THE SEMANTIC INTERPRETATION OF COMPOUND NOMINALS

BY

TIMOTHY WILKING FININ

S.B., Massachusetts Institute of Technology, 1971
M.S., University of Illinois, 1977

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1980

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

February, 1980

WE HEREBY RECOMMEND THAT THE THESIS BY

TIMOTHY WILKING FININ

ENTITLED THE SEMANTIC INTERPRETATION OF COMPOUND NOMINALS

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

[illegible]
Director of Thesis Research

[illegible]
Head of Department

Committee on Final Examination†

[illegible] Chairman

[illegible]

[illegible]

[illegible]

[illegible]

† Required for doctor's degree but not for master's.

O-517

Acknowledgments

I wish to thank the many people who enabled me to complete this work.

I thank my thesis advisor, David L. Waltz, for his support and encouragement for the last five years.

I thank all of the people who have worked at the Coordinated Science Laboratory, making it a unique place to do research.

I thank, in particular, the other members, past and present, of the Advanced Automation Research Group.

I thank my two children, Katie and Peter, who helped me to keep a human perspective.

I thank most of all my friend and wife, Janet Peters Finin, who supplied me with love, guidance and an unending wealth of support and encouragement.

This work was supported in part by the Office of Naval Research under contract number N00014-75-C-0612.

TABLE OF CONTENTS

List of Figures

1 INTRODUCTION

1.1 The Semantic Interpretation of Nominal Compounds

This thesis is an investigation of how a computer can be programmed to understand the class of linguistic phenomena loosely referred to as nominal compounds, i.e. sequences of two or more nouns related through modification. Examples of the kinds of nominal compounds dealt with are: "engine repairs", "aircraft flight arrival", "aluminum water pump", and "noun noun modification".

The interpretation of nominal compounds is divided into three intertwined subproblems: lexical interpretation (mapping words into concepts), modifier parsing (discovering the structure of strings with more than two nominals) and concept modification (assigning an interpretation to the modification of one concept by another). This last problem is the focus of this research. The essential feature of this form of modification is that the underlying semantic relationship which exists between the two concepts is not explicit. Moreover, a large number of relationships might, in principal, exist between the two concepts. The selection of the most appropriate one depends on a host of semantic, pragmatic and contextual factors.

As a part of this research, a computer program has been written which builds an appropriate semantic interpretation when given a string of nouns. This program has been designed as one component of the natural language question answering system JETS. The interpretation is done by a set of semantic interpretation rules. Some of the rules are very specific, capturing the meaning of idioms and canned-phrases. Other rules are very general, representing fundamental case-like relationships which can hold between concepts. A strong attempt has been made to handle as much as possible with the more general, highly productive interpretation rules.

The approach has been built around a frame-based representational system which represents concepts and the relationships between them. The concepts are organized into an abstraction hierarchy which supports inheritance of attributes. The same representational system is used to encode the semantic interpretation rules. An representational system is used to encode the semantic interpretation rules. An important part of the system is the concept matcher which, given two concepts, determines whether the first describes the second and, if it does, how well.

This chapter begins with a short introduction to the topic. The class of linguistic forms referred to as "nominal compounds" is described as well as the notion of "semantic interpretation". The fundamental problem is identified and its study justified. The next section places this research in the larger context of natural language processing systems and the PLANES and JETS systems in particular. The third section highlights the basic themes of my approach to the problem. The forth and final section provides the reader with an outline of the rest of the report.

1.1.1 What is a Nominal Compound?

The term nominal compound means different things to different people. There is no standard definition for it, and, worse yet, there is no standard term which describes the class of objects that this thesis is about. Some closely related terms that have been used by others include complex nominals, nominal phrases and noun-noun modification. The definition of a nominal compound used for this work is:

> A nominal compound is the concatenation of any two or more nominal concepts which functions as a third nominal concept.

There are several features of this definition which I would like to point out.

The first feature is that the implicit definition of a nominal concept is recursive, i.e. a nominal concept can be a combination of two or more nominal concepts. Since a nominal compound is a nominal concept, a nominal compound can be composed of nominal compounds. As a result, I am concerned with compounds composed of an arbitrary number of nominals. For example, the following two fragments show examples(1) of long compounds:

> "Investigation revealed that the port main gear door rear hook operating spring strut plunger had fractured."
>
> "U.S. naval navigation equipment acquisition cost estimates"

1 These are naturally occuring examples given to me by Bill Woods.

Although these examples may sound strained in the absence of any context, they apparently did not to the authors (and, we hope, to the readers of the documents from which they were extracted).

The second feature of my definition is that the word _noun_ does not appear in it. The analysis which I offer in this thesis is appropriate for words referring to concrete objects (i.e. concrete nouns), but also to words which refer to events, states and situations (i.e. typically referred to with verb-derived words) and even to some adjectives (i.e. those which have a non-predicating meaning). The criterion which ties these together is that all refer to nominal concepts. For example, many words which have the surface form of an adjective can be used to refer to concepts that are underlyingly nominal in character. A _logical fallacy_ is not a _fallacy that is logical_ nor is a _criminal lawyer_ always a _lawyer who is criminal_. These phrases can be interpreted as compounds involving the nominal concept pairs _logic + fallacy_ and _crime + lawyer_.

The last feature I wish to point out is that I do not make any of several common discriminations. These include the distinctions between idiomatic and productive compounds, between lexicalized and novel compounds, and between compounds with stress placed on the first and second words.

1.1.2 _What Kind of Semantic Interpretation?_

The goal of semantic interpretation is to identify the concepts that are being referred to and to make explicit the underlying relationships between them. Although this work has been undertaken within the context of a natural language system which retrieves information from a database, I have chosen not to tie the semantic representation directly to the database.

My initial problem formulation and approach was in terms of a relational database model, i.e. it dealt with relations, roles, attributes and values existing in a particular database. There were several reasons why I decided that this was a poor strategy. The formalism for semantic representation that has ultimately been chosen is more general and expressive than the relational database model. It provides a way to describe intentional knowledge about the world as well as extensional knowledge. The demands of interacting with other language users forces us to be able to deal with multiple views and representations, many of which will not match a given database. To sum up these points, we could say that it is important to represent the things that the database describes, not the things in the database. A more complete discussion of the representation requirements of a natural language database retrieval system can be found in [TENNAN79] and [FININ79].

1.1.3 What is the Problem?

Let's restrict our attention for a moment to a much simpler class of compounds - those made up of just two words, both of which are nouns that unambiguously refer to objects that we know and understand. What is the fundamental problem in interpreting the modification of the second noun by the first? The problem is to find the underlying relationship that the utterer intends to hold between the two concepts that the nouns denote. For example, in the compound "aircraft engine" the relationship is part of, in "meeting room" it is location, in "salt water" it is dissolved in. There are several aspects to this problem which make it difficult.

First, we see that the relationship is not always evident in the surface form of the compound. What is it about the compound GM cars which suggests the relationship made by? This interpretation depends on our knowledge of several facts. We must know that GM is the name of an organization that manufactures things, and in particular, automobiles. Another fact that helps, although it is not necessary, is that the identity of a car's manufacturing organization is a salient property.

A second important aspect is the general lack of syntactic clues to guide the interpretation process. The interpretation of clauses involves discovering and making explicit the relationships between the verb and its "arguments", e.g. the subject, direct object, tense marker, aspect, etc. Clauses have well developed systems of syntactic clues and markers to guide interpretation. These include word order (e.g. the agent is usually expressed as the subject, which comes before an active verb), prepositions which suggest case roles, and morphemic markers. None of these clues exists in the case of nominal compounds.

Third, even when the constituents are unambiguous, the result of compounding them may be multiply ambiguous. For example, a woman doctor may be a doctor who is a woman or a doctor whose patients are women. Similarly, Chicago flights may be those which are going to Chicago, those coming from Chicago or even those which make a stop in Chicago.

A forth aspect is that compounds exhibit a variable degree of lexicalization and idiomaticity. In general, the same compound form is used for lexical items (e.g. duck soup, hanger queen) and completely productive expression (e.g. faculty meeting).

Finally, I point out that it is possible for any two nouns to be combined as a compound and be meaningful in some context. In fact, there can be arbitrarily many possible relationships between the nouns, each appropriate for a particular context. (2)

1.1.4 Why Study it?

Real natural language processing systems need to be able to understand nominal compounds because people use them. Nominal compounds serve a useful function in our language. Compounds are based on the process of nominalization which is one way that a sentence can be packaged and used as a unit. One of their important functions is naming. The compounding form is often used to refer to know concepts.

The interpretation of nominal compounds brings out many issues that are general to the interpretation of any linguistic forms. It provides a good vehicle for the study of such problems as how to represent and handle both syntactic and semantic ambiguity, the effects of pragmatic knowledge and the relation of discourse context to the interpretation.

Finally, the problem of interpreting nominal compounds has received little attention to date, at least from the perspective of computational linguistics. Most of the effort has been directed toward the interpretation of the clause and the

2 This makes a good game. Player A picks two nouns and player B tries to invent a context in which the two nouns make sense as a compound. This is reminiscent of the game invented by MIT linguists in which one tried to invent a context forcing the violation of any proposed selectional restriction.

semantics of adjectival modification. This has been the result, in part, of having better developed theories and paradigms for these areas. A consequence is that past and current natural language processing systems have analyzed nominal compounds in one of several unsatisfying ways. Typical approaches have been to match common compounds at the lexical level and replace them with unique lexical items, to treat them as adjectives, or to write special interpretation rules for each modifying and/or modified word. I know of no general facility which does an adequate job of interpreting nominal compounds in any existing system.

1.2 The Context of the Research

One of the goals of this research has been the design of a specialist whose domain is the interpretation of nominal compounds. This specialist will become a component of the natural language data base accessing system JETS [FININ79]. The JETS system is currently being designed at the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign and is an outgrowth of our earlier system, PLANES [WALTZ76].

1.2.1 The PLANES System

PLANES was developed to study the problem of natural language access to a large data base. The primary goal was to construct a system which would allow a non-programmer to obtain information from the data base by entering queries in a relatively unconstrained subset of English. Briefly, PLANES (1) received a request from the user; (2) parsed the request into an internal representation with a semantic grammar; (3) translated this representation into a formal query via a query generator; (4) executed the resulting query to retrieve the information; and (5) displayed this information in an appropriate manner.

The PLANES system accesses a large relational data base which contains information on Naval aircraft maintenance and flight records [NALDA][TENNAN78]. Maintenance records include such information as time and duration of the maintenance action, who performed it, what action was taken, and whether the service was scheduled or unscheduled. Aircraft flight data includes information such as the number of flights made by an aircraft for each day and month, the purpose of each flight, and the number and types of landings made for each flight. JETS is being designed to access the same data base.

Some simple kinds of questions that one could ask of the PLANES system are:

How many planes had engine damage in January 1976?

Which of them made more than 15 flights in December?

Show me the repair dates for the F4 with Buser number 30045.

List those F4's having more than 15 NOR hours that month.

More that 100?

During 1975, did plane 30045 require any electrical maintenance?

76?

Some typical compounds from this domain, all of them involving the head noun "damage", are:

engine damage	weather damage
January damage	corrosion damage
plane 3 damage	crash damage
acid damage	catapult-launch damage
night landing damage	bird strike damage

Three methods for handling compounds were used in the PLANES system. The first was to recognize many common compounds as words from the query were fed to the parser(3) These compounds were replaced by unique lexical items. The second technique was to have the grammar look for explicit words at the appropriate places. Since this was a semantic grammar it was possible to write special rules to handle modifiers for, say, the word "damage". The third and most general method was to write special sub-grammars for certain important semantic categories and then call these from the main grammar in appropriate places.

1.2.2 The Proposed JETS System

The major thrust of our new design is to increase the coverage of the natural language processing. There are two nearly independent components to the concept of coverage. The conceptual coverage of a natural language system refers to the set of

3 . The parser consisted of a semantic grammar expressed as an augmented transition network (ATN).

concepts that it can deal with. The linguistic coverage of a system refers to the linguistic knowledge that it has which enables it to understand variations in the way concepts are introduced, referenced, and described. A detailed discussion of coverage and the ways in which it can be measured can be found in [TENNAN79].

This research on nominal compounding addresses both components of coverage. Extending the linguistic coverage is the direct goal of this work. The semantic representation that JETS builds should not be highly sensitive to stylistic variations in the way a concept is described. For example, we want to build similar representations for the following phrases:

engine housing acid damage
acid damage to engine housings
acid damage to the housings of engines
damage by acid to engine housings
damage resulting from the corrosion of engine housings by acid
corrosion on engine housings
~~engine housing acid corrosion damage~~

This requires that the semantic interpretation rules be able to discover or infer the concepts to which the words in the phrase refer and the underlying relationships between them.

The indirect result of this work is the extension of the conceptual coverage of JETS. To compute similar semantic representations for the phrases in the above example, the concepts for engine, housing, acid, corrosion and damage must be represented in such a way as to enable and facilitate the discovery of the relationships between these words and words which they modify or are modified by. For example, the concept of damage should include the information that damage can be the result of another event (e.g. corrosion) and that the object that was damaged is important. We must represent the fact that acid can cause corrosion and that the corrosion event can lead to a state in which something is damaged.

1.2.3 Achieving Semantic Closure

A major part of our work on the JETS system is centered on achieving a high degree of closure. By closure, we mean the ability to handle user utterances which are consistent with the conceptual domain of the system, but which have not been foreseen. Woods [WOODS77] defines the concept of closure for natural language processors as:

"The difficulty in natural language understanding is not so much being able to formulate rules for handling phenomena exhibited in a particular dialog, but to do so in such a way that closure is eventually obtained -- i.e., subsequent instances of the same or similar phenomena will not require additional or different rules, but will be handled automatically by generalized rules. The formulation of such rules requires a good formalism for expressing general rules and a methodology for obtaining the correct degree of interdependence among individual rules. It also requires a good linguistic intuition and/or knowledge of linguistic results for determining the correct generalizations of the phenomena."

One of the goals of this work has been to achieve a high degree of closure in the semantic interpretation of nominal compounds. The central feature of the modification involved in nominal compounds is that the semantic relationship which exists between the two nouns is not explicit in the utterance. Moreover, a large number of relationships may, in principle, be possible between the two concepts represented by the nouns. It is the responsibility of the system to attempt to infer or discover an appropriate relationship, given its understanding of the two concepts involved, general pragmatic knowledge, and the current discourse context.

An example: Time

As an example, consider the use of a time phrase used to modify a noun, as in the phrases "January Skyhawks repairs" and "1976 flights". If the system can interpret phrases referring to time (as almost any system must) then it should attempt to interpret the modification of any other concept which could conceivably have a time phrase attached to it.

In the semantics we are developing for JETS, a time phrase can only be used to modify a concept which is, or can be viewed as, a kind of an EVENT. A minimal amount of closure is achieved when any event or event related concept can be successfully modified by a TIME concept. What if the modified concept is not an EVENT but something else, say an OBJECT? If a time phrase is hypothesized to modify something which is a kind of OBJECT, we want the system to attempt to derive an underlying event associated with that object to attach the time phrase to. For example, in the standard PARTS-SUPPLIERS-PROJECTS (4) domain, the phrase "January

4 This domain is often used to describe the operations of data base query systems. Codd's RENDEZVOUS system is one natural language system which uses this domain. In its simplest form, it contains information on parts (e.g. part numbers and names), projects (e.g. name, location, inventory of parts), suppliers (e.g. name, location, rating), and shipments (e.g. from which supplier, to what project, part number, quantity).

parts" might suggest the interpretations:

> parts which were shipped in January
> parts which were received in January
> parts which were ordered in January

In such an impoverished domain this is almost trivial, as one can precompute the set of events in which a concept can partake.

In a semantically rich domain, such as our 3-M data base [NALDA], the problem is much more difficult. One can not (or perhaps should not) always enumerate the potential relationships which might exist between even two simple concepts. The ability to handle references to entities and relations mentioned earlier in the discourse makes the problem even more complex. This allows for more potential relationships between any two concepts. For example, the phrase "the January planes" could be used to refer to a set of planes introduced previously in the discourse. The successful interpretation of this phrase would require a search through the recent discourse to discover a set of planes which was involved in an event which occurred in January. For example, the context might have been the one shown in figure 1.1. In this case, "the January planes" should be interpreted as referring to "the planes which received engine maintenance in January 1978".

1.3 The Approach to the Problem

This work can be characterized by three themes. The first is that a powerful, expressive representational system should be the foundation on which everything else is built. The second involves the rule based approach to semantic interpretation. The third theme is the importance of having a uniform knowledge base accessible to all components of the overall language processing system.

1.3.1 The Representational System

The choice of a representational system has been very important to this research. The system is built on an object based language for representing concepts, objects and relationships. It is also used to encode meta-knowledge - e.g. knowledge about its own representations.

The most interesting and powerful of the semantic interpretation rules for nominal compounds work by examining the representation of the constituent concepts.

```
<user>  Show me the engine maintenance performed on F4's in
        the last three months.

<JETS>  DATE          PLANE  MAINTENANCE CODE

        1/2/78          3       ...
        1/10/78         3       ...
        1/12/78         23      ...
        1/20/78         23      ...
        1/26/78         3       ...
        1/28/78         48      ...
        2/6/78          4       ...
        2/10/78         32      ...
        2/10/78         23      ...
        .               .       .
        .               .       .
        .               .       .

<user>  Which of the January planes also required maintenance
        in December?
```

A Discourse Context

figure 1.1

This means that much of the knowledge needed to interpret nominal compounds can be directly encoded in the nominal concepts themselves.

The power of the representation system does not come from its style - which is a frame or object based system - but from its ability to express intentional as well as extensional knowledge, its ability to treat its own representations as formal objects which can be reasoned about, and its ability to capture generalizations through an abstraction hierarchy.

Chapter Four presents the representational system in detail. Chapter Five describes a closely related topic - the matcher.

1.3.2 Rule Based Semantic Interpretation

The model of interpretation that has been chosen for this work is one in which knowledge is captured in interpretation rules. Each rule has two main components - a "left hand side" which describes the situations in which it applies and a "right hand side" which specifies the changes that take place when the rule is applied. An important component of the system is the rule applying executive. Its duties are to find rules appropriate to the current situation, choose which ones to try, apply them and keep track of the results.

The problems of rule contention (5) have been handled by using a heuristic which attempts to find rules in order of their "appropriateness". That is, those rules which are deemed to best match or describe the current situation are tried first. The executive continues to try rules until they have all been exhausted or one reports having a high degree of success. In either case, a local decision is then made to select the best outcome. The alternatives, both tried and untried, are saved in a way that allows them tc be explored at a later date if the local decision turns out to have been a poor one. The present system, however, does not make use of this information, i.e. it will never backup.

5 i.e.what to do when more than one rule applies to a given situation.

1.3.3 A Uniform Knowledge Base

The third theme of this research is the use of a central uniform knowledge base. The knowledge base is central in that all parts of the system (and any language processing system that it might be embedded in) have equal access to it. It is uniform in that the same representational system is used to express all of the knowledge. A set of conventions has been developed which will allow a high degree of sharing of knowledge between different components of the overall language processing system.

The knowledge necessary to interpret nominal compounds is typically not special to the task and could be utilized by other components. This is important, since the task requires a thorough understanding of the concepts involved in the compounds.

1.4 A Note to the Reader

The rest of this thesis is divided into seven chapters. This section outlines their contents and gives some suggestions for the reader who is short on time.

Chapter Two discusses nominal compounds from a number of different viewpoints. A variety of facts about them are presented and examples of many of the common forms given.

Chapter Three discusses related work by linguists and computer scientists. This chapter might be skipped by readers who are not interested in the background of this topic.

The fourth chapter describes the representational system used in this work. The reader might skip this chapter if he is familiar with similar systems and finds that he can make sense of the examples in the chapters which follow.

ChapterFive presents the pattern matching system on which much of the work is based. The first two sections of this chapter provide a general discussion and the ~~third the details. The impatient reader might want to postpone exploring the~~ details which are brought out in the third section.

The sixth and seventh chapters are the core of the thesis. The sixth details the method used for modifier parsing - discovering the internal structure of compounds involving three or more concepts. The seventh describes the conceptual modification process in which semantic relationships between concepts are proposed and weighed. These two chapters should not be skipped.

The eighth and final chapter summarizes the goals, what has and has not been accomplished and makes some suggestions for future work.

Appendix A lists a taxonomy for nominal compounds that was proposed by Lees [LEES68]. Appendix B is an outline of the abstraction hierarchy of the concepts that have been used in exercising the implemented system. Appendix C shows some examples of the matcher in operation. Appendix D shows some examples of the entire interpretation system in operation. It might help the reader to glance at this last appendix to get a general idea of the system's operation.

2 NOMINAL COMPOUNDS

This chapter introduces the topic of the thesis, nominal compounds, and discusses various facts about them.

The first section describes nominal compounds from four different viewpoints. The syntactic, Semantic, pragmatic and idiomatic aspects are explored.

The second section presents a taxonomy of some of the nominal compound forms. This taxonomy is more than just a classification of the data, for it provides a basis for a theory of the semantic interpretation of noun-noun modification.

The third section brings up the problem introduced by the recursive nature of nominal compounding. The result of one noun modifying another has all the features of a nominal, i.e. it can occur anywhere a simple noun can occur. Thus one finds arbitrarily long strings of nouns related through modification.

2.1 Aspects of Nominal Compounds

2.1.1 Syntactic Considerations

Syntactically there is not much to say about nominal compounding, unless one's goal is the development of a grammatical system in which nominal compounds can be derived from underlying sentential forms. This section presents a mixed bag of facts which are, if not syntactic, at least not semantic or pragmatic.

We can note, for example, that the modifying noun is typically singular. One goes _duck hunting_ and not _ducks hunting_ even though one really hopes to shoot many ducks.

In spoken language, the stress pattern placed on a compound can effect its interpretation. The primary-secondary stress pattern is very common with compounds that fit the traditional definition of a "compound noun", e.g. in _hot dog_, and _atom bomb_. The tertiary-primary stress pattern is common with what has been referred to as "nominal phrases", e.g. _apple pie_ and _faculty meeting_. This fact can be used to discriminate between two possible interpretations of the same compound, as in "woman´ doctor" which is a doctor whose patients are women and "woman doctor´" which is a doctor who is a woman.

as in "woman' doctor" which is a doctor whose patients are women and "woman doctor'" which is a doctor who is a woman.

In written forms compounds which fit the "compound-noun" description are often hyphenated, as in court-martial and hot-dog. Others have become more lexicalized and are written as one word, e.g. bookcase and tablecloth.

2.1.2 Semantic Considerations

A nominal compound is composed of two nouns with an implied semantic relationship existing between them. There are several things we can say, in general, about the relationship. First, it often is supplied by a verb from which one of the nouns is derived. In other cases, it comes from our knowledge of the concepts that the nouns denote, e.g. in what relationships they characteristically partake.

Modifying one noun with another usually does not change its basic nature (endocentric modification), particularly in compounds that are not thought of as "compound nouns". Exocentric modification, in which the nature of the head noun is changed is a good test for "compound-nounness", e.g. a hot-dog is not a kind of four-legged domesticated animal commonly kept as a house pet.

The semantic relationships between the nouns represent characteristic or habitual relations rather than accidental, temporary, or fortuitous ones. For example, if you are a man and once happened to collect some garbage, you are not a garbage man. A winter coat is not a coat that was given to you during one winter.

Similarly, the relationship between the nouns should not be totally predictable. A house guest is not a guest who happens to live in a house. Although we say tomato sauce when we mean sauce made from tomatoes, we do not say milk butter to mean butter made from milk because this is a totally predictable relation. If such a compound were uttered, we would try to assign it some other interpretation.

Another apparent constraint is that the relation should not be a negative one. We would not entertain that mouse food might refer to food that a mouse did or will NOT eat.

All of these constraints can have exceptions, however. They can also be explained to some degree by higher linguistic functions, such as conversational postulates. A good discussion of the source of some of these constraints and the situations when they don't apply can be found in [DOWNING].

2.1.3 Pragmatic Considerations

What is the function of nominal compounding in English or in any other language? Two functions come to mind immediately: compression and naming.

A concept expressed as a nominal compound is a concept expressed compactly. The compression occurs in both syntactic and semantic domains. Consider having to paraphrase the nominal compound examples given on page 2 without compounding! The second example might become:

> "Estimates of the cost for the Navy of the United States to acquire equipment for navigation"

This paraphrase makes heavy use of prepositional phrases, a device which is not far removed from nominal compounding. Consider having to use a full sentential paraphrase. This compacting function depends, of course, on our ability to reliably recover the intended meaning of a nominal compound.

Nominal compounding in English serves as a productive naming device. It is one process through which new entries may be made in the lexicon.(6) The naming function of nominal compounding can occur spontaneously and have a limited duration or it can occur gradually and enter the lexicon of entire dialects. For example, in a given context, one might use noun-noun modification to refer to an object as if the compound were a name (or unique description). Such compounds function as deictics, e.g. "pointing words".

2.1.4 Idiomatic Analysis

Finally, many compound nominals require an idiomatic analysis. These are typically exocentric and thought of as "compound-nouns". It is often clear how they were derived from their constituents. This is probably no accident, as it may aid the learning of the compound and its meaning or even facilitate the retrieval of the proper meaning.

6 It has been called "a back door to the lexicon" by Downing [DOWNING].

It is quite typical that a true "nominal phrase" to become, over a period of time, a compound-noun. One method for a compound to become lexicalized is for one of the constituent nouns to undergo semantic narrowing or broadening (e.g. brief case, grammar school, bath room).

It is also apparent that lexicality is a matter of degree. The meaning of compounds range from utterly opaque to completely transparent. Figure 2.1, adapted from [LEVI] shows such a scale.

2.2 Common Forms

This section presents many of the common forms that nominal compounds take. My goal is not to develop a complete taxonomy, but rather to familiarize the reader with some of the regularities that can be recognized. By and large I will be considering two word compounds in this section. The analysis, of course, does not depend on this fact. Appendix A lists the taxonomy developed by Lees [LEES68]. The interested reader may wish to compare the two.

2.2.1 Compounds Containing Nominalized Verbs

This class includes nominal compounds in which one or both of the constituents is a nominalized verb. Nominalized verbs can be broken down into differing types, e.g. those describing activities, processes, events, states, results, etc. All of these cases, however, have similar characteristics.

In these cases, one interpretation for the compound is that the verb from which the one noun is derived supplies the relationship for the other noun. Clauses have a fairly well defined structure which is determined by the governing verb. The verb can be viewed as having a set of associated case roles which determine its relationship to the nouns that fill them. These roles can be grammatical (e.g. subject, object, etc.) or more semantically oriented (e.g. agent, object, recipient, instrument, location, etc.). For example, we can have the following patterns:

agent + verb	cat scratch
object + verb	engine repair
instrument + verb	knife wound
vehicle + verb	boat ride
recipient + verb	Unicef donations
location + verb	ocean fishing
time + verb	summer rains
source + verb	Chicago flights
destination + verb	target shooting

productive forms

/\

	...examples...
transparent	mountain village, family reunion, lemon peel, ...
partly opaque	grammar school, brief case, polar bear, boy friend, ...
excocentric	birdbrain, razorback, ladyfinger, blockhead, ...
partly idiomatic	polka dot, monkey wrench, flea market, ...
completely idiomatic	honeymoon, fiddlesticks, duck soup, ...

\/

idiomatic forms

A scale of the productivity of compounds
(after Levi)

Opacity Scale

figure 2.1

cause + verb	drug killings

The list can be extended to cover the case roles of your favorite case role system. In general, the patterns can be reversed, with the modifier being some form of nominalized verb, as in:

verb + agent	repair man
verb + object	throwing knife
verb + instrument	cooking fork
verb + location	meeting room
verb + time	election year
verb + source	shipping depot

And, of course, there are cases in which both nouns are nominalized verbs, such as "maintenance contract".

2.2.2 Compounds Containing Role Nominals

~~Another common form is characterized by the presence of a special kind of~~ nominalized verb, what I call a role nominal. A role nominal is a word that refers to an underlying case role of a verb. A simple example is that driver refers to the agent role of a driving.

In the English language there are very productive ways to generate words which refer to the agent role of a verb. Almost any verb can be transformed into an agentive role nominal, typically by the addition of the -er affix (e.g. worker, driver, writer, employer, killer). The affix is sometimes -or as in inspector, actor and survivor. In some agentive nouns following this form, there is no English verb from which they are so derived, as in doctor and author. A less common form of agentive production is the use of the -ant suffix, as in inhabitant, participant, and informant. Many of these words can also be used to refer to the instrument role of the verb. Some examples of words which are ambiguous between the agent and instrument roles are cleaner, opener and holder. Others seem to only refer to the instrument role, specially those which have the -ant morpheme, as in lubricant and disinfectant.

The ambiguity between the agent and instrument roles is not a feature of this morphology but is rather indicative of a basic syntactic and semantic similarity between these two roles. For many verbs, English allows the instrument to appear in a clause wherever the agent may. For example consider the sentence pair:

John broke the window with a rock.
A rock broke the window.

What one does not, in general, find in the English language are productive ways to refer to roles other than the agent and possibly instrument of a verb. The only semi-productive form is use of the _-ee_ suffix to refer to the object or patient role of some verbs. Examples are:

payee	lessee
employee	draftee
trainee	nominee
appointee	

Note that in all cases the result is a personal noun. A good discussion of this issue for this type of morphological analysis is [QUIRK].

These facts are indicative of the surface structure of our language and not the underlying semantics. It can be very fruitful to analyze many words as denoting concepts which refer to roles of other concepts. This is quite common with events, as the following examples show:

noun	role nominal interpretation
Feed	the object of an eating, typically by animals, typically domestic.
Pump	an instrument of a pumping.
Pipe	the conduit of a flowing, typically by water or gas.
Gun	the instrument of a shooting.

The _role nominal_ is a useful generalization, especially when trying to interpret nominal compounds. When a nominal compound contains a word that can be interpreted as a role nominal, then the underlying relation can be suggested by the verb to which the role nominal refers. A _truck driver_ is a person who (characteristically) drives a truck. A similar analysis is available for _LA drivers_ (location) and _night drivers_ (time). Some other examples are:

cat food	The food that a cat _eats_.
elephant gun	A gun that is used to _shoot_ elephants.
oil pump	A pump that is used to _pump_ oil.
dog house	A house that a dog _dwells_ in.

Another way to view the role nominal is that it serves to tie an object to a characteristic activity in which it participates. It is very much like a relative clause except that the characteristic or habitual nature of the relationship is emphasized.

2.2.3 Attribute transfers

This class of nominal compounds covers cases where an attribute of the modifying noun is transferred to the modified noun. Consider the following examples:

iron will
crescent wrench
circle opening
elephant legs

The analysis is that an attribute or property of the modifier is transferred to the modifier noun. An iron will is a will that is strong. Elephant legs are probably legs that are large. An elephant memory, on the other hand, would be a memory with a high degree of recall. Note that the attribute selected for transfer depends on both of the nouns. It is also possible for the transferred property to undergo a shift in meaning to make it more appropriate in the domain of the modifier.

It is interesting to speculate on the typical sources for such expressions. The world of mythology and personified animal stories is a rich one. Consider the following business names and what properties they suggest transferring from the animal to the business:

Greyhound Bus
Dove soap
Atlas Tire
Fox Foto

These share properties with similies and metaphorical expressions. Methods for selecting and transferring attributes are suggested in [ORTONY] and [WINSTON].

2.3 Recursive Compounds

In this section I want to discuss several aspects of the recursiveness of nominal compounds. Since a nominal compound functions as a noun the process is inherently recursive. This can and does lead to compounds which are composed of arbitrarily many nouns. Recall the example I gave earlier which exhibits a compound with ten nouns:

> "Investigation revealed that the port main gear door rear hook operating spring strut plunger had fractured."

This is one source of the ambiguity associated with nominal compounds (the other being the ambiguity due to the unspecified relationship). It is also the source of many jokes, especially those favored by George Burns and and Gracie Allen.(7)

A general model of modification is that it can be represented as a binary tree. For example, if we have a compound consisting of three nouns, A, B and C, then either:

[1] A modifies the result of B modifying C
(A (B C))

[2] The result of A modifying B in turn modifies C
((A B) C)

This leads to a large number of possibilities if the number of nouns in the string is large. In fact, the number of possible parses grows exponentially with the number of nouns, following the Catalan sequence.(8) Worse yet, we might want to consider an even more general model in which A and B can modify C independently, as in:

[3] A and B independently modify C
(A B C)

This leads to still more possibilities. There are several factors, however, which ameliorate this problem.

First, there is a strong preference, in English at least, for compounds which have structure [2] over structure [1]. There is also, I believe, a preference for structure [3] over either [1] or [2]. This seems to be particularly true in the longer sequences. See section 3.2.4 for more details.

Another factor is that endocentric compounds are far more common than exocentric. An endocentric compound is one in which the modifier does not change

---

7 An example: interpreting a "silver serving tray" as "a tray for serving silver".

8 See section 3.2.4 on page 43 for further discussion.

the basic nature of the modified noun. For example, a house dog is still a kind of dog. In an exocentric compound, the basic character of the modified noun is altered by the modifier. These compounds are often lexicalized, as in arrow head. One can take advantage of this property in exploring the possibilities by assuming that all compounds are endocentric unless they are completely lexicalized. This has the effect of reducing the number of possibilities that need to be considered. For example, in a three word compound we need to evaluate the "goodness" of five instances of modification: (A B), (B C), (A (B C)), ((A B) C) and (A B C). If we assume endocentricity, we need only check three: (A B), (B C) and (A C). The goodness of the others can be estimated by developing a calculus of modification goodness measures that allows one to combine them.

Finally, in many cases it may not be necessary to determine the structure or the exact nature of the relationships between the nouns at all. Often, a compound is used as a means to refer to an object or concept that both the speaker and hearer know. In some cases the compound overspecifies and the hearer can find the referent without a complete analysis. In others, the hearer may know that there is no possible referent in his memory after a partial analysis (even after identifying the head noun and verifying that there is no exocentric modification).

3 RELATED WORK

This section describes previous work which relates to the problem of interpreting the meaning of nominal compounds. There are two primary sources of related research: the fields of Linguistics and Computational Linguistics/Artificial Intelligence. Many linguists have been drawn to the problem of the formation of nominal compounds. Primarily, their studies have been concerned with the problem of discovering the constraints our language places on the formation of compounds rather then the problem of interpreting their meaning. AI research, on the other hand, is primarily interested in computing a meaning for an instance of modification, be it nominal or other. Relevant AI research can be found in work on natural language understanding and question answering systems, of course. The work in the more general area of knowledge understanding and representation is also relevant, as the interpretation of natural language in general and of nominal compounds in particular, requires a good knowledge representation system.

3.1 The Linguists

Linguists have been interested in nominal compounds for a long time. The traditional approaches have addressed a number of issues and take a variety of approaches. Traditional approaches, such as those by Lees [LEES60] and the Gleitmans [GLEITMAN], have attempted to develop taxonomies (based on the syntactic and/or semantic nature of the compounds), to develop systems which will derive compounds from underlying sentential forms, and to try to express the constraints on compound formation. More recent approaches, such as those by Levi, Reimold and Downing, have turned more toward attempts to understand nominal compounding from a functional and process-oriented point of view.

This section will discuss, or at least mention, much of this work.

3.1.1 Lees

Lees is the author of a monumental attempt on the nominal compound from the traditional transformational grammarian's point of view [LEES60]. His task was to

3.1.1 Lees

Lees is the author of a monumental attempt on the nominal compound from the traditional transformational grammarian's point of view [LEES60]. His task was to describe a system for the derivation of noun compounds from underlying syntactic structures. Thus the relations between nouns of compounds are grammatical, in his analysis. One result of his study, his taxonomy for nominal compounds, can be found in appendix A.

Ten years after his original work, Lees took a different approach to the same problem. In [LEES70] he suggests an analysis which is based on a case frame model. The case roles, rather than the grammatical relations, providing the underlying structure. In addition, he proposes that there is a limited set of "generalized verbs" that can appear in nominal compounds. He suggests, for example, one generalized verb to cover "impel, propel, energize, activate, power, drive, actuate, etc.".

3.1.2 Gleitman and Gleitman

Lila and Henry Gleitman undertook a psycholinguistic investigation of English speaker's performance in paraphrases of three word compounds. Their investigation [GLEITMAN] was based on the following experiment.

Three word compounds were formed by taking two constant nouns, "house" and "bird", and one variable word chosen from a set of twelve monosylabic words, "boot, foot, black, bright, thin, eat, kill, wash, glass, stone, dry, shirt". All possible permutations of each combination of three words were computed, for a total of 144 three word phrases. Subjects were then presented with the entire corpus of phrases and asked to give paraphrases for each.

The paraphrases were then evaluated as correct or incorrect, depending on the appropriateness of the compound. For example, given the phrase "house bird foot", an appropriate paraphrase would be "the foot of your pet bird". An incorrect paraphrase might be "a house for birds shaped like a foot".

They also attempted to characterize the erroroneous paraphrases by classifying them into one of four categories: errors of order (the paraphrase was appropriate for a different order), stress (the paraphrase was appropriate for a different stress pattern), chaos (more than one change in stress and/or order were made), or format (the subject failed to follow the order to provide a nominal paraphrase).

The results of their study were that the ability to suggest appropriate (i.e. correct) paraphrases for these novel phrases correlated with increased educational background.

3.1.3 Reimold

In his Ph.D. thesis, Reimold [REIMOLD] presents a formal theory of sentence comprehension that is based on "perceptual strategies". As such, Reimold develops a theory of linguistic performance rather than competence. His emphasis is on the computation of the semantic information rather than the discovery of the syntactic structure.

Reimold considers the interpretation of nominal compounds to be a significant test of his approach in that they contain no syntactic information to guide the processing. Consequently he discusses some strategies and rules for the interpretation of compound nouns.

Reimold bases his interpretation rules on a taxonomy of seven different types of compounds. His types, and examples drawn from his thesis, are:

[1] "Identity with an explicit argument" (e.g. carpet sale, truck driver, wife swapping, bull fight)

[2] "Identical variable" (e.g. oak tree, glass table pet snake)

[3] "During" (e.g. day dream, October election, winter coat)

[4] "Part of" (e.g. summer day, coat button, arm chair, table leg)

[5] "Normal use" (e.g. oil tank, wash basin, cat comb)

[6] "Location" (e.g. bed room, salt water, school teacher)

[7] "Resemble" (e.g. mushroom lamp, butterfly hat)

The first type is the most productive. Reimold represents many nouns (especially those referring to events) as n-argument relations. For example, the noun SALE is related to the SELL relation which has six arguments: agent, object, recipient, payment, time and place.

His postulated interpretation process is composed of three steps. First, the underlying representation of the two nouns are retrieved from the lexicon and concatenated to form a Preliminary Semantic Representation (PSR). Secondly, the PSR is transformed into an Intermediate Semantic Representation by adding certain implications. These implications are the result of the application of certain meaning- and encyclopedic-rules. Finally, the ISR is transformed in to the FSR, the Final Semantic Interpretation by specifying the semantic relations between the constituent nouns and some or all of the implied terms. These semantic links are postulated by seven rules, one for each of the seven types in Reimold's taxonomy. Links are tested for semantic anomalies by a semantic marker-like system which describes the nouns and their constituents. Any links which survive the tests give rise to a FSR.

Reimold does not address the problem of selecting the most appropriate interpretation when more than one is possible. His system will, in general, produce many FSR's for most compounds if it is based on a good representation. Moreover, he provides no way to even rank the candidate hypotheses, clearly a desirable and important part of a performance model of semantic interpretation.

With respect to his taxonomy of compound types, I find that the first could be extended to cover many of the rest. For example, if one considers that _location_ is an attribute (or argument) which any concrete noun or event nominal can have, then his sixth rule is just a special case of the first. One can treat time in a similar manner, eliminating the third rule. This does require the ability to infer underlying event-type concepts from some concrete nominals, however.(9)

3.1.4 _Levi_

Judith Levi [LEVI] has made an extensive exploration of one set of prenominal modifiers. She defines the notion of a Complex Nominal (CN) which includes three kinds of expressions, all of which are fundamentally instances of noun-noun modification.

The first set, _nominal compounds_, covers the classical noun-noun modification

9 For example, to interpret "winter coat" one views "coat" as the object of a "to-clothe" event which is modified by the temporal concept "winter".

form. Nominal compounds include instances in which the prenominal modifier is also a noun. Examples are:

apple cake
brush fire
dog house
piston ring
wing tip

The second set, nominalizations, include pairs in which the modifying word is a noun and the modified head noun is derived from an underlying verb. Example from this set are:

city planner
data encoding
signal detection
engine damage
maintenance reports

The third set defined by Levi as "NP's with non-predicating adjectives". This set includes phrases in which the head noun is modified by a word having the surface form of an adjective, but an underlying derivation from a noun. Examples are:

rural route
electrical engineer
musical clock
constitutional amendment

Levi gives many detailed arguments to support the position of viewing these "pseudo-adjectives" as noun-like terms. Because I believe this analysis to be valid and relevant to my own work, I will sketch some of the principal arguments here.

Non-predicating adjectives (NPAs) do not normally appear in the predicate, or post-copula, position where a bona fide adjective can be used. To see this, consider the potential paraphrases listed below.

a chemical engineer	an engineer who is chemical
an atomic bomb	a bomb which is atomic
a linguistic argument	an argument which is linguistic

More appropriate paraphrases might be "an engineer whose specialty is chemistry", "a bomb in which the explosive energy comes from an atom", and "an argument about Linguistics". There are some NPAs which can also serve as predicating adjectives, but these are not synonymous when used as such, as the following phrase pairs show:

a criminal lawyer	a lawyer who is criminal
a logical fallacy	a fallacy which is logical

dramatic criticism	criticism which is dramatic

Similarly, we can distinguish between predicating and non-predicating adjectives by their ability to be modified by "degree adverbials" such as very, quite, and slightly. For example, we do not say "a very atomic bomb" or "a very electrical conductor". For NPAs which have a predicating sense, the use of a degree adverbial forces the selection of the predicating reading. Thus one might refer to R. M. Nixon as a "very criminal lawyer".

There is also evidence which supports the claim that NPAs are derived from ancestor nouns. Briefly these are:

(1) NPAs can be conjoined with nouns.

Generally, English only allows like constituents to be conjoined. We are able to conjoin a noun and NPA, as in "solar and gas heating systems" and "domestic and farm animals".

One can assign such semantic features as DEFINITE, ANIMATE and GENDER to NPAs as well as to nominals. For example, presidential and feline would be +ANIMATE whereas electric and automotive would be -ANIMATE.

(3) NPAs are amenable to a case-frame analysis.

Unlike bona fide adjectives, NPAs are readily analyzed in terms of case relations, particularly when the modified noun is a nominalized verb. For example, presidential can be seen as filling the agentive case in "presidential veto" and lunar as filling the objective case in "lunar explorations".

(4) NPAs may not be nominalized.

Unlike adjectives and like nouns, NPAs may not be nominalized. This is easiest to see when one considers adjectives which have both a predicating and a non-predicating reading. When an adjective is being used with its predicating sense it can be nominalized but not when it is being used with its non-predicating sense. Consider the word "mechanical" which has a predicating sense in "mechanical reaction" and a non-predicating sense in "mechanical engineer". When the predicating sense of this word is used it can be nominalized as in "the mechanicalness of the reaction". The non-predicating sense can not undergo nominalization, ruling out a phrase like "the mechanicalness of the engineer". Similar examples are listed below.

predicating sense	non-predicating sense
mechanical reaction The mechanicalness of her reaction	mechanical engineer *The mechanicalness of the engineer
nervous reaction the nervousness of the reaction	nervous disorder *the nervousness of the disorder
marginal contribution the marginality of the contribution	marginal width *the marginality of the width

Levi's analysis

A brief summary of her analysis is as follows. Complex nominals are derived from an underlying structure which consists of a head noun modified by a sentential construction. This construction can be either a relative clause or a NP complement. All CNs are derived through the application of one of two transformations: the deletion of the predicate in the modifying sentence, or the nominalization of the predicate in the modifying sentence.

CNs derived from Predicate Deletion

The predicate deletion transformation may only apply to a proposition whose predicate is a member of a small set of Recoverably Deletable Predicates (RDP). This set consists of the following eight predicates:

CAUSE	HAVE
BE	USE
FOR	IN
ABOUT	FROM

Figure 3.1 gives some examples, many from Levi, which exhibit the possibilities. Note that for three of the RDPs (CAUSE, HAVE and MAKE), there are two examples, corresponding to the active and passive forms of the deleted predicate.

Since the predicate relating the head noun and its modifier is deleted by this transformation, the meaning of the resulting CN is multiply ambiguous. Each of these predicates are good candidates for the relation between the two nominals. The ambiguity is constrained, however, by the smallness of the set of potential predicates, the RDP. In practice, this ambiguity is further reduced by semantic and pragmatic knowledge as well as the discourse context.

CNs derived from Nominalization

The second class of CNs discussed by Levi are those derived from the nominalization of the predicate in an underlying proposition modifying the head noun. Some examples from this class are:

faculty meeting
tree traversal
urban studies
fire detection
mathematics teacher

RDP	phrase	paraphrase
CAUSE	tear gas	gas which causes tears
	drug deaths	deaths which are caused by drugs
HAVE	gun boat	boat which has guns
	box top	top that a box has
MAKE	musical clock	clock which makes music
	program errors	errors made by a program
USE	steam iron	iron which uses steam
BE	soldier ant	ant which is a soldier
IN	city bus	bus which is in a city
FOR	animal doctor	doctor who is for animals
FROM	city visitors	visitors who are from a city
ABOUT	physics book	book which is about physics

Levi's Recoverably Deletable Predicates

figure 3.1

In these cases, the head noun is derived from an underlying verb (i.e. meet, traverse, study, detect, amplify and teach). The modifying noun can stand in one of two relationships to this underlying verb. In <u>subjective nominalization</u>, the modifying noun is derived from the subject of the verb in the underlying form. Examples are:

University purchases
acid corrosion
bird damage

<u>Objective nominalization</u> occurs when the modifying noun is derived from the object relation of the verb in the underlying sentence. Examples from this case are:

automobile purchases
engine corrosion
propeller damage

A third case, which Levi analyzes as a sub-case of objective nominalizations, is one in which the head noun denotes the agent of an action rather than the action itself. Examples are:

science teacher
coal miner
electrical conductor

In these cases, the modifying noun or non-predicating adjective is again the object of the verb in the underlying form. Levi's work is very impressive and covers much of the groundwork concerning noun-noun modification. It is also a wonderfully rich source of examples of such modification. The focus of her work, however, is quite different from that proposed here. The nature of this difference is a common one when comparing work done by Linguists and AI researchers. This section briefly discusses some of the major points at which her work and my own diverge.

Levi's analysis is primarily Syntactic. The primary goal of the study was the elaboration of detailed derivations for complex nominals within the framework of generative semantics. Generative semantics, the name notwithstanding, is heavily concerned with the form and structure of language and the structural transformations which operate on sentences.

Levi's semantic analysis is too shallow. Her set of eight Recoverably

Deletable Predicates are extremely vague. It is my feeling that such vague predicates as HAVE, FOR and IN should not be the stopping point of the semantic analysis. Most of the difficult but interesting work is in specifying exactly what these predicates mean for a particular case of noun noun modification. The following quote from Levi suggests that she would acknowledge this view:

"In the course of this exploration, it has become clear that a complete description of the role of complex nominals in natural language must include not only the kinds of syntactic and semantic facts that formal derivations can account for, but also a description of the broader semantic and pragmatic principles that influence the ways in which both speaker and hearers manipulate the formal regularities in actual discourse. Although this latter aspect of the grammar of CNs lies outside the scope of this study, its indisputable relevance to "the larger picture" as well as its intrinsic interest suggests that a brief discussion of the major issues may be appropriately included here."

Levi does not treat the use of complex nominals in definite descriptions. When one is making an anaphoric reference it is common to find complex nominals being used to "telescope" the description of the referent. An example described earlier in this paper showed how the phrase "the January planes" could refer to a set of planes which had engine maintenance in january. Levi's approach can not possibly help us for such cases.

Ambiguity is ignored. Levi is content to reduce the ambiguity to a small set of cases. The problem of using deeper semantic and pragmatic knowledge to resolve the ambiguity should, I believe, be a part of the analysis. Again, I feel that she would not quarrel with this. She says:

"What is not yet clear is how best to represent those kinds of knowledge that are not "strictly grammatical" in the common sense of the term (especially the basically ephemeral knowledge of "institutionalised" readings of CNs, and the highly context-sensitive variables which enter into our stylistic judgments), or how these different but related kinds of knowledge may best be integrated within a single description."

Processing strategies are ignored. Finally, her approach is more from the viewpoint of language production than language understanding. Little attention is directed to the problems of how people (or machines) might process complex nominals to extract an interpretation consistent with a larger context.

3.1.5 Downing

Pamela Downing has made a study of the creation and use of novel compound nouns in English (as opposed to lexicalized compounds). Her investigation consisted of four interesting experiments, they were reported in [DOWNING]:

"a Evaluation of the nature and relative frequency of the semantic relationships underlying attested but clearly non-lexicalized compounds.

b Naming task: subjects were asked to create names for drawings of entities with no conventionalized name.

c Context-free interpretation task: subjects were asked to provide interpretations for novel compounds in the absence of context.

d Ranking task: subjects were asked to evaluate the appropriateness of various interpretations proposed for a number of novel and lexicalized compounds."

The results of these experiments, according to Downing, suggest that the "constraints on N+N compounds in English cannot be characterized in terms of absolute limitations of the semantic or syntactic structures from which they are derived". Her view is that there are tendencies for compounds to be based on relationships which are permanent, non-predictable and ultimately based on the nature of the entity being denoted. The fact that some relationships are highly productive and very common indicates more about the process of categorization and description than it does about, say, derivational constraints on the compounding process.

Downing does identify, on the basis of her analysis of the novel compounds produced by the subjects in her experiments, an inventory of the most common underlying relationships. These are [DOWNING]:

relation	example (remember - these are novel)
whole-part	duck foot
half-whole	giraffe-cow
part-whole	pendulum clock
composition	stone furniture
comparison	pumpkin bus
time	summer dust
place	Eastern Oregon meal
source	vulture shit
product	honey glands
user	flea wheelbarrow
purpose	hedge hatchet
occupation	coffee man

3.2 The Artificial Intellegensia

The problem of interpreting nominal compounds has been studied by a number of people in Computer Science and Artificial Intelligence. In this section I will discuss some of this work.

3.2.1 Brachman

The problem of representing and understanding nominal compounds was used as an example domain by Ron Brachman [BRACHMAN] in his work on the Structured-Inheritance Network (SI-Net) knowledge representation formalism(10).

Brachman's SI-Net formalism is an associative network formalism for representing conceptual knowledge. After presenting his formalism and arguing for its power and adequacy, he discusses its application to two task domains. The first is the development of a document retrieval consultant whose knowledge base is an annotated bibliography. The second domain is the design of an intelligent on-line consultant for the Hermes message-processing system.

In discussing the language processing problems which would be faced by a document retrieval consultant, Brachman notes that annotated bibliographies make heavy use of the nominal compound construction to compactly describe the documents. This brings him to the problem of representing and understanding nominal compounds in a very broad semantic domain.

The primary motivation for his study of nominal compounding is to convince the reader that the SI-Net formalism provides a reasonable basis from which to attack the difficult problems of their representation and understanding. In fact, he wants to show that "A mechanism with the power like that of the Structured Inheritance Network notation is a necessity, rather than a luxury." [BRACHMAN].

This task domain gives Brachman a place to illustrate the use of some of the structures in his formalism and the general philosophy of how one uses it to represent concepts. It also shows "how the new network scheme holds up under the stress of some difficult representational problems" [BRACHMAN].

Brachman begins his discussion of nominal compounding by sketching the classic

10 In his more recent work this formalism is now called KL-ONE.

study of Lees [LEES60] on nominalization and nominal compounding. In doing so he quickly focuses on how one might represent nominalized verbs and propositions in SI-Net notation. He synthesized the analyses of Lees and Fraser [FRASER] and comes up with the following types of action nominals:

Agentive	The name for the agent of an action, as in <u>owner</u> or <u>lover</u>.
Factive	A reference to the fact that an event occurred, as in <u>that the plane crashed</u>.
Substantive	A reference to the result of an event, as in <u>application</u> in the sentence <u>he application was received last month</u>.
Activities	References to an event, as in <u>flying</u> or <u>maintenance</u>.

To represent these classes of nominals, Brachman introduces five new primitive links. The DFACTIVE link connects a factive nominal to its underlying event. Activity nominals are refined into two sub-categories, those referenced as ongoing activities and those referenced as completed actions. These are connected to their underlying events by the DACTIVITY/PROCESS and DACTIVITY/COMPL-ACTION links, respectively. A substantive nominal is connected to its underlying event with a DRESULT link. The agentive nominal is handled somewhat differently, being linked to an agent role of the appropriate event with a DROLE link.

The semantics of these primitive links is similar in each of the cases. Each link acts as a cable along which the roles of the event concept are inherited by the nominal concept. With a way to represent these nominalized verbs, Brachman can then discuss the representation of the various kinds of nominal compounds. These he divides into the following types:

object-verb	"news broadcasting"
verb-object	"drinking water"
subject-verb	"snake bite"
verb-prepositional obj.	"washing machine"
noun-plus-dattr	"arm chair"
dattr-dattr	"field mouse"

Brachman sketches an approach to the understanding of compounds in which one of the nouns is derived from a verb. Basically, it involves checking the roles of the verbal constituent for those which can accept the other constituent as a new value (by checking the value/restriction description). In addition, the candidate value must pass the structural conditions imposed by the verbal concept. He also suggests that one express preferences in the structural conditions, although a method for this is not discussed.

3.2.2 Rhyne

James Rhyne [RHYNE] has made a study of nominal compounding within a computational linguistic framework. In his work he developed a procedural model for generating nominal compounds from a noun phrase represented in a case-frame formalism. His basic analysis of nominal compounds is that their interpretation and generation depends on the existance of a characteristic relationship between the modifying noun and a verb in a paraphrase using a relative clause construction. Compounding is a process of systematically deleting information from an utterance just when the speaker expects the hearer to be able to reconstruct it.

Linguistic issues

Rhyne identifies three structural forms of nominal compounds in his work. The first (N-N) is one in which a noun is modified by another (surface) noun. The compounds computer terminal, telephone cord and aircraft engine are examples of this form. He briefly discusses the potential ambiguity which arises when there are two or more modifying nouns and notes that, in English, there is a slight preference for interpreting such phrases in a left-to-right manner. Thus, it is more common to find the (N-N)-N forms like:

typewriter repair man
electric typewriter repair man
engine damage report
jet engine damage report

than the N-(N-N) forms given below:

liquid roach poison
aluminum water pumps
January aircraft repairs

The second and third forms are N-participle-N and N-gerund-N, respectively. Rhyne discusses these forms only in passing.

Rhyne argues that nominal compounds in English are the result of one of two processes. The first involves the reduction of a relative clause followed by the preposing of of the remaining element. The second process is one in which the verb contained in the relative clause is nominalized and then preposed to modify the head noun.

Rhyne chose for an underlying representation a shallow case grammar rather than a deep case representation. This was motivated by his belief that the rules used to generate nominal compounds are primarily lexical. That is, their relevance and application strongly depend on the actual lexical items (e.g. words) which appear in the case frames. His case grammar is fairly typical, being similar to others developed at the University of Texas [SIMMONS]. It includes the following verb case roles:

PERFORMER
CAUSE
ENABLER
OBJECT
GOAL
SOURCE
LOCATION
MEANS

In addition, he uses two "structural" case roles: RELCLS and COMP. The RELCLS (relative clause) role is used to attach a relative clause to a noun. The COMP (compound) role is used to attach a modifying compound to a noun.

Constraints

Rhyne proposes three general constraints on potential rules for generating nominal compounds. The first is that nominal compounds are used to express characteristic or habitual relationships. A shrimp boat is a boat which is characteristically used to catch shrimp. The fact that the same boat was once used to catch sharks does not allow one to refer to it as a shark boat.

The second constraint involves the use of a proper noun as a noun modifier. Rhyne claims that this can only occur when the proper noun is the name of a process or a source, performer or goal of an act of giving.

The third constraint involves the degree to which terms in his rules match the lexical items in the structure being transformed. Rhyne's rules include class terms which could potentially match many instantiations. For example, one rule might involve the class <person>, which could match the lexical items man, woman, child, Indian or midget. He states that compounds are not generally formed when the lexical item is several levels below the class term used in the rule. Thus, a rule could be given which transformed "a <person> who repairs things" into "a repair <person>". This would generate the compound "repair man" from "a man who repairs things" but would not transform "a midget who repairs things" into "a repair midget".

<u>Rhyne's Computer Model</u>

Rhyne developed a simple computer model which transformed expressions in his shallow case grammar into surface nominal compounds. It consisted of a recursive rule interpreter and a collection of lexical transformation rules. As an example of one of his rules, consider a rule to map "a market which sells flowers" into the compound "flower market". It might be expressed as:

```
(market (RELCLS (sell +CHARACTERISTIC
                      (LOC market)
                      (OBJ flowers))))

==>

(market (COMP flowers))
```

We can write a more productive version of this rule by replacing the term "flowers" with the generalization <goods>. The rule would then be:

```
(market (RELCLS (sell +CHARACTERISTIC
                      (LOC market)
                      (OBJ <goods>))))

==>

(market (COMP <goods>))
```

This rule would then account for the following compounds:

```
meat market
fish market
computer market
```

Rhyne approaches the general problem from the point of view of language production rather than language understanding. This places the focus on issues which are somewhat different from mine. Rhyne begins with a complete semantic representation of a phrase which contains all the relevant information. His goal is to produce a surface level representation of the phrase using nominal compounds whenever possible. This bypasses many of the problems which I hope to address in my research. In principle, the production rules he uses to generate the surface level phrase from the internal semantic representation could be reversed to produce the semantic representation from the surface one. However, one then has to face the problems of multiple word senses, ambiguity (both structural and semantic), and the interaction of intra- and inter-sentence context.

3.2.3 Borgida

Alexander Borgida's 1975 thesis [BORGIDA] contains a chapter on the semantic interpretation of the noun phrase in which he proposes a simple classification of noun-noun modification types. His basic approach is to find an underlying verb which relates the head noun and its modifier. Given a head noun N and a modifying noun M, his classification is as follows.

(1) The head noun is an agential nominalization (e.g. owner, student, buyer).

In this case the underlying relationship between N and M is the event (or verb) from which N is derived. The head noun fills the agent case role of the event and the modifying noun can fill any of the other case roles. For example, the modifier M could fill the object role of the verb ("physics teacher"), the place/location role ("university student") or the time characteristic role ("night guard").

(2) The head noun is a result nominal (e.g. application).

As in the first case, the underlying relationship is determined by the verb that N is derived from. In this case, the agent case role is also free to accept the modifying noun M, as in "student application".

(3) The modifier is derived from a verb.

In this case the underlying relationship is determined by the verb from which the modifier is derived. The head noun can fill any of the case roles associated with this verbs case frame. Examples of this case are:

reception committee	(N is agent)
fish hook	(N is instrument)
completion date	(N is time characteristic)
meeting room	(N is place/location)

(4) Neither the head noun nor the modifier is derived from a verb, but one of them is closely related to a verb.

In this vague class, Borgida puts such examples as "steel factory", "dog house" and "university degree". His idea is that a noun such as "factory" is closely associated with the verb "make". The interpretation of "steel factory" would then be something like "a factory in which steel is made".

(5) A common fixed relationship holds between the head noun and its modifier.

This case includes a class of compounds which are related by a relation from a set of common fixed relationships. As examples of these relationships, Borgida gives: part of, type of, made of, and produces/yields.

In passing, Borgida mentions the problem on representing the limited productivity of noun compounding rules. One could formulate a rule to interpret "bus stop", "train stop" and (in general) "vehicle stop". Presumably, the interpretation would involve the case frame for the verb "stop" from which the noun "stop" is derived and result in "a place where a bus|train|vehicle stops". The compound "man stop" is perceived as bizarre, however, even when the same semantic relationship holds (i.e. we wish to refer to a place where a person stops). His proposal is to indicate on some nodes of his semantic network the compounds they can form and then to deduce that all subconcepts of these node may also participate in like compounds.

Borgida admits that his discussion of noun-noun modification is brief. In fact, he says of it that it "seems to be the most complicated form of noun modification". His analysis is too dependent on finding an underlying verb or event associated with one of the two nouns. This seems to be a reasonable heuristic, especially when one is dealing with nominalizations of verbs, but it does lead to several problems. One problem is that he slights the cases where it is not evident what the related verb/event might be (his cases number four and five). He offers no suggestion as to what it means for a verb to be closely related to a noun or how this is to be represented in a uniform way. Similarly, he gives no rules or heuristics for evaluating the appropriateness of relationships from the fixed class (his case number five).

A more serious problem arises when we attempt to constrain the productivity of some forms. Using his own example of a rule to interpret "bus stop", it seems we need some way to prevent the general find-a-related-verb heuristic from producing the interpretation of "man stop" as a place where people stop. Even if we can constrain the modifier to be a kind of vehicle, we run into trouble. Such a constraint would allow the bizarre compounds:

plane stop	golf-cart stop
van stop	fork-lift stop
motorcycle stop	

3.2.4 Marcus

In his thesis, Marcus [MARCUS] proposes a simple theory to solve what I call the modifier parsing problem. His hypothesis is that a parser with a buffer "window" of three constituents is sufficient to analyze deterministically noun-noun modifier strings of arbitrary length. Given the phrases:

water meter adjustment screw
ion thruster performance calibration
boron epoxy rocket motor chambers

he wants to produce the parses:

[[[water meter] cover][adjustment screw]]
[[ion thruster][performance calibration]]
[[boron epoxy][[rocket motor] chambers]]

His procedure is based on two assumptions. First, he assumes a semantic component which can decide upon the relative "goodness" of two possible noun-noun modifier pairs. For example, given the pairs "water meter" and "meter cover", this oracle would judge the first to be superior to the second, even though both are acceptable. The second assumption (which is the one with theoretical interest) is that arbitrarily long strings of nouns can be analyzed by examining the three left-most nouns (simple or compounded) in the string.

A third assumption which he does not explicitly mention captures the slight bias in English for constructions like ((N N) N) over (N (N N)). The kernel of this algorithm is a rule for parsing a string of three nouns. Assume that there are three nouns in the buffer: N1, N2 and N3. Let [N1 N2] stand for the modification of N2 by N1. His rule is then:

If [N2 N3] is semantically better than [N1 N2] then replace the buffer with N1 [N2 N3]. Otherwise, replace the buffer with [N1 N2] N3.

These assumption yield a simple algorithm which reads the first three nouns of a long string into the buffer and forms a compound noun out of ether the first and second noun or the second and third (under the direction of his semantic oracle). The buffer is then contracted and the next noun is pulled in. This process is repeated until the buffer has been reduced to two nouns, the first of which is taken to modify the second.

This is an interesting theory and one which would be important if true. His theory would greatly reduce the number of possible parses which would need to be considered for a long string of nouns. Without this constraint, the number of different parses for a string of N nouns is given by the recurrence relation:

$$f(n) = \mathrm{SIGMA}_{i=1}^{n-1} [\, f(i) * f(n-i) \,]$$

which has the closed form:

$$f(n) = \frac{1}{n+1} * \binom{2n}{n}$$

This function is bounded from above by the inequality:

$$f(n) < \frac{4^{n}}{n*\mathrm{sqrt}(pi*n)} + O(4^{n} * n^{-5/2})$$

With the Marcus constraint, the number of possible parses is reduced to:

$$f'(n) = 2^{n-2}$$

Figure 3.2 gives a table which shows values for f and f' for some small values of n.

Another way to state Marcus's constraint is to characterize the trees that can be produced. Let the <u>right depth</u> of a leaf of a tree be the number of right daughter links traversed on a path from the root of the tree to a that leaf. The <u>maximum right depth</u> of a tree is the maximum right depth over its leaves. Marcus's constraint is that the parse tree have a <u>maximum right depth</u> of two or less.

The Marcus constraintseems to work for the great majority of long strings of

number of nouns	$f(n)$ number of unconstrained trees	$f'(n)$ number of constrained trees
1	0	1
2	1	1
3	2	2
4	5	4
5	14	8
6	42	16
7	132	32
8	429	64
9	1430	128
10	4862	256

The Number of Constrained vs. Unconstrained Trees

figure 3.2

nouns related through modification. There are counter-examples, however. Marcus himself mentions a single counter example which he discovered. He assigns the phrase:

1970 balloon flight solar cell standardization program

the structure:

[1970 [[balloon flight][[solar-cell standardization] program]]]

which violates the *maximum right depth* constraint. He was unable, he says, to discover any other counter-examples.

There are, however, many more counter examples. Figure 3.3 lists several along with the structures which seem appropriate to me. What is interesting is the fact that a very high proportion of long sequences of nouns *do* observe this constraint. This is a fact which I would like to explain. It might be possible to characterize the kinds of relationships for which it is more likely to find a violation of the constraint. It may be that the more "primitive" relationships are more amenable to structures which do not obey the constraint (e.g. relationships like *source*, *time* or *location*). It is also likely that the constraint is a side-effect of the processing strategy which is used by people in attempting to interpret long sequences of nouns. If this is true, then it would be an important heuristic to capture in any computer program which attempts to do the same, if only to rank the potential interpretations with respect to the probability of matching the speakers intended meaning.

[Aluminum [automobile [water pumps]]]

["J.C. Whitney" [Automobile [water pumps]]]

[January [[automobile [[water pump] cover]] shipments]]

[[solid state] [RCA [color television]]]

[[back yard] [brick [dog house]]]

[Illinois [ERA [voice vote]]]

[prototype [MIT [LISP machine]]]

[plastic [toy [fire truck]]]

Examples which Violate the Marcus Constraint

figure 3.3

4 THE REPRESENTATIONAL FRAMEWORK

Much of the work described in this thesis, and its degree of success or failure, depends heavily on the representation of knowledge. In fact, the issues of knowledge representation are so important to this work that they are an active part of the research.(11) This chapter describes the representational system that has been chosen to encode the knowledge base used in this work.

The representational system I have used is an object-based language embedded (and implemented) in Lisp. The language is based on a collection of utilities for creating, accessing and manipulating objects called <u>frames</u>. These utilities, part of the FRL language [ROBERTS], have been modified and extended to provide a representational basis for this thesis. The resulting system is called FFRL for "Finin's Frame Representation Language" (apologies to JOVIAL).

This chapter describes the FFRL representation system. It begins with a brief attempt to place FFRL in the context of similar systems. The primitive structures of FFRL are then described, followed by a presentation of the surface language, A. The chapter ends with a discussion of data generator functions and their usefulness in inheritance networks.

4.1 <u>Other Frameworks</u>

The FFRL system is similar in spirit if not in abilities to several other recent representation systems. The original catalyst for the development of these systems is commonly held to be a paper by Marvin Minsky [MINSKY] in which he applied the word frame to an object-oriented representation language. The work of Schank, et. al. [SCHANK] and Fillmore [FILMOR68] also contained similar ideas. Researchers too numerous to mention have worked at developing these ideas. Let me list briefly some of the more developed systems: KL-ONE [BRACHMAN], KRL [BOBROW], FRL [ROBERTS], NETL [FAHLMAN], OWL [HAWKINSO], MDS, and TAXIS [MYLOPOUL]. More general discussions can be found in [CODD] and [FINDLER].

11 Sometimes threatening to overshadow the original topic, the study of the understanding of nominal compounds!

All of these systems share some common properties. They are object or entity oriented systems. The objects (whether they are called frames, scripts, concepts, prototypes, or descriptions) can represent and describe objects, events and concepts in the world as well as the representation system and its own descriptions. The objects are organized into a network by links between them. Special links are used to organize the entire collection of objects into an abstraction hierarchy, with links representing sub- and super-concept relationships.(12) Attributes and properties can be inherited through these links. A common feature of these knowledge representation systems is the ability to attach procedures to the objects. These procedures are run under certain specified conditions, such as the instantiation of a new object or the addition of a new link between objects.

4.2 The Foundation

The FFRL system is based on a data structure called a frame a schematic of which is shown in figure 4.1. A frame consists of a name and an arbitrary number of constituents called slots. Each slot, in turn, is composed of a name and an arbitrary number of facets. Finally, each facet has a name and an arbitrary number of data items. Frames are accessed through an index of their names.(13) In the following section I will discuss successively descending levels of this structure.

4.2.1 Frames

A frame is the basic data type of the system. It is an object with a name and a collection of slots or roles (I will use these names interchangeably). Figure 4.2 shows a sketch of a frame for the TO-FLY concept. Some of the roles of this frame are given, e.g. AKO, INSTANCE, SELF, AGENT, INSTRUMENT, etc. The semantics of these slots differ and can be broadly divided into several classes.

Two of the slots (AKO and INSTANCE) are primarily structural, serving to place the frame in the abstraction hierarchy. The AKO slot is named as an abbreviation for "a kind of" and points to a frame's super-concepts. The INSTANCE slot is the

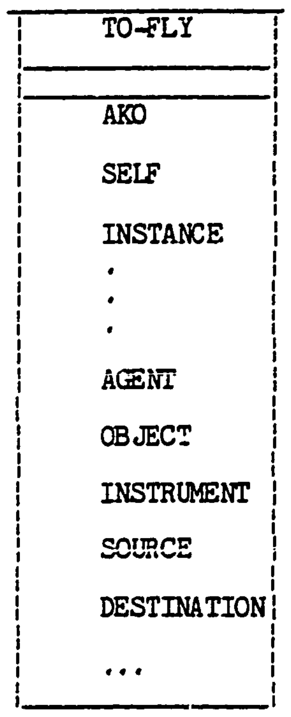
12 These are viewed as sub- and super-set relationships in some systems.

13 They are stored on the property list of the atom representing their name as the value of the FRAME property.

```
(FrameName

        (SlotName1

                (FacetName1
                        Datum1
                        Datum2
                        ...more data...
                        Datumn ))

                 (FacetName2 ... )

                   ...more facets...

                 (FacetNamen ... ))

        (SlotName2 ... )

           ...more slots...

        (SlotNamen ... ))
```

The Data Structure of a Frame

figure 4.1



A Frame for TO-FLY

figure 4.2

inverse of the AKO slot and points to a frames sub-concepts or instantiations.(14) Thus, the values stored in these slots must be the names of other frames.

Other slots, such as the SELF slot, contain information about the frame as a frame or meta-information about the frame as a concept. For example, the SELF slot is used as a repository for various kinds of knowledge about the frame. One facet is used to hold suggestions for methods to use when this frame is used as a pattern in the matcher and another facet can hold a list of the slots present in the frame.

Finally, the slots such as AGENT, INSTRUMENT and OBJECT in the example can be viewed as constituent roles of the concept that the frame describes. The semantics of these slots are not specified at the level of the representation system. When I talk of a concept's roles, I will usually be referring to this class of slots as opposed to the structural or "meta" slots.

A central feature of this and similar representations is that the frames are organized into an abstraction hierarchy (here defined by the AKO and INSTANCE slots). Figure 4.3 shows some fragments of an example abstraction hierarchy. In this figure, the most general concept meta-thing appears at the top of the hierarchy. All other concepts are descendant from this one. At the bottom of the hierarchy fragment we find the concept to-fly326 which represents an instance of the generic to-fly event, perhaps one which represents a particular event which is believed to have occurred in the real world. Appendix B gives the hierarchy of concepts used to exercise the nominal compound interpretation system. The basic role of the hierarchy is to allow attributes and properties to be inherited from a concept to its descendants. The details of how this is done will be described later. Let me point out that the abstraction hierarchy is not really a hierarchy (i.e. a rooted tree) but rather a directed graph with no cycles. That is, a given frame can have several immediate ancestors.

14 Whether or not one should or needs to differentiate generic sub-concepts from individual instantiations is a topic of debate. For a sample, see discussions in [BRACH77], [BRACH78A], and [BOBROW77]. I do not distinguish the two in any fundamental way in FFRL.

```
                        meta-thing
                           /\
                          /  \
                         /    \...
                      thing
                       /\
                      /  \
                     /    \
        abstract-thing   concrete-thing
                 .              .
                 .              .
                 .
              event
               /\
              /  \...
             /
        move-event
            /\
           /  \...
          /
transport-move-event
           /\
          /  \...
         /
       to-fly
        /\
       /  \...
      /
 to-fly326
```

Fragments from the Abstraction Hierarchy

figure 4.3

4.2.2 Slots

A slot or role (again, I use these terms interchangeably) is a component of a frame. It too has an internal structure, being composed of a name and a set of facets. Each facet expresses some information about the particular slot. Figure 4.4 shows a sketch of the instrument slot of the TO-FLY frame and some of its facets. In general, each slot can contain an arbitrary number of facets and each facet can contain an arbitrary number of data. In practice, the number of facets, and their semantics, is fixed and a property of a particular application or system. I will shortly give, for example, a catalog of the facets that we have used in this system.

In FFRL, slots can be (and are) described by frames which share their name. For example, there is a frame named AKO which includes information about the AKO slot. Various kinds of information are placed in these slot describing frames, such as the existence of an inverse slot (e.g. INSTANCE is the inverse of AKO), special information used by the matching procedure, etc. Procedures which must reason about the slots which a concept has can utilize these descriptions. This also allows the slots to be hierarchically organized.

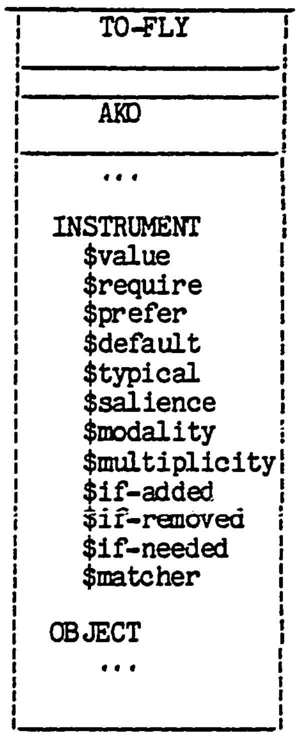
4.2.3 Facets

Just as a frame is composed of slots and a slot is composed of facets, a facet has an internal structure. A facet has a name and, in general, an arbitrary number of data. The nature and interpretation of the data is, of course, dependent on the facet. Each datum, in turn, contains two components - the information and a set of comments attached to it. Throughout this thesis, I will, for the most part, ignore this fact and assume that a datum consists solely of its information.

A Catalogue of Facets

The following paragraphs describe the more important facets that I have used. These are the primary facets found in role-type slots, i.e. those which describe the concept that the frame describes, rather than the structural relationship or meta-type knowledge about the frame or concept.

The $VALUE facet holds any actual values stored in the slot. If we have an



The Instrument Slot of TO-FLY

figure 4.4

individual TO-FLY event (say TO-FLY445) in which the destination is "SanFrancisco", then the $VALUE facet of the destination role of the TO-FLY445 frame would contain "SanFrancisco" (which is the name of the concept frame representing that city). The $VALUE facet can hold, in general, any kind and number of data items. Constraints for type and number can be defined with the $REQUIRE and $MULTIPLICITY facets.

The $REQUIRE facet describes requirements that all values (i.e. data filling the $VALUE facet) must meet. This facet can contain the names of one or more frames which are to be used as descriptions that a candidate value must match. In the TO-FLY frame, a requirement on the source and destination roles might be that they match the concepts for a city or an airport. This would be expressed by filling the $REQUIRE facet with the frames (a city) and (an airport).(15)

The $PREFER facet describes the preferred values for the slot. This facet can contain the names of one or more frames which are to be used as descriptions that a candidate value should match.

The $DEFAULT facet contains a value which can be used as a default value for the slot. It can be any sort of object, but only one datum is allowed.

The $TYPICAL facet can contain one or more objects which are typical or commonly occuring values.

The $MATCHER facet contains information for the concept matcher. The use of this facet is discussed in the chapter describing the matcher, but as a preview, I will say that the data in this facet suggest specialist procedures to use when attempting to match this slot against another.

The $IF-ADDED, $IF-REMOVED and $IF-NEEDED facets contain expressions to be executed under certain conditions. The expressions in the $IF-ADDED facet are executed whenever a value is added to the $VALUE facet. Those in the $IF-REMOVED facet are run whenever a value is removed and expressions in the $IF-NEEDED facet whenever an attempt is made to retrieve a value from an empty $VALUE facet.

The $MULTIPLICITY facet describes the number of values that may be stored in

15 A more accurate representation would be that the source and destination must specify an airport. One way to do this is to refer to one by name. Another is to name the city in which the airport is located.

the $VALUE facet. It contains a tuple (as a dotted pair) of the minimum and maximum number of values allowable. A NIL in the maximum position (which, due to the dotted-pair notation is the same as having a singleton) denotes that any number of values may be stored. Some examples are:

(0 . 0)	No values may be stored.
(0 . 1)	Zero or one value may be stored.
(1 . 1)	Exactly one value must be present.
(1 . 3)	At least one and at most three values may be present.
(0)	Any number of values may be stored.
(1)	One or more values may be stored.

The $MODALITY facet describes the criterial relationship of this slot to the enclosing frame. The possibilities for the facet are _obligatory_, _optiona_, _prohibited_ or _dependent_. If this facet contains the symbol _obligatory_ then the slot must be filled with a value for the frame to be logically complete. A _optional_ marker says that the slot need not be filled. The _prohibited_ marker says that the slot must NOT be filled (16). Finally, _dependent_ indicates that the value of this slot is logically dependent on the contents of another of the frame's slots.

The $SALIENCE facet contains a symbol measuring the overall importance of this role to the concept as a whole. It should contain a measure from the "low...high" scale, these being the five concepts: very-low, low, medium, high, very-high.(17)

Figure 4.5 summarizes these facets and what they can contain.

4.2.4 _Data_

Facets contain data. A datum can be, in general, any Lisp s-expression. Particular facets can impose further restrictions, such as a AKO's $VALUE facet requiring a frame name or a $MODALITY facet requiring one of the atoms "optional", "obligatory", "prohibited" or "dependent". A facet can contain more than one datum in the general case as well. There is one further level of structure beyond the datum, the _comment_. In the FFRL system as I have used it in this research, I have

16 One might wonder how this would ever be useful. Why not just remove the entire slot from the frame? The answer is that the _prohibited_ moda ity provides a way to "cancel out" a slot inherited from a super-concept.

17 The "low...high" scale is defined as "(a low-high-measure is (a linear-measure) with instances = (a very-low) & (a low) & (a medium) & (a high) & (a very-high) lowend = (a very-low) topend = (a very-high))

facet	description	number	datum
$VALUE	value(s)	any	anything
$REQUIRE	restriction(s)	one	frame(s)
$PREFER	preference(s)	any	frame(s)
$DEFAULT	default value	one	anything
$TYPICAL	typical value(s)	any	anything
$SALIENCE	salience measure	one	a low..high measure
$MODALITY	modality	one	a modality (optional, etc.)
$MULTIPLICITY	multiplicity	one	an integer range
$IF-ADDED	if-added demons	any	executable expression(s)
$IF-NEEDED	if-needed demon	any	executable expression(s)
$IF-REMOVED	if-removed demons	any	executable expression(s)

Facet Summary

figure 4.5

made little use of comments. Suffice it to say that any datum can have a comment attached to it. The comment is a list composed of a "label" (an atom) and an arbitrary number of "messages" (arbitrary s-expressions).

4.2.5 Referring to Slots

In the FRL system only frames are first class representational citizens. There is no way for one to "point to" a slot of a frame or a facet of a slot of a frame. My work has necessitated the ability to refer to a frame's slot in order to talk about, for example, the agent of an event. I have developed a convention in FFRL for referring to a slot or role of a concept.

To refer to a slot in FFRL one needs to instantiate a frame which is descendant from the generic concept for a role. A role frame contains two main slots, a role which contains the name of the role being referred to and a frame which contains the name of the frame containing the role. Figure 4.6 shows a part of the concept hierarchy under the generic role frame as well as some individual role frames.

4.2.6 Inheritance

Frames can inherit slots, facets and data from any other frame which can be reached by following chains of AKO links. The mechanism is similar whether one is retrieving an entire slot of a frame, a certain facet of a slot of a frame, or even a particular datum on a certain slot of a given frame. There are many different kinds of inheritance systems possible, each having its own behavior and uses. The issue of data scoping is discussed in a separate section. Here I want to introduce the most common inheritance mechanism used in FFRL.

Given an item to retrieve (e.g. a slot, facet or datum), the procedure begins by looking for the item in the local frame. If it is stored there, then we have found it and can stop. Otherwise, we look for the item in each of the frame's immediate ancestors, i.e. the frames found in the AKO slot. We stop whenever we've found the item. Thus, a frame's ancestors are searched in depth-first order.

```
(a role is (a Meta-thing) with
   ; The genric role frame.
   basic = *
   role preferably (a slot)
        multiplicity =1
   frame matching (a Meta-Thing)
         multiplicity =1))))


; The following three specializations are for events.


(an AgentRole is (a role) with
    role = 'agent
    frame matching (an event))


(an ObjectRole is (a role) with
    role = 'object
    frame matching (an event))


(an InstrumentRole is (a role) with
    role = 'instrument
    frame matching (an event))


; And here are some examples of some individuals


(an AgentRole with frame = (a to-fly404))


(a Role with
   role = (a self)
   frame = 'Meta-Thing)
```

Frames for Roles

figure 4.6

4.3 The Surface Representation

The representation language that I have based my work on comes with a host of functions for creating, manipulating, altering, displaying, and accessing frames. I have found it convenient, however, to define a simple language (called A) to perform some of the more common functions. This section describes this surface language.

The A language was designed with several goals in mind. The primary goals were (1) to make it easier to define new concepts; (2) to facilitate the creation of specializations of existing concepts; and (3) To ease the extracting of data from concepts. One can add information, but there are no real facilities for altering information in an existing frame. A has also been used as a language in which to describe individual frames and even networks of frames.

4.3.1 The Basics

A is a simple language, consisting of just seven main functions: A, AN, THAT, THE, THIS, THESE and THOSE. Furthermore, two of the seven functions, A and AN, are identical! These functions take an arbitrary number of arguments and create, modify or find a frame which the arguments describe. The A and AN functions are used to create, define and describe FFRL frames. Some examples of very simple A expressions, all involving the concept frame to-maintain, are:

```
[1] (a to-maintain)
[2] (a to-maintain is (an event))
[3] (a to-maintain is (an event) with agent matching (a person))
```

The actions of these three examples depends on whether or not a concept frame named "to-maintain" exists or not.

Let us first assume that one does not. The first example will simply create an empty frame and give it the name "to-maintain". The second example will create a frame named "to-maintain" as an instance of the frame referred to by the A expression "(an event)". Thus the "is" term introduces a frame's super-concept (18). The third example instantiates an event frame, names it "to-maintain", and adds to it a slot specified by the phrase "agent matching (a person)". This phrase

18 What actually happens is that the expression after the "is" term is evaluated and this result is added as a value of the AKO slot.

creates a slot named "agent" with one facet, a $REQUIRE which contains the datum resulting from the evaluation of "(a person)" (which is just a reference to a frame named "person").

Why does the term "matching" map into the $REQUIRE facet? Convention. The A language allows one to associate arbitrary synonym atoms with the standard facet names. The synonyms that I use are given in the following table. (Refer to section 4.2.3 on page 54 for the semantics of these facets.)

facet	synonyms
$VALUE	=, equal, value
$REQUIRE	matching, restricted-to, requirement
$PREFER	preferably, preference
$DEFAULT	default, defaulting-to
$TYPICAL	typically, often, typical
$MODALITY	modality
$MULTIPLICITY	multiplicity
$SALIENCE	salience, importance
$MATCHER	to-match, matcher
$IF-ADDED	if-added
$IF-REMOVED	if-removed
$IF-NEEDED	if-needed, computed-from

In a similar vein, to make the entering and display of the modality values more natural, the following bindings are provided:

variable	value	variab e	value
=0	(0 . 0)	<2	(0 . 1)
=1	(1 . 1)	<3	(0 . 2)
=2	(2 . 2)	>0	(1)
1-2	(1 . 2)	>1	(2)

Returning to our example A expressions, suppose that a frame named "to-maintain" already exists? In this case, the first example simply returns a reference to the frame so named. The second example adds to the existing "to-maintain" frame the information that "(an event)" is one of its super-concepts.(19) In the final example, the expression also adds to the existing "to-maintain" frame the information that it has an agent slot, and that this slot exhibits a preference for values matching the frame referenced as "(a person)". If there is no "is..." clause and there is a "with..." clause, as in the example,

19 What if this information is already stored in the "to-maintain" frame? In general, adding information that is already there is a no-op. It does not trigger any If-Added procedures.

```
(a to-maintain with object = (a plane))
```

then a new instance is always created. Thus the inclusion of an "is..." clause indicates that we are defining a concept or, if a definition already exists, that we are adding to that definition.

4.3.2 Creating New Instances of a Frame

We have seen two ways to create a new frame: (1) invoking A with a previously unknown frame name and (2) including a "with ..." clause. To enable one to create a new instantiation of a previously defined frame, one can place the modifier "new" before the name of the frame as in the examples:

```
(a new to-maintain)
(a new to-maintain is (a repair-event))
```

The first example returns a reference to a newly instantiated "to-maintain" frame and the second a reference to a newly instantiated "to-maintain" that is also a sub-concept of the "repair-event" frame. One can also apply the synonymous modifiers "unique" and "individual" with identical results.

4.3.3 Adding to an Existing Frame

So far, we have only seen one way to add information to a previously defined frame with A. Including an "is..." clause in the description causes an existing frame to be modified by the rest of the description (i.e. the "is..." clause and any "with..." clause). Placing the modifier "old" before the name of a frame will cause the existing, generic definition to be used in every case. Thus, with the application of this modifier, one can add slots, facets of slots, or data of facets of slots to an existing frame. Consider the following examples in which the "old" modifier is used:

```
(an old to-maintain with object preferably (an artifact))
(an old to-maintain with location typically (a repair-shop))
```

The first example modifies the existing definition of the generic "to-maintain" frame with the information that it has an object role and that this role has a preference for values which match the description (an artifact). The second adds the information that "to-maintain" has a location role which has a typical value matching the frame (a repair-shop).

4.3.4 Referring to Roles

The A language supports a special syntax to allow reference to a role of a concept. The details of how such a reference is actually represented has already been discussed. The A expression

```
(an instrument of (a to-maintain))
```

is a reference to the instrument role of the concept frame named "to-maintain". Recall that this is actually represented as a frame descendant from the generic ROLE frame. Thus this reference can be used wherever a frame is expected. In particular, we can have such expression as:

```
(a pump is (an instrument of (a to-pump)) ....)
(a teacher is (an agent of (a to-teach)) ...)
(a student is (a recipient of (a to-teach)) and
                 (an agent of (a to-learn)))
```

4.3.5 Conjoining Values

As you may have noticed, some of the constituents (or arguments, if you like) of these A expressions are evaluated and some are not. Those which are evaluated are called values and appear in one of three places:

```
(1) As the object of an "is..." clause, as in
    (a wrench is (a tool) with ...)

(2) As the datum of a facet, as in
    (a wrench with
       raw-material = 'Steel )

(3) As the object of an "of..." clause, as in
    (a destination of (a transport with
                         vehic e matching (a p ane)))
```

All other constituents of an A expression are unevaluated. Values share another property in their ability to be conjoined. Wherever one can use a value (except as the object of an "of..." clause), one can use a conjunction of two or more values. The conjunctions are "and", "or", and "&". Presently, all have the same effect, that of collecting values into a set. Some examples of conjoined values are:

```
(a pump is (a machine) and (an instrument of (a to-pump))
(a person with children matching (a boy) and (a girl))
(an integer with divisors = 2 & 3 & 7)
```

4.3.6 Adding Comments

The underlying representation language allows any datum on any facet to have a comment attached to it. This comment can be any arbitrary s-expression. What the comment means and how it is used is up to the user (20). Any value (e.g. an expression following a facet name) can have a comment attached to it with the "->" modifier. For example, in

```
(a to-maintain with
   time = (a date with
             year = 1980
             month = 'February
             day = 29) -> 'LeapDay
   object = (an F4 with Buser = 800436))
```

there is a comment "LeapDay" attached to the datum in the $VALUE facet of the time slot.

4.3.7 Deictics

This section describes the deictic functions THIS, THAT, and THOSE. Briefly, THIS is used to refer to a concept that is in the process of being described, THAT is used to refer to one other concept by definite description and THOSE to one or more concepts by definite description.

In some of the examples described in the A language that you will see in this thesis, you may discover a frame reference beginning with "this" as in:

```
(a man with
   spouse matching (a woman with
                      son = (this man)))
```

In the present system, the "this" article is used to create embedded reference. In this example "(this man)" refers to the entire A expression. In the course of creating a frame from an A expression, the language interpreter maintains a stack of frames created and their prototypes. If a "this" reference is encountered, the referent is found by examining this stack for the topmost entry with a matching prototype. In the example, when the "(this man)" is encountered, the stack looks like:

```
( woman . woman345 ) <- top
( man   . man344   )
```

20 Comments are used to control the scope of inherited data, for example.

and the referent is found to be <u>man344</u> since that is the top-most entry with a prototype equal to <u>man</u>.

The THAT introduces a description which is used to search the database for a concept which matches it. Some examples are:

```
(that person with
      occupation matching (a teacher)
      address matching (an address with city = 'Urbana))

(that event with
      agent matching (a person)
      object = (that plane with Buser = 120034))
```

Note that the arguments to THAT are the name of a frame and a "with..." clause. These arguments are used to create a pattern frame (by simply applying the A function to them). The database, or rather a portion of it, is then searched to find a frame which matches this pattern. The portion that is searched is just those frames which are descendants of the immediate ancestor(s) of the pattern (21). In the examples, the portions would be all descendants of the PERSON and EVENT frames. The referent is taken to be the first matching frame encountered.(22)

THOSE differs from THAT in that it can be used to find more than one referent. Some examples are:

```
(those event with object matching (an F4)
                  time matching (a time with month = 'January
                                             year = 1980))

(those 4 to-repair with object matching (an A7))
```

The first example would retrieve all frames which represent an event occurring in January, 1980 in which the object was an A7 aircraft. The second example would return up to four frames which were to-repair events involving an A7 as an object.

4.3.8 <u>Data Retrieval</u>

A final function, THE, is used to extract data from a frame. Some sample expressions are:

21 The descendants are searched in depth first order.

22 Since the pattern is an descendant of its immediate ancestors, it will be tried as a candidate referent. Thus we include the additional requirement that the referent be distinct (i.e. not EQ) from the pattern.

```
(the object value of 'to-replace345)
(the age of (the spouse of 'TimFinin))
(the child (s) of 'TimFinin)
(the vehicle preference's of (a to-move))
(the Type of (the instrument of (that to-fly with time = ...)))
```

The first example, which extracts the datum stored in the $VALUE facet of the frame TO-REPLACE345, shows one form of the arguments: slot-name, facet-name, "of" and frame-name. In the second example, we notice that the facet-name is optional and defaults to $VALUE. This example also shows that the frame is specified by an arbitrary expression which is evaluated. In both of these examples we get only the _first_ datum that is stored in the specified location. Any other data will be ignored. To retrieve _all_ of the data, one must put the _plural_ indicator "(s)" (23) after the slot or slot and facet specification. Example three shows the use of the (s). The fourth example extracts all of the data in the $PREFER facet of the TO-MOVE event. The fifth example shows how one can combine this extraction function with the retrieval function.

One last feature that must be mentioned is that the last argument, the one which specifies the frame, can in fact specify a set (or list) of frames. In such a case, data is retrieved from the specified slot and facet of all of the frames in the list. Some examples,

```
(the name of (the child (s) of 'TimFinin))
(the day of (the time of (those to-repair with
                                 object matching (an engine))))
```

The first example will return a list consisting of the name value from each of the children from the frame 'TimFinin. The second will extract the day part of the times associated with all occurances of repairs to engines.

4.3.9 _Generating Descriptions_

So far we have described a system for translating expressions in one language, A, into forms of another, FFRL. There is also a component which can perform the

---

23 An alternative is the "'s" indicator.

inverse translation, from FFRL frames into A expressions. This section describes the generation of A expressions from FFRL frames.

The task is to create an A expression which describes an FFRL frame. Some parts of this task are relatively straightforward, e.g. transforming FFRL structures into corresponding A structures, mapping FFRL facet names into the descriptions used in A, supplying quotes to suppress the evaluation of values, etc. Other parts addresses some non-trivial issues, namely how much should one describe and how can one best describe a network of FFRL frames in a linear format.

Let's look at the first question first. What should one do when one wishes to describe a datum that is itself an FFRL frame? A narrow view of the problem would be that it is our task only to describe in detail the original frame we were given. Any other frames that the original points to should be described in the briefest possible way, i.e. by name reference. This is a simple but unnatural solution. Some of the frames pointed to by the original one we were given to describe may be an integral part of the concept, a part that any description of the concept should describe as well.

Look at the problem as one in which we are given a network and told to describe one of the constituent nodes. There may exist some neighborhood around the given node that we should describe as well. How can we draw the boundary?

A view at the other end of the spectrum would be that we should expand every node we encounter. In general, the FFRL frames form a highly connected network so that this approach would require the elaboration of the entire network in order to describe any one frame. Worse yet, the network will contain many cycles. There must be some way to prevent the attempt of an infinite description.

To solve the problem of cyclical self-embedded descriptions, the following is done. During the process of describing a frame we keep a list of the frames which we have described or are in the process of describing. Whenever we encounter a frame and decide to describe it (rather than just using its name), we first check the list. If the frame is a member of the list, then we use its name rather than generating a description. If it is not a member of the list then we add it to the list and begin to describe it in detail. Thus in the course of a given description, no frame will ever be elaborated more than once.

To partially solve the problem of how much to describe, the description component of the A language includes several hooks whereby the user can specify what to describe. The default is that only the original frame is described. Any other frames which are pointed to are described by name only. Through the hooks, the user can cause some of the other frames to be described. This is implemented by maintaining a set of variables which guide the description component in deciding which frames to expand. The variables are:

* A list of "unimportant" frames never to describe.
* A list of "important" frames to describe whenever possible.
* A list of "unimportant" slots, the contents of which should never be described.
* A List of "important" slots, the contents of which should be described whenever possible.

A more general solution might be to allow for a function which, when given the names of a frame, slot and datum, would return a recommendation on whether or not to expand the datum. We could then, for example, make the expansion decision depend on the _salience_ facet of the slot.

4.3.10 The Syntax of the A Language

Figure 4.7 gives an informal BNF description of the syntax of the A language. In this description, literal terms are enclosed in double-quotes, meta-descriptions are enclosed in square-brackets and parentheses are used to delimit optional constituents.

4.4 Data Generators

The notion of a _data generator_ has been found to be very useful in this work. Briefly, a _data generator_ is a function which can produce all of the data accessible from a particular frame, slot and facet one datum at a time. It is not necessary to collect all of the data prior to using it. Instead, a generator can be created and, each time a new datum is needed, it can be obtained by asking the generator to produce the next item.

This is not a novel idea. It is essentially the same as CONNIVER's generator functions [SUSSMAN], LUNAR's enumeration functions [WOODS77B] and many other stream

```
A-exp   -> "(" A-exp1 ")"

A-exp1  -> a ¦ this ¦ that ¦ those ¦ the

a       -> "a" phrase ¦ "an" phrase

this    -> "this" framename

that    -> "that" phrase

those   -> "those" ( integer ) phrase

the     -> "the" slotname ( facetname ) ( plural ) "of" F-values

phrase  -> framename ( "is" values ) ( "with" slots ) ¦ rolename "of" frame

slots   -> slot ¦ slot slots

slot    -> slotname ( facets )

facets  -> facet ¦ facet facets

facet   -> facetname values

values  -> L-value ¦ L-value conj values

comment -> "->" expression

plural  -> "(s)" ¦ "'s"

conj    -> "and" ¦ "&" ¦ "or"

framename       -> ( modifier ) atom

modifier        -> "new" ¦ "unique" ¦ "individual" ¦ "old" ¦ "generic"

facetname       -> "=" ¦ "matching" ¦ "preferably" ¦ [...]

rolename        -> [a lisp atom denoting a role]

integer         -> [any integer]

atom            -> [any lisp atom]

expression      -> [any lisp s-expression]

L-value         -> [any lisp s-expression, evaluated]

F-values        -> F-value ¦ [any Lisp expression evaluating to a list
                             of frames, evaluated]

F-value         -> [any lisp expression which evaluates to a frame, evaluated]

; literals are in double-quotes.
; meta-descriptions are in square-brackets.
; optional constituents are in parentheses.
```

An Informal BNF for A

figure 4.7

oriented facilities. There are, however, sufficient details of some interest to warrant a general discussion here.

4.4.1 Data Scoping in an Abstraction Hierarchy

Generator functions are very useful when dealing with information stored in an abstraction hierarchy. One common use of such a hierarchy is to allow for a concept's data to be either stored locally or, if no local value is present, to be inherited from an ancestor. Usually, the first value found while searching from the local concept frame upwards is the on y one which is considered relevant. This allows one to easily encode a general rule high in the hierarchy and place exceptions or specializations lower in the hierarchy as needed. Thus, the scope of a datum attached to a frame is some subset of that frame's descendants.

Another convention for scoping the data in the hierarchy is slightly less common. In this convention, a datum placed on a concept applies to all of that concept's descendants, even if some of the descendants have local data stored in the corresponding slot and facet. Thus, local data is additive and serves to further specify rather than to completely describe. Cancellation of a frame's ancestor's data (e.g. to encode an exception) must be accomplished by another method. Thus, to gather all of the data accessible from a particular frame, slot and facet, one must always search from the given frame all the way back to the root. Data may be found anywhere along this path and is always applicable.

4.4.2 Data Generators improve efficiency

When this second kind of scoping convention is used, one should worry about the efficiency of retrieving data attached to a particular frame, slot and facet. Need one always scan the entire tree to process the data?

A typical task is to examine the data attached to a particular frame, slot and ~~facet until some condition is met or there is no more data to examine. In cases~~ where termination is caused by the condition's satisfaction, the work done in collecting the unexamined data is wasted.

This is reminiscent to the McCarthy style of boolean evaluation in LISP [MCCARTHY]. LISP's two primitive boolean functions (OR and AND) take an arbitrary number of arguments. In evaluating an N component OR, LISP stops evaluating when the first non-NIL value is encountered. Similarly, LISP stops evaluating the arguments to the AND function when the first NIL value is found.

The notion of a data generator function is a useful device for increasing the general efficiency in the processing of inherited data which is scoped in this manner. To get the data attached to a certain frame, slot and facet, one first creates a generator for this task. Then, each time a new datum is needed, the generator is "pulsed" and it does whatever is necessary to produce the next datum. As a side effect, invisible to the user, the generator "remembers" its state so that the next time it is pulsed it will yield the next appropriate datum. This frees the user from recording and remembering the details of the searches through the hierarchy when processing the data accessible from a frame, slot and facet.

In addition to a general data generator, I have made heavy use of specialized generators for enumerating a frame's ancestors and descendants.

5 CONCEPT MATCHING

Much of my approach to the interpretation of nominal compounds depends on having a good matcher. The application of the semantic interpretation rules generally involves the testing of constituents to see if they fit certain conditions. In considering a candidate rule, we ask "Does the modifying concept fit _this_ description? Does the modifier fit _that_ description?". This task of determining whether or not a given concept frame fits a given description is the job of the matcher.

This chapter starts by describing the concept of matching as it has been used in other systems. The matching task, as it is formulated for the work in this thesis is then described. The FFRL matcher is next described in some detail. Finally, some examples of the matcher's behavior are shown.

5.1 Other Matchers

There is a long tradition of matchers in AI. The general notion of a matcher is a natural one whenever one's behavior depends on recognizing a given situation, formula or expression as an instance of a more general prototype. Thus matchers find their place in such disparate work as computer vision, image processing, theorem proving, automatic programming, problem solving and natural language processing. I will briefly mention several related matchers and contrast them with the one used in this research. The matchers that I will discuss can be found in the MERLIN system [MOORE], in the KRL representation system [BOBROW77][BOBROW77] and in [ROSENBER].

In many respects, my matcher is most similar to the one proposed for the KRL representation system. Both provide a framework in which many specialized methods can be applied to the matching task. In both, the items being matched can suggest methods to try. Both can deal with positive and negative evidence. Both access descriptions which are deduced or inherited from generic information about the concepts being matched. The primary differences are that the KRL matcher contains a multiprocessing control structure and has the ability to allocate computational resources to subtasks. The matcher that I have developed uses a straightforward depth-first approach when pursuing a match and has no control over the amount of computation to be dedicated in the pursuit.

The matcher described in [ROSENBER] also shares many similarities with our matcher. A major difference, though, is that Rosenberg's matcher is designed to do several different kinds of matching. It is similar to a unification-syle matcher in that either of the two concepts to be matched can contain pattern-like elements. The matcher is also responsible for finding referants for descriptions.

A unique feature of the MERLIN matcher [MOORE] that is not explicitly present in my matcher is the ability to compute a forced match. In a forced match, the matcher is not trying to determine whether or not a match exists between the pattern and the object. Rather, its task is to find a mapping between constituents of the pattern and constituents of the object which would admit a match.

5.2 The Task

The task of the matcher is to take two concept frames, a pattern and a target, and indicate whether or not a match exists between them. What is meant by _a match_ between two concepts requires some elaboration. The notion of _a match_ used in this work has been motivated more by the demands of the task at hand (understanding nominal compounds) than by very general issues. In particular, it is not the goal of this work to provide the most general matching facility or one which can do a wide variety of kinds of tasks all referred to as matching.

For the purposes of this work, the pattern concept is take to be a general description of a class of objects. The target is taken to be a description of another class of objects. The target concept may or may not refer to an individual. The match succeeds if the description specified by the pattern includes the one provided by the target. That is, if every object described by the target description is also described by the pattern description.

The matcher is designed to provide more than just a match/no match answer. The implemented matcher returns a four tuple whose elements are, in order, the _result_, the _score_, the _bindings_ and the _justification_. The _result_ element is one of the atoms +MATCH, -MATCH or ?MATCH. A +MATCH value indicates that a definite match was established. The -MATCH value indicates that a definite mismatch was proved. The ?MATCH value reports that the matcher was unable to establish either a match or a mismatch, i.e. it represents a "don't know". The second element, the _score_, is an integer used to rate the strength of the match. The third element, the _bindings_ provides a way to extract the associations between constituents of the pattern concept and those in the target concept. The exact format of this term and how it is used will be described later. The final element of the four-tuple returned from the matcher is the _justification_, an s-expression which identifies the name of the particular match method which produced the result.

A few examples may be the fastest way to demonstrate these aspect of the matcher. Figure 5.1 shows some examples of the matcher in operation.

In example [1] the matcher is given the pattern (A VEHICLE) to match against the object (A CAR). It reports that it did establish a match (+MATCH is the first element), that the match had a merit figure or score of four (4 is the second element), that there were no bindings (NIL is the third element) and that the method used to establish the match was FM-AKO (the fourth element).

In example [2] the matcher is applied to the pattern (A WOMAN) and the object (A PERSON). Here the method FM-AKO is used again, and reports a mismatch (a -MATCH) with a score of four, and no bindings. Note that when the pattern is a sub-concept of the object, we say that a match does NOT exist between them.

In the third example from figure 5.1 the pattern and object descriptions are more complex. Here the matcher first notes that, in general, the object concept (A BICYCLE) matches the pattern concept (A VEHICLE). It then goes on to compare the specifications of the two concepts. If the object is to match the pattern, each role in the pattern should match a corresponding role in the object. Here this condition will be met if the two object roles match. Thus, the matcher is recursively applied with the pattern concept (A WOMAN) and the target concept "JanetFinin" (an individual). Since these match, the overall match is successful.

Our fourth example is another in which there is a mismatch. Note that the result is -MATCH, the score is minus infinity (-oo), there are no bindings and that

```
[1] >> (fmatch (a vehicle)(a car))

    (+match 4 nil fm-ako)


[2] >> (fmatch (a woman)(a person))

    (-match 4 nil fm-ako)


[3] >> (fmatch (a vehicle with owner matching (a woman))
               (a bicycle with owner = 'JanetFinin))

    (+match 2 nil fm-recurse)


[4] >> (fmatch (a vehicle)(a animal))

    (-match -oo nil fm-basic)


[5] >> (fmatch (an event with agent = ?who)
               (a to-repair with agent = 'person324))

    (+match 2 ((who . person324)) fm-recurse)


[6] >> (fmatch (a move-event with
                 agent matching (a person with
                                 home matching
                                   (a city with
                                     state matching
                                       (a state with
                                         region = 'MidWest)))
                 source matching (a city with state = 'Illinois or
                                                      'Indiana)
                 destination matching (a city with
                                        state = 'California))
               (a to-fly with
                 agent = 'TimFinin
                 source = 'Urbana
                 destination = 'SanFrancisco
                 carrier = 'TWA and 'Ozark
                 flight-number = 846 and 142))

    (+match 4 nil fm-recurse)
```

Examples of the Matcher

figure 5.1

the method used is FM-BASIC. This method works by discovering that the two concepts (a vehicle) and (an animal) belong to two mutually exclusive basic categories.

In the fifth example, we demonstrate the ability to extract associations between elements of the pattern and elements of the object. The ?WHO term in the pattern concept signals that it will match anything and, furthermore, that whatever it does match should be remembered. This match does succeed with ?WHO corresponding to PERSON324. This fact is reflected in the third element of the four tuple returned, the bindings. The bindings element is an association list whose members are pairs of identifiers and their corresponding matched data.

In the sixth and final example we see that the pattern concept can be arbitrarily complex and involving varying degrees of specificity. Note that some logical connectives are allowed. This pattern concept represents a trip from a city in Illinois or Indiana to a California city made by a midwestern resident. This example also points out that it is sufficient for each element of the pattern to be matched with an element of the target. There is no penalty for the existence of "extra", unmatched elements in the target concept.

A simple convention allows the matcher to be applied to targets which are not FFRL frames. If the target is not a frame, then the matcher will attempt to match the pattern against a newly instantiated <u>Lisp-s-expression</u> frame with its VALUE slot filled with the object. For example, (fmatch (an integer) 3.) gets reformulate as

```
    (fmatch (an integer)
            (a Lisp-s-expression with value = 3))
```

To make this work requires the addition of a special match method which can be attached to the integer concept, but this will be discussed later.

5.3 <u>The Organization of the FFRL Matcher</u>

The matcher is organized to supply a unified environment in which to apply one or more specialist matching methods to the pattern and target concepts. In this respect it is similar to the matcher proposed for the KRL system. The matcher function takes from one to three arguments. The first is a concept frame which is to be used as the pattern or description. The second, the target, is a concept frame which is to be compared to the first. The final argument provides a list of matching function which are to be used to establish whether or not a match exists between the pattern and target.

Defaults for the second and third arguments are as follows. If the second argument is not specified, it defaults to the current binding of the variable :VALUE.(24) If the third argument does not specify a set of match methods, they are taken from a list which contains a number of general purpose methods.

The matcher's job is to maintain certain bookkeeping functions (such as keeping a dynamic stack of the concepts being matched) and to apply the match methods to the pattern and target until one reports that a match or mismatch has been established. If all of the available methods are exhausted without success, a ?MATCH type result is returned. In most cases, this is ultimately taken to be evidence for a mismatch.(25)

The matching methods can come from four sources. First, they can be explicitly suggested through the match function's third argument. Second, they can be found on the list of default global match methods. These methods are ones which are applicable to all concepts and which have a high degree of general utility. The third and fourth sources are the concepts participating in the match. Both the pattern and the target frames can specify particular match methods to apply to the match. The $MATCH facet of the SELF slot of each frame is examined to find these methods.

As an example of a specialist match method from the PLANES/JETS domain, consider matches involving aircraft. Every individual aircraft in the data base (these are represented by the generic concept 3M-PLANE) has a unique serial number. Thus, one specialist which is attached to the 3M-PLANE concept is:

> If both the pattern and the object concepts have a value for the Bureau Serial Number, then return a +MATCH if the values are identical and a -MATCH if they are not. If one or both of the concepts does not have a serial number value, then return a ?MATCH result.

It is important to remember that the specialist matching methods attached to a

24 This follows a convention used in the underlying representation language, FRL.

25 In the implemented system mismatches are not weighted as positive matches are. If they were weighted, this could be used to produce a -MATCH with a weak score.

frame are inherited by all of that frame's descendants. Thus the example specialist may be invoked whenever a match involving any concept descendant from 3M-PLANE is attempted. Thus, this method provides a quick way to establish a match or mismatch just in case both the pattern and the target are individual 3M-planes.

As another example, consider the problem of recognizing (a Lisp-s-expression with value = 3) as an instance of (an integer). The solution is to attach a match specialist to the integer concept which checks to see if the target is a Lisp-s-expression which has a number in its value slot.

As a final example, consider a representation of a generic SET frame which has slots for the actual members and one for a description of the typical member. A specialist for matching concepts descendant from the SET frame might involve checking that all of the values in the target's _members_ slot match the pattern's _typical-member_.

5.3.1 The Frame Matching Methods

In this section I will describe the set of frame matching methods that I have found to be useful. The most powerful one, _fm-recurse_, is a general recursive matcher. A discussion of its operation will be given in the next section. The other specialized methods are all relatively simple. Each one was written to take advantage of a test for matching one frame against another that is either very common or very easy to compute.

The method _fm-same-frame_ is the simplest. It checks to see it the pattern and target are the same. The fact that this occurs fairly often and that the test is trivial justified the inclusion of this method. If the pattern and the target are identical, a +MATCH with a high score is returned. Otherwise a ?MATCH is returned.

The matching method _fm-recall-result_ is also simple. It tries to recall a previous result of matching the pattern concept and the target concept. Each time the matcher is applied to a pattern and target concept, the result of the process is associated with the two concepts in a kind of short-term memory. This information is stored in the SELF role of the concepts, under the $MATCHES facet for the pattern concept and the $MATCHED-BY facet for the target concept. For example,

```
(FMATCH (a water-vehicle) (a car with owner = ...))
```

where (a car with owner ...) becomes CAR322, might result in

(-MATCH -oo nil fm-basic)

and have the side-effect that WATER-VEHICLE has the following datum added to its SELF's $MATCHES slot

(car322 -MATCH -oo nil fm-basic)

and CAR322 would have

(water-vehicle -MATCH -oo nil fm-basic)

added to its SELF's $MATCHED-BY facet. fm-recall-result's job is trivial, it simply checks the $MATCHES facet of the SELF slot of the pattern concept to see if the result is already there. Only the values which are local to particular pattern concept are examined, i.e. no inherited values for the facet are considered. This is because the inherited values will not, in general, apply to the result of the particular match in question.

The semantics of the inheritance of match results can be specified, however, so that useful information could be obtained in many cases. Figure 5.2 contains a table which shows which match results can and can not be inherited. In the table "x" stands for a pattern concept and "y" for a target concept. An ancestor of frame "f" is represented by A(f) and a descendant by D(f), respectively. The table can be read according to the following plan. If the matcher is applied to the frames in the first two columns, then column three gives the result if X matched Y and column four gives the result if X mismatched Y. This information has not been incorporated into the matcher in its present implementation, however.

In order to prevent a buildup of these remembered match results, they are also organized into a short-term memory. Each time a result is attached to the two concepts participating in the match it is also entered into a short-term memory queue. If the queue is full, the oldest result is removed and "forgotten" by detaching it from the pattern and object frames.(26)

The method fm-assume-goal does as its name suggests. In the course of

26 The management of this short-term memory is very similar to the paging problem in virtual memory systems. Better algorithms for deciding what to forget are, of course, available.

		(fmatch pattern target)	
pattern	target	+match	-match
x	y	+match	-match
x	A(y)	?match	-match
x	D(y)	+match	?match
A(x)	y	+match	?match
A(x)	A(y)	?match	?match
A(x)	D(y)	+match	?match
D(x)	y	?match	-match
D(x)	A(y)	?match	-match
D(x)	D(y)	?match	?match

The Heritability of Match Results

figure 5.2

attempting to establish a match between two frames, subsequent attempts to match the same two will be assumed to succeed. The use of this method prevents certain kinds of infinite loops. When the recursive matcher is applied to a pattern PC and a target TC, the first thing it does is to push the list (PC TC) onto a dynamic stack fmatch-agenda. If the matcher is subsequently asked to match PC against TC, it will find the pair on the agenda and assume they match. This facility is necessary to handle matches involving concepts which are in some way self-referential. For example, consider the following match, which tries to identify a man who is married to his own mother. Suppose we have:

```
(a man44 is (a man) with
         mother matching (a woman with spouse = 'man44))
```

and the task is:

```
(fmatch (a man44) 'Oedipus)
```

To establish a match in this case it is necessary to establish a match between the spouse roles of MAN44 and OEDIPUS. This creates the subgoal:

```
(fmatch (a woman with spouse = 'man44) 'Jocasta)
```

which creates the subgoal of matching the spouse roles which leads to the subgoal of matching MAN44 against OEDIPUS. It is at this point that fm-assume-goal intervenes and returns a +MATCH, allowing the process to unwind successfully.(27)

The matching method fm-ako checks to see if the target concept is known to be a sub-concept or instantiation of the pattern concept. It does this by searching for a path between the pattern and target following only AKO links. If a path from the target to the pattern is found, then the target is a sub-concept of the pattern and a match is reported. If not, then a path from the pattern to the target is sought. If found, then a mismatch is reported. If neither path can be found, then a ?MATCH is reported.

When the first version of the matcher was written, it was saddening to watch it whirl away while attempting to match two totally unrelated concepts, like (a time)

27 The detection of some cycles may be delayed. Suppose we have the goal chain: (match a b) -> (match a' b') -> (match a c) and c is a direct descendant of b. The matcher will continue to try matching and eventually arrive again at the subgoal of (match a c).

and (an aircraft-engine). The method fm-basic was introduced to more quickly discover a mismatch between obviously disjoint concepts. The idea is loosely borrowed from the KRL system which, in turn, adapted it from the work of cognitive psychologists [ROSCH]. The idea is to identify certain concepts as partitioning the world into non-overlapping classes of things. In the KRL system, no individual could belong to more than one basic category. Thus a simple, efficient test available to a matcher was to find the basic concepts "nearest" to the pattern and target and compare them. If they were different, then there could be no match. If they were identical, then a match was possible. To find the basic category "nearest" to a concept one need only move up the abstraction hierarchy until a super-concept marked basic is found.

My use of this idea is somewhat different. Basic categories still partition the world, but a basic category is allowed to overlap another basic category if and only if it completely contains it or is completely contained by it. Thus if the concept (a person) is defined to be a basic category, the concepts (a man) and (a woman) can also be basic since they are both completely contained by (a person). The concept (a female-animal) could not be declared basic, however, since (a person) and (a female-animal) since there are female animals that are not people and people who are not female animals.

Every concept is either a basic category or it is a member of the basic categories to which its parents belong. The basic categories to which a concept belongs are stored as the values of the BASIC role. For efficiency reasons, I decided that a concept's basic categories would always be explicit. That is, they are always stored locally and never inherited. Thus retrieving the set of basic categories to which a concept belongs can be done in constant time. In order to do this, demon procedures were added to the basic role to maintain each concept's basic values whenever an addition or deletion is done somewhere in the hierarchy. The if added demon is:

> Whenever a value V is added to the basic role of concept C, then, [1] if V equals C then remove any other values from C's basic role; [2] Otherwise, if C is a basic category, then reject the value V; [3] Otherwise, add the value V to each of C's subconcepts.

The if removed demon is:

> Whenever a value V is removed from concept C's basic role, then, remove value V from all of C's sub-concepts. If V equals C (i.e. C is no longer to be considered a basic category) then the values of all of C's parents are added to C's basic role.

A final demon must be run whenever a concept C is instantiated. In the implemented system it is attached as an if-added demon to the generic instance role. It is:

> Whenever a value V is added to concept C's instance role, then add all of C's basic's values to the concept V.

The matching method fm-basic provides a quick test to identify matches which can not be made because the pattern and target concepts belong to incompatible basic categories. The match immediately fails unless one of the basic categories of the pattern equals or is a kind of one of the basic categories of the target. Thus, this method can only supply negative evidence for a match.

5.3.2 Fm-recurse is the most Powerful Matching Method

The most powerful of the predefined general matching methods is fm-recurse. Its basic approach is to pair up slots in the pattern frame with those in the target frame and then check to see if the contents of the paired slots match. The comparison of the contents of one slot with the contents of another is done by a procedure which is analogous to the frame matcher. Thus, the two steps are [1] slot alignment and [2] slot matching.

5.3.3 Aligning the Slots

Before the matcher begins to compare the slots of the pattern with those of the target, it tries to align them by finding a unique slot in the target to correspond to each slot in the pattern. In the current implementation, this is done by a simple procedure. Alignment has been made into an identifiable step in order to allow a place for more complex extensions. In the current matcher, the slots in the pattern are either pattern elements (i.e. atoms beginning with the character "?") or literal slot names. The pattern-element slots can match any slot in the target. The alignment procedure is to separate the slots of the pattern frame into two sets: the pattern slots and the literal slots. Each of the literal slots is paired with a slot in the target with the same name. If there isn't an identically named slot accessible from the target, then the alignment step fails. This, in turn, causes the entire match to fail. Once all of the literal slots have been paired with corresponding slots from the target, the pattern slots are aligned with the remaining target slots. This step is essentially:

```
(forall S1 in pattern-slots
        (forsome S2 in target-slots
                 (if (matchslots S1 S2)
                     (setq target-slots (delete S2 target-slots))
                     T)))
```

A future version of the matcher will be sensitive to the knowledge about the slots themselves. For example, the slots can be organized into an abstraction hierarchy. A slot X might then be allowed to be aligned with a slot Y if X is known to be a descendant of Y. More generally, slot X alligns with slot Y if the generic slot X matches the seneric slot Y.

5.3.4 Matching the Contents of Two Slots

The procedure for comparing the contents of two slots is similar to the one developed for matching two concept frames. The approach is to apply a set of slot matching methods until one of them reports that a definite match or mismatch has been established. If none of the appropriate methods is successful, a -MATCH is returned. As with the frame matcher, the methods used can come from several sources. There is a set of default slot matching methods which is available for any match. Specialist slot matching methods can be attached to a particular slot (in the $MATCHER facet of the slot) or can be place in the frame which describes the slot (in the $MATCHER slot). In applying the methods, those suggested by the particular slot are tried first, those suggested by the slot-describing frame next, and the default slot matching methods last.

5.3.5 Methods for Matching Slots

This section describes the set of predefined slot matching methods. The set currently consists of the methods fms-ignore-slot, fms-equal-slots, and fms-recurse which utilizes the additional methods fms-compare-values, fms-compare-requirements, and fms-existence.

The slot matching method fms-ignore-slot is the simplest. It ignores the contents of the slot and always reports a match. This method is used on slots which should not take part in the matching (INSTANCE and SELF for example).

The method fms-equal-slots checks to see if the slot from the pattern and the slot from the target are both inherited intact from a common ancestor. If so it returns a +MATCH. This is a common situation due to the high degree of inheritance through the abstraction hierarchy.(28)

fms-recurse is the most powerful slot matching method. It begins by matching the slot name from the pattern against the slot name from the target. If this succeeds, it compares the contents of the pattern slot to those of the target slot. The only facets which take part in the comparison are the $VALUE, $REQUIRE and $PREFER facets. This method utilizes the three sub-methods described in the following paragraphs.

The method fms-compare-values examines the contents of the $VALUE facets of the two slots. If the pattern slot has no values, then a ?MATCH is returned. A -MATCH is returned if there exists a value from the pattern slot for which there is no identical value in the target slot. A +MATCH is returned if, for every value in the pattern slot, there is an identical value in the target slot. Note that this means that the target slot may have extra values. Some examples are are shown in figure 5.3.

The method fms-compare-requirements handles situations in which the pattern slot has no values but does have one or more requirements. If the pattern slot has no requirements, a ?MATCH result is returned. If there are requirements, then these are compared to the values and requirements of the target slot. If the target slot has values, then there is a match if and only if each value of the target slot satisfies all of the requirements of the pattern. If the target does not contain values but does have requirements, then the requirements of the two slots are compared. There is a match if and only if each of the target's requirements is subsumed by one of the pattern's requirements. There is a mismatch if one of the target's requirements is not subsumed by one of the pattern's requirements. If the

28 Ideally, one would collect just the information that is not shared between the two concepts. If the conceptual space is really a tree then this is easy - just gather the information between each concept and their common ancestor. When we allow the concept space to become a non-cyclic connected graph (i.e. a concept can have multiple immediate ancestors), then determining the differences is slightly more complicated.

Pattern & Target slots	result
P: agent = ´person324 T: agent = ´person324	(+MATCH 1 nil fms-compare-value)
P: agent = ´person324 T: agent = ´man32	(-MATCH 1 NIL fms-compare-value)
P: object = ´car122 T: object = ´car345 and ´car122	(+MATCH 1 NIL fms-compare-value)
P: object matching (a vehicle) T: object = (a plane)	(+MATCH 1 NIL fms-compare-requirements)
P: object matching (a location) T: object matching (a city)	(+MATCH 1 NIL fms-compare-requirements)

Examples of Matching Slots

figure 5.3

target has neither values nor requirements, then it is assumed that a weak match exists and a +MATCH result is returned. See figure 5.3 for examples.

If the pattern slot has neither values nor requirements but does have preferences, then the preferences are compared to the values, requirements or preferences of the target slot. Failure to match does not prohibit the match, but does decrease its score.

The method fms-existence recognizes the situation in which the pattern slot has no facets, i.e. it is only mentioned. This method returns a +MATCH if the slot exists in the target frame and a -MATCH if the slot does not exist.

5.3.6 Pattern Elements and Variable Bindings

Pattern atoms are atoms which begin with the ? character. They provide a primitive way to extract portions of the target frame. A pattern atom can appear in place of a slot name or a value. If the match is successful, the name of the pattern atom is associated with the term in the target with which it corresponds. The set of all these associations is returned as the third term of the match result. It has the structure of an association list, i.e. a list of pairs of variable name and associated value.

In addition, there is also a way to attach an arbitrary condition to a pattern variable. This condition must evaluate to a non-NIL expression in order for the match to succeed. In the following example, the ?AGE term has the condition (< ?AGE 30) attached to it. example,

```
(fmatch (a person with
           spouse
           age = ($r ?age (< ?age 30))
           address matching (an address with state = illinois))
         'tim )
```

5.4 Examples

Appendix C shows some extended examples of the matcher in operation.

6 MODIFIER PARSING

This section describes the algorithm used to "parse" strings of nominal modifiers. Given a string of more than two nouns, there are a large number of possible structures indicating which nouns modify which other nouns. This algorithm attempts to identify the most likely structure based on the likelihood of its two-concept constituent compounds.

6.1 Introduction

The principal data structure used by the algorithm is a string of constituents. Initially, this string is made up of the concepts to which the nouns refer. The goal is to "reduce" this string to a single constituent. The only operation is to decide that one of the constituents of the string modifies the one to its right and then to replace the pair by the result of the modification. In considering how well one constituent can be interpreted to modify its right neighbor, the algorithm only looks at the constituents in a small window. The current algorithm, in fact, uses a window of only three constituents.

This style of parsing was suggested by similar windowing algorithms used in a natural language parser developed by Mitch Marcus [MARCUS]. Marcus makes the strong claim that any natural language can be parsed by using an algorithm which utilizes a buffer that can hold a fixed number of constituents. Words enter the buffer from the right as complete or partial syntactic nodes are removed. Constituents may only leave the buffer by being attached to the developing parse tree. This constraint implies that once nodes leave the buffer they must be used in the final parse and, moreover, we must know exactly how and where they will be used. Another constraint is that the grammar rules apply only to the constituents in the buffer. The experimental parser that Marcus developed for English uses a three constituent buffer for everything but noun phrases, which were found to require a four constituent buffer.

The windowing algorithm used in this system is more powerful than the one used by Marcus. The constraint that constituents may only leave the buffer by being attached to the developing parse tree is removed. As a consequence, words or constituents can be removed from the window and later pulled back in. What I have done, in effect, is to split Marcus' constraint into two parts and retained only one.

In Marcus' parser the buffer serves two functions. First, it limits the number of constituents that one can hold without knowing their ultimate function. Second, it restricts the view of the grammar rules, i.e. the rules may only examine constituents contained in the buffer. In my windowing algorithm the first constraint has been relaxed but the second is still in force.

An Alternative Description

Another way to describe this algorithm is in terms of a buffer and two stacks. An _input stack_ holds words that have not yet been examined, the fixed size _buffer_ holds those constituents under direct examination, and a _temporary stack_ holds those constituents which we do not yet know how to process. When two constituents in the buffer are joined, a new constituent is added to the buffer by popping the temporary stack, if it is not empty, or the input stack, if it is.

6.2 Parsing with an N-constituent Window

This section describes the general algorithm using a window that can hold _n_ constituents. The next section will discuss the algorithm using a three-constituent window. In brief, for an n-constituent window, the algorithm computes a semantic interpretation of each pair of adjacent constituents (i.e. the first constituent modifying the second, the second modifying the third, etc). Each interpretation has an associated score, represented by an integer. A negative score indicates a non-interpretation, i.e. a rule was applied but failed to produce a viable interpretation.

If at least one interpretation is acceptable (i.e. its semantic interpretation score is above a minimum threshold) then the interpretation with the highest score is taken. If there is more than one interpretation with the highest score, then the leftmost one is selected. The two constituents involved are then removed from the string and the result of the interpretation put in their place, shortening the string by one element. The window is then readjusted to encompass three constituents by bringing in the next element to the left, if possible. If there are no left-lying elements, than one is brought in from the right. If there are none to the right, then a null element is added.

If none of the pairs forms an acceptable interpretation, the window is shifted one place to the right. If there are no more items to the right, then processing resumes with the threshold lowered to zero. This allows interpretations which had earlier been considered unacceptable to be used. A default interpretation, with a score of zero, involves the undifferentiated MODIFIER relationship. That is, if no other interpretation can be made at this stage, the modifier is attached to the modified concept via the MODIFIED relation..ss The Three-position Window Algorithm

The algorithm with a three-position window is described here in more detail. Assume that the three constituents in the window are C1, C2 and C3 and that there are others to the left of the window (l1, l2, ...) and to the right of the window (r1, r2, ...). When indexing the left and right lying terms, we count from the window outward, e.g.:

```
                      --------------
#  ...  l3  l2  l1   | C1   C2   C3 |   r1   r2   r3   ...  #
                      --------------
```

The # symbols are used to mark the ends of the string.

In the initial state, the window is empty. There are no items to the left and all of the words in the compound are to the right. The lexical interpreter is called three times to take the next word in the input stream, map it into the appropriate concept and place it in the window. The state is then:

```
       --------------
  #   | C1   C2   C3 |   r1   r2   r3   ...  #
       --------------
```

Of course, if there are only two words in the compound, they fill the first two slots in the window and the third remains empty.

The algorithm proceeds by computing the semantic interpretation of C1 modifying C2. If the score of the resulting interpretation is very bad, i.e. less than a threshold BAD, then this interpretation is immediately rejected. This causes C1 to be shifted to the left, leaving the window. C2 and C3 move left, as well, opening a slot for a new constituent. If there are any more words in the input stream, the lexical interpreter is invoked to interpret the next word and place it in the C3 position.

If the score is very good, i.e. greater than a threshold BEST, then this interpretation is immediately accepted. Concept C2 is replaced by the interpretation of C1 modifying C2 and C1 is removed. This opens up a slot at the left end of the window. If there are any constituents to the left, then l1 is brought into the window and placed in the C1 slot. If there are no left-lying constituents, then C2 and C3 shift to the left and the C3 slot filled from the input stream, if possible.

If neither of these cases are true, i.e. the score is between BAD and BEST, then we consider the interpretation of C2 modifying C3. If the resulting score of this interpretation is greater than that of C1 modifying C2, then it is taken. Otherwise, we accept the interpretation of C1 modifying C2. In each case, the constituents involved are replaced by the corresponding interpretation and the window adjusted by bringing in a constituent from the left, if there is one, or from the right through the invocation of the lexical interpreter.

Figure 6.1 summarizes these cases.

```
                             --------------
# ...  l3  l2 l3     | C1  C2  C3 |  r1  r2  r3  ...  #
                             --------------
```

(a) Initial State

```
                             --------------
# ...  l2  l1  C1    | C2  C1  r1 |  r2  r3  r4  ...  #
                             --------------
```

(b) If C1+C2 is very bad

```
                             ----------------
# ...  l4  l3  l2    | l1  C1+C2  C3 |  r1  r2  r3  ...  #
                             ----------------
```

(c) If C1+C2 is very good or
prefered to C2+C1

```
                             ----------------
# ...  l4  l3  l1    | l1  C1  C2+C3 |  r1  r2  r3  ...  #
                             ----------------
```

(d) If C2+C3 is prefered over C1+C2

The 3-Slot Window

figure 6.1

7 CONCEPTUAL MODIFICATION

This chapter describes "Concept Modification", the process of building a semantic interpretation of one concept modifying another. Our task, in this chapter, is to explore the process which begins with two concepts and a set of interpretation rules and produces a set of likely interpretations.

The first section contains a general discussion of two possible approaches to the problem. Section two introduces the particulars of this rule-based interpretation system. The third section describes the rule application process. The fourth section describes three classes of rules and gives some example of the first two. The fifth section presents the rules of the third and most important class. The sixth and final section discusses the process of finding good relationships between concepts.

7.1 Approaches

There are two fundamental approaches to the problem of deciding what relationship was intended when one nominal concept modifies another. The first is concept independent and assumes that the relation is one chosen from a small set of potential relations (lets call this set the NNR for Noun-Noun Relations). This set is fixed and does not vary with the modified or modifying concept. The NNR set can thus be used to produce a list of candidate interpretations, one for each relation in the set. The problem of forming an interpretation for the modified concept is thus reduced to one of deciding which of the possibilities is most appropriate.

The second fundamental approach is concept dependent. In this approach the two concepts are examined to discover the kinds of relationships in which they can or prefer to partake. This set presumably includes all the candidate relations which might have been intended by the utterer. The composition of this set is a function of the two concepts. Once it has been computed, the problem of selecting the most appropriate relation remains.

In an attempt to reconcile these two approaches we can view the conceptual modification problem as having two parts: (1) finding candidate relations and (2) selecting the most appropriate candidate. This is, of course, the classic weak method Generate and Test [SIMON].

In the concept independent approach, the first step is trivialized. The set of candidate relations between the two concepts is just the fixed NNR set. In order to supply adequate semantic descriptions, the NNR set must either be very large or consist of very general relations. In either case the real work is passed on to the second step, selecting (or refining) the most appropriate relationship.

The concept dependent approach can be seen to be the more general one. The balance between processing in the generation step and the selection step is not inherent in the general strategy but is governed by the representational system. Finding the candidate relations becomes an important and interesting component of the problem. If the generating procedure is too liberal, the candidate set will be too large. Too conservative a generator will lead to a lack of semantic closure.

The approach that we have taken in this work is concept dependent. The key idea has been to represent concepts in a way that allows one to discover the kinds of relationships in which they partake. Moreover, the representation should also enable one to know which relationships are characteristic and/or preferred. The sections to follow in this chapter will describe the details of the system that we have designed.

7.2 Rule Based Interpretation

The basic mechanism of semantic interpretation used in this thesis is driven by interpretation rules. We are given an object (involving the modification of one nominal concept by another) to interpret and a set of interpretation rules. We collect rules which are applicable to the given object and apply them. Each rule application results in one or more interpretations.

These rules are, themselves, represented by frames and organized into the abstraction hierarchy. The use of frames to represent the interpretation rules has had several benefits.

First, this has facilitated experimentation with the kinds of knowledge that goes into a rule. The addition or deletion of attributes of rules has been done simply by adding of removing slots from the generic rule frame. Moreover, the information stored in the rule frames can be easily augmented with ancillary information, such as the contexts in which the information is important.

Second, this representation has allowed the system to treat the rules as formal objects which can be the object of inference and manipulation. This has facilitated the writing of meta-level rules, such as the rules for sets described in an earlier section of this thesis.

Finally, organizing the rules into an abstraction hierarchy has aided the recognition of regularities. It has also provided one way to restrict the application of a rule if a more specific rule is found to apply. If two rules both apply to a given object and one is a descendant of the the other, then we apply the more specific.

7.2.1 The Generic Rule Frame

Figure 7.1 shows the FFRL frame for the generic rule. Its principal roles are described in the following paragraphs.

The PATTERN role holds a pattern which describes the object to be interpreted It is the rule´s "left hand side". In our case, this node should be an instance of (a NominalCompound). A typical pattern might be:

```
(a NominalCompound with
   Modifier matching (a ChemicalCompound)
   Modified matching (a Liquid)
            preferably (a Water))
```

If the pattern matches the object to be interpreted, then the rule is a candidate. This example shows a preference facet on the Modified role. This sort of information can be used to weigh how applicable the rule is with respect to a given object to be interpreted. The current system, however, pays attention only to the role´s requirements (e.g. the matching facets). That is, if the object matches the pattern at all, then the rule is a candidate for application.

Expressing a preference does, however, have a positive effect on the efficiency of the interpretation process. This is due to the algorithm used to index and retrieve the rules. Each rule, when it is created, is linked to all of the concepts it references. When retrieving rules to interpret a particular object, those

```
(an sirule is (a meta-thing) with

    ;
    ; the generic frame for a Semantic Interpretation rule.
    ;

    basic = *

    name defaulting-to ´%(fname :frame)
         multiplicity =1

    pattern if-added ´(index-sirule :frame)
            modality ´obligatory
            multiplicity =1
            if-needed ´(break |no pattern for this rule|)
            matching (a Lisp-list)

    lambda-variables matching (a Tuple)
                     modality ´optional

    test = t
         modality = ´optional
         multiplicity ´(0)
         defaulting-to t
         matching (a S-expression)

    action matching (an S-expression)
           modality ´obligatory
           multiplicity ´(0)

    score matching (an integer)
          defaulting-to 2

    instance if-added ´index-sirule -> ´(finherit: continue)
             if-removed ´unindex-sirule -> ´(finherit: continue))
```

The Generic Rule Frame

figure 7.1

attached to semantically close concepts are found before those attached to more general concepts. Thus, if the object is:

```
(A NominalCompound with
   Modifier = (A Salt)
   Modifier = (a Water))
```

the example rule will be found earlier because of its expressed preference for a modifier matching (a water). See section 3 on page 106 for the details of rule indexing and retrieval.

The LAMBDA-VARIABLES role of the generic rule frame is used to create and initialize local variables for use by expressions in the TEST and ACTIONS roles. Thus it defines an evaluation environment for the rule. Each atomic value of this slot is taken to be the name of a local variable, bound and given an initial value of NIL. A non-atomic value should be a tuple whose first element provides the variable name and whos second, when evaluated, provides an initial value. An example is shown in the following rule fragment

```
(an sirule with

    pattern = (a NominalCompound with
                 modifier matching (a ChemicalCompound)
                 modified matching (a Liquid))

    lambda-variables = (CC (the modifier of TheObject)) &
                       (L (the modified of TheObject))

    ...)
```

In the context of the LAMBDA-VARIABLES slot, the atom Theobject is bound to the expression that we are trying to interpret. Thus, if this rule is applied to the compound:

```
(a NominalCompound with modifier = (a Salt)
                        modified = (a Water))
```

then the atoms CC and L will be bound to (a Salt) and (a Water), respectively.

The TEST role is used to further constrain the situations in which the rule can be used. It holds one or more s-expressions which all must evaluate to non-NIL. If any of the expressions evaluates to NIL then the rule is not applied. This role can be used to include restrictions which are difficult or even impossible to express in the rule's PATTERN. For example, an expression might be included to restrict the rule to particular discourse contexts.

The ACTIONS role specifies the rule's "right hand side". It contains one or more s-expressions which are evaluated. The value returned by the last expression should be either NIL (in case the rule failed totally) or an instance of an interpretation frame or an instance of an interpretation-set.

Note that the INSTANCE slot has both an _if-added_ and an _if-removed_ facet. The procedures in these facets are responsible for maintaining an index of semantic interpretation rules. The _if-added_ demon is run whenever a new instance of a semantic interpretation rule is created,(29) and the _if-removed_ demon whenever a rule is erased (i.e. removed from the database). The _if-added_ demon creates links between the new rule and all of the frames that are referenced in its pattern.(30) Thus, for the example, a link will tie this rule to the frames for NominalCompound, ChemicalCompound and Liquid. See section 3 on page 106 for a detailed discussion of rule indexing.

The SCORE role is used to provide a default score for the resulting interpretation. If this role contains a value, it is evaluated and used to fill the SCORE role in the current interpretation frame. If the role is empty, then it is the responsibility of the rule's actions to see that the current interpretation is assigned a score. The NAME role holds an expression which acts as the rule's name. Finally, note that the * value in the BASIC slot indicates that an Sirule is a basic category.

7.2.2 _The Generic Interpretation Frame_

The application of an interpretation rule should result in a set of interpretations. The set may be a singleton set. The generic frames for an interpretation and an interpretation-set are shown in figures 7.2 and 7.3, respectively.

29 Note that creating an instance of an sirule will cause the name of the newly created instance to be automatically added to the instance slot of the generic sirule frame. Similarly, the erasure of a rule will automatically cause its name to be removed as an instance of the generic sirule. In this way one can simulate _when-instantiated_ and _when-destroyed_ demons.

30 To find all frames referenced from another, one collects all of the data in all of the frame's slots and removes anything that is not a frame name.

```
(an interpretation is (a meta-thing) with

    ;
    ; the generic frame for a semantic interpretation.
    ;

    basic = *

    interpretation matching (a thing)
                   modality ´obligatory
                   multiplicity =1
                   if-added ´add-inverse-link

    object matching (a meta-thing)
           modality ´obligatory

    score multiplicity <2
          defautling-to 2
          if-added ´f-remove-old-value-demon
          matching (an integer)

    interpretation-set matching (an interpretation-set)
                       if-needed ´(an interpretation-set with
                                      members = :frame)
                       if-added ´add-inverse-link &
                                ´f-remove-old-value-demon
                       multiplicity <2

    modifier if-added ´f-copy-value-demon
             matching (a thing)

    modified if-added ´f-copy-value-demon
             matching (a thing)

    source preferably (an sirule))
```

The Generic Frame for an Interpretation

figure 7.2

```
(an interpretation-set is (a set) with

    ;
    ; the generic frame for a set of interpretations.
    ;

    basic = *

    object matching (a meta-thing)
           if-added ´add-inverse-link

    typical-member = (an interpretation)

    members matching (an interpretation)
            multiplicity ´(0)
            if-added ´add-inverse-link &
                     ´f-add-new-interpretation

    RuleGenerators if-needed ´f-get-rule-generators

    RulesTried matching (an Sirule)

    best matching (an interpretation)
         modality ´derived
         multiplicity ´(0)

    bestscore = *-infinity*
              matching (an integer)
              defaulting-to *-infinity*
              modality ´derived
              multiplicity =1
              if-added ´f-superlative-interpretation-demon

    modifier matching (a thing)
             modality ´derived

    modified matching (a thing)
             modality ´derived )
```

The Generic Frame for a Set of Interpretations

figure 7.3

Why do we have a separate concept for an interpretation and, worse yet, for a set of interpretations? We argue that these concepts are necessary because our system needs to be able to reason about a concept _as an interpretation of an instance of another_ in a manner distinct from reasoning about it _as an instance of another_. For example, suppose we have a situation where we have the concepts for "engine", "repair" and the compound "engine+repair". We also need a concept for "engine+repair" being an interpretation of "engine" modifying "repair". In practical terms, having a frame for an interpretation allows us to do the following important things:

[1] To associate the interpretation of a compound (or any other form) with the interpretations of its constituents. The compound may not, in general, refer to the constituents at all. This is done through the use of the OBJECT, MODIFIER and MODIFIED roles of the interpretation frame.

[2] To associate an interpretation with a set of competing or alternative interpretations. This link is made through the INTERPRETATION-SET slot.

[3] To associate a measure of how strong or appropriate this interpretation is. The SCORE slot holds an integer which represents the strength.

[4] To link the interpretation with the interpretation rule which produced it. The SOURCE slot holds the name of the interpretation rule which produced the interpretation.

Having a frame for a set of interpretations facilitates the management of alternative interpretations. Opportunities for alternative interpretations abound. A rule may generate several interpretations for a given compound. Many interpretation rules may apply and yield interpretations. Finally, different syntactic structures may be possible when one is dealing with a sequence of more than two nouns.

An instance of an interpretation-set has a place to hold all of the alternatives (the MEMBERS slot), a description of the typical member (the TYPICAL-MEMBER slot), one or more distinguished members who are currently favored (in the BEST slot) and the strength measure for these favored interpretations (the BEST-SCORE slot).(31) In addition, there are slots to hold the object being interpreted,

31 The contents of the best and bestscore slots are maintained by an if-added demon on the members slot. Whenever a new member is added, its score is compared to the set's bestscore. If it is greater, then the new interpretation and its score fill the best and bestscore slots, respectively. If the new score is equal, then the new interpretation is added to the best slot.

and the modifying and modified concepts. Two slots of the interpretation set frame are used to store the state of an ongoing interpretation. The RULESTRIED and the RULEGENERATORS slots contain a list of the interpretation rules that have been tried and functions capable of producing the rest of the untried rules, respectively. Having this information available through the frame allows one to generate some of the possible interpretations of an object, suspend the process, and later resume. This capability is not used in the current system, however.

7.3 The Rule Application Process

With these preliminary descriptions past us, we are in a position to explore the process by which rules are applied to a compound to yield a semantic interpretation. Recall that, at this point, we are only concerned with computing an interpretation of one nominal concept modifying another. Thus, we start with a frame which represents this modification and want to end up with another frame which represents the resulting interpretation. The procedure, which we will soon see in some detail, is to collect the interpretation rules suggested by the two concepts involved, filter out the inappropriate ones, and then apply them to the compound. Each application adds new interpretation candidates to a growing set. We stop when we find an interpretation with a very high score or when we run out of rules to apply.

7.3.1 Preliminaries

Before we begin this process, there are several preliminary steps. First, we create a new instantiation of an interpretation-set to hold the candidate interpretations. We place in its roles the frame for the compound to be interpreted, the frame for the modifying concept, and the frame for the modified concept. We also create an instantiation of an individual interpretation which will act as the typical member of the set. This frame also has its OBJECT, MODIFIER and MODIFIED roles filled. As each rule wants to create a new interpretation, it will instantiate a copy of this typical member frame. Thus, it represents the generic concept for an interpretation of <u>this</u> compound.

Let's use the interpretation of the compound "engine repair" as an example.

After lexical interpretation and conceptual parsing, our task is to find an interpretation for the concept:

```
        (a NominalCompound with Modifier = (an engine)
                                Modified = (a to-repair))
```

Assume that this concept gets the name "NominalCompound21". The interpretation set we will create will be:

```
        (an interpretation-set with
            typical-member = (an interpretation with
                                 member-of = (this interpretation-set)
                               object = ´NominalCompound21
                               modifier = (an engine)
                               modified = (a to-repair))
            object = ´NominalCompound21
            modifier = (an engine)
            modified = (a to-repair))
```

We will also assume that the prototypical interpretation´s name is "interpretation22" (i.e. the set´s typical member).

The second major preliminary step is to create data generators which will find interpretation rules that claim to apply to the current compound. A data generator is created for each frame that is referenced in the compound. For our example, we will have three generators, starting at the concepts "NominalCompound", "engine" and "to-repair". Each generator, starting with its initial frame, will move up the concept hierarchy, reporting back the rules that are attached to the concepts it passes. Before reporting a rule, however, the generator applies a two-part test to ensure its applicability. First, the rule´s pattern must match the actual node being interpreted. This is a simple call to the pattern matcher. Second, the rule is compared to the set of rules that have already been applied to the node. If this rule, or a more specific version of it, (32) has already been tried, then it is rejected.(33)

Once these steps are completed, we enter into a simple loop in which we ask a generator for a new rule and then apply the rule to the object. Each rule adds any interpretations it produces as new members of the interpretation set. There are two

32 Recall that the rules are hierarchically organized.

33 Recall, too, that when a rule is instantiated, a demon is invoked which links the rule to each of its constituent concepts. It is just these links that the generators are looking for.

ways to exit this loop. The first is for all of the generators to become "exhausted". The second is for some rule to add an interpretation whose score exceeds a threshold for "superlative" interpretations. This second condition is detected by an if-added demon on the BESTSCORE facet of the interpretation set frame. If the newly added value for the best interpretation exceeds the threshold, control is returned to the process which invoked the conceptual modification procedure.(34)

In either case, once this process has been terminated, we are left with an interpretation set which contains one or more favored interpretations as well as collection of less favored alternatives. The next section takes a detailed look at the application of an individual rule.

7.3.2 The Application of One Rule

The rule applier needs only two things - the frames for the interpretation set and for the rule to try. From the interpretation set it can extract the object to be interpreted and all of the other relevant information. The steps it takes are as follows.

[1] An environment is created in which the atom THEOBJECT is bound to the object of interpretation.

[2] The local variables found in the rule's LAMBDA-VARIABLES slot are added to the environment and bound to their initial values.

[3] The expressions found in the TEST slot of the rule are evaluated. If any of these tests yield NIL, then the application is aborted and control is returned to the Concept Modifier (which will look for additional rules to try).

[4] A new copy of the interpretation set's TYPICAL-MEMBER is instantiated. This copy will represent this rule's interpretation. The SOURCE role is filled with the rule's name.

[5] The rule's ACTION expressions are evaluated. They are responsible for filling the INTERPRETATION and SCORE roles of the interpretation frame.

[6] If the the SCORE has been filled with a non-negative integer (indicating that the actions successfully computed an interpretation), the interpretation frame is added to the set's MEMBERS role. This will trigger an if-added demon which will update the set's BESTSCORE and BEST roles if necessary. If the SCORE contains a negative integer, the ACTIONS have failed to produce a viable interpretation and the interpretation frame is erased.

[7] Finally, control is returned to the Concept Modifier.

34 This is done via the MacLisp THROW facility for non-local goto's (see [MOON]).

7.3.3 Rule Indexing and Retrieval

Rules are indexed by the concepts that appear in their patterns. A link is made between the rule and each frame that is a datum in one of the slots of the pattern. The links are stored in the $RULES facet of the SELF slot along with the name of the slot of the pattern that points to the frame. An example will make this clearer. Consider the interpretation rule fragment:

```
(a SIRULE303 is (an sirule) with
   pattern = (a NominalCompound with
                Modifier = (a time)
                Modified = (an event))
   ... )
```

This rule, SIRULE303, will be indexed under the concepts NominalCompound, Time and Event. The SELF slot of the Time concept might look like:

```
(a TIME is ...
   (SELF ... ($RULES ...(Modifier SIRULE303 ...other rules...) ..) ..)
)
```

Note that a datum in the $ROLES slot is a list whose first element is the name of a slot in a rule pattern and whose other elements are the names of the rules which point to this frame through that slot. Thus, the "modifiers" entry in the example $RULES facet lists all the rules which involve a Time as a modifier.

When a pattern is indexed, its AKO slot is treated just like any other. This means that the example pattern would be indexed under the NominalCompound frame.

To retrieve the set of rules which may apply to a given object, we work the process in reverse. For each slot S of the object and each frame F which is referenced in that slot, one searches the abstraction hierarchy from F to the root. At each concept along the way, the $RULES facet of the SELF slot is checked for a datum begining with the slot S. If one is found, the rules listed in that datum are added to the candidate set.

The actual procedure uses a data generator to retrieve the rules in a stream. It also tests the candidates as they are found to ensure that the rule's patterns actually match the object in question.

7.3.4 Recursive Rule Application

Some of the very general rules which you will later see involve a recursive call to the Concept Modifier. This brings up the danger of infinite (or at least very large) regression in an attempt to explore possible interpretations. In general, interpretations which involve many levels of recursions are very unlikely to have been intended by the utterer. Language is a means of communication and a cooperative process. Much of its structure and use is governed by the principle that the utterances are intentional forms which have been designed to be easy to interpret by the hearer.

To limit arbitrarily long recursions, a simple threshold mechanism is enforced. No recursive calls with depth greater than a fixed threshold are allowed. The current threshold is three levels. In some situations one may want to vary this threshold, depending on some features of the context or dialogue (e.g when reading a difficult philosophical treatise?) but, for this work, the threshold has been fixed at three levels.

7.4 Rule Classes

Several general classes of interpretation rules have been used for the interpretation of nominal compounds. Although there is some overlap between these classes, I believe it is fruitful to think of them independently.

The first class we refer to as idiomatic rules. These rules are characterized by the fact that the relationships they create are totally dependent on the identity of their constituents and on the rule. These rules will typically match surface lexical items directly. Often, the compounds will have an exocentric meaning. As an example, consider the Navy's term for a plane with a very poor maintenance record - a "hanger queen". A rule to interpret this phrase would have a pattern which require an exact match to the words "hanger" and "queen".

The second class consists of productive rules. These rules attempt to capture forms of modification which are productive in the sense of defining a general pattern which can produce many instantiations. The are characterized by the semantic relationships they create between the modifying and modified concepts.

That is, the nature of the relationship is a property of the rule and not the constituent concepts. The nature of the concepts only determines whether or not the rule applies and, perhaps, how strong the resulting interpretation is.

The third class are the <u>structural rules</u>. These are rules which can be characterized by the structural relationships they create between the modifying and modified concepts. The semantic nature of the relationship that a structural rule creates is a function of the concepts involved in the modification. Many of these rules are particularly useful for analyzing compounds which contained nominalized verbs.

It is this last class, the structural rules, which have been the most important in this work. They are not capable of capturing all the phenomena, but they cover much of the ground. For this reason, I will put off their discussion until the following section. For the remainder of this section, I will give an example of an idiomatic and a productive rule.

7.4.1 <u>An Idiomatic Rule</u>

The following rule interprets the phrase "nor hour" as an instanstance of the special concept NOR-Hour. NOR is the Navy's abreviation for the phrase "Not Operationally Ready". A "Nor Hour" refers the state of a piece of equipment being in the NOR state for one-hour.

```
(an sirule with
   pattern = (a NominalCompound with
                Modifier = 'NOR
                Modified = 'Hour)
   actions = '(a new Nor-Hour))
```

7.4.2 <u>A Productive Rule</u>

The following is an example of a productive rule which interprets compounds of <u>the form</u> "<u><plane></u> <u><integer></u>" <u>as</u> "<u><plane></u> <u>with</u> <u>Bureau Serial Number</u> <u><integer></u>".(35) The rule is:

35 The Navy assigns each plane a Bureau Serial Number or BUSER which acts as an identification handle

```
(a sirule with
   pattern = ´(a NominalCompound with
                 modifier matching (a 3m-plane)
                 modified matching (an integer))

   lambda-variables ´(Plane (the modifier of TheObject)) &
                    ´(Integer (the modified of TheObject))

   test ´(not (the buser of Plane))

   actions _(a new ,Plane with Buser = Integer)

   score 20.)
```

This rule does the following. If the object to be interpreted is a nominal compound in which the modifier matches a 3M-plane and the head concept is an integer, and the plane does not have a value for the Buser slot, then the interpretation is a new instance of the plane in which the Buser slot is filled with the number.

7.5 The Structural Interpretation Rules

This section describes the general structural rules for nominal compounding. The first six have been implemented and are in use. The last two are less central and have not yet been implemented. My current thoughts for these rules are described.

7.5.1 RULE: N1 fills one of N2´s roles

The first structural rule that I present is perhaps the most commonly used. This rule interprets the modifier concept as specifying or filling one of the roles of the modified concept. Some examples of compounds which can be successfully interpreted by this rule are:

compound	interpretation
engine repair	(a to-repair with object = (an engine))
January flight	(a to-fly with time = (a January))
F4 flight	(a to-fly with vehicle = (an F4))
engine housing	(a housing with superpart = (an engine))
magnesium wheel	(a wheel with raw-material = (a magnesium))

Note that when the compound fits the form "subject+verb" or "object+verb" this works very nicely. The applicability of this rule is not limited to compounds in which the modified concept is underlyingly verbal or event-related, however. The last two examples, involving a housing and a wheel, do not fit this form.

To apply this rule we must be able to answer two questions. First, which of N2's roles can N1 fill? Obviously some roles of the modified concept may be inappropriate for filling with the modifier. The concept for the to-repair event has many roles, such as an agent doing the repairing, an object being repaired, an instrument, a location, a time, etc. The concept for an engine is clearly an inappropriate filler for the agent and time roles, probably inappropriate as a filler for the location and instrument roles, and a highly appropriate filler for the object slot.

Secondly, given that we have found a set of roles that the modifier may fill, how do we select the best one? Moreover, is there a way to measure how well the modifier fits a role? Having such a figure of merit would allow one to rate the overall interpretation. The process of determining which roles of a concept another may fill and assigning scores to the alternatives is called <u>role fitting</u>. This is discussed in section 7.6.

The role fitting process will return a list of the roles of N2 that N1 can fill and, for each, the figure of merit for the fit. Each possibility in this list represents one possible interpretation. Not all of the possibilities are worthy of becoming an interpretation, however. The selection process used by this rule is a simple one. The maximum for all the scores is determined and, if it is greater than zero, all possibilities with this score are made into interpretations. Making a role fit into an interpretation simply involves making a new instantiation (i.e. copy) of N2, and filling the appropriate role with N1. The score for this interpretation is just the score for the role fit. Each interpretation is created and added to the current interpretation set.

If none of the possibilities has a strictly positive score, this rule application is aborted and no new interpretations are added to the current interpretation set. One heuristic which is not currently used, but is undoubtedly valid, deals with the situation in which there are many equal or nearly equal possibilities all with a low or mediocre score. In such a case it would be wise to lower all of their scores. The justification for this heuristic is that, if the hearer had really intended one of these mediocre interpretations, he would have found a way to express it in a less ambiguous manner.

7.5.2 RULE: N2 fills one of N1's roles

This rule is similar to the first, except that the concepts change places. In interpretations produced by this rule, the modified concept is seen as filling a role in the modifier concept. Note that the object referred to by the compound is still an instance of the modified concept. Some examples where this rule yields the most appropriate interpretation are:

compound	interpretation
drinking water	(a new water is (an object of (a to-drink with object = (this water))))
washing machine	(a new machine is (an instrument of (a to-wash with instrument = (this machine))))
maintenance crew	(an crew which is (an agent of (a to-maintain with agent = (this crew))))

Again, the application of this rule is mediated by the role fitting process described in section 7.6. The results of the role fitting process are handled in a way identical to the previous rule.

7.5.3 RULE: Thing + Role Nominal

This rule is applicable when the modified concept is in the class we call role nominals. In section 2.2.2 we defined a role nominal to be a noun that refers to a role of an underlying concept. Some examples were that owner refers to the agent of an ownership concept and pilot refers to the agent role of a to-fly event. In that section, we identified a useful generalization so that a role nominal could refer to any role of an underlying verb. Here, we generalize again, and define a role nominal to be a concept that refers to a role of another concept of any type, not just a verbal one.

This semantic interpretation rule is invoked whenever any concept modifies a role nominal. It tries to find an interpretation of N1 modifying the underlying concept of which N2 is a role. For example, given "F4 Pilot", the rule notes that "pilot" is a role nominal refering to the agent role of the to-fly event and attempts to find an interpretation in which "F4" modifies a to-fly event. The result is something like "an F4 pilot is the agent of a to-fly event in which the vehicle is an F4". Some other examples are:

cat food	(an object of (a to-eat with agent = (a cat)))
elephant gun	(an instrument of (a to-shoot with object = (an elephant)))
oil pump	(an instrument of (a to-pump with object = (an oil)))
dog house	(a place of (a to-dwell with agent = (a dog)))

The role nominal could be viewed as serving the function of tying an object to a characteristic activity in which it participates. It is very much like a relative clause except that the characteristic or habitual nature of the relationship is emphasized.

7.5.4 RULE: Role Nominal + Thing

This rule is very similar to the previous one except that it applies when the modifier concept is a role nominal. The action is to attempt an interpretation in which N2 is modified by the underlying concept of which N1 is a role nominal. For example, given the compound "pilot school" we could derive the concept for "an organization that teaches people to fly". This is done by noting that pilot refers to the agent of a to-fly event and then trying to modify "school" by this "to-fly". This, in turn, can be interpreted by the "thing + role nominal" if school is defined as "an organization which is the agent of a to-teach" This leads to an attempt to interpret to-fly modifying to-teach. The "filler + concept" rule can interpret to-fly as filling the object (or discipline) role of to-teach.

Some other examples of compounds that benefit from this interpretation rule are:

newspaper glasses	glasses used to read a newspaper
driver education	teaching people to drive
food bowl	a bowl used to eat food out of

7.5.5 RULE: Specific + Generic

This rule handles cases in which the modifier (N1) and the modified concept (N2) can be anything. The condition for its applicability is that N1 be a subconcept of N2. As an example, consider "F4 planes". A paraphrase of compounds that fit this rule is "an N2 that is an N1". An interpretation of this form is simply the more specific of the two concepts, N1.

Compounds which fit this pattern are considerably rarer that those fitting the previous ones. One explanation is that such compounds are inherently redundant. Such redundancy is, of course, counter to conventional rules of conversation [GRICE]. There are contexts in which such forms are useful, however. A situation in which this form does not violate the conversational postulate of brevity through non-redundancy is one in which it is used for contrast. For example, we might have the following dialogue with PLANES or JETS:

```
> How many F4's flew in May 79?
42 F4'S FLEW DURING MAY 1979.
> How many planes flew in June.
107 AIRCRAFT FLEW IN JUNE 1979.
> No, I mean How many F4 aircraft flew in June.
```

This form seems to have the function of creating or bringing into the hearers mind the generic concept (the set of all aircraft), the specific concept (the set of F4's), and the contrasting concept for the difference between the generic and specific concepts (the set of all aircraft except F4's). Another function is emphasis, as in the phrases "girl child". Here the hearers attention is drawn to the sex of the child.

7.5.6 RULE: Generic + Specific

This is the flip side of the previous rule. Here the modifier is a super-concept of the modified concept. Some examples are:

the integer three
building NE43
president Carter
vehicle planes

Again, the more specific of the pair is relatively unchanged. This form is more common and can serve the function of placing a perspective on the modified concept. An interpretation of "president Carter" is "Carter viewed as a president". The attributes and knowledge associated with other perspectives is, presumably, to be suppressed.

Unfortunately, in my present representation system, there is no way to do this. The knowledge associated with a concept is the union of all possible perspectives. The representation language does not support a method to separate this knowledge. The current rule for this form simply returns an interpretation which is the head or modified concept intact.

7.5.7 RULE: N1 be N2

This rule, proposed here but not yet implemented, is intended to handle cases in which the relationship between the two concepts might be paraphrased as "be"; for example:

woman doctor	a woman who is a doctor

Note that this example could be handled by previous rules, but only if we assume a certain knowledge representation. For example, if the woman concept has or inherits an occupation role and if the doctor concept matches the requirements of this role, then doctor might be analyzed as filling this role. Another alternative is that the SEX attribute of the woman concept be transferred to the doctor concept, a process discussed in the following rule for attribute transfer.

This rule, however, provides another way to assign an interpretation to this and similar compounds. It involves a process in which an instance of one concept can be transformed into an instance of another. I call this process viewing. The process is mediated by a frame called a viewpoint frame, which specifies the transformation.

The Generic Viewpoint

Figure 7.4 displays the generic viewpoint frame. A viewpoint frame is intended to specify changes which must be made to a source frame to allow it to be viewed as an instance of a target frame. The SOURCEFRAME and TARGETFRAME roles contain patterns that must match the initial and final concepts, respectively. The LAMBDA-VARIABLES provides local variables in a fashion identical to the generic rule frame. Similarly, the TEST slot holds conditions which must met before the viewpoint frame can be applied. The ADD slot contains properties which are to be added to the source frame. The MAP slot contains three tuples which specify how the contents of certain slots are to be mapped from the target to the source. Finally, the ACTION slot contains expressions to evaluate to finish the transformation.

Figure 7.4 also contains two examples of individual viewpoint frames. The first specifies how to transform a FLONUM (floating point number) into an integer. It is analogous to a coercion rule. The second provides a viewpoint of a person as a woman.

```
(a viewpoint is (a meta-thing) with

   sourceframe matching (a thing)
               if-added 'add-inverse-link
               if-removed 'remove-inverse-link

   targetframe matching (a thing)
               if-added 'add-inverse-link
               if-removed 'remove-inverse-link

   lambda-variables matching (a Tuple)
                    multiplicity '(0)
                    modality 'optional

   test matching (a Lisp-expression)
        defaulting-to T
        multiplicity '(0)
        modality 'optional

   map matching (a List)
       multiplicity '(0)
       modality 'optional

   add matching (a tuple)
       modality 'optional
       multiplicity '(0)

  action matching (an S-expression)
         multiplicity '(0)
         modality 'optional )

(a vp:flonum->integer is (a viewpoint) with
    sourceframe = (a flonum)
    targetframe = (a integer)
    map '(value value fix) )

(a viewpoint with
   ;; view a woman as a person.
   sourceframe = (a person)
   targetframe = (a woman)
   test = '(eq NIL (the sex of SourceFrame))
   add '(ako (a woman)))
```

The Generic Viewpoint

figure 7.4

7.5.8 RULE: Attribute Transfer

This rule, proposed here but not yet implemented, would handle cases in which an attribute or property of the modifier is predicated of the modified concept. For example, this rule might be used in the interpretation of the following compounds:

iron will	a will that is strong
elephant legs	legs that are large

Its action is to look for a role in the modifying concept which meets certain criteria and which can be transferred to the modified concept. Initially, all of the modifier's roles can be considered as candidates and checked to see how well they fit the criteria. If a role matches the criteria well, then the role and its value are transferred to the modified concept.

Favorable characteristics that a candidate for transfer can have are:

[1] The role is part of the defining characteristics of the modifier.

[3] The role and its value are characteristic of the modifier.

[2] The role is marked as having a high degree of salience with respect to the modifier.

[3] The role has an extreme value (e.g. its domain is a scale and the value which is present is one which is or is near one of the ends of the scale).

[4] The modified concept can accept the role and its value.

[5] The modified concept's role expressed a preference for the value.

[6] The role is marked as having a high degree of salience with respect to the modified concept.

[7] The role does not already have a value in the modified concept.

Some of these criteria are very similar to those which play a part in the role fitting process, described in the next section. They are also similar to criteria proposed to handle the recognition and interpretation of metaphorical expressions (see [ORTONY], and [SEARLE]).

7.5.9 RULE: N1 Modifies N2

This rule handles cases where we want to force the modification of N2 by N1 but can find no other rule which can make an interpretation. It simply adds N2 to a special MODIFIERS slot of N1. Since its score is 0, the interpretation suggested by this rule will normally be rejected. In cases where the parsing procedure has concepts left over that it can not attach, it will lower the threshold of acceptable rules to 0 and be thus assured that some (possible semantically empty) interpretation will be made.

7.6 The Role Fitting Process

Role fitting is the process of trying to interpret one frame as being a filler or value for one of the roles of another. It is the basic process on which the structural interpretation rules discussed in sections 7.5.1 and 7.5.2 are founded. That is, given a frame X and another Y, what role of Y can best accept X as a value. This problem is similar to the general frame matching problem. The roles of Y, with their local and inherited characteristics, provide a kind of description against which the candidate value X must be compared. Since, in general, there may be several acceptable roles for which X can be a value, the process should provide some method of ranking them.

The process that I have developed to accomplish this role fitting task has four major steps. These are:

⌊1⌋ Collect the local and inherited slots of Y.

⌊2⌋ Filter out the obviously inappropriate ones (e.g. structural and meta-slots).

⌊3⌋ Compute a score for X as a filler for each of the remaining slots.

⌊4⌋ Select the best slots(s) remaining.

The following sections will describe these steps in detail.

7.6.1 Collecting the Slots

The first step in attempting to interpret X as filling one of Y's roles is to gather all of Y's slots. We start with a function which when given a frame, returns a list of all of its local slots. To get all of a frame's slots, one applies this function to the frame and then recurses on each of the frame's immediate ancestors. A set is then made of the results, eliminating any duplications.

For reasons of efficiency, the following modification was done. The first time the set of slots attached to a frame is computed, it is stored in a facet (named $SLOTS) of the frame's SELF slot. The next time one needs the frames slot set, it can be directly retrieved. Thus, the algorithm becomes:

> If there is a local value for the $slots facet of the frame's self slot, then return this. Otherwise, compute the frames slot set and store it as the contents of the frame's self's $slot facet.

All of this depends, of course, on the assumption that new slots are not being created dynamically. I believe that this is a reasonable assumption. If a frame can accept a slot, then that slot should be found on the frame or one of its ancestors. The creation of a totaly new slot is something which should occur infrequently. To allow for the occasional definition of a new slot for some concept, one can define demons which would be invoked whenever a slot is added or removed. These demons would perform the necessary bookkeeping functions to insure that the contents of any existing $slots facets were correct.

7.6.2 Filtering the Slots

Once a frame's slot set has been computed, the next step is to remove any obvious non-candidates for the role fitting. Slots which can be immediately eliminated from further consideration include those which are primarily structural in nature (e.g. AKO, INSTANCE, VIEWPOINT, etc.) and those which contain "meta" or "self" knowledge (e.g. CLASSIFICATION, SELF, etc.).(36) The surviving candidates, hereafter called the frame's roles, are stored in the $ROLES facet of the frame's SELF slot. Thus, subsequent needs for this frame's role set can be immediately filled by examination of the self slot.

7.6.3 Scoring a Fit

The third step is the most important. For each of the remaining roles, we must compute a score which rates the candidate value as a potential filler of the role. Lets assume that we want to see how well the value V fits into the role R of the frame F. Some characteristic of a good fit are:

[1] It should be possible for V to fill R, i.e. the role should be open to accept another value.

[2] The value V must match R's requirements. It's good if it matches R's description of preferred values. It's even better if it is a typical or default value for R.

[3] It is good if R is known to be a salient role of F.

36 This information can be found in the frame which describes the slot.

[4] It is better for R to be empty of values than to already have some values, even if it can accept more.

[5] It is bad if R already contains the value V or a value that matches V or is matched by V.

In the implemented scoring regime, we have chosen to represent the score of a value-role fit as an integer and to factor these criteria with respect to the facets found on a role. Each facet in the role may have an associated _facet scoring function_ whose job it is to compute a increment to the score for fitting the value V into the role R. For example, the facet scoring function for the $MODALITY facet compares the contents of this facet and the number of values already in the role. It returns a negative value if the role can accept no more values and a positive one if it can. The overall score for the fit is just the sum of the individual scores for the role's facets.

The names of the facet scoring functions associated with a particular facet are found in the frame which describes that facet. The default facet scoring function, used when a facet has no scoring function of its own, simply returns a neutral answer (a 0, in fact) to every case. Thus, facets which have no associated scoring functions are essentially ignored and a facet-less role accepts any value with a score of 0.

Computing a score for a role fit may be a computationally expensive task. It is desirable to be able to terminate the process as soon as it becomes apparent that the fit is impossible, e.g. when the candidate value fails one of the role's requirements or when the role is discovered to be full. To facilitate this, I have introduced a distinguished value which represents negative infinity. If any of the facet scoring functions contributes this value as an increment, the scoring executive assigns negative infinity to the overall score and returns immediately.

A more general scheme, which has not yet been implemented, uses two thresholds to control the executive behavior. The thresholds are the minimum possible value for a fit (MIN) and the maximum possible value (MAX). In the process of applying the facet scoring functions, if the current partial score is ever less than or equal to the MIN threshold, the process ceases and the overall score is assumed to be this lower threshold. Similarly, if the partial score ever reaches or exceeds the MAX threshold, the process is terminated and the MAX threshold used as the fit score.

This scheme has two advantages over the present one. First, the process is symmetric with respect to very bad and very good fits. Any individual facet scoring function can force the fit to take on the best possible or the worst possible value. Second, by making the process depend on the values of variables (i.e. the thresholds MIN and MAX), one can adjust the scoring process to the overall demands of the system. For example, when one suspects a metaphorical interpretation is necessary. one can shift the thresholds to allow interpretation which would otherwise be prohibited.

7.6.4 The Current Facet Scoring Functions

This section discusses the current facet scoring system. The facets which presently play a part in the score for a role fit are REQUIRE, PREFER, DEFAULT, TYPICAL, VALUE, MODALITY, MULTIPLICITY, and SALIENCE. The scoring function for each of these will be described in the following paragraphs.

The REQUIRE facet is an important one for detecting obviously bad candidates. If the proposed value fails to match the requirements, a partial score of -oo (negative infinity) is returned, causing the overall score to become -oo and the fit to be rejected. If there is a match between the candidate value and the requirements, then a score of +2 is reported.

The PREFER facet encodes descriptions of the preferred values for the role. If there is a match between the preferences and the candidate value, a +4 is added to the overall score. A mismatch results in a -1 increment to the score.

The TYPICAL facet describes typical values. If the candidate matches, a +8 is scored. A mismatch yields a 0 increment.

The DEFAULT facet describes the default value for this role. Thus it is a stronger description than the TYPICAL facet. A match generates a +8 score and a mismatch a 0 score.

The MODALITY facet contributes a +2 if it is filled with "obligatory", a 0 if filled with "optional" and a -oo if filled with "prohibited". Thus there is a preference for filling a role that is logically necessary to the concept.

The SALIENCE facet can contribute a score ranging from -1 to +8, depending on its contents. If this facet contains "very-low" then a -1 is scored. The +8 is generated when the facet is filled with "very-high".

Figure 7.5 summarizes the contributions to the overall score made by each of these facets.

facet	min	max
require	$-\infty$	2
prefer	-1	4
default	0	8
typical	0	8
modality	$-\infty$	2
multiplicity	$-\infty$	1
salience	-1	8

Facet score summary

figure 7.5

8 CONCLUSIONS

This final chapter attempts to briefly restate the goals of this research, assess its achievements and suggest related areas worthy of future work.

8.1 The Goals

The primary goal of this work has been to develop a computational theory for the semantic interpretation of nominal compounds. The class of a nominal compounds has been taken to include any string of nominal concepts related through modification.

A dependent goal has been the construction of a computer program which implements the theory. This program has been designed to be one component of the JETS natural language query system. The implementation of the theory has served several functions. First, it assured that the theory is reasonably specific and concrete. Second, it has acted as a test bed in which to experiment with the adequacy of the theory. Third, it has provided an important source of feedback during the theory's evolution. I discovered that some ideas just didn't work well and dropped them. The demands of the program raised some important problems which I had not recognized before, requiring the theory to address them as well.

There are several parallel goals and constraints which have made an important contribution to the overall shape of this research: the attempt to achieve a high degree of semantic closure, the design of an expressive representational system, the development of a general rule-based interpretation system, and the need for a central, uniform knowledge base.

8.2 The Accomplishments

I have developed a theory of semantic interpretation for compound nominals and implemented it in the form of a computer program. The theory can assign a sequence of two or more nominals a "most likely" interpretation in a null context, given adequate representations for the concepts involved. The theory is based on a system of rules which interpret instances of one concept modifying another. Each interpretation is also assigned a score which measures its strength. I developed a parsing algorithm which uses these local scores to choose a structure for the complete compound.

The rules fall into three broad classes: idiomatic, productive and structural. The structural rules are the most interesting, forming a kind of competence theory for non-idiomatic compounds. The idiomatic and productive rules can be viewed as providing a performance theory.

The development and evolution of the representational system which supports the theory is, I believe, an important accomplishment in its own right.

8.3 Future Work

There are a number of problems which I did not directly address in this research, but are important to both the conceptual theory and a practical implementation. I will briefly mention some of them in this section.

My initial decomposition of the problem involved three necessarily inter-related steps: lexical interpretation, modifier parsing, and conceptual modification. This thesis has concentrated on the last problem, touched on the second, and glossed over the first. I feel that the general problem of mapping surface level words into one or more underlying concepts is a good area for further work. In particular, we need to develop good ways to represent polysemous words and to find ways to select appropriate senses.

My treatment of "modifier parsing" is somewhat cursory. I would like to explore alternative algorithms, such as the stack collapsing algorithm used by Ginsparg [GINSPARG]. In this work I made two strong assumptions which might be relaxed. The effects on the parsing algorithm might be significant. The first assumption is that any compound might be endocentric, i.e. the modifier might change the basic nature of the head concept. The second assumption is that separate modifiers should never be assumed to modify independently, i.e. one must always check for interactions. Removing or relaxing these two assumptions would allow for different parsing strategies. In particular, one could develop a system for combining local interpretations and their scores.

The general issue of the effect of context has not been addressed in this thesis. There is a great deal of interesting work to be done here. This raises the problem of identifying a concept that is referred to with a <u>deictic compound</u>.

I would also like to explore the possibility of delaying the interpretations of a compound (or any utterance, for that matter), until either (1) there is sufficient evidence from the discourse context to strongly suggest a preferred interpretation; or (2) there is a strong need to compute a specific interpretation. This might be characterized as a theory of <u>lazy interpretation</u> in which decisions are delayed until they are necessary or trivial.

9 REFERENCES

[BOBROW77] Bobrow, D.G. and Winograd, T.W.; An Overview of KRL-O, a Knowledge Representation Language; Cognitive Science, volume 1, number 1; January 1977.

[BOBROW77] Bobrow, D.G. and Winograd, T.W.; An Overview of KRL-O, a Knowledge Representation Language; Proceedings of the 5th International Joint Conference on Artificial Intellegence; August 1977.

[BORGIDA] Borgida, A. T.; Topics in the Understanding of English Sentences by Computer; TR78, Department of Computer Science, University of Toronto; February 1975.

[BRACH77] Brachman, R. J.; What's in a Concept: Structural Foundations for Semantic Networks; International Journal of Man-Machine Studies, vol 9., pp. 127-152; 1977.

[BRACH78A] Brachman, R. J.; A Structured Paradigm for Representing Knowledge; Report No. 3605, Bolt Beranek and Newman; May 1978.

[BRACH78B] Brachman, R. J.; Theoretical Studies in Natural Language Understanding; Report No. 3833, Bolt Beranek and Newman; September 1978.

[BECKER] Becker, Joseph D.; The Phrasal Lexicon; Theoretical Issues in Natural language Processing - 1; June 1975..

[BULLWINK] Bullwinkle, C.; Levels of Complexity in Discourse for Anaphora Disambiguation and Speech Act Interpretation; Proceedings of the 5th International Joint Conference on Artificial Intellegence; August 1977.

[CHANG] Chang, C. L.; Finding Missing Joins for Incomplete Queries in Relational Data Bases; Report RJ2145, IBM Research Laboratory, San Jose; February 1978.

[CODD] Codd, E. F., Arnold, R.S., Cadiou, J-M., Chang, C. L. and Roussopoulos, N.; RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases; Report RJ2144, IBM Research Laboratory, San Jose; January 1978.

[CODD79] Codd, E. F.; Extending the Database Relational Model to Capture More Meaning; ACM Transactions on Database Systems, volume 4, number 4; December 1979.

[DOWNING] Downing, Pamela; On the Creation and Use of English Compound Nouns; Language, volume 53, number 4; December 1977.

[FAHLMAN] Fahlman, Scott E.; A System for Representing and Using Real-Work Knowledge; MIT-AI Memo AI-TR-450, Massachusetts Institute of Technology Artificial Intelligence Laboratory; 1977.

[FILLMO68] Fillmore, Charles J.; The Case for Case; in Bach and Harms (eds), Universals in Linguistic Theory, Holt, Reinhart and Winston, New York; 1968.

[FILLMO76] Fillmore, Charles J.; The Case for Case Reopened; in Syntax and Semantics; 1976.

[FINDLER] Findler, Nicholas V.; Associative Networks: Representation and Use of Knowledge by Computers; Academic Press, New York; 1979.

[FININ76] Finin, Timothy W.; Determining Verb Sense; unpublished working paper; 1976.

[FININ79] Finin, Timothy, Goodman, Bradley and Tennant, Harry; JETS: Achieving Completeness through Coverage and Closure; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979.

[FININ79B] Finin, Timothy W.; The Semantic Interpretation of Noun-Noun Modification; Working paper 21, Advanced Automation Group, Coordinated Science Laboratory, University of Illinois; 1979.

[FRASER] Fraser, Bruce; Some Remarks on Action Nominalization in English; in Roderick A. Jacobs and Peter S. Rosenbaum, eds., Readings in English Transformational Grammar, Ginn and Co., Waltham, Mass.; 1970.

[GERSHMAN] Gershman, A.V.; Conceptual Analysis of Noun Groups in English; Proceedings of the 5th International Joint Conference on Artificial Intellegence; August 1977.

[GINSPARG] Ginsparg, Jerrold M.; Natural Language Processing in an Automatic Programming Domain; Computer Science Department Report No. STAN-CS-78-671, Stanford University, Stanford, CA.; June 1978.

[GLEITMAN] Gleitman, Lila R. and Gleitman, Henry; Phrase and Paraphrase; W. W. Norton, New York; 1970.

[GRICE] Grice, H. P.; Logic and Conversation; in Cole and Morgan (eds.), Syntax and Semantics: Speech Acts, volume 3, Academic Press, New York; 1975.

[HAWKINSO] Hawkinson, Lowell B.; The representation of Concepts in OWL; Proceedings of the 4th International Joint Conference on Artificial Intellegence; August 1975.

[LEES60] Lees, Robert B.; The Grammar of English Nominalizations; Indiana University, Bloomington, IN; 1960.

[LEES70] Lees, Robert B; Some Problems in the Grammatical Analysis of English Nominal Compounds; in Progress in Linguistics, Moulton; 1970.

[LEVI] Levi, Judith N.; The Syntax and Semantics of Complex Nominals; Academic Press, New York; 1979.

[LEITNER] Leitner, Henry H. and Freeman, Michael W.; Structured Inheritance Networks and Natural Language Understanding; Proceedings of the 6th International Joint Conference on Artificial Intellegence; 1979.

[MARCUS] Marcus, Mitchell; A Theory of Syntactic Recognition for Natural Language; M.I.T. Press; 1979.

[MCCARTHY] McCarthy, John, Paul W. Abrahams, Daniel J. Edwards, Timothy P. Hartand Michael I. Levin; LISP 1.5 Programmer's Manual; M.I.T. Press, Cambridge, MA.; 1962.

[MCDONALD] McDonald, D. and Hayes-Roth, F.; Inferential Searches of Knowledge Networks as an Approach to Extensible Language Understanding Systems; in Waterman and Hayes-Roth (eds.) Pattern-Directed Inference Systems, Academic Press; 1978.

[MINSKY] Minsky, M.; A Framework for Representation Knowledge; in Winston (ed.) The Psychology of Computer Vision, McGraw Hill, New York; 1975.

[MOON] Moon, David A.; MacLisp Reference Manual; Project MAC, M.I.T., Cambridge, MA.; 1974.

[MOORE] Moore, J. and Newell, A.; How Can MERLIN Understand; in L. Gregg (Ed.), Knowledge and Cognition, Hillsdale, N.J. : Lawrence Erlbaum Assoc.; 1973.

[MYLOPOUL] Mylopoulos, J., P. A. Bernstein and H. K. T. Wong; TAXIS: A Language Facility for designing Interactive Database-intensive Applications; ACM Transactions on Database Systems; .

[NALDA] NALDA (Naval Air Logistics Data Analysis) System Data Requirements Determination Report; Naval Aviation Integrated Support Center, Patuxent River, Maryland; 1975.

[ORTONY] Ortony, Andrew; Metaphor and Thought; Cambridge University Press, Cambridge; 1979.

[QUIRK] Quirk, Randolph, Sidney Greenbaum, Geoffrey Leech and Jan Svartvik; A Grammar of Contemporary English; Longman Group Limited, London; 1972.

[RANDOM] The Random House Dictionary of the English Language, College Edition; Random House; 1968.

[REIMOLD] Reimold, Peter M.; An Integrated System of Perceptual Strategies: Syntactic and Semantic Interpretation of English Sentences; unpublished Ph.D. dissertation, Columbia University; 1976.

[RHYNE] Rhyne, J. R.; A Lexical Process Model of Nominal Compounding in English; American Journal of Computational Linguistics, microfiche 33; 1976.

[ROSCH] Rosch, E.; Cognitive representations of Semantic Categories; Journal of Experimental Psychology: General, 104; 1975.

[ROSENBER] Rosenberg, Steve and Roberts, Bruce R.; Coreference in a Frame Database; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979.

[ROBERTS] Roberts, Bruce R., and Goldstein, Ira P.; The FRL Manual; MIT-AI Memo 409, Massachusetts Institute of Technology Artificial Intelligence Laboratory; September 1977.

[SEARLE] Searle, J.R.; Metaphor; in Ortony (ed), Metaphor and Thought; 1979.

[SIMMONS] Simmons, R.; On Managing Sentence Meaning; CAI Laboratory report NL-6, Computer Science Department, The University of Texas, Austin; 1972.

[SOWA] Sowa, J. F.; Conceptual Graphs for a Data Base Interface; IBM Journal. of Research Development; July 1976.

[SUSSMAN] Sussman, Gerald J. and Drew V. McDermott; The CONNIVER Reference Manual; MIT-AI Memo 259A, Massachusetts Institute of Technology Artificial Intelligence Laboratory; 1974.

[TENNAN78] Tennant, Harry; The PLANES Database; Working paper 14, Advanced Automation Group, Coordinated Science Laboratory, University of Illinois; June 1978.

[TENNAN79] Tennant, Harry; Experience with the Evaluation of Natural Language Question Answerers; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979.

[WALTZ76] Waltz, D.L., Finin, T.W., Green, F., Conrad, F., Goodman, B. and Hadden, G.; The PLANES system: Natural Language Access to a Large Data Base; Technical Report T-34, Coordinated Science Laboratory, University of Illinois; November 1976.

[WALTZ77] Waltz, D. L. and B. A. Goodman; Writing a Natural Language Data Base System; Proceedings of the 5th International Joint Conference on Artificial Intellegence; 1977.

[WALTZ78] Waltz, D. L.; An English Language Question Answering System for a Large Relational Data Base; CACM, vol. 21, pp. 526-539.; July, 1978.

[WALTZ79] Waltz, D. L.; Relating Images, Concepts and Words; Proceedings of the 6th International Joint Conference on Artificial Intellegence; August 1979 .

[WINSTON] Winston, Patrick H.; Learning by Creating Transfer Frames; Artificial Intelligence, vol. 10, pp. 147-172; 1978.

[WOODS72] Woods, W.A., Kaplan, R.M., Nash-Webber, B.; The Lunar Sciences Natural Language Information System: Final report; Bolt Beranek and Newman report No. 2378; June 1972.

[WOODS75] Woods, W.A.; Whats in a Link: Foundations for Semantic Networks; in D.G. Bobrow and A.M. Collins, eds,Representation and Understanding: Studies in Cognitive Science, Academic Press; 1975.

[WOODS77] Woods, W. A.; A Personal View of Natural Language Understanding; SIGART Newsletter, number 61, pp. 17-20; February 1977.

[WOODS77B] Woods, W. A.; Semantics and Quantification in Natural Language Question Answering; Report no. 3687, Bolt Beranek and Newman; November 1977.

A Lees´ Taxonomy of Compound Nouns

This appendix reproduces the taxonomy developed by Lees in his classic work The Grammar of English Nominalizations [LEES60].

I Subject - Predicate	
A Predicate Noun	
1 Kernal Noun:	GIRL FRIEND
2 Agentive Noun:	FIGHTER PLANE
B Adjective	
1 Endocentric:	MADMAN
2 Endocentric:	REDSKIN
II Subject - Middle Object	
A Possesive Genitive:	DOCTOR´S OFFICE
B Of-periphrasis:	ARROW HEAD
C With-periphrasis:	RATTLESNAKE
III Subject - Verb	
A Gerundive Adjective:	TALKING MACHINE
B Verb - Subject:	PAY LOAD
C Subject - Nominalized Verb	
1 Of-periphrasis	
a Abstracta:	POPULATION GROWTH
b Concreta:	EARTH QUAKE
2 By-periphrasis:	FARM PRODUCTION
D Subject - Verb:	ASSEMBLY PLANT
IV Subject - Object	
A Subject - Object:	STEAM BOAT
B Object - Subject:	CAR THIEF
C From-periphrasis:	WATER SPOT
D For-periphrasis:	AUTOMOBILE PLANT
V Verb - Object	
A Infinative	
1 Endocentric:	SET SCREW
2 Exocentric:	PICK POCKET
B For-adverbial:	EATING APPLE
C Action Nominals	
1 with -ing:	SIGHT SEEING
2 with Nml	
a Abstracta:	COST REDUCTION
b Concreta:	BOOK REVIEW
D Obsolescent Object - Verb:	CHIMNEY SWEEP
VI Subject - Prepositional Object	
A Copulative	
1 Object - Preposition - Subject	
a for:	GUNPOWDER
b from,in,on,at,etc:	GARDEN PARTY
2 Object - like - Subject	
a Endocentric:	EGGPLANT
b Exocentric:	EGGHEAD

- 3 Subject - Object: CELL BLOCK
- B Subject - Prepositional Object
  - 1 Subject - Object: DEWPOINT
  - 2 Object - Subject: NIGHTOWL

VII Verb - Prepositional Object

- A Nominalization plus Object
  - 1 Infinitive: GRIND STONE
  - 2 Gerundive
    - a For-periphrasis: WASHING MACHINE
    - b Of-periphrasis: BOILING POINT
  - 3 Action Nominal: RECOVERY TIME
- B Object - Nominalization
  - 1 with -ing: PLAY GOING
  - 2 with Nml
    - a Abstracta: STEAM DISTILATION
    - b Concreta: BOAT RIDE

VIII Object - Prepositional Object

- A from Verb-phrase
  - 1 Object - Prepositional Object: BULL RING
  - 2 Prepositional object - Object: STATION WAGON
- B From NPN
  - 1 from: WOOD ALCOHOL
  - 2 of: BLOCK HOUSE
  - 3 with: DILL PICKLE
  - 4 misc.: IRON AGE

IX Proper Nouns and Naming

- A Common Noun: FERRIS WHEEL
- B Proper Noun
  - 1 with the: MARSHALL PLAN
  - 2 w/o the: STATE STREET

B The Abstraction Hierarchy

This appendix outlines a part of the space of generic concepts used in the development and testing of the implemented system. Note that some of the concepts are listed more than once, i.e. they form a directed non-cyclic graph rather than a true hierarchy. This representation of the world is incomplete and sketchy. It was constructed in order to experiment with the interpretation of nominal compounds. I do not propose it as a model of the world.

```
META-THING
   THING
       CONCRETE-THING
           ANIMATE-THING
               LEGAL-PERSON
                   CORPORATION
                   PERSON
                       MAN
                       WOMAN
           SOLID
               ARTIFACT
                   DEVICE
                   TOOL
                       SCREW-DRIVER
                   VEHICLE
                       CAR
                       PLANE
                           3MPLANE
                               A7
                               F4
                       VEHICLE29
                   PLANEPART
                       WING
                       FUSELAGE
                       ENGINE                          ; note --|
                   MECHANICAL-THING                    ;        |  tangled!
                       ENGINE                          ; <------|
                       PUMP
                       PUMP-INPUT
                       PUMP-OUTPUT
           RAW-MATERIAL
               WOOD
               METAL
                   STEEL
                   IRON
                   ALUMINUM
                   COPPER
           LIQUID
               WATER
               OIL
               ACID
           BASE
           CONCRETE-PART
```

```
          PART
     CONCRETE-THING39
ABSTRACT-THING
     COLOR
          RED
          GREEN
          YELLOW
          BLUE
     MEASURE
          LINEARMEASURE
               HIGHLOWMEASURE
                    VERY-HIGH
                    HIGH
                    MEDIUM
                    LOW
                    VERY-LOW
     NUMBER
          INTEGER
          FLONUM
     EVENT
          RESULT
          ACTION
               TRANSFER-ACTION
                    TO-SHIP
                    TO-GIVE
                    TO-REPORT
                    COMMUNICATION-EVENT
                         SEND-MESSAGE
                         UTTERANCE
                              DECLARATION
                              QUESTION
                              COMMAND
                    TO-PUMP
               DAMAGE
               CORROSION
               TO-CONSUME-ACTION
                    TO-DRINK
               TO-MANUFACTURE
               MAINTENANCE-ACTION
                    TO-REPAIR
                         TO-REPAIR24
                    TO-REPLACE
                    TO-MAINTAIN
                    TO-REMOVE
               MOVE-ACTION
                    TO-SELF-PROPEL-MOVE-ACTION
                         TO-RUN
                    TO-MOVE-WITH-A-VEHICLE
                         TO-FLY
                              TO-FLY-WITH-PLANE
                         TO-DRIVE
          OWN
          TO-MEASURE
          TO-PREVENT
          TO-INSPECT
          TO-BEGIN-A-MOVE-EVENT
               TO-TAKE-OFF
          TO-END-A-MOVE-EVENT
               TO-LAND
          TO-MODIFY
          TO-MEET
     SOURCE
     CONSUMER
```

```
        LOCATION
            REGION
                GEOGRAPHICALREGION
                POLITICALREGION
                    COUNTRY
                    STATE
                        ILLINOIS
                        INDIANA
                    COUNTY
                        CHAMPAIGNCOUNTY
                    CITY
                        CHAMPAIGNCITY
                        URBANA
            ADDRESS
        STRING
        TIME
            DATE
            YEAR
            MONTH
                JANUARY
                FEBRUARY
                MARCH
                APRIL
                MAY
                JUNE
                JULY
                SEPTEMBER
                OCTOBER
                NOVEMBER
                DECEMBER
            WEEK
            DAY
            NIGHT
            HOUR
        ABSTRACT-PART
            PART
    THING22
    PART2
ROLE
    AGENTROLE
        GIVER
        OWNER
        MECHANIC
        PILOT
        DRIVER
        MODIFIER
    OBJECTROLE
        GIFT
    INSTRUMENTROLE
        INSTRUMENTROLE30
            PUMP
    ROLE26
    ROLE27
    ROLE28
        REPLACED-PART
    ROLE38
        PART1
SIRULE
    SIRULE33 ...
INTERPRETATION
```

APPENDIX C

Examples of Concept Matching

This appendix contains traces of some sample calls to the matcher.

```
(setq fmt? T)
T


(fmatch (a person)(a man))
 matching  PERSON and  MAN
 MAN is a direct descendant of PERSON
 succeeded.
T


(fmatch (a man)(a person))
 matching  MAN and  PERSON
 MAN is a direct descendant of PERSON
 failed.
NIL


(fmatch (an event with agent object)(a to-fly))
 matching  EVENT43 and  TO-FLY
Applying the recursive frame matcher to EVENT43  and  TO-FLY
   matching contents of EVENT43's LOCATION and  TO-FLY's LOCATION
   EVENT43's LOCATION and  TO-FLY's LOCATION are identical!
   succeeded.
   matching contents of EVENT43's TIME and  TO-FLY's TIME
   EVENT43's TIME and  TO-FLY's TIME are identical!
   succeeded.
   matching contents of EVENT43's RESULT and  TO-FLY's RESULT
   EVENT43's RESULT and  TO-FLY's RESULT are identical!
   succeeded.
   matching contents of EVENT43's CAUSE and  TO-FLY's CAUSE
   EVENT43's CAUSE and  TO-FLY's CAUSE are identical!
   succeeded.
   matching contents of EVENT43's OBJECT and  TO-FLY's OBJECT
     checking the existence of TO-FLY's OBJECT.
     succeeded.
   succeeded.
   matching contents of EVENT43's AGENT and  TO-FLY's AGENT
     checking the existence of TO-FLY's AGENT
     succeeded.
   succeeded.
 succeeded.
T


(fmatch (an event with agent matching (a person))(a to-fly))
 matching  EVENT44 and  TO-FLY
   Applying the recursive frame matcher to EVENT44  and  TO-FLY
   matching contents of EVENT44's LOCATION and  TO-FLY's LOCATION
   EVENT44's LOCATION and  TO-FLY's LOCATION are identical!
   succeeded.
   matching contents of EVENT44's TIME and  TO-FLY's TIME
   EVENT44's TIME and  TO-FLY's TIME are identical!
   succeeded.
```

```
      matching contents of EVENT44's RESULT and  TO-FLY's RESULT
      EVENT44's RESULT and  TO-FLY's RESULT are identical!
      succeeded.
      matching contents of EVENT44's CAUSE and  TO-FLY's CAUSE
      EVENT44's CAUSE and  TO-FLY's CAUSE are identical!
      succeeded.
      matching contents of EVENT44's AGENT and  TO-FLY's AGENT
        matching requirements of EVENT44's AGENT
          matching requirements of EVENT44's AGENT and  TO-FLY's AGENT
            matching requirements PERSON and  ANIMATE-THING
              matching  PERSON and  ANIMATE-THING
                frames  PERSON and  ANIMATE-THING are in incompatable
                           basic categories.
                   PERSON : PERSON and  ANIMATE-THING : ANIMATE-THING
              failed.
            failed.
          failed.
        failed.
      failed.
    failed.
   NIL


   (fmatch (a man)(a woman))
    matching  MAN and  WOMAN
     Applying the recursive frame matcher to MAN  and  WOMAN
     matching contents of MAN's LOCATION and  WOMAN's LOCATION
     MAN's LOCATION and  WOMAN's LOCATION are identical!
     succeeded.
     matching contents of MAN's RAW-MATERIAL and  WOMAN's RAW-MATERIAL
     MAN's RAW-MATERIAL and  WOMAN's RAW-MATERIAL are identical!
     succeeded.
     matching contents of MAN's COLOR and  WOMAN's COLOR
     MAN's COLOR and  WOMAN's COLOR are identical!
     succeeded.
     matching contents of MAN's MASS and  WOMAN's MASS
     MAN's MASS and  WOMAN's MASS are identical!
     succeeded.
     matching contents of MAN's SUPERPART and  WOMAN's SUPERPART
     MAN's SUPERPART and  WOMAN's SUPERPART are identical!
     succeeded.
     matching contents of MAN's NAME and  WOMAN's NAME
     MAN's NAME and  WOMAN's NAME are identical!
     succeeded.
     matching contents of MAN's ADDRESS and  WOMAN's ADDRESS
     MAN's ADDRESS and  WOMAN's ADDRESS are identical!
     succeeded.
     matching contents of MAN's AGE and  WOMAN's AGE
     MAN's AGE and  WOMAN's AGE are identical!
     succeeded.
     matching contents of MAN's SUBPART and  WOMAN's SUBPART
     MAN's SUBPART and  WOMAN's SUBPART are identical!
     succeeded.
     matching contents of MAN's SEX and  WOMAN's SEX
       matching values of MAN's SEX and  WOMAN's SEX
         matching literals MALE and  FEMALE
         failed.
       failed.
     failed.
    failed.
   NIL
```

```
(fmatch (a plane)(a to-repair))
 matching  PLANE and  TO-REPAIR
   frames  PLANE and  TO-REPAIR are in incompatable basic categories.
      PLANE : PLANE and  TO-REPAIR : EVENT
 failed.
NIL

(fmatch (a man )(a man))
 matching  MAN and  MAN
 MAN is the same as MAN
 succeeded.
T
```

D Examples of Interpretation

This appendix contains traces of the interpretaion of three compounds. The first two, engine repairs and water pump, are shown in slightly more detail than the third, F4 water pump repairs.

```
(hello)

 >> engine repairs

   trying to interpret ENGINE modifying  TO-REPAIR
   Looking for sirules to interpret NNMODIFY ENGINE TO-REPAIR

   Rule  SIRULE38 Suggested.
        TEST: (AKO? THING1 THING2) failed.
        Rule  SIRULE38 failing.

   Rule  SIRULE39 Suggested.
        TEST: (AKO? THING2 THING1) failed.
        Rule  SIRULE39 failing.

   Rule  SIRULE40 Suggested.
        Rule  SIRULE40 applies to NNMODIFY ENGINE TO-REPAIR
        Hypothesis is that ENGINE fills one of TO-REPAIR 's roles.
        looking for best fit in TO-REPAIR for  ENGINE
        fillable slots are: AGENT OBJECT INSTRUMENT CAUSE RESULT TIME LOCATION

         Scoring  TO-REPAIR's AGENT <-- ENGINE ...
         Scoring  TO-REPAIR's OBJECT <-- ENGINE ...  4.
         Scoring  TO-REPAIR's INSTRUMENT <-- ENGINE ...  1.
         Scoring  TO-REPAIR's CAUSE <-- ENGINE ...
         Scoring  TO-REPAIR's RESULT <-- ENGINE ...
         Scoring  TO-REPAIR's TIME <-- ENGINE ...
         Scoring  TO-REPAIR's LOCATION <-- ENGINE ...
         Non-negative scores for ENGINE as the filler for one
         of TO-REPAIR 's slots.
               INSTRUMENT      1.
               OBJECT  4.

        There is only one candidate, OBJECT , with score 4.

        Hypothesis is: ENGINE fills  TO-REPAIR's OBJECT
```

```
          Scoring  ENGINE's USE <-- TO-REPAIR ...  1.
          Scoring  ENGINE's RAW-MATERIAL <-- TO-REPAIR ...
          Scoring  ENGINE's MASS <-- TO-REPAIR ...
          Scoring  ENGINE's COLOR <-- TO-REPAIR ...
          Scoring  ENGINE's LOCATION <-- TO-REPAIR ...
          Non-negative scores for TO-REPAIR as the filler for one
           of ENGINE 's slots.
                USE     1.

         There is only one candidate, USE , with score 1.

         Hypothesis is:  TO-REPAIR fills  ENGINE's USE

         Rule  SIRULE41 succeeds with INTERPRETATION52 which has score 1.

     Rule  SIRULE38 Suggested.
     I have tried rule SIRULE38 before.

     Rule  SIRULE39 Suggested.
     I have tried rule SIRULE39 before.

     Rule  SIRULE40 Suggested.
     I have tried rule SIRULE40 before.

     Rule  SIRULE41 Suggested.
     I have tried rule SIRULE41 before.
     No more rules matching NNMODIFY ENGINE TO-REPAIR
     my interpretation of ENGINE MODIFYING TO-REPAIR is:

(A INTERPRETATION50 IS (AN INTERPRETATION) WITH
   BASIC = (AN INTERPRETATION)
   MODIFIER = (AN ENGINE)
   MODIFIED = (A TO-REPAIR)
   SCORE = 4.
   INTERPRETATION = (A TO-REPAIR51 IS (A TO-REPAIR) WITH
                       BASIC = (AN EVENT)
                       OBJECT = (AN ENGINE)
                       INTERPRETED-FROM = (AN INTERPRETATION50))
   IMPORTANTFRAMES = (AN ENGINE) AND (THE TO-REPAIR51 )
   SIRULE = '(|interpret N1 as filling one of N2's roles|))

My interpretation of ENGINE REPAIRS is:

(A TO-REPAIR51 IS (A TO-REPAIR) WITH
   BASIC = (AN EVENT)
   OBJECT = (AN ENGINE)
   INTERPRETED-FROM = (AN INTERPRETATION50))
```

```
  >> water pump

I am assuming that the word WATER refers only to the concept WATER
I am assuming that the word PUMP refers only to the concept PUMP
    trying to interpret WATER modifying  PUMP
    Looking for sirules to interpret NNMODIFY WATER PUMP

    Rule  SIRULE38 Suggested.
         TEST: (AKO? THING1 THING2) failed.
         Rule  SIRULE38 failing.

    Rule  SIRULE39 Suggested.
         TEST: (AKO? THING2 THING1) failed.
         Rule  SIRULE39 failing.

    Rule  SIRULE40 Suggested.
         Rule  SIRULE40 applies to NNMODIFY WATER PUMP
         Hypothesis is that WATER fills one of PUMP ´s roles.
         looking for best fit in PUMP for  WATER
         fillable slots are: SUBPART FRAME SUPERPART RAW-MATERIAL
                             MASS COLOR LOCATION ROLE

          Scoring  PUMP´s SUBPART <-- WATER ...
          Scoring  PUMP´s FRAME <-- WATER ...
          Scoring  PUMP´s SUPERPART <-- WATER ...
          Scoring  PUMP´s RAW-MATERIAL <-- WATER ...
         Applying the recursive frame matcher to RAW-MATERIAL  and  WATER
          Scoring  PUMP´s MASS <-- WATER ...
          Scoring  PUMP´s COLOR <-- WATER ...
          Scoring  PUMP´s LOCATION <-- WATER ...
          Scoring  PUMP´s ROLE <-- WATER ...
         No viable role candidates!
         Hypothesis rejected.
         Rule  SIRULE40 failed.


    Rule  SIRULE41 Suggested.
         Rule  SIRULE41 applies to NNMODIFY WATER PUMP
         Hypothesis is that PUMP fills one of WATER ´s roles.
         looking for best fit in WATER for  PUMP
         fillable slots are: ROLE FRAME SUBPART SUPERPART MASS COLOR
                             RAW-MATERIAL LOCATION

          Scoring  WATER´s ROLE <-- PUMP ...
          Scoring  WATER´s FRAME <-- PUMP ...
          Scoring  WATER´s SUBPART <-- PUMP ...  2.
          Scoring  WATER´s SUPERPART <-- PUMP ...  2.
          Scoring  WATER´s MASS <-- PUMP ...
          Scoring  WATER´s COLOR <-- PUMP ...
          Scoring  WATER´s RAW-MATERIAL <-- PUMP ...
         Applying the recursive frame matcher to RAW-MATERIAL  and  PUMP
          Scoring  WATER´s LOCATION <-- PUMP ...
         Non-negative scores for PUMP as the filler for one of WATER ´s slots.
                SUPERPART       2.
                SUBPART         2.

         best weight is: 2.
         candidates are: SUBPART SUPERPART
         I am arbitrarily selecting SUBPART as best.

         Hypothesis is:  PUMP fills  WATER´s SUBPART
```

```
        Rule  SIRULE41 succeeds with INTERPRETATION56 which has score 2.


    Rule  SIRULE42 Suggested.
        Rule  SIRULE42 applies to NNMODIFY WATER PUMP
        PUMP refers to the INSTRUMENT role of a TO-PUMP
        Perhaps  WATER modifies an instance of TO-PUMP ...
            trying to interpret WATER modifying  TO-PUMP61
            Looking for sirules to interpret NNMODIFY WATER TO-PUMP61

            Rule  SIRULE38 Suggested.
                TEST: (AKO? THING1 THING2) failed.
                Rule  SIRULE38 failing.

            Rule  SIRULE39 Suggested.
                TEST: (AKO? THING2 THING1) failed.
                Rule  SIRULE39 failing.

            Rule  SIRULE40 Suggested.
                Rule  SIRULE40 applies to NNMODIFY WATER TO-PUMP61
                Hypothesis is that WATER fills one of TO-PUMP61 ´s roles.
                looking for best fit in TO-PUMP61 for  WATER
                fillable slots are: INSTRUMENT AGENT OBJECT SOURCE
                                    RECIPIENT CAUSE RESULT TIME LOCATION

                 Scoring  TO-PUMP61´s INSTRUMENT <-- WATER ...
                Applying the recursive frame matcher to PUMP  and  WATER
                 Scoring  TO-PUMP61´s AGENT <-- WATER ...  1.
                 Scoring  TO-PUMP61´s OBJECT <-- WATER ...  12.
                 Scoring  TO-PUMP61´s SOURCE <-- WATER ...  1.
                 Scoring  TO-PUMP61´s RECIPIENT <-- WATER ...  2.
                 Scoring  TO-PUMP61´s CAUSE <-- WATER ...
                 Scoring  TO-PUMP61´s RESULT <-- WATER ...
                 Scoring  TO-PUMP61´s TIME <-- WATER ...
                 Scoring  TO-PUMP61´s LOCATION <-- WATER ...
                Non-negative scores for WATER as the filler for one
                of TO-PUMP61 ´s slots.
                    RECIPIENT      2.
                    SOURCE  1.
                    OBJECT  12.
                    AGENT   1.

                There is only one candidate, OBJECT , with score 12.

                Hypothesis is: WATER fills  TO-PUMP61´s OBJECT

                Rule  SIRULE40 succeeds with INTERPRETATION62
                                         which has score 12.


            my interpretation of WATER MODIFYING TO-PUMP61 is:

(A INTERPRETATION62 IS (AN INTERPRETATION) WITH
   BASIC = (AN INTERPRETATION)
   MODIFIER = (A WATER)
   MODIFIED = (A TO-PUMP61)
   SCORE = 12.
   INTERPRETATION = (A TO-PUMP6163 IS (A TO-PUMP61) WITH
                       BASIC = (AN EVENT)
                       OBJECT = (A WATER)
                       INTERPRETED-FROM = (AN INTERPRETATION62))
   IMPORTANTFRAMES = (A WATER) AND (THE TO-PUMP6163)
   SIRULE = ´(¦interpret N1 as filling one of N2´s roles¦))

         Rule  SIRULE42 succeeds with INTERPRETATION59 which has score 12.

    my interpretation of WATER MODIFYING PUMP is:

(A INTERPRETATION59 IS (AN INTERPRETATION) WITH
   BASIC = (AN INTERPRETATION)
   MODIFIER = (A WATER)
   MODIFIED = (A PUMP)
   INTERPRETATION = (A PUMP60 IS (A PUMP) WITH
                       BASIC = (AN ARTIFACT) AND (A ROLE)
                       INTERPRETED-FROM = (AN INTERPRETATION59)
                       FRAME = (A TO-PUMP61) AND (A TO-PUMP6163))
   SCORE = 12.
   IMPORTANTFRAMES = (A WATER) AND (THE PUMP60) AND (A TO-PUMP6163)
   SIRULE = ´(¦interpret N2 as a role of a underlying frame modified by N1¦))

My interpretation of WATER PUMP is:

(A PUMP60 IS (A PUMP) WITH
   BASIC = (AN ARTIFACT) AND (A ROLE)
   INTERPRETED-FROM = (AN INTERPRETATION59)
   FRAME = (A TO-PUMP61) AND (A TO-PUMP6163))
```

```
(hello)

  >> f4 water pump repair

I am assuming that the word F4 refers only to the concept F4
I am assuming that the word WATER refers only to the concept WATER
I am assuming that the word PUMP refers only to the concept PUMP
    trying to interpret F4 modifying  WATER
    Looking for sirules to interpret NNMODIFY F4 WATER
          Rule  SIRULE40 applies to NNMODIFY F4 WATER
          Hypothesis is that F4 fills one of WATER 's roles.
          looking for best fit in WATER for  F4
          fillable slots are: ROLE FRAME SUBPART SUPERPART MASS COLOR
                               RAW-MATERIAL LOCATION

          Applying the recursive frame matcher to RAW-MATERIAL  and  F4
          Non-negative scores for F4 as the filler for one of WATER 's slots.
                 SUPERPART       2.
                 SUBPART         2.

          best weight is: 2.
          candidates are: SUBPART SUPERPART
          I am arbitrarily selecting SUBPART as best.

          Hypothesis is: F4 fills  WATER's SUBPART

          Rule  SIRULE40 succeeds with INTERPRETATION43 which has score 2.


          Rule  SIRULE41 applies to NNMODIFY F4 WATER
          Hypothesis is that WATER fills one of F4 's roles.
          looking for best fit in F4 for  WATER
          fillable slots are: TYPE STATUS ROLE FRAME SUBPART SUPERPART
                               RAW-MATERIAL MASS COLOR LOCATION

          Applying the recursive frame matcher to RAW-MATERIAL  and  WATER
          No viable role candidates!
          Hypothesis rejected.
          Rule  SIRULE41 failed.


          Rule  SIRULE42 applies to NNMODIFY F4 WATER
          WATER refers to the OBJECT role of a TO-DRINK
          Perhaps  F4 modifies an instance of TO-DRINK ...
               trying to interpret F4 modifying  TO-DRINK48
               Looking for sirules to interpret NNMODIFY F4 TO-DRINK48
                    Rule  SIRULE40 applies to NNMODIFY F4 TO-DRINK48
                    Hypothesis is that F4 fills one of TO-DRINK48 's roles.
                    looking for best fit in TO-DRINK48 for  F4
                    fillable slots are: OBJECT AGENT CAUSE RESULT TIME LOCATION

                    Applying the recursive frame matcher to LIQUID  and  F4
                    No viable role candidates!
                    Hypothesis rejected.
                    Rule  SIRULE40 failed.


                    Rule  SIRULE41 applies to NNMODIFY F4 TO-DRINK48
                    Hypothesis is that TO-DRINK48 fills one of F4 's roles.
                    looking for best fit in F4 for  TO-DRINK48
                    fillable slots are: TYPE STATUS ROLE FRAME SUBPART
                              SUPERPART RAW-MATERIAL MASS COLOR LOCATION
```

```
                No viable role candidates!
                Hypothesis rejected.
                Rule  SIRULE41 failed.


            No more rules matching NNMODIFY F4 TO-DRINK48
            I have several candidate interpretations: INTERPRETATION50 and
                                                      INTERPRETATION49
            All having the score -34359738367.
            I pick the first!
            I failed to interpret F4 modifying  TO-DRINK48
        oh well.....
        Rule  SIRULE42 failed.


    No more rules matching NNMODIFY F4 WATER
    my interpretation of F4 MODIFYING WATER is:

(A INTERPRETATION43 IS (AN INTERPRETATION) WITH
   BASIC = (AN INTERPRETATION)
   MODIFIER = (A F4)
   MODIFIED = (A WATER)
   SCORE = 2.
   INTERPRETATION = (A WATER44 IS (A WATER) WITH
                      BASIC = (A WATER)
                      SUBPART = (A F4)
                      INTERPRETED-FROM = (AN INTERPRETATION43))
   IMPORTANTFRAMES = (A F4) AND (THE WATER44)
   SIRULE = ´(|interpret N1 as filling one of N2´s roles|))

The score for that interpretation is only moderate.
I am trying an alternative
    trying to interpret WATER modifying  PUMP
    Looking for sirules to interpret NNMODIFY WATER PUMP
        Rule  SIRULE40 applies to NNMODIFY WATER PUMP
        Hypothesis is that WATER fills one of PUMP ´s roles.
        looking for best fit in PUMP for  WATER
        fillable slots are: SUBPART USE FRAME SUPERPART RAW-MATERIAL MASS
                            COLOR LOCATION ROLE

        No viable role candidates!
        Hypothesis rejected.
        Rule  SIRULE40 failed.


        Rule  SIRULE41 applies to NNMODIFY WATER PUMP
        Hypothesis is that PUMP fills one of WATER ´s roles.
        looking for best fit in WATER for  PUMP
        fillable slots are: ROLE FRAME SUBPART SUPERPART MASS COLOR
                            RAW-MATERIAL LOCATION

        Applying the recursive frame matcher to RAW-MATERIAL  and  PUMP
        Non-negative scores for PUMP as the filler for one of WATER ´s slots.
               SUPERPART       2.
               SUBPART         2.

        best weight is: 2.
        candidates are: SUBPART SUPERPART
        I am arbitrarily selecting SUBPART as best.

        Hypothesis is:  PUMP fills  WATER´s SUBPART
        Rule  SIRULE41 succeeds with INTERPRETATION52 which has score 2.


        Rule  SIRULE42 applies to NNMODIFY WATER PUMP
        PUMP refers to the INSTRUMENT role of a TO-PUMP
        Perhaps  WATER modifies an instance of TO-PUMP ...
            trying to interpret WATER modifying  TO-PUMP57
            Looking for sirules to interpret NNMODIFY WATER TO-PUMP57
                Rule  SIRULE40 applies to NNMODIFY WATER TO-PUMP57
                Hypothesis is that WATER fills one of TO-PUMP57 ´s roles.
                looking for best fit in TO-PUMP57 for  WATER
                fillable slots are: INSTRUMENT AGENT OBJECT SOURCE
                                    RECIPIENT CAUSE RESULT TIME LOCATION

                Applying the recursive frame matcher to PUMP  and  WATER
                Non-negative scores for WATER as the filler for one
                of TO-PUMP57 ´s slots.
                    RECIPIENT       2.
                    SOURCE  1.
                    OBJECT  12.
                    AGENT   1.

                There is only one candidate, OBJECT , with score 12.

                Hypothesis is: WATER fills  TO-PUMP57´s OBJECT

                Rule  SIRULE40 succeeds with INTERPRETATION58 which has
                          score 12.


            my interpretation of WATER MODIFYING TO-PUMP57 is:

    (A INTERPRETATION58 IS (AN INTERPRETATION) WITH
       BASIC = (AN INTERPRETATION)
       MODIFIER = (A WATER)
       MODIFIED = (A TO-PUMP57)
       SCORE = 12.
       INTERPRETATION = (A TO-PUMP5759 IS (A TO-PUMP57) WITH
                           BASIC = (AN EVENT)
                           OBJECT = (A WATER)
                           INTERPRETED-FROM = (AN INTERPRETATION58))
       IMPORTANTFRAMES = (A WATER) AND (THE TO-PUMP5759)
       SIRULE = ´(¦interpret N1 as filling one of N2´s roles¦))

            Rule  SIRULE42 succeeds with INTERPRETATION55 which has score 12.


       my interpretation of WATER MODIFYING PUMP is:

    (A INTERPRETATION55 IS (AN INTERPRETATION) WITH
       BASIC = (AN INTERPRETATION)
       MODIFIER = (A WATER)
       MODIFIED = (A PUMP)
       INTERPRETATION = (A PUMP56 IS (A PUMP) WITH
                           BASIC = (AN ARTIFACT) AND (A ROLE)
                           INTERPRETED-FROM = (AN INTERPRETATION55)
                           FRAME = (A TO-PUMP57) AND (A TO-PUMP5759))
       SCORE = 12.
       IMPORTANTFRAMES = (A WATER) AND (THE PUMP56) AND (A TO-PUMP5759)
       SIRULE = ´(¦interpret N2 as a role of a underlying frame modified by N1¦))

    Alternative 2 is better!
       trying to interpret F4 modifying  PUMP56
    Looking for sirules to interpret NNMODIFY F4 PUMP56
         Rule  SIRULE40 applies to NNMODIFY F4 PUMP56
         Hypothesis is that F4 fills one of PUMP56 ´s roles.
         looking for best fit in PUMP56 for  F4
         fillable slots are: FRAME SUBPART USE SUPERPART RAW-MATERIAL MASS
                                       COLOR LOCATION ROLE

         Applying the recursive frame matcher to MECHANICAL-THING  and  F4
         Non-negative scores for F4 as the filler for one of PUMP56 ´s slots.
                SUPERPART       6.

         There is only one candidate, SUPERPART , with score 6.

         Hypothesis is: F4 fills  PUMP56´s SUPERPART

         Rule  SIRULE40 succeeds with INTERPRETATION60 which has score 6.


    my interpretation of F4 MODIFYING PUMP56 is:

(A INTERPRETATION60 IS (AN INTERPRETATION) WITH
   BASIC = (AN INTERPRETATION)
   MODIFIER = (A F4)
   MODIFIED = (A PUMP56)
   SCORE = 6.
   INTERPRETATION = (A PUMP5661 IS (A PUMP56) WITH
                       BASIC = (AN ARTIFACT) AND (A ROLE)
                       SUPERPART = (A F4)
                       INTERPRETED-FROM = (AN INTERPRETATION60))
   IMPORTANTFRAMES = (A F4) AND (THE PUMP5661)
   SIRULE = ´(¦interpret N1 as filling one of N2´s roles¦))

big win as F4 modifies  PUMP56
    trying to interpret PUMP5661 modifying  TO-REPAIR
    Looking for sirules to interpret NNMODIFY PUMP5661 TO-REPAIR
         Rule  SIRULE40 applies to NNMODIFY PUMP5661 TO-REPAIR
         Hypothesis is that PUMP5661 fills one of TO-REPAIR ´s roles.
         looking for best fit in TO-REPAIR for  PUMP5661
         fillable slots are: AGENT OBJECT INSTRUMENT CAUSE RESULT TIME
                                       LOCATION

         Applying the recursive frame matcher to DEVICE  and  PUMP5661
         Applying the recursive frame matcher to TOOL  and  PUMP5661
         Non-negative scores for PUMP5661 as the filler for one
         of TO-REPAIR ´s slots.
                INSTRUMENT      1.
                OBJECT  4.

         There is only one candidate, OBJECT , with score 4.

         Hypothesis is: PUMP5661 fills  TO-REPAIR´s OBJECT

         Rule  SIRULE40 succeeds with INTERPRETATION62 which has score 4.


         Rule  SIRULE41 applies to NNMODIFY PUMP5661 TO-REPAIR
         Hypothesis is that TO-REPAIR fills one of PUMP5661 ´s roles.
         looking for best fit in PUMP5661 for  TO-REPAIR
         fillable slots are: SUPERPART FRAME SUBPART USE RAW-MATERIAL MASS
                                       COLOR LOCATION ROLE

         Non-negative scores for TO-REPAIR as the filler for one
         of PUMP5661 ´s slots.
```

```
            USE     4.

        There is only one candidate, USE , with score 4.

        Hypothesis is:  TO-REPAIR fills  PUMP5661´s USE

        Rule  SIRULE41 succeeds with INTERPRETATION64 which has score 4.


    No more rules matching NNMODIFY PUMP5661 TO-REPAIR
    I have several candidate interpretations: INTERPRETATION64 and
                                                INTERPRETATION62
    All having the score 4.
    I pick the first!
    my interpretation of PUMP5661 MODIFYING TO-REPAIR is:

(A INTERPRETATION64 IS (AN INTERPRETATION) WITH
   BASIC = (AN INTERPRETATION)
   MODIFIER = (A PUMP5661)
   MODIFIED = (A TO-REPAIR)
   SCORE = 4.
   INTERPRETATION = (A TO-REPAIR66 IS (A TO-REPAIR) WITH
                       BASIC = (AN EVENT)
                       ROLE = ´USE
                       FRAME = (A PUMP566165)
                       INTERPRETED-FROM = (AN INTERPRETATION64))
   IMPORTANTFRAMES = (A PUMP566165) AND (THE TO-REPAIR66)
   SIRULE = ´(|interpret N2 as filling one of N1´s roles|))


My interpretation of F4 WATER PUMP REPAIR is:

(A TO-REPAIR66 IS (A TO-REPAIR) WITH
   BASIC = (AN EVENT)
   ROLE = ´USE
   FRAME = (A PUMP566165)
   INTERPRETED-FROM = (AN INTERPRETATION64))
```

VITA

Timothy Wilking Finin was born in Decatur, Illinois on August 4, 1949. He received the S.B. degree in Electrical Engineering from the Massachusetts Institute of Technology in 1971. From 1971 to 1974 he was a member of the research staff of the Artificial Intelligence Laboratory at M.I.T. where he worked in the areas of computer vision and robotics.

In 1974 he enrolled in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He received the M.S. degree in 1977 with a thesis entitled "An Interpreter and Compiler for Augmented Transition Networks". In 1980 he received the Ph.D. degree with a thesis entitled "The Semantic Interpretation of Compound Nominals".

While attending the University of Illinois he was a research assistant at the Coordinated Science Laboratory. During the summer of 1977 he was a research associate at the I.B.M. Research Laboratory at San Jose, California. He is a member of the Association of Computing Machinery, the Institute of Electronic and Electrical Engineers, the Association of Computational Linguistics and Sigma Xi. He is the author of several scientific papers in the areas of computer vision and computational linguistics.