American Journal of Computational Linguistics *Microfiche 53*

Computer Understanding of PHYSICS PROBLEMS Stated in Natural Language

Gordon S. Novak Jr.

Computer Science Department
University of Texas
Austin 78712

SUMMARY

This paper describes a computer program, called ISAAC, which can read, understand, solve, and draw pictures of physics problems stated in English. The program has solved twenty problems, most of which were taken unedited from high school and college physics texts. These problems involve rigid bodies in static equilibrium, and include such objects as levers, pivots, weights, ropes, and springs in various configurations. An example of the class of problems solved is the following (from *Schaum's Outline of College Physics*):

> The foot of a ladder rests against a vertical wall and on a horizontal floor. The top of the ladder is supported from the wall by a horizontal rope 30 ft long. The ladder is 50 ft long, weighs 100 lb with its center of gravity 20 ft from the foot, and a 150 lb man is 10 ft from the top. Determine the tension in the rope.

In order to understand and solve such a problem, it is necessary to build an internal model of the problem in which the various objects and their interrelationships are adequately represented. Many of the relationships and features of the objects are not specified explicitly in the problem statement but must be inferred by using common sense knowledge of what is usual. In the above example, we assume that the man is standing on the ladder, although this is not explicitly stated. Thus, the understanding of a physics problem is an active process in which the sentences of the problem statement are used to guide the construction of a model which represents the relationships and features of objects with much greater detail and specificity than they are specified in the original problem statement.

In this paper, we investigate ways in which the meanings of phrases and sentences may be understood and related to a developing model of the problem, using common sense knowledge (represented by computer programs) to aid the understanding process. Ways of representing objects and their relationships are developed. These representations, which are originally created in response to the sentences in the problem statement, are further elaborated by processes which construct a geometric model of the problem, associate canonical objects (such as a point mass) with physical objects (such as a person), write and solve equations which describe the interactions of the objects and construct a diagram of the problem.

This paper is a slightly edited version of the author.s Ph.D. dissertation. I am grateful to my committee members, Professors John Loehlin, Woodrow Bledsoe, and Norman Martin, and especially to my supervising professor, Robert F. Simmons.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction and Overview

1.1 Introduction

This paper describes a computer program, called ISAAC, which is able to read and understand physics problems stated in English, write equations for the problems and solve them, and draw diagrams showing the objects in the problems and their spatial relationships. The program has solved twenty problems, which were taken essentially unedited from physics textbooks; some sample problems are shown, with the drawings and answers generated by the program, in Appendix A.

While the diagram and answer to a problem are the most easily observable outputs of the program, another significant output is its robust internal model of the objects in the problem and their relationships. It is this model which makes possible the generation of the diagram and the answer to the problem. The internal model is robust in the sense that it represents, in an explicit and readily accessible form, most of the information which a competent human reader might be expected to derive from the English problem statement. In addition to the ways in which it is currently used, the model could be used for answering questions about the objects and their relationships, or for generating a description of the problem in English or in another language, or for generating other types of diagrams (such as a force diagram). Since it makes all of the features and relationships of the objects explicit, the internal model is many times larger than the original problem statement, which specifies only the major features and leaves many details to be filled in by the reader.

1.2 Overview of the Program

The overall organization of the program and its data elements is shown in Figure 1.1; programs are represented by boxes with double lines, and data structures by plain boxes. In this section, we present an overview of the functions performed by each group of programs and an overview of the types of information represented in the data structures.

The process of understanding and solving a physics problem occurs in several distinct steps. First, the problem statement is translated from English into a structured parsing of the sentences, which is interpreted semantically to construct an initial internal model. This model is interpreted to form a model in terms of canonical physical objects (such as a point mass). A geometric model which represents the spatial position and orientation of each object is constructed. Equations which describe the

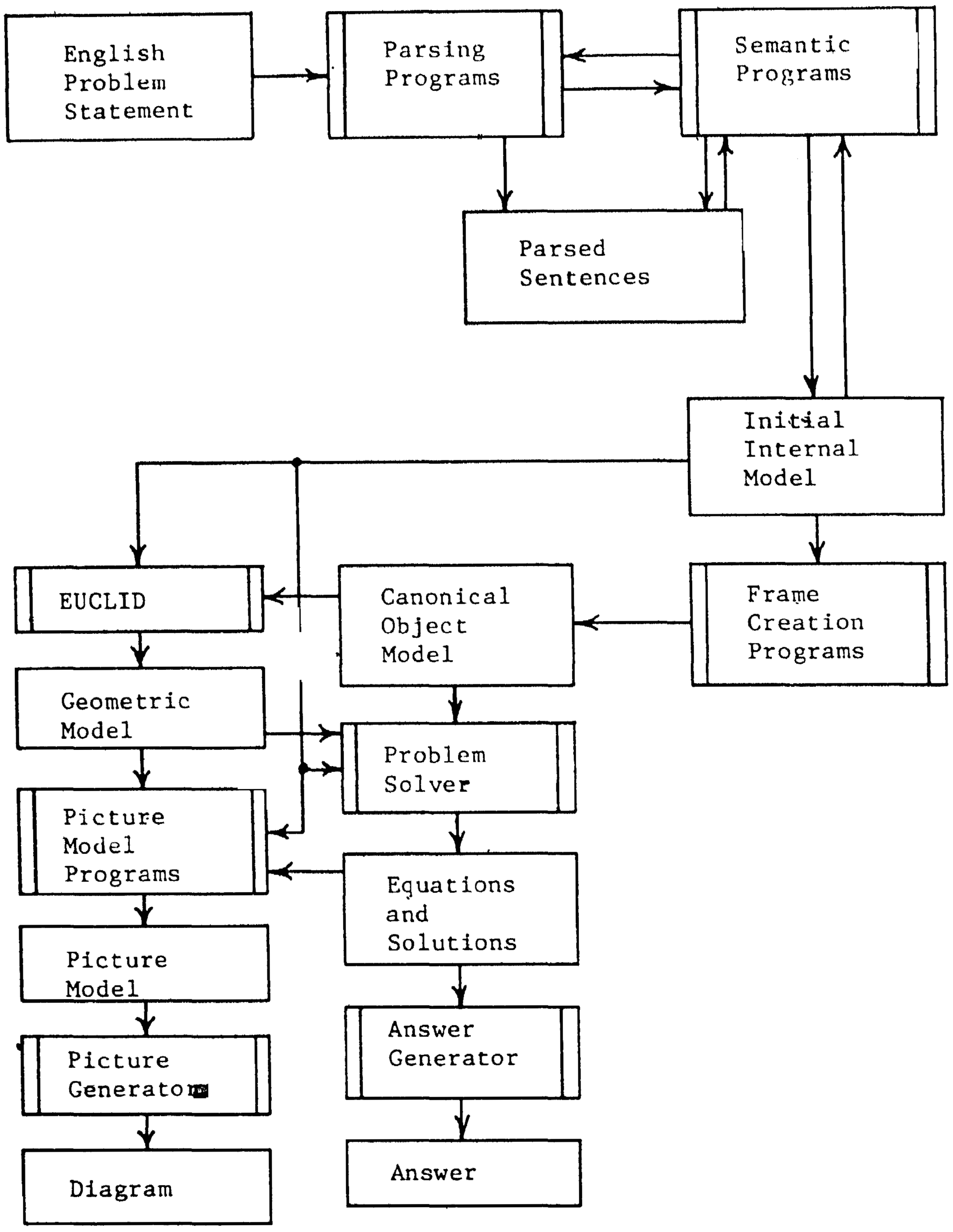


Figure 1.1: Overall Program Organization

interactions of the objects according to physical laws are written and solved, and answers are generated from the solutions. Finally, a picture model is constructed and used to guide the drawing of a diagram of the problem. These processes are described below in somewhat greater detail.

The parsing programs transform each sentence from a linear string of words into a more structured form in which the relationships of words and phrases to each other are clearly defined. Each type of phrase is parsed by a specialist program which implements the grammar of the phrase as an augmented transition network. The grammar programs call the semantic programs both to interpret the semantic network structure produced as a result of the parsing and to guide the parsing process itself. Whether a particular modifier can be used to modify a noun phrase, for example, may depend on the actual relationships between objects in the model of the problem. After each clause has been parsed, the semantic routine associated with the main verb is called to complete the semantic processing of the clause. This semantic processing transfers the information provided by the sentence to a growing model of the objects in the problem and their relationships. Once the semantic processing is completed, the semantic network structures produced by the parser are no longer used. All of the sentences in the problem statement are parsed and processed semantically before the remaining parts of the program are executed.

The routines grouped under the heading of "Semantic Programs" perform a variety of tasks. Semantic routines are associated with verbs and prepositions, and in some cases with other words. Preposition semantic routines must determine the appropriate sense-meaning of the preposition (using a decision network based on rough semantic classifications of the modified phrase and the object of the preposition) and then make the appropriate changes in the semantic network token of the modified phrase. Verb semantic routines typically act to transfer information from the semantic network to the internal model, or define relationships among objects in the model; determination of the proper sense-meaning of the verb is often needed as well. Another major semantic task is the identification of the referent of a noun phrase. Given a noun phrase which denotes an object or a location, it is necessary to decide whether the phrase refers to an object or location which already exists in the model (and if there are several possibilities, which one it refers to), or whether a new object or location must be created as the referent of the phrase and added to the model.

After all the sentences of the problem statement have been read, the frame creation programs are called to assign to each object a canonical object type (such as a point mass or an "ideal" spring) which represents the behavior of the object as it appears in the problem. The same type of actual object may be represented by different canonical objects, depending on its context in the problem. Thus, a person may be represented as a pivot when carrying a plank, or as a point mass when standing on one. Once a canonical object frame has been selected, the model of the object is examined to see if it is complete for that type of object; if not, assumptions are made to correct the deficiencies. Thus, for example, if the length of a pole is unspecified, a symbolic constant representing the length is created and added to the model.

After canonical object frames have been created for all the objects, the program EUCLID is called to construct a geometric model of the problem. EUCLID computes the orientation of each object and assigns geometric coordinates to each object and the locations on the object; this process may require some geometric problem solving. For example, in the sample problem (P8) it is necessary to solve a triangle given two sides and an angle. Since the sizes of some of the objects may be represented by symbolic constants, the geometric coordinates of some points may contain algebraic expressions.

After the geometric model has been constructed, problem-solving programs are called to write equations which describe the interactions of the objects according to physical laws and to solve the resulting equations. There are specialist programs which write equations for particular objects (such as springs and rigid bodies), as well as more general programs which require, for example, that the sum of the forces at a point of attachment must be zero. The equations, which may contain symbolic constants as well as numbers and variables, are solved by a small symbolic manipulation package.

Answers to the problem are generated from the solutions to the equations by a small set of programs. These programs use information about the desired form of the answer which was saved when the problem statement was read. Answer generation may require small additional computations, and may require the generation of short English descriptions of constant terms or locations.

A picture model is generated to allow a diagram of the problem to be drawn. This model is similar to the geometric model, except that all dimensions of objects must be numerical. Sizes must be chosen for objects which have zero size in the geometric model (for example, a person who is represented as a point mass), and appropriate locations for such objects to be attached to other objects (for example, the feet of a person) must be chosen if they are unspecified. A picture generator calls appropriate routines to generate drawings of the objects in the problem, using the size, position, and orientation specified in the picture model.

The organization of the body of the paper follows the organization of the diagram in Figure 1.1. Chapter 3 describes the parsing programs and the parse structures which they produce. Chapter 4 describes the semantic programs which create the initial internal model from the parsed sentences. Chapter 5 describes frame creation and the construction of the geometric model. Chapter 6 describes problem solving and answer generation. Chapter 7 describes how the diagram is constructed. Finally, Chapter 8 examines the methodology of this work, indicated directions for future research, potential applications of this type of program, statistics of the program, and the problems involved in extending the program.

2. Review of Previous Work

This review of previous work is grouped into three sections: programs which solve problems stated in natural language, natural language processing, and theoretical work.

2.1 Natural Language Problem Solvers

2.1.1 Bobrow's STUDENT

The first natural language problem-solving program was the STUDENT system of Bobrow [Bobrow 68] for solving algebra story problems. The natural language processing of this program is based on pattern matching around key words and phrases. The phrases around the key words become the "variables" in the equations which are constructed from the sentences. Thus, in one of Bobrow's examples,

> If the number of customers Tom gets is twice the square of 20 percent of the number of advertisements he runs, and the number of advertisements he runs is 45, what is the number of customers Tom gets?

the two phrases "the number of customers Tom gets" and "the number of advertisements he runs" are treated as variables. This problem is thus treated as if it were stated

> If x is twice the square of 20 percent of y, and y is 45, what is x?

The pattern-matching rules break the input sentences into a possibly embedded set of "kernel sentences", in an order determined by priority values assigned to the keywords. In the above example, since "percent" has the highest priority, it would be processed first. There is a fairly direct transformation from English sentences into equations: in fact, the transformations are made upon the input sentences themselves until the sentences become the equations used in solving the problem. Large segments of the original sentences remain as "variables" in the equations.

When the equations constructed from the input are insufficient to find a solution, other equations can be retrieved (based on words in common with "variable" phrases) expressing general relationships, such as

(EQUAL (DISTANCE) (TIMES (SPEED) (TIME))).

Bobrow's program was impressive for its time (about 1965). However, this type of approach has definite limitations. The technique of transforming sentences directly into equations works only when the sentences express algebraic relationships among quantities. The "variable" phrases must be similar in each occurrence so they can be matched properly, and the key words must not be used in multiple ways which might confuse the pattern matcher. These limitations make it difficult to extend the techniques Bobrow used to more complex problem areas.

2.1.2 Charniak's CARPS

Charniak's CARPS program [Charniak 68] is a program for solving calculus rate problems. In many ways, it is an extension of Bobrow's STUDENT program. The analysis of the English input sentences is done by pattern matching which is slightly more sophisticated than that of STUDENT. The type of rate problem (distance or volume) is determined by the occurrence of certain key words in the problem statement. Two sets of patterns are used to analyze the sentences appearing in the two types of problems. Many of the patterns used are very ad hoc.

The CARPS program builds a structure (generally a single tree) containing the information derived from the problem statement. This structure is used to generate the equations required to solve the problem. Additional equations may be derived from "world knowledge", but this is again very ad hoc. Thus, while the problems solved by CARPS appear very impressive, the program is tailored so closely to this specific set of problems that it would be difficult to extend it to additional problems or problem areas.

2.1.3 Gelb's HAPPINESS

HAPPINESS [Gelb 71] is a program which solves basic probability problems stated in English. This program seems much like Charniak's: it builds a single tree structure representing a single problem, and selects a solution method based on the occurrence of keywords in the problem statement. The input sentences are broken into simple clauses and phrases by pattern matching. These simple clauses are then analyzed by a context-free grammar to extract the canonical verb and its voice, subject, and predicate. If certain key words (e.g., those referring to dice and coins) are found, a special search for possible modifiers of these words is made.

This program, like Charniak's, is tailored very closely to a small set of specific problem types. It would be difficult to extend a program using these techniques to handle a new problem area.

2.1.4 Heidorn's Simulation Programming System

The NLPQ system of Heidorn [Heidorn 72] accepts an English statement of a queueing simulation problem, and produces from it a program in the GPSS simulation language which will simulate the problem. The system is interactive: it requests additional information from the user when the problem statement is incomplete, allows the user to ask questions about the simulation model, and can generate a complete problem description in English from its internal model.

English sentences are parsed and generated from two interpreted phrase structure grammars augmented by some semantic programs. These grammars go down to the character level, and handle English morphology as well as phrase structure. The grammar is based in part on the theory of stratificational linguistics. The basic unit of storage in the internal model is the "record", which is computationally equivalent to a LISP atom with its property list.

This program represents an advance over those considered previously in this section. It uses a legitimate grammar to parse the input sentences, and can construct a model which expresses relationships among a number of objects. The grammar is specialized for simulation problems, and would have to be modified to extend the program to other areas. However, the performance of this system is quite impressive.

2.2 Natural Language Processing

2.2.1 Woods' Augmented Transition Networks

The Augmented Transition Network (ATN) of Woods [Woods 70] is a powerful formalism for representing grammars. The grammar of ISAAC, while written in "pure" LISP, is equivalent to an ATN grammar. A transition network consists of a set of nodes (representing states) and a set of directed arcs between the nodes which specify transitions between states based upon the input string being scanned. An ATN is augmented in several respects. The test associated with a state transition may be arbitrarily complex, depending on the previously parsed structure as well as the input. The test may be the name of another transition network, in which case control is given to that network at a lower level, effecting a "subroutine call" to the subordinate network. These calls may be recursive. Transition arcs may also be augmented by arbitrarily complex structure-building actions: The structures so built are passed among network levels in designated registers. If an attempted parsing of a subnet fails, the ATN interpreter automatically handles backup from the failure point and tries another possible transition. The automatic backup and clearly defined interface (via named registers) between network levels make the ATN a very "clean" formalism for writing grammars.

2.2.2 Winograd's SHRDLU

Winograd's widely known SHRDLU program [Winograd 72] allows a person to converse with a simulated robot about a "micro-world" consisting of various colored blocks on a table. The robot may be asked to perform actions such as moving blocks or building structures and to answer questions about the state of the micro-world or about its motivations for performing particular actions. The system employs a large grammar, based on Halliday's theory of Systemic Grammar. Much of the knowledge in the system is represented in the form of MICRO-PLANNER theorems. This makes it easy for programs to be generated to find the answers to questions about the world model, and allows a number of logical forms such as conjunction, disjunction, and quantification to be handled naturally. The theorem prover base is a source of considerable power for certain types of semantic operations. The semantics is made much easier by the small, finite world of very simple objects (colored blocks). Still, the SHRDLU system remains one of the largest and most powerful natural language systems produced to date, and its fame is well deserved.

2.2.3 Wilks' Preferential Semantics

The work of Wilks [Wilks 75] is unique among "artificial intelligence" approaches to natural language processing in that Wilks is interested primarily in machine translation, rather than in deep understanding of natural language by computer. However, there is an interesting parallel between the semantic templates used by Wilks and some of the semantic processing done in ISAAC, which is of course concerned with deep understanding.

In Wilks' system, a sense meaning of an English word is represented by a formula, which is a list of element names. The elements are approximately 70 semantic classes which roughly classify the entities, qualities, actions, etc. which occur in English sentences. Examples of such elements are MAN (human being), STUFF (substances), KIND (qualities), and CAUSE (cause to happen). These elements may be combined into a formula to represent a word meaning, as in (FLOW STUFF) for the word "liquid". A sentence is analyzed by trying to fit a template (which is a list of element types) to some of the possible sense meanings of the words occurring in the sentence. The templates are intended to represent the basic types of "messages" that people wish to convey in language. For example, the template MAN BE KIND would represent the class of messages in which the sentence "My sister is pretty" is included.

After (possibly several) templates have been fitted to a piece of text, "preferences" of parts of each template are examined to see if they are satisfied. A verb, for example, may prefer an animate subject. The template for which the greatest number of preferences are fulfilled will be chosen as the intended meaning; however, possible fillers for the template slots will be accepted even though they do not meet the preferences, provided that the template as a whole is the best match.

There is considerably more detail to Wilks' system which will not be covered here. We mention Wilks because there seem to be parallels between some of his techniques and techniques used in ISAAC. One such parallel is the use of rough semantic classes to distinguish between sense meanings of words, which we found to be particularly useful for determining preposition meanings. Others have certainly used rough semantic classes to distinguish sense meanings in special applications; Wilks' work is valuable for investigating this technique over a large subset of English.

A second parallel lies in the acceptance of a word (or larger unit) which fails to meet the preferences of the template which covers it. In the ISAAC system, this acceptance is an active process in which an acceptable interpretation must be constructed from the given unit. These processes are discussed in more detail later.

2.2.4 Simmons' Semantic Networks

The Semantic Network formalism of Simmons [Simmons 73; Simmons and Bruce 71] provides a powerful and convenient method for representing the elements of a sentence and the semantic relations (derived from a variety of syntactic forms) which hold between them. In effect, it produces an ordering of the arguments of a semantic grouping (such as a verb and its case arguments, modality, and optional modifiers) which is invariant over the various syntactic orderings which express the same relationships. Thus,

John gave Mary the book.
John gave the book to Mary.
The book was given to Mary by John.

would all generate the same semantic network structure. The semantic network formalism has been used for language generation [Simmons and Slocum 72] as well as parsing. [Simmons and Bennett-Novak 75] shows how these structures may be used to produce a small natural language understanding system with a minimum of effort.

The structures used by ISAAC in understanding sentences are a natural extension of Semantic Networks as used by Simmons. In order to handle multiple sentence discourse, links are made from tokens in individual sentences to the referents of the tokens in the problem model which is being constructed. Semantic interpretations are placed on some tokens as their meanings are determined. Particular semantic interpretations may be specified based on information from many different sources. Making an interpretation of a token may cause links to be made from that token to objects not mentioned in the sentence and may generate additional inferences about the relationships of the objects involved. These processes are discussed in detail in later chapters.

2.2.5 Schank's Conceptual Dependency

The Conceptual Dependency system of Roger Schank [Schank 73, 75] is a theory (embodied in a series of computer programs) which postulates that the concepts transmitted in natural language can be represented as complex structures based on a small number of primitive actions. The primitive actions are linked by named links to their case arguments (some of which are other primitive action groups) and to other groups to which they are related, e.g., causally. Some case arguments are mandatory, so that in "John hit Mary" we must infer an instrument in this case John's hand) used in performing the action.

While the structures and actions used by Schank are not very useful for physics problems, some of the concepts he uses (such as inferring a required semantic object when it is unspecified) are basic to almost any language understander. Schank's work is also important because he has defined a set of primitive concepts and actions which can be used to express a fairly wide range (though certainly not all) of natural language sentences.

2.3 Minsky's Frame System Theory

Minsky's frame system theory [Minsky 74] proposes that knowledge is organized (in humans and, potentially, in computers) in terms of interconnected elements called Frames.

> A frame is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.

A frame may have "slots" which can be filled by the particular "arguments" involved in an instantiation of the frame. There may be procedures associated with a frame to determine the suitability of proposed arguments and to infer values for those which are unspecified.

Minsky also makes some general comments about how frames may be used in computational linguistics.

> ... in understanding a discourse, the synthesis of a verb-structure with its case-assignments may be a necessary but transient phase. As sentences are understood, the resulting substructures must be transferred to a growing "scene-frame" to build up the larger picture.

Minsky's frame system theory has been immensely popular—so popular that many people are claiming that frames are exactly what they have been doing all along. There are many similarities between the processes and data structures used by ISAAC and the frame systems described by Minsky, and the term "frame" will be used in describing some of them. The interpretations given will of course be those of the author. The idea of frames is a powerful one, but the mechanics of their implementation remains a problem for research.

3. Parsing

3.1 Introduction

Parsing, as used in this chapter, means the process of assigning a structure to the linear string of words comprising a sentence so that the syntactic relationships among the words and phrases in the sentence are made explicit. The processes of relating the structures in the sentence to parts of the developing model of the problem and of determining the meaning of the structures will be treated in the chapter on Semantics. Obviously, there is no clear division between what is syntax and what is semantics; many constructions could be claimed to be either. In the sentence processing done by ISAAC, syntactic and semantic processing are frequently intermixed. We shall describe the two parts separately to make them easier to understand, while trying to indicate the points at which they interact. How to best organize the interactions of syntactic and semantic processes in a language understanding program remains an unsolved problem.

Although the parsing programs in ISAAC are written in "pure" LISP, their structure is strongly influenced by the Augmented Transition Network (ATN) formalism of Woods [Woods 70]. An ATN grammar allows sub-grammars for phrases to be called (recursively) as subroutines by other grammars. A grammar program may build structures which are passed back to the program which calls it. In case an attempted subgrammar fails, the grammar interpreter automatically backs up from the failure point and tries the next possible alternative which is specified. These features of ATN grammars are also present in the parsing programs of ISAAC. The grammar programs are organized as a set of parsing functions, most of which parse a single functional unit, such as a noun phrase. This organization in terms of functional units seems natural because it allows the grammar functions to communicate with each other by passing pointers to complete, well-defined functional structures. A noun phrase, for example, causes the production of a noun phrase token structure which has a standard form, independent of the function of the noun phrase in the sentence. Grammar functions which parse larger syntactic units, such as a clause, connect the smaller structures, such as noun phrase and verb phrase tokens, by means of named links which specify the relationships of the phrases in the sentence.

The structures which are produced by the parsing programs bear a strong resemblance to the semantic networks of Simmons [Simmons 73]. The grammar functions which parse the major phrases, such as noun phrases and verb phrases, produce "token" structures which represent the information in the phrase in a standard and readily accessible form. Other grammar functions, such as those which parse prepositional phrases and other modifiers, may make changes and add information to the modified token structures rather than creating new structures themselves. The links between token structures may specify semantic relationships (for example, that one noun phrase names a location on the object referred to by another noun phrase) as well as syntactic relationships. In some cases (e.g., with prepositional phrase modifiers), the semantics may be done at once, so that semantic links among the tokens are not needed. As semantic processing proceeds, the token network structure is elaborated by adding semantic interpretations to some tokens and by creating links between some tokens and the objects to which they refer in the program's model of the world. These semantic processes may render a token unnecessary and leave it unlinked to the rest of the structure. After all the semantic processing has been done, the information in the sentence has been transferred to the world model, and the network of tokens is no longer needed.

3.2 Control Structure of the Parser

The parsing programs are written as LISP functions, without using an additional interpreter as a Woods system does. Automatic backup and control of the scanner which points to the current position in the sentence being parsed are accomplished by a set of small functions which are called from within the individual parsing programs. These functions set the system registers (global variables) appropriately for the current state of the parser.

A sentence is represented internally as an ordered list of words. As the sentence is scanned from left to right, the global variable SENT is set to point to the current position in the sentence. The current word (or multi-word unit) being scanned is put into the * register. Thus, a grammar program could test whether the word currently under the scanner is "and" by using the LISP code (EQ * "AND), where the quotation marks are an abbreviation for the function QUOTE. The parsing of a sentence is initiated by setting SENT to the sentence and calling the function SET* to set the * register. When a grammar program wishes to move the scanner one position to the right, it does so by calling the function (=>). The next word to the right and the second word to the right may be gotten by using the functions (NEXT) and (NEXT2), respectively, without affecting the position of the scanner. The function CAT (category) is frequently used to test whether the word currently under the scanner is in a particular category, as defined in the lexicon. Thus, (CAT "ADJ) may be used to test whether the current word is an adjective.

Since the parser operates from left to right, it sometimes happens that a grammar program fails to find the type of phrase it expects, after it has moved the scanner from its initial position. For example, in parsing the sentence "To err is human", the parser might first attempt to parse a prepositional phrase. The preposition would be found, and the noun phrase parser would be called after moving the scanner. The noun phrase parser would find the verb "err", and so it and the prepositional phrase parser would fail. In order to handle such cases, it is necessary to be able to save the current position in the sentence so that the parser can back up and try something else when an attempted parsing fails. This is accomplished by calling three small functions, SAVE, SUCCESS, and FAIL, within each parsing function. Normally, a parsing function will execute (SAVE) as its first action and execute either (FAIL) or (SUCCESS), as appropriate, immediately before it exits. SAVE saves the pointers to the current point in the sentence on a push-down stack. In addition, it saves the current point in the list of generated atoms, so that any atoms generated by a function which later fails can be deleted. SUCCESS removes one set of pointers from the stack: since the attempted parsing was successful, these pointers are no longer needed. FAIL restores the pointers to the sentence to their original position, and calls SET to restore the * register. In addition, it releases any atoms which may have been generated by the function which failed.

In order to illustrate how the parsing functions are actually written, a simple function to parse a noun phrase (using the same conventions as the parsing programs of ISAAC) is shown below. This program parses a simple noun phrase consisting of an optional determiner, zero or more adjectives, and a noun. The program succeeds and returns True if it finds such a phrase; otherwise, it restores the pointers using FAIL and returns NIL. No structures are built by this program, but it is easy to see how structure-building code could be added.

```
(NP  (LAMBDA ( ) (PROG ( )
     (SAVE)
              (COND  ((CAT "DET) (=> )))
         A    (COND  ((CAT "ADJ) (=> ) (GO A))
                     ((CAT      "NOUN)     (=> )     (RETURN
                     (SUCCESS)))
                     (T (RETURN (FAIL))))
         )))
```

This program accepts a noun phrase equivalent to that accepted by the following grammar:

```
<np> → <det> <npl>
<np> → <npl>
<npl> → <adj> <npl>
<npl> → <noun>
```

Using the Woods ATN formalism, such a program could be written as follows:

```
(NP/ (CAT DET T (TO NP1))
     (TST T T (JUMP NP1)))
(NP1 (CAT ADJ T (TO NP1))
     (CAT NOUN T (TO NP2)))
(NP2 (POP T T))
```

Our method of writing parsing functions requires the writing of slightly more code than is required for a Woods interpreter system, but it avoids the overhead of interpretive execution.

The function SET*, which sets the value of the * register, checks for multiple-word units, and replaces them with single words in the * register. "As much as", "center of gravity", "cross section", "point of application", and "so that" are recognized as multiple-word units. These groupings could have been handled by other methods, but replacing them by a single "constructed" word is a convenient way to do it. A large parsing system would need to be able to back up in case the multiple-word interpretation was incorrect; in our limited field of physics problems no such ambiguities occurred. Becker [Becker 75] has argued that such groupings play a major role in language.

Values are passed between levels of the grammar using the normal LISP conventions of function arguments and returned values. A returned value of NIL always indicates failure of a grammar program. A grammar program which succeeds may return a generated token atom (as in the case of a Noun Phrase), or it may attach its results to existing atoms and simply return True (as in the case of a Prepositional Phrase). Some grammar functions have no arguments, but others (such as the Verb Phrase) have quite a few.

3.3 Data Structures Produced During Parsing

As a sentence is parsed, the grammar programs create a set of interlinked nodes representing the major phrases (primarily Noun Phrases and Verb Phrases) of a sentence. These networks initially bear a strong resemblance to the Semantic Networks of Simmons [Simmons 73]. As semantic processing of the sentence progresses, modifiers of the nodes are removed or changed in form, semantic interpretations are added, and links are made from the nodes to objects and relations in the developing model of the problem. Finally, after execution of the verb semantics, the network is discarded.

Each node in the parse network is a GENSYM atom whose name is TOK followed by a number. Features of the node (also called a "token atom" or "token") are stored on its property list. The "main" word of the phrase (usually, but not always, a word from the sentence) is stored under the indicator TOK. The type of phrase is stored under the indicator LFRAME (Linguistic Frame); the possible types of LFRAMEs are NP (Noun Phrase), VP (Verb Phrase), QNP (Question Noun Phrase), and RELNP (Relative Noun Phrase). The noun phrase "each end" in P3*, for example, would generate the following token:

```
TOK89    TOK        END
         LFRAME     NP
         NBR        (NS)
         MODS       ((QNTFR EACH))
         SFRAME     LOCPART
         SEMOBJ     (SCAFFOLD85)
         RFNT       (LOC91 LOC90)
```

The first four items on the property list of the token are created by the parsing program. NBR is the Number (Noun Singular), and MODS is a list of modifiers, in this case the quantifier EACH. The remaining property list items are added during semantic processing: SFRAME (Semantic Frame) is LOCation PART; SEMOBJ (Semantic Object) is a link to the object in the problem model which the location refers to, in this case the scaffold SCAFFOLD85. RFNT (Referent) is a list of pointers to the items in the problem model to which the phrase refers: the locations LOC91 and LOC90. When the semantic function for the verb is executed, it will deal directly with the Referents of the phrase, independent of the syntactic construction in the original sentence which caused those referents to be selected.

3.4 Noun Phrase Parsing

In this section we will examine in some detail the parsing of the noun phrase and its modifiers.

3.4.1 Basic Noun Phrase

A flowchart of the NP parsing program is shown in Figure 3.1. A flowchart is used to describe the program because a transition net of this size would be unwieldy, and because a flowchart can more closely follow the actual program structure. A few nonstandard conventions are used in the parser flowcharts in this chapter. A test consisting of a word in capital letters indicates a test of whether the word currently under the scanner (the word in the * register) is in that category. (NEXT) indicates the next word to the right, and (NEXT2) indicates the second word to the right. The symbol => appearing next to a line indicates that the scanner is moved to the right along that control path. The symbol ++ indicates that the right part is appended to the left;

* References to the example problems are denoted by the letter P followed by the problem number.

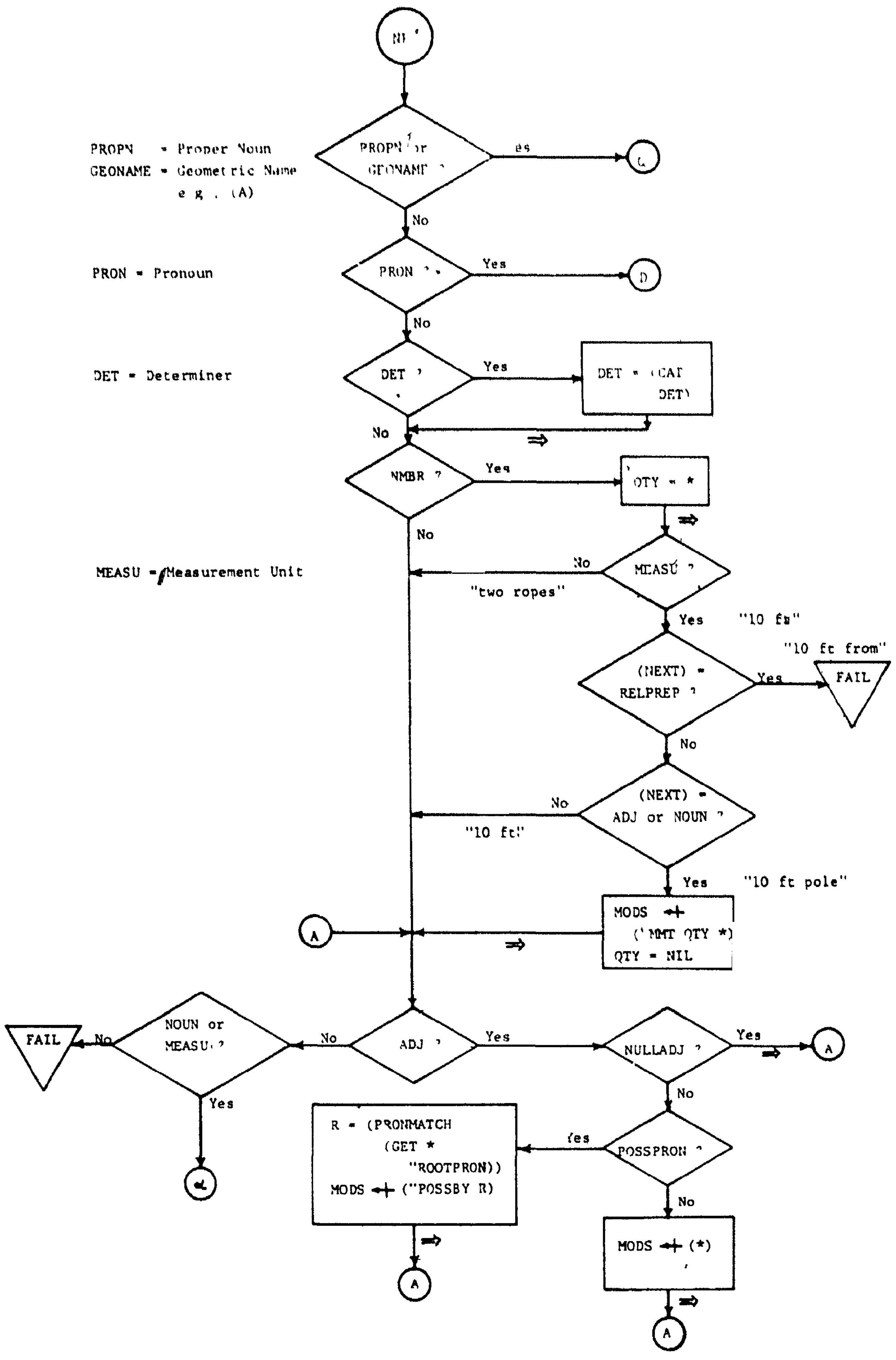


Figure 3.1 (3 pages): Noun Phrase Parsing Program

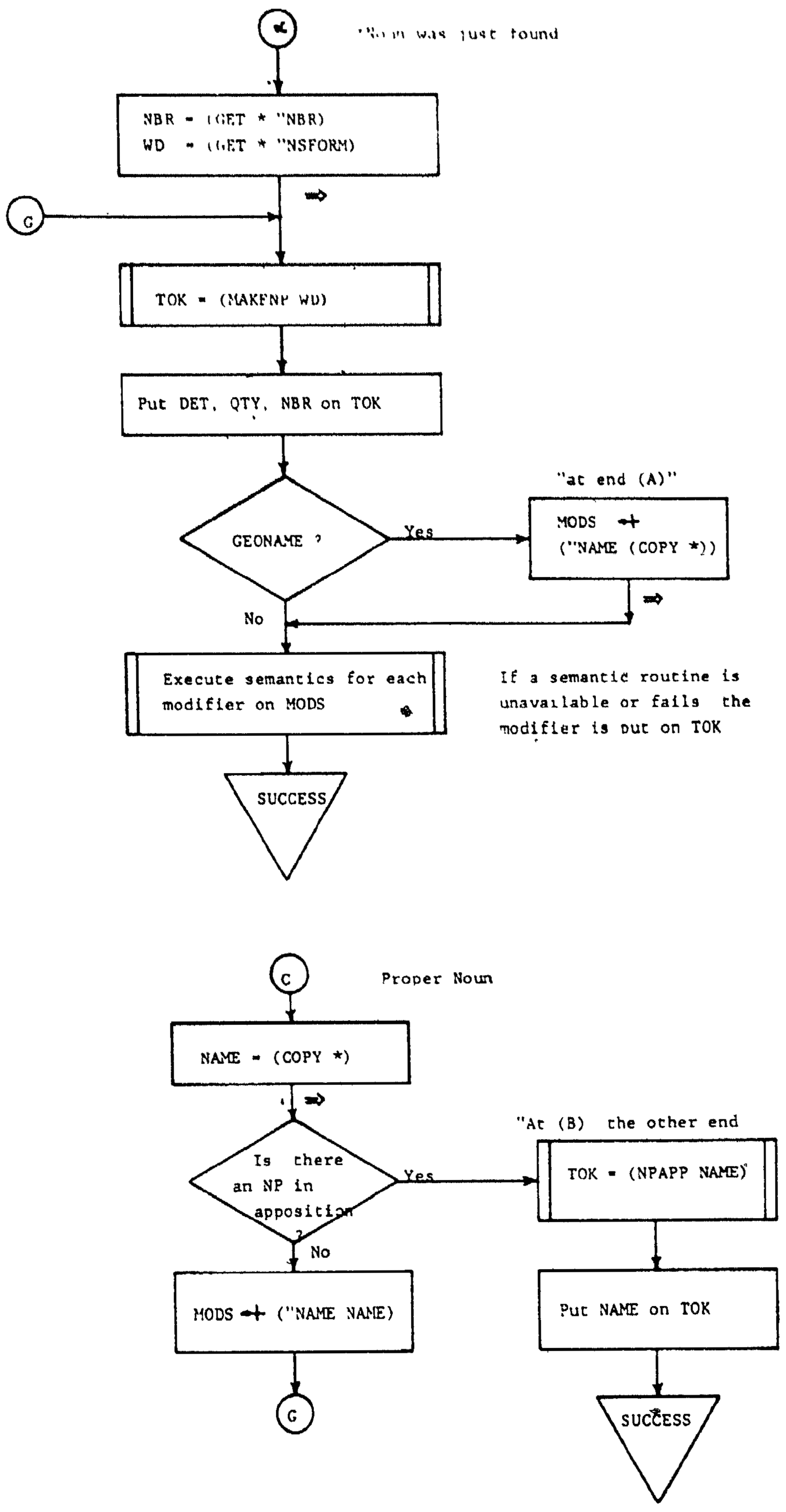


Figure 3.1 (page 2)

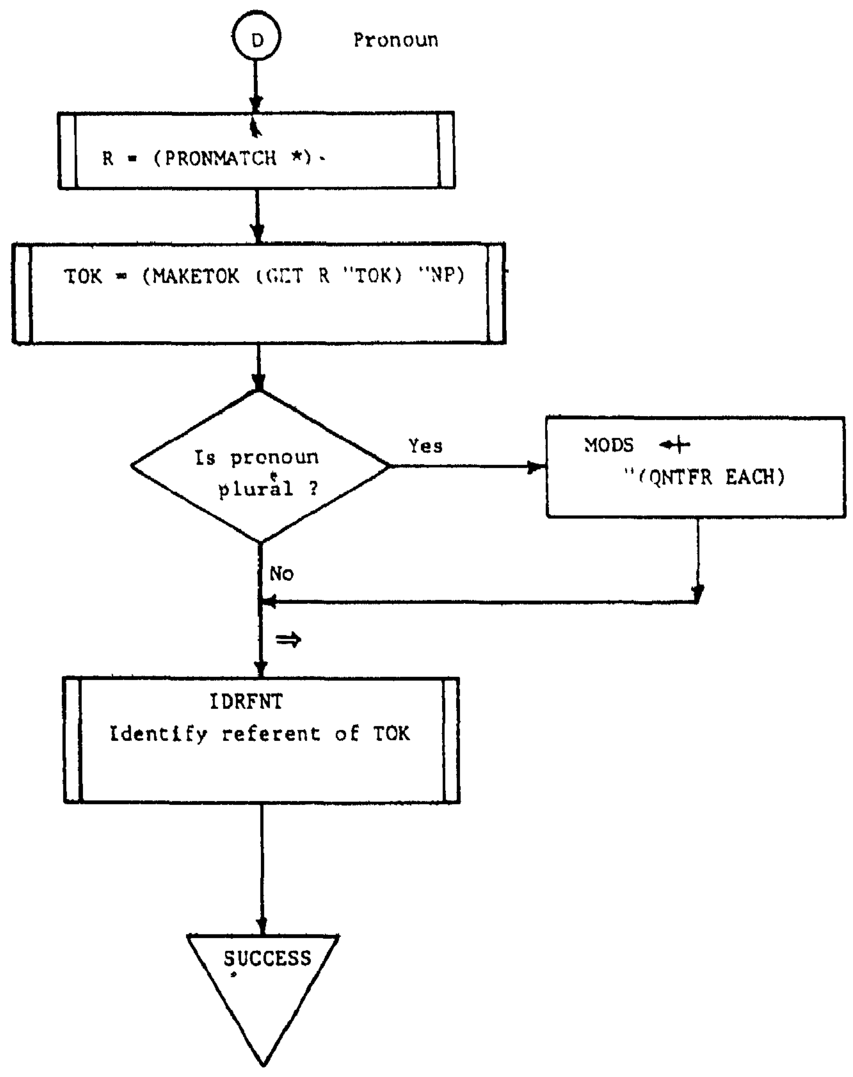


Figure 3.1 (page 3)

generally, A ++ B is implemented as

(SETQ A (NCONC A (LIST B))).

Phrases in quotes next to control paths are examples of phrases which would follow the indicated paths.

The initial tests in the flow diagram test for proper nouns, geometric names, and pronouns, which are handled separately. [Geometric names, as in "AT END (A)", are represented in LISP as lists containing the names; in the original sources from which the problems were taken, such names were written as italic capitals.] The determiner, if present, is saved. A series of tests separates the use of a measurement (e.g., "10 ft") as a noun phrase by itself or as a modifier ("a 10 ft pole"), while prohibiting it if it precedes a relative preposition (as in "10 ft from . . .") since this form is more easily handled as part of the prepositional phrase. Adjectives which are marked NULLADJ are ignored. Thus, "a tapering wooden telegraph pole" (P11) is treated the same as "a pole". This is one of the few cases in the parser where information from the problem statement is ignored. Possessive pronouns are rewritten at once; the referent of the corresponding root pronoun is found, and a modifier of the form (POSSBY referent) is constructed. This modifier retains the ambiguity of the type of possession. Not surprisingly, there is considerable similarity between the semantics of POSSBY and some of the sense meanings of OF. Thus, for example, "its end" and "the end of the lever" will be reduced to an identical form when processed semantically.

When the noun is found, a token atom is created for the noun phrase, usually using the singular form of the noun as the token name. In some cases, however, an expanded definition is used, resulting in the use of a different token name and the generation of additional modifiers. Thus, PAUL becomes PERSON, (SEX MALE), (NAME PAUL) and BOY becomes PERSON, (SEX MALE), (AGE YOUNG). This expansion eases the identification of the same object when it is referred to by different words; the identification of these two tokens will result in the inference that Paul is young. The modeling of words as carriers of modifiers to be applied to their root concepts is an interesting area of research; [Simmons and Amsler 75] are investigating this type of modeling for verbs of motion and communication.

After the noun token is made, an attempt is made to execute the semantics of each of the modifiers which have been found. Some modifiers will make changes directly to the NP token; others will create new modifiers which are saved for later processing. "Both", for example, will create modifiers equivalent to "Each of the two . . .".

The pronoun matching algorithm which is used is very simple. A pronoun which was previously used is matched to the same referent as before. Otherwise, the last-mentioned candidate which matches the pronoun in number and is appropriately human or inanimate is chosen. This technique is fairly crude, but it worked for this class of problems. In general, finding pronoun referents can be very difficult. [Charniak 72] considers this problem in some detail.

3.4.2 Noun Phrase Compounds and Modifiers

Conjunctions and modifying phrases introduce many potential ambiguities into the parsing of a sentence. In a noun phrase containing two prepositional phrases, for example, the second prepositional phrase (PP) might modify either the top-level noun phrase or the noun phrase in the first PP. A conjunction between two noun phrases might join them into a compound noun phrase, or it might connect two top-level clauses containing the noun phrases. Although syntactic constraints may select the correct interpretation in some cases, in many other cases the choice can be made only on semantic grounds. For example, in

> Lowering the level of the lake allows city officials to kill weeds and residents to repair their docks.

we must use semantic interpretations to reject the possibility that "weeds and residents" is a compound noun phrase. People seem to make these choices easily and correctly the first time they read or hear a sentence; only rarely do they have to back up and re-parse a sentence in order to interpret it correctly. The parsing programs of ISAAC rely heavily on semantic tests to reject incorrect combinations of phrases.

A noun phrase may be modified by a prepositional phrase, an adjective phrase, or a dependent clause. In each case, the parser for the modifying phrase is called with the NP token as an argument. The modifying phrase parser may reject the combination on semantic grounds even though the appropriate syntactic constituent is found. This is especially important in the case of prepositional phrases. Compound modifiers, as in "a uniform scaffold 12 ft long and weighing 100 lb" (P5), are permitted.

Conjoined noun phrases are required to all be members of the same semantic class, which may be one of the set PERSON, PHYSOB (physical object), LOCNAME (location name), ATTRNAME (attribute name), or MEASU (measurement unit). Pronouns are prohibited as members of compound noun phrases. These tests handled almost all cases which occurred in the set of test problems. One pathological sentence required additional treatment:

> If it is placed on the edge of a block 1.5 m from the light end and a weight of 750 nt added to the light end, it will be balanced. (P14)

Since the auxiliary "is" is omitted in the second clause, "added . . ." could be considered a dependent clause modifying "weight", and "block" and "weight" could be combined as a compound noun phrase under the above rules. This problem was solved by a semantic test associated with the preposition "of" which prohibits a compound object noun phrase for such cases. This is not a very pleasing solution. People probably accept "edge of a block" as a well-formed unit before reaching the second clause, and thus do not consider combining "block" and "weight". The depth-first operation of this parser allows it to go fairly far afield in such cases; additional semantic tests to allow some constituents to be combined earlier would be a desirable, but difficult, improvement.

3.4.3 Noun Phrase Variants

There are three small parsing programs which accept variants of noun phrases. THERENP accepts "there" as a noun phrase in cases such as "there is . . .". QNP accepts a noun phrase beginning with a question word, as in "what force . . .". RELNP parses a "relative noun phrase" containing "as much as", as in "the man supports twice as much as the boy" (P7). The multiplication factor is saved, and a link is made to the noun phrase involved in the comparison.

3.5 Verb Phrase Parsing

3.5.1 Basic Verb Group

The verb group, which is parsed by the program VG, consists of a set of auxiliary verbs, a main verb, and optional adverbs. The flowchart of VG is shown in Figure 3.2. Since tense and modality are not needed for our type of physics problems, the auxiliary verbs are ignored except for determining whether the verb group is active or passive. Other authors (for example, [Winograd 72]) have given procedures for determining verb tense from the auxiliary verbs.

The program VG has six arguments. NPHD is a noun phrase token which is the syntactic subject of the verb. VPHD (if specified) is a verb phrase token which is either the first part of a compound verb phrase or the initial auxiliary verb which is separated from the rest of the verb group in a question. CMPND is a flag which is true if the verb group is part of a compound verb phrase. DCLF is true if the verb group is part of a dependent clause; DCLP is true if the dependent clause construction is passive. QFLG is true if the verb group is a top-level verb group in a question.

The flowchart of VG is fairly straightforward. If a previous verb phrase is available from a separated verb group in a question, it is deleted and incorporated into the main verb group. The syntactic subject is attached to the verb phrase as subject or object depending on whether the verb is active or passive. This transformation frees the verb semantic programs from having to concern themselves with the voice of the verb. In a compound verb phrase without a subject (object if passive), the corresponding phrase from the first verb phrase is used. Thus, in "John was tarred and feathered", "John" would be used as the object of "feathered".

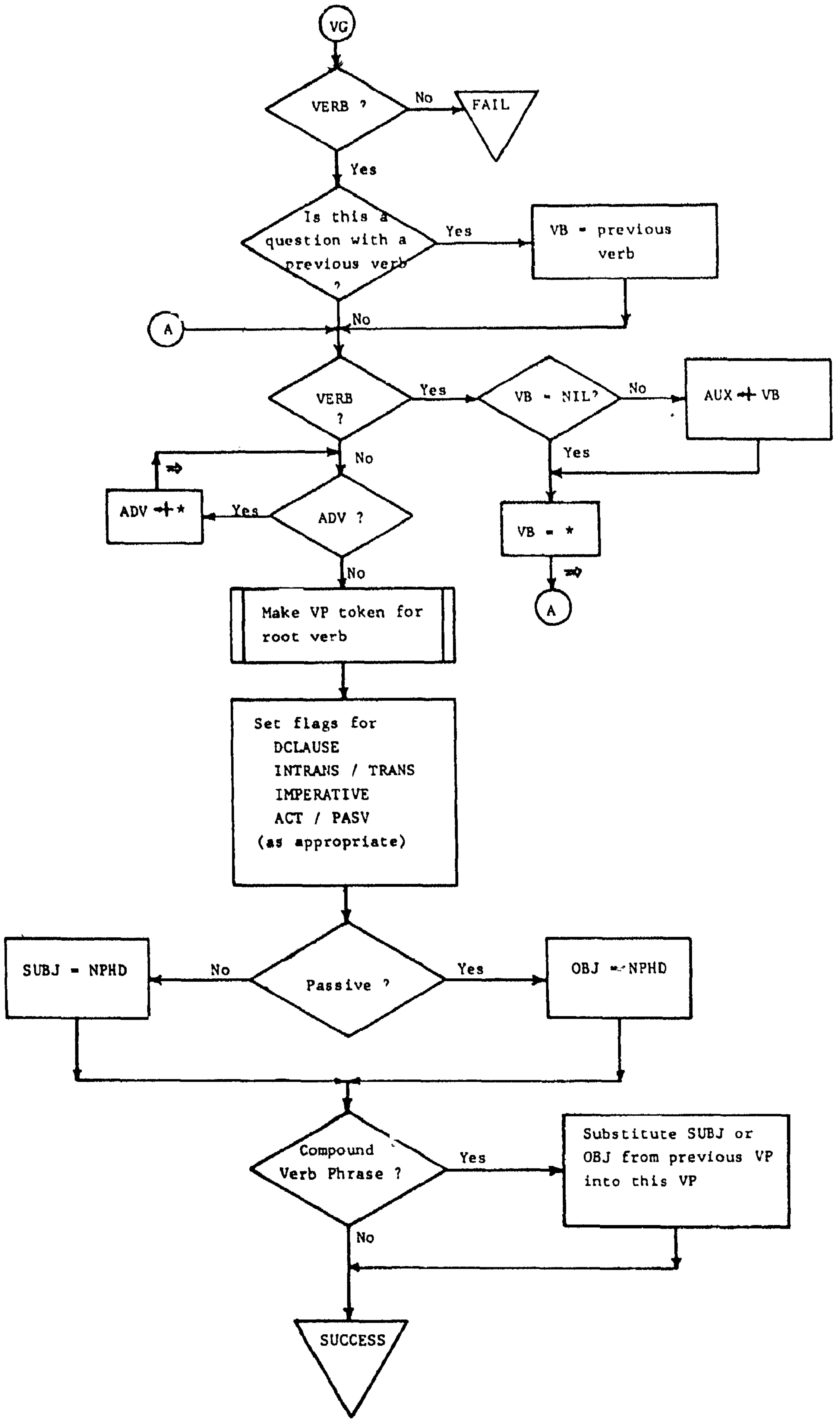

VG
VERB ?
No
FAIL
Yes
Is this a question with a previous verb ?
Yes
VB = previous verb
No
A
VERB ?
Yes
VB = NIL?
No
AUX ← VB
Yes
VB = *
⇒
A
No
ADV ?
Yes
ADV ← *
⇒
No
Make VP token for root verb
Set flags for DCLAUSE INTRANS / TRANS IMPERATIVE ACT / PASV (as appropriate)
Passive ?
No
SUBJ = NPHD
Yes
OBJ ← NPHD
Compound Verb Phrase ?
Yes
Substitute SUBJ or OBJ from previous VP into this VP
No
SUCCESS

Figure 3.2: Verb Group Parsing Program

3.5.2 Verb Phrase

A flowchart of the verb phrase parsing program, VP, is shown in Figure 3.3. VP first parses a verb group by calling VG, then collects the remaining predicate phrases and modifiers and attaches them to the verb phrase token. These phrases include the syntactic object noun phrase or adjective phrase, an infinitive verb phrase object (as in "they want to go . . ."), and prepositional phrases or adverbs modifying the verb.

After parsing a verb group, VP calls VPMODX to collect verb modifiers (adverbs and prepositional phrases). An infinitive verb phrase object is collected if present and attached to the verb phrase token under the indicator INFOBJ. If a question is being parsed, the subject and the remainder of the separated verb group are collected. A prepositional phrase on HOLD (that is, one which occurred at the start of the sentence and could not be attached to anything, e.g. "At (B), the other end of the pole, there is . . ." (P15) is attached to the verb phrase if possible. The predicate noun phrase or adjective phrase is collected, along with any remaining modifiers. If the verb phrase is part of a dependent clause, it is required to contain more than just a verb. A dependent clause (DCLAUSE) is attached to the token of the phrase it modifies.

3.6 Prepositional Phrase Parsing

The structure of the prepositional phrase is fairly simple. In addition to the usual PP consisting of a preposition and noun phrase, the PP parser accepts a phrase involving a measurement and a preposition and noun phrase (as in "10 ft from one end") as a single prepositional phrase. Both types may involve question phrases, as in "at what point" (P7) and "how far from the center" (P20).

The PP parser behaves differently from the other parsing programs in that it saves a well-formed result which cannot be attached to the head token which was specified, due to semantic constraints. If the PP parser is called again to reparse the same phrase (as it surely will be), it applies the semantic tests to its previous result and the new head token. This not only saves the work of reparsing an identical phrase twice, but more importantly, it prevents side effects which occur during the parsing from being repeated. These side effects (such as the creation of a new object in the model of the problem) violate the restriction on a pure Woods net parser that all results be passed between programs in designated registers; hence, the effects are not undone when backup is made from a failing parse attempt. We could make all such actions reversible, as in CONNIVER [McDermott and Sussman 72], but such an approach exacts a high penalty in computational overhead. Our approach is probably safe for prepositional phrases, which cannot in general be parsed as anything else. The pure Woods net approach makes it difficult to mix syntactic and semantic processing. More research is

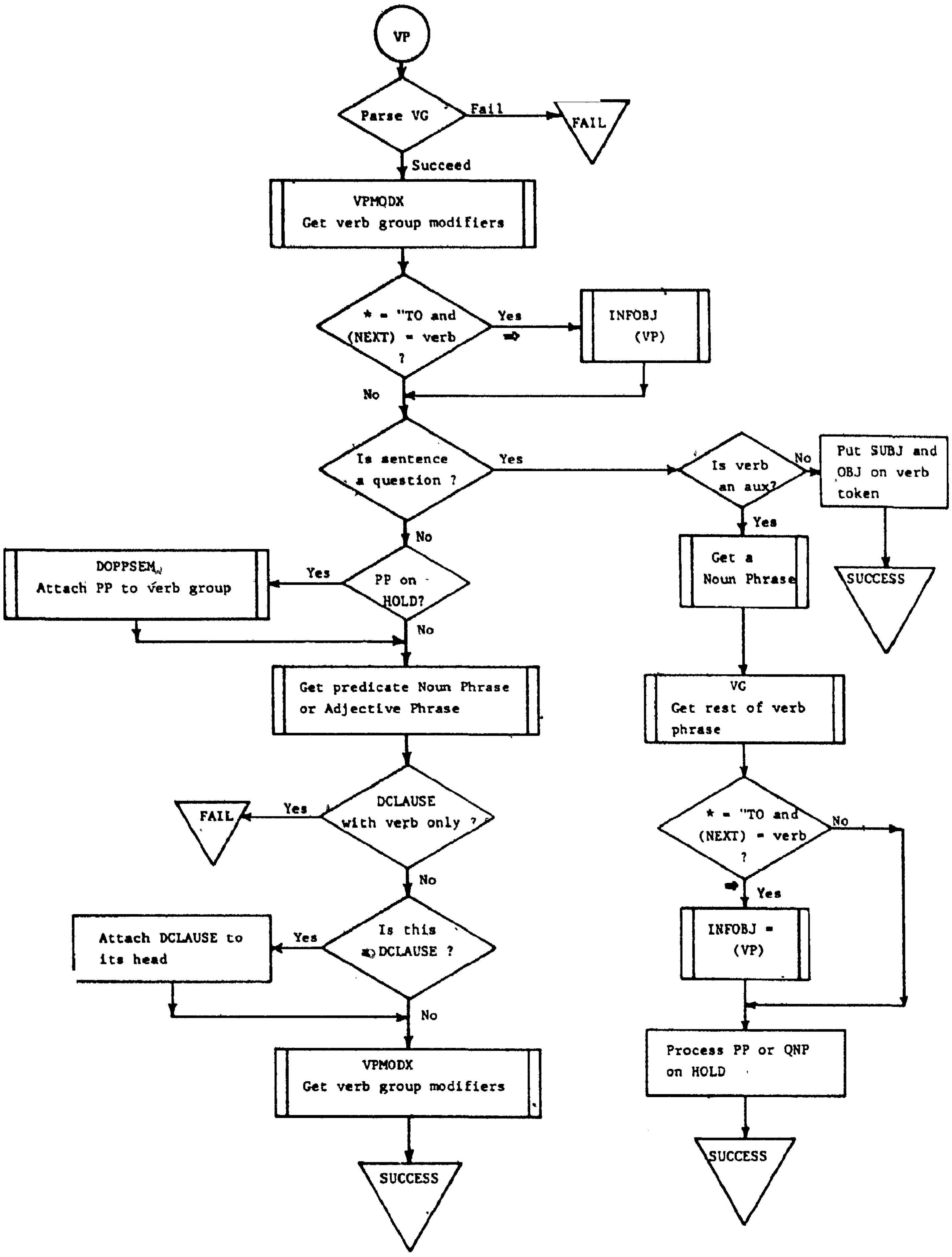


Figure 3.3: Verb Phrase Parsing Program

needed on ways to intermix the two and still be able to back up when necessary, without incurring too much overhead.

Preposition semantics are executed, when possible, as soon as the prepositional phrase has been parsed. In some cases, the semantic routine will elect to delay the semantic processing. In these cases, the PP is saved on the head token under the indicator PPS for later processing.

3.7 Clause Parsing

The clause parsing programs, CLAUSE and QCLAUSE, are relatively simple programs which accept the several forms of declarative, imperative, and question clauses. An initial prepositional phrase, if present, is placed on a HOLD list for later processing. A noun phrase or question phrase is parsed and then used as an argument for calling the verb phrase parser. The result of parsing a clause is the verb phrase token, which contains pointers to its various arguments. This verb phrase token is passed as an argument to the verb semantics driver, EXVBSEM, which completes semantic processing of the sentence.

Figure 3.4 shows the structure formed after parsing and semantic processing of a complete sentence. Much of the information in the structure is produced by the semantic programs after parsing, but we will describe it briefly as an introduction to the semantic processes. The root of the parse tree (the value returned by the SENTENCE parsing function) is TOK185, the verb phrase token for the main verb of the sentence. The object of the verb is TOK181, which was the syntactic subject (since the verb phrase is passive); the subject (agent) of the verb is TOK186, which was introduced by the preposition "by". The semantic routine for "by" simply attached its object phrase token, TOK186, to the verb phrase token as the subject of the verb; hence, there is no need for "by" to appear anywhere in the structure. TOK181 is the noun phrase token produced from the initial noun phrase of the sentence; it is a TOKen of the word SCAFFOLD, is a Linguistic FRAME of type NP (Noun Phrase) has an INDEFinite DETerminer, and has a NBR (number) of NS (Noun Singular). The modifier "12 ft long" has been converted to the form (LENGTH 12 FT); the same modifier form would be produced for the phrase "a 12 ft scaffold". TOK181 is the syntactic subject of a DCLAUSE (Dependent CLAUSE) whose verb token is TOK182. The SFRAME (Semantic FRAME) interpretation of TOK181 is PHYSENT (PHYSical ENTity), and its RFNT (Referent) is SCAFFOLD184, which is an object in the model of the problem. The remaining tokens shown in the figure have a similar structure. The modifier "vertical" of TOK186 has been converted to the form (ROTN 90); this token has two referent objects. The modifier "its" of TOK188 was converted to a modifier of the form (POSSBY SCAFFOLD184), which was semantically processed to make TOK188 a

LOCPART (LOCation/PART) SFRAME whose SEMOBJ (SEMantic OBJect) is SCAFFOLD184; identification of the location referents of TOK188 yielded the two locations LOC190 and LOC189, which are locations on SCAFFOLD184 in the model of the problem. Since TOK188 was the object of a preposition, semantic processing of the prepositional phrase transferred its referents to a modifier of the verb phrase TOK187; this left TOK188 unconnected to the rest of the structure.

3.8 Conclusion

The computer time required for parsing and semantic processing averages about one second per sentence, running on a CDC 6600 and using interpreted LISP. The parsing programs constitute only about 15% of the total; the semantic programs are twice as large. Syntactic processing is thus a relatively small part of the complete process of language understanding. On the other hand, this program has convinced the author that even in so constrained and well-defined an area as physics problems, syntactic processing cannot reasonably be isolated and done without recourse to semantic tests, some of which ultimately involve reasoning based on the particular facts which are known about the objects being discussed.

"A uniform scaffold 12 ft long and weighing 100 lb is supported horizontally by two vertical ropes hung from its ends."

```
TOK181                                      TOK182
TOK       SCAFFOLD                          TOK       WEIGH
LFRAME    NP                                LFRAME    VP
DET       INDEF                             MAINVB    WEIGHING
NBR       (NS)                              DCLAUSE   *T*
MODS      (UNIFORM (LENGTH 12 FT))          INTRANS   *T*
DCLAUSE   (TOK182)                          ACT       *T*
SFRAME    PHYSENT                           SUBJ      TOK181
RFNT      (SCAFFOLD184)                     COMP      TOK183

TOK183                                      TOK186
TOK       LB                                TOK       ROPE
LFRAME    NP                                LFRAME    NP
QTY       100                               QTY       2
                                            NBR       (NPL)
TOK185                                      MODS      ((ROTN 90))
TOK       SUPPORT                           DCLAUSE   (TOK187)
LFRAME    VP                                SFRAME    PHYSENT
MAINVB    SUPPORTED                         RFNT      (ROPE192 ROPE191)
AUX       (IS)
TRANS     *T*                               TOK188
PASV      *T*                               TOK       END
OBJ       TOK181                            LFRAME    NP
SUBJ      TOK186                            NBR       (NPL)
                                            SFRAME    LOCPART
TOK187                                      SEMOBJ    (SCAFFOLD184)
TOK       HANG                              RFNT      (LOC190 LOC189)
LFRAME    VP
MAINVB    HUNG
DCLAUSE   *T*
INTRANS   *T*
PASV      *T*
OBJ       TOK186
MODS      (LOC FROM (LOC190 LOC189))
```

Figure 3.4: Structures Produced for a Complete Sentence

It may be helpful at this stage to realize that the primary form of mathematical communication is not description, but injunction. In this respect it is comparable with practical art forms like cookery, in which the taste of a cake, although literally indescribable, can be conveyed to a reader in the form of a set of injunctions called a recipe.
—G. Spencer Brown

4. Semantics

4.1 Introduction

Semantics, for our purposes, is the process of constructing the meaning of a sentence: the process of relating the objects in the sentence to objects in the world model of the reader, and of updating the world model to reflect the meaning of the sentence. The sentence itself is not a description of the meaning, but rather a set of injunctions, a recipe which can be followed to construct the meaning from what the reader already knows.

As the above definition implies, the way in which a sentence is interpreted depends strongly on the knowledge, intelligence, and inclinations of the reader. As is well known, different readers will interpret the same text (even in physics problems) in different ways. A semantic interpretation of a sentence may be viewed as satisfactory or unsatisfactory for a particular purpose, but it would be difficult to judge it as "right" or "wrong."

Updating the world model to reflect the meaning of a sentence can be a very involved process, since the meaning of a single sentence can have many consequences. In our physics problems, these deductions do not propagate very far beyond the immediate understanding of a sentence during the time when the sentences are being read. In this chapter, then, we will primarily discuss "linguistic semantics," which we may define as the semantic processing up to the point at which the parsing of a sentence may be discarded. This distinction is well defined within the computer program. Under this heading there are a number of distinct semantic processes: determining the meaning of ambiguous words and phrases; finding anaphoric referents (such as pronoun referents) and elliptical referents (such as the physical object referred to when a location is named alone as in "one end"); determining the meaning of groups of words whose meaning in combination is more than a combination of their individual meanings; determining the meanings of modifiers of nouns and verbs and saving the meanings so that they can be effective at the proper place in the processing; determining whether an object or location mentioned in a sentence is a new one, or whether it refers to one mentioned previously; adding objects and relations to the world model, and updating existing ones to reflect new information; expanding the model of an object so that its subparts may be referenced; testing a modifier to determine whether it can reasonably modify a given phrase (which may require reasoning based on the particulars of the world model); interpreting an object of a given type as an object of a desired type (for example, interpreting an object as a location or vice versa). All of these processes will be discussed in this chapter.

4.2 Preliminary Modifier Processing

Adjective and adverb modifiers of noun and verb phrases frequently have their effects at a relatively late stage of semantic processing: the identification of the referent of a noun phrase, or the execution of verb semantics. These modifiers must therefore be saved for later reference. In some cases, a semantic routine will be associated with the modifier itself; in other cases, it is more convenient for a larger routine to look for the existence of certain modifiers to guide its processing. Preliminary modifier processing puts modifiers of certain classes into a standard form so that they will be easy to identify or so that a single semantic routine can be used for the whole class. In some cases, different meanings for a modifier may be selected depending on the modified phrase.

Adjectives such as "one", "other", "first", and "second" are put directly on the noun phrase token under the indicator DET2. These modifiers are referenced in determining the referent of the noun phrase. Adjectives such as "heavy", "left", and "upper" are converted to modifiers of the form (SELECT adj); they are used in selecting a particular referent from several possible ones. Quantifiers such as "each" become modifiers of the form (QNTFR adj). Adjectives such as "horizontal", "vertical", and "upward" are converted to rotation modifiers of the form (ROTN ang), where "ang" is the appropriate angle.

Adjective phrases indicating measurement (as in "a 10 ft pole" or "a pole 10 ft long") are converted to modifiers where the measured quantity is made explicit, e.g., (LENGTH 10 FT). When the referent of the noun phrase is found, the modifier is transferred to the property list of the referent. In the case of unspecified force measurements, tests are made on the modified noun phrase to determine the measured quantity. Thus, a 150 lb man is a man whose weight is 150 lb, while a 150 lb force is a force vector whose magnitude is 150 lb.

4.3 Preposition Semantics

Preposition semantics is an interesting area. A single preposition can have a number of sense-meanings (as many as seven in our set of physics problems) depending on the types of objects it connects. The actions required of the semantic routine are in general quite different for each sense-meaning; for our purposes, sense-meanings are differentiated by the different actions required to process them adequately. Discrimination net tests based on rough semantic classifications of the phrases connected by the preposition were found to be adequate to distinguish the preposition sense-meanings in our sample problems. We shall discuss in detail the semantic processing for some prepositions, and then compare our sense-meaning classifications with dictionary classifications and postulate that techniques similar to ours may be useful for machine translation of prepositional phrases.

4.3.1 Semantics of the Preposition "OF"

The sense-meaning classifications for the prepositions were determined by listing the occurrences of each preposition together with the modified (or "head") phrase and the object phrase of each. Occurrences which seemed to be of the same semantic class were grouped together, and a set of discrimination net tests was developed which would distinguish between preposition uses in each of the different classes. Using this procedure, seven distinct sense-meanings of the preposition "OF" were found in our small sample of twenty physics problems—a surprisingly large number considering that the problems are all of a similar type. The seven sense-meaning classes are listed below with examples. Although the classes were determined from our physics problems, it is easy to think of examples of each class which are in common usage and are not limited to the narrow area of physics problems.

1. < quantifier> OF < objects>	each of the ropes
2. < measurement> OF < value>	a length of 10 ft
3. < object> OF < value> < attribute>	a pole of uniform cross section
4. < location> OF < object>	the left end of the lever
5. < attribute> OF < object>	the weight of the lever
6. < group> OF < objects>	pair of legs
7. < part> OF < object>	hinges of a door

The semantic classes for the head and object phrases are given for each sense-meaning in the left-hand column; the discrimination net at the beginning of a preposition semantic routine uses tests for these semantic classes to determine the proper sense-meaning for a given use of the preposition. Once the proper sense-meaning has been determined, the processing required is fairly simple. For sense-meaning 1 of "OF", < quantifier> OF < objects>, the quantifier token is replaced by the object token, and the quantifier is made a modifier of the token; thus, "each of the ropes" is put into the same form as "each rope". For sense-meaning 4, < location> OF < object>, the SFRAME (Semantic Frame) of the head is set to LOCPART, and the referent of the object phrase is put on the head token under the indicator SEMOBJ

(Semantic Object). [The process of referent identification is discussed in a later section.] In cases such as this one, the determination of the preposition sense-meaning also serves to determine the proper semantic frame for the head phrase. The prepositional phrase itself serves to supply one of the arguments of the semantic frame. Just as parsing makes explicit the syntactic relations which hold among the words in a sentence, the reduction of phrases to their semantic frame form makes explicit the semantic relations which hold among the objects referred to (explicitly or implicitly) by the phrases. The semantic frames constitute a set of standard forms into which phrases representing similar meanings are translated; thus, numerous different ways of expressing the same meaning can all be translated into an identical semantic frame form.

In the case of sense-meaning 7, < part> OF < object> , a special semantic routine may be called into play to define the parts and their relation to the object they are part of. In our example, "the hinges of a door" (P9), the correct formulation of the problem requires the use of world knowledge that a door has two hinges (if the number is unspecified) which are arranged vertically and attached to the door on one side. This pragmatic knowledge is contained in a semantic routine for defining parts of doors. By representing the knowledge in this way, it is possible to refer to the parts of a complex object if necessary without expanding the internal model of the object into its parts if they are not referenced.

In our sample problems, there was only one case where a prepositional phrase modified a conjoined noun phrase: "magnitude, direction, and point of application of the equilibrant" (P12 and P15). In this instance, the prepositional phrase "of the equilibrant" should be assumed to modify each of the three conjoined phrases; this is handled within the semantic routine for "OF". It would be desirable to handle such cases at a higher level and thus in a more general fashion. More research is needed to find rules to govern the interpretation of prepositional phrases which modify compound phrases.

4.3.2 Semantics of Other Prepositions

In this section, we will briefly describe the sense meaning classifications and semantic processing for the remaining prepositions. In each case, of course, there are some sense-meanings of the preposition which are not handled by the program; we discuss only those which are.

BY is used only to specify the agent of the verb in a passive verb phrase. The object of the preposition is put on the verb token under the indicator SUBJ.

AGAINST is used to specify a case argument for a verb, as in "rests against a vertical wall" (P8). The referent of the object phrase is identified, and the preposition and referent are put on the verb token as a modifier under the indicator CASEARG.

CASEARG modifiers are processed by the verb semantic routines, which may have specific interpretations for case arguments indicated by certain prepositions.

TO with a location object is used to specify a location for a verb phrase. The location referent is identified, and the preposition and referent are used as a modifier under the indicator LOC. Since modifiers are kept as a list under the property list indicator MODS, there can be multiple LOC modifiers. The preposition is kept in case the verb semantic routine can derive additional information from it; usually, however, only the location referent will be used.

Proposed LOC modifiers are tested against the head phrase to determine whether the modification is acceptable; this is done by a function called LOCTST. If the head is a verb phrase, the verb is tested to see whether it can properly take a location modifier. For example, in the sentence "There is a man weighing 150 lb at one end", the LOC modifier "at one end" would be rejected as a modifier of "weighing", while in the sentence "There is a man standing at one end" the LOC modifier would be accepted as a modifier of "standing". If the modified phrase is a noun phrase, the object referred to (explicitly or implicitly) by the location modifier is tested against the head object; if they are the same, the modification is rejected. Thus, in the sentence "one painter . . . stands on the scaffold 4.0 ft from one end" (P3), the location modifier "4.0 ft from one end" is rejected as a modifier of "scaffold", since its implicit object referent is the scaffold. This rejection on semantic grounds (making reference to the relationships among objects in the model of the problem which has been constructed so far) will cause the parsing in which the prepositional phrase modifies "scaffold" to be rejected, so that the prepositional phrase will eventually be interpreted as a LOC modifier of the verb "stands". In the case "a boy 3 ft from one end" (P7), the location modifier is accepted since different objects are referenced by the head and object phrases.

FROM (without a measurement phrase preceding it) modifies a verb as a CASEARG, as in "supported from the wall" (P8), or a LOC, as in "From end (A) a weight of 2500 nt is hung" (P15). FROM2 (preceded by a measurement phrase) always specifies a LOC; the measurement phrase may be a question phrase, as in "how far from the center" (P20). If the object of FROM2 specifies a physob rather than a location on a RELPOBJ (an object on which relative positions are defined), an appropriate object for the location to be on must be found. This is done by finding an attachment point between the specified object and a RELPOBJ; thus, "0.5 m from Paul" (P17) specifies a location on the pole Paul is carrying which is 0.5 m from the point of attachment between Paul and the pole. The semantic routine for FROM2 must interpret the given object (a physob) as an object of the desired type (a location on a different physob of a particular type).

BETWEEN occurs only once in our problems: "on a pole between them" (P17). When it connects a single physob and two physobs, as in this case, BETWEEN is interpreted to mean that the first object is attached to the other two at the "usual" places for the object (in this case, the ends of the pole).

AT always specifies a location, which may be a question phrase, as in "at what point" (P7).

IN specifies either a location, as in "stand in the center", or an attribute of an object, as in "the tension in each rope" (P5). In the latter case, the SFRAME of the head noun phrase is set to ATTROF and its SEMOBJ is set to the referent of the object noun phrase. The same semantics are used for sense-meaning 5 of OF, < attribute> OF < object> , and for one sense-meaning of ON.

WITH may be used to connect an object with an attribute and value, as in "a spring with a constant of 40 lb/ft" (P1), or to connect a second participant in a relationship with the relationship, as in "an angle of 60 deg with the horizontal" (P4). The latter sense-meaning is frequently used in English to define the participants in a relationship, usually using the verbs "have" and "make".

There are five sense-meanings of ON which are recognized by the program:

1. < physob> on < loc>	the rope on the left end (P4)
2. < attribute> on < physob>	the tension on each of the ropes (P3)
3. < action> on < physob>	the forces on the supports (P6)
4. < verb> on < physob>	stands on the scaffold (P3)
5. < verb> on < loc>	placed on the edge of a block (P14)

Sense-meanings 1 and 5 are processed as LOC modifiers; meaning 2 is converted to an ATTROF SFRAME; meaning 4 is converted to a CASEARG modifier. Meaning 3 is converted to the SFRAME ACTON, with the referent of the object noun phrase as its SEMOBJ.

4.3.3 Definition and Translation of Prepositions

Out of curiosity, the sense-meaning classifications for the preposition OF (which had the most sense-meanings of any in the program) were checked against the definitions given for OF in several dictionaries. The agreement with the dictionary definitions was very poor. Often, several of our classes would fit in a single dictionary class, or one of our classes would fit in several dictionary classes. Prepositions are of course hard to define, and native speakers of a language rarely need to look them up in a dictionary. However, in translating from one natural language to another (whether done by a human or by a machine), the correct translation of prepositional phrases is a difficult problem. For example, the preposition OF can be translated into about a dozen different prepositions in German; some uses of OF are translated into the genitive case or other constructions which do not use prepositions. It seems plausible that discrimination nets similar to those used in our preposition semantic routines might be used to discriminate preposition sense-meanings for machine translation. Hopefully, sense-meaning classes could be found such that all usages of a preposition which fall within each class could be acceptably translated into the same form in the target language.

4.4 Referent Identification

Referent Identification is the process of associating the phrases in a sentence with the objects and relationships they refer to (explicitly or implicitly) in the reader's model of the world. Such a process involves a number of possible steps. Candidate referents must be found. In some cases the candidates will be identified by the same word used in the sentence, or will be members of the same class which can be matched together (e.g., "Paul ' and "boy", both of which are members of the class PERSON with the restriction (SEX MALE)). In other cases, the phrase in the sentence identifies the candidates implicitly by identifying their relationships or attributes. (For example, in (P17) the word "load" refers to a sack which is being carried on a pole.) In such a case, the candidate can be considered an instance of the phrase in the sentence in its particular instantiation, but not in general. If there are no candidates (or if there are not enough), a referent must be created and added to the model. If there are several candidates, it may be necessary to select a particular one, either arbitrarily or based on modifiers of the phrase in the sentence. If modifiers are used, problem solving may be required to determine which of the candidates satisfies the modifiers. Once the referent(s) of the phrase have been identified, modifiers of the phrase must be processed to add information to the referent as appropriate.

ISAAC contains programs to identify three types of referents: Physical entities (objects and non-material physical entities such as forces), locations, and attachments. These referent identification programs are described below.

4.4.1 Identifying Physical Entity Referents

Physical entity referents are identified by the function IDRFNT. If the referent was previously identified, it is retrieved from the noun phrase token's property list. Otherwise, the referent is identified using the function PHYSNP and put on the token's property list under the indicator RFNT. (The "referent" is a list of pointers to each of the objects or relations denoted by the noun phrase.) If the noun phrase is compound, the referent of each component noun phrase is determined, and the concatenation of all the referents is used as the referent of the compound.

A flowchart of PHYSNP is shown in Figure 4.1. The first step in identifying

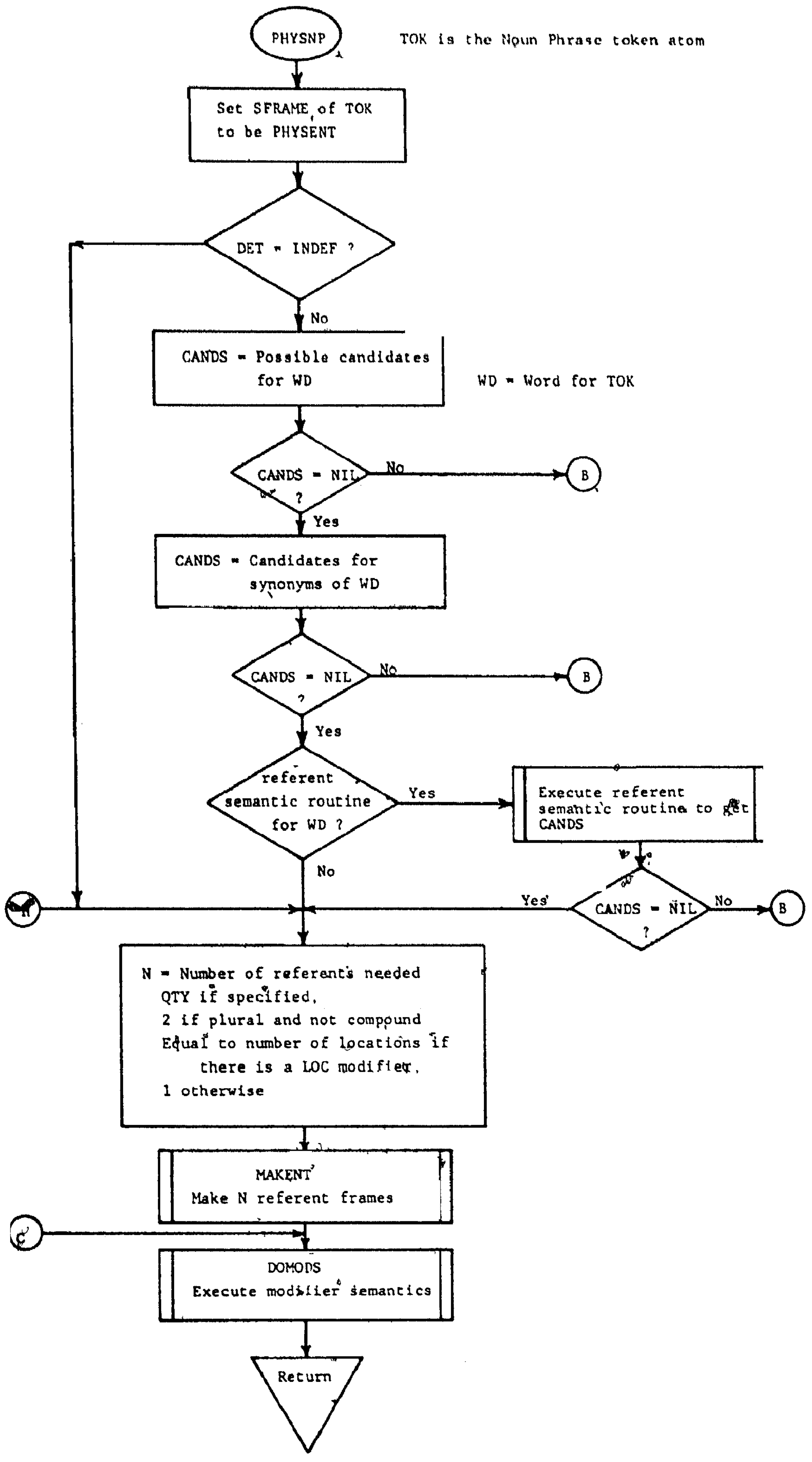

PHYSNP
TOK is the Noun Phrase token atom
Set SFRAME of TOK to be PHYSENT
DET = INDEF ?
No
CANDS = Possible candidates for WD
WD = Word for TOK
CANDS = NIL ?
No
B
Yes
CANDS = Candidates for synonyms of WD
CANDS = NIL ?
No
B
Yes
referent semantic routine for WD ?
Yes
Execute referent semantic routine to get CANDS
CANDS = NIL ?
Yes
No
B
No
N = Number of referents needed
QTY if specified,
2 if plural and not compound
Equal to number of locations if there is a LOC modifier,
1 otherwise
MAKENT
Make N referent frames
C
DOMODS
Execute modifier semantics
Return

Figure 4.1 (2 pages): Flowchart of PHYSNP

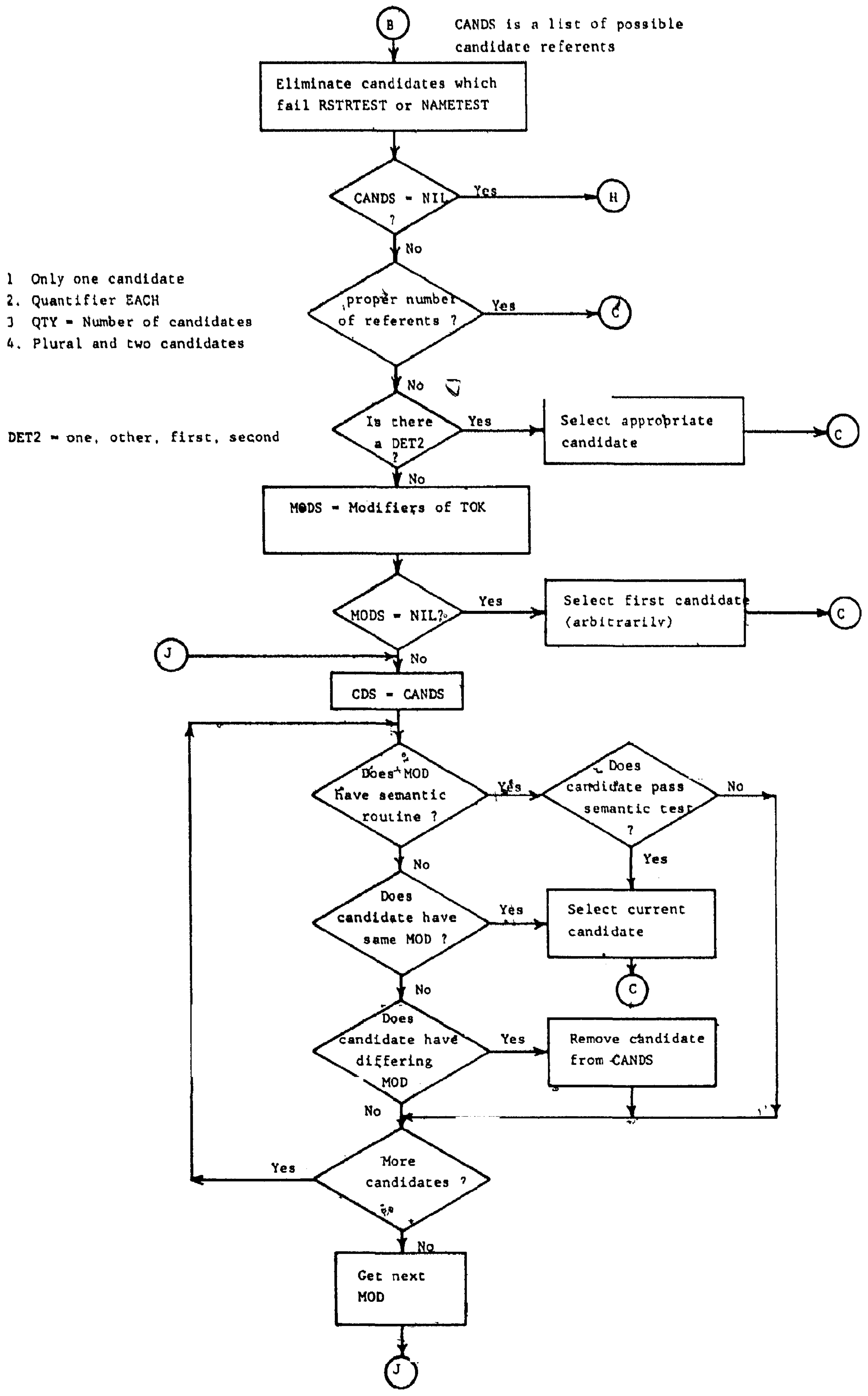

B
CANDS is a list of possible candidate referents
Eliminate candidates which fail RSTRTEST or NAMETEST
CANDS = NIL ?
Yes
H
No
1 Only one candidate
2. Quantifier EACH
3 QTY = Number of candidates
4. Plural and two candidates
proper number of referents ?
Yes
C
No
DET2 = one, other, first, second
Is there a DET2 ?
Yes
Select appropriate candidate
C
No
MODS = Modifiers of TOK
MODS = NIL?
Yes
Select first candidate (arbitrarily)
C
J
No
CDS = CANDS
Does MOD have semantic routine ?
Yes
Does candidate pass semantic test ?
No
Yes
Select current candidate
C
No
Does candidate have same MOD ?
Yes
No
Does candidate have differing MOD
Yes
Remove candidate from CANDS
No
More candidates ?
Yes
No
Get next MOD
J

Figure 4.1 (page 2)

the referent is to find the existing objects in the world model to which the noun phrase might refer. (If the determiner is indefinite, it is assumed that a new object is being referred to, and this step is bypassed.) The list of existing objects is searched first for objects with the same token word as the noun phrase, and then for objects whose token words are synonyms of the token word of the noun phrase. If no candidates are found by either of these searches, a semantic routine associated with the noun phrase token word is executed (if available) to see if there is a suitable referent for that word in the model. Such a semantic routine allows the noun phrase "the load" in (P17) to be matched to the object whose token word is "sack" The referent semantic routine for "load" selects an object which is a physical entity, is not a person, is supported by something, and does not support anything itself. The semantic routine for "support" selects the appropriate number of objects which all support the same object. If candidate objects are found by any of these searches, they are subjected to further testing beginning at the flowchart label (B) (page 2 of Figure 4.1).

If no candidate objects are found, or if all candidates are rejected on semantic grounds, new referent objects must be created. The number of objects to be created is set equal to the QTY (quantity) attribute of the noun phrase if specified (as in "two boys" (P20)), to two if the noun phrase is plural and not compound, to the number of locations if there is a location modifier (as in "a pier at each end" (P13)), or to one otherwise. The proper number of objects is then created using the function MAKENT.

In most cases, MAKENT simply creates a GENSYM atom, sets its token word appropriately, and adds it to the list of created objects. Provision is made, however, for special semantic routines to create referents for particular words. A seesaw, for example, is not a single object, but a rigid plank pivoted at its center. The semantic routine to make a referent for "seesaw" creates both objects and specifies their attachment. Similarly, an equilibrant is a force which is applied to a rigid body to produce equilibrium. The semantic routine to create a referent for "equilibrant" creates a force, finds an appropriate rigid body, and specifies the attachment of the force to the rigid body at an unknown position.

When the referents of the noun phrase have finally been determined or created, the function DOMODS is called to execute the modifier semantics for each of the modifiers which remain on the noun phrase token. Modifier semantics is discussed in a later section.

The second page of Figure 4.1 shows the tests which are performed on candidate referents for a noun phrase in order to reject those candidates which are clearly inappropriate on semantic grounds and to select the proper candidate(s) from those which remain. First, each candidate is subjected to RSTRTEST (restriction test) and NAMETEST. RSTRTEST requires that if the candidate and the noun phrase have

RESTRICT modifiers with the same indicator, the restriction values must be equal. Thus, "Paul" and "boy", both of which have the modifier (RESTRICT (SEX MALE)), would match, while "Paul" and "girl" would not. NAMETEST requires that if both the candidate and the noun phrase token have names, the names must match.

After any candidates which fail RSTRTEST or NAMETEST have been removed, the remaining candidates are examined to see if they constitute the proper number of referents. If there is only one candidate, if the quantifier "each" is present, if the number of candidates matches the QTY (quantity) of the noun phrase, or if the noun phrase is plural and there are two candidates, then the existing set of candidates is accepted without further tests. If a determiner adjective is present, the corresponding candidate is picked: the first for "one" or "first", or the second for "other" or "second". Otherwise, the candidates are tested against modifiers of the noun phrase. If a candidate is found which has a matching modifier (e.g., both have the modifier (WEIGHT 125 LB)), that candidate is selected. If a candidate has a mismatching modifier (e.g., (WEIGHT 150 LB)), that candidate is removed from the list of possibilities. Some modifiers, such as location modifiers, may have special semantic routines for selecting candidates. A candidate is selected by the location semantic routine if the location referent of the location modifier is a member of one of the attachment relations of the candidate. Thus, "the rope on the left end" (P4) will select the rope which is attached to the left end of the bar. If multiple candidates remain after all the modifiers have been tested, the first one is selected arbitrarily.

In some cases, the number of referents created for a noun phrase is not enough when the context of the noun phrase is considered; in such cases, the function MORERFNT may be called to create additional referents. For example, "the pier at each end of the bridge" (P18) will cause two "pier" objects to be created because of the two locations in the location modifier generated by the prepositional phrases. However, in "a plank .... supported at each end by a stepladder" (P19), the location modifier is attached to the verb phrase, so that initially only a single "stepladder" referent is created. The verb semantics for SUPPORT, however, requires a separate supporting object for each specified location, so that MORERFNT will be called to make a second "stepladder" referent.

4.4.2 Identifying Location Referents

There are two primary functions involved in the identification of location referents, IDLOC and LOCNP. IDLOC identifies a location given the object to which the location is relative, the location name, an optional SELECT modifier, and an optional list of location frames to be excluded from the selection process. For example, the phrase "the left end of the lever" would result in a call to IDLOC with the referent object for "the lever", the location name "END", the SELECT modifier "LEFT", and a null exclusion list as arguments. IDLOC is used both by internal processes and by LOCNP.

LOCNP identifies the location(s) referred to by a noun phrase. Since a location may be specified by a wide variety of syntactic forms, LOCNP must identify the form of the noun phrase and the features of the location which are specified. These features are collected, and missing features are filled in by making inferences; finally, IDLOC is called to identify the location referents. Thus, LOCNP serves as an interface function to collect the arguments for IDLOC and put them into a standard form. IDLOC and LOCNP are described in detail below.

A flowchart of IDLOC is shown in Figure 4.2. IDLOC first examines all the existing locations on the specified object to see if one of them is suitable. An existing location is rejected if it is a member of the excluded locations list, if it has the wrong location name, or if it has a relative position (displacement) from the named position. If the location passes these tests, it is examined for the specified SELECT value. In most cases, the SELECT semantics consists of a test for an identical SELECT modifier (e.g., RIGHT or LEFT). In some cases, however, a special semantic routine must be used to test the world model and determine whether a location meets the selection criterion. To find "the heavy end" (P12), for example, it is necessary to examine the object frame for the object involved; the "heavy" end is the one which is closest to the center of gravity of the object. Which end is the "heavy" one could be changed by changing the numeric value of either the length of the bar or the distance from one end of the center of gravity, while leaving all the English words the same. Thus, numerical problem solving by a specialist program, based on the particular values specified for certain parameters, is required to determine the proper location referent.

If no SELECT parameter is specified to IDLOC, or if the object being examined has no SELECT modifier, the object is saved as a second choice in case a better candidate is not found. Thus, if a SELECT value of LEFT is specified, all the locations on the object with the proper location name (e.g., END) will be examined for a SELECT LEFT modifier. If none is found, a location with no SELECT modifier will be chosen; when the modifiers of the noun phrase are processed, the select value will be added to that location frame.

In addition to its use by LOCNP, IDLOC is used internally by semantic routines to identify particular locations on objects. For example, when a referent object for "seesaw" is created, IDLOC is called to create a location frame for the center of the newly created seesaw plank; this location is then used in specifying the attachment of the plank to the pivot which is created.

LOCNP identifies the referent(s) of a location noun phrase; such a location

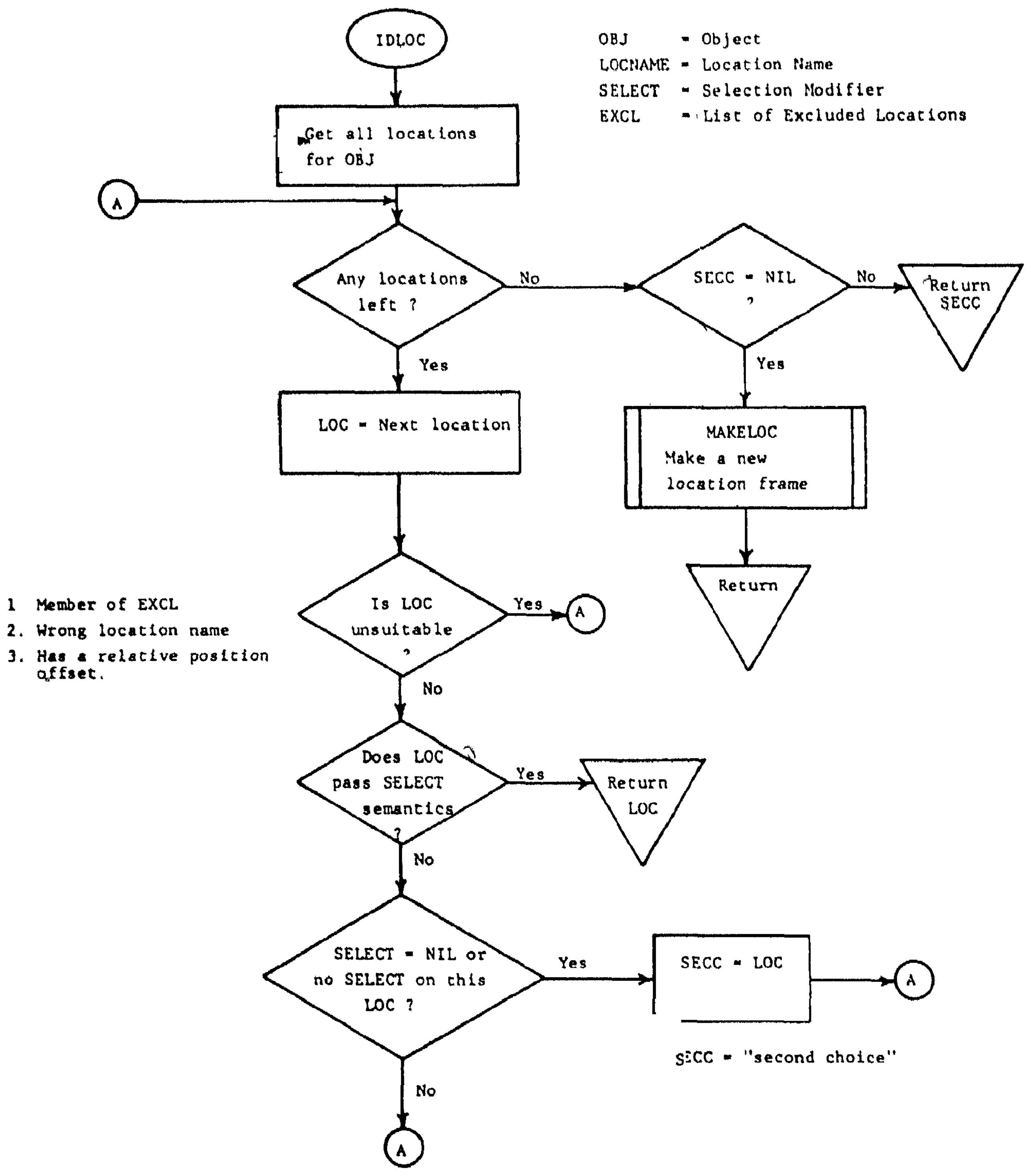


Figure 4.2: Flowchart of IDLOC

may be denoted in many different ways. If the location has a name, the name alone may be used (as in "80 cm from (A)" (P6)); the object to which the location is relative may or may not be named ("the left end of the lever" or simply "the left end"); a physical object name may be used to specify a location, since every physical object occupies a position in space.Most of the function LOCNP consists of code to make the inferences and collect the arguments.needed to identify a location when the location is denoted by any of the noun phrase forms mentioned above.

A flowchart of LOCNP is shown in Figure 4.3. If the referent of the noun phrase is known, it is returned at once. Otherwise, a series of tests is made to determine the type of location noun phrase. If a location is specified by name, the existing location frames are searched for a location with that name. When the correct location is found, it is saved on the noun phrase token under the indicator RFNT, and the function DOMODS is called to process any modifiers of the noun phrase. If the noun phrase is already marked as being a LOCPART SFRAME, the object to which the location is relative is already known; this will be the case if a modifier of the location noun phrase specifies the object, as in "the end of the lever" or "its left end". In such cases, LOCNP transfers directly to the label "B" (page 2 of Figure 4.3). If a location is named without an object (as in "one end"), it is necessary to find an appropriate object. This is done by examining the GEOMODEL (geometric model) of each object in the model of the problem until an object for which the location name is appropriate has been found. Once the appropriate object for the location has been inferred, the noun phrase token is converted to a LOCPART SFRAME, and control is transferred to label "B". If the noun phrase names a physical object or person, IDRFNT is called to identify the physical object referent. If the object to which the location is relative is specified in the call to LOCNP and is different from the object named by the noun phrase, a search is made for a location at which the named object is attached to the desired object; thus, in "0.5 m from Paul" (P17), which specifies a location on a pole which Paul is carrying, "Paul" is interpreted as a location on the pole by finding the point on the pole where Paul is attached to it. If the desired object is unspecified, a location is made for the default location of the named object.

At label "B" of the flowchart, where LOCPART SFRAMEs are processed, a test is made to see if the noun phrase is plural or modified by the quantifier EACH, as in "its ends" or "each end". If so, the number of such locations is gotten from the GEOMODEL of the object, and that number of locations is identified by calls to IDLOC. Thus, "each end" (P3), referring to a scaffold, will cause two "end" location frames to be generated. If the noun phrase is singular, IDLOC is called to identify a single location referent. If a location name is specified, the location found is required to pass NAMETEST, having either the correct name or no name. Once the proper referent

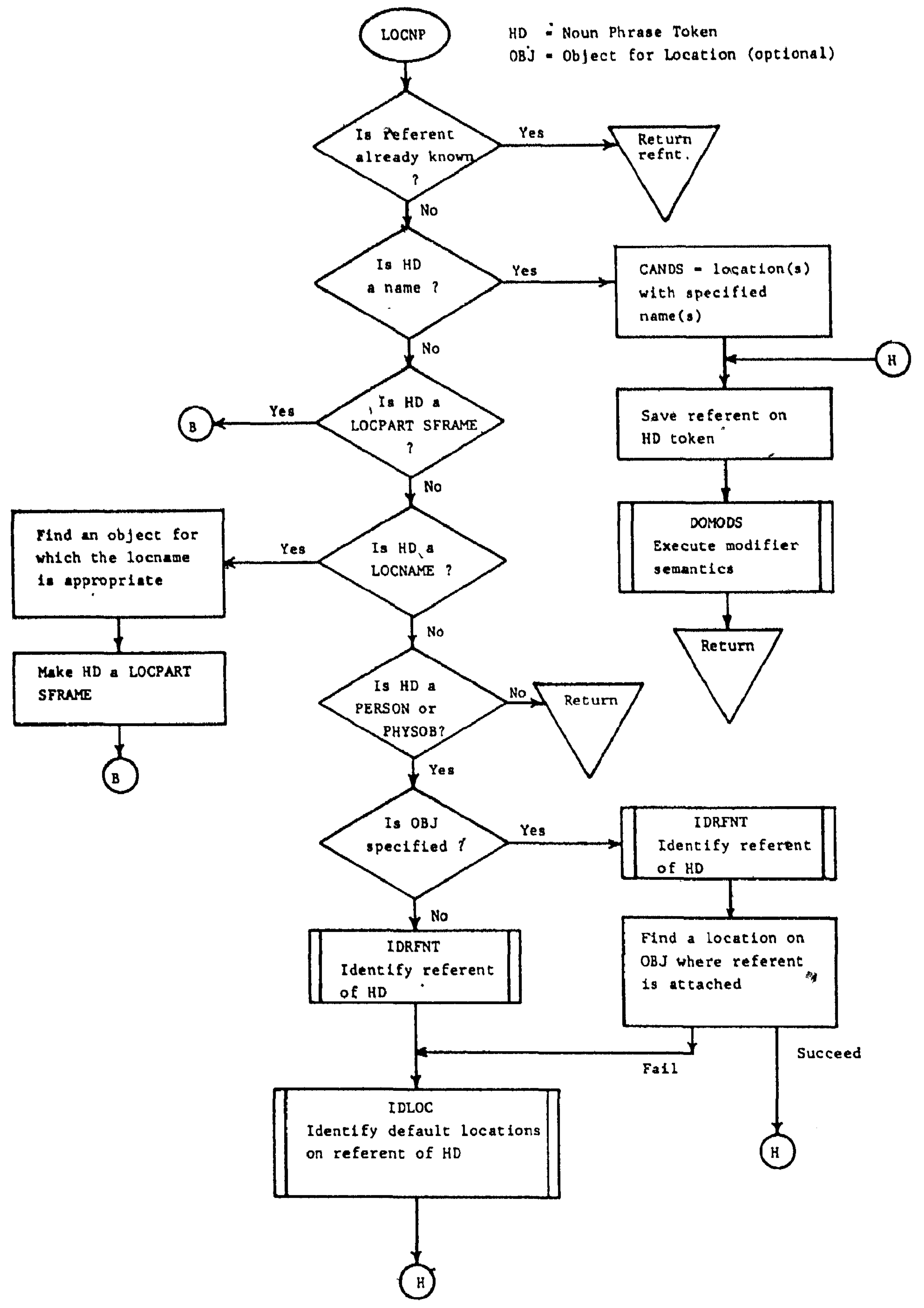

LOCNP
HD = Noun Phrase Token
OBJ = Object for Location (optional)
Is referent already known ?
Yes
Return refnt.
No
Is HD a name ?
Yes
CANDS = location(s) with specified name(s)
H
Save referent on HD token
DOMODS Execute modifier semantics
Return
No
Is HD a LOCPART SFRAME ?
Yes
B
No
Is HD a LOCNAME ?
Yes
Find an object for which the locname is appropriate
Make HD a LOCPART SFRAME
B
No
Is HD a PERSON or PHYSOB?
No
Return
Yes
Is OBJ specified ?
Yes
IDRFNT Identify referent of HD
Find a location on OBJ where referent is attached
Fail
Succeed
H
No
IDRFNT Identify referent of HD
IDLOC Identify default locations on referent of HD
H

Figure 4.3 (2 pages): Flowchart of LOCNP

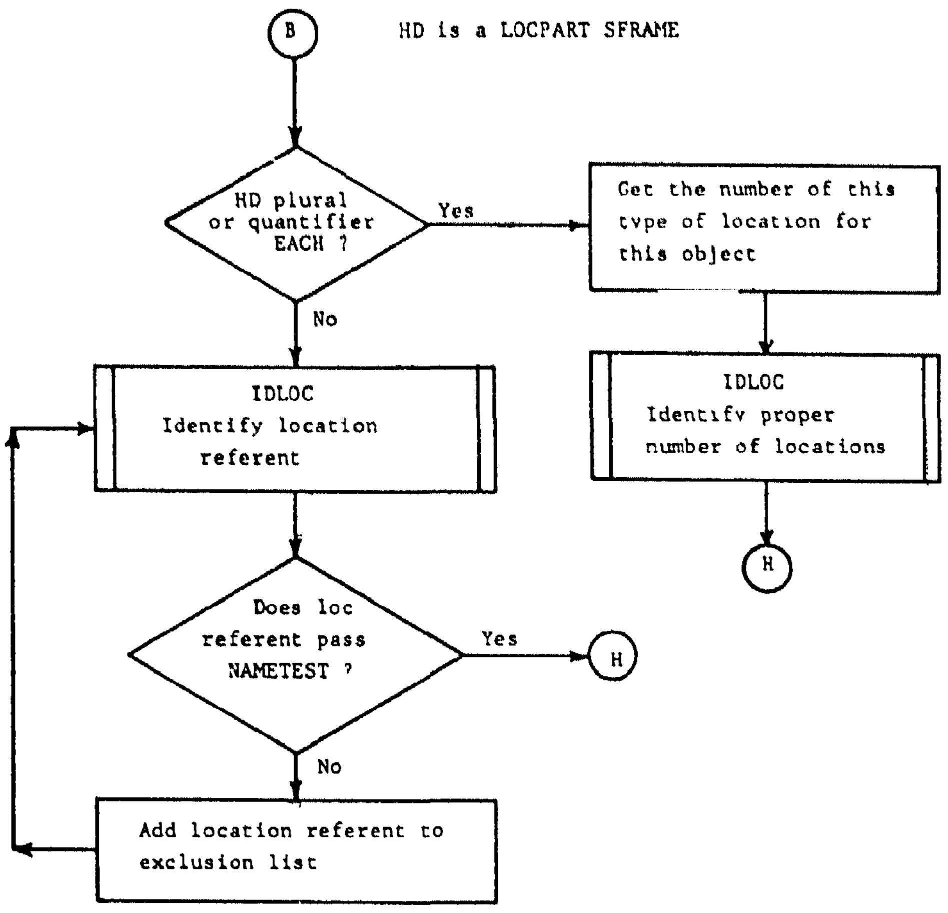

B
HD is a LOCPART SFRAME
HD plural or quantifier EACH ?
Yes
Get the number of this type of location for this object
No
IDLOC Identify location referent
IDLOC Identify proper number of locations
H
Does loc referent pass NAMETEST ?
Yes
H
No
Add location referent to exclusion list

Figure 4.3 (page 2)

has been found, control is passed to the label "H" to save the referent and process modifers of the noun phrase.

4.4.3 Attachment Identification

An attachment relationship among two or more objects is identified by the function IDATT. Attachment relations are not the direct referents of phrases in a sentence, but are defined by verb semantic routines or modifier semantic routines. The argument of IDATT is a paired list of objects and locations on the objects; one member of each pair may be nil. IDATT identifies an attachment frame which specifies the attachment of all the objects in the list; if no such attachment frame exists, one is created, along with links between it and the objects involved. (The structure of attachments and other frames is described in Appendix B.) If an existing attachment which matches the list is found and the list contains locations which were previously unspecified, the locations are put into the existing attachment frame. Thus, in cases such as

A painter . . . stands on a plank . . .
If he stands 1.0 m from one end of the plank . . . (P19),

the second attachment will be identified with the earlier one and will cause the location on the plank to be added to the attachment frame. The order in which the object/location pairs are specified in the call to IDATT is unimportant.

A second parameter in the call to IDATT is the type of attachment: CONTACT (as in the above example) or PINJOINT. The type of attachment is not used by IDATT, but is saved with the attachment frame for later use. The interaction of objects at an attachment point may depend on the type of attachment. A CONTACT attachment with a "smooth" surface, for example, implies that the force exerted by the surface is nonnegative and perpendicular to the surface. A PINJOINT attachment may transmit a force in any direction, but may not transmit a torque. Although other types of attachments could be used, CONTACT and PINJOINT are the only ones used by the program in its present form.

4.5 Modifier Semantics

Modifiers of noun phrases are saved, after some preliminary processing (Section 4.2), on the property list of the noun phrase token under the indicator MODS. After the referent of the noun phrase has been determined, the semantic routines of these saved modifiers are executed so that the appropriate changes may be made to the referent of the noun phrase. (Some modifiers, which are used in selecting the proper referent, are deleted before this stage is reached.)

Modifier semantic processing is controlled by the driver function DOMODS, which calls PUTMODR for each modifier. PUTMODR (which is also used for modifier processing by some verb semantic routines) transfers the modifier to the property list of each referent, or executes a special semantic routine if there is one associated with the modifier. Thus, in simple cases such as "a 150 lb man", the modifier (WEIGHT 150 LB) generated from the adjective phrase is transferred to the referent's property list as the value (150 LB) under the indicator WEIGHT. In other cases, semantic routines may make inferences from modifiers, e.g., that an object which is at a location on another object is attached to the other object at that location.

RESTRICT modifiers are concatenated and placed on the referent object under the indicator RESTRICT; this allows an object to have multiple RESTRICT modifiers, which are used in determining noun phrase referents.

Measurement modifiers are transferred directly to the property list of the referent. In the process, the measurement units for each type of measurement are saved for use in answer generation. It would be easy to modify the measurement modifier semantic routine to allow differing units (e.g., feet and meters) to be used in the same problem.

NAME modifiers are processed in different ways depending on the type of name and the type of object which is named. Simple names are transferred directly to the property list of the named object. Geometric names which modify locations are distributed to the named locations. If geometric names are assigned to a physical object, as in "a uniform bar (A B)" (P6), location referents are created for the appropriate locations on the object (in this case the ends of the bar) as determined by the object's GEOMODEL, and the geometric names are assigned to the location referents.

An APART modifier gives the distance between two locations, as in "the hinges of a door . . . are 12 ft apart" (P9). This modifier not only gives the distance between the two locations, but also implicitly determines the size of the object if the two locations are on the same object. In the above case, for example, we can infer that the door is at least 12 ft tall. The semantic routine for APART modifiers consults the GEOMODEL of the object, calculates the overall size dimension which would give the specified distance between the two points, and assigns that size to the object.

In our set of physics problems, a location modifier of a noun phrase always implies that the referent object is attached to something at that location, as in "an automobile . . . which is 30.0 ft from one end of the bridge" (P18). The modifier semantics routine for location modifiers calls IDATT to define the attachment. In a larger system which handled a wider range of problems, some additional semantic tests would be needed to determine whether an attachment was actually implied by the location modifier.

4.6 Verb Semantics

The semantic functions performed by verbs are very diverse. Some verbs (for example, certain sense-meanings of the verbs "is", "have", and "make") serve only as function words which connect other phrases; the semantics of such verbs resides primarily in the phrases they connect. Other verbs (e.g., "need" or "wish") introduce verb phrases to which they pass some of their case arguments. Some verbs carry case arguments and other inferences to be used with their "underlying" verbs; for example, "stand on . . ." specifies an attachment by contact between the feet of the subject and the object of "on", with the subject in a standing position. A single verb may have multiple sense-meanings; as in the case of prepositions, we found that discrimination net tests based on rough semantic classifications of the case arguments of the verb were sufficient to differentiate the sense-meanings.

In this section, we will describe the semantic functions for a number of verbs as they are implemented in the program. In the cases where a verb appeared infrequently in the sample problems, the verb semantic routines handle only the limited sense-meanings necessary for those cases; often, there are not many error checks to keep the program from going astray if it were presented with different cases. Some of the verb semantic routines handle a number of variations in the types of their case arguments; it seems likely that general rules for handling different types of arguments which would be applicable to classes of similar verbs might be found. This would be an interesting area for further research.

The execution of a verb semantic routine is initiated by EXVBSEM, which is called when a clause or dependent clause has been parsed. EXVBSEM executes the semantic routines for any prepositional phrases or adverbs which modify the verb. It then binds some of the case arguments of the verb (and their referents) to global variables so that they will be easily accessible, and calls the semantic function associated with the main verb of the verb phrase.

4.6.1 Semantics of the Verb "BE"

There are seven sense-meanings of the verb "BE" which are recognized by the program; the sense-meaning classes are listed with examples below.

1. THERE BE <physob> <loc>	At (B)...there is a weight (P15)
2. <physob> BE <loc>	a man is 10 ft from the top (P8)
3. <physob> BE <adj phrase>	the door is 3 ft wide (P9)
4. <attrof> BE <measurement>	the weight of the lever is 8 lb (P1)
5. <attrof> BE WHAT	what is the weight of the bar (P4)
6. <locpart> BE <loc>	its center of gravity is 6.0 ft from one end (P11)
7. <subj> BE TO <verb phrase>	the bar is to be supported...(P6)

These sense-meanings are easily separated by a set of discrimination net tests, most of the semantic classes being tested at this point are SFRAME types, so that any of the syntactic forms which result in the creation of a particular SFRAME will be accepted. Once the sense-meanings have been separated into these classifications, we find happily that most of the semantics has already been done: it is only necessary to pass the arguments of the verb to routines which were written to do the same semantics for different syntactic forms. Sense-meaning 1 is changed to the same form as 2; IDATT is called for both cases to define an attachment of the object at the location specified. For sense-meanings 3 and 4, PUTMODR is called to execute the semantics of the modifier for the referent of the object involved. For sense-meaning 5, the argument is converted to arguments for the question routine WHATIS; WHATIS is explained in a later section. For sense-meaning 6, the location is saved on the property list of the object referent using the location name as the indicator. For sense-meaning 7, the function SUBSTINF is called to substitute the subject of the verb as the syntactic subject of the infinitive verb phrase and execute its verb semantics. Thus, in "the bar is to be supported" (P6), the subject "the bar" is substituted as the syntactic subject of the passive verb phrase, so that the referent of "the bar" becomes the semantic object of the verb "support".

4.6.2 Semantics of the Verb "SUPPORT"

Six sense-meaning classes of the verb "SUPPORT" are recognized by the program; these are listed with examples below.

1. < physob> SUPPORT < physob>	the lever is supported by a spring (P1)
2. < physob> SUPPORT < N times > AS MUCH AS < physob>	the boy...supports ⅓ as much as the man (P2)
3. < physob> SUPPORT WHAT < force>	what load does each pier support (P13)
4. < nil> SUPPORT < physob>	a beam...is supported at both ends (P16)
5. < physob> SUPPORT < locpart>	the top of the ladder is supported from the wall by a horizontal rope (P8)
6. < physob> SUPPORT < attrof>	the weight of the door is supported by the upper hinge (P9)

It might be argued that these are not distinct sense meanings, but rather six different ways of specifying the arguments for a single sense-meaning. Essentially, the verb SUPPORT (for our purposes) specifies an attachment of two objects at a particular location on each object; a force is exerted on one object by the other object in order to support it. For sense-meaning classes 1, 4, and 5, the arguments of the verb are arranged to serve as arguments for IDATT so that the attachment relation may be specified. (In the case of sense-meaning 4, a pivot object is created to serve as the unnamed supporting object.) In the remaining sense-meaning classes (2, 3, and 6), the force exerted in the attachment relation is referenced. Such a force is identified by the function IDFORCE, which creates variables for the force vector and adds them to the attachment relation if necessary. For sense-meaning 2, < physob> SUPPORT < N times> AS MUCH AS < physob> , equations are written which relate the two force vectors so that one is N times as much as the other. For sense-meaning 6, < physob> SUPPORT < attrof> , an equation is written equating the force and the specified force attribute. For sense-meaning 3, < physob> SUPPORT WHAT < force> , the force vector variables are marked as desired unknowns, and an entry is made to use the force vector values as a reply. (The latter operations are discussed in more detail in the section on question semantics.)

4.6.3 Semantics of Other Verbs

Verbs such as WEIGH and STRETCH express attributes in verbal form. The semantic routines for these verbs call the function ATTRVBSEM with the appropriate case argument of the verb (subject for WEIGH, object for STRETCH) specified as the object which is modified. ATTRVBSEM uses the attribute associated with the verb to make a modifier, whose semantics are executed by PUTMODR. (In the case of a question, the case argument and attribute are used as parameters for the function WHATIS.) Thus, a semantic transformation is used to transform the verbal form into a modifier form for which the semantics already exists. The forms "a man weighs 150 lb", "a 150 lb man", and "the weight of a man is 150 lb" are all reduced to an identical "semantic deep structure", which consists of the referent object for "a man" and the modifier (WEIGHT 150 LB), by the time the semantics of the modifier are to be executed. A single modifier semantic routine performs the final semantic operations for all three cases.

In addition to the verb SUPPORT, the verbs REST, PIN, BALANCE, SIT, HANG, CARRY, ATTACH, STAND, LIFT, and EXERT can all be used to specify attachment relations. SIT and STAND imply that particular locations on the person who is sitting or standing are involved in the attachment, and that the attachment is of type CONTACT. (These verbs could also determine the person's posture for the picture-making programs, but that is not done in the present system.) A number of the verbs imply that one of the objects in the attachment relation supports the other. These support relations are marked by SUPPORT and SUPPORTBY links among the objects; they are used in later inferences, such as inferring whether a person should be modeled as a pivot or as a weight by the problem solver. The verb PIN implies (as used in these problems) a pivot object which must be created as the other object for the attachment relation. CARRY, if used with an instrument, implies that the subject is attached to the instrument and that the object is attached to and supported by the instrument, as in "Paul and Henry carry a sack . . . on pole between them" (P17).

The verbs WISH NEED. and REQUIRE are used in the sample problems with infinitive verb phrase objects, as in "two boys . . . wish to balance on a seesaw" (P20). For our purposes, the "modal" information provided by these verbs can be ignored. The verb semantic routines for these verbs call the function SUBSTINF to substitute the appropriate argument as the syntactic subject of the infinitive verb phrase and execute its verb semantics. The above example is processed as if it were simply "two boys balance on a seesaw".

HAVE appears with only one sense-meaning, < physob> HAVE < locpart> < loc> as in "a bar . . . has its center of gravity 1.5 m from the heavy end" (P4). The location is put on the property list of the subject referent using the LOCPART name as the indicator This sense-meaning is similar to sense-meaning 6 of the verb BE, < locpart> BE < loc> , except that the arguments are in a different order.

MAKE is used with a relation name as an object, as in "the rope . . . makes an angle of 45 deg with the horizontal" (P4). In such cases, the semantics is determined primarily by the relation involved (in this case, "angle"). The verb semantic routine for MAKE calls the semantic routine for the relation, passing to it the arguments of the verb. The semantic routine for "angle" creates a relative rotation modifier and attaches it to the former subject referent. The ambiguity of the direction of rotation is maintained by the relative rotation modifier; later, absolute rotations are chosen (based on symmetry considerations) to provide a plausible interpretation of the problem.

FIND, CALCULATE, COMPUTE, and DETERMINE are all handled by a common semantic routine. If the object of the verb is an ATTROF SFRAME, as in "find the tension in each rope" (P5), the object and attribute are used as arguments for the question routine WHATIS. If the object of the verb is an ACTON SFRAME, as in "compute the forces on the supports" (P6), the desired force is identified using IDFORCE. The force variables are marked as desired unknowns, and an entry is made to print the value of the force as a reply.

4.7 Question Semantics

A question of the type found in our physics problems specifies two types of information: a set of variables whose values must be found in order to answer the question, and the manner in which the information provided by the variables is to be presented in the answer. For example, the sentence "Determine the magnitude, direction, and point of application of the equilibrant" (P15) identifies the two variables in the equilibrant's force vector and the distance variable in the equilibrant's attachment relation as "desired unknowns", or variables whose values are required to generate the answer. In addition, the sentence specifies that the magnitude and direction of the force vector are to be computed from its (x y) component form, and that the location of the equilibrant on the rigid body is to be described. These two types of information are maintained separately by the program. Variables which are identified by a question as desired unknowns are put onto a globally bound list called DESUNKS; the problem solver attempts to find values for all the variables on DESUNKS, and stops when values for all of them have been found. In addition, the question semantic routines determine what type of reply is required and what objects the reply should be generated from; this information is formatted as a series of function calls to the answer generation routines with pointers to the proper objects in the model as arguments. These function calls are put onto a globally bound list called SYSREPLY. At the end of the problem-solving process, the function calls on SYSREPLY are executed to generate the answers.

There are five answer-generation routines. PRTVAR prints the value of a simple variable. PRTFV prints the value of a two-component force vector composed of two variables. PRTMAG and PRTDIR compute and print the magnitude and direction, respectively, of a force vector object (which is different from a force vector associated with an attachment, since it is a separate object in the model). PRTLOC generates a description of a location, typically as a point which is a certain distance from a known location.

Questions involving an attribute of an object, as in "What is the weight of the bar'' (P4), result in calls to the function WHATIS. If there is a special question semantics routine associated with the attribute, that routine is executed; such routines for the attributes "magnitude'', "direction'', and "point of application'' generate calls to PRTMAG, PRTDIR, and PRTLOC, respectively. If there is no special semantic routine, a variable is created for the specified attribute, marked as a desired unknown, and inserted into a call to PRTVAR.

When a question involves an ACTON SFRAME, as in "compute the forces on the supports'', (P6), the force vector involved is identified using the function IDFORCE. This force vector is then used as the argument in a call to PRTFV.

A question involving a location, as in "where'' or "at what point'', causes a location to be created on an appropriate object (either an object which is a case argument of the verb or an inferred object from the model of the problem) with an unknown distance from a known location on that object. The unknown distance is marked as a desired unknown, and a call to PRTLOC is generated to describe the location.

A question involving an unknown distance from a known point, as in "how far from the center'' (P20), causes a variable to be created for the distance and marked as a desired unknown. In contrast to "where'' questions, for which a description of the location is generated, this type of question generates a call to PRTVAR to print the value of the distance variable.

5. Construction of Object Frames and the Geometric Model

5.1 Introduction

In reading the English problem statement of a physics problem, ISAAC builds an internal model of the problem in which most of the objects and relationships in the problem are represented. A number of steps are necessary to convert this model into a model for which equations describing the interactions of the objects can be written. It is necessary to determine for each object the canonical object frame which represents the object in its particular instantiation in the problem for the purpose of solving a physics problem. (The frame representing a similar object in a different situation or for a different purpose might be a completely different type of canonical object.) A person, for example, might be modeled as a weight when sitting on a pole, or as a pivot when carrying it. Once the canonical object frame has been selected, it is necessary to make appropriate assumptions to fill in information necessary for the canonical frame which may not be present in the original problem statement. A "weight" object must have a weight, although it need not have a geometric size, if the weight is unspecified and is not a variable, a symbolic constant is created for it. A "lever" object need not have a weight, but must have a length.

Once the canonical object frames have been selected for all the objects in the model, a geometric model of the problem in which the locations and orientations of the objects are made explicit must be constructed. Since the sizes of some objects may be symbolic constants, the geometric locations for some points may contain algebraic expressions. Problem solving by specialist programs (for example, solving a triangle given two sides and an angle) may be necessary in order to create a complete geometric model.

After the geometric model of the problem has been created, the canonical frames for each object are completed by filling in any necessary information that remains unspecified. The weight of an object, for example, is modeled as a force exerted on the object at its center of gravity. (The geometric model is needed to determine the location of the center of gravity.) Attachment relations are completed by creating force variables for each object involved in the attachment. After all of these processes have been completed, the problem solver is called to write equations for the interactions of the objects and solve the resulting equation set.

This chapter describes the processes of making canonical object frames, creating the geometric model of the problem, and completing the frames which were created.

5.2 Making Canonical Object Frames

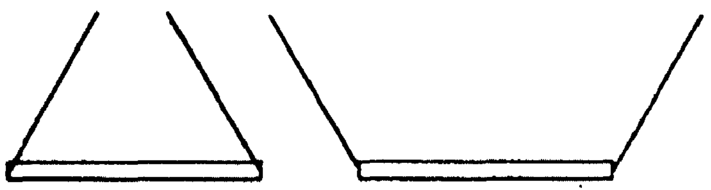
A Canonical Object is an idealization of an actual physical object which represents its salient characteristics for a particular physics problem. A pole, for example, may be represented as a weightless rigid body; this is an idealization of an actual pole, which has a finite weight and is not perfectly rigid. The idealized canonical objects used in physics problems, such as weightless poles and frictionless pulleys, rarely exist in the real world, but often give good approximations to the behavior of real-world objects. The same object may be represented in different problems by different canonical object frames, depending on its relationship to other objects in each problem. For each object in the problem, it is necessary to decide which canonical frame should be used to represent it, to mark the object with the canonical frame type, and to fill in any information necessary for the frame which is missing.

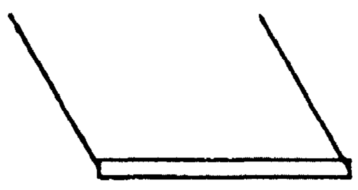
The function MFDRIVER calls the appropriate frame making routine for each physical entity in the model of the problem. Associated with each physical entity token word is a list of the frame-making routines which might be applicable to that type of object; there may be a specialist routine for a particular object (as in the case of a person) which decides which of several possible canonical object frames to use, or there may be a list of more general routines which can fail if they are inappropriate for a particular object in a particular context. In the present system, only a single frame-making routine is needed for each physical entity token.

The functions REQUIREVAL and REQUIREVAR examine an object frame for a specified quantity; if the quantity is unspecified, they create a symbolic constant or a variable, respectively, to represent the missing quantity, and add the constant or variable to the property list of the object frame atom. A constant or variable is a GENSYM atom which is added to the list of objects in the model; it has property list values which tell the canonical object frame it is associated with, the quantity it measures (e.g., TENSION), the units (e.g., LB), and whether it is a constant or variable.

There are seven canonical object types in the present system: LEVER, WEIGHT, SPRING, PIVOT, ROPE, SURFACE, and FORCE. The simplest, the PIVOT and SURFACE frames, do not require any attributes. A WEIGHT is required to have a weight; if absent, a constant is generated for it. A SPRING or ROPE must have a TENSION (variable); a SPRING must also have a STRETCH (variable) and CONSTANT (constant). [The type of symbol generated for each quantity if it is unspecified is given in parentheses.] A SPRING or ROPE must have a LENGTH (constant) only if it is attached to more than one object. CKROTATION is called to check and disambiguate the orientation of a SPRING or ROPE, and DISAMLOCS is called to disambiguate locations; these functions are described below. A FORCE frame is required to have an orientation; if absent, an orientation of zero is assumed. A LEVER (actually, rigid body) frame is required to have a LENGTH (constant). If a width is specified (as in the case of the door in (P9)), it is used in making the geometric size vector; otherwise, a width of zero is assumed. Unless a LEVER is oriented vertically, it is required to be attached at more than one point; if it is not, a PIVOT object is created and attached to a point similar to the existing attachment point. Thus, in a problem such as "What force is needed to lift one end of [a beam]" (P10), a pivot is created to hold up the other end of the beam while one end is being lifted. DISAMLOCS is called for a LEVER frame to disambiguate its locations. The function MFPERSON, which makes a frame for a PERSON, examines the context to determine whether to model the PERSON as a WEIGHT or as a PIVOT. If the PERSON is supported by something or supports something, a WEIGHT or PIVOT model is used, respectively. Otherwise, the objects the person is attached to are examined to see whether they support something or are supported. A person is assumed to be supported by an object which is supported, and to support an object which supports something. (A function to infer support relationships based on "usual" relationships and a more careful examination of the known relations of objects in the problem would not be too difficult, and would give correct answers in more general cases than the above heuristic can handle.)

CKROTATION examines an object to see if its orientation is specified by a relative rotation, as in "the rope on the left end makes an angle of 45 degrees with the horizontal" (P4). If so, the relative rotation is converted to an absolute rotation. In addition, the objects to which the specified object is attached are examined to see if a similar object is attached to one of them with a relative rotation; if so, the rotation of the other object is made absolute in a direction symmetrical to that of the first object. This insures that if an object is hanging from two ropes, for example, the orientations of the ropes will be made symmetrical:


Right

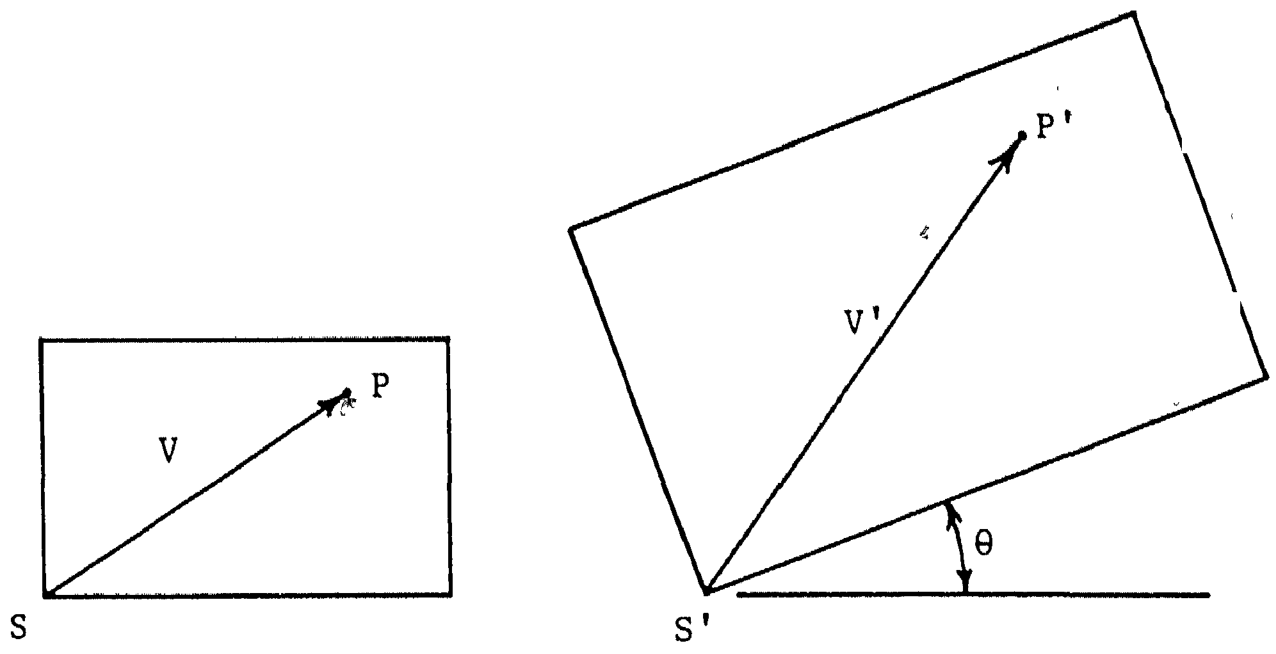

Wrong

DISAMLOCS disambiguates locations by assigning specific locations on an object to location frames which were originally specified by ambiguous location names. The ends of a bar, for example, may be specified by "one end . . . the other end", "the left end . . . the other end", "ends (A) and (B)", and so forth. These locations must be assigned to specific locations on the object so that geometric positions can be computed. DISAMLOCS first assigns location names to locations which have specific SELECT modifiers; the appropriate SELECT modifiers and corresponding absolute location names are specified as part of the GEOMODEL of the object. Thus,a location with the name END and the modifier (SELECT LEFT) is assigned the absolute location name LEFTEND. After those locations which have specific SELECT values have been assigned, the remaining locations are given unique absolute location names; thus, "the other end" would be given the absolute name RIGHTEND if it appeared with "the left end". Absolute location names are propagated to locations relative to named locations (e.g., "2 m from the right end" (P4)) by the function RENAMELOC.

5.3 Geometric Model Construction

After a canonical object frame has been made for an object, its geometric size and (frequently) its absolute rotation are known, and absolute location names are assigned to all of its locations. This information is sufficient to construct a geometric model of the problem in which absolute locations (coordinates which are numeric or composed of expressions involving constants or variables) are assigned to each object and (implicitly) to all of its locations. The geometric model is two-dimensional. The position of an object is completely specified by three quantities: the coordinates of its starting point, its rotation relative to its standard orientation, and its geometric size. The GEOMODEL of the object gives the coordinates relative to the starting point for each named absolute location. The geometric position of a named point on the object can be found by taking the coordinates of the point relative to the starting point, scaling this vector by the geometric size, rotating it by the object's rotation, and adding the resulting vector to the geometric coordinates of the starting point. This process is illustrated in Figure 5.1. The vector V, which is the position of the point P relative to the starting point S in the GEOMODEL of the object, is scaled to the appropriate geometric size and rotated through the angle θ to give the vector V′. Adding V′ to S′, the geometric starting point of the object in the problem, yields P′, the coordinates of the point corresponding to the point P.

Once an object has been added to the geometric model by specifying values for its GSTART, GSIZE, and ROTN (rotation), the geometric coordinates for any location on the object may be obtained by calling the funtion EXECLOCA with the location frame as an argument. If the location specifies a position relative to a named location, EXECLOCA calls itself to find the position of the named location. A relative position vector of the appropriate size is created and added to the geometric position of the named point to give the position of the relative point. The direction of the relative position vector is taken as the direction of a vector from the named point toward the



GEOMODEL of Object Object as it appears in problem

Figure 5.1: Finding Relative Position on an Object

center of gravity of the object; if the named point is the center of gravity, the direction of the rotated x-axis of the object is used.

Construction of the geometric model is performed by the function EUCLID. When EUCLID is initiated, every object has a geometric size (in terms of length units, e.g., meters) specified under the property list indicator GSIZE. The GSIZE of each object is recomputed by dividing each component by the corresponding scale factor (stored under the indicator FRMSCL) for the GEOMODEL of the object. After this has been done, a relative position vector from the GEOMODEL can be multiplied by the object's GSIZE to yield the corresponding vector in length units.

The geometric model is built up by repeatedly adding objects which are attached to objects which are already part of the model. (The first object is selected arbitrarily and assigned a starting point of (0 0).) In order to add the object to the model, its rotation must be determined. If the rotation is unspecified, the function TRITEST is called to test whether the object is part of a triangle; if so, its rotation is computed by the function TRIANGLE. Otherwise, the "normal" rotation for the object, or zero, is assumed. Given the rotation and geometric size of the object, its starting point can be calculated from a point of attachment to an object which is already in the model. The coordinates of the point are calculated for the object in the model, and for the new object assuming a starting point of zero; subtracting the latter vector from the former yields the starting point for the new object. This is illustrated in Figure 5.2, where the new object O_2 is to be added to the model based on its attachment to the existing object O_1 at point P. The coordinates of the point P in the geometric model are computed, and the vector V_2 is calculated by finding the coordinates of P relative to O_2 with S_2 assumed to be zero. Subtracting V_2 from the geometric model coordinates of P gives the geometric coordinates of the new starting point, S_2.

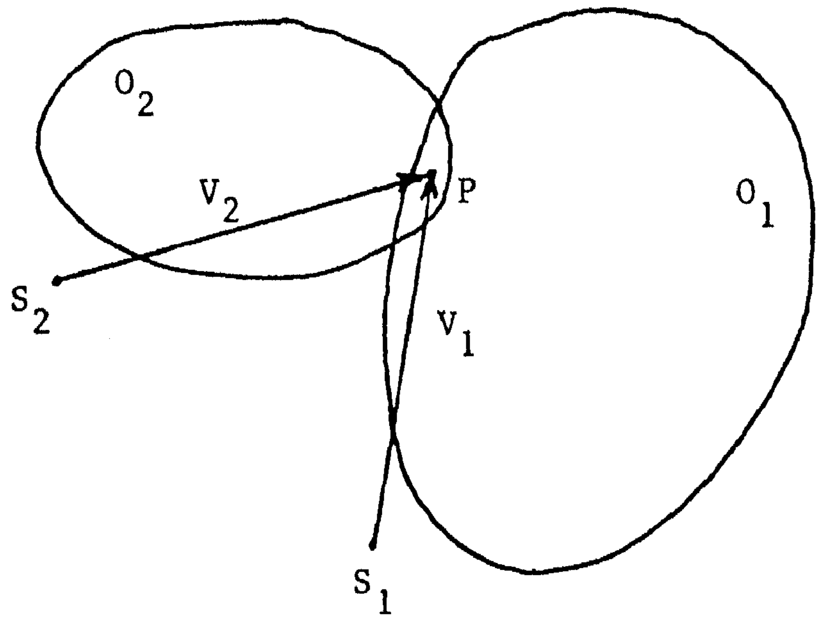


Figure 5.2: Calculating the Starting Point for a New Object

After the starting point of the new object has been determined, the coordinates of all of its attachment points are computed and saved. Any objects to which it is attached which are not part of the model or on the waiting list are added to the waiting list. Finally, the next object from the waiting list is selected to be added to the geometric model. When the waiting list has been emptied, the model is complete.

If three objects are attached to each other so that they form a triangle, it will generally be necessary to solve the triangle for one or more sides and angles in order to properly construct the geometric model. Since the triangle may be implicitly specified by specifying the attachments of the three objects, it is necessary to test objects which have a finite size to see if they are part of a triangle; this test is performed by the function TRITEST. Given an object A, TRITEST looks for objects B and C such that A is attached to B and C and B is attached to C; if such a set of objects is found, TRITEST returns a list of the three objects as its value. This list may then be used as the argument for the function TRIANGLE.

Given a list of three objects which are attached so that they form a triangle, the function TRIANGLE attempts to solve the triangle to find the unknown sides and angles. Since a triangle is solvable given three sides, two sides and an angle, or a side and two angles, there are a number of ways in which the known data for a solvable triangle may be present. TRIANGLE first enumerates the known data for the three objects in the order in which they are given. The function GDIST calculates the geometric distance between the two attachment points for each object. The function GANGLE computes the angle between two objects whose rotations are known. (GANGLE as implemented does not handle all possible cases, but it would be fairly straightforward to make it do so.) Lists are made of the sides and angles, and a transfer is made to the appropriate subsection based on the types of known quantities. (Only the section for solving triangles for which two sides and an angle are given is coded, but provision is made for the other sections.) The triangle is "normalized" by circularly shifting the order of the sides so that the single known quantity (e.g., the known angle) is in the first position; this makes it relatively easy to test for the remaining unknown and solve the triangle. After all unknowns have been found, the triangle is un-normalized by shifting back to the initial order of the objects, and the newly found information is transferred to the objects which comprise the triangle. In the case of computed angles, the function DEFANG defines the rotation of the object based on the angle it makes with other objects in the triangle.

The geometry found in elementary physics problems is usually fairly simple; the solution of a triangle is the most difficult geometric problem which is typically found. EUCLID and its subroutines solve such problems in a general way which is based on legitimate geometric rules, rather than on "canned" formulas which work for particular problems but are not true in general. Geometric programs like EUCLID (but much more sophisticated) might be of great benefit to scientists and engineers for solving problems in geometry, just as symbolic manipulation packages are now used to aid in solving algebraic problems.

Although the present program does not do so, it would be easy to generate a geometric diagram, similar to the "force diagrams" often found in physics texts, from the geometric model of the problem. Such a diagram would be useful if a program similar to ISAAC were to be used for computer-assisted instruction in physics.

5.4 Frame Completion

After the geometric model has been completed, the function CFDRIVER is called to complete the canonical object frame for each object. Since the canonical object frame has already been selected for each object, CFDRIVER simply calls the frame-completion routine associated with the canonical frame for each object.

The primary operation performed during frame completion is the completion of attachment relations by associating appropriate force vectors for each object with the attachment frame. In some cases, the geometric model is required in computing the force vectors. The function CMPATT, which is used for LEVER and PIVOT frames, associates a two-variable force vector with each attachment for which the force vector is unspecified. (A separate force vector is added to an attachment frame for each object which is attached there.) The forces exerted by the object and the geometric position of the point at which each force is exerted are collected and saved on the property list of the object under the indicator FORCES. For a LEVER frame, the location of a PIVOT attached to it is noted if there is one. The function MKWTFRC makes a weight force, exerted on the object at its center of gravity, for a lever object if it has a weight.

In the case of a WEIGHT object, the weight of the object (which must exist since the frame creation routine requires it) is used to make a downward force vector. This vector is inserted directly into the attachment frame.

In the case of a spring (or rope), the force exerted by the spring is equal to the tension in the spring and directed from the end of the spring toward its center. This law is so "obvious" that it is almost never stated in a physics text; nevertheless, it is a physical law of the SPRING and ROPE canonical objects, and is necessary to solve the problems. The frame completion routine for SPRINGs and ROPEs calculates the unit vector from each attachment point to the center of the object. Each component of the unit vector is multiplied by the tension, and the resulting force vector is put into the attachment frame.

A FORCE may be specified as a two-component force vector, or in magnitude and direction form. If the vector form is specified, it is used directly. If the magnitude and direction are used, they are converted to vector form for use in the attachment relation.

A SURFACE is assumed to be a "smooth" surface as found in physics texts; that is, it can only exert a force perpendicular to the surface. The unit vector perpendicular to the surface is calculated and multiplied by a single force variable to give the force vector.

Once the canonical object frame for each object has been completed, the problem model is ready to be turned over to the problem solver.

5.5 Conclusion

The processes of frame selection, geometric model construction, and frame completion which were described in this chapter are relatively simple processes; yet, they are crucial for solving physics problems. We shall argue that selecting and completing canonical object frames is a primary skill which is taught in a physics class, that this skill is taught mostly by example rather than explicitly, and that failure to learn the skill from the examples is what causes people to be "bad" at physics.

6. Problem Solving

Problem solving, as described in this chapter, is the process of writing equations which describe the interactions of objects according to well-known physical laws, solving these equations for the desired unknowns, and printing the answer in the desired form. Compared to the processes of language understanding and frame construction which precede it, the problem solving process seems very simple: it consists mostly of elementary algebra, which is well understood.

6.1 Generation of Equations

Some equations may have been generated directly in response to statements in the problem, e.g. "the man supports twice as much as the boy" (P7). Any existing equations such as these are passed to the equation solver, SOLVEQ, at the beginning of the problem solving process. The remaining equations are generated and solved by the functions ATTDRIVER and PSOLVER.

ATTDRIVER writes equations for each attachment relation according to the physical law that the sum of the x forces and the sum of the y forces must each be zero for a body in static equilibrium. The x and y components of each force involved in the attachment are added to two accumulators using the function SPLUS (symbolic PLUS); two equations are then written setting each of the accumulators equal to zero, and the equations are used as arguments in calls to SOLVEQ. These equations are generally quite simple, and result in a numeric value for a variable or a substitution equation which allows one variable to be rewritten as a function of another. The following equations from the set generated for (P4) are typical of the types of equations generated by ATTDRIVER:

```
(EQUALS 0 FORCE179)
(EQUALS 0 (PLUS -100 FORCE180))
(EQUALS 0 (PLUS (TIMES TENSION173 .7071) FORCE175))
```

The first two equations give numerical values for the variables, and the last equation allows FORCE175 to be expressed as a function of TENSION173, thus reducing the number of active unknowns by one.

The function PSOLVER calls the problem-solving functions which are associated with some canonical object frames to write equations for objects of that type and solve them. The objects which have desired unknowns associated with them are selected first, followed by objects which involve other unknowns. After a problem-solving function has been called for an object, a test is made by TESTSOL to see whether values have been found for all of the desired unknowns; if so, PSOLVER returns, without requiring that values be found for the other variables. In the present system, there are problem-solving functions for SPRING and LEVER canonical objects.

SOLVESPRING generates a single equation for the spring law, which states that the tension on a spring is equal to the spring constant times the distance the spring is stretched. The other laws which apply to a spring, namely that the sum of the forces exerted on it must be zero (in static equilibrium) and that the force exerted by the spring is directed from the end of the spring toward its center, are made true implicitly by the way the force vectors are generated by the frame completion routine for springs.

SOLVELEVER generates the three equations which govern a rigid body in static equilibrium, namely that the sums of forces (in the x and y directions) must be zero and that the sum of moments on the body must be zero. If a PIVOT object attached to the LEVER was found by the frame completion routine for the LEVER, the pivot point is chosen as the point around which moments are summed; otherwise, a point whose position is known and which has one or more unknown forces exerted there is chosen.

The number of equations generated for a single problem is surprisingly large: between seven and thirteen equations per problem, with an average of about ten equations per problem. For a reasonably skilled human problem solver, all of our sample problems can be solved using two equations except for (P4), which requires three. This large discrepancy suggests that the human problem solver performs a number of steps (which become largely subconscious with practice) to reduce the number of equations which must be written. Some equations, such as those involving horizontal forces in a problem where all the significant forces are vertical, are simply ignored. Others, such as our attachment equations, are eliminated by substitution of variables which is done mentally. Since these processes are largely subconscious in a skilled person, it may be difficult to teach them to a person who is unable to acquire the skill by watching the solution of example problems. A program such as ISAAC, which makes all of the steps explicit, might be useful for teaching physics to such persons.

6.2 Equation Solving

The equations which are generated to describe the interactions of objects in the model of the problem are solved by a set of routines for simplifying expressions and solving linear equations. This small symbolic manipulation package is fairly primitive compared to the state of the art in symbolic manipulation. Much more powerful packages exist, such as MACSYMA [Moses 74]; a more powerful program for solving physics problems could easily be interfaced to such a system (as was Charniak's CARPS program), allowing problems involving more complex mathematics to be solved.

Equations are solved by the method of substitution, that is, by expressing one variable as a function of another variable and substituting this function for the variable when it occurs in other equations. Since this reduces the number of active variables by one, the process can be repeated until a value is found for some variable. This value can then be substituted into the functions to calculate the values of other variables, and so forth until values have been calculated for all of the variables. This method is the one generally used by humans for solving simple equations. For equations as simple as those generated for our sample problems, the method works well and is reasonably efficient; for more complex equations, other methods (such as Gaussian elimination) would be needed.

Equations are written using the five functions SPLUS, SMINUS, SDIFF, STIMES, and SQUOT. These functions perform some elementary simplifications on their arguments when possible; for example, (SPLUS 0 x) = x, where x is any expression. If no simplification is possible, these functions construct a prefix subexpression using the corresponding LISP function name.

The function SIMPLIFY may be used to simplify an expression (not necessarily an equation) by operations such as removing double negations, combining constant factors of a variable, and so forth. SIMPLIFY is used in making the geometric model as well as in the problem solving process. The function SIMVECT simplifies a vector by calling SIMPLIFY for each component.

The function COPYSUB copies an expression, substituting the VALUE of each variable (gotten from its property list) for the variable if the value is defined. Such a value may be either a numeric value or a substitution function in terms of another variable.

SOLVEFOR solves an equation for a given variable, which should occur only once in the equation. This is easily done by finding a path from the root of the tree representing the equation to the desired variable. Inverse operations are then generated along this path to bring the desired variable to the top. For example, to solve the equation (EQUALS A (TIMES B C)) for C, we generate the inverse operation QUOTIENT to obtain (EQUALS C (QUOTIENT A B)). A similar process is easily applied to an arbitrarily large expression.

The function LISTVC examines an expression and constructs a list of all the variables and constants used in the expression and the number of times each symbol appears. This list is used by SOLVEQ to guide the equation solving process.

The function SETEQUAL is used to define the value of a variable based on an equation. The equation is solved for the value of the variable using SOLVEFOR; this value is put on the variable's property list under the indicator VALUE. The value is then substituted in the value expression for each variable whose value is expressed as a function of the variable just defined; a list of all such variables is stored on the property list of the variable under the indicator USEDIN. For each such variable, COPYSUB is used to copy its value, substituting the new value of the variable just defined. The resulting expression is made into an equation, and SETEQUAL is called again (recursively) to define the new value of the variable. In this way, a new definition of a variable is propagated to all the variables whose values are dependent on it. Since the new definition of a variable may make some saved equations solvable, each equation on the list EQUATIONS is copied using COPYSUB; SOLVEQ is then called to solve the resulting equation.

SOLVEQ attempts to solve an equation; if it succeeds, the results are propagated to related equations and variables, which may lead to the solution of additional equations. SOLVEQ first uses COPYSUB and SIMPLIFY to substitute values for any variables whose values are known or defined in terms of other variables and simplify the resulting expression. LISTVC is then called to list the number of constants and variables in the expression and the number of times each occurs. If there are more than two variables, the equation is considered temporarily unsolvable and is put on the EQUATIONS list. If there is only one variable in the equation, SETEQUAL is called to define the value of the variable based on the equation; SETEQUAL will propagate the consequences of this definition, possibly causing SOLVEQ to be called again. If the equation involves two variables, an attempt is made to solve for one variable in terms of the other. (If both variables occur more than once in the equation, it is saved on the EQUATIONS list). After defining one variable as a function of the other and adding it to the USEDIN list of the other, the new value of the variable is propagated to all members of its USEDIN list, which is then set to NIL. The propagation is done by using COPYSUB and SIMPLIFY on the value of each variable on the USEDIN list to substitute the value of the new variable, then adding the variable to the USEDIN list of the other variable in the equation. Thus, for example, if a were defined in terms of b as a = f(b) and b was then redefined as b = g(x), we would redefine a as a = f(g(x)) and put both a and b on the USEDIN list for x. If the newly defined variable is used in any of the equations on the EQUATIONS list, the new value is substituted using COPYSUB, and SOLVEQ is called recursively to attempt to solve the resulting equation.

The time required to solve a set of equations varies, but typically is about one second (using interpreted LISP on a CDC 6600) for our sample problems.

6.3 Answer Generation

Once the values of the desired unknowns have been calculated, answer generation is fairly easy. The name of an answer-generation routine and the object to be used as its argument are saved (for each part of the answer) on the list SYSREPLY in response to the question asked in the problem statement. The function PRTSOL evaluates each of the members of this list in turn, putting commas between the generated answers.

PRTVAR prints the value of a variable and the units associated with it. If the answer is an expression which contains constants, the function EXPLCON is called to explain each constant. EXPLCON gets the object with which the constant is associated and the attribute which it measures from the constant's property list, and outputs these in a standard format, e.g., "where LENGTH76 is the length of the pole" (P2). EXPLCON is called by most of the answer generation routines if the answer is an expression involving constants.

PRTFV prints the two components of a force vector in parentheses, separated by a comma. PRTMAG and PRTDIR compute and print the magnitude and direction, respectively, of a force vector.

PRTLOC generates a description of a location; typically, a location which is the object of a question will be represented as a point which is a certain distance from a known point, with the distance an unknown. PRTLOC prints the distance from the known point, then generates a description of it. If the known point has a name, the name is printed following the location name, as in "end (A)"; if it has a meaningful SELECT modifier, the modifier is printed with the location name, as in "the heavy end". Otherwise, an attempt is made to find an object which is attached at the known point; if such an object is found, it is used to describe the location, as in "7.4 ft from the boy" (P7).

All of the answer generation functions comprise about two pages of LISP code, compared to 44 pages of code for input parsing and semantics. Language generation to describe the answers to physics problems is a relatively easy task, since the "objects" to be described are so simple. [Simmons and Slocum 72] describe a method for generating fairly complex sentences using semantic networks and an ATN grammar.

7. Picture Construction

The process of constructing a picture from the internal model of the problem is in many ways similar to the process of constructing the geometric model of the problem; however, there are some significant differences. While a WEIGHT object is represented as a point in the geometric model, it must be drawn at a reasonable size. A size must be chosen for each object whose size is a symbolic constant, and relative positions on the object must be scaled accordingly. The size of the picture must be scaled to the space available for the drawing, independent of the size of the objects in the problem.

Construction of a picture is done in two stages. First, a picture model is constructed, specifying the position and size of each object. From this model, global offsets and a scale factor are computed to properly scale and position the picture within the drawing area. Finally, picture generation functions are called to generate each object in the picture.

7.1 Constructing the Picture Model

The picture model for the problem is constructed by the function DIAGRAM. Each object in the picture is assigned a starting point and a size, which are stored under the property list indicators STVAL and PSIZE, respectively. The rotation, stored under the indicator ROTN, is the same as for the geometric model. A set of objects arranged in a picture is represented by a "picture frame", or PFRAME, consisting of a set of minimum and maximum x and y values which bound all of the objects in the picture, and a list of the objects in the picture frame. The starting point value for each object is relative to its picture frame set. Two picture frame sets may be combined by specifying the coordinates relative to each of a point which is to be made common to both. A new set of bounds is computed, and objects from one picture frame set are incorporated into the other by adjusting their starting points and adding them to the object list of the other picture frame set.

DIAGRAM first calls the function PICSCALE to determine the picture scaling factor for each object. Some objects are scaled according to the value of a certain attribute: poles according to their length, weights according to their weight, springs according to their spring constant, and so forth. If such an attribute is defined for an object and the attribute has a numerical value, the attribute name and value are saved on the property list of the object under the indicator SIZEDET. In addition, PICSCALE keeps a list of the different attributes and the maximum value found for each attribute. This list and the saved SIZEDET value are used later to determine the scale factor to be used for each object in the picture. If a scaling attribute is not specified for an object or is not defined as a numeric value, a test is made to see if there is a special function to determine the scaling factor for the object; such functions exist for FORCE and rigid body objects. PSIZEFORCE computes the magnitude of a two-component force vector and returns this value as the scaling factor. In addition, it computes the rotation of the force vector and stores this on the force object's property list under the indicator ROTN. PSIZERB is used to compute the scaling factor for rigid body objects, including both SURFACE and LEVER objects. The attachment points of the object are examined. If the attachment points have numeric geometric positions, then the largest distance in the x or y directions between two attachment points is used as a LENGTH scale factor. Thus, in the picture for (P8), the unspecified length of the vertical wall is set equal to the distance between the rope and ladder which are attached to it. If numeric values are not available for the attachment points, but there are some numerical relative positions, the minimum distance from the center of gravity of the object to its boundaries in the x direction is made equal to the maximum relative position offset; this guarantees that all of the relative positions will be drawn within the area of the object in the picture. Thus, in the picture for (P20), the seesaw is made large enough so that both boys are drawn as being on the seesaw, with their relative distances from the center in correct proportion. If neither of the above methods can be used, the maximum dimension of the drawing of the object is used as the scaling factor with the artificial attribute name CLENGTH. This will cause objects of unspecified size to be drawn at a size proportional to the unscaled size produced by their drawing programs.

Once the picture scaling factors have been computed by PISCALE, DIAGRAM constructs the picture model in a manner similar to the way the geometric model is constructed by EUCLID. An initial object is chosen arbitrarily to start the picture. Objects are added to the picture by combining a new object with the existing picture at a point of attachment between the new object and an object already in the picture. Objects which are attached to the new object but are not in the picture are added to the waiting list of objects to be added to the picture. The subroutines which are used in performing this process are described below.

MAKEPF is a function which makes a picture frame for a single object. In order to do so, it must compute the drawing size to be used for the object and a set of picture frame boundaries which will completely enclose the drawing of the object. Some objects, such as a door or person, have special size computation routines; these are used to compute the size for an object if they are defined. If a SIZEDET attribute and value were found for the object, its size is scaled in proportion to the maximum value found for that attribute in the problem. (For some objects, such as WEIGHT objects, the picture could be made more realistic by using a special function to make the picture size proportional to, say, the square root of the weight proportion. This was done in an earlier version of the program, but is not in the present version.) If all else fails, the scale factor is set to one. The size computation routine for a door computes separate scale factors for the height and width of the door. The picture making function for a door draws a square, but with separate scale factors for the x and y coordinates; this allows a door to be drawn to scale for the specified width and height. The size computation routine for a person uses the SIZEDET value if it is available. Otherwise, a test is made to see if the person has a RESTRICT YOUNG modifier; if so, the size is reduced slightly. Thus, in (P2) the boy is drawn slightly smaller than the man. If the size of an object is defined in terms of length, the scale factor between length and picture size is computed and stored on the object's property list under the indicator PSCALE. The picture size (which is a vector, although in most cases only one component is used) is stored under the indicator PSIZE. The initial picture frame is computed by scaling the basic picture size (stored under the indicator FRMSCL in the GEOMODEL of the object) by PSIZE; the minimum values and starting point are defined by convention to be (0 0). If the object is rotated, its picture frame is recomputed by ROTPF. This is done by computing the positions of the corners of the picture frame after rotation, and computing a new frame which encloses all of these points. This process is illustrated in Figure 7.1. As the figure shows, the rotated picture frame may be somewhat larger than needed to contain the object. However, it is easily computed in this manner, and is certain to be large enough. The only effect on the final drawing from a picture frame which is too large is to make the drawing slightly smaller than it might have been.

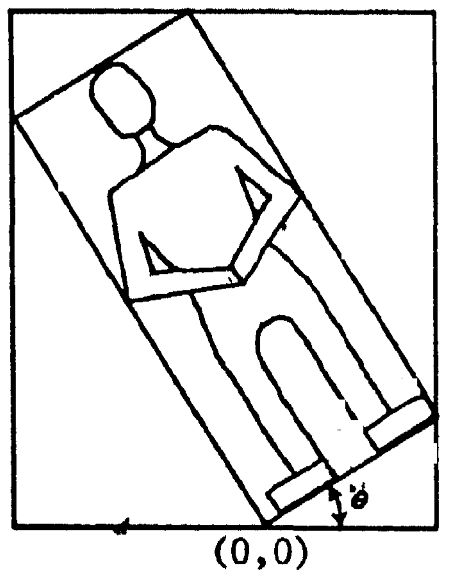


Figure 7.1: Computing Picture Frame for a Rotated Object

After a picture frame has been made for an object by MAKEPF, DIAGRAM searches the attachment relations of the object to find a point at which it is attached to an object which is already in the picture. When such an attachment is found, PICTLOC is called twice to find the position of the point of attachment on the new object relative to its picture frame and the point of attachment on the other object relative to the larger picture frame. These two positions are then used in a call to COMPFRM to combine the new object's picture frame into the total picture frame which is being constructed. Finally, objects which are attached to the new object and are not already in the picture or on the waiting list are added to the waiting list. After all the objects on the waiting list have been processed, DIAGRAM exits with the completed picture frame set as its value.

PICTLOC calculates the position of a point on an object relative to the object s picture frame. When the geometric size of the object and the name of the location are specified, the position is calculated by simple vector operations as described in section 5.3 and illustrated in Figure 5.1 for geometric positions. If there is a relative position offset from a known location and the geometric size of the object is a symbolic constant (as in (P2), where the weight is attached 0.75 times the length of the pole from the boy), PSIZERB will have made a CLENGTH size factor for the object. When it does so, PSIZERB also defines the VALUE of the length constant to be the same factor. Thus, by performing COPYSUB and SIMPLIFY on the relative position expression, the correct proportional length on the object in the diagram is obtained. (If the relative position were a function of other constants, this procedure would fail, and the relative position would be ignored. This does not happen in our sample problems.) If no location name is specified for the object, a default location must be found for the object's point of attachment in the picture. (This is not usually necessary in the geometric model, where such an object is typically treated as a single point.) The default location for the object may be stored on the property list of its token word, or there may be a function to compute it. Such a function is provided for PERSON objects; this function selects HANDS as the default location if the person is modeled as a PIVOT object, or FEET otherwise. (Some verbs, such as SIT and STAND, specify the location as part of the verb semantics, so that a default location is not needed.)

COMPFRM combines two picture frame sets, given a point relative to each picture frame which is to be made a common point in the combined picture frame. A constant translation vector is easily computed from the two given points; by adding this vector to the coordinates of each point in the second picture frame, the coordinates of the corresponding point in the first picture frame (which will become the combined frame) are obtained. Since the position of each object is relative to its starting point, only the starting point coordinates of the objects in the second picture frame need to be recomputed. A simple loop is used to recompute the starting point of each object in the second picture frame and add it to the object set of the first picture frame. The picture boundaries are recomputed by calculating the offset positions of the boundaries of the second picture frame, then choosing boundaries for the combined set which enclose both of the component picture frames. This process is illustrated in Figure 7.2, where the frame drawn with solid lines is the combined frame for the two smaller frames drawn with dotted lines. (The solid lines are drawn outside of the dotted lines for clarity where they would be in the same place.)

Many of the functions used for constructing the picture model and drawing the pictures are similar or identical to those used in [Simmons and Bennett-Novak 75]. The picture frame concept used in constructing the picture model is so simple and obvious that it probably is not new; it is described here for completeness.

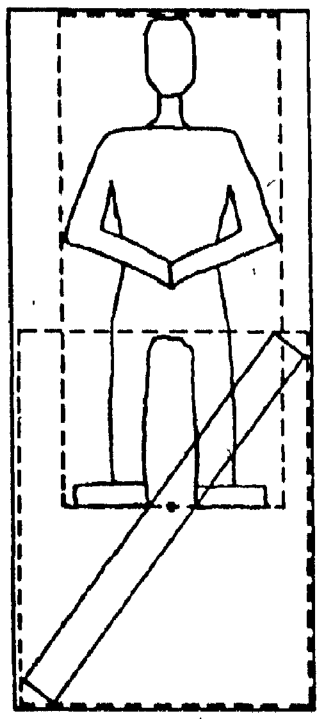


Figure 7.2: Combining Two Picture Frames

7.2 Drawing the Diagram

The completed picture frame set is passed as an argument to the function DRAWPICS, which controls the drawing of the picture. The size of the picture frame in the x and y directions is computed from the frame boundaries. These size values are used in conjunction with the size of the available picture area to set the global constant GLOBALSIZE so that the finished picture will occupy 0.9 of the available space along its maximum dimension. The frame boundaries and global size are used to compute an offset base vector so that the picture will be centered in the available area in each dimension. For each object which is to be drawn, DRAWPICS calculates the proper offset starting position, sets the initial position and heading, and calls the program to draw the object with the size as an argument.

The functions used from LISP to draw the pictures have a structure similar to the LOGO language of Papert [Papert 72]. The "turtle" concept of plotter commands used in LOGO is convenient for drawing objects because an object can be drawn in any orientation if the turtle is initially pointed in the right direction.

8. Conclusion

In the preceding chapters, we have described a particular program which is capable of reading, understanding, solving, and drawing pictures of a class of physics problems which are stated in English. In this chapter, we shall examine the methodology of this research, some directions for future research which are suggested by this work, and potential applications of programs similar to this one. Finally, we present some data on the program's size and execution time, and examine what extensions would be necessary to handle additional problems.

8.1 Methodology

The area of physics problems involving rigid body statics is certainly a "micro-world", and a fairly small one at that; however, in the opinion of this author, it is a fruitful one for research in computational linguistics. The area is sufficiently circumscribed to be tractable for programming, but still involves a number of interesting problems—many more than a casual glance at the sample problems would suggest. In some cases, the correct parsing of a sentence depends on the particular relations of objects in the model of the problem, thus forcing the integration of syntax, semantics, and world knowledge in the parsing program. The difficult problem of referent identification must be solved (though of course in a limited way) for both physical objects and locations. Different sense-meanings of words (particularly verbs and prepositions) must be disambiguated. Canonical object frames must be selected to represent objects in the model, and inferences must be made to construct a complete and consistent model. Geometrical models of the problem must be constructed both for solving the problem and for drawing the picture. Thus, although the problem solving is specific to the area of physics problems, the process of understanding the English statement of the problem involves a number of interesting sub-processes which are likely to be important in any language-understanding program. The area of physics problems is a good one for investigating these sub-processes because there is a relatively clear understanding of what the result of understanding a physics problem must consist of: a model of the problem in which the attributes and relationships of objects are represented with sufficient specificity to allow equations to be written describing the interactions of the objects and to allow a diagram of the problem to be constructed.

The twenty sample problems used to test the program were selected before the major version of the program was written. (When the program was almost complete, one problem which involved a great deal of world-knowledge required only for that problem was deleted and replaced by another problem.) Thus, in a sense, the program was written to solve twenty specific problems—not a very large number. However, we tried to solve the problems in a legitimate, general way, using a minimum of "tricks" We hope (but have not yet shown) that the program could be expanded considerably without rewriting very much of the existing code, and that it could be made to solve twenty more problems of the same type with relatively little difficulty. (Adding the problem which replaced the deleted one required only a semantic routine for one word.) The use of twenty preselected problems by several different authors actually made the program much more difficult than it might have been. Almost every problem had some idiosyncracy which required additional capabilities of the program or prevented an easy trick from being used in a superficially similar situation in another problem. On the other hand, the diversity of the problems led to the discovery of many interesting regularities which would have been missed if we had (say) selected problems that an existing program could solve or edited the problems to make it easier on the program. Thus, in a sense we are treating computational linguistics as an experimental science, in which the experimental data are existing examples of linguistic performance by competent native speakers, and in which the goal of the research is the production of programs which can adequately understand the examples of language performance. In this author's opinion, this is a valuable approach. Many interesting problems which would never have been noticed were made glaringly apparent when the program failed to work. Likewise, many regularities were found by suddenly realizing that a subroutine almost identical to the one needed for the current task was written earlier. This approach does not replace theory, but rather lays the groundwork for theories which can be powerful because they account for a large number of examples of linguistic performance. Because a program such as this one deals with the whole process of understanding language, it can serve as the basis of a more complete theory of language, rather than a theory which deals only with a narrow aspect such as syntax.

8.2 Directions for Future Research

In this section, we will comment briefly on some interesting possibilities for future research which are suggested by some of the techniques used in this program.

The SFRAME (semantic frame) concept, in which a semantic interpretation is assigned to a phrase, inferences are made to fill in missing arguments of the semantic frame, and specialist routines are associated with the frame to perform tasks associated with that type of semantic object, is an interesting one. Only a few types of SFRAMEs are used in ISAAC; it would be interesting to see if this technique is useful for understanding language in other areas besides physics problems, and to investigate how the use of SFRAMEs might be integrated into the parsing process.

The process of referent identification is an important one for understanding virtually all types of language. The procedures used for referent identification by ISAAC are fairly rudimentary, are specific to the area of physics problems, and deal only with extensionally specified referents. This area deserves much more research to determine rules for referent identification in wider contexts and ways to represent and use intensionally specified referents. (For example, when identifying the phrase "the 8 million people of New York", we would like to create an intensional referent, rather than creating 8 million PERSON referents.) While a PLANNER theorem can be used as an intensional representation, it would be desirable to have a representation which is more accessible as a data object than a PLANNER theorem is.

The concept of the canonical object frame (due primarily to Minsky) is a powerful one. The canonical frames dealt with in ISAAC are particularly simple ones. It would be interesting to develop canonical frames for more complex objects in physics and engineering. Analysis done by engineers is based very heavily on the use of canonical object frames; it would be interesting to study how such frames are selected and used, and how such frames are used when the modeled object doesn't fit the canonical frame very well (as, for example, when piecewise linear analysis is used to simulate a nonlinear device characteristic).

Since the model of a problem is constructed before equations are written to solve the problem, the existing program could be used as a test bed for investigating other strategies for solving this type of problem. It is clear that the present method generates many more equations than are usually generated by humans; it would be interesting to investigate how the few critical equations could be written more directly, and what rules might be used to select and inhibit such shortcut methods.

It would be interesting to extend ISAAC to additional types of physics problems. Although the present program handles only static problems, most dynamic problems are handled as a sequence of (usually two) "static" situations with a specific relationship (such as a conservation law) which holds between the two situations. [de Kleer 75] investigates the interaction of qualitative and quantitative knowledge in solving dynamic problems.

8.3 Potential Applications

Programs similar to ISAAC, but with expanded capabilities, might find useful application in two areas: as engineering assistants, and in technical education.

There are many specialized programs to aid in the analysis of engineering problems. Often, however, these programs are not used for problems of small to moderate size, either because considerable knowledge of a system is required in order to use it (and it isn't worth the effort to acquire this knowledge for a small problem), or because the data must be laboriously prepared in a rigidly specified format. A program which, like ISAAC, could accept a problem statement in English could overcome these problems.

Another potential application of a program such as ISAAC is in computer-assisted instruction (CAI). Other CAI programs using natural language, such as the SOPHIE program [Brown and Burton 75] for teaching electronic circuit analysis, have been successfully developed and used. ISAAC is particularly interesting for application in this area because of the insights it gives into the problem solving process. The primary skill which is taught in a physics class to enable the students to solve problems is the application of physical laws to actual problems. The physical laws themselves are of less importance—in fact, not all of the laws necessary to solve a problem are taught explicitly (such as, for example, the "laws" that the force exerted by a rope is directed toward the center of the rope and cannot be negative). Many of these laws are "buried" in the procedures for setting up a problem solution. These procedures are usually taught by example—often with many steps left out. The student who does not understand how the missing steps are being skipped may become completely lost. A program such as ISAAC could be valuable for teaching physics (and similar subjects) because it could present all of the steps in detail, progressing to more abbreviated forms once the student grasped the steps that were to be skipped.

8.4 Program Statistics

The time required by the program to process a complete problem (including parsing, semantics, problem solving, and picture generation) averages about 10 seconds per problem, using interpreted LISP on a CDC 6600. This is really quite fast. By using compiled LISP instead of interpreted LISP, an increase in speed of several times might be obtained, so that the processing time per problem in a "production" system might be reduced to a second or two. It took the author about 45 minutes to solve all the problems (drawing only minimal diagrams as a mnemonic aid); two of the answers were wrong due to "careless" errors. Thus, even in its present form, the program is more than ten times as fast as a human test subject and (assuming the problem is within its range of competence) more accurate.

The program is coded in UT LISP 1.5, using a virtual memory package for function definitions which was written by Mabry Tyson. Virtual memory is particularly good for programs such as this one because it allows semantic functions for a large vocabulary to be available without clogging the machine when they are not in use. Some of the standard transcendental functions needed for the geometry and picture generation were coded in LAP to increase their execution speed.

The complete program comprises about 5000 lines of LISP source code, including comments. (This is admittedly an imprecise measure of the program's size.) Breaking the program down roughly into functional categories, the percentages of the total code in each of the categories are approximately as follows:

Syntax	15%
Semantics	30%
Canonical Frame Programs	8%
Geometric Model	4%
Problem Solving	2%
Symbolic Algebra Package	11%
Answer Generation	3%
Picture Model	7%
Picture Generation	7%
Lexicon and Other Data	8%
Miscellaneous	5%
	100%

The current version of the program has a vocabulary of about 200 words. Some of these (for example, different ways of writing measurement units) do not appear in the sample problems. Not counting different forms of the same root word, the twenty sample problems use a total of 135 words. It is interesting to graph the number of words required as a function of the number of problems, even though this is somewhat dependent on the arbitrary ordering of the problems. Such a graph is shown in Figure 8.1; the graph is extended to include the vocabulary for five additional problems, which are discussed in the next section. The graph suggests that twenty problems (even though they are of the same type) are not enough to reach a plateau where the existing vocabulary will handle many new problems. In the next section, we discuss the program's ability to handle new problems.

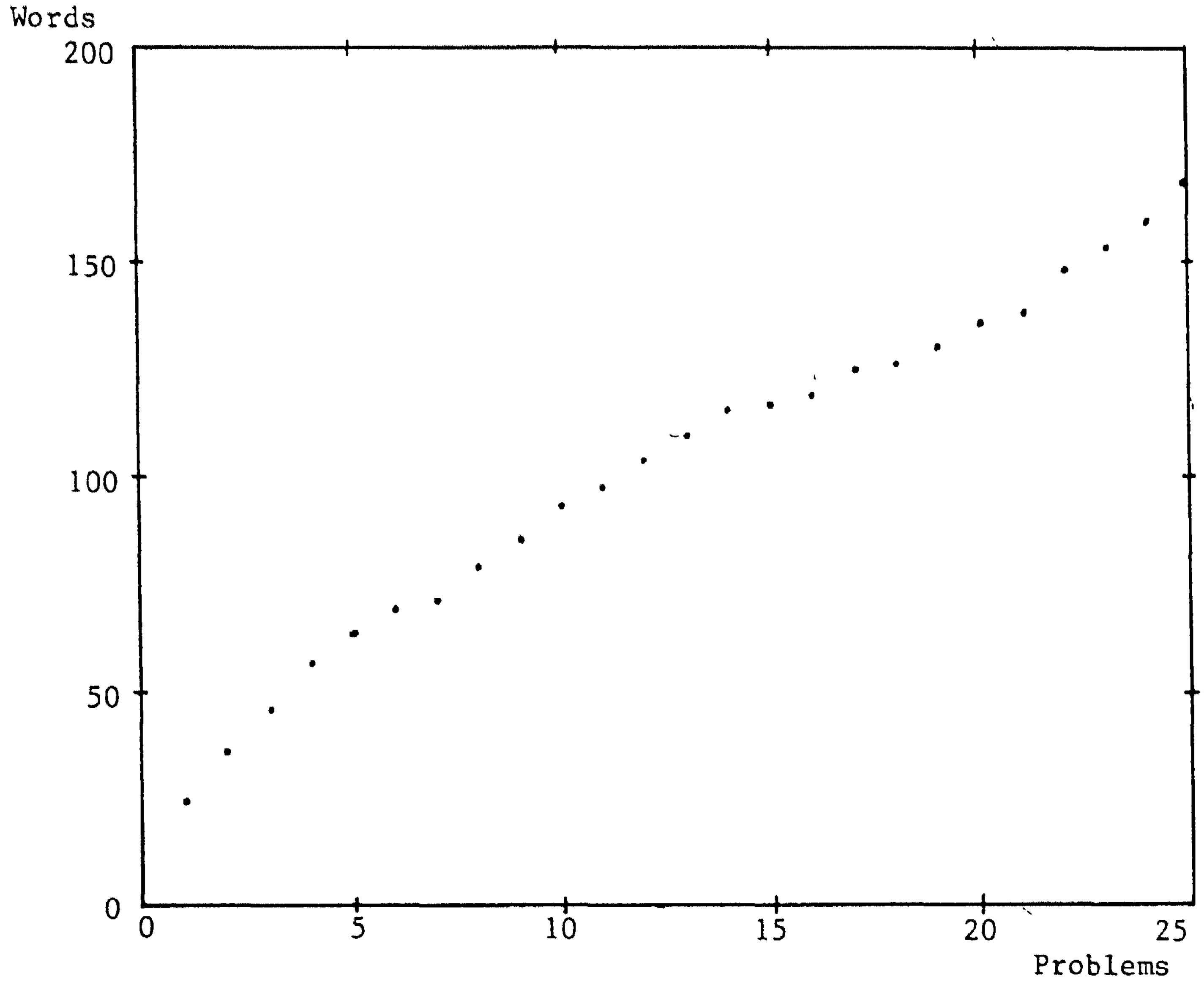


Figure 8.1: Required Vocabulary as a Function of Number of Problems

8.5 Handling Additional Problems

The ultimate test of an artificially intelligent program is its ability to handle new situations for which it was not specifically programmed. Unfortunately, many artificial intelligence programs turn out to be "toy" programs which cannot solve many new problems beyond the few test cases used and cannot easily be extended. It is of interest, therefore, to examine the ability of ISAAC to solve new problems and, more important, to examine the specific improvements (in the many abilities of the program) which are required to handle new problems. In order to do this, we asked our colleague, Michael K. Smith, to select independently five additional test problems. The restrictions on this selection were that the problems should be problems involving rigid body statics, and that they should be stated in English without requiring a diagram as part of the problem statement. The five additional problems are reproduced below.

P21. A uniform steel meter bar rests on two scales at its ends. The bar weighs 4.0 lb. Find the readings on the scales.

P22. A 60 ft ladder weighing 100 lb rests against a wall at a point 48 ft above the ground. The center of gravity of the ladder is one-third the way up. A 160 lb man climbs halfway up the ladder. Assuming that the wall is frictionless, find the forces exerted by the system on the ground and the wall.

P23. A uniform beam is hinged at the wall. A wire connected to the wall a distance d above the hinge is attached to the other end of the beam. The beam makes an angle of 30 deg with the horizontal when a weight w is hung from a string fastened to the end of the beam. If the beam has a weight W and a length l, find the tension in the wire and the forces exerted by the hinge on the beam.

P24. A door 7.0 ft high and 3.0 ft wide weighs 60 lb. A hinge 1.0 ft from the top and another 1.0 ft from the bottom each support half the door's weight. Assume that the center of gravity is at the geometrical center of the door and determine the horizontal and vertical force components exerted by each hinge on the door.

P25. An automobile weighing 3000 lb has a wheel base of 120 in. Its center of gravity is located 70 in behind the front axle. Determine the force exerted on each of the front wheels (assumed the same) and the force exerted on each of the back wheels (assumed the same) by the level ground.

These problems are taken from *Physics* [Halliday and Resnick 67]*, pages 327-339. This is a somewhat harder book than the texts from which we took the original twenty sample problems; nevertheless, all of the new problems except (P23) are within the existing capability of the problem-solving, geometry, and picture-generation parts of the program. However, the program could not complete any of these problems without some modifications. In order to solve all five of these problems, it would be necessary to extend the capabilities of the program in the areas of vocabulary, grammar, world knowledge, and algebraic manipulation. We do not feel that these modifications would be too difficult, and we believe that they could be made within the existing framework of the program. In the sections below, we consider the specific extensions needed in each of these areas to solve the additional problems.

*Copyright 1967 by John Wiley & Sons, Inc. Used by permission.

8.5.1 Vocabulary

Each of the new problems requires additional vocabulary. The average increase of seven words per problem is higher than that of the last few problems of the original set, probably due to the fact that the problems are written by different authors and are somewhat harder. The new words required for each problem are listed below. Of the thirty-six words, ten (those marked with an asterisk) could be added trivially as simple lexicon entries or as synonyms of existing words. For example, "one-third" could be defined as a number with a value of 0.33333333; "connect" could be made synonymous with "attach", and "wire" and "string" could be made synonymous with "rope"

P21	meter bar*
	scale
	reading
P22	above
	ground
	one-third*
	way
	up
	climb*
	halfway
	assume
	frictionless
	system
P23	hinge [verb]*
	wire*
	connect*
	distance
	string*
	fasten*
	length
P24	another
	each [pronoun]
	half*
	geometrical
	force [adj]
	component
	high
	bottom
P25	wheel base
	front
	axle
	wheels
	same
	back
	level*
	behind

Of the remaining words, some (such as "wheel base") are useful only for individual problems; however, there are still a number of more general words (such as "above" and "distance") which are likely to be used in a number of problems out of a large sample. This seems to indicate that it would take a much larger vocabulary (perhaps twice as large) to include most of the "general" words likely to be encountered in this type of physics problems. It also indicates that several times more than twenty test cases would be needed before we could have confidence in the program's ability to solve a new,, independently selected problem.

8.5.2 Grammar

There are several constructions in the new problems which are not handled by the existing grammar. We shall discuss these below, with the caveat that it is easy to overlook subtle features of sentences which might confuse the existing grammar and require some debugging.

In (P22), the phrase "one-third the way up" would not be handled by the present grammar. Such a phrase would become a type of LOCPART SFRAME, with inferences required to determine the object involved, the starting point for the relative position, and the length of the object. A slight grammar extension might be required to handle the initial clause "assuming that . . .".

In (P24), an extension would be necessary to accept the "each" in the second sentence. Extensions would also be needed to accept "half the door s weight", both to handle the "half" and to accept the possessive form of nouns as an adjective (this would not be hard, since possessive pronouns are already handled). An extension would be needed for the compound adjectives in "horizontal and vertical force components".

In (P25), it would be necessary to handle the two parenthetical expressions "(assumed the same)".

8.5.3 World Knowledge

"World knowledge", as we use the term here, is knowledge of the usual relationships and. features of objects which is used in making inferences used to understand a problem. Additional world knowledge is needed for several of the new problems.

In (P22), we need to infer that the bottom of the ladder is resting on the ground. Similar knowledge is needed for (P25), where we need to infer that the ground supports the automobile in four places (the four wheels). In (P24), we need to know that (whatever their vertical position) the hinges are on one side of the door. This would require additional semantic routines to control the generation of these locations.

Additional research on ways to represent and control world knowledge such as that described in this section would be very valuable.

8.5.4 Comments on Individual Problems

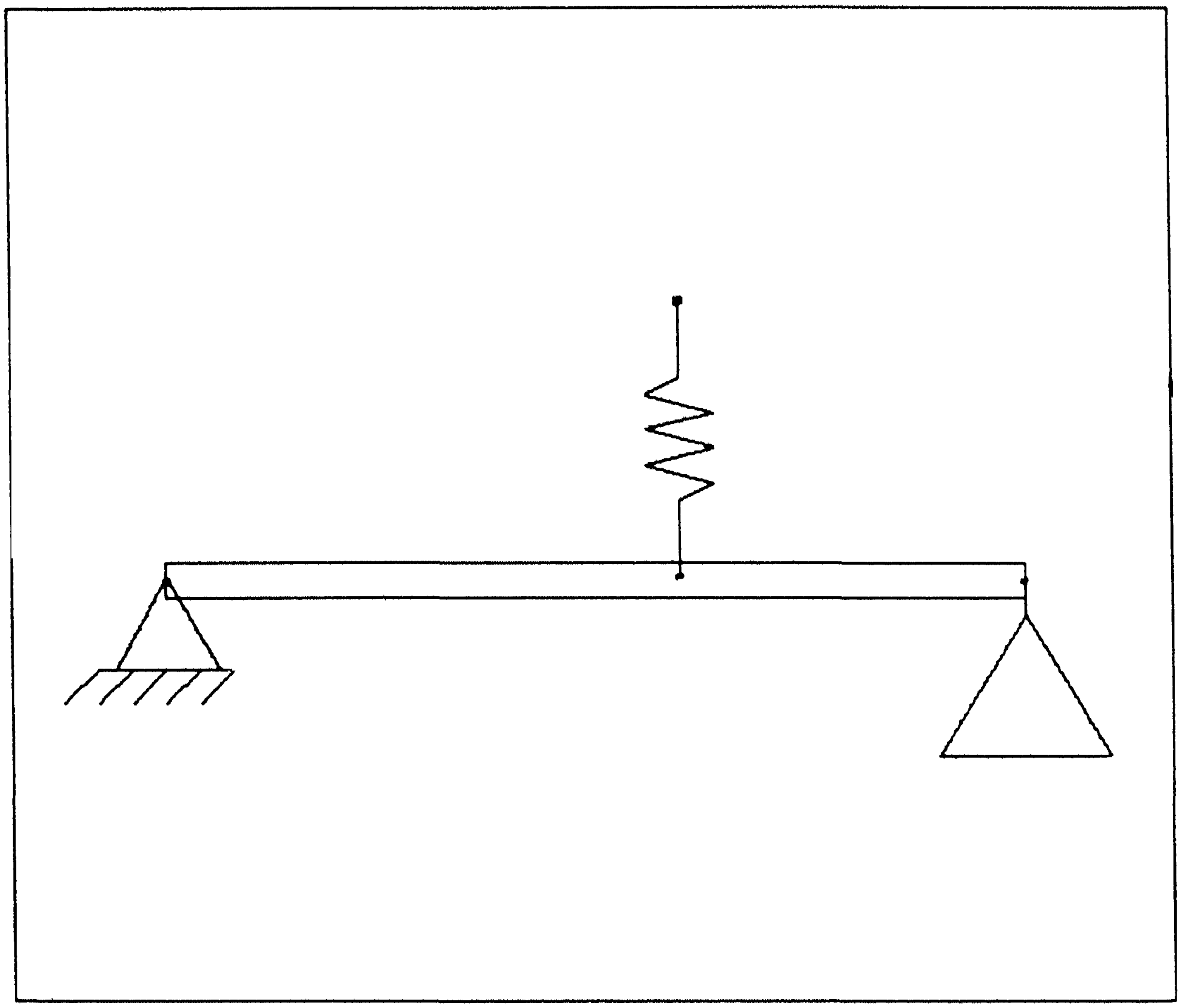
(P21) is of course very simple. If we substituted another word (say "supports") for "scales" and substituted "forces" for "readings", the present program could solve it. To handle the problem as stated, we would need to add a SCALE canonical object (which has a reading equal to the force on it) and add a drawing program to draw a scale.

In (P22), we could take a static view of "climbs" and make it equivalent to "stands". Although the program automatically assumes that walls are frictionless, it would be easy to write a semantic routine to set a zero coefficient of friction if desired. A semantic routine would be needed to identify the referent of "system".

(P23) is beyond the algebraic capabilities of the present program, since it involves algebraic arguments of transcendental functions. It would not be hard to allow this, although the resulting expressions might be intractable for the present expression simplifier and equation solver. The present program would work if d and l were constants.

(P24) would present no problems beyond the ones previously mentioned.

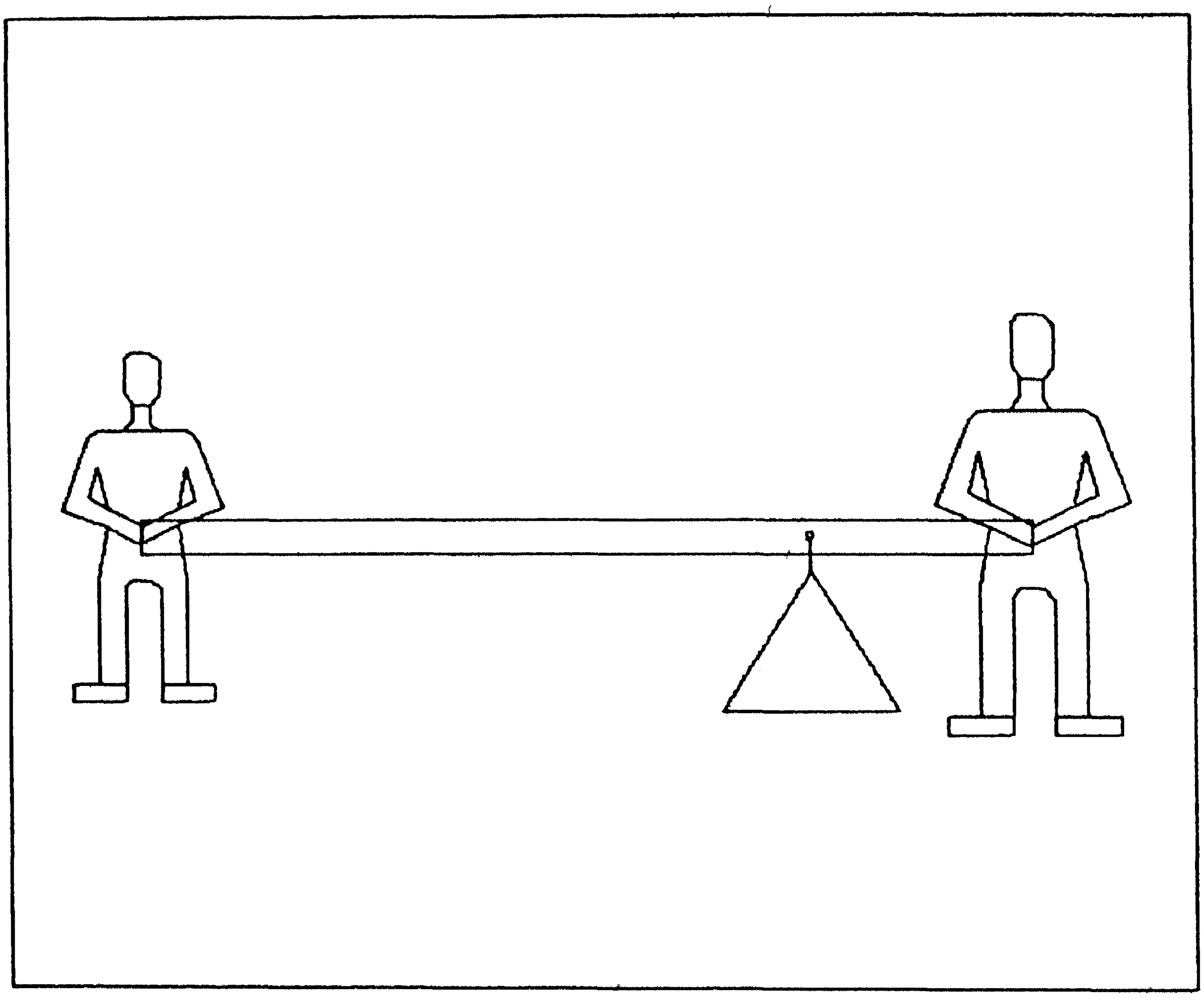
For (P25), a semantic program (or a more general program which referenced the object's GEOMODEL) would be needed to correctly define the wheel base of the car as the distance between the two axles. The car's GEOMODEL would have to be expanded to include wheels and axles. It would be desirable to be able to handle an object such as a car either as a single point, as in (P18), or as a lever system in its own right, as in (P25).



P1

(A LEVER 10 FT LONG IS PINNED AT ITS LEFT END
)(THE LEVER IS SUPPORTED BY A SPRING WITH A
CONSTANT OF 40 LB/FT)(THE SPRING IS ATTACHED
6 FT FROM THE LEFT END OF THE LEVER)(A WEIGHT
OF 20 LB IS ATTACHED AT THE OTHER END OF THE
LEVER)(THE WEIGHT OF THE LEVER IS 8 LB)(HOW
MUCH IS THE SPRING STRETCHED)

ANSWER: 1.00000 FT

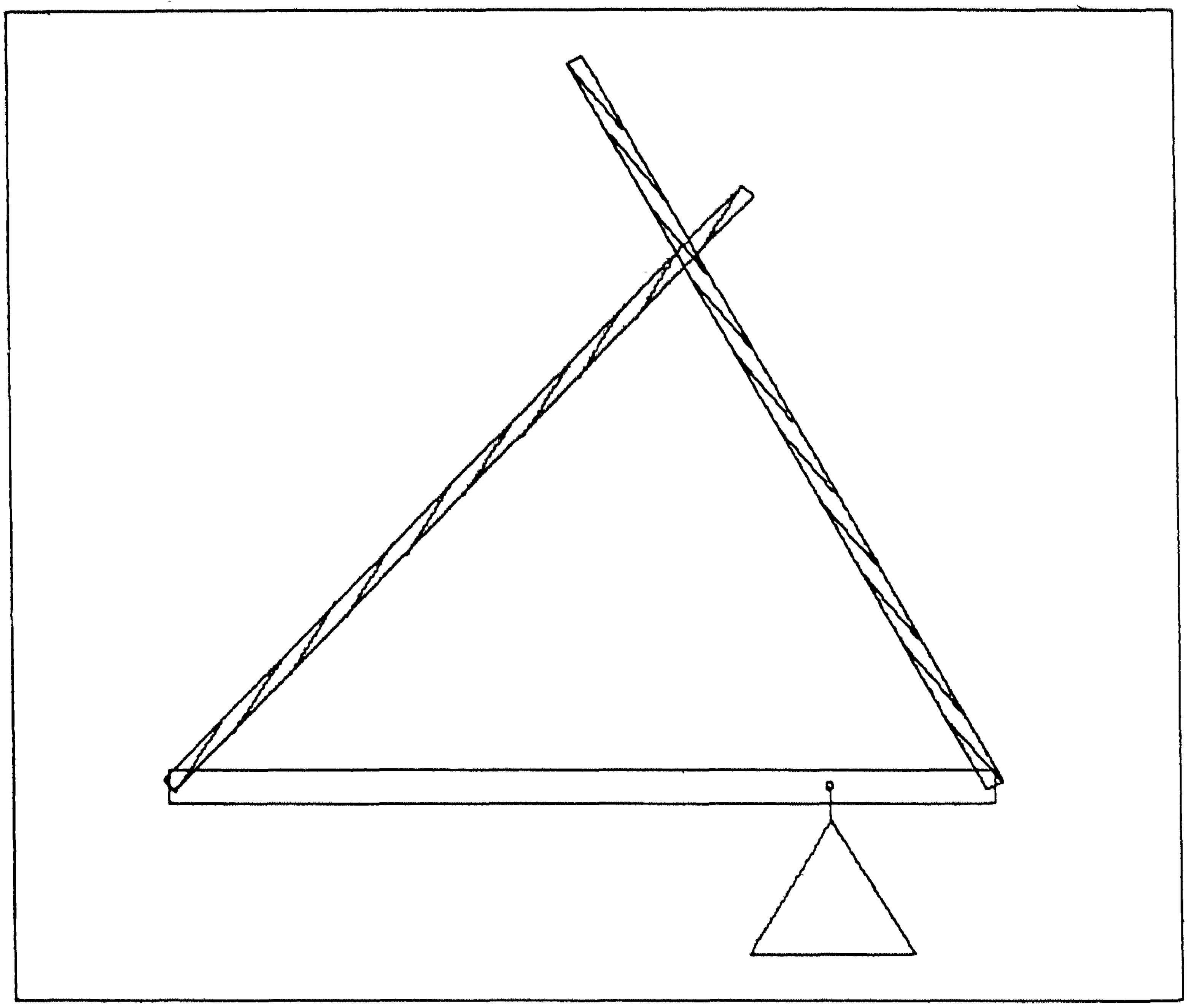


```
P2  SCHAUM* PAGE 12  NUMBER 4

(WHERE MUST A WEIGHT BE HUNG ON A POLE . OF
NEGLIGIBLE WEIGHT   SO THAT THE BOY AT ONE
END SUPPORTS 1/3 AS MUCH AS THE MAN AT THE
OTHER END)

ANSWER:  (TIMES LENGTH76 7.50000E-1) FROM THE
 BOY . WHERE LENGTH76 IS THE LENGTH OF THE
POLE
```

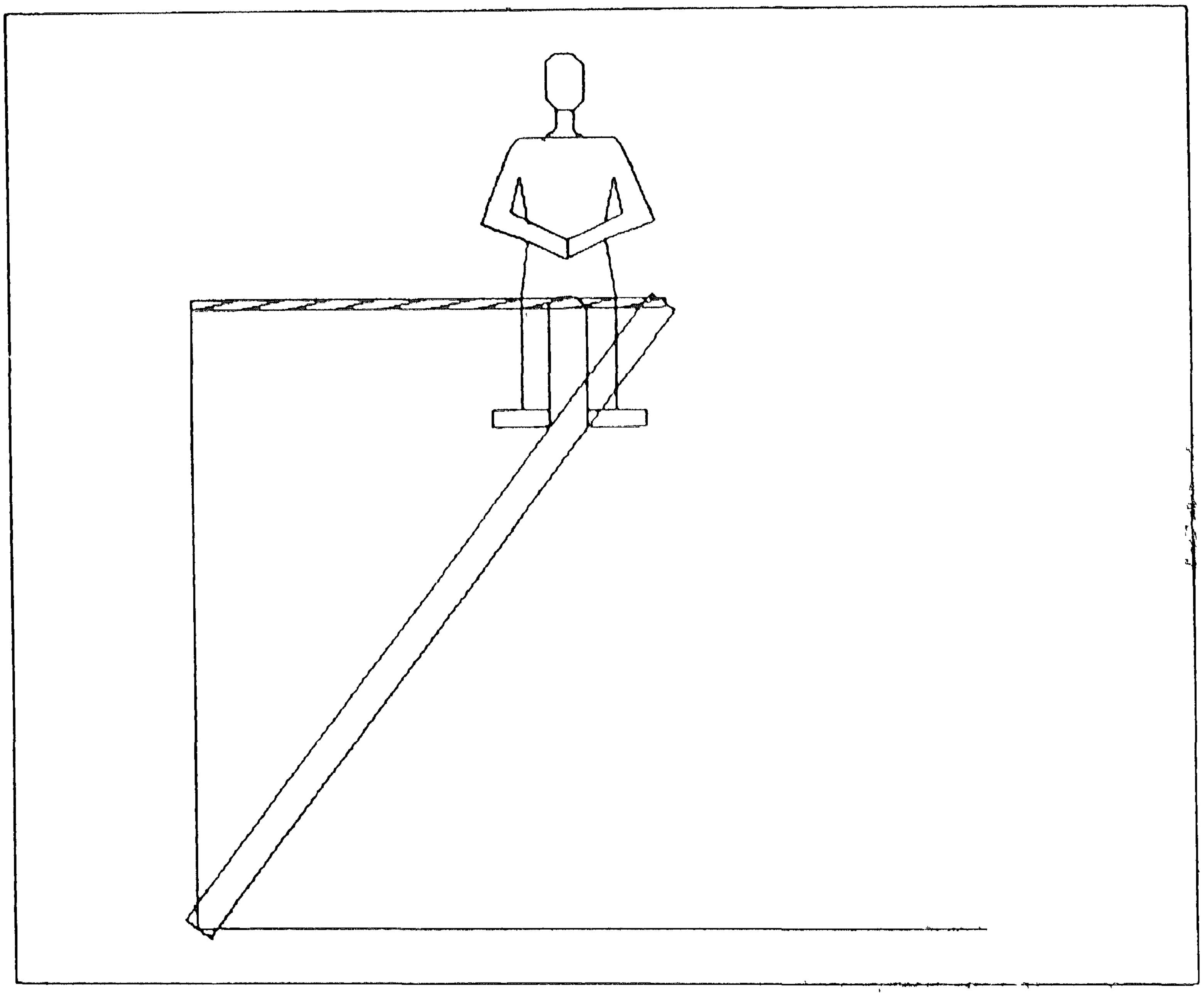
*Problems marked SCHAUM are from *College Physics* [Schaum 61]. Copyright 1961 by McGraw-Hill Book Company.
Used by permission.



P4

(A HORIZONTAL UNIFORM BAR 10 M LONG IS SUPPORTED BY TWO ROPES ATTACHED AT ITS ENDS)(THE ROPE ON THE LEFT END MAKES AN ANGLE OF 45 DEGREES WITH THE HORIZONTAL . WHILE THE ROPE ON THE RIGHT END MAKES AN ANGLE OF 60 DEGREES WITH THE HORIZONTAL)(A WEIGHT OF 100 NT IS ATTACHED 2 M FROM THE RIGHT END OF THE BAR)(WHAT IS THE WEIGHT OF THE BAR)

ANSWER: 123.92305 NT



P8 SCHAUM PAGE 25 NUMBER 19

(THE FOOT OF A LADDER RESTS AGAINST A VERTICAL WALL AND ON A HORIZONTAL FLOOR)(THE TOP OF THE LADDER IS SUPPORTED FROM THE WALL BY A HORIZONTAL ROPE 30 FT LONG)(THE LADDER IS 50 FT LONG . WEIGHS 100 LB WITH ITS CENTER OF GRAVITY 20 FT FROM THE FOOT . AND A 150 LB MAN IS 10 FT FROM THE TOP)(DETERMINE THE TENSION IN THE ROPE)

ANSWER: 120.00000 LB

Appendix B: Object Frame Representations

This appendix briefly describes the representations of objects and their relationships which are constructed by the various parts of the program. The various items of information associated with each object are stored in its property list under named indicators; in describing each type of information, we give the name of the indicator under which it is stored, followed by a description of the information itself.

B.1 Physical Entity Representation

The property list indicators for physical entities (which include explicit forces as well as physical objects) and the type of information stored under each one are described below.

ENTITY:	PHYSENT	Identifies the object as a physical entity.
TOK:	word	Identifies the token-word of which this object is an example. Usually the word representing the object in the sentence.
WORD:	word	Identifies the specific word describing the object in a sentence, if different from the TOK. For example, "boy" would be represented by TOK: PERSON and WORD: BOY.
NAME:	name	Name of the object if it has one.
RESTRICT:	((attribute value) . . .)	Restrictions on the TOK for this object. For example, "boy" would have (RESTRICT (SEX MALE) (AGE YOUNG)).
SELECT:	(selection)	Selection used to select a particular object. For example, "the upper hinge" would have SELECT: (UPPER).
PARTOF:	object	Object which this object is a part of.
PARTS:	(object . . .)	List of all objects which are part of this object.
COFG:	(location)	Location object which is the location of the center of gravity of this object (if specified).
LOCS:	(location . . .)	List of all locations on this object.
ATTACH:	(attachment . . .)	List of all attachment relations which involve this object.

SUPPORT:	(object . . .)	List of objects which this object supports.
SUPPORTBY:	(object . . .)	List of objects which support this object.
UNKNOWNS:	(variable . . .)	List of all variables associated with this object.
CONSTANTS:	(constant . . .)	List of all constants associated with this object.
ROTN:	(angle)	Rotation of the object (counterclockwise in degrees) from its GEOMODEL orientation.
FRAME:	frame	Name of the canonical object frame which represents the object in its current instantiation, e.g., LEVER.
GSIZE:	(S_x S_y)	Geometric size scaling vector.
GSTART:	(x y)	Geometric starting point.
PIVOT:	(x y)	Preferred pivot location for a rigid body object.
FORCES:	(((x y) (f_x f_y)) . . .)	Position and force vector for each force exerted by the object.
SIZEDET:	(measurement . value)	Measured quantity and value for this object, used to determine picture size scaling.
PSCALE:	scale	Scale factor between geometric·length and picture size.
PSIZE:	(S_x S_y)	Picture size scaling vector.
STVAL:	(x y)	Starting point for object in drawing.
IMLACITEM:	*T*	True if the object has been drawn. Not really used in current system.
LENGTH: WEIGHT: TENSION: CONSTANT:	(value units) (value units) (value units) (value units)	Measurements of various attributes, as appropriate for a particular type of object.

B.2 Location Representation

ENTITY:	LOCATION	Identifies the object as a location entity.
FRAME:	LOCATION	Identifies a location frame.
OBJECT:	object	Identifies the object with which the location is associated.
LOCNAME:	word	Location name, e.g., END.
NAME:	name	Name of the location, if specified.

SELECT:	(selection)	Selection used to select this particular location. For example, "the left end" would have SELECT: (LEFT).
REFLOC:	location	Location to which this location is relative.
REFLOCS:	(location . . .)	List of all locations which are relative to this location.
RELPOS:	(type (quantity units))	Specifies position relative to the reference location. For example, "6 ft from one end" would have RELPOS: (FROMLOC (6 FT)).
POSITION:	(x y)	Geometric position of the location.

B.3 Attachment Representation

FRAME:	ATTACH	Identifies this as an attachment frame.
TYPEATT:	type	Type of attachment, e.g., PINJOINT or CONTACT.
LOCS:	((object location (f_x-f_y)) . . .)	Specifies each object involved in the attachment, along with the location on that object and the force vector for the force exerted by the object at that location.
POSITION:	(x y)	Geometric position of the point of attachment.

B.4 Constant or Variable Representation

ENTITY:	CONSTANT or VARIABLE	Identifies this object as a constant or variable.
SYSTEM:	object	Physical entity object with which the constant or variable is associated.
MEASURE:	attribute	Attribute which is measured, e.g., TENSION
UNITS:	units	Units of the measurement, e.g., FT.
VALUE:	value	Numeric or symbolic expression which is the calculated value of a variable.

Appendix C: Generated Structures for a Sample Problem

This appendix contains snapshots of some of the major data structures produced by ISAAC at various stages in the processing of the sample problem (P8). The first part of the listing shows the structures produced by the parsing and semantic processing of each sentence. After each sentence, the result of the parsing is shown; < S> indicates that a major clause was parsed, and the list of tokens which follows gives the root of the parse tree (the verb) for each of the major clauses which was parsed. This is followed by a listing of each of the tokens produced during parsing. The dump follows all of the semantic processing; the tokens are not used thereafter except in finding pronoun referents.

Following the dump of the parse structures is a listing of the internal model of the problem as it exists just after all of the input sentences have been processed, but before frame creation and all the subsequent processing. Each of the GENSYM atoms is a separate object in the model. The four atoms UNKNOWNS, DESUNKS, SYSREPLY, and SYSUNITS are global variables whose bindings are the list of all unknowns, the list of desired unknowns, the reply to be generated once the problem is solved, and the measurement units used for various types of measurements, respectively.

The next part of the dump shows each of the equations presented to SOLVEQ for solution, followed by the answer generated by PRTVAR. The equation (EQUALS 0 0) is caused by a deficiency in CFSURFACE (complete frame for SURFACE) which went unnoticed because it didn't cause any problems. CFSURFACE requires that the force exerted by a surface be perpendicular to the surface; this is fine for an attachment of type CONTACT, but not for one of type PINJOINT, such as the attachment between the rope and the wall. Thus, one of the zeros should be a variable representing the vertical force exerted by the wall. The other zero, representing the vertical force exerted by the rope on the wall, is correct.

The final part of the dump shows the model of the problem at the end of the problem solving and picture generation processes. It can be seen that a great deal of information has been added to the model beyond that which was available immediately after the problem statement was read. The meaning of the information associated with each of the objects in the model is explained in Appendix B.

Sentence Structures After Parsing and Semantics

```
(THE FOOT OF A LADDER RESTS AGAINST A VERTICAL WALL AND ON A
     HORIZONTAL FLOOR)

((<S> (TOK292)))

TOK289  ((TOK . FOOT) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . LOCPART) (SEMOBJ LADDER291) (RFNT LOC297))

TOK290  ((TOK . LADDER) (LFRAME . NP) (DET . INDEF) (NBR NS)
     (SFRAME . PHYSENT) (RFNT LADDER291))

TOK292  ((TOK . REST) (LFRAME . VP) (MAINVB . RESTS)
     (INTRANS . *T*) (ACT . *T*) (SUBJ . TOK289) (MODS
     (CASEARG AGAINST (WALL294))(CASEARG ON (FLOOR296))))

TOK293  ((TOK . WALL) (LFRAME . NP) (DET . INDEF) (NBR NS)
     (MODS (ROTN 90)) (SFRAME . PHYSENT) (RFNT WALL294))

TOK295  ((TOK . FLOOR) (LFRAME . NP) (DET . INDEF) (NBR NS)
     (MODS (ROTN 0)) (SFRAME . PHYSENT) (RFNT FLOOR296))


(THE TOP OF THE LADDER IS SUPPORTED FROM THE WALL BY A
     HORIZONTAL ROPE 30 FT LONG)

((<S> (TOK302)))

TOK300  ((TOK . TOP) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . LOCPART) (SEMOBJ LADDER291) (RFNT LOC309))

TOK301  ((TOK . LADDER) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . PHYSENT) (RFNT LADDER291))

TOK302  ((TOK . SUPPORT) (LFRAME . VP) (MAINVB . SUPPORTED)
     (AUX IS) (TRANS . *T*) (PASV . *T*) (OBJ . TOK300)
     (MODS (CASEARG FROM (WALL294))) (SUBJ . TOK304))

TOK303  ((TOK . WALL) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . PHYSENT) (RFNT WALL294))

TOK304  ((TOK . ROPE) (LFRAME . NP) (DET . INDEF) (NBR NS) (MODS
     (ROTN 0) (LENGTH 30 FT)) (SFRAME . PHYSENT) (RFNT ROPE305))


(THE LADDER IS 50 FT LONG , WEIGHS 100 LB WITH ITS CENTER OF
     GRAVITY 20 FT FROM THE FOOT   AND A 150 LB MAN IS 10 FT
     FROM THE TOP)

((<S> (TOK312 TOK320)))

TOK311  ((TOK . LADDER) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . PHYSENT) (RFNT LADDER291))
```

```
TOK312  ((TOK . BE) (LFRAME . VP) (MAINVB . IS) (INTRANS .
     *T*) (ACT . *T*) (SUBJ . TOK311) (COMP LENGTH 50 FT)
     (VPCONJ TOK313))

TOK313  ((TOK . WEIGH) (LFRAME . VP) (MAINVB . WEIGHS)
     (INTRANS . *T*) (ACT . *T*) (SUBJ . TOK311) (COMP . TOK314))

TOK314  ((TOK . LB) (LFRAME . NP) (QTY . 100))

TOK315  ((TOK . COFG) (LFRAME . NP) (NBR NS) (SFRAME .
     LOCPART) (SEMOBJ LADDER291) (MODS (LOC AT (LOC317))))

TOK316  ((TOK . FOOT) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . LOCPART) (SEMOBJ LADDER291) (RFNT LOC297))

TOK318  ((TOK . PERSON) (LFRAME . NP) (WORD . MAN) (MODS
     (RESTRICT (SEX MALE)) (RESTRICT (AGE ADULT)) (WEIGHT
     150 LB)) (DET . INDEF) (NBR NS) (SFRAME . PHYSENT)
     (RFNT PERSON319))

TOK320  ((TOK . BE) (LFRAME . VP) (MAINVB . IS) (INTRANS .
     *T*) (ACT . *T*) (SUBJ . TOK318) (MODS (LOC AT LOC322))))

TOK321  ((TOK . TOP) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . LOCPART) (SEMOBJ LADDER291) (RFNT LOC309))


(DETERMINE THE TENSION IN THE ROPE)

((<S> (TOK324)))

TOK324  ((TOK . DETERMINE) (LFRAME . VP) (MAINVB . DETERMINE)
     (TRANS . *T*) (IMPERATIVE . *T*) (ACT . *T*) (OBJ . TOK325))

TOK325  ((TOK . TENSION) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . ATTROF) (SEMOBJ ROPE305))

TOK326  ((TOK . ROPE) (LFRAME . NP) (DET . DEF) (NBR NS)
     (SFRAME . PHYSENT) (RFNT ROPE305))
```

Initial Model After Reading Problem Statement

```
LADDER291  ((TOK . LADDER) (ENTITY . PHYSENT) (LOCS LOC297
     LOC309 LOC317 LOC322) (ATTACH ATTACH298 ATTACH299
     ATTACH310 ATTACH323) (SUPPORTBY FLOOR296 WALL294
     ROPE305) (COFG LOC317) (LENGTH 50 FT) (WEIGHT 100 LB))

WALL294  ((TOK . WALL) (ENTITY . PHYSENT) (ROTN 90) (ATTACH
     ATTACH299 ATTACH308) (SUPPORT LADDER291))

FLOOR296  ((TOK . FLOOR) (ENTITY . PHYSENT) (ROTN 0) (ATTACH
     ATTACH298) (SUPPORT LADDER291))
```

```
LOC297  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT
     LADDER291) (LOCNAME . FOOT) (REFLOCS LOC317))

ATTACH298  ((FRAME . ATTACH) (TYPEATT . CONTACT) (LOCS
     (LADDER291 LOC297) (FLOOR296 NIL)))

ATTACH299  ((FRAME . ATTACH) (TYPEATT . CONTACT) (LOCS
     (LADDER291 LOC297) (WALL294 NIL)))

ROPE305  ((TOK . ROPE) (ENTITY . PHYSENT) (ROTN 0) (LENGTH
     30 FT) (LOCS LOC306 LOC307) (ATTACH ATTACH308
     ATTACH310) (SUPPORT LADDER291) (UNKNOWNS TENSION327)
     (TENSION TENSION327 LB))

LOC306  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     ROPE305) (LOCNAME . END))

LOC307  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     ROPE305) (LOCNAME . END))

ATTACH308  ((FRAME . ATTACH) (TYPEATT . PINJOINT) (LOCS
     (WALL294 NIL) (ROPE305 LOC307)))

LOC309  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . TOP) (REFLOCS LOC322))

ATTACH310  ((FRAME . ATTACH) (TYPEATT . PINJOINT) (LOCS
     (ROPE305 LOC306) (LADDER291 LOC309)))

LOC317  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . FOOT) (REFLOC . LOC297) (RELPOS
     FROMLOC (20 FT)))

PERSON319  ((TOK . PERSON) (WORD . MAN) (ENTITY . PHYSENT)
     (RESTRICT (SEX MALE) (AGE ADULT)) (WEIGHT 150 LB)
     (ATTACH ATTACH323))

LOC322  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . TOP) (REFLOC . LOC309) (RELPOS
     FROMLOC (10 FT)))

ATTACH323  ((FRAME . ATTACH) (TYPEATT . CONTACT) (LOCS
     (PERSON319 NIL) (LADDER291 LOC322)))

TENSION327  ((ENTITY . VARIABLE) (SYSTEM . ROPE305) (MEASURE
     . TENSION) (UNITS . LB))

UNKNOWNS (TENSION327)

DESUNKS (TENSION327)

SYSREPLY ((PRTVAR TENSION327))

SYSUNITS ((FORCE . LB) (LENGTH . FT))
```

Equations Generated By Problem Solver

```
SOLVEQ  (EQUALS 0 FORCE328)

SOLVEQ  (EQUALS 0 (PLUS FORCE329 FORCE338))

SOLVEQ  (EQUALS 0 (PLUS FORCE330 (TIMES FORCE336 -1)))

SOLVEQ  (EQUALS 0 FORCE331)

SOLVEQ  (EQUALS 0 (PLUS (TIMES FORCE337 -1) TENSION327))

SOLVEQ  (EQUALS 0 0)

SOLVEQ  (EQUALS 0 (PLUS (TIMES TENSION327 -1.00000) FORCE332))

SOLVEQ  (EQUALS 0 FORCE333)

SOLVEQ  (EQUALS 0 FORCE334)

SOLVEQ  (EQUALS 0 (PLUS -150 FORCE335))

SOLVEQ  (EQUALS 0 (PLUS 250 (MINUS FORCE338)))

SOLVEQ  (EQUALS 0 (PLUS FORCE332 FORCE336))

SOLVEQ  (EQUALS 0 (PLUS (PLUS (PLUS (DIFFERENCE (TIMES
     FORCE332 8.00000) (TIMES FORCE333 6.00000)) (DIFFERENCE
     (TIMES FORCE330-32.00000) (TIMES FORCE331 -24.00000)))
     (DIFFERENCE (TIMES FORCE328 -32.00000) (TIMES FORCE329
     -24.00000))) 1200.00000))

ANSWER:  120.00000 LB
```

Final Model of the Problem

```
LADDER291  ((TOK . LADDER) (ENTITY . PHYSENT) (LOCS LOC297
     LOC309 LOC317 LOC322) (ATTACH ATTACH298 ATTACH299
     ATTACH310 ATTACH323) (SUPPORTBY FLOOR296 WALL294
     ROPE305) (COFG LOC317) (LENGTH 50 FT) (WEIGHT 100 LB)
     (GSIZE 1.00000 0) (FRAME . LEVER)(SUPPORT PERSON319)
     (GSTART 0 0) (ROTN 53.13010) (FORCES ((24.00000
     32.00000) (FORCE334 FORCE335)) ((30.00000 40.00000)
     (FORCE332 FORCE333)) ((0 0) (FORCE330 FORCE331)) ((0 0)
     (FORCE328 FORCE329)) ((12.00000 16.00000) (0 100)))
     (UNKNOWNS FORCE328 FORCE329  FORCE330 FORCE331 FORCE332
     FORCE333 FORCE334 FORCE335) (SIZEDET LENGTH . 50)
     (PSIZE 1.00000 1.00000) (PSCALE . 1.00000) (STVAL 0 0)
     (IMLACITEM . *T*))

WALL294  ((TOK . WALL) (ENTITY . PHYSENT) (ROTN 90) (ATTACH
     ATTACH299 ATTACH308) (SUPPORT LADDER291) (GSIZE 0 0)
     (FRAME . SURFACE)(GSTART 0 0) (UNKNOWNS FORCE336
     FORCE337) (FORCES ((6.13909E-12 40.00000) ((TIMES
     FORCE337 -1) 0)) ((0 0) ((TIMES FORCE336 -1) 0)))
     (SIZEDET LENGTH . 40.00000) (PSIZE 8.00000E-1
     8.00000E-1) (PSCALE . 1.25000) (STVAL -8.00000E-1
     6.00000E-1) (IMLACITEM . *T*))

FLOOR296  ((TOK . FLOOR) (ENTITY . PHYSENT) (ROTN 0) (ATTACH
     ATTACH298) (SUPPORT LADDER291) (GSIZE 0 0) (FRAME .
     SURFACE) (GSTART 0 0) (UNKNOWNS FORCE338) (FORCES ((0
     0) (0 FORCE338))) (PSIZE 1 1) (STVAL -8.00000E-1
     6.00000E-1) (IMLACITEM . *T*))

LOC297  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . FOOT) (REFLOCS LOC317) (POSITION 0 0))

ATTACH298  ((FRAME . ATTACH) (TYPEATT . CONTACT) (LOCS
     (LADDER291 LOC297 (FORCE328 FORCE329)) (FLOOR296 NIL (0
     FORCE338))) (POSITION 0 0))

ATTACH299  ((FRAME . ATTACH) (TYPEATT . CONTACT) (LOCS
     (LADDER291 LOC297 (FORCE330 FORCE331)) (WALL294 NIL
     ((TIMES FORCE336 -1) 0))) (POSITION 0 0))

ROPE305  ((TOK . ROPE) (ENTITY . PHYSENT) (ROTN 0) (LENGTH
     30 FT) (LOCS LOC306 LOC307) (ATTACH ATTACH308
     ATTACH310) (SUPPORT LADDER291) (UNKNOWNS TENSION327)
     (TENSION TENSION327 LB) (GSIZE 6.00000E-1 0) (FRAME .
     ROPE) (GSTART 6.13909E-12 40.00000) (FORCES((30.00000
     40.00000) ((TIMES TENSION327 -1.00000) 0))
     ((6.13909E-12 40.00000) (TENSION327 0))) (SIZEDET
     LENGTH . 30) (PSIZE 6.00000E-1 6.00000E-1) (PSCALE .
     1.66667) (STVAL -8.00000E-1 40.30000) (IMLACITEM . *T*))

LOC306  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     ROPE305) (LOCNAME . RIGHTEND) (POSITION 30.00000 0))

LOC307  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     ROPE305) (LOCNAME . LEFTEND) (POSITION 6.13909E-12
     40.00000))

ATTACH308  ((FRAME . ATTACH) (TYPEATT . PINJOINT) (LOCS
     (WALL294 NIL ((TIMES FORCE337 -1) 0)) (ROPE305 LOC307
     (TENSION327 0))) (POSITION 6.13909E-12 40.00000))

LOC309  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . TOP) (REFLOCS LOC322) (POSITION
     30.00000 40.00000))

ATTACH310  ((FRAME . ATTACH) (TYPEATT . PINJOINT) (LOCS
     (ROPE305 LOC306 ((TIMES TENSION327 -1.00000) 0))
     (LADDER291 LOC309 (FORCE332 FORCE333))) (POSITION
     30.00000 40.00000))

LOC317  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . FOOT) (REFLOC . LOC297) (RELPOS
     FROMLOC (20 FT)) (POSITION 12.00000 16.00000))

PERSON319  ((TOK . PERSON) (WORD . MAN) (ENTITY . PHYSENT)
     (RESTRICT (SEX MALE) (AGE ADULT)) (WEIGHT 150 LB)
     (ATTACH ATTACH323) (GSIZE 0 0) (SUPPORTBY LADDER291)
     (FRAME . WEIGHT) (GSTART 24.00000 32.00000) (ROTN 0)
     (FORCES ((24.00000 32.00000) (0 -150))) (SIZEDET FORCE
     . 150) (PSIZE 6.00000E-1 6.00000E-1) (STVAL 17.80000
     32.60000) (IMLACITEM . *T*))

LOC322  ((FRAME . LOCATION) (ENTITY . LOCATION) (OBJECT .
     LADDER291) (LOCNAME . TOP) (REFLOC . LOC309) (RELPOS
     FROMLOC (10 FT)) (POSITION 24.00000 32.00000))

ATTACH323  ((FRAME . ATTACH) (TYPEATT . CONTACT) (LOCS
     (PERSON319 NIL (0 -150)) (LADDER291 LOC322 (FORCE334
     FORCE335))) (POSITION 24.00000 32.00000))

TENSION327  ((ENTITY . VARIABLE) (SYSTEM . ROPE305) (MEASURE
     . TENSION) (UNITS . LB) (VALUE . 120.00000))

FORCE328  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 0))

FORCE329  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . -250))

FORCE330  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . -120.00000))

FORCE331  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 0))

FORCE332  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 120.00000))

FORCE333  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 0))

FORCE334  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 0))

FORCE335  ((ENTITY . VARIABLE) (SYSTEM . LADDER291) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 150))

FORCE336  ((ENTITY . VARIABLE) (SYSTEM . WALL294) (MEASURE .
     FORCE) (UNITS . LB) (VALUE . -120.00000))

FORCE337  ((ENTITY . VARIABLE) (SYSTEM . WALL294) (MEASURE .
     FORCE) (UNITS . LB) (VALUE . 120.00000))

FORCE338  ((ENTITY . VARIABLE) (SYSTEM . FLOOR296) (MEASURE
     . FORCE) (UNITS . LB) (VALUE . 250))
```

Bibliography

Becker, Joseph D. "The Phrasal Lexicon", pp. 60-63 in *Theoretical Issues in Natural Language Processing*, proceedings of a conference held at M.I.T. on June 10-13, 1975.

Bobrow, Daniel G. "Natural Language Input for a Computer Problem-Solving System", in [Minsky 68].

Bobrow, Daniel and Collins, Allan (Ed.). *Representation and Understanding*. New York: Academic Press, 1975.

Brown, G. Spencer. *Laws of Form*. New York: Bantam Books, 1972.

Brown, John Seely and Burton, Richard R. "Multiple Representations of Knowledge for Tutorial Reasoning" In Bobrow and Collins.

Charniak, Eugene. "Computer Solution of Calculus Word Problems" *Proc. 1st Intl Joint Conf. on Artificial Intelligence*, pp. 303-316. Boston: Mitre Corporation, 1969.

____________. "CARPS, a Program which Solves Calculus Word Problems." M.I.T. report MAC-TR-51, available from NTIS as AD 673 670, July 1968.

____________. "Toward a Model of Children's Story Comprehension", M.I.T. A.I. Lab Report AI TR-266, Dec. 1972.

de Kleer, Johan. "Qualitative and Quantitative Knowledge in Classical Mechanics" M.I.T. A.I. Lab Report AI-TR-352, December 1975.

Dull, C.E., Metcalfe, H.C., and Williams, J.E. *Modern Physics*. New York: Holt, Rinehart and Winston, 1964.

Gelb, Jack P. "Experiments with a Natural Language Problem-Solving System", *Proc. 2nd Intl. Joint Conf. on Artificial Intelligence*, pp. 455-462. London: The British Computer Society, 1971.

Halliday, David and Resnick, Robert. *Physics* (Parts I and II). New York: John Wiley & Sons, 1967.

Heidorn, George E. "Natural Language Inputs to a Simulation Programming System." NPS-55HD72101A, Naval Postgraduate School, Monterrey, California, 1972.

McCarthy, John, et al. *LISP 1.5 Programmer's Manual*. Cambridge, Mass.: M.I.T. Press, 1965.

McDermott, D.V. and Sussman, G.J. "The CONNIVER Reference Manual", M.I.T A.I. Lab Memo No. 259, 1972.

Minsky, Marvin. *Semantic Information Processing*. Cambridge, Mass.: M.I.T. Press, 1968.

____________. "A Framework for Representing Knowledge", M.I.T. A.I. Memo No. 306, 1974. Also appears in Winston (ed.) 1975.

Moses, Joel. "MACSYMA-The Fifth Year." ACM *SIGSAM Bulletin*, Vol. 8, No. 3 (Aug. 1974), pp. 105-110.

Papert, Seymour. "Teaching Children to be Mathematicians vs. Teaching About Mathematics." *Int. J. Math. Educ. in Science & Tech.*, New York: Wiley & Sons, 1972; M.I.T. A.I. Memo No. 249, July 1971.

Schank, Roger C. "Identification of Conceptualizations Underlying Natural Language", in Schank and Colby (eds), 1973.

____________. *Conceptual Information Processing.* New York: American Elsevier, 1975.

Schank, Roger C. and Colby, K.M. (eds). *Computer Models of Thought and Language.* San Francisco: W.H. Freeman and Co., 1973.

Schaum, Daniel. *Theory and Problems of College Physics,* 6th ed. New York: McGraw-Hill, 1961.

Simmons, R.F. "Natural Language Question Answering Systems: 1969," *Comm. ACM,* vol. 13, no. 1 (January, 1970), pp. 15-30.

____________. "Semantic Networks: Their Computation and Use for Understanding English Sentences", in Schank and Colby (eds.), 1973.

Simmons, R.F. and Amsler, R.A. "Modeling Dictionary Data" Technical Report NL-25, Computer Science Dept., The University of Texas at Austin, 1975.

Simmons, R.F. and Bennett-Novak, G. "Semantically Analyzing an English Subset for the Clowns Microworld", *American Journal of Computational Linguistics,* Microfiche 18, 1975.

Simmons, R.F. and Bruce, B. "Some Relations between Predicate Calculus and Semantic Net Representations of Discourse." *Proc. 2nd International Joint Conf. on Artificial Intelligence,* London: British Computer Society, 1971.

Simmons, R.F. and Slocum, J. "Generating English Discourse from Semantic Networks," *Comm. ACM,* vol. 15, no. 10 (October 1972), pp. 891-905.

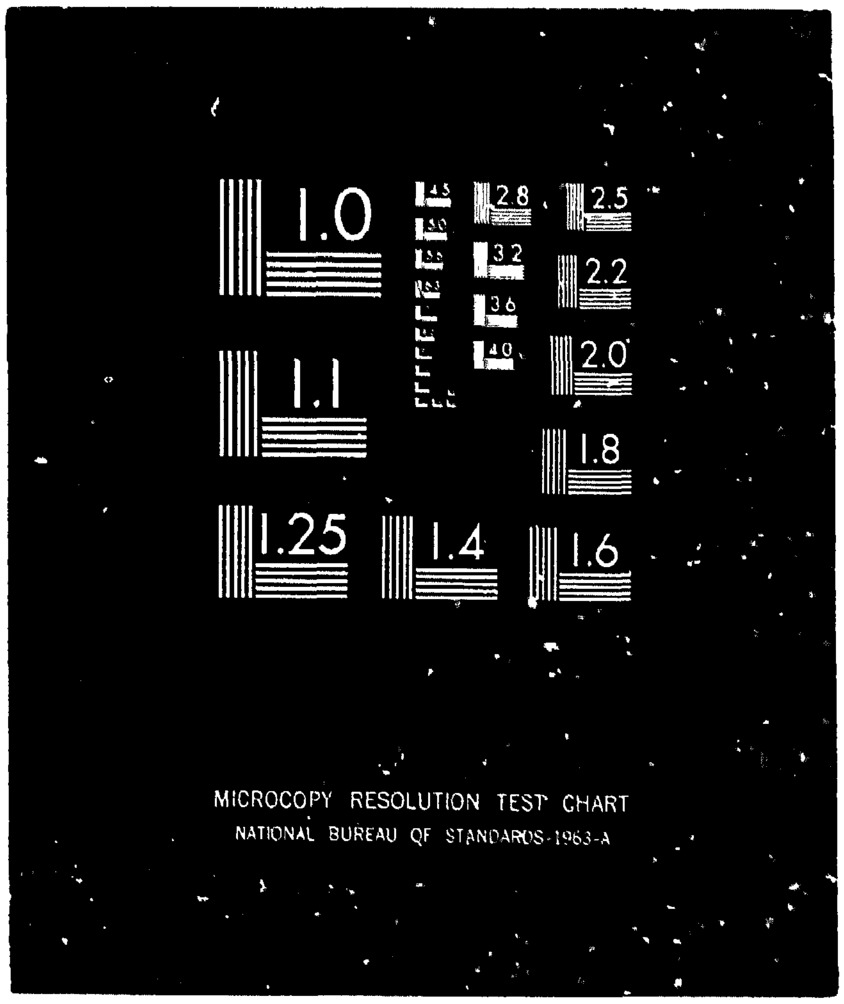
Wilks, Yorick. "An Intelligent Analyzer and Understander of English" *Communications of the ACM,* vol. 18, no. 5 (May 1975), pp. 264-274.

Winograd, Terry. *Understanding Natural Language.* New York: Academic Press, 1972.

Winston, Patrick H. *The Psychology of Computer Vision.* New York: McGraw-Hill, 1975.

Woods, William A. "Transition Network Grammars for Natural Language Analysis" *Communications of the ACM,* vol. 13, no. 10 (October 1970), pp. 591-606.

END


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A