into play since an action is something that one would not normally expect to grow, whereas the amount of something can grow. The compounds in this category can be handled in many cases by the system as it currently stands if certain simplifying assumptions are made.

6.5. Multiple Interpretations due to More than One Slot

About 2% of the total compounds would produce two or more meanings because more than one slot is available to be filled. For example, the compound "police search" (#525 page 239), could mean either a search to find the police, a search of the police for something, or a search conducted by the police. The system as it currently stands would be most likely to select the first meaning, because

there may be other compounds where the first noun is the thing being searched for, and the similar compounds heuristic would find these, and say this is the most likely. Another example where this causes problems is with the compound "consumer safeguard" (#512 page 238). The meaning here could be either a safeguard to protect consumers against something, consumers could be safeguarding something against something else, or consumers are the thing that are to be protected against. Again, the similar compounds heuristic is likely to select the wrong meaning in this example, because there may be other compounds where safeguard is used and the thing being protected against slot is the one filled. The only way this problem can be overcome is to use the context in which the compound occurred to disambiguate the compound. Other examples of these kinds of compounds include: "consumer preference" (#13 page 193), "consumer demand" (#20 page 194), "search time" (#199 page 209; time taken for a search to be performed, or the time at which the search took place), "computer learning" (#309 page 219; learning where the subject learned is computers or learning where the learner is a computer).

The solution to this kind of problem seems to rely heavily on the "slot already filled" heuristic. Most of these compounds seem to come up in contexts in which at least one of the slots has already been filled, and thus is unavailable to use as the basis for determining the meaning of the compound. For example, the compound "police search" (#525 page 239) occurred in the context "police search for the boy". The slot for the thing searched for is taken up by "the boy", and can not be filled by police. However, the other two meanings (i.e., where the police are the ones searched, and where the police are doing the search) were not ruled out by the context in which the phrase occurred. It seems likely that these two meanings can be differentiated with the use of the "similar compounds" heuristic or possibly the "instances of an interpretation" heuristic. With the "similar compounds" heuristic, it is likely that there may be other compounds in the data base in which someone similar to the police have performed an action similar to a search. If there is no information of this nature, then the "similar compounds" heuristic will provide no help in disambiguating the compound. The "instances of an interpretation" heuristic may be useful (at least for a person) because a story (or stories) might have been read where the police have been the ones doing the searching. Similarly, the slot for the thing being safeguarded against in the case of the compound "consumer safeguard" (#512 page 238) can be ruled out by the "slot already filled" heuristic due to the context in which the compound occurred. To resolve the other two meanings of the compound, it is necessary to rely on some of the other heuristics. Again, the "similar compounds" heuristic or the "instances of an interpretation" heuristic may supply the necessary constraint to have the correct meaning selected (i.e., where consumers are the ones being protected).

Thus, the kinds of problems arising from this category seem to be solved with the use of mechanisms that already exist in the system. It does not seem necessary to develop new mechanisms to handle many of the compounds placed into this category. Rather, it is necessary to use all the information that is normally available to a person for the system to process the compound correctly. The program will generate more than one possible interpretation for the compound, and will need to decide which slot should be used to form the correct interpretation. As with the previous category, the most effective way for the program to make this choice is to use heuristics based on context.

6.6. No Meaning is Generated for the Compound

For about 10% of the compounds examined no meaning was found for the compound because no slot could be found specifying a relationship to use as the basis for an interpretation. Examples of such compounds include: "methanol route" (#112 page 201; in the context where methanol is being considered as a possible fuel for vehicles), "hash table" (#200 page 209), "hash chain" (#218 page 211), "hash function" (#219 page 211), "ball bearing" (#439 page 231), "1980 model year" (#80 page 199), among several others. There do not seem to be any slots that are used to form the meaning of the compound in these cases. It seems that at least some of these compounds must be considered as cognates, or at least that a much deeper metaphorical type of reasoning is going on than the model proposed above can currently process. Other compounds (e.g., "methanol route" (#112 page 201) can not be considered as cognates in any reasonable manner. This is where a more complex metaphorical meaning must be found. The current system has no mechanisms for doing such complex inferences.

6.7. The Wrong Meaning is Generated for the Compound

For about 6% of the compounds, the program would generate the wrong meaning for the compound. For example, the compound "chain search" (#166 page 206) is most likely to be taken as a search for a chain in many contexts. However, in the context of a discussion about hash tables, and a "hash chain" (#218 page 211; i.e., a list of tokens that are hashed to the same table entry are placed in a list or chain) used to resolve conflicts, then a "chain search" means that a "hash chain" is the thing being searched to find the token that is the object of the search. Another example is the compound "power plant" (#26 page 194). The system would find the meaning where there is a plant (consisting of buildings, etc.) that produces power which is a correct meaning. However, the compound occurred in the context of discussing how automobiles are propelled. The system would not understand that the "power plant" had to fit within the confines of a car. Although the meaning

is technically correct, the system has no way of realizing that a "power plant" in this sense really means the engine of a car. Other examples include: "market close" (#627 page 248; the time at which a market (especially the stock market) closes, but the program would find the act of closing the market), "two-passenger car" (#28 page 195; a car with a capacity for two passengers, but the program would find the meaning as a car that transports two passengers), key assumption (#286 page 217; an assumption that is critical, but the program would find the meaning where an assumption is made about a key), and "engine size" (#73 page 198; the size of the space in the cylinders, whereas the program would find the meaning as the physical size of the whole engine). These examples show some of the strange behaviour that can be produced from the current program. Is there any method to fix these problems?

It seems reasonable to place some of these compounds in NETL's data base as cognates, i.e., storing a predefined meaning for the compound in NETL's data base. This will solve some of these problem cases in a reasonable manner. For example, consider the compound "noun phrase" (#343 page 222) which the system would say is a phrase that consists of nouns. However, this is not quite correct, since it has a particular meaning in a linguistic context (i.e., a phrase containing nouns, but also other parts of speech) which would not be found by the system; unless it was made into a cognate. The compound "power plant" (#26 page 194) can probably also be handled by using cognates, since it is a fairly frequent term that could be defined. A compound like "chain search" (#166 page 206), however, could probably be understood by a person who had not seen it before. There does not seem to be a good reason to make this compound a cognate. Instead, a metaphorical meaning would have to be generated for this compound.

6.7.1. Ellipsis

Some of the compounds are short hand for a phrase occurring earlier in the text or contain nouns that are short hand for phrases occurring earlier in the text. For example, in an article about the grain embargo against the Soviet Union the phrase "curb of grain sales" occurred. Later in the same article, the compound "grain curb" (#456 page 233) occurred. The system as it currently exists would not make the connection between the two phrases, i.e., that the second phrase is really short hand for the first one. The system would instead create no meaning for the compound "grain curb". In an article about semantic networks, the phrase "semantic network" occurred in the text. Later the phrase "network notation" (#334 page 221) occurred. The current system would create a meaning for "network notation" and not establish a connection between the two phrases. Another example is the compound "bureau agent" (#563 page 242) where previously in the article the phrase "Federal

Bureau of Investigation" occurred. Again the system would construct a meaning for "bureau agent" without realizing "bureau" stands for the "Federal Bureau of Investigation" or FBI. One further, and somewhat more complex, example involves the two phrases "higher-order representation of quantification" (#356 page 224) and "higher-order quantifier representation" (#373 page 225) which occurred in the order given here. The two phrases seem to refer to the same concept and thus should be represented by the same node in NETL's data base. The current system does not handle the kinds of problems raised above. NETL is built around structures that are meant to be searched in a very orderly manner. What is needed is some way to prime the network to look for looser associations between concepts than the normal NETL searches currently allow. The following paragraphs describe an approximation within the NETL framework to this more general mechanism.

The basic problem seems to be that a noun or sub-compound occurring in a compound represents a more specific phrase because of the context in which the phrase occurred. In all interesting cases the more specific phrase has occurred before the less specific one. Note that the more specific one may occur several paragraphs before the less specific one. A proposal for processing these kinds of phrases is to keep a list of the concepts previously seen, especially ones containing more than one word. When an article is about to be read this list would start empty. As new concepts are read or formed, they would be added to this list. With a NETL implementation, it would probably also be wise to reserve a marker, say the current concepts marker (CCM), to mark all the concepts seen during the reading of the article. As new concepts are encountered or created in NETL's data base, they would be added to the list as well as being marked by the CCM marker. When a compound is encountered the following additional processing can be performed:

1. For each meaning of a noun in the compound, a downward scan should be performed. This will just mark all the more specific instances of the meaning for a noun. Assume these concepts are marked with marker M1.
2. All the concepts marked by both marker CCM and marker M1 are found. This gives a list of potential concepts that could be a basis for an elliptical meaning for the noun or phrase.
3. If no concepts are in the intersection defined by these two markers, there are no possible elliptical meanings and the process can terminate.
4. There seem to be two cases depending on whether the concept being checked represents the meaning of a head noun or a modifying noun as follows:
 - a. If the concept being checked is the head noun, then the modifying nouns should occur in the structure of the concept found in the intersection search. The search to find these concepts may be somewhat complicated because it may involve several levels of slots (e.g., there are two levels slots that need to be searched in the case of the compound "grain curb").

- b. If the concept being checked is not the head noun, and if there is only one concept found in the intersection search, then it is probably the correct meaning to use for the noun. If more than one meaning is found in the intersection search, the system should do nothing.
5. Finally, the system in case (a) would use the concept that best matches the compound under consideration as the meaning for the compound. In case (b) the noun would be replaced with the more specific instance in the meaning of the compound.

Looking back at the examples mentioned above, how would this proposal work in practice? The phrase "curb of grain sales" occurred before the compound "grain curb" (#456 page 233). Assume the system created a structure representing the phrase "curb of grain sales" below the type node for "curb" (note that a structure for "grain sales" would also have to be created that would be pointed at by this structure). This structure would be marked with the CCM marker and placed in the list of current concepts as described above. When "grain curb" is found, the system would mark downward from "grain" finding nothing, and also mark downward from "curb" finding the structure created earlier for the phrase "curb of grain sales". This structure would have to be checked to see if it referred to the concept "grain" anywhere within it, since "curb" is the head noun in this case. Since it does, this structure would be taken as the meaning for the compound "grain curb". That is, the meaning for "grain curb" would be expanded to mean "curb of grain sales".

Another example is the compound "bureau agent" (#563 page 242) where FBI is the only bureau seen earlier in the article. The processing would proceed as before, i.e., a downward scan from both "bureau" and "agent" is made. Nothing will be found for "agent", but FBI will be found below "bureau". Since "bureau" is not the head noun and only one concept below it is active, FBI can be used to replace "bureau" in the compound. The "network notation" example would be processed similarly.

A harder example is "higher-order quantifier representation" (#373 page 225) where "higher-order representation of quantification" (#356 page 224) has been seen previously. The processing would follow the "grain curb" (#456 page 233) example described above. However, a problem arises because "quantifier" and "quantification" are two different concepts. The search for concepts in the original structure would have to be relaxed to allow the ellipsis processing to accept the structure "higher-operator representation of quantification" as the meaning for "higher-operator quantifier representation". Note that if either "quantifier" or "quantification" is used consistently, there would be no problem. The connection between the two phrases seems to be that "quantifier" is a slot of "quantification". Thus for ellipsis processing to work, in this case it would be necessary to extend the search to allow for one concept to be a slot of the other.

The above paradigm for ellipsis processing should handle a fair number of cases where this problem arises. It is fairly simple in nature, but should provide a starting point for implementing a strategy for handling ellipsis where noun compounds are concerned, as well as possibly other phrases. A more sophisticated approach to this problem could involve the use of focus mechanisms such as those proposed by Sidner [Sidner 78] or Grosz [Grosz 78].

6.8. A Questionable Slot is used to Generate the Correct Meaning

For about 13% of the compounds, the correct meaning for the compound is generated with a slot that is of a questionable nature, i.e., may or may not be associated with the concept that needs it to produce the correct meaning. Examples of such compounds include: "pollution problem" (#106 page 201; the problem caused by pollution), "spelling problem" (#151 page 205; the problem of correcting mis-spelled words), "transposition error" (#165 page 206; the error caused by transposing two letters in a word), "letter error" (#162 page 206; the error caused by mis-typing a letter in a word), "computer file" (#214 page 211; a file that is part of a computer system), and "telephone call" (#567 page 243; a call made via telephone).

There seems to be a need for a slot in the concepts representing the meaning of "problem" and "error" that specifies the type of problem or error. In cases like these it seems that such a slot is not unreasonable. Assuming that such a slot is reasonable, what restrictions should be placed on it? Unfortunately it is very difficult to specify precisely what restrictions should be placed on such a slot. No restrictions could be placed on the slot by allowing anything to fill the slot. This means that any compound of the form "x problem" will have at least the meaning where x fills this "type-of" slot. In most cases this is probably the desired meaning, but how should the few cases where this is not the desired result be handled? The most likely answer is the heuristics the system uses have to take care of those cases where this is not the desired meaning. In particular, the heuristics based on context are most likely to produce the correct interpretations in such situations.

The compound "computer file" (#214 page 211) also has a problem with slots. The obvious slots such as "part-of" or "owned-by" do not seem to carry quite the right meaning for this compound. To be part of a computer the part should be a physical part of the computer. A file is not a physical part of the computer. The way to arrive at such a meaning is to extend the meaning of computer to include all the logical parts as well. The question then becomes how far should this extension be carried out. One solution is to relax the constraints placed on this particular slot to allow "file" to be part of a computer. Though a "computer file" can be thought of as being owned by a "computer",

this is a major relaxation of the constraints on the "owned-by" slot, since one would not normally expect a "computer" to own something in the same way a person or a corporation might. The "part-of" relationship does seem to be the one wanted, even though it does not quite capture the correct connotation.

The compound "telephone call" (#567 page 243) is another example in which it is questionable whether the appropriate slot should exist. To understand this compound it is necessary to have a slot that represents the instrument used to make the call. It seems inappropriate to add such a slot just to make one compound work. Are there other times when this slot could be used to understand a phrase? The answer seems to be affirmative. For example, consider the compound "radio call". This has a similar meaning and is appropriate in a context where radio is the normal form of long distance communication, such as in wilderness situations. This is one way that the compound could be formed, but there is some doubt whether the right connotation for "call" is being used. It seems to be used as a "short visit", and as such a "telephone call" or "radio call" should be established as cognates to arrive at the correct connotation.

There does not seem to be an easy solution to the problems raised above. In most cases it seems necessary to create new well defined slots. The main problem seems to be that in some cases there are tenuous relationships that are used to form the meaning of some compounds. One solution is to relax the constraints placed on the relationships used as the basis for forming interpretations for a compound. Rather than place a large amount of constraint on the search by looking for particular patterns in the data base (i.e., looking for slots in which nouns can fit), the system could look for any relationship between the concepts in the compounds. If all constraints are removed in this search, the system will find many spurious relationships. For example, any two concepts have the relationship that they are both inferiors of the top-most node called the *thing* node. This relationship would not be interesting in itself. However, there may be other relationships that could be. For instance it may be possible to find relationships between concepts based on paths through roles involving more than one level which is all the current system allows. To relax constraints in this manner requires much more powerful inference mechanisms than are currently used by the system. This suggestion is similar to the kind of searches used by Quillian [Quillian 68, Quillian 69].

6.9. A Questionable Slot Assignment is used to Generate the Correct Meaning

For about 3% of the compounds, the correct meaning for the compound is generated by filling a slot with a questionable value. This means that the restrictions likely to be placed on the slot will be violated by the concept that is to fill the slot. Examples of such compounds include: "tow vehicle" (# 396 page 227; vehicle whose purpose is to tow the space shuttle), "post-flight picture" (# 397 page 227; picture taken after the flight of the space shuttle), "thruster dance" (# 403 page 228; dance of the rockets of the space shuttle on launching), and "explosive separator" (# 394 page 227; separator that uses an explosive to detach fuel tanks from the shuttle).

All these compounds could cause problems because the restrictions on slots would probably not allow the first noun to fill a slot of the second. For example, the compound "thruster dance" (# 403 page 228) uses a metaphorical meaning for dance to produce a meaning where a thruster performs actions similar to a dance. For most applications a thruster should not be able to perform a dance in the same way a person does. In this case it is necessary to relax the constraints placed on the dancer slot of "dance" so that thruster can fill it. Note that an interpretation of a compound is not given by just filling a slot, there must exist in the data knowledge about what the implications of the relation between the main concept and its slots. The compound "post-flight picture" (# 397 page 227) has the problem that "post-flight" must be taken as a time for it to fill the time taken slot for picture. What must happen is that the adjective "post" must be processed correctly. This should turn the phrase post-flight into a time which can then fill the slot with no trouble. The current program does not perform such an inference.

Most of these kinds of compounds should be processed correctly once there is some mechanism for relaxing the constraints placed on slots. Some may have to be handled by adding more possible parts of speech to the system and performing inferences based on the information they add. These problems should not be insurmountable.

6.10. The Correct Meaning as far as it Goes

For about 2% of the compounds, the correct meaning is found as far as it goes. The meaning is technically correct, but it is does not capture the full meaning that the author intended. Examples of such compounds include: "pickup truck" (# 6 page 193), "diesel particulates" (# 104 page 201), "word space" (# 138 page 204), "property list" (# 338 page 222), "news conference" (# 508 page

237), "newspaper editor" (#583 page 244). The program would generate technically correct meanings for all of these compounds. To get the full meaning of the compound the system would need to be provided with the meaning of the compound, i.e., the compound would have to be made a cognate with an idiomatic or "conventionalized" meaning.

For example, consider the compound "pickup truck" (#6 page 193). The program would generate the meaning where a truck is used to transport a pickup (a pickup as defined by the New American Heritage Dictionary is "one who is picked up, especially: passengers or freight"). This is correct, but the full meaning is a particular style of truck. There is no way a person or the program could get this special meaning without being shown an example of such a truck. Another case is the compound "property list" (#338 page 222) as used in the context of Lisp. The program would generate the meaning "a list whose elements are properties". This is correct, but in Lisp a property list has a very special meaning that would be missed by the program. The compound "diesel particulates" (#104 page 201) would have the meaning "particulates from diesel fuel", but what it really means is "particulates from diesel fuel after it has been burned". This example is not as easy to pass off as a cognate, since there is a process going on that people at least would consider in generating the meaning. There seems to be a very large amount of knowledge about what goes on during burning that has to be brought to bear to correctly interpret the meaning of the compound. The program as it currently stands does not have this knowledge nor does it have the inference techniques that are necessary to follow this chain of reasoning.

For most of the compounds in this category it is reasonable to assume that they are cognates. If such is the case the program would have no trouble with most of these compounds. A few of the compounds could be said to be derivable from the knowledge that people have, even though the current program does not.

6.11. The First Noun Provides a Slot for the Second One

About 2% of the total compounds seem to fit a different pattern than the patterns described in chapter 3. In particular, it seems that the first noun in the compound provides the slot that is filled by the second noun of the compound. Examples of such compounds include: "input file" (#134 page 204; a file that is input), "input token" (#196 page 209; a token that is input), "output file" (#198 page 209; a file that is output), "production vehicles" (#30 page 195; vehicles that are being produced), and "production worker" (#471 page 234; where worker is performing the production).

This structure for a compound violates one of the major assumptions of the model described above, i.e., that nouns later in the compound provide slots for the earlier ones, except in the case where there is a nominalized verb in the middle of the compound. The number of compounds the system can process by adding the possibility of having the first noun provide the slot for the second is only increased by a small amount, but it adds somewhat to the coverage of the program.

Adding this pattern to the ones that already exist may cause the system to create many more meanings for compounds than it currently does. It seems it is necessary to use this pattern only when there is no other interpretation for the compound. The system at any time has a list of all the possible interpretations for a compound. If this list is empty, the system could try fitting the last noun found into a slot of the previous noun. All the information that the system needs to check for this possibility exists in the current implementation, although it is not currently used. In particular, the first noun of all compounds is processed in the same way that all succeeding nouns are, and thus has all of its slots properly marked for determining whether the second noun can fill one of its slots.

6.12. The Meaning of the Compound is Provided in the Text

For about 1% of the compounds, the meaning of the compound is described in the text because the author intends it to have a very specific meaning. Examples of such compounds are: "confusion matrix" (#170 page 207; matrix giving the chance of inputting the wrong character given the correct one), "text portion" and "code portion" (#184 page 208; two parts of a program; a text portion is the part that describes the algorithm and a code portion is the part that gives the code for the algorithm), "comment frame" (#254 page 214; frame that comments on the contents of a Minsky style frame), "black bag job" (#568 page 243; illegal break-ins performed by the FBI).

The program would do various things with these compounds. No meaning would be found for some of the compounds (e.g., "confusion matrix" (#170 page 207), and "black bag job" (#568 page 243). For the others, the program would find the wrong meaning. For example, for the compound "text portion" (#184 page 208) and "code portion" (#185 page 208) the program would generate the meanings "a portion consisting of text" and "a portion consisting of code" respectively. For the compound "comment frame" (#254 page 214), the program would probably generate the meaning as "a frame that contains a comment" which is close to the correct meaning. The program currently has no way to handle such compounds, other than having their definitions put into the data base by hand. Eventually, it would be necessary to have a natural language understanding system be able to input the descriptions in the text and store the correct meaning for the compound in the data

base. Note that the meaning for the compound should normally only be used within the context of the article in which the compound occurred.

6.13. A Group of Concepts Fill the Same Slot of the Head Noun

For about 1% of the compounds, the meaning of the compound is best described by having several concepts fill the same slot in the head noun of the compound. Examples of such compounds are: "cost-benefit analysis" (#114 page 202; analysis of the trade off between the cost and benefit of something), "voltage-resistance-current situation" (#238 page 213; a situation involving voltage, resistance, and current of a circuit), "agent-act-object combination" (#253 page 214; a combination of the agent, act, and object of an information structure), "frame-slot-value combination" (#255 page 214; combination of a frame, one of its slots, and the value filling the slot), "case-filler combination" (#261 page 215; combination of the case and the concept filling it), "attribute-value pair" (#339 page 222; pair consisting of an attribute and its value).

The program as it currently stands does not handle these types of compounds. The head noun of each of these compounds gives an indication of how the previous nouns are related to one another. These compounds seem to be talking about a single thing whose constituents are made up of the concepts associated with the nouns preceding the head noun. For example, the compound "frame-slot-value combination" (#255 page 214) is talking about the single concept made up from a frame, a slot that exists in the frame, and the value used to fill the slot. This meaning for the compound is quite complex, since to obtain the full meaning it is necessary to decide on what kind of relation exists between the concepts preceding the head noun. It should be possible to establish this relationship using the current program if the hyphens are ignored. The one problem is that the head noun must accept almost anything in its main slot. For most of these examples, this should not be a problem. In the case of the compound "cost-benefit analysis" (#114 page 202) it is a problem. However, "cost-benefit analysis" is one compound that could possibly be considered a cognate.

6.14. Multiple Word Senses

One of the problems that has not been addressed fully in the above discussion is what happens when all the possible word senses for a word placed in the data base. There are several issues that are raised when this is done. One is how many candidates will be generated during the phase that generates all the possible interpretations for a compound. Another is how to represent the multiple word senses in the data base. And finally, what types of constraints are necessary to select the right word sense and/or relationship that forms the basis for the correct interpretation for a compound.

If all the possible word senses and all the slots these word senses are stored in the data base, the number of possible interpretations for a compound is going to increase. In some cases this will cause a large number of interpretations to be added to the list of possible ones. The largest number of interpretations seems to be on the order of about five. This is a large number of interpretations for the heuristics to process. However, five total interpretations is far less than the maximum number that could be generated. This means that even the current system provides a fair amount of constraint.

One problem with multiple word senses is how to represent the different senses. There seem to be two extremes. One is to represent each word sense as a separate concept with very strict restrictions on the slots for each one. This method has the disadvantage that there will be many distinct nodes representing a meaning for a word. Another way is to have all the word senses represented by one concept with many slots. The various relationships between the slots could be used to select the appropriate sense for a word. This method has the disadvantage that the most common slots will have very loose restrictions. These restriction will provide very little constraint on any searches looking at such concepts. What seems to be a better solution is a combination of these two schemes. All the distinct or major meanings of a word should be represented by a separate concept. Below each major meaning of a concept could be a hierarchy of concepts that represent the various senses of the major meaning. The slots of the major meaning would be common to all the senses; various senses could place more stringent restrictions on these slots. If only an approximation to the correct meaning is needed, the major meaning could be used. To get more detail it would be necessary to search the hierarchy below the main meaning for the correct sense of the word. The slots of the word senses in the hierarchy should provide a significant amount of constraint for this search. This representation should allow a fairly flexible approach to the problem of representing multiple word senses.

With multiple word senses how can sufficient constraint be obtained to restrict the set of possible interpretations for a compound? There are at least the following possible methods to obtain more constraint:

- The use of focus such as that proposed by Sidner [Sidner 78] or Grosz [Grosz 78] should provide a significant amount of constraint. Meanings more closely related to the focus are better than ones that are not as closely related.
- The concepts filling slots of the concept with multiple word senses should provide some constraint. In particular, the fillers of the various slots may be such that only one word sense is correct.

- The role that a word with multiple senses plays in a sentence or phrase should also provide constraint. In particular, the slots that such a word sense must fill can provide constraint on what sense is meant.
- A more flexible approach to how restrictions are placed on a slot should provide some more constraint. In particular, the suggested extensions proposed in chapter 8 should provide more constraint on what concepts fill a slot.

The suggestions above if taken one at a time are probably not sufficient to provide enough constraint to reduce the number of possible interpretations. However, combining all of these suggestions should provide much more constraint than the current system can provide. Note that these suggestions assume a complete natural language understanding system including some methods to perform focusing and slot assignment similar to a case-frame system [Filmore 68].

6.15. The Possible Relationships Between Two Nouns

One interesting question is what are the possible ways that a relationship can be formed between two nouns within the NETL framework. The following list gives the ways a relationship between the nouns of a compound can be established within NETL. Note that only a few of these are currently implemented in the current system.

- **The first noun fills a slot of the second.** This is the most common relationship between two nouns and is currently implemented. An example of such a compound is "truck driver".
- **The second noun fills a slot of the first.** This is an infrequently used relationship and is not currently implemented. An example of such a compound is "output sentence".
- **The first noun is a descendant of the second noun.** This relationship occurs infrequently and is not currently implemented. An example of such a compound is "baseball game".
- **The first noun is an ancestor of the second noun.** This relationship occurs infrequently and is not currently implemented. An example of such a compound is "the integer five".
- **The first noun overrides a default assumption of the second noun.** This relationship occurs fairly infrequently and is currently implemented. An example of such a compound is "woman doctor". Note that this relationship involves accessing relationships provided by split statements in NETL.
- **The two nouns fill a slot of a third concept.** For example, in the compound "diesel particulates", "diesel fuel" is burned to form "particulates". This relationship occurs infrequently and is not currently implemented.
- **The first noun fills a second level slot of the second noun.** For example, the compound "oil executive" should probably be formed by finding the concept representing company,

and then finding the "produces/provides" slot of company and filling it with "oil". Note that the "company" is left unspecified and can be filled in later if necessary or if the name of the company is found later. Note also that this may have to be extended to a third level of slots or even further.

- The compound is idiomatic. No relationship exists in the data base. This way of forming a compound occurs fairly infrequently.

The above relationships can be formed in NETL's data base and seem to account for a large number of noun compounds.

6.16. Summary

This section has tried to give some idea of how well the system would perform on real-world examples by simulating the behaviour of the system on compounds found in various articles. It performs well on about 54% of the approximately six hundred examples processed by hand. Another 10% are processed correctly using various heuristics, especially the ones based on context. If certain assumptions are made about the information available to the system, its performance can be increased significantly from 64%. A summary of these assumptions are as follows:

- Some compounds seem to require slots that are of a questionable nature. If most of the compounds put into this class could be processed assuming the existence of these questionable slots where it seems appropriate, the system's performance should be enhanced by about 10%.
- Certain compounds have no meaning found. Many of these should probably be considered as cognates. If about three-quarters are considered as cognates, then the system's performance would improve by about 7%.
- The current system would produce the wrong meaning for some compounds. Some of these compounds should be considered as cognates. This assumption would improve the system's performance by about 3%.
- Using ellipsis processing as proposed in this section should improve the system's performance by at least 3% and possibly much more.
- Certain patterns do not currently exist in the system. If the system was extended to handle these patterns, its performance would improve by about 3%.

With the assumptions made above, the system's performance can be improved to around 90% with relatively little effort. This leaves approximately 10% of the compounds found in real-world examples that the current system has no chance to process correctly. What kinds of mechanisms are needed for the current system to have at least some possibility of arriving at a reasonable meaning for

a compound? There seem to be a few areas where work needs to be done for the current system to be able to process most of the 10% of the compounds that cause major problems. These areas are explored further in the following paragraphs.

A more general association mechanism would allow the system to find some meanings that it currently does not. Such a mechanism would help in at least two ways: it would allow the ellipsis processing to be more general by priming the network for appropriate associations between concepts, it may also allow the system to find associations between concepts that it would not normally find. In some cases these "deeper" associations may be the basis for the correct interpretation for a compound.

Some of compounds have a metaphorical meaning that is impossible for the current system to find because of its reliance on the very rigid structure of NETL. The type of processing that could possibly lead to finding meanings for at least some of these compound could come from using more general association mechanisms similar to the ones mentioned in the paragraph above. Many of the metaphorical meanings for compounds have a basis in some tenuous relationship that exists between the nouns in the compound. The current scheme has a very rigid idea about what relationships must exist between compounds for such a relationship to form the basis for such an interpretation, and thus will not find these much deeper relationships on which many metaphorical types of meanings seem to rely.

The system uses local context fairly effectively in many cases to disambiguate the meaning of words in a compound as well as relationships that can exist between words in a compound. However, there are cases where the local context suggests an interpretation for a compound that is not one that the system would normally consider. There seem to be two possible problems arising in this area. One is where a slot is meant to be filled by something which can not normally fill the slot. The context should be sufficient to overrule this slot restriction and allow the slot to be filled by a concept that would not normally be allowed to fill the slot. The other problem arises when the system would find an appropriate slot, but overrules it because the system finds other slots that it feels are better suited as the basis for forming an interpretation of the compound. The phrase level context is not sufficient to provide the correct answer, it is often somewhat more global context (i.e., possibly the previous sentence) as well as an understanding of the processes being discussed that is beyond the understanding of the current program.

In the case of a few compounds, the relationship between the concepts is sometimes dictated by a

process that involves the concepts. Being able to connect the concepts involved in the compound to the associated process often leads to the correct meaning of a compound. For example in the compounds "diesel particulates" and "diesel fumes", there is a "burning" process in which diesel fuel is burnt and the result is particulates and fumes. The current system doesn't contain any method to look for such processes and try to establish a meaning for the compound based on them. The interesting point here is that people seem to be able to focus in on such processes fairly easily and thus have a much better understanding of such compounds than the approach taken in the current system.

The areas discussed above seem to be the major areas that would allow the current program to correctly process more compounds than it currently does. These areas seem to be the ones that people are able to process relatively easily most of the time. The current program, however, does not address the issues discussed above. It takes a much more rigid approach to the problems raised above than a solution to them seems to require.

Chapter 7

Results

This chapter discusses several topics regarding the results the program produces. Section 7.1 describes the results produced by the program itself, i.e., the meanings of the compounds it produces, and how they can be evaluated. Section 7.2 describes how useful the various heuristics seem to be in selecting the meaning of a compound. Section 7.3 gives an indication of how much coverage the program has, i.e., how well it performs on examples which were not used to debug the program. Section 7.4 discusses some timing issues with regards to how fast the program runs, both in the simulated version as it is now, and on one possible architecture for a parallel NETL machine.

7.1. Results of the Program

An important issue is how to judge whether the program has produced a valid meaning for a compound. There are several ways for the program to show that it "understood" the compound. One is to form an English phrase or sentence that fills in the information that was omitted when the compound was formed. For example, the meaning for the compound "dog owner" could be paraphrased as "an owner that owns a dog". However, this leads into the problem of text generation from an underlying semantic representation, which is a research problem in itself [Goldman 74, Winograd 72, Simmons 72]. It was decided not to add this problem to the tasks necessary to understand noun compounds, since it would have directed a large amount of effort to a problem that is interesting but not directly related to this thesis. Another option, and the one that the current program uses, is to create the structure necessary to represent the meaning of a compound. By examining the created structure, it is possible to determine whether the program "understood" the compound. The structures created in the NETL network can be fairly complex, as seen in section 3.4.1. Of course, it is awkward to examine the structure created, so the program prints out a description of the structure created, without any attempt to try to generate English text to paraphrase the meaning. Thus the results of the program are a description of the interpretations created in the NETL data base.

For example, consider the results for the relatively simple compound "wine glass", as shown below:

```
(meaning '(wine glass). nil)
There is 1 compound after finding all possibilities.
```

```
Creating compound: WINE GLASS
WINE GLASS[&WINE GLASS-A] is a GLASS[&GLASS-B].
GLASS-B|CONTAINEE: WINE[&WINE].
```

The example above shows what is printed out by the program when it is given the task of finding the meaning of the compound "wine glass". The first line is the invocation of the function "meaning" which is the top-level function of the program to understand noun compounds. The first argument provides the list of words in the proper order that make up the compound. The second argument is used to provide context to the program. In this example, NIL is used, which means that no context is to be used by the program to understand the compound "wine glass". The second line states that only one possible interpretation for the compound is found by the program. The third line means that the program is creating a meaning of the compound in NETL's data base. The fourth line states that a "wine glass" is a "glass". The phrases in square brackets are the names of the NETL nodes that represent the concepts before them. For example, the node "&wine glass-A" represents this particular meaning of the compound "wine glass". Similarly, the node "&glass-B" represents a particular meaning of "glass", in this case, a container (there is also the material made from silicon dioxide). The last line of the example gives the name of the role that the noun "wine" fills (i.e., "glass-B|containee" which describes the things that a glass as a container can contain). The meaning of "wine" is represented by the node with name "&wine", since only one meaning for the word "wine" exists. This structure provides all the information that is needed to "understand" the compound "wine glass", i.e., that the two objects involved are a "glass" that normally contains liquids, and the liquid "wine", and also that there is a containment relation in which the "wine" is contained in the "glass".

As a more complex example, consider the compound "water meter cover adjustment screw". The following description of the structure built to represent the meaning of this compound is printed by the program:

(meaning '(water meter cover adjustment screw) nil)
There is 1 compound after finding all possibilities.

Creating compound: WATER METER COVER ADJUSTMENT SCREW
 WATER METER[&WATER METER-A] is a METER[&METER].
 METER!MEASURES: WATER[&WATER]
 WATER METER COVER[&WATER METER COVER-A] is a COVER[&COVER].
 COVER!COVERS: WATER METER[&WATER METER-A]
 WATER METER COVER ADJUSTMENT SCREW[&WATER METER COVER
 ADJUSTMENT SCREW-A] is a SCREW[&SCREW].
 &ADJUST-1
 ADJUST!TOOL-OF-&ADJUST-1: SCREW[&SCREW]
 ADJUST!ADJUSTEE-OF-&ADJUST-1:
 WATER METER COVER[&WATER METER COVER-A]

The first line of the example is the invocation of the program with the compound "water meter cover adjustment screw". The second line just states that there is only one interpretation for the compound. The third line states that the program is going to create the representation for an interpretation of the compound in NETL's data base. The fourth and fifth lines present the representation for the meaning of the sub-compound "water meter", i.e., a meter that measures water. The sixth and seventh lines present the representation for the meaning of the sub-compound "water meter cover", i.e., a cover that covers a water meter. The eighth and ninth lines state that a "water meter cover adjustment screw" is a "screw". The last four lines say that a "water meter cover adjustment screw" is a screw used to adjust a "water meter cover". This is an example of the representation used for a compound containing a nominalized verb in the middle of the compound (i.e., adjustment). The tenth line says that there is an adjustment action that is being described. The next three lines say that the tool used in the adjustment is a screw, and that the thing being adjusted is a "water meter cover". Thus the structure resulting from forming an interpretation for a compound can be quite complex. Note that there are two sub-compounds that must be created before the meaning of the top level one can be created.

Heuristics are necessary in many instances to allow the system to disambiguate a compound. The effectiveness of the various heuristics is very dependent on the information used by the heuristic. The context heuristic is very effective, because it uses information about what occurred around a compound to help disambiguate it. For example, consider the compound "golf club" when no context is provided to the program. In such a case, the program creates the following two interpretations for the compound:

(meaning '(golf club) nil)
There are 2 compounds after finding all possibilities.

Creating compound: GOLF CLUB
GOLF CLUB[&GOLF CLUB-A] is a CLUB[&CLUB-B].
CLUB-BIUSED-IN: GOLF[&GOLF]

Creating compound: GOLF CLUB
GOLF CLUB[&GOLF CLUB-B] is a CLUB[&CLUB-A].
CLUB-AIPURPOSE: GOLF[&GOLF]

The program creates the two possible meanings for the compound, i.e., a club that is a stick used in the game of golf, and an organization whose purpose is golf. The program can be provided with context corresponding to the following two sentences:

1. The golf club hit the ball.
2. The golf club fired its manager.

When the program is provided with the context provided by the two sentences above, it produces the following results:

(meaning '(golf club) .
'(((hit (hit|hitter hit|with)
(hit|hitee &ball-1))))))
There are 2 compounds after finding all possibilities.

Creating compound: GOLF CLUB
Using cognate: &GOLF CLUB-A
GOLF CLUB[&GOLF CLUB-A] is a CLUB[&CLUB-B].
CLUB-BIUSED-IN: GOLF[&GOLF]

(meaning '(golf club)
'(((fire (fire|firer)
(fire|firer &manager-1))))))
There are 2 compounds after finding all possibilities.

Creating compound: GOLF CLUB
Using cognate: &GOLF CLUB-B
GOLF CLUB[&GOLF CLUB-B] is a CLUB[&CLUB-A].
CLUB-AIPURPOSE: GOLF[&GOLF]

In the first case, the program produces only the interpretation where a golf club is a stick used in the game of golf. Note that the second argument in the call to the function "meaning" provides the context by specifying a hit action where the thing hit is a ball. The definition of "hit" includes information about what things can be hit, who can hit them, and what things can be used to hit them. The program finds that the role representing the thing used to hit other things can be filled by the stick form of "club", and thus supports that meaning of "golf club". None of these roles can be filled by the organizational meaning of "golf club". In the second case, the program produces the

interpretation where it means an organization whose purpose is golf. Note that the program still produced two possible interpretations in both examples. However, it kept one or the other because the context provided support only for the one kept.

Another example where context can be used is for the compound "computer maintenance". The program finds two possible meanings for the compound when no context is provided, as shown below:

```
(meaning '(computer maintenance) nil)
There are 2 compounds after finding all possibilities.

Creating compound: COMPUTER MAINTENANCE
COMPUTER MAINTENANCE[&COMPUTER MAINTENANCE-A]
is a MAINTENANCE[&MAINTENANCE].
MAINTENANCE!MAINTAINER: COMPUTER[&COMPUTER]

Creating compound: COMPUTER MAINTENANCE
COMPUTER MAINTENANCE[&COMPUTER MAINTENANCE-B]
is a MAINTENANCE[&MAINTENANCE].
MAINTENANCE!MAINTAINEE: COMPUTER[&COMPUTER]
```

The first interpretation of "computer maintenance" describes a maintenance action where a computer is the thing doing the maintenance. The second interpretation describes a maintenance action where the computer is the thing maintained. To cause the first meaning to be produced, context of the form "computer maintenance of a data base" can be provided to the program, as in the following example:

```
(meaning '(computer maintenance)
'((( &maintenance ()
(maintenance!maintainee &data/ base-1))))))
There are 2 compounds after finding all possibilities.

Creating compound: COMPUTER MAINTENANCE
Using cognate: &COMPUTER MAINTENANCE-A
COMPUTER MAINTENANCE[&COMPUTER MAINTENANCE-A]
is a MAINTENANCE[&MAINTENANCE].
MAINTENANCE!MAINTAINER: COMPUTER[&COMPUTER]
```

In this example, only the meaning where a computer is the thing doing the maintenance is produced, even though both interpretations are considered. The reason that the meaning where a computer is the thing maintained is not chosen is because the slot that computer would fill to obtain this meaning is already filled by the node "&data base-1" which represents a particular instance of a data base. Note that the program prints out a message of the form "using cognate:". This is because there is already the node "&computer maintenance-A" in NETL's data base that represents this meaning of computer maintenance from the previous example.

If no context is supplied to the program, the meaning where a computer is the thing being maintained is probably preferable to the one where a computer is the thing doing the maintenance. If there was more data in the data base, the similar compounds heuristic can come into play and provide enough support for this meaning, allowing the other one to be ignored. If the meaning where a computer performs a maintenance action is the one that is really wanted, context has to be provided to the system to obtain the correct meaning. For example, the meanings for the compounds "car maintenance" and "truck maintenance" can be constructed by the program and added to NETL's data base, as follows:

```
(meaning '(car maintenance) nil)
There is 1 compound after finding all possibilities.

Creating compound: CAR MAINTENANCE
CAR MAINTENANCE[&CAR MAINTENANCE-A]
    is a MAINTENANCE[&MAINTENANCE].
MAINTENANCE!MAINTAINEE: CAR[&CAR]

(meaning '(truck maintenance) nil)
There is 1 compound after finding all possibilities.

Creating compound: TRUCK MAINTENANCE
TRUCK MAINTENANCE[&TRUCK MAINTENANCE-A].
    is a MAINTENANCE[&MAINTENANCE].
MAINTENANCE!MAINTAINEE: TRUCK[&TRUCK]
```

At this point, there is sufficient information in the network to cause the program to select the meaning where a computer is the thing maintained, rather than the thing doing the maintenance. The following example shows the similar compounds heuristic in action:

```
(meaning '(computer maintenance) nil)
There are 2 compounds after finding all possibilities.

Creating compound: COMPUTER MAINTENANCE
Using cognate: &COMPUTER MAINTENANCE-B
COMPUTER MAINTENANCE[&COMPUTER MAINTENANCE-B]
    is a MAINTENANCE[&MAINTENANCE].
MAINTENANCE!MAINTAINEE: COMPUTER[&COMPUTER]
```

Again, both possible interpretations are found, but only the one where a computer is the thing maintained is selected, because the similar compound heuristic finds the meanings of the compounds "car maintenance" and "truck maintenance" which are similar to the interpretation of "computer maintenance" where a computer is the thing maintained. If NETL's data base contains a large number of meanings of compounds, the similar compounds heuristic becomes a very powerful tool in selecting the best interpretation.

Appendix A shows the program processing several examples. The appendix includes the above examples with a tracing of what the program does as it processes them.

7.2. Effectiveness of the Heuristics

The various heuristics have varying degrees effectiveness when it comes to deciding which interpretation is the best one. Some heuristics are more effective than others. This section evaluates the effectiveness of the various heuristics.

7.2.1. Context

The three heuristics based on context seem to be some of the most effective heuristics for deciding which interpretation of a compound is the best one. This sub-section evaluates the effectiveness of each of the three heuristics that use context to select the best interpretation for a compound. The next three sub-sections look at each of the individual context heuristics in more detail.

7.2.1.1. Selectional Restrictions

Selectional restrictions, such as "golf club" with the restriction provided by the phrase "the golf club hit the ball", can provide support for one or more of the possible interpretations of a compound. This support comes from information that is relatively local to the compound, i.e., usually from the sentence or phrase in which the compound occurred. This information is usually very specific and can provide a large amount of constraint for the meaning of the compound. The one disadvantage of this heuristic is that it is only useful for disambiguating the head noun (i.e., last noun) of a compound. Thus, this heuristic can come into play only when the head noun of a compound is ambiguous. It provides no constraint if some other noun in the compound is ambiguous. Thus, although it is extremely effective, its application is restricted to a relatively few compounds. From the experiment described in chapter 6, about 3% of the approximately six hundred compounds could be disambiguated correctly using the "selectional restriction" heuristic as described in chapter 4. Note that this heuristic can only provide support for an interpretation of a compound, it can not directly rule out an interpretation for a compound the way the following heuristic can.

7.2.1.2. Slot Already Filled

The "slot already filled" heuristic is another heuristic based on context that is very effective. It has the potential of being able to rule out an interpretation for a compound, such as for the compound "computer maintenance" with the constraint provided by the phrase "computer maintenance of a data base". This means subsequent heuristics do not have to examine as many possible interpretations to make their decision. However, this heuristic is only applicable when the meaning

of the head noun of an interpretation has more than one slot used as the basis for an interpretation. It is not likely to help disambiguate a compound in which an interior noun has a meaning with more than one slot that provides the basis for a possible interpretation. Again, this heuristic is very effective in ruling out possible interpretations for a compound. However, it can only be applied to a few compounds, since it generally deals with only the last noun of a compound. From the experiment described in chapter 6, about 2% of the approximately six hundred compounds are disambiguated correctly using this heuristic. Note that this is one of the few heuristics that can prune a possible interpretation and be almost certainly correct.

7.2.1.3. General Topic

The general topic under discussion heuristic is a context heuristic that can only provide a preference for a compound, it can not directly rule out an interpretation. This heuristic is the only context heuristic that can help disambiguate compounds containing interior nouns with multiple meanings. This is because the heuristic is providing a very loose form of constraint, by trying to simulate the effects of a discussion with a few key concepts. Again, this heuristic is very effective in deciding which interpretation is best. It is more general than the two types of context mentioned above, but it is probably less accurate than them. It is a fairly general heuristic, and thus has the potential of selecting a larger number of possible interpretations. In the experiment described in chapter 6 this heuristic disambiguated about 3% of over six hundred compounds. This figure is probably a low estimate of how effective this heuristic could be. A more general association mechanism than the one used for this heuristic could help with at least another 3% of the compounds.

7.2.2. Cognate and Embedded Cognate Heuristics

The two cognate heuristics (i.e., the cognate heuristic and the embedded cognate heuristic) have the problem that once all the meanings for a compound have been created in NEIL's data base, they no longer provide any indication of which interpretation is the best one. Thus, they are of limited usefulness, and will come into play in the selection process only when a few of the interpretations already exist in NEIL's data base. Not all the possible meanings may exist in the network, if the first few times a compound is encountered, the heuristics are able to remove some of the possible interpretations from consideration. If all the interpretations already exist, then these heuristics will support all of them equally. Although these two heuristics are very useful for selecting the best meaning for a compound, their usefulness is likely to be for a short duration for any particular compound, until the first exceptional case is seen and stored. If there are unlikely to be no

exceptional case, as in about 10% of the compounds found in the experiment described in chapter 6, then idiomatic or "conventionalized" cognates play a very important role in selecting correct meanings for compounds.

7.2.3. Instances and Embedded Instances of Interpretations

The instances of an interpretation heuristic and the embedded instances of an interpretation heuristic may give a preference for certain interpretations. However, it is difficult to test these heuristics to any degree of certainty, because it is very easy to bias the experiment in favour of a particular interpretation without even noticing the biasing effect. The only way that the program can use these heuristics is if it is provided with instances of an interpretation. There is no way, without the possibility of biasing the experiment, that it can gather the information, the way a person might, to allow these heuristics to come into play. Certainly, examples can be shown where it is reasonable for these heuristics to come into play, but there is no method to guarantee that some of the other interpretations could have supporting evidence of the same nature. On a few occasions during the experiment described in chapter 6, these heuristics could come into play. It is hard to put an exact figure on how many compound would be disambiguated using these heuristics, but they could help in a large number of the ambiguous cases. These heuristics do have the advantage that they will continue to be useful as long as the number of instances for each possible interpretation of a compound remain somewhat different throughout a run of the program.

7.2.4. Slot Verification Heuristic

The slot verification heuristic is a heuristic that evaluates how well a concept fills a slot. This evaluation is sometimes necessary because the algorithm used to find the possible interpretations for a compound finds all the slots into which a concept can fit. It only checks to make sure a concept fits into a slot, not how well it fits. This heuristic seems to come into play relatively infrequently, since it is only used when more than one possible interpretation is found. When it does come into play, it provides a very good indication of which interpretation is the best one. This heuristic can be applied to each slot that is the basis for an interpretation, whether the interpretation is for a sub-compound or for the top-level compound.

7.2.5. Similar Compounds

The similar compounds heuristic can provide very useful constraints to the problem of finding the best interpretation for a compound. It has the disadvantage that it is only effective after the system has seen several compounds and has stored their meanings in NETL's data base. When the system first starts up, this heuristic plays a very small role in the selection of the best interpretation for a compound, since there are few if any meanings of compounds in NETL's data base. However, as the system stores more and more meanings in NETL's data base this heuristic becomes much more useful. A few times during the experiment described in chapter 6 this heuristic could be used to help disambiguate a compound. It can help disambiguate a large number of ambiguous compounds if the data base contains sufficient information.

7.2.6. Occurrence Heuristic

The occurrence heuristic attempts to make up for the binary nature of the cognate and embedded cognate heuristics. However, it can not be considered an overly reliable heuristic. If one possible interpretation has been seen more often than another one, it is still quite possible that the other one is the correct interpretation to use. Thus this heuristic only gives a slight preference for a particular interpretation. This heuristic was never considered during the experiment in chapter 6 since all the compounds in the articles were used consistently throughout. Other heuristics, such as the context heuristic or the similar compounds heuristic are allowed to overrule the occurrence heuristic.

7.3. Completeness

One problem that is hard to address with a program of this nature is how well it covers all the possible meanings for a compound. The data base is hand built, and thus may have been unknowingly biased to select particular meanings for a compound. There is no easy answer to this problem. One method to attempt to address this problem is to allow a naive user to play with the system and see how often he breaks it. Breaking it means for the program to produce incorrect interpretations or to miss interpretations for compounds for which the data base contains enough information so that it should be able to produce the correct result.

Another approach to this problem is the one taken in chapter 6. In this approach a large number of compounds are examined to see how well the program would handle them. Chapter 6 gives details of how this experiment was carried out and the results. The program as it currently stands can

process about 60% of the compounds encountered in real-world text. The program requires access to the context in which a compound occurs to process about 8% of these compounds. Another approximately 30% of the compounds can be processed correctly if some reasonable assumptions are made and if a couple of new patterns are added to the system. The remaining 10% of the compound can not be processed by the paradigm presented in this thesis. They seem to need more sophisticated processing using pragmatic information as well as semantic information.

7.4. Program Timings

The program currently runs under MacLisp on a DEC PDP 20 model 2060. It uses about 215K words of storage when it is started up. The compiled program uses about 14K words of storage. A simulator for the NETL machine also runs under MacLisp and is used by the program as its underlying knowledge representation system. The NETL simulator takes up another 12K words of storage. Another 35K words are used to store knowledge for about two hundred English words. A further 70K is used for working storage. The rest of the space is taken up by the MacLisp system and utility functions. There are two types of timings that must be considered for this program. One is how much CPU time it takes to process a compound. The other is how many cycles of a possible NETL machine are necessary to process a compound.

The amount of time the program takes to process a compound using the simulator for NETL is somewhat misleading. The search algorithms used to process the meaning of a compound rely on the fact that NETL is meant to run on a parallel machine. This means they do things that are cheap on a parallel machine but more expensive on a serial one, such as scanning up or down VC links from any node. With a parallel implementation of NETL, such a scan takes an amount of time proportional to the longest chain of VC links. With a serial implementation, as with the current simulator, such a search takes time proportional to the number of nodes encountered. For example, the scan used to mark all the slots that a node can not fill (see section 3.1.1 for details of this scan) uses an algorithm that is likely to look at a large number of nodes because it first does an upward scan marking all the ancestors of one node. It then steps across split links twice marking possibly many different nodes. Finally, a downward scan from all these newly marked nodes is done. In a parallel implementation, this search takes time proportional to the longest VC chain, i.e., about twice as long as an upward or downward activation scan. This search has the potential of encountering more than half the nodes in the data base, and thus can be very expensive in a serial implementation. This means that the amount of knowledge stored in the network must be kept relatively small, otherwise it would take a long time for the program to process a compound. Currently, the initial data base used by the program has

approximately fifteen hundred nodes that are used to represent the meanings of about two hundred words plus other concepts that are not directly mapped into a single word. As the system processes compounds, the number of nodes in the system increases as meanings for compounds are added to this data base. Even with the small number of words stored in the data base, the system uses up over eighty percent of the address space available on a PDP 20, before it has processed any compounds. A fair amount of this storage is used as working storage, but without a fairly large amount of working storage, garbage collection would slow the system down to an unacceptable level.

The amount of CPU time it takes to process a compound depends on the compound. If the compound is simple, i.e., a compound with two nouns and a single meaning (e.g., wine glass), the program takes from two to five seconds of CPU time. If the compound contains several words, but still only has one meaning (e.g., water meter cover adjustment screw), the program takes about five to ten seconds of CPU time to process it. If the compound, has multiple meanings (e.g. golf club), then the amount of CPU time can increase to as much as about fifteen seconds. For long compounds that are ambiguous, the program can take even more time. One compound (car assembly plant) takes over thirty seconds of CPU time to be processed. The CPU time is very dependent on how much information is available to the program. If the program, can make a decision as to which meaning is the best one by using context or cognates, then the CPU time used will be significantly reduced than if it has to use the extremely costly similar compounds heuristic. For example, without using context with the compound "golf club", the program takes about twelve seconds of CPU time to produce two meanings and to rate them as equally good. However, using context that is sufficient to select only one of the meanings, the program takes about four seconds to arrive at the correct one. These times will vary depending upon how many garbage collections take place during the processing. Each garbage collection seems to cost somewhat more than a second of CPU time. In the more complex cases, several garbage collections are likely to occur, and thus will add a significant amount of time to that needed to actually discover the meanings for a compound.

Another measure that can be used to determine the amount of processing needed to process a noun compound is the number of cycles that a parallel NETL machine would need. As part of the simulator that runs on a PDP 10, a crude estimate of the number of cycles a NETL machine needs to perform searches is kept. It is possible to make a rough estimate of how much time it would take to process a noun compound using these estimates. One proposal for building a NETL machine [Fahlman 80] suggests that it is possible to build a million element (nodes and links) NETL machine

that has a cycle time of about one milli-second¹. This design is not fully parallel, in that a significant amount of time sharing is done so that the cost of the machine is not prohibitive. However, using the assumption of a one milli-second cycle time, and the estimate of the number of cycles a search takes from the simulator, it is possible to make an estimate of how long it would take to process a compound.

In the simple case, i.e., where there are only two nouns and a single meaning, the NETL simulator estimates that it would take about four hundred cycles. This translates into less than half a second of real time using a NETL machine. For a compound containing several nouns (e.g., water meter cover adjustment screw), the number of cycles is about seventeen thousand, which translates into about seventeen seconds of real time on the NETL machine. Over half of this time is spent creating the compound. This is because it must search the network to find the verb associated with the noun "adjustment". This is necessary so that the correct structure is built in the network for the compound "water meter cover adjustment screw", since it has a nominalized verb in the middle of the compound. Compounds that have multiple meanings, such as "golf club", seem to take about thirteen hundred cycles, or about 1.3 seconds on the NETL machine. In the case, of "golf club", if context is provided that disambiguates it, the number of cycles used drops by about seven hundred. These estimates do not include any time used by the serial machine used to drive the parallel NETL machine. The amount of actual time used would be somewhat higher, although, most of the work goes into processing the searches. Also note that no matter how many nodes or links exist in the NETL machine, these times should remain fairly constant, since the number of cycles does not depend on the number of nodes or links in the network, only the length of the longest path. It is possible to increase the number of cycles by increasing the length of paths that occur in the data base. These increases should be fairly small, and should under no circumstances make as much difference as an order of magnitude.

So that the results of the two types of timings can be compared, table 7-1 shows the approximate times of some of the compounds the system is able to process successfully. The column labelled "CPU time" gives the time the program took on a PDP-20 including the time if any used in garbage collections. Unfortunately it is difficult to get an estimate of the time used excluding the amount of time used in garbage collections. The column labelled "NETL machine time" is an estimate of the amount of time a NETL machine having a one milli-second cycle time would take to process the

¹Actually, [Fahlman 80] uses a cycle time of about ten seconds due to the use of commercially available 64K rams. If custom designed chips are used, the one milli-second cycle can be achieved.

same compound. These timing are only meant to give an indication of the performance of the program and may vary somewhat depending on various factors. The most important aspect of these times is how they grow with the size of the data base. CPU time should grow roughly proportionally to the total amount of knowledge, whereas NETL machine time should grow very slowly. As more knowledge is added to the network, the length of the longest path between concepts may increase by at most a factor of two. The NETL machine times assume the use of a simple network that does not contain all the mechanisms that are necessary for a more complex data base. Another factor of two is necessary for NETL to be able to process cancellation links in the type hierarchy. Another factor of two is probably needed for more complex matching in the network. With these additional mechanisms and a much larger data base, the NETL machine time should only increase by a factor of six to eight.

<u>Compound</u>	<u>CPU Time</u>	<u>NETL Machine Time</u>
Wine Glass	3 secs.	0.4 secs.
Water Meter Cover Adjustment Screw	8 secs.	17 secs.
Golf Club (no context)	13 secs.	1.3 secs.
Golf Club (The golf club hit the ball)	4 secs.	0.7 secs.
Golf Club (The golf club fired its manager)	4 secs.	0.6 secs.

Table 7-1: Table of Comparative Timings for Simulator and NETL Machine

Chapter 8

Conclusions

This chapter presents several conclusions from the research described in the previous chapters. The answers to the following questions form the basis for the conclusions from this work:

- **How hard is it to extend the program to process new compounds?** Since much of the knowledge used to process compounds is contained in NETL's data base, in most cases it should only be necessary to add new knowledge to the data base to have the program process new compounds. No new procedures should have to be written to handle new compounds. It will be necessary to run the program on any new compounds because the data base is likely to contain errors.
- **Is a real-world knowledge base required?** One of the problems with understanding noun compounds is to reconstruct information that is omitted when the compound was formed. To reconstruct this information it is sometimes necessary to perform complex inferences about the concepts involved. To do this effectively requires a data base containing real-world knowledge.
- **What features of NETL are necessary to process noun compounds?** NETL provides several facilities the program to understand noun compounds uses to its advantage. Other knowledge representation systems could be used as the underlying representation scheme, but some searches would have to be done differently.
- **How well does the program model the way people process noun compounds?** It probably does not model the way people process compounds very well in the current implementation. It does things that people do not seem to.
- **Does the program learn from experience?** The "similar compounds" heuristic, the "cognate" heuristics, and the "instances of an interpretation" heuristics provide a simple but effective means of learning. Since the meanings of new compounds are added to the data base, these heuristics use this information to help disambiguate subsequent compounds.
- **How well does the program deal with ambiguity?** The program seems to be able to cope with some forms of ambiguity that any natural language system must cope with.
- **How hard would it be to interface the program to a parser?** Interfacing the program as it currently stands to a parser is probably the wrong way to proceed. As discussed in chapter 5, it should be possible to use the ideas presented in this thesis to help a parser disambiguate noun compounds.

The following sections present these conclusions in more detail. The last section of this chapter describes some areas for future research.

8.1. Extending the Program to Process New Noun Compounds

One of the goals of this research from the start was to develop a small set of rules or patterns that could be used to understand noun compounds. This goal, to a large extent, has been satisfied. There are a few patterns into which a compound can be "parsed". There are somewhat more rules (heuristics) necessary to actually find the best interpretation for ambiguous compounds. The important thing is whether it is necessary to add many more rules to the system for it to be able to understand new compounds. In most cases, adding knowledge about the words making up a new compound should be sufficient to allow the system to produce the correct meaning (or meanings) for the compound. This means it should be fairly easy to extend the system by adding new declarative knowledge, since it should not be necessary to add new procedures for the system to process new compounds. As shown in chapter 6, there are at least two patterns for noun compounds the current program does not process correctly, i.e., when the head noun fills a slot of a previous noun and when a group of related concepts fill the same slot in the head noun. Also, about 10% of the compound do not fit the paradigm of one noun filling a slot of another noun in the compound. These compounds seem to be formed by one of several processes:

- A more general relationship exists between the nouns.
- The compound is formed by using a metaphorical meaning for one of the nouns.
- Local context suggests an interpretation that would not normally be considered.
- The relationship between the nouns is based on some process that occurs between them.

With these additional capabilities, the system should be able to process almost all the compound it would come across in real-world text.

The approach taken to the problem of understanding noun compounds by this thesis is significantly different than that of most linguists, but not all. Much of the linguistic literature [Hatcher 65, Jespersen 46, Lees 60, Lees 70, Marchand 69] attempts to determine a fixed set of underlying relationships that can be used to produce noun compounds. The current system also tries to find underlying relationships between the nouns of a compound. However, it does not explicitly rely on a fixed set of relationships as does much of the work of linguists. At any point in time, there is only a fixed set of relationships that the system can use as defined by the relationships existing in

NETL's data base. However, if a new relationship is introduced into the data base, the system is able to use it immediately to construct meanings for compounds without the necessity of changing any of its internal structure. Thus, the system is limited to the relationships existing in the data base, not to some arbitrary fixed set of relationships.

8.2. Need for Real-Word Knowledge to Understand Noun Compounds

The system to understand noun compounds uses the NETL representation system to represent its knowledge about the meanings of words, as well as, other real-world knowledge. It is unlikely that any system to understand noun compounds could understand them without the use of such knowledge. In some cases, it is necessary to use the context in which a compound occurs to be able to disambiguate its meaning. A purely syntactic approach to understanding noun compounds could not possibly be able to produce a meaning for such compounds, even though syntax does help decide what role a compound can play in a sentence. Part of the motivation for this research was to design a system that could be integrated with a natural language system. The important aspect is that during the parsing of a phrase, it is necessary to have access to semantic knowledge, as well as syntactic knowledge. In many cases, this should produce fewer possible paths that a parser has to follow to process a phrase.

8.3. Features used in NETL and their Necessity

The program to understand noun compounds has used some of the features of NETL to its advantage. For example, the parallelism provided by NETL allows certain types of searches to be processed very quickly. One instance of this is the search to find the slots that can not be filled by a concept. This search involves processing a large number of nodes in the network. If the parallelism provided by NETL is not available, it may be cheaper to use the simpler search of placing a concept in a slot and finding out whether it can fill the slot or not. Thus, there are certain searches that would be more expensive if NETL's parallelism is not available, or that would have to be done differently.

Another very important feature that NETL provides is the split statement. The program takes great advantage of this when it searches for possible interpretations for a compound. The split statement in NETL provides a very simple method to distinguish between several different conflicting concepts. For example, when a concept is to be placed into a slot, the system checks split statements to see if this concept would cause a clash if it is placed into the slot. If it does, the interpretation for a compound based on the concept filling this particular slot is rejected.

The features of NETL, in terms of knowledge representation, that the program uses can be provided by other representation systems, such as KRL [Bobrow 76] or the partitioned semantic networks of Hendrix [Hendrix 75]. In particular, the program could be rewritten using a different knowledge representation system. The important features for a knowledge representation system that is to be used for understanding noun compounds or other natural language phrases are:

- It must have a method to specify relationships between concepts, such as roles in NETL, relationships in semantic networks, slots in KRL. This is the most important requirement. The current system relies on being able to find relationships between the meanings of nouns to form the basis for an interpretation. Without this ability, it would be impossible to use the current paradigm to process noun compounds.
- It must have some method to distinguish between two or more classes of concepts. NETL provides split statements that do this efficiently. This ability is necessary so that it is easy to place restrictions on slots and then be able to quickly check to see if the restrictions are violated when a concept is placed in the slot.
- It should provide a hierarchy of concepts so that it is possible for concepts to inherit information from their superiors. Without a hierarchy of concepts, such as that provided by NETL, it is necessary to have a large amount of redundant information stored in the data base. Much of the redundant information can be inherited via a hierarchy. If a hierarchy of concepts is not used, every time a meaning for a compound is formed a large amount of knowledge would have to be copied into the newly created description.
- It must allow for ambiguity in word meanings, i.e., a many to many mapping between words and concepts is necessary. This is an important requirement for any knowledge representation system that is going to be used as part of a natural language understanding system. A word may have meanings represented by many different concepts. Similarly, a concept may be named by many different words.
- To increase the number of compounds the current system can process, it is necessary to add new knowledge to the data base. It is easier to add new knowledge to a declarative data base than one that is not declarative. Also, with a declarative data base there should be no need to change any of the search routines. As mentioned in section 8.1, there are some mechanisms that have to be added to the current program for it to process more compounds than it currently can.
- The program to understand noun compounds needs to make several searches in the data base to find the information it needs to form meanings for noun compounds. The underlying knowledge representation must allow for these complex searches and must be able to execute them efficiently.

Any knowledge representation system that has the facilities described above could be used as the basis for a program similar to the current one.

8.4. Modelling the Way People Process Noun Compounds

As shown in previous the chapter, the system produces meanings for some fairly complex compounds. When it has sufficient information, it is able to disambiguate many inherently ambiguous compounds. The question arises whether the model of understanding noun compounds described above provides any insight about how people perform. It seems that the current system, or even if it is moved to the NETL machine described above [Fahlman 80], runs slower than the time it seems to take a human to process similar compounds. It also does its processing somewhat differently. It carries along all the possible meanings for a compound as it scans the words of the compound left to right, and then processes these possibilities at the end to find the best one. Thus, the system never backs up, since if a particular meaning becomes invalid, it can continue with the meanings remaining. People, on the other hand, seem to have a particular meaning in mind as they process a phrase from left to right. When that meaning becomes invalid, they reconsider and try a different interpretation. The current system does not behave in this fashion, and thus can not be said to be a very good model of the way people process compounds. The design of the program was not based on any model of how people process noun compounds, and thus it is not surprising that it does not model the way people seem to process noun compounds.

8.5. Learning

The program uses some simple learning techniques to improve its performance. The learning the program does comes from the use of various heuristics. The following paragraphs describe in more detail the learning the program performs.

The "similar compounds" heuristic provides a simple but effective form of learning. As the system adds meanings of new compounds to NETL's data base, the "similar compounds" heuristic becomes more useful. When the system is first started up, it has very few meanings of compounds stored in its data base. As it processes compounds, it stores the meanings found in NETL's data base. These meanings can be used by the "similar compounds" heuristic to help disambiguate subsequent compounds.

Another form of learning is provided by the "instances" and "embedded instances of an interpretation" heuristics. The knowledge used by these heuristics comes from every day experiences where instances of objects are observed. These instances can then be used to guide the program to the best meaning of a compound. This knowledge is of a non-linguistic nature, but because it is

stored in the same representation system, it can be used by linguistic processes, such as understanding noun compounds.

The "cognate" heuristics also provides a simple ability for the program to learn. Once an interpretation has been stored in the data base, it will be selected over other interpretations for a compound. Thus, if there are several possible interpretations and only a few of them have cognates, this information will help the system select the best compound.

8.6. Dealing with Ambiguity

Ambiguity is a problem that any program dealing with natural language must address. The current program addresses issues related to some of the following kinds of ambiguity:

- One of the most common forms of ambiguity is caused by a word having two or more parts of speech. A word has a different meaning according to the part of speech it is used as. The current program does not address this problem.
- Another common form of ambiguity is caused by a word having multiple meanings for the same part of speech. A word used as a noun, for instance, may have several meanings. The program described above is able to cope with this problem to a large extent. The main problem seems to be having sufficient information to allow the program to disambiguate the meaning of a noun. This information may come from context, constraints provided by the structure of the compound itself, or other heuristics.
- Another form of ambiguity arises due to multiple relationships between underlying concepts. Syntax provides no clues when the concepts in a compound can be related by more than one relationship. The program seems to be able to cope with this problem to a large extent. Again, the main problem is having sufficient knowledge available to allow it to disambiguate which slot provides the basis for the best interpretation for a compound. This information may come from context, constraints provided by the structure of the compound, the existence of similar interpretations stored in the data base, and other heuristics.

These three types of ambiguity cause problems for any natural language processing system. Two of them have been dealt with fairly successfully by the current program.

8.7. Interfacing the Program to a Parser

As mentioned earlier, the program was designed with the possibility of eventually interfacing it to a parser. How hard would it be to actually do this? Chapter 5 gives a relatively simple design of how the ideas in the thesis could be interfaced to Parsifal [Marcus 80]. The program as it currently stands is not easily interfaced directly to Parsifal or any other. It is best to split the program into several parts, one part for each of the various patterns described in chapter 3, and one part that can finish processing the compound if the contextual information necessary to process the compound comes after the compound. All these parts would need to have access to the various heuristics and NETL's data base. Two other parsers, Diamond [Robinson 80] and Rus [Bobrow 78, Woods 79, Bobrow 80], could be interfaced to the system in a similar manner, though no details for these two parsers have been worked out.

8.8. Directions for Future Research

This section describes several areas where future research could be undertaken.

8.8.1. Extending the Coverage of the System

As discussed in chapter 6 and section 8.1 of this chapter, there are several compounds that can not be processed correctly by the current system. One direction for future research is to attempt to develop methods to deal with these kinds of compounds. Some of the new features should be fairly easy to add, such as allowing the head noun of a compound to fill a slot of the previous one or having a group of related concepts fill the same slot of the head noun.

The interesting problems arise when more complex pragmatic and semantic processing is necessary to process a compound. It seems that several new techniques need to be developed to process some compounds correctly. These techniques would involve allowing a more flexible method to find relationships between concepts, allowing for metaphorical uses of concepts, overriding restrictions on slots in some contexts, and allowing for the relationships between nouns to be specified by a complex process. If all these capabilities could be added successfully to the system, it would be able to process most of the compounds encountered in real-world text.

8.8.2. Increasing the Size of the Data Base

One direction for future research is to create a data base significantly larger (say by a factor of ten) than the current one which contains definitions for less than two hundred words. The interesting question here is whether the system would still be able to process the compounds as well as it currently does. With a bigger data base, it is much more likely that the information in it will interact in strange ways and possibly produce errors. Also, with a larger data base, it is more likely to have multiple word senses and this issue would have to be addressed more fully. Errors in a large data base would become a major concern and techniques would have to be developed to deal with them. One way to help is to have the data base make consistency checks as new information is added to the data base. The user could be asked to make corrections to information that is inconsistent with what is already in the data base. This should catch many of the errors that would be introduced into a large data base. The current implementation is close to its limit, since the address space of a PDP 20 is too small to hold the amount of information necessary. With the current implementation, it might be possible to double the number of words, but the system would very quickly run out of working space necessary to process compounds. Thus, to increase the size of the data base to a reasonable one, it would be necessary to either redesign the system so that a lot more space is available, or to move to a different machine that has a larger address space.

8.8.3. Extending the Paradigm to Other Phrases

The paradigm described in this research seems to work well for the problem of understanding noun compounds. Can it be extended to other types of phrases occurring in natural language? If it can be extended to other phrases, it might be possible to build a natural language understanding system that is extensible by adding new information to the data base, not by developing new rules or procedures. This means a significant number of procedures must be developed to access the NELL data base to form meanings for phrases and sentences.

As a simple test to see whether this is feasible, the system was extended to handle adjectives occurring before a compound, i.e., simple adjective phrases. This extension was fairly simple in nature, in that it used many of the mechanisms already developed for noun compounds. Adjectives are currently represented in NELL as a type node that represents the class of things that the adjective describes. For example, the colour red is represented as a type node that is the parent of all the red things. Other colours also have similar type nodes. The nodes for all the different colours are included in a split statement that says they are all distinct. This means that something that is red, can

not at the same time also be blue. Adjectives are allowed to modify the head nouns of compounds (including sub-compounds). To see if a concept can be modified by an adjective the system tests to see if the concept can be placed into the class of objects described by the adjective. If it is already in the class, then the information specified by the adjective is redundant. If it can not fit into the class, the system rejects that interpretation for the adjective phrase. This happens because a split statement somewhere detects a clash. It could be the split statement between colours if, for example, an attempt is made to modify something that is already blue with the adjective red. Or, if the object can not be modified by a colour. For example, "green ideas" would cause a clash, because only physical objects can normally be modified by colours, and an "idea" is not a physical object. If NETL detects no clash and does not already contain the information, the system creates an interpretation for the adjective phrase by creating a sub-type of the noun compound and adding it to the class of things described by the adjective. This is a fairly simple scheme to process adjectives. If it is to be used more extensively, it would have to be improved to handle more difficult cases. It may be possible to handle other types of phrases, such as verb phrases, using similar techniques. Appendix A contains a couple of examples showing how adjective phrases are processed.

8.8.4. Using Default Values for Slots

Slots in NETL may have restrictions placed on them by having a VC link pointing at the type node for the class of things the slot can accept. When a concept is placed into a slot, there are three possible outcomes:

- The concept can not fill the slot, because a split statement exists somewhere in the hierarchy between the ancestors of the concept the slot expects and the concept actually placed into the slot.
- The concept can be placed into the slot, but is not a member of the class of concepts that the slot expects.
- The concept can be placed into the slot, and is a member of the class of concepts that the slot expects.

Currently, this is the only information available to the program about the status of a slot with respect to a concept to be placed into it. The distinction between the last two possibilities is provided by the "slot verification" heuristic. It would be better to allow a more flexible scheme for specifying what can fill a slot. The reason this is important is that there are some compounds that can be formed by filling a slot with something that is not expected, but still makes sense. For example, the meaning of the compound "sand cup" can be formed by placing "sand" in the "contains" slot of cup. Normally,

"cup" would expect a liquid, and the scheme presented in the body of the thesis would not find this interpretation. The proposal here allows for a more flexible way to find interpretations.

One way to do this is to allow VC links providing restrictions on what can fill a slot to also provide an indication of the strength of the restriction. For example, a scheme having the following categories could provide a much more flexible method for deciding how well a concept fills a slot:

- An unbreakable VC link, i.e., only a concept that does not conflict with the class defined by the concept at the head of the VC link can be placed in the slot. The single VC link in use now corresponds to this type of VC link.
- A strong VC link, i.e., provides a fairly strong constraint, but can be violated in a few special cases.
- A default VC link that specifies a default only, i.e., the set of concepts that are expected to fill a slot. Other concepts could fill the slot if necessary.

To actually use this knowledge, the system would perform most of the searches it does now with very little change. The major change comes from deciding which VC links are active during a search. Since there are three levels of VC links, it may be necessary to perform three different searches, as follows:

1. All VC links are active. This provides the most constraint, since default values for slots will be used to rule out possible interpretations for a compound, because a concept to fill a slot can not conflict with a default.
2. All VC links except the default ones are active. This provides less constraint, since defaults are no longer respected, and may be overridden.
3. Only unbreakable VC links are active. This provides the least constraint. Anything that does not conflict with the concepts pointed at by unbreakable VC links is allowed to fill a slot.

The system would start out using search method (1) which provides the most constraint. If no possible interpretations are found for a compound using this search method, search method (2) could be used. This provides less constraint, and thus is more likely to find possible interpretations. Finally, if method (2) fails to find any interpretations, method (3) could be used. This still provides some constraint, but reduces it significantly, and is likely to allow more possible interpretations. Some minor details would have to be worked out before this type of search could be integrated into the program, but it should improve the performance of the program in some cases.

8.8.5. Moving the Heuristics Back into the Generator

It is possible to move some of the heuristics back into the part of the program that generates possible interpretations for a compound. Some of the information is already available during this phase (e.g., the information for the "cognate", "embedded cognate", the "instances of an interpretation", and the "embedded instances of an interpretation" heuristics is calculated during the phase used to generate the list of possible interpretations). The results are not used until after all the possible interpretations have been found. Some of the other heuristics could also be calculated during the generation phase. This has the advantage that only the best interpretations would be considered at any point in the computation. It has the disadvantage that the program would need to be able to back up if it follows a path that leads to a dead end. Other heuristics that could be processed during the generation phase include: the heuristics based on context, the "slot verification" heuristic, and the "occurrence" heuristic. The "similar compounds" heuristic probably should not be moved back into the generator, since it is very expensive to process.

Bibliography

- [Bobrow 76] Bobrow, D. G. and Winograd, T.
An Overview of KRL, A Knowledge Representation Language.
Technical Report, Xerox PARC, 1976.
- [Bobrow 78] Bobrow, R. J.
The RUS System.
Technical Report 3878, Bolt Beranek and Newman Inc., 1978.
- [Bobrow 80] Bobrow, R. J. and Webber, B. L.
Knowledge Representation for Syntactic/Semantic Processing.
In *Proceedings, AAAI Conference*. AAAI, 1980.
- [Bruce 75] Bruce, B.
Case Systems for Natural Language.
Artificial Intelligence 6(4):327-360, 1975.
- [Collins 75] Collins, A., Warnock, E. H., Aiello, N., and Miller, M. L.
Reasoning from Incomplete Knowledge.
In Bobrow, D. G. and Collins, A. (editors), *Representation and Understanding: Studies in Cognitive Science*, pages 383-415. Academic Press, New York, 1975.
- [Downing 77] Downing, P.
On the Creation and Use of English Compound Nouns.
Language 53(4):810-842, 1977.
- [Doyle 79] Doyle, J.
A Glimpse of Truth Maintenance.
In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 232-237. Tokyo, Japan, 1979.
- [Fahlman 79] Fahlman, S. E.
NETL: A System for Representing and Using Real-World Knowledge.
Mit Press, Cambridge, 1979.
- [Fahlman 80] Fahlman, S. E.
Design Sketch for a Million-Element NETL Machine.
In *Proceedings of the First AAAI Conference*. Stanford University, 1980.

- [Filmore 68] Filmore, C. J.
The Case for Case.
In Bach, E. and Harms, R. T. (editors), *Universals in Linguistic Theory*, chapter 4.
Holt, Rinehart, and Winston, New York, 1968.
- [Finin 80] Finin, T. W.
The Semantic Interpretation of Nominal Compounds.
PhD thesis, University of Illinois at Urbana-Champaign, 1980.
- [Gershman 79] Gershman, A.
Knowledge-Based Parsing.
PhD thesis, Yale University, 1979.
- [Goldman 74] Goldman, N.
Computer Generation of Natural Language from a Deep Conceptual Base.
PhD thesis, Stanford University, 1974.
- [Grosz 78] Grosz, B. J.
Focusing in Dialog.
In *TINLAP-2: Theoretical Issues in Natural Language Processing*, pages 96-103.
Urbana, Illinois, 1978.
- [Hatcher 65] Hatcher, A. G.
An Introduction to the Analysis of English Noun Compounds.
Word 16(3):357-373, Dec, 1965.
- [Hayes 77] Hayes, P. J.
Some Association-Based Techniques for Lexical Disambiguation by Machine.
Technical Report 75, University of Rochester, June, 1977.
- [Hendrix 75] Hendrix, G. G.
Expanding the Utility of Semantic Networks Through Partitioning.
In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 115-121. Tbilisi, USSR, 1975.
- [Jespersen 46] Jespersen, O.
A Modern English Grammar Part VI.
George Allen and Unwin Ltd., London, 1946.
- [Katz 63] Katz, J. J. and Fodor, J. A.
The Structure of a Semantic Theory.
Language 39(2):170-210, 1963.
- [Leech 60] Leech, R. B.
The Grammar of English Nominalizations.
Indiana University Research Center in Anthropology, Folklore, and Linguistics,
Publication No. 12, 1960.

- [Lees 70] Lees, R. B.
Problems in Grammatical Analysis of English Nominal Compounds.
In Bierwisch, M. and Herdolph, K. (editors), *Progress in Linguistics*, pages 174-186. Mouton, The Hague, 1970.
- [Marchand 69] Marchand, H.
The Categories and Types of Present Day English Word Formation.
C. H. Beck, Munich, 1969.
- [Marcus 80] Marcus, M. P.
A Theory of Syntactic Recognition for Natural Language.
MIT Press, Cambridge, 1980.
- [McDermott 79] McDermott, D and Doyle, J.
An Introduction to Non-Monotonic Logic.
In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 562-567. Tokyo, Japan, 1979.
- [McDonald 78] McDonald, D. B. and Hayes-Roth F.
Inferential Searches of Knowledge Networks as an Approach to Extensible Language-Understanding Systems.
In Waterman, D. and Hayes-Roth, F. (editors), *Pattern Directed Inference Systems*, pages 431-453. Academic Press, New York, 1978.
- [Minsky 75] Minsky, M.
A framework for representing knowledge.
In Winston, P. H. (editor), *The psychology of computer vision*, pages 211-277.
McGraw-Hill, New York, 1975.
- [Newell 73] Newell, A.
Production Systems: models of control structures.
In Chase, W. G. (editor), *Visual Information Processing*, pages 463-526. Academic Press, New York, 1973.
- [Quillian 68] Quillian, M.R.
Semantic Memory.
In Minsky, M. (editor), *Semantic Information Processing*, pages 227-270. MIT Press, Cambridge, 1968.
- [Quillian 69] Quillian, M.R.
The 'Teachable Language Comprehender: a Simulation Program and Theory of Language'.
Communications of the Association for Computing Machinery 12(8):459-476, 1969.
- [Rhync 76] Rhync, J. R.
Lexical Rules and Structures in a Computer Model of Nominal Compounding.
PhD thesis, University of Texas, Austin, 1976.
- [Robinson 80] Robinson, J. J.
DIAGRAM: A Grammar for Dialogues.
Technical Report 205, SRI International, 1980.

- [Russell 75] Russell, S.
Computer Understanding of Conceptually Complex Phrases.
PhD thesis, Stanford University, 1975.
- [Schank 73] Schank, R. C.
Identification of Conceptualizations Underlying Natural Language.
In Schank, R. C. and Colby, K. M. (editors), *Computer Models of Thought and Language*, chapter 5. W. H. Freeman, San Francisco, 1973.
- [Schank 75] Schank, R.C. and Abelson, R. P.
Scripts, plans, and knowledge.
In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 151-157. Tbilisi, U.S.S.R., 1975.
- [Sidner 78] Sidner, C. L.
The Use of Focus as a Tool for Disambiguation of Definite Noun Phrases.
In *TINLAP-2: Theoretical Issues in Natural Language Processing*, pages 86-95.
Urbana, Illinois, 1978.
- [Simmons 72] Simmons, R. F. and Slocum, J.
Generating English Discourse from Semantic Nets.
Communications of the Association for Computing Machinery 15(10):891-905, 1972.
- [Simmons 73] Simmons, R. F.
Semantic Networks: Their Computation and Use for Understanding English Sentences.
In Schank, R. C. and Colby, K. M. (editors), *Computer Models of Thought and Language*, chapter 2. W. H. Freeman, San Francisco, 1973.
- [Su 69] Su, Stanley Y. W.
A Semantic Theory Based upon Interactive Meaning.
Technical Report 68, University of Wisconsin, July, 1969.
- [Winograd 72] Winograd, T.
Understanding Natural Language.
Academic Press, New York, 1972.
- [Woods 70] Woods, W. A.
Transition Network Grammars for Natural Language Analysis.
Communication of the Association for Computing Machinery 13(10):591-606, 1970.
- [Woods 79] Woods, W. A. et al.
Research in Natural Language Understanding Annual Report.
Technical Report 4274, Bolt Beranek and Newman Inc., 1979.
- [Zimmer 73] Zimmer, K.
Some General Observations about Nominal Compounds.
Technical Report 3, Stanford University, Department of Linguistics, 1973.

Appendix A

Example Run of the Program

This appendix contains an example run of the program.

```
@compound
Compound, Created: 2-Sep-81 20:39:52!
(setq AskUser NIL)
NIL
(setq debug 51.)
51
(meaning '(wine glass) nil)
*** It took 353 cycles to find all the possibilities.
There is 1 compound after finding all possibilities.
The possible compounds are:
WINE-GLASS:
  WINE-GLASS is a GLASS [&GLASS-B [&CONTAINER]].
  GLASS-B!CONTAINEE: WINE [&WINE [&BEVERAGE]]
```

Using the only meaning found for the compound.

```
Creating compound: WINE-GLASS
  WINE-GLASS [&WINE-GLASS-A] is a GLASS [&GLASS-B [&CONTAINER]].
  GLASS-B!CONTAINEE: WINE [&WINE [&BEVERAGE]]

*** It took 439 cycles to process the phrase: WINE GLASS.
It took 2.665 seconds of CPU time to process the phrase:
  WINE GLASS.
((NODE &WINE-GLASS-A))
:
:
(meaning '(glass wine glass) nil)
*** It took 853 cycles to find all the possibilities.
There is 1 compound after finding all possibilities.
The possible compounds are:
GLASS-WINE-GLASS:
  WINE-GLASS [&WINE-GLASS-A] is a GLASS [&GLASS-B [&CONTAINER]].
  GLASS-B!CONTAINEE: WINE [&WINE [&BEVERAGE]]
  GLASS-WINE-GLASS is a WINE-GLASS [&WINE-GLASS-A [&GLASS-B]].
  PHYSOB!MADE-OF: GLASS [&GLASS-A [&MATERIAL]]
```

Using the only meaning found for the compound.

Creating compound: GLASS-WINE-GLASS

Using cognate: &WINE-GLASS-A

WINE-GLASS [&WINE-GLASS-A] is a GLASS [&GLASS-B [&CONTAINER]].

GLASS-B!CONTAINEE: WINE [&WINE [&BEVERAGE]]

GLASS-WINE-GLASS [&GLASS-WINE-GLASS-A] is a

WINE-GLASS [&WINE-GLASS-A [&GLASS-B]].

PHYSOB!MADE-OF: GLASS [&GLASS-A [&MATERIAL]]

*** It took 927 cycles to process the phrase: GLASS WINE GLASS.

It took 3.105 seconds of CPU time to process the phrase:

GLASS WINE GLASS.

((NODE &GLASS-WINE-GLASS-A))

;

;

(setq debug 55.)

55

(meaning '(golf club) nil)

*** It took 402 cycles to find all the possibilities.

There are 2 compounds after finding all possibilities.

The possible compounds are:

GOLF-CLUB:

GOLF-CLUB is a CLUB [&CLUB-B [&STICK]].

CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

GOLF-CLUB:

GOLF-CLUB is a CLUB [&CLUB-A [&ORGANIZATION]].

CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 0 cycles to process context.

The following interpretations remain after context:

GOLF-CLUB:

GOLF-CLUB is a CLUB [&CLUB-B [&STICK]].

CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

GOLF-CLUB:

GOLF-CLUB is a CLUB [&CLUB-A [&ORGANIZATION]].

CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 21 cycles to verify the slots of the interpretations.

*** It took 739 cycles to check for similar compounds.

The compounds and their ratings are:

GOLF-CLUB is a CLUB [&CLUB-B [&STICK]].

CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

Final Rating: 8.0

Context: 0.0 Embedded Cognate: 0.0 Slot Verification: 8.0

Instances: 0.0 Embedded Instances: 0.0 Occurrence: 0.0

Similar Compounds: 0.0

GOLF-CLUB is a CLUB [&CLUB-A [&ORGANIZATION]].
 CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]
 Final Rating: 8.0
 Context: 0.0 Embedded Cognate: 0.0 Slot Verification: 8.0
 Instances: 0.0 Embedded Instances: 0.0 Occurrence: 0.0
 Similar Compounds: 0.0

Creating compound: GOLF-CLUB

GOLF-CLUB [&GOLF-CLUB-A] is a CLUB [&CLUB-B [&STICK]].
 CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

Creating compound: GOLF-CLUB

GOLF-CLUB [&GOLF-CLUB-B] is a CLUB [&CLUB-A [&ORGANIZATION]].
 CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 1302 cycles to process the phrase: GOLF CLUB.
 It took 12.605 seconds of CPU time to process the phrase:
 GOLF CLUB.

((NODE &GOLF-CLUB-A) (NODE &GOLF-CLUB-B))

:
 :

(meaning '(golf club)

'((((hit (hit!hitter hit!with) (hit!hitee &ball-1))))))

*** It took 418 cycles to find all the possibilities.
 There are 2 compounds after finding all possibilities.
 The possible compounds are:

GOLF-CLUB:

GOLF-CLUB [&GOLF-CLUB-A] is a CLUB [&CLUB-B [&STICK]].
 CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

GOLF-CLUB:

GOLF-CLUB [&GOLF-CLUB-B] is a CLUB [&CLUB-A [&ORGANIZATION]].
 CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 252 cycles to process context.
 For Context: ((&HIT (HIT!HITTER HIT!WITH) (HIT!HITEE &BALL-1)))
 There are 1 compounds that are supported.
 There are 0 compounds that are ruled out.
 Checking for context produced one meaning.

Creating compound: GOLF-CLUB

Using cognate: &GOLF-CLUB-A

GOLF-CLUB [&GOLF-CLUB-A] is a CLUB [&CLUB-B [&STICK]].
 CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

*** It took 670 cycles to process the phrase: GOLF CLUB.
 It took 4.201 seconds of CPU time to process the phrase:


```

      GOLF CLUB.
((NODE &GOLF-CLUB-A))
;
;
(meaning '(golf club) '(((&fire (fire!firer) (fire!firee &manager-1))))))
*** It took 418 cycles to find all the possibilities.
There are 2 compounds after finding all possibilities.
The possible compounds are:
GOLF-CLUB:
  GOLF-CLUB [&GOLF-CLUB-A] is a CLUB [&CLUB-B [&STICK]].
  CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

GOLF-CLUB:
  GOLF-CLUB [&GOLF-CLUB-B] is a CLUB [&CLUB-A [&ORGANIZATION]].
  CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 142 cycles to process context.
For Context: ((&FIRE (FIRE!FIRER) (FIRE!FIREE &MANAGER-1)))
  There are 1 compounds that are supported.
  There are 0 compounds that are ruled out.
Checking for context produced one meaning.

Creating compound: GOLF-CLUB
Using cognate: &GOLF-CLUB-B
  GOLF-CLUB [&GOLF-CLUB-B] is a CLUB [&CLUB-A [&ORGANIZATION]].
  CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 560 cycles to process the phrase: GOLF CLUB.
It took 3.773 seconds of CPU time to process the phrase:
  GOLF CLUB.
((NODE &GOLF-CLUB-B))
;
;
(meaning '(golf club) '((&organization)))
*** It took 418 cycles to find all the possibilities.
There are 2 compounds after finding all possibilities.
The possible compounds are:
GOLF-CLUB:
  GOLF-CLUB [&GOLF-CLUB-A] is a CLUB [&CLUB-B [&STICK]].
  CLUB-B!USED-IN: GOLF [&GOLF [&GAME]]

GOLF-CLUB:
  GOLF-CLUB [&GOLF-CLUB-B] is a CLUB [&CLUB-A [&ORGANIZATION]].
  CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 146 cycles to process context.
For Context: (&ORGANIZATION)

```

There are 1 compounds that are supported.
 There are 0 compounds that are ruled out.
 Checking for context produced one meaning.

Creating compound: GOLF-CLUB

Using cognate: &GOLF-CLUB-B

GOLF-CLUB [&GOLF-CLUB-B] is a CLUB [&CLUB-A [&ORGANIZATION]].

CLUB-A!PURPOSE: GOLF [&GOLF [&GAME]]

*** It took 564 cycles to process the phrase: GOLF CLUB.

It took 3.776 seconds of CPU time to process the phrase:

GOLF CLUB.

((NODE &GOLF-CLUB-B))

:

:

(meaning '(computer maintenance) nil)

*** It took 266 cycles to find all the possibilities.

There are 2 compounds after finding all possibilities.

The possible compounds are:

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 0 cycles to process context.

The following interpretations remain after context:

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 25 cycles to verify the slots of the interpretations.

*** It took 432 cycles to check for similar compounds.

The compounds and their ratings are:

COMPUTER-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

Final Rating: 8.0

Context: 0.0 Embedded Cognate: 0.0 Slot Verification: 8.0

Instances: 0.0 Embedded Instances: 0.0 Occurrence: 0.0

Similar Compounds: 0.0

COMPUTER-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

Final Rating: 8.0
 Context: 0.0 Embedded Cognate: 0.0 Slot Verification: 8.0
 Instances: 0.0 Embedded Instances: 0.0 Occurrence: 0.0
 Similar Compounds: 0.0

Creating compound: COMPUTER-MAINTENANCE
 COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-A]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

Creating compound: COMPUTER-MAINTENANCE
 COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-B]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 847 cycles to process the phrase: COMPUTER MAINTENANCE.
 It took 9.937 seconds of CPU time to process the phrase:
 COMPUTER MAINTENANCE.

```
(( (NODE &COMPUTER-MAINTENANCE-A). (NODE &COMPUTER-MAINTENANCE-B))
;
;
(meaning '(computer maintenance)
          '((( &maintenance ()
              (maintenance!maintainee. &data/ base-1))))))
```

*** It took 282 cycles to find all the possibilities.
 There are 2 compounds after finding all possibilities.
 The possible compounds are:

COMPUTER-MAINTENANCE:
 COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-A]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

COMPUTER-MAINTENANCE:
 COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-B]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 63 cycles to process context.
 For Context: ((&MAINTENANCE NIL
 (MAINTENANCE!MAINTAINEE &DATA BASE-1)))

There are 1 compounds that are supported.
 There are 1 compounds that are ruled out.
 Checking for context produced one meaning.

Creating compound: COMPUTER-MAINTENANCE
 Using cognate: &COMPUTER-MAINTENANCE-A
 COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-A]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 345 cycles to process the phrase: COMPUTER MAINTENANCE.
It took 2.005 seconds of CPU time to process the phrase:

```
COMPUTER MAINTENANCE.
((NODE &COMPUTER-MAINTENANCE-A))
;
;
(meaning '(car maintenance) nil)
*** It took 241 cycles to find all the possibilities.
There is 1 compound after finding all possibilities.
The possible compounds are:
CAR-MAINTENANCE:
  CAR-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].
  MAINTENANCE!MAINTAINEE: CAR [&CAR [&AUTOMOBILE]]
```

Using the only meaning found for the compound.

```
Creating compound: CAR-MAINTENANCE
CAR-MAINTENANCE [&CAR-MAINTENANCE-A]
  is a MAINTENANCE [&MAINTENANCE [&ACTION]].
MAINTENANCE!MAINTAINEE: CAR [&CAR [&AUTOMOBILE]]
```

*** It took 309 cycles to process the phrase: CAR MAINTENANCE.
It took 1.509 seconds of CPU time to process the phrase:

```
CAR MAINTENANCE.
((NODE &CAR-MAINTENANCE-A))
;
;
(meaning '(truck maintenance) nil)
*** It took 239 cycles to find all the possibilities.
There is 1 compound after finding all possibilities.
The possible compounds are:
TRUCK-MAINTENANCE:
  TRUCK-MAINTENANCE is a MAINTENANCE [&MAINTENANCE [&ACTION]].
  MAINTENANCE!MAINTAINEE: TRUCK [&TRUCK [&VEHICLE]]
```

Using the only meaning found for the compound.

```
Creating compound: TRUCK-MAINTENANCE
TRUCK-MAINTENANCE [&TRUCK-MAINTENANCE-A]
  is a MAINTENANCE [&MAINTENANCE [&ACTION]].
MAINTENANCE!MAINTAINEE: TRUCK [&TRUCK [&VEHICLE]]
```

*** It took 307 cycles to process the phrase: TRUCK MAINTENANCE.
It took 3.008 seconds of CPU time to process the phrase:

```
TRUCK MAINTENANCE.
((NODE &TRUCK-MAINTENANCE-A))
;
;
(meaning '(computer maintenance) nil)
```

*** It took 284 cycles to find all the possibilities..
 There are 2 compounds after finding all possibilities.
 The possible compounds are:

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-A]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-B]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 0 cycles to process context.

The following interpretations remain after context:

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-A]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

COMPUTER-MAINTENANCE:

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-B]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 25 cycles to verify the slots of the interpretations.

*** It took 146 cycles to check for similar compounds.

The compounds and their ratings are:

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-A]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINER: COMPUTER [&COMPUTER [&MACHINE]]

Final Rating: 17.0

Context: 0.0 Embedded Cognate: 8.0 Slot Verification: 8.0

Instances: 0.0 Embedded Instances: 0.0 Occurrence: 1.0

Similar Compounds: 0.0

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-B]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].
 MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

Final Rating: 20.5

Context: 0.0 Embedded Cognate: 8.0 Slot Verification: 8.0

Instances: 0.0 Embedded Instances: 0.0 Occurrence: 0.5

Similar Compounds: 4.0

Creating compound: COMPUTER-MAINTENANCE

Using cognate: &COMPUTER-MAINTENANCE-B

COMPUTER-MAINTENANCE [&COMPUTER-MAINTENANCE-B]
 is a MAINTENANCE [&MAINTENANCE [&ACTION]].

MAINTENANCE!MAINTAINEE: COMPUTER [&COMPUTER [&MACHINE]]

*** It took 455 cycles to process the phrase: COMPUTER MAINTENANCE.
 It took 4.051 seconds of CPU time to process the phrase:
 COMPUTER MAINTENANCE.
 ((NODE &COMPUTER-MAINTENANCE-B))
 ;
 ;
 (setq debug 51.)
 51
 (meaning '(woman coal miner) nil)
 *** It took 700 cycles to find all the possibilities.
 There is 1 compound after finding all possibilities.
 The possible compounds are:
 WOMAN-COAL-MINER:
 COAL-MINER is a MINER [&MINER [&PERSON]].
 MINERIMINES: COAL [&COAL [&MINERAL]]
 WOMAN-COAL-MINER is a COAL-MINER and is a WOMAN [&WOMAN [&PERSON]].

Using the only meaning found for the compound.

Creating compound: WOMAN-COAL-MINER
 COAL-MINER [&COAL-MINER-A] is a MINER [&MINER [&PERSON]].
 MINERIMINES: COAL [&COAL [&MINERAL]]
 WOMAN-COAL-MINER [&WOMAN-COAL-MINER-A]
 is a COAL-MINER [&COAL-MINER-A [&MINER]]
 and is a WOMAN [&WOMAN [&PERSON]].

*** It took 927 cycles to process the phrase: WOMAN COAL MINER.
 It took 2.562 seconds of CPU time to process the phrase:
 WOMAN COAL MINER.

((NODE &WOMAN-COAL-MINER-A))
 ;
 ;
 (meaning '(boy king) nil)
 *** It took 389 cycles to find all the possibilities.
 There is 1 compound after finding all possibilities.
 The possible compounds are:
 BOY-KING:
 BOY-KING is a KING [&KING [&MAN]] and is a BOY [&BOY [&CHILD]].

Using the only meaning found for the compound.

Creating compound: BOY-KING
 BOY-KING [&BOY-KING-A] is a KING [&KING [&MAN]]
 and is a BOY [&BOY [&CHILD]].

*** It took 528 cycles to process the phrase: BOY KING.
 It took 3.218 seconds of CPU time to process the phrase:
 BOY KING.

```

((NODE &BOY-KING-A))
;
;
(meaning '(water meter cover adjustment screw) nil)
*** It took 5369 cycles to find all the possibilities.
There is 1 compound after finding all possibilities.
The possible compounds are:
WATER-METER-COVER-ADJUSTMENT-SCREW:
  WATER-METER is a METER [&METER [&INSTRUMENT]].
  METER!MEASURES: WATER [&WATER [&LIQUID]]
  WATER-METER-COVER is a COVER [&COVER [&NON-LIVING-PHYSOB]].
  COVER!COVERS: WATER-METER
  WATER-METER-COVER-ADJUSTMENT-SCREW
    is a SCREW [&SCREW [&INSTRUMENT]].
&ADJUST-1
  ADJUST!TOOL-OF-&ADJUST-1: SCREW [&SCREW [&INSTRUMENT]]
  ADJUST!ADJUSTEE-OF-&ADJUST-1: WATER-METER-COVER

```

Using the only meaning found for the compound.

```

Creating compound: WATER-METER-COVER-ADJUSTMENT-SCREW
WATER-METER [&WATER-METER-A] is a METER [&METER [&INSTRUMENT]].
  METER!MEASURES: WATER [&WATER [&LIQUID]]
  WATER-METER-COVER [&WATER-METER-COVER-A]
    is a COVER [&COVER [&NON-LIVING-PHYSOB]].
  COVER!COVERS: WATER-METER [&WATER-METER-A [&METER]]
  WATER-METER-COVER-ADJUSTMENT-SCREW
    [&WATER-METER-COVER-ADJUSTMENT-SCREW-A]
      is a SCREW [&SCREW [&INSTRUMENT]].
&ADJUST-1
  ADJUST!TOOL-OF-&ADJUST-1: SCREW [&SCREW [&INSTRUMENT]]
  ADJUST!ADJUSTEE-OF-&ADJUST-1:
    WATER-METER-COVER [&WATER-METER-COVER-A [&COVER]]

```

```

*** It took 17022 cycles to process the phrase:
      WATER METER COVER ADJUSTMENT SCREW.
It took 7.625 seconds of CPU time to process the phrase:
      WATER METER COVER ADJUSTMENT SCREW.
((NODE &WATER-METER-COVER-ADJUSTMENT-SCREW-A))
;
;

```

```

(meaning '(blood donor recruiter) nil)
*** It took 635 cycles to find all the possibilities.
There is 1 compound after finding all possibilities.
The possible compounds are:
BLOOD-DONOR-RECRUITER:
  BLOOD-DONOR is a DONOR [&DONOR [&PERSON]].
  DONOR!DONATES: BLOOD [&BLOOD [&LIQUID]]
  BLOOD-DONOR-RECRUITER is a RECRUITER [&RECRUITER [&PERSON]].
  RECRUITER!RECRUITEE: BLOOD-DONOR

```

Using the only meaning found for the compound.

Creating compound: BLOOD-DONOR-RECRUITER

BLOOD-DONOR [&BLOOD-DONOR-A] is a DONOR [&DONOR [&PERSON]].

DONOR!DONATES: BLOOD [&BLOOD [&LIQUID]]

BLOOD-DONOR-RECRUITER [&BLOOD-DONOR-RECRUITER-A]

is a RECRUITER [&RECRUITER [&PERSON]].

RECRUITER!RECRUTEES: BLOOD-DONOR [&BLOOD-DONOR-A [&DONOR]]

*** It took 791 cycles to process the phrase: BLOOD DONOR RECRUITER.

It took 4.372 seconds of CPU time to process the phrase:

BLOOD DONOR RECRUITER.

((NODE &BLOOD-DONOR-RECRUITER-A))

;

;

(meaning '(coal mine supervisor) nil)

*** It took 641 cycles to find all the possibilities.

There is 1 compound after finding all possibilities.

The possible compounds are:

COAL-MINE-SUPERVISOR:

COAL-MINE is a MINE [&MINE [&BUSINESS]].

MINE!PRODUCES: COAL [&COAL [&MINERAL]]

COAL-MINE-SUPERVISOR is a SUPERVISOR [&SUPERVISOR [&EMPLOYEE]].

EMPLOYEE!EMPLOYER: COAL-MINE

Using the only meaning found for the compound.

Creating compound: COAL-MINE-SUPERVISOR

COAL-MINE [&COAL-MINE-A] is a MINE [&MINE [&BUSINESS]].

MINE!PRODUCES: COAL [&COAL [&MINERAL]]

COAL-MINE-SUPERVISOR [&COAL-MINE-SUPERVISOR-A]

is a SUPERVISOR [&SUPERVISOR [&EMPLOYEE]].

EMPLOYEE!EMPLOYER: COAL-MINE [&COAL-MINE-A [&MINE]]

*** It took 797 cycles to process the phrase: COAL MINE SUPERVISOR.

It took 4.044 seconds of CPU time to process the phrase:

COAL MINE SUPERVISOR.

((NODE &COAL-MINE-SUPERVISOR-A))

;

;

(meaning '(watch repair person) nil)

*** It took 4284 cycles to find all the possibilities.

There is 1 compound after finding all possibilities.

The possible compounds are:

WATCH-REPAIR-PERSON:

WATCH-REPAIR-PERSON is a PERSON [&PERSON [&MAMMAL]].

&REPAIR-A-1

REPAIR-A!REPAIRER-OF-&REPAIR-A-1: PERSON [&PERSON [&MAMMAL]]

REPAIR-A!REPAIREE-OF-&REPAIR-A-1: WATCH [&WATCH [&INSTRUMENT]]

Using the only meaning found for the compound.

Creating compound: WATCH-REPAIR-PERSON

WATCH-REPAIR-PERSON [&WATCH-REPAIR-PERSON-A]
is a PERSON [&PERSON [&MAMMAL]].

&REPAIR-A-1

REPAIR-A!REPAIRER-OF-&REPAIR-A-1: PERSON [&PERSON [&MAMMAL]]

REPAIR-A!REPAIREE-OF-&REPAIR-A-1: WATCH [&WATCH [&INSTRUMENT]]

*** It took 15961 cycles to process the phrase: WATCH REPAIR PERSON.

It took 4.809 seconds of CPU time to process the phrase:

WATCH REPAIR PERSON.

((NODE &WATCH-REPAIR-PERSON-A))

;

;

(setq debug 0.)

0

(meaning '(small wine glass) nil)

There is 1 compound after finding all possibilities.

Creating compound: SMALL-WINE-GLASS

Using cognate: &WINE-GLASS-A

WINE-GLASS [&WINE-GLASS-A] is a GLASS [&GLASS-B [&CONTAINER]].

GLASS-B!CONTAINEE: WINE [&WINE [&BEVERAGE]]

SMALL-WINE-GLASS [&SMALL-WINE-GLASS-A]

is a WINE-GLASS [&WINE-GLASS-A [&GLASS-B]]

and is SMALL [&SMALL].

((NODE &SMALL-WINE-GLASS-A))

;

;

(meaning '(red car) nil)

There is 1 compound after finding all possibilities.

Creating compound: RED-CAR

RED-CAR [&RED-CAR-A] is a CAR [&CAR [&AUTOMOBILE]]

and is RED [&RED].

((NODE &RED-CAR-A))

(quit)

Appendix B

Examples of Noun Compounds

This appendix presents the list of noun compounds used in chapter 7 to evaluate the program to understand noun compounds. The description of each compound consists of several lines. The first line contains the noun compound, the parse that I believe is meant by the author, a code giving some information about how the program described in this thesis would process the compound, and finally a description of what I believe the author was trying to say when he used the compound. The parse of the compound shows how the words in the compound are to be grouped. Any words in curly braces are not nouns, and would not be processed by the system described in the main body of the thesis. They are there to provide context or when they are essential part of the phrase. The code given in the first line is a short phrase enclosed in square brackets. These short phrases have the following meanings:

- **Correct left to right parse:** The compound should be processed correctly by the program using a left to right parse. This means selecting the correct slot from possibly many as the basis of the relationship between the nouns. Also, in a few cases multiple meanings of one or more nouns should be resolved successfully with just the context provided by the compound itself.
- **Correct middle ternary parse:** The program finds the correct interpretation for the compound by forming a compound using the middle ternary pattern.
- **Correct end ternary parse:** The program finds the correct interpretation for the compound by forming a compound using the end ternary pattern.
- **Multiple meanings due to word ambiguity:** The program would find multiple meanings for the compound because at least one of the words in the compound has more than one meaning. With the appropriate context, these compounds should be correctly processed by the program.
- **Multiple meanings due to slot ambiguity:** The program would find multiple meanings for the compound because one of the words provides two or more slots that can be used to form an interpretation for the compound. With the appropriate context, these compounds should be correctly processed.

- **No meaning found:** The program would not find any slots to use as a basis to form the meaning for the compound.
- **Wrong meaning found:** The program would probably find the wrong meaning for the compound.
- **Questionable slot used:** The program would find the correct meaning for the compound if the slot used to form the meaning was available. This code shows that a slot may be of a questionable nature.
- **Questionable slot assignment:** The program would find the correct meaning for the compound if the slot used to form the meaning could accept the value placed into it. This code shows that the slot assignment may violate restrictions place on concepts that can fill the slot.
- **Correct as far as it goes:** The program would find the indicated meaning. This code shows that the meaning does not cover the full meaning of the compound, but only part of the meaning.
- **First noun provides a slot for the second:** The program would find no slots to use as the basis for forming a meaning of the compound. However, if the second noun is placed into a slot of the first one, the correct meaning for the compound would be formed.
- **Compound defined in the text:** The compound is defined in the text to have a specific meaning. The program can not currently input such a definition other than by hand.
- **Union of concepts forms meaning:** The compound seems to be formed from a union of concepts all filling the same slot of the head noun.

Finally, the last few lines give the interpretation for the compound that I believe the program would generate. Note that the program has not been run on these examples.

B.1. Examples from an article in Scientific American

The following noun compounds are from *The Fuel Economy of Light Vehicles* in Scientific American, Volume 244, Number 5, May 1981.

1. [Fuel Economy] [Multiple meanings due to word ambiguity]: Economy as a saving of fuel.
 Economy: Resource Managed: Fuel
 Economy: Based on: Fuel
2. [Oil Production] [Correct left to right parse]: Production where thing produced is oil.
 Production: Product: Oil

3. [Underground Oil] [Correct left to right parse]: Location of oil is underground.
Oil:
Location: Underground
4. [Post-Petroleum Era] [Questionable slot used]: Era after all the petroleum has been used up.
Era:
Time-Span: Post-Petroleum
5. [Passenger Car] [Correct left to right parse]: Car that is for transporting passengers.
Car:
Transports: Passenger
6. [Pickup Truck] [Correct as far as it goes]: A truck of a particular style used to transport pickups.
Truck:
Transports: Pickup
7. [Petroleum Products] [Correct left to right parse]: Product made from petroleum.
Product:
Source: Petroleum
8. [Oil Embargo] [Correct left to right parse]: Embargo where the thing embargoed is oil.
Embargo:
Object Embargoed: Oil
9. [Gasoline Shortage] [Correct left to right parse]: Shortage where gasoline is in short supply.
Shortage:
Thing Short of: Gasoline
10. [Oil Imports] [Correct left to right parse]: Import where the thing imported is oil.
Import:
Thing Imported: Oil
11. [Automobile Industry] [Correct left to right parse]: Industry that produces automobiles.
Industry:
Product: Automobile
12. {Personal}-{Consumption Expenditures} [Questionable slot used]: Expenditures for personal-consumption items.
Expenditure:
Purpose: Personal-Consumption
13. [Consumer Preference] [Multiple meanings due to slot ambiguity]: Preference where consumer prefers efficient vehicles.
Preference:
Thing Preferred: Consumer
Who Prefers: Consumer
14. [Automobile Worker] [Correct left to right parse]: Worker who works producing automobiles.
Worker:

Produces: Automobile

15. [Car Maker] [Correct left to right parse]: Maker who makes a car.
 Maker:
 Makes: Car
16. [Capital Expenditure] [Correct left to right parse]: Expenditure of capital for a particular purpose.
 Expenditure:
 Thing Expended: Capital
17. [Production Facility] [Correct left to right parse]: Facility where production takes place.
 Facility:
 Thing Facilitated: Production
18. [Automobile Fleet] [Correct left to right parse]: Fleet made up of automobiles.
 Fleet:
 Fleet-Member: Automobile
19. [[Fuel Economy] Standard] [Correct left to right parse]: Standard for saving of fuel.
 Standard:
 Standard-For: Fuel Economy
20. [Consumer Demand] [Multiple meanings due to slot ambiguity]: Consumer demands light vehicles.
 Demand:
 Thing Demanded: Consumer
 Demander: Consumer
21. [Investment Program] [Multiple meanings due to word ambiguity]: Program for investing.
 Program: as in schedule.
 Program for: Investment.
 Program: as in computer program.
 Program for: Investment.
22. [Dynamometer Test] [Correct left to right parse]: Test using a dynamometer.
 Test:
 Tool used in test: Dynamometer.
23. {Light} [Vehicle Miles] [No meaning found]: Miles travelled by light vehicles.
24. [Oil Consumption] [Correct left to right parse]: Consumption where oil is consumed.
 Consumption:
 Thing Consumed: Oil
25. [Drive Lines] [No meaning found]: Method of getting power to the wheels of a car.
26. [Power Plant] [Wrong meaning found]: A car engine.
 Plant: as in a factory.

Product: Power

27. [Passenger Vehicle] [Correct left to right parse]: Vehicle used to transport passengers.
Vehicle:
 Transports: Passenger
28. [[{Two}-Passenger] Car] [Wrong meaning found]: Car with a capacity for two passengers.
Car:
 Transports: Two-Passenger
29. {Internal}-{Combustion Engine} [Questionable slot used]: Engine where fuel is burned within the engine.
Engine:
 Method of using fuel: Internal-Combustion
30. [Production Vehicles] [First noun provides a slot for the second]: Vehicle produced in quantity.
31. [Fuel Efficiency] [Correct left to right parse]: Efficient use of fuel.
Efficiency:
 Efficient in use of: Fuel
32. [Emission Standard] [Correct left to right parse]: Standard for emission of car exhaust.
Standard:
 Standard for: Emission
33. [Gasoline Price] [Correct left to right parse]: Price of gasoline.
Price:
 Price-of: Gasoline
34. [Vehicle Capacity] [Correct left to right parse]: Capacity of a vehicle.
Capacity:
 Capacity-of: Vehicle
35. [Transportation Requirements] [Correct left to right parse]: Requirements for transportation of people.
Requirement:
 Required-for: Transportation
36. [Market Behaviour] [Correct left to right parse]: Behaviour of the market for a car.
Behaviour:
 Behaviour-of: Market
37. [Price Increase] [Correct left to right parse]: Increase where the price of something is increased.
Increase:
 Thing increased: Price
38. [[{Passenger Car} Sales] Mix] [Correct left to right parse]: Mix of sales of passenger cars.
Mix:

- Mix-of: Passenger Car Sales
 Sales:
 Item Sold: Passenger Car
 Car:
 Transports: Passenger
39. [Weight Reduction] [Correct left to right parse]: Reduction where weight is reduced.
 Reduction:
 Thing Reduced: Weight
40. [Interior Space] [Questionable slot used]: The space inside a car.
 Space:
 Space taken up by: Interior
41. [Passenger Compartment] [Correct left to right parse]: Compartment of a car for holding passengers.
 Compartment:
 Contains: Passenger
42. [Trunk Space] [Multiple meanings due to word ambiguity]: Space taken up by the trunk of a car.
 Space:
 Space taken up by: Trunk [car or luggage]
43. [Engine Horsepower] [Questionable slot used]: The power rating of an engine.
 Horsepower:
 Horsepower-of: Engine
44. [[Weight Reduction] Program] [Correct left to right parse]: Program to reduce the weight of a car.
 Program:
 Schedule-for: Weight Reduction
45. [{Front}-Wheel] Drive] [Questionable slot used]: Power applied by the front wheels of a car.
 Drive:
 Applied-by: Front-Wheel
46. [Engine Compartment] [Correct left to right parse]: Compartment of a car that contains the engine.
 Compartment:
 Contains: Engine
47. [Luggage Compartment] [Correct left to right parse]: Compartment of a car that contains luggage.
 Compartment:
 Contains: Luggage
48. [Energy Consumption] [Correct left to right parse]: Energy is the thing consumed.
 Consumption:

Thing Consumed: Energy

49. [Automobile Engine] [Correct left to right parse]: Engine that is part of an automobile.
Engine:
Part-of: Automobile
50. [Performance Requirement] [Correct left to right parse]: Requirement for performance of a car.
Requirement:
Requirement-for: Performance
51. [Acceleration Capability] [Correct left to right parse]: The capability of a car to accelerate.
Capability:
Thing capable of: Acceleration
52. [Air Conditioning] [No meaning found]: Thing conditioned is air.
Conditioning:
Thing Conditioned: Air
53. [Test Weight] [Compound defined in the text]: Weight of a car plus 300 pounds for load.
54. [Air Resistance] [Multiple meanings due to slot ambiguity]: Resistance of air pressing against a car.
Resistance:
Thing Doing Resisting: Air
Thing being resisted: Air
55. [Prototype Car] [First noun provides a slot for the second]: A new car to be tested before production.
56. [{High}-Pressure] Tire [No meaning found]: Tire that contains air under high pressure.
57. [Engine Power Output] [Correct end ternary parse]: Output of power produced by an engine.
Output:
Thing Output: Power
Thing Doing the Output: Engine
58. [Engine Speed] [Correct left to right parse]: Speed of an engine.
Speed:
Speed-of: Engine
59. [Road Speed] [Questionable slot used]: Speed of a car on a road.
Speed:
Location: Road
60. [Power Output] [Correct left to right parse]: Output where power is the thing output.
Output:
Thing Output: Power
61. [Power Loss] [Correct left to right parse]: Loss where power is the thing lost.

Loss:

Loss-of: Power

62. [Engine Power] [Correct left to right parse]: Power that is produced by an engine.

Power:

Power Source: Engine

63. [Peak Power] [No meaning found]: Amount of power produced when an engine is at full power.

64. [Diesel Engine] [Correct as far as it goes]: Engine that runs on diesel fuel. Miss invented by Diesel.

Engine:

Fuel: Diesel

65. [{36} Horsepower] Engine [Questionable slot used]: Engine with a power rating of 36 horsepower.

Engine:

Power Rating: 36 Horsepower

66. [{5- to 15}-horsepower] range [No meaning found]: Range in horsepower from 5 to 15.

67. [Fuel Energy] [Questionable slot used]: Energy contained in a particular fuel.

Energy:

Energy derived from: Fuel

68. [Emission Standard] [Correct left to right parse]: Standard for emission of exhaust gases.

Standard:

Standard for: Emission

69. [Engine Designer] [Correct left to right parse]: A person who designs engines.

Designer:

Thing designed: Engine

70. [Truck Diesel] [Correct left to right parse]: A truck powered by a Diesel engine.

Diesel:

Part-of: Truck

71. [Exhaust Gases] [Questionable slot used]: Gases exhausted from a combustion engine.

Gas:

Source: Exhaust

72. [Power Stroke] [No meaning found]: Stroke of an engine cycle in which power is delivered.

73. [Engine Size] [Wrong meaning found]: Size of an engine in terms of power rating.

Size:

Size-of: Engine

74. [Fuel Economy] Advance [Correct left to right parse]: Advances in saving fuel in cars.

Advance:

Thing advanced: fuel economy

75. [Engine Friction] [Correct left to right parse]: Friction in an engine.

Friction:

Friction in: Engine

76. [Efficiency Map] [No meaning found]: Map of how efficient an engine is.

77. [Accessory Power Requirements] [Correct end ternary parse]: Requirements for power to run accessories.

Requirements:

Requires: Power

Required-for: Accessory

78. [Driving Cycle] [Wrong meaning found]: Lifetime of a vehicle.

Cycle:

Cycle Repeated: Driving

79. [City Driving] [Correct left to right parse]: Driving that takes place in a city.

Driving:

Location: City

80. [{1980} model] year] [No meaning found]: The year for a particular style of car.

81. [Vehicle Weight] [Correct left to right parse]: Weight of a vehicle.

Weight:

Weight-of: Vehicle

82. [Weight Range] [Correct left to right parse]: Range of possible weights.

Range:

Range-of: Weight

83. [Performance Specifications] [Correct left to right parse]: Specifications for the performance of a car.

Specifications:

Thing specified: Performance

84. [Vehicle Fleet] [Correct left to right parse]: Fleet made up of vehicles.

Fleet:

Fleet member: Vehicle

85. [Propulsion Energy] [Correct left to right parse]: Energy used for propulsion of a car.

Energy:

Used-for: Propulsion

86. [Drive-Train] [Efficiency] [No meaning found]: Efficiency of delivering power to the wheels of a car.

Efficiency:

Efficiency-of: Drive-Train

87. [Prototype Vehicle] [First noun provides a slot for the second]: A new vehicle that is tested before being produced.
88. {Direct}-[Injector Diesel] [No meaning found]: Diesel engine where fuel is directly injected into the cylinder.
89. [Gasoline-{Equivalent}] [[[Composite City-Highway] Fuel] Efficiency]] [No meaning found]: A measurement of the fuel efficiency of a car.
90. [[[Recent] Model] Car] [Wrong meaning found]: A car made recently. Probably taken as a toy car.
91. [[Fuel Economy] Goal] [No meaning found]: A particular goal for fuel saving.
Goal:
Objective: Fuel Economy
92. [Seat Belts] [No meaning found]: Belts attached to seats of cars for safety.
93. [Injury Risk] [Correct left to right parse]: Risk of being injured.
Risk:
Thing Risked: Injury
94. [Automobile Makers] [Correct left to right parse]: Makers that make automobiles.
Maker:
Thing Made: Automobile
95. [[Crash-Test] Standard] [Correct left to right parse]: Standard for testing crash worthiness of cars.
Standard:
Standard-for: Crash Test
Test:
Test-of: Crash
96. [Air Pollution] [Correct left to right parse]: Pollution of the air.
Pollution:
Thing Polluted: Air
97. [Diesel Engine] [Multiple meanings due to word ambiguity]: Engine powered by diesel fuel. Also developed by Diesel.
Engine:
Powered by: Diesel
Developed by: Diesel
98. [Diesel Emissions] [Multiple meanings due to word ambiguity]: Emissions caused by burning diesel fuel in a diesel engine.
Emissions:
Caused-by: Diesel [Engine]

Source: Diesel [Oil]

99. [Nitrogen Oxide] [Correct left to right parse]: Oxide of Nitrogen produced in combustion process.

Oxide:

Oxide-of: Nitrogen

100. [Acid Rain] [No meaning found]: Rain that is more acidic than normal.

101. [[[Pre-Chamber] Diesel] Engine] [No meaning found]: A type of diesel engine.

102. [[Nitrogen Oxide] Emissions] [Correct left to right parse]: Emissions of nitrogen oxide.

Emission:

Thing Emitted: Nitrogen Oxide

103. [Particulate Emissions] [Correct left to right parse]: Emissions of particulate matter.

Emission:

Thing Emitted: Particulate

104. [Diesel Particulates] [Correct as far as it goes]: Particulates emitted after combustion of diesel fuel in a diesel engine.

Particulate:

Source: Diesel [Engine or Fuel]

105. [Diesel Fumes] [Wrong meaning found]: Fumes produced by burning diesel fuel.

Fumes:

Source: Diesel [Fuel or Engine]

106. [Pollution Problem] [Questionable slot used]: Problem of pollution.

Problem:

Problem-type: Pollution

107. [[[Direct] Injector] [{Stratified} Charge] Engine] [No meaning found]: A type of diesel engine.

108. [[Fuel-Air] Mixture] [Union of concepts forms meaning]: A mixture containing fuel and air.

109. [Combustion Chamber] [Questionable slot used]: Chamber where combustion takes place.

Chamber:

Contains: Combustion

Purpose: Combustion

110. [Plant Material] [Multiple meanings due to word ambiguity]: Material derived from green plants.

Material:

Source: Plant [at least two different meanings]

111. [Storage Battery] [No meaning found]: A particular type of battery.

112. [Methanol Route] [No meaning found]: Possibility of using methanol as a fuel for cars.

113. [[Fuel Economy] Improvements] [Correct left to right parse]: Improvement in fuel economy of cars.

Improvement:

Thing Improved: Fuel Economy

114. [[Cost-Benefit] Analysis] [Union of concepts forms meaning]: Analysis of costs and benefits of something.

115. [Fuel Savings] [Correct left to right parse]: Savings where thing saved is fuel.

Savings:

Thing Saved: Fuel

116. [Fuel Cost] [Correct left to right parse]: Cost of fuel.

Cost:

Cost-of: Fuel

117. [Fuel Waster] [Correct left to right parse]: A car that wastes fuel.

Waster:

Thing wasted: Fuel

118. [Overseas Production] [Questionable slot used]: Production of cars overseas.

Production:

Location: Overseas

119. [Automobile [Fuel Economy]] [Wrong meaning found]: Fuel use of an automobile.

Economy:

Resource Managed: Automobile Fuel

Fuel:

Fuel-for: Automobile

120. [[Motor Vehicle] Fuel] [Correct left to right parse]: Fuel used to power a vehicle powered by a motor.

Fuel:

Fuel-for: Motor Vehicle

Vehicle:

Propelled-by: Motor

121. [[Fuel Economy] Test] [Correct left to right parse]: Test of the fuel economy of a car.

Test:

Test-of: Fuel Economy

122. [[Fuel Economy] Race] [Questionable slot assignment]: Race to improve the fuel efficiency of cars.

Race:

Race-for: Fuel Economy

B.2. Examples from the Communications of the Association for Computing Machinery

This first set of compounds comes from the article *Computer Programs for Detecting and Correcting Spelling Errors* by Peterson, J. L. in *Communications of the Association for Computing Machinery*, Volume 23, Number 12, December 1980.

123. [Document Preparation] [Correct left to right parse]: Preparation of a document.

Preparation:

Thing Prepared: Document

124. [File System] [Questionable slot used]: System that contains computer files.

System:

Element-of-System: File

125. [Text Editor] [Multiple meanings due to word ambiguity]: Editor on a computer that edits text.

Editor: [Person or computer program]

Thing edited: Text

126. [Text Formatting Program] [Correct middle ternary parse]: Program that formats text on a computer.

Formatting:

Formatter: Program

Thing formatted: Text

127. [Text Formatter] [Correct left to right parse]: Formatter that runs on a computer that formats text.

Formatter:

Thing formatted: Text

128. [Text File] [Correct left to right parse]: A computer file that contains text.

File:

Contains: Text

129. [Batch Program] [Questionable slot used]: A program that is run in batch mode.

Program:

How executed: Batch

130. [Spelling Error] [Correct left to right parse]: Error in the spelling of a word.

Error:

Type of error: Spelling

131. [Text Processing System] [Correct middle ternary parse]: System used to process text on a computer.

Process:

Thing processed: Text

Processor: System

132. [Spelling Checker] [Correct left to right parse]: A program that checks the spelling of words.
 Checker:
 Checked for: Spelling
133. [Spelling Corrector] [Correct left to right parse]: A program that corrects the spelling of words.
 Corrector:
 Thing corrected: Spelling
134. [Input File] [First noun provides a slot for the second]: A file that is input to a program.
135. [Pattern Recognition] [Correct left to right parse]: Recognition of a pattern.
 Recognition:
 What is recognized: Pattern
136. [Letter Sequence] [Correct left to right parse]: Sequence whose elements are letters.
 Sequence:
 Element of sequence: Letter
137. [Candidate Word] [First noun provides a slot for the second]: Word that is a candidate for the correct spelling of an incorrect word.
138. [Word Space] [Correct as far as it goes]: The space of all words that can occur.
 Space:
 Contains: Word
139. [Author Ignorance] [Correct left to right parse]: Ignorance of an author of the correct spelling.
 Ignorance:
 Who is ignorant: Author
140. [Finger Movement] [Correct left to right parse]: Movement of the finger that cause particular types of spelling errors.
 Movement:
 Thing Moved: Finger
141. [Storage Error] [Questionable slot used]: Errors caused in the storage of information.
 Error:
 Cause: Storage
142. [Transmission Error] [Questionable slot used]: Errors caused by transmitting information.
 Error:
 Cause: Transmission
143. [Transmission Mechanism] [Correct left to right parse]: Mechanism for transmission of information.
 Mechanism:
 Used-for: Transmission
144. [Spelling Correction] [Correct left to right parse]: Correction of spelling errors.
 Correction:

- Thing Corrected: Spelling
145. [{Optical} Character Recognition Input] [Wrong meaning found]: Input of characters through optical recognition.
 Input: Thing Input: Character Recognition
 Recognition: Thing recognized: Character
146. [Data Entry] [Correct left to right parse]: Entry of data into a computer system.
 Entry: Entry-of: Data
147. [Input Device] [Correct left to right parse]: Device used to input information to a computer system.
 Device: Purpose: Input
148. [Airline Flight] [Correct left to right parse]: Flight scheduled by an airline.
 Flight: Flight by: Airline
149. [Inquiry Name] [First noun provides a slot for the second]: Name entered as an inquiry to a airline flight reservation system.
150. [Key Word] [No meaning found]: Word that specifies a function in a programming language.
151. [Spelling Problem] [Questionable slot used]: Problem of correcting the spelling of words.
 Problem: Problem type: Spelling
152. [Software Tool] [Questionable slot used]: Tool used to build software.
 Tool: Used while working with: Software
153. [String Matching] [Correct left to right parse]: Matching where the things matched are strings.
 Matching: Thing matched: String
154. [Corrector Algorithm] [Questionable slot used]: Algorithm for a program that correct spelling errors.
 Algorithm: Algorithm for: Corrector
155. [Application Program] [Questionable slot used]: A program that is used for a particular application.
 Program: Type: Application
156. [Software Technology] [Correct left to right parse]: Technology used in designing and building software.

Technology:

Technology-of: Software

157. [Compound Words] [First noun provides a slot for the second]: A word that is compounded from two or more other words.

158. [Case Problems] [Multiple meanings due to word ambiguity]: Problems with the case of a letter.
Problem:

Type: Case

159. [Computer Literature] [Questionable slot used]: Literature containing information about computers.

Literature:

Literature area: Computer

160. [Candidate List] [Correct left to right parse]: List whose elements are candidates for the correct spelling of a word.

List:

List element: Candidate

161. [Comparison Operator] [Questionable slot used]: Operator that performs a comparison.

Operator:

Operation performed: Comparison

162. [Letter Error] [Questionable slot used]: Error caused by a single mistyped letter.

Error:

Type: Letter

163. [Character Position] [Correct left to right parse]: Position of a character in a word.

Position:

Position of: Character

164. [Dictionary Structure] [Correct left to right parse]: Structure of a dictionary of words.

Structure:

Structure of: Dictionary

165. [Transposition Error] [Questionable slot used]: Error caused by transposition of two letters in a word.

Error:

Cause: Transposition

166. [Chain Search] [Wrong meaning found]: Search of the entry of a hash table that uses chaining to resolve conflicts.

Search:

Search-for: Chain

167. [Dictionary Word] [Correct left to right parse]: Word that is in a dictionary.

Word:

Location: Dictionary

168. [Correction Accuracy] [Correct left to right parse]: Accuracy of a program for spelling correction.

Accuracy:

Accuracy of: Correction

169. [Correction Speed] [Questionable slot used]: Speed of a program that does spelling correction.

Speed:

Speed-of: Correction

170. [Confusion Matrix] [Compound defined in the text]: Matrix giving the chance of reading the wrong character.

171. [Digram Frequency] [Correct left to right parse]: The frequency of digrams (two letter combinations).

Frequency:

Frequency of: Diagram

172. [Trigram Frequency] [Correct left to right parse]: The frequency of trigrams (three letter combinations).

Frequency:

Frequency of: Trigram

173. [Typewriter Keyboard] [Correct left to right parse]: Keyboard that is part of a typewriter.

Keyboard:

Part-of: Typewriter

174. [Word Sound] [Correct left to right parse]: Sound made when speaking a word.

Sound:

Sound of: Word

175. [Performance Reason] [Questionable slot used]: Reason for doing something based on the performance of a program.

Reason:

Type: Performance

176. [Affix Removal] [Correct left to right parse]: Removal of the affix of a word.

Removal:

Thing Removed: Affix

177. [Root Word] [First noun provides a slot for the second]: Word that is the root of several words.

178. [Suffix Removal] [Correct left to right parse]: Removal of the suffix of a word.

Removal:

Thing removed: Suffix

179. [Affix Analysis] [Correct left to right parse]: Analysis of the affix of a word.

Analysis:

Thing Analyzed: Affix

180. [Programming Principle] [Questionable slot used]: Principle used in programming computers.
Principle:
Domain of principle: Programming
181. [Spelling Program] [Wrong meaning found]: Program that performs spelling correction.
Program:
Type: Spelling
182. [Programming Project] [Questionable slot used]: Project that is to program a computer.
Project:
Project domain: Programming
183. [Document Word] [Correct left to right parse]: Word that occurs in a document.
Word:
Location: Document
184. [Text Portion] [Compound defined in the text]: Comments in a program describing the algorithm.
Portion:
Portion of: Text
185. [Code Portion] [Compound defined in the text]: The code to implement the algorithm above.
Portion:
Portion of: Code
186. [Code Representation] [Correct left to right parse]: The coded representation of an algorithm.
Representation:
Represented-by: Code
187. [Design Choice] [Questionable slot used]: Choice made in designing an algorithm.
Choice:
Thing Chosen: Design
188. [Design Document] [Correct left to right parse]: Document that describes the design of a program.
Document:
About: Design
189. [Data Type] [Correct left to right parse]: Type of data used in a computer program.
Type:
Type-of: Data
190. [Variable Declaration] [Correct left to right parse]: Declaration of a variable in a computer program.
Declaration:
Thing declared: Variable
191. [[Index Variable] Declaration] [Questionable slot used]: Declaration of a variable used for indexing.

Declaration:

Thing declared: Index Variable

Variable:

Purpose: Index

192. [[Procedure Call] Overhead] [Questionable slot used]: Overhead needed to invoke a procedure.

Overhead:

Cost-of: Procedure Call

Call:

Thing called: Procedure

193. [Execution Time] [Multiple meanings due to slot ambiguity]: The time taken to execute a program.

Time:

Time-of: Execution

Time taken for: Execution

194. [Program Readability] [Correct left to right parse]: Readability of a program.

Readability:

Readability-of: Program

195. [Memory Storage Requirements] [Correct end ternary parse]: Requirements of memory for storage of words.

Requirements:

Thing Required: Memory

Required-for: Storage

196. [Input Token] [First noun provides a slot for the second]: Token that is input to a computer program.

197. [Variable Name] [Correct left to right parse]: Name of a variable.

Name:

Name-of: Variable

198. [Output File] [First noun provides a slot for the second]: File that is output by a computer program.

199. [Search Time] [Multiple meanings due to slot ambiguity]: The time taken to perform a search.

Time:

Time-of: Search

Time-to: Search

200. [Hash Table] [No meaning found]: A method of implementing a symbol table in a computer.

201. [Frequency Count] [Correct left to right parse]: Count of the frequency of something.

Count:

Thing Counted: Frequency

202. [Table Structure] [Multiple meanings due to word ambiguity]: Structure of a table in computer memory.

Structure:

Structure-of: Table

203. [[Token List] Concept] [Correct left to right parse]: Concept of using a list of tokens.

Concept:

Concept-of: Token List

List:

List element: Token

204. [[System Dictionary] Administrator] [Correct left to right parse]: Administrator of a dictionary stored in a computer.

Administrator:

Thing administered: System Dictionary

205. [System Dictionary] [Correct left to right parse]: Dictionary stored in a computer system.

Dictionary:

Part of: System

206. [Topic Area] [Correct left to right parse]: Area of a particular topic.

Area:

Area-of: Topic

207. [Individual User] [No meaning found]: Standard user of a computer system.

208. [Batch Checker] [No meaning found]: A program that checks spelling that runs in batch mode.

209. [Response Time] [Multiple meanings due to slot ambiguity]: The time take for a program to respond.

Time:

Time-of: Response

Time taken for: Response

210. [Computer System] [Questionable slot used]: A system that allows people to compute things.

System:

System type: Computer

211. [Memory Size] [Multiple meanings due to word ambiguity]: Size of the memory of a computer system.

Size:

Size-of: Memory [Human or Computer]

212. [[File Access] Method] [Correct left to right parse]: Method used to access a file in a computer system.

Method:

Method for: File Access

Access:

Thing accessed: File

213. [Data Structures] [Correct left to right parse]: Structure of data in a computer.

Structure:

Structure of: Data

214. [Computer File] [Questionable slot used]: File that is part of a computer system.

File:

Part-of: Computer

215. [Token Length] [Correct left to right parse]: Length of a token.

Length:

Length of: Token

216. [Search Algorithm] [Questionable slot used]: Algorithm used to search a dictionary.

Algorithm:

Algorithm for: Search

217. [[Hash Chain] Table] [No meaning found]: Table whose entries are hash chains.

218. [Hash Chain] [No meaning found]: A chain of words that has to the same entry of a hash table.

219. [Hash Function] [No meaning found]: Function used to find an entry in a hash table.

220. [[Hash Table] Entry] [Multiple meanings due to slot ambiguity]: Entry in a hash table.

Entry:

Entry in: Hash Table

221. [Tree Structure] [Multiple meanings due to word ambiguity]: Structure of a tree in computer memory.

Structure:

Structure of: Tree [Plant or Computer]

222. [Drum Storage] [Questionable slot used]: Storage of information on a drum.

Storage:

Location: Drum

223. [Disk Storage] [Questionable slot used]: Storage of information on a disk.

Storage:

Location: Disk

224. [In-Core Index] [Correct left to right parse]: Index located in the memory of a computer.

Index:

Location: In-Core

225. [Table Size] [Multiple meanings due to word ambiguity]: Size of a table in a computer.

Size:

Size-of: Table

226. [Search Strategy] [Correct left to right parse]: Strategy used to search a table in computer memory.

Strategy:

Strategy for: Search

227. [Search Time] [Questionable slot used]: Time taken to perform a search.

Time:

Time of: Search

Time taken to: Search

228. [Subject Area] [Questionable slot used]: The area in which a subject occurs.

Area:

Type: Subject

229. [Segmentation Scheme] [Correct left to right parse]: Scheme for segmentation of memory.

Scheme:

Scheme for: Segmentation

230. [[Memory Management] Scheme] [Correct left to right parse]: Scheme for the management of memory in a computer system.

Scheme:

Scheme for: Memory Management

Management:

Thing Managed: Memory

231. [Suffix Analysis] [Correct left to right parse]: Analysis of the suffix of a word.

Analysis:

Thing Analyzed: Suffix

232. [Prefix Analysis] [Correct left to right parse]: Analysis of the prefix of a word.

Analysis:

Thing Analyzed: Prefix

233. [Syntax Checker] [Correct left to right parse]: A program that checks the syntax of a document.

Checker:

Thing checked: Syntax

The following compounds come from *Learning and Reasoning by Analogy* by Winston, P. H. in Communications of the Association for Computing Machinery, Volume 23, Number 12, December 1980.

234. [Water Pipe] [Correct left to right parse]: A pipe that contains water.

Pipe:

Contains: Water

235. [Constraint Description] [Correct left to right parse]: Description of a constraint on a situation.

Description:

Thing Described: Constraint

236. [Case Study] [Multiple meanings due to word ambiguity]: A study of all the possible cases of a problem.
Study:
 Thing Studied: Case
237. [Constraint Relation] [Questionable slot used]: A relation provided by a constraint.
Relation:
 Relation type: Constraint
238. [Voltage-Resistance-Current Situation] [Union of concepts forms meaning]: Situation involving voltage, resistance, and current.
Situation:
 Situation type: Voltage-Resistance-Current
239. [[Water Pipe] Law] [Multiple meanings due to word ambiguity]: Physical law about the way that water in pipes behaves.
Law: [Physical or Legal]
 Law about: Water Pipe
240. [Plot Outline] [Multiple meanings due to word ambiguity]: Outline of the plot of a story.
Outline:
 Outline of: Plot [Story or Land]
241. [Word Group] [Correct left to right parse]: A group whose elements consist of words.
Group:
 Group element: Word
242. [Glass Hose] [Correct left to right parse]: A hose made from glass.
Hose:
 Made from: Glass
243. [Case-Grammar] Theory] [Questionable slot used]: Theory based on the use of a grammar using cases of words.
Theory:
 Subject: Case-Grammar
244. [Sentence Meaning] [Correct left to right parse]: Meaning of a sentence.
Meaning:
 Meaning of: Sentence
245. [Case Slots] [No meaning found]: Slots based on the cases of words.
246. [Sentence Analysis] [Correct left to right parse]: Analysis of a sentence.
Analysis
 Thing Analyzed: Sentence
247. [Case Grammar] [No meaning found]: Grammar based on the cases of words.
248. [Situation Parts] [Correct left to right parse]: Parts of a situation.

Part:

Part of: Situation

249. [{Extensible}-Relation] Representation] [Correct left to right parse]: Representation based on extensible-relations.

Representation:

Represented by: Extensible-Relation

250. [Information Content] [Questionable slot used]: Information contained in a sentence.

Content:

Type: Information

251. [State Relation] [Multiple meanings due to word ambiguity]: Relations that specify the state of something.

Relation:

Type: State [the state something is in or part of a country]

252. Case Names [Multiple meanings due to word ambiguity]: Name of the cases that a word can fill.

Name:

Name of: Case [Case of a word or letter or court case]

253. [[Agent-Act-Object] Combination] [Union of concepts forms meaning]: Combination of agent, act, and object.

Combination:

Combination of: Agent-Act-Object

254. [Comment Frame] [Compound defined in the text]: A frame that comments on an Agent-Act-Object Combination.

255. [[Frame-Slot-Value] Combination] [Union of concepts forms meaning]: Combination of frame, slot, and value.

Combination:

Combination of: Frame-Slot-Value

256. [Knowledge Representation] [Correct left to right parse]: Representation of knowledge:

Representation:

Thing represented: Knowledge

257. [Situation Description] [Correct left to right parse]: Description of a situation.

258. [Input Transparency] [No meaning found]: Input of information to a system is made easy.

259. [Translator Complexity] [Correct left to right parse]: Complexity of a program to translate information from one form to another.

Complexity:

Complexity of: Translator

260. [Case Information] [Multiple meanings due to word ambiguity]: Information about case of a word.

Information:

Information about: Case

261. [[Case-Filler] Combination] [Union of concepts forms meaning]: Combination of a case and its filler.

Combination:

Combination of: Case-Filler

262. [Input Language] [First noun provides a slot for the second]: Language that is input to a program.

263. [Reference Feature] [Correct left to right parse]: Feature that is used to refer to something easily.

Feature:

Feature of: Reference

264. [Control Scheme] [Correct left to right parse]: Scheme for control of something.

Scheme:

Scheme for: Control

265. [Story Plot] [Correct left to right parse]: Plot of a story.

Plot:

Plot of: Story

266. [Relation Pairs] [Correct left to right parse]: Pairs of relations.

Pair:

Pair of: Relation

267. [Input Interface] [Questionable slot used]: Interface to a program for input of data.

Interface:

Type: Input

268. [Similarity Score] [Questionable slot used]: Score that shows how similar two frames are.

Score:

Type: Similarity

269. [Scoring Module] [Correct left to right parse]: Module that computes the score.

Module:

Module function: Scoring

270. [Total Score] [No meaning found]: The score after all computation is finished.

271. [Class Abstraction] [Correct left to right parse]: Abstraction of the classes that data falls into.

Abstraction:

Abstraction of: Class

272. [Class Membership] [Correct left to right parse]: Membership of an item in a particular class.

Membership:

Membership in: Class

273. [Analogy Process] [Multiple meanings due to slot ambiguity]: The process of using an analogy in reasoning about a problem.

Process:

Process type: Analogy

Processes: Analogy

274. [Cause Relation] [Questionable slot used]: Relation based on the cause for some effect.

Relation:

Type: Cause

275. [Act Abstraction] [Correct left to right parse]: Abstraction of an act that was performed.

Abstraction:

Abstraction of: Act

276. [{Tragic} Event] Situation [Correct left to right parse]: Situation specifying an event that is tragic.

Situation:

Situation of: Tragic Event

277. [Feature Sets] [Correct left to right parse]: A set whose members are features.

Set:

Member: Feature

278. [Water Pressure] [Correct left to right parse]: The pressure of water in a pipe.

Pressure:

Pressure of: Water

279. [Water Flow] [Correct left to right parse]: The flow of water in a pipe.

Flow:

Thing flowing: Water

280. [Symbol Arrangement] Conventions [Correct left to right parse]: Conventions for the arrangement of symbols.

Convention:

Convention of: Symbol Arrangement

Arrangement:

Thing Arranged: Symbol

281. [Pipe Law] [Multiple meanings due to word ambiguity]: Physical law about the behaviour of water in pipes.

Law: [Physical or Legal]

About: Pipe

282. [Water Pipe] Situation [Correct left to right parse]: Situation involving a particular arrangement of water pipes.

Situation:

Situation of: Water Pipe

283. [Pipe Resistance] [Multiple meanings due to slot ambiguity]: Resistance to water flowing in a pipe.

Resistance:

Thing resisted: Pipe

Thing causing resistance: Pipe

284. [Resistance Situation] [Correct left to right parse]: A situation in which resistance is involved.

Situation:

Situation of: Resistance

285. [[Water Pipe] Analogy] [Correct left to right parse]: Analogy of an electrical circuit to water pipes.

Analogy:

Analogy to: Water Pipe

286. [Key Assumption] [Wrong meaning found]: Assumption made that is crucial to solving a problem.

Assumption:

About: Key

287. [Cause Structure] [Correct left to right parse]: Structure built up to represent the causes of effects.

Structure:

Structure of: Cause

288. [[[Common]-Sense] Reasoning] [Questionable slot used]: Reasoning done using common sense.

Reasoning:

Type: Common-Sense

289. [Situation Analysis] [Correct left to right parse]: Analysis of a situation.

Analysis:

Thing Analyzed: Situation

290. [Similarity Net] [Questionable slot used]: A net showing similarity between concepts.

Net:

Net Element: Similarity

291. [Data Base] [No meaning found]: A store of information or data.

292. [Information Retrieval] [Correct left to right parse]: Retrieval of information.

Retrieval:

Thing retrieved: Information

293. [Classification Information] [Wrong meaning found]: Information used to classify items.

Information:

About: Classification

294. [List Structure] [Correct left to right parse]: Structure of a list.

Structure:

Structure of: List

295. [Deduction System] [Correct left to right parse]: System used to perform deduction.
System:

Purpose: Deduction

296. [Miniature Analogy] [Wrong meaning found]: Use of a small analogy to learn the use of something.

Analogy:

About: miniature

297. [Resistor Law] [Multiple meanings due to word ambiguity]: Physical law about resistors in electrical circuits.

Law: [Physical or Legal]

About: Resistor

298. [Capacitor Law] [Multiple meanings due to word ambiguity]: Physical law about capacitors in electrical circuits.

Law:

About: Capacitor

299. [Inductance Law] [Multiple meanings due to word ambiguity]: Physical law about inductances in electrical circuits.

Law:

About: Inductance

300. [Damper Law] [Multiple meanings due to word ambiguity]: Physical law about the way physical systems are damped.

Law:

About: Damper

301. [Momentum Law] [Multiple meanings due to word ambiguity]: Physical law about momentum of physical objects.

Law:

About: Momentum

302. [Spring Law] [Multiple meanings due to word ambiguity]: Physical law of the behaviour of springs that bounce.

Law:

About: Spring

303. [Learning Ideas] [Correct left to right parse]: Ideas about how learning is performed.

Idea:

Idea about: Learning

304. [Reasoning Ideas] [Correct left to right parse]: Ideas about how reasoning is done.

Idea:

Idea about: Reasoning

305. [Learning System] [Correct left to right parse]: System that is able to learn.

System:

Purpose: Learning

306. [{Geometric} Analogy] Problem [Questionable slot used]: Problem of using analogy in geometric problems.

Problem:

Problem of: Geometric Analogy

307. [Key Representation Idea] [Wrong meaning found]: Idea about how the primary needs for representation of knowledge.

Idea:

Idea About: Key Representation

Representation:

Thing Represented: Key

308. [Discovery System] [Correct left to right parse]: System that discovers new theorems.

System:

Purpose: Discovery

309. [Computer Learning] [Multiple meanings due to slot ambiguity]: Learning done by a computer.

Learning:

About: Computer

Thing doing the learning: Computer

B.3. Examples from an Article in a Book

The examples here come from the article *What's in a Link: Foundations for Semantic Networks* by Woods, W. A. in *Representation and Understanding* edited by Bobrow, D. G. and Collins, A. M. published by Academic Press, New York, 1975)

310. [{Semantic} Network] Representation [Correct left to right parse]: Representation using semantic networks.

Representation:

Represented by: Semantic Network

311. [{Conceptual} Dependency] Representation [Correct left to right parse]: Representation using Conceptual Dependency.

Representation:

Represented by: Conceptual Dependency

312. [{Semantic} Network] Structure [Correct left to right parse]: Structure of a semantic network.

Structure:

Structure of: Semantic Network

313. [Solution Technique] [Correct left to right parse]: Technique for arriving at a solution to a problem.

Technique:

Technique for: Solution

314. [[{Semantic} Network] Notation] [Correct left to right parse]: Notation for a semantic network.
Notation:

Notation for: Semantic Network

315. [[{Programming Language} Theorists] [Correct left to right parse]: Theorists who work with languages for programming computers.

Theorists:

Subject: Programming Language

Language:

Language for: Programming

316. [Truth Conditions] [Questionable slot used]: Conditions to establish the truth of a statement.
Condition:

Condition for: Truth

317. [Truth Values] [Wrong meaning found]: The values used when working with logic.

Value:

Value of: Truth

318. [Inference Technique] [Correct left to right parse]: Technique for performing inferences.

Technique:

Technique for: Inference

319. [Word Sequence] [Correct left to right parse]: Sequence whose elements are words.

Sequence:

Sequence Element: Word

320. [Inference Capabilities] [Correct left to right parse]: Inferences that something is capable of performing.

Capability:

Thing capable of: Inference

321. [Retrieval Capabilities] [Correct left to right parse]: Capability to retrieve certain information.

Capability:

Thing capable of: Retrieval

322. [Language Processing] [Correct left to right parse]: Processing of language.

Processing:

Thing Processed: Language

323. [Inference Process] [Correct left to right parse]: The process of performing an inference.

Process:

Process type: inference

324. [Input Sentence] [First noun provides a slot for the second]: Sentence input to a computer program.

325. [[Programming Language] Theory] [Correct left to right parse]: Theory about languages used in programming computers.
Theory:
Subject: Programming Language
326. [Case Representation] [Questionable slot used]: Representation using the cases of words.
Representation:
Represented by: Case
327. [[{Canonical} Form] Function] [Correct left to right parse]: Function that translates something into canonical form.
Function:
Result: Canonical Form
328. [Data Base] [No meaning found]: Store of information or data.
329. [Inference Chain] [Correct left to right parse]: Chain of inferences needed to reach a conclusion.
Chain:
Elements of Chain: Inference
330. [Retrieval Process] [Correct left to right parse]: Process used to retrieve data.
Process:
Process Type: Retrieval
331. [[{Canonical} Form] Mechanism] [Correct left to right parse]: Mechanism for using canonical forms to hand equivalence.
Mechanism:
Mechanism for: Canonical Form
332. [[{Canonical} Form] Solution] [Wrong meaning found]: Solution of equivalence problem using canonical form.
Solution:
Solution of Problem: Canonical Form
333. [Family Unit] [Questionable slot used]: Family as a single unit.
Unit:
Unit Name: Family
334. [Network Notation] [Wrong meaning found]: Notation for a semantic network.
Notation:
Notation for: Network
335. [Morning Star] [Wrong meaning found]: A planet seen in just before sunrise.
Star:
Time: Morning
336. Evening Star [Wrong meaning found]: A planet seen during the evening.
Star:

Time: Evening

The system would not realize that the above two stars are planets and also that they are likely to be the same. Would people if they heard these phrases without having the added knowledge that they are really the same.

337. [[Computer Science] Application] [Correct left to right parse]: Application of the science that studies computers.

Application:

Application of: Computer Science

Science:

Studies: Computer

338. [Property List] [Correct as far as it goes]: Property list in the Lisp sense.

List:

List element: Property

339. [[Attribute-Value] Pair] [Union of concepts forms meaning]: A pair consisting of an attribute and a value.

Pair:

Element: Attribute-Value

340. [Attribute Name] [Correct left to right parse]: Name of an attribute.

Name:

Name of: Attribute

341. [Attribute Link] [Questionable slot used]: Link that represents the meaning of an attribute.

Link:

Type: Attribute

342. [Link Name] [Correct left to right parse]: Name of a link.

Name:

Name of: Link

343. [Noun Phrase] [Wrong meaning found]: Linguistic sense that is more encompassing than phrase containing only nouns.

Phrase:

Elements: Noun

344. [[Case Structure] Representation] [Multiple meanings due to word ambiguity]: Representation of knowledge by structures based on the cases of words.

Representation:

Represented by: Case Structure

Structure:

Structure of: Case

345. [Knowledge Representation] [Correct left to right parse]: Representation of knowledge.
Representation:
Represents: Knowledge
346. [{Relative} Clause] Relationship [Questionable slot used]: Relationship specified by a relative clause.
Relationship:
Type: Relative Clause
347. [{Relative} Clause] Mechanism [Questionable slot used]: Mechanism used to represent relative clauses.
Mechanism:
Used for: Relative Clause
348. [Stage Directions] [Wrong meaning found]: Directions used to cause something to happen.
Directions:
Direction to: Stage
349. [{Relative} Clause] Problem [Questionable slot used]: Problem of dealing with relative clauses.
Problem:
Type: Relative Clause
350. [Memory Contents] [Multiple meanings due to word ambiguity]: Contents contained in memory.
Contents:
Contents of: Memory [Human or Computer]
351. [Transient Process] Account [Questionable slot used]: Account of a process that is transient.
Account:
Account of: Transient Process
Process:
Type: Transient
352. [Search Specification] [Correct left to right parse]: Specification for a search for information.
Specification:
Thing Specified: Search
353. [Slot Filler] [Correct left to right parse]: Filler that fills a slot.
Filler:
Fills: Slot
354. [Network Structure] [Wrong meaning found]: Structure of a semantic network.
Structure:
Structure of: Network
355. [Quantifier Prefix] [Wrong meaning found]: In logic the quantifiers before a statement.
Prefix:
Prefix of: Quantifier

356. [[[Higher]-Operator] Representation] {of Quantification}] [Correct left to right parse]:
Representation of quantification using higher-operators.
Representation:
 Represented by: Higher-Operator
357. [[Case Representation] Notation] [Correct left to right parse]: Notation for representing things
using the cases of words.
Notation:
 Notation of: Case Representation
358. [Inverse [Agent Link]] [Wrong meaning found]: The inverse of the link that represents an agent.
Link:
 Type: Inverse Agent
359. [Patient Link] [Correct as far as it goes]: Link that is used to represent the patient case.
Link:
 Type: Patient
360. [Quantifier Node] [Correct left to right parse]: Node used to represent a quantifier.
Node:
 Represents: Quantifier
361. [Theorem Proving] [Correct left to right parse]: Proving of a theorem.
Proving:
 Thing Proved: Theorem
362. [Resolution Theorem Proving] [Wrong meaning found]: Proving of a theorem using the
Resolution method.
Proving:
 Thing Proved: Resolution Theorem
363. [[[Natural] Language] Dialog] [Questionable slot used]: A dialog taking place in a natural
language.
Dialog:
 Dialog uses: Natural Language
364. [[Skolem Function] Notation] [Multiple meanings due to word ambiguity]: Notation using
Skolem functions.
Notation:
 Notation using: Skolem Function
 Notation for: Skolem Function
365. [[Theorem Proving] Circles] [No meaning found]: People working in the area of theorem
proving.
366. [[Skolem Form] Expression] [Correct left to right parse]: Expression using Skolem form to
represent information.
Expression:
 Expression using: Skolem Form

367. [Negation Operator] [Correct left to right parse]: Operator that performs a negation.
Operator:
Operation: Negation
368. [[Quantifier Prefix] Form] [Wrong meaning found]: Form containing quantifier prefixes.
Form:-
Form of: Quantifier Prefix
369. [Church's Lambda Notation] [No meaning found]: Notation using lambda that was developed by Church.
370. [Variable Name] [Correct left to right parse]: Name of a variable.
Name:
Name of: Variable
371. [Lambda Expression] [Questionable slot assignment]: Expression using lambda notation.
Expression:
Consists of: Lambda
372. [Lambda Abstraction] [Correct as far as it goes]: Abstraction of information using lambda notation.
Abstraction:
Abstraction of: Lambda
373. [{Higher}-Operator Quantifier Representation] [Wrong meaning found]: Higher-operator representation of quantification.
Representation:
Represents: Quantifier
Represented by: Higher-Operator
374. [Lambda Operator] [Questionable slot assignment]: operator that operates on lambda notations.
Operator:
Operates on: Lambda
375. [{Semantic} Network] Formalism] [Correct left to right parse]: Formalism developed for semantic networks.
Formalism:
Thing formalized: Semantic Network
376. [Human Performance] [Correct left to right parse]: Performance as would be done by a human.
Performance:
Performance by: Human

B.4. Examples from the April 27, 1981 Newsweek Article on the Space Shuttle

377. [Rocket Engine] [Correct left to right parse]: An engine used to propel a rocket.

Engine:

Part of: Rocket

378. [Flight Director] [Correct left to right parse]: Director who directs a flight.

Director:

Directs: Flight

379. [Mission Control] [Correct left to right parse]: The thing that controls a mission.

Control:

Controls: Mission

380. [Fuel Tank] [Correct left to right parse]: Tank that contains fuel.

Tank:

Contains: Fuel

381. [Liquid Oxygen] [No meaning found]: Oxygen that is in liquid form.

382. [Elbow Room] [Wrong meaning found]: Room for people to move around without knocking elbows.

Room:

Contains: Elbow

Room:

Room for: Elbow

383. [Relay Station] [Correct left to right parse]: Station used to relay signals.

Station:

Purpose: Relay

384. [Space Shuttle] [Questionable slot used]: Shuttle used to get things into space.

Shuttle:

Travels in/on: Space

385. [Satellite Communications] [Correct left to right parse]: Communication done via satellites.

Communications:

Communicate with/via: Satellite

386. [Test-Flight Program] [Correct left to right parse]: Schedule of flights to test the space shuttle.

Program:

Schedule for: Test-Flight

387. [Space Business] [Correct left to right parse]: Business related to the development of space.

Business:

Location: Space

388. [Veteran Pilot] [First noun provides a slot for the second]: A pilot who has flown a lot.
389. [[Space-Suit] Helmet] [No meaning found]: Helmet of a suit worn in space to protect people.
 Helmet:
 Part of: Space-Suit
 Space-Suit:
390. [Computer Glitch] [Correct left to right parse]: A problem with a computer.
 Glitch:
 Thing doing the glitching: Computer
391. [Engine Pod] [Correct left to right parse]: Pod that contains an engine.
 Pod:
 Encloses: Engine
392. [Heat Shield] [Correct left to right parse]: Shield used to protect something from heat.
 Shield:
 Shielded from: Heat
393. [{Main}-Tank Attachment Point] [Correct middle ternary parse]: Point used to attach the main tank to the space shuttle.
 Attachment:
 Thing attached: Main-Tank
 Attached to: Point
394. [Explosive Separator] [Questionable slot assignment]: Thing used to separate fuel tanks from shuttle with explosives.
 Separator:
 Separated with: Explosive
395. [[Ground-Crew] Work] [Correct left to right parse]: Word done by a crew located on the ground.
 Work:
 Worker: Ground-Crew
 Crew:
 Location: Ground
396. [Tow Vehicle] [Questionable slot assignment]: Vehicle used to tow some other vehicle.
 Vehicle:
 Purpose: Tow
397. [{Post}-Flight] Picture] [Questionable slot assignment]: Pictures taken after the flight of the shuttle.
 Picture:
 Time: Post-Flight
398. [Ground Crew] [Correct left to right parse]: Crew that works on the ground.
 Crew:
 Location: Ground

399. [Fuel Lines] [No meaning found]: Pipes used to deliver fuel to engines from the fuel tank.

400. [Shuttle Operation Director] [Correct middle ternary parse]: Director who directs the operation of the shuttle.

Director:

Directs: Shuttle Operation

Operation:

Thing Operated: Shuttle

Operation:

Thing Operated: Shuttle

Operator: Director

401. [[Solid]-Fuel] Boosters [Correct left to right parse]: Boosters that are fueled with solid-fuel.

Booster:

Fueled by: Solid-Fuel

402. [Computer Hours] [No meaning found]: Time spent by a computer to calculate information.

403. [Thruster Dance] [Questionable slot assignment]: Dance done by the thrusters on take off.

Dance:

Dancer: Thruster

404. [[Cargo Bay] Doors] [Correct left to right parse]: Doors allowing access to a bay for containing cargo.

Door:

Door of: Cargo Bay

Bay:

Contains: Cargo

405. [Payload Bay] [Correct left to right parse]: A bay used to contain the payload of the shuttle.

Bay:

Contains: Payload

406. [Flight Recorders] [Correct left to right parse]: Recorders used to record a flight.

Recorder:

Records: Flight

407. [Television Camera] [Questionable slot used]: A type of camera used to take television pictures.

Camera:

Type: Television

408. [Cargo Bay] [Correct left to right parse]: Bay that contains cargo.

Bay:

Contains: Cargo

409. [Ascent Leader] [Correct left to right parse]: Person or thing that leads the ascent of the shuttle.

Leader:

Directs: Ascent

410. [Flight Operation] [Correct left to right parse]: Operation involving the flight of the shuttle.
Operation:
 Thing Operated: Flight
411. [Entry Profile] [Correct left to right parse]: The profile of the entry of the shuttle.
Profile:
 Thing Profiled: Entry
412. [Spy Satellite] [Questionable slot used]: A satellite used to perform spying.
Satellite:
 Purpose: Spy
413. [Tile Hunt] [Correct left to right parse]: A hunt where the thing searched for is a tile.
Hunt:
 Thing hunted: Tile
414. [Security Considerations] [Correct left to right parse]: Thing considered is the security of something.
Consideration:
 Thing considered: Security
415. [Radio Channel] [Correct left to right parse]: Channel used to transmit radio messages.
Channel:
 Type: Radio
416. [Ground Controllers] [Correct left to right parse]: Controllers of the shuttle located on the ground.
Controller:
 Location: Ground
417. [Space Rookie] [No meaning found]: Someone who has never been in space before.
418. [Shoulder Strap] [Questionable slot assignment]: A strap used to secure the shoulder of a person.
Strap:
 Secures: Shoulder
419. [Entry Trajectory] [Correct left to right parse]: Trajectory of the entry of the shuttle into atmosphere.
Trajectory:
 Trajectory of: Entry
420. [T-38 Chase Plane] [Questionable slot assignment]: A plane with model number T-38 whose purpose is to chase things.
Plane:
 Purpose: Chase
 Type: T-38
421. [Space Agency] [No meaning found]: Agency that is involved in exploring space.

422. [Space Vehicle] [Correct left to right parse]: A vehicle that travels in space.
Vehicle:
Travels on/in: Space
423. [Space Truck] [Correct left to right parse]: A truck like object that travels in space.
Truck:
Travels on/in: Space
424. [Corporate Jet] [First noun provides a slot for the second]: A jet used to fly corporation officers.
425. [Launch Vehicle] [Questionable slot used]: A vehicle used to launch the shuttle into orbit.
Vehicle:
Purpose: Launch
426. [Payload Area] [Correct left to right parse]: Area taken up by the payload for the shuttle.
Area:
Area of: Payload
427. [Radio Technology] [Questionable slot used]: Technology needed to use radio.
Technology:
Type: Radio
428. [Radio Platform] [Correct left to right parse]: A platform for using radio.
Platform:
Platform for: Radio
429. [Communication Network] [Questionable slot used]: Network over which communication happens.
Network:
Purpose: Communication
430. [Plant Metabolism] [Correct left to right parse]: The metabolism of a living plant.
Metabolism:
Metabolism of: Plant
431. [Animal Metabolism] [Correct left to right parse]: The metabolism of a living animal.
Metabolism:
Metabolism of: Animal
432. [Space Colony] [Correct left to right parse]: A colony located in space.
Colony:
Location: Space
433. [Space History] [Questionable slot assignment]: The history of man's exploration of space.
History:
History of: Space
434. [Astronaut Crew] [Correct left to right parse]: A crew whose members consist of astronauts.
Crew:

Crew member: Astronaut

435. [Space Telescope] [Correct left to right parse]: A telescope located in space.

Telescope:

Location: Space

436. [[[Deep]-Space] Probe] [Correct left to right parse]: A probe used to probe space not near the Earth.

Probe:

Thing Probed: Deep-Space

437. [[[Low]-Earth] Orbit] [No meaning found]: Orbit that is close to the Earth.

438. [Earth's [Sister Planets]] [No meaning found]: Venus and Mars.

439. [Ball Bearings] [No meaning found]: Bearings shaped like a ball-used to reduce friction.

440. [[[Space-Processing]-Program] Manager] [Questionable slot assignment]: A manager who manages a program to do certain processing in space.

Manager:

Manages: Space-Processing-Program

Program:

Schedule for: Space-Processing

Processing:

Thing Processed: Space

441. [[Power Utilization] Platform] [Questionable slot assignment]: A platform that is to utilize power.

Platform:

Platform for: Power Utilization

Utilization:

Thing utilized: Power

442. [Power Platform] [No meaning found]: meaning as above.

443. [Shuttle Mission] [Correct left to right parse]: A mission that uses the space shuttle.

Mission:

Mission by: Shuttle

444. [Space Suit] [No meaning found]: A suit used in space to protect men.

445. [Sales Promotion] [Correct left to right parse]: Promotion of sales of space on the shuttle.

Promotion:

Thing promoted: Sales

B.5. Examples from Articles in the New York Times

The examples in this section came from several articles in the New York Times newspaper during the last couple of weeks of April, 1981.

446. [Oil Executive] [Correct left to right parse]: Executive who works in the oil industry.

Executive:

Executive of industry: Oil

447. [Petroleum Industry] [Correct left to right parse]: Industry that produces petroleum and associated products.

Industry:

Product: Petroleum

448. [Price Tag] [Questionable slot used]: Tag used to specify the price of an item for sale.

Tag:

Specifies: Price

449. [Surveillance Plane] [Correct left to right parse]: A plane used in the surveillance of something.

Plane:

Purpose: Surveillance

450. [Grain Sale] [Correct left to right parse]: Sale of grain.

Sale:

Thing Sold: Grain

451. [Grain Exports] [Correct left to right parse]: Exports that are grain.

Export:

Thing exported: Grain

452. [Export Curb] [Correct left to right parse]: Curb of exports of grain in particular.

Curb:

Thing curbed: Export

453. [Grain Decision] [Questionable slot used]: Decision of whether to export grain to Russia.

Decision:

Decide about: Grain

454. [Policy Matters] [Correct left to right parse]: Matters associated with the policy of the administration.

Matters:

Concerning: Policy

455. [Campaign Pledge] [Questionable slot used]: Pledge made during a campaign for political office.

Pledge:

Time: Campaign

456. [Grain Curb] [No meaning found]: Curb of the sale of grain to Russia.
457. [Military Pressure] [Correct left to right parse]: Pressure brought to bear by the military.
Pressure:
 Pressure by: Military
458. [Military Preparation] [Multiple meanings due to slot ambiguity]: Preparation by the military for action if necessary.
Preparation:
 Preparer: Military
 Thing Prepared: Military
459. [Farm Bill] [Multiple meanings due to word ambiguity]: A Bill in Congress dealing with farms.
Bill: invoice
 Bill for: Farm
Bill: legislative
 Topic: Farm
460. [Grain Agreement] [Correct left to right parse]: Agreement about the sale of grain to Russia.
Agreement:
 About: Grain
461. [Price Index] [Wrong meaning found]: Index showing how prices change over time.
Index:
 Thing indexed: Price
462. [Energy Prices] [Correct left to right parse]: Prices for energy.
Price:
 Price of: Energy
463. [[Oil Price] Controls] [Correct left to right parse]: Controls used to control the price of oil.
Control:
 Thing Controlled: Oil Price
Price:
 Price of: Oil
464. [Inflation Figures] [Multiple meanings due to word ambiguity]: Figures showing the current rate of inflation.
Figure: numbers
 Data about: Inflation
Figure: diagram
 Figure showing: Inflation
Figure: of person.
465. [{Single}-Digit] Rates [Questionable slot used]: Change in inflation is less than 10%.
Rate:
 Measured in: Single-Digit
466. [[Cost-of-Living] Increase] [Correct left to right parse]: Increase in the cost of living.

Increase:

Thing increased: Cost-of-Living

467. [Consumer Prices] [Multiple meanings due to slot ambiguity]: Price charged to consumers.

Price:

Price of: Consumer

Price charged to: Consumer

Price charged by: Consumer

468. [[Home Purchase] Price] [Correct left to right parse]: Price paid to buy a home.

Price:

Price of: Home Purchase

Purchase:

Thing purchased: Home

469. [Energy Cost] [Correct left to right parse]: Cost of energy.

Cost:

Cost of: Energy

470. [Comparison Report] [Correct as far as it goes]: Report on the comparison of prices.

Report:

About: Comparison

471. [Production Worker] [First noun provides a slot for the second]: Worker who produces a product rather than a service.

472. [Production Performance] [Correct left to right parse]: Amount of product produced.

Performance:

Performance of: Production

473. [Money Supply] [Multiple meanings due to word ambiguity]: The amount of money in circulation.

Supply: action of supplying something

Thing supplied: Money

Supply: amount available

Amount of: Money

474. [Money Creation] [Correct left to right parse]: Creation of money.

Creation:

Thing Created: Money

475. [[Money Supply] Series] [Correct left to right parse]: Series of figures showing the money supply over time.

Series:

Element: Money Supply

476. [Oil Prices] [Correct left to right parse]: Price of oil.

Price:

Price of: Oil

477. [Transportation Component] [No meaning found]: The component of inflation caused by transportation costs.
478. [Housing Component] [No meaning found]: The component of inflation caused by the cost to house people.
479. [Food Prices] [Correct left to right parse]: Price of food.
Price:
Price of: Food
480. [Beverage Prices] [Correct left to right parse]: Price of beverage.
Price:
Price of: Beverage
481. [Gasoline Prices] [Correct left to right parse]: Price of gasoline.
Price:
Price of: Gasoline
482. [Detroit's [Rebate Program]] [No meaning found]: Car makers giving rebates for people to buy cars.
483. [{{New}-Car} Index] [Correct left to right parse]: Index of how many new cars are sold during a period of time.
Index:
Thing indexed: New-Car
484. [Food Element] [No meaning found]: The part of inflation caused by increases in food prices.
485. [Beverage Element] [No meaning found]: The part of inflation caused by increases in beverage prices.
486. [[Savings and Loan] Association] [No meaning found]: An institution that provides many of the services of a bank
487. [Interest Rate] [Multiple meanings due to word ambiguity]: Rate of interest to borrow money.
Rate:
Rate-of: Interest [monetary or attention]
488. [Credit Market] [Correct left to right parse]: The market where credit can be sold or bought.
Market:
Thing marketed: Credit
489. [[Treasury Bill] Rates] [Correct left to right parse]: The rate paid by credit instruments sold by the Treasury of the U.S.
Rate:
Rate of: Treasury Bill
Bill:
Issued by: Treasury

490. [Consumer Representative] [Multiple meanings due to slot ambiguity]: Representative who represents consumer interests.

Representative:

Represents: Consumer

Belongs to: Consumer

491. [Savings Executive] [Correct left to right parse]: An executive who works in the savings industry.

Executive:

Executive of industry: savings

492. [{Variable} Rate] Mortgages [Questionable slot used]: Mortgages whose interest rate changes due to credit conditions.

Mortgage:

Type: Variable Rate

493. [Home Financing] [Correct left to right parse]: Financing the purchase of a home.

Financing:

Thing financed: Home

494. [{Fixed}-Rate] Home Mortgage [Questionable slot used]: Mortgage whose interest rate stays constant.

Mortgage:

Thing mortgaged: Home

Type: Fixed-Rate

495. [Home Loans] [Correct left to right parse]: Loan to purchase a home.

Loan:

Loan for: Home

496. [Construction Industry] [Correct left to right parse]: Industry that constructs new buildings.

Industry:

Type: Construction

497. [Savings Units] [Wrong meaning found]: Institutions at which people save money.

Units:

Measure of: Saving

498. [Consumer Groups] [Correct left to right parse]: Groups whose members are consumers.

Group:

Group Member: Consumer

499. [{Variable} Rate] Instrument [Questionable slot used]: A financial obligation with a variable interest rate.

Instrument:

Type: Variable Rate

500. [Home Ownership] [Correct left to right parse]: The ownership of a home.

Ownership:

Thing owned: Home

501. [[Interest Rate] Fluctuations] [Correct left to right parse]: Change in the interest rate.
 Fluctuations:
 Thing fluctuating: Interest Rate
502. [{Adjustable} [Mortgage Loan]] [Correct left to right parse]: Loan for a mortgage that has an adjustable interest rate.
 Loan:
 Purpose: Mortgage
503. [[Loan Repayment] Schedule] [Correct left to right parse]: Schedule for the repayment of a loan.
 Schedule:
 Thing scheduled: Loan Repayment
 Repayment:
 Thing Repayed: Loan
504. [Loan Balance] [Correct left to right parse]: The amount of money left to pay off a loan.
 Balance:
 Balance of: Loan
505. [Industry Trade Group] [Wrong meaning found]: Group that represents a particular industry.
 Group:
 Group Element: Industry Trade
 Trade:
 Trade of: Industry
506. [{Free-Floating} [[Interest Rate] Adjustments]] [Correct left to right parse]: Free-floating adjustments to the interest rate.
 Adjustments:
 Thing Adjusted: Interest Rate
507. [[Housing Finance] Sector] [No meaning found]: Sector of the economy responsible for financing housing.
508. [News Conference] [Correct as far as it goes]: Conference called to impart news to the public.
 Conference:
 Topic: News
509. [Mortgage Rate] [Correct left to right parse]: Interest rate charged on a mortgage.
 Rate:
 Rate of: Mortgage
510. [Home Buyers] [Correct left to right parse]: People who buy homes.
 Buyer:
 Thing Bought: Home
511. [[Interest Rate] Changes] [Correct left to right parse]: Changes in the interest rate.

Change:

Thing Changed: Interest Rate

512. [Consumer Safeguards] [Multiple meanings due to slot ambiguity]: Safeguards that protect the consumer.

Safeguard:

Thing safeguarded: Consumer

Thing safeguarded against: Consumer

Safeguarder: Consumer

513. [[Interest Rate] Increase] [Correct left to right parse]: Increase in the current interest rate.

Increase:

Thing Increased: Interest Rate

514. [Deposit [Interest Rate]] [Questionable slot used]: Interest rate paid by savings institutions for deposits.

Interest Rate:

Rate for: Deposit

515. [Space Shuttle] [Questionable slot used]: Shuttle used to get things into space.

Shuttle:

Travels in/on: Space

516. [[Space Agency] Official] [Questionable slot used]: Official employed by the agency responsible for activities in space.

Official:

Employer: Space Agency

Agency:

Concerned with: Space

517. [Shuttle Operations] [Correct left to right parse]: Operations where the shuttle is the thing operated.

Operation:

Thing operated: Shuttle

518. [[Flight Test] Manager] [Correct left to right parse]: Manager who manages the test of the flight of the shuttle.

Manager:

Manages: Flight Test

Test:

Thing tested: Flight

519. [Test Flight] [First noun provides a slot for the second]: Test the flight worthiness of the space shuttle.

520. [[[Hard]-Clay] Surface] [Correct left to right parse]: Surface made from hard-clay.

Surface:

Made of: Hard-Clay

521. [Heroin Addict] [Correct left to right parse]: A person addicted to heroin.
Addict:
 Thing addicted to: Heroin
522. [Eyewitness Account] [Correct left to right parse]: An account of something by an eyewitness.
Account:
 Accounter: Eyewitness
523. [Needle Mark] [Correct left to right parse]: Mark made by a needle.
Mark:
 Made-by: Needle
524. [Executive Editor] [No meaning found]: An official of a newspaper.
525. [Police Search] [Multiple meanings due to slot ambiguity]: Search made by the police for a boy.
Search:
 Search for: Police
 Search by: Police
 Thing Searched: Police
526. [Burn Victim] [Correct left to right parse]: A person who has been burned by a fire.
Victim:
 Victim of: Burn
527. [Shark Cartilage] [Correct left to right parse]: Cartilage that comes from a shark.
Cartilage:
 Part of: Shark
528. [Skin Grafts] [Questionable slot used]: Grafts made to the skin of a burn victim.
Graft:
 Graft to: Skin
529. [Burn Patient] [Questionable slot used]: A person who has been burned.
Patient:
 Disease, Injury: Burn
530. [Burn Care] [Correct left to right parse]: Care of the burn of someone who has been burned.
Care:
 Thing cared for: Burn
531. [Scar Formation] [Correct left to right parse]: Formation of scar tissue on a burn victim.
Formation:
 Thing Formed: Scar
532. [Assassination Attempt] [Correct left to right parse]: Attempt to assassinate the president.
Attempt:
 Thing attempted: Assassination

533. [Gunshot Wound] [Correct left to right parse]: A wound caused by a gunshot.

Wound:

Cause: Gunshot

534. [Family Friend] [Correct left to right parse]: A friend of the whole family.

Friend:

Friend of: Family

535. [Laboratory Analysts] [Multiple meanings due to slot ambiguity]: Analysts who work in a laboratory.

Analyst:

Location: Laboratory

Employer: Laboratory

536. [Metal Fragments] [Correct left to right parse]: Fragments made from a metal.

Fragment:

Made of: Metal

537. [Television Appearance] [Questionable slot assignment]: Appearance of the president on television.

Appearance:

Location: Television

538. [Radio Address] [Questionable slot assignment]: Address made via radio.

Address:

Via: Radio

539. [Business Possibility] [Correct left to right parse]: Possibility for doing business.

Possibility:

Possibility for: Business

540. [Consumer Market] [Correct left to right parse]: The market for items bought by consumers.

Market:

Buyers: Consumer

541. [Debt Ceiling] [Correct left to right parse]: The maximum amount of debt that the U.S. can have outstanding.

Ceiling:

Maximum limit of: Debt

542. [Spot News] [Photography] [Questionable slot assignment]: Photography of news that happens on the spot.

Photography:

Type: Spot News

News:

Location: Spot

543. [Budget Proposal] [Correct left to right parse]: Proposal for the budget of the U.S.

Proposal:

- Thing proposed: Budget
544. [Budget Cutters] [Correct left to right parse]: People who want to reduce the size of the budget.
Cutter:
Thing cut: Budget
545. [Tax Cut] [Correct left to right parse]: Cut in taxes.
Cut:
Thing cut: Tax
546. [Tax Plan] [Correct left to right parse]: Plan to reduce taxes.
Plan:
Thing planned: Tax
547. [Press Secretary] [Wrong meaning found]: The press secretary of the White House.
Secretary:
Secretary to: Press
548. [Income Tax] [Correct left to right parse]: Tax on the income earned by people.
Tax:
Thing taxed: Income
549. [Security Adviser] [Correct left to right parse]: Person who advises the president on security matters.
Adviser:
Advises about: Security
550. [[Income Tax] Form] [Correct left to right parse]: Form used to calculate the income tax of someone.
Form:
Form for: Income Tax
551. [Tax Rate] [Correct left to right parse]: The rate at which income is taxed.
Rate:
Rate for: Tax
552. [Tax Drive] [No meaning found]: Drive to reduce the taxes paid by people.
553. [[Tax Reduction] Plan] [Correct left to right parse]: Plan that is to reduce tax.
Plan:
Thing Planned: Tax Reduction
Reduction:
Thing reduced: Tax
554. [Recovery Program] [Multiple meanings due to word ambiguity]: Plan for recovery of the economy.
Program: Schedule or Computer Program
Program for: Recovery

555. [Rate Cut] [Correct left to right parse]: Cut in the rate charged for something.

Cut:

Thing cut: Rate

558. [Constituent Pressure] [Questionable slot assignment]: Pressure brought on members of the government by their constituents.

Pressure:

Pressure by: Constituent

559. [Budget Cut] [Correct left to right parse]: Reduction in the size of the budget.

Cut:

Thing Cut: Budget

558. [[{No}-Compromise] Campaign] [Questionable slot used]: Campaign in which no compromised are to be made.

Campaign:

Type: No-Compromise

559. [Administration Leader] [Correct left to right parse]: Leader of the administration.

Leader:

Thing lead: Administration

560. [Government Agent] [Correct left to right parse]: Agent employed by the government.

Agent:

Employed by: Government

561. [Intelligence Division] [No meaning found]: A particular division of the FBI responsible for intelligence operations.

562. [Prosecution Proceedings] [Correct left to right parse]: Proceedings to prosecute someone for a crime.

Proceedings:

Thing proceeded: Prosecution

563. [Bureau Agent] [Questionable slot assignment]: An agent employed by the FBI.

Agent:

Employed by: Bureau

564. [Fugitive Member] [No meaning found]: A member of the Weather Underground organization who is also a fugitive.

565. [Intelligence Community] [No meaning found]: Group of people involved in intelligence operations.

566. [Law Enforcement Official] [Correct middle ternary parse]: An official responsible for the enforcement of the law.

Enforcement:

Enforcer: Official

Thing Enforced: Law

567. [Telephone Call] [Questionable slot used]: Call made via telephone.

Call:

Made via: Telephone

568. [{Black} Bag] Job] [Compound defined in the text]: Illegal entries by FBI agents.

569. [Search Warrant] [Correct left to right parse]: Warrant allowing law officers to perform a search.
Warrant:

Purpose: Search

570. [{Foreign} Intelligence] Search] [Questionable slot assignment]: Search for foreign intelligence operations.

Search:

Search For: Foreign Intelligence

571. [Budget Proposals] [Correct left to right parse]: Proposals for the budget of the U.S.

Proposal:

Proposal for: Budget

572. [Assassination Attempt] [Correct left to right parse]: Attempt to assassinate the president.

Attempt:

Thing attempted: Assassination

573. [{Tax Cut} Proposals] [Correct left to right parse]: Proposals for a cut in taxes.

Proposal:

Proposal for: Tax Cut

Cut:

Thing cut: Tax

574. [Administration Official] [Correct left to right parse]: Official employed by the administration.
Official:

Employed by: Administration

575. [News Conference] [Correct as far as it goes]: Conference called to impart information to the public.

Conference:

Topic: News

576. [{foreign} Policy] Developments] [Correct left to right parse]: Development in the policy towards foreign countries.

Development:

Thing developing: Foreign Policy

577. [News Attention] [Multiple meanings due to slot ambiguity]: Attention focused on something by the news media.

Attention:

Thing attended to: News

Thing attending: News

578. [Grain Sale] [Correct left to right parse]: Sale of Grain.

Sale:

Sale of: Grain

579. [[Income Tax] Cut] [Correct left to right parse]: Cut in the tax on income.

Cut:

Thing Cut: Income Tax

Tax:

Thing taxed: Income

580. [Election Campaign] [Correct left to right parse]: Campaign waged for a person to get elected.

Campaign:

Thing campaigned for: Election

581. [Budget Cut] [Correct left to right parse]: Cut in the budget.

Cut:

Thing Cut: Budget

582. [Budget Package] [Questionable slot assignment]: Package of the whole budget.

Package:

Thing packaged: Budget

583. [Newspaper Editor] [Correct as far as it goes]: Editor who oversees the day to day operation of a newspaper.

Editor:

Edits: Newspaper

584. [Debt Collection] [Correct left to right parse]: Collection where the thing collected is a debt.

Collection:

Thing collected: Debt

585. [Tuition Loan] [Correct left to right parse]: Loan made to someone to pay for tuition.

Loan:

Loan for: Tuition

586. [College Graduate] [Correct left to right parse]: Someone who has graduated from college.

Graduate:

Graduate from: College

587. [Veteran Loan] [Multiple meanings due to slot ambiguity]: Loan to a veteran of the armed services.

Loan:

Loaned to: Veteran

Loaned by: Veteran

588. [Export Loans] [Correct left to right parse]: Loans for exporting products.

Loan:

Loan for: Export

589. [Energy Negotiations] [Correct left to right parse]: Negotiations about energy.
Negotiation:
 Thing negotiated about: Energy
590. [Government Pension Plan] [Multiple meanings due to slot ambiguity]: Plan made by the government for pensions.
Plan:
 Plan for: Pension
 Plan by: Government
Plan:
 Plan for: Government Pension
Pension:
 Provided by: Government
591. [Capital Project] [Wrong meaning found]: Project that requires a large amount of money.
Project:
 Location: Capital
592. [[Oil Production] Cutback] [Correct left to right parse]: Cutback in the production of oil.
Cutback:
 Thing cutback: Oil Production
Production:
 Thing produced: Oil
593. [Resource Revenue] [Correct left to right parse]: Revenue generated by resources.
Revenue:
 Revenue source: Resource
594. [Energy Agreement] [Correct left to right parse]: Agreement about energy.
Agreement:
 Agreement about: Energy
595. [[Oil Company] Earnings] [Correct left to right parse]: Earnings by a company that is involved in oil.
Earnings:
 Earnings of: Oil Company
Company:
 Produces/Provides: Oil
596. [Automobile Sales] [Correct left to right parse]: Sales of automobiles.
Sales:
 Thing sold: Automobile
597. [[[Oil Field] Service] Company] [Multiple meanings due to word ambiguity]: A company that provides service to a field that produces oil.
Company:
 Produces/Provides: Oil Field Service
Service:
 Thing serviced: Oil Field
Field:

- Service: Produces: Oil
 Thing serviced: Oil Field
 Servicer: Company
598. [Oil Exploration] [Correct left to right parse]: Exploration for oil.
 Exploration:
 Thing explored for: Oil
599. [Gas Exploration] [Correct left to right parse]: Exploration for gas.
 Exploration:
 Thing explored for: Oil
600. [[[Health Care] Product] Company] [Questionable slot used]: Company that produces products for taking care of health.
 Company:
 Produces/Provides: Health Care Product
 Product:
 Type: Health Care
 Care:
 Thing cared for: Health
601. [[Brokerage House] Issue] [Multiple meanings due to word ambiguity]: Stock of a company involved in trading securities.
 Issue:
 Issue about: Brokerage House
 House:
 Produces/Provides: Brokerage
 Issue: Stock
 Stock of: Brokerage House
602. [Credit Market] [Correct left to right parse]: Market where instruments of credit is sold and bought.
 Market:
 Thing marketed: Credit
603. [[Corn Futures] Market] [Correct left to right parse]: Market where future production of corn is traded.
 Market:
 Thing marketed: Corn Futures
 Futures:
 Commodity: Corn
604. [Wheat Futures Market] [Correct left to right parse]: Market where future production of wheat is traded.
 Market:
 Thing marketed: Wheat Futures
 Futures:
 Commodity: Wheat

605. [Bond Market] [Correct left to right parse]: Market where bonds are traded.
Market:
 Thing marketed: Bond
606. [Discount Bond] [Questionable slot used]: A bond that is selling at a discount from its denomination.
Bond:
 Type: Discount
607. [Market Rate] [Wrong meaning found]: Rate set by the market.
Rate:
 Rate set by: Market
 Rate of: Market
608. [Stock Price] [Correct left to right parse]: Price of a stock.
Price:
 Price of: Stock
609. [{High} Technology] Issue [Wrong meaning found]: Stock of a company producing high technology products.
Issue:
 Issue about: High Technology
610. [Dow Jones Industrial Average] [No meaning found]: An indicator of the performance of the New York Stock Market.
611. [Oil Exploration] Concern [Multiple meanings due to word ambiguity]: A company involved in the exploration for oil.
Concern:
 Concern about: Oil Exploration
Concern: business
 Produces/Provides: Oil Exploration
612. [Auto Export] [Correct left to right parse]: Export of automobiles.
Export:
 Thing exported: Auto
613. [Record Price] [Wrong meaning found]: The best price for something.
Price:
 Price of: Record [as in stereo]
614. [Technology Stock] [Questionable slot used]: Stock of a company producing technology.
Stock:
 Type: Technology
615. [Research Director] [Correct left to right parse]: Person who directs research.
Director:
 Directs: Research

616. [Technology Company] [Correct left to right parse]: A company that produces technology.
 Company:
 Produces/Provides: Technology
617. [Market Prospect] [Correct left to right parse]: Prospect of what is going to happen to the stock market.
 Prospect:
 Prospect of: Market
618. [Brokerage Group] [Questionable slot assignment]: Group of stock issues of brokerage houses.
 Group:
 Group member: Brokerage
619. [Brokerage Stock] [Questionable slot assignment]: Stock of a brokerage house.
 Stock:
 Type: Brokerage
620. [Market Leader] [Correct left to right parse]: A stock that leads the market.
 Leader:
 Thing lead: Market
621. [Merger Announcement] [Correct left to right parse]: Announcement of a merger of two companies.
 Announcement:
 Thing announced: Merger
622. [{Strong} [Unit Growth]] [Correct left to right parse]: Strong growth in the sales of a particular unit.
 Growth:
 Thing growing: Unit
623. [Earnings Growth] [Correct left to right parse]: Growth in the earnings of a company.
 Growth:
 Thing growing: Earnings
624. [{Global} [Oil Glut]] [Correct left to right parse]: Over supply of oil.
 Glut:
 Glut of: Oil
625. [{Trading Volume} Figures] [Correct left to right parse]: Figures showing the volume of trading on a stock exchange.
 Figures:
 Figures showing: Trading Volume
 Volume:
 Volume of: Trading
626. [{Short} Interest] Ratio [No meaning found]: Ratio of short sales of stock to long sales.
627. [Market Close] [Wrong meaning found]: Time at which the stock market closes.

Close:

Thing closed: Market

628. [Composite Index] [No meaning found]: Index showing what happened to all stocks on the New York Stock Exchange.