American Journal of Computational Linguistics *Microfiche 18*

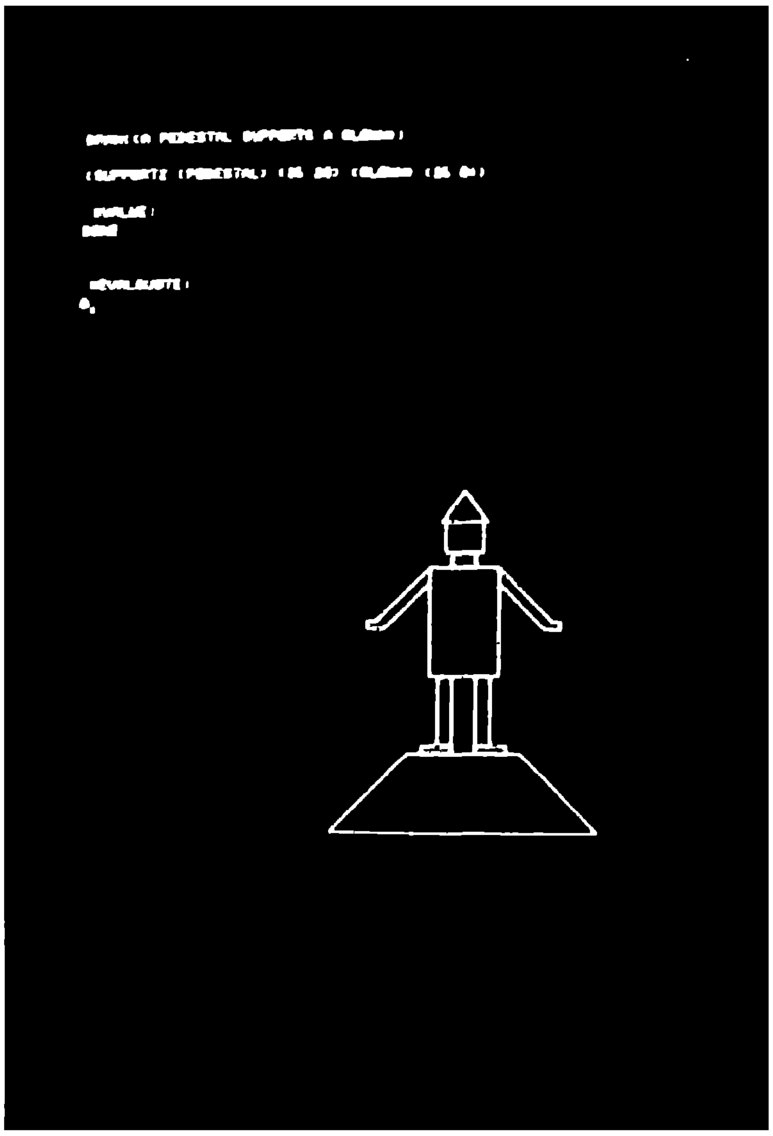
SEMANTICALLY ANALYZING AN ENGLISH SUBSET FOR THE CLOWNS MICROWORLD

Robert F. Simmons

Gordon Bennett-Novak

Department of Computer Sciences

The University of Texas at Austin



ABSTRACT

A microworld system is described for displaying visual representations of the meaning of a subset of English that concerns a clown that can balance objects and can participate in motion scenarios. Nouns such as "clown", "lighthouse", "water" etc. are programs that construct images on a display screen. Other nouns such as "top", "edge", "side", etc. are defined as functions that return contact points for the pictures. Adjectives and adverbs provide data on size and angles of support. Prepositions and verbs are defined as semantic functions that explicate spatial relations among noun images. Generally, a verb produces a process model that encodes a series of scenes that represent initial, intermediate and final displays of the changes the verb describes.

The system is programmed in UTLISP for CDC equipment and uses an IMLAC display system. It currently occupies $32_{10}K$ words of core and requires less than a second to translate a sentence into a picture. Applications to teaching linguistics and languages are suggested.

CONTENTS

FIGURES

ACKNOWLEDGMENTS

This research was supported in part by NSF Grant GJ509E. I am indebted to Bill Henneman, Jonathan Slocum, Michael K. Smith, Ken Speaker and Bob Amsler for productive discussions and help in designing and debugging the programs described here My thanks to Professor Woodrow Bledsoe for making available the IMLAC display and its operating systems.

NATURAL LANGUAGE RESEARCH FOR CAI

Sponsored by

THE NATIONAL SCIENCE FOUNDATION

Grant GJ 509E

Privately circulated as:

TECHNICAL REPORT NL-24

Department of Computer Sciences

THE UNIVERSITY OF TEXAS AT AUSTIN

April 1975

SEMANTICALLY ANALYZING AN ENGLISH SUBSET FOR THE CLOWNS MICROWORLD

I Introduction

Several examples of semantically based grammars have appeared in the literature since 1970. The most complete of these are Winograd's (1972) outline of a systemic grammar for commanding and questioning the robot hand in the MIT blocks world, Heidorn's (1972) rewrite rules for analyzing and generating English descriptions and transforming them into GPSS programs, and the ATN grammar of questions for the Lunar Rocks Data Base presented by Woods, Kaplan and Nash-Webber (1972). Most other grammars of significan size, such as that of the NYU String Analysis Project (Grishman and Sager 1973) and numerous grammars developed for mechanical translation are largely syntactic in orientation and not easily accessible. Riesbeck also presents a semantic grammar in the form of a set of LISP programs to comput conceptual dependencies (1975).

A difficulty with these reports is that the systems using the grammars are typically quite large programs--100K+--and the interactions between the grammar and the rest of the system are frequently quite complicated. The reader who wishes to use them as a basis for constructing a small natural language understanding system may well be at a loss as how to begin. He may have the impression that a natural language processing system is a vast undertaking involving great complexity of programming.

He will not be completely incorrect in these impressions, but in fact programming a grammar and semantic system for a microworld model to understand a small subset of English is no longer a formidable task. The vocabulary can be restricted to one hundred or so words, a minimally sufficient syntactic and semantic system can be expressed in a few dozen rules supported by a dozen or so semantic functions, and the pragmatics of such microworlds as the STRIPS robot, the blocks world, or the CLOWNS world presented here, can be modelled very simply. The simplest microworld models that communicate in English require an effort somewhere between a two week homework exercise and a graduate term project. CLOWNS represents about 6 man-months of effort so far.

But is there any real purpose in studying English communication in these trivial microworld situations? If we model language behavior in one microworld we remain several orders of magnitude short of understanding the general use of the language in text, or in verbal discourse and equally far from the possible goal of instructing computers in English to accomplish a general run of tasks.

I remain incurably optimistic. The generalizations about tiny subsets of language and behavior that emerge from microworld models gradually accumulate in our human minds into what may eventually prove sufficient understanding for the accomplishment of socially useful tasks. The initiation ritual of programming a mini-intelligence is a necessary pre-requisite to programming one that is more sophisticated.

In this paper, CLOWNS, a simple microworld model is presented with an explicit tutorial intent. A brief grammar is described that accounts for much of the embedding logic of English constructions; a system of transformations of English constituents to property list representations of semantic network structures is followed by their representation in a dynamic process model that can be operated to produce successive states described by the English. The principles used in the system are a concise representation of my gleanings from recent literature and of course from work of my own and my students.

II Background

In this section only a few of hundreds of natural language processing papers are suggested as entries to the literature. At least a dozen reviews of this literature are available; Walker's is not only among the most recent and complete (Walker 1973), but it includes a section that cites the reviews.

Since 1970, the language processing literature has been rich in reports of natural language systems that can understand subsets of English with respect to various microworlds. In addition to previously mentioned work by Woods, Heidorn and Winograd, there are less frequently cited but quite interesting theses by Badre (1972) that learns to do very simple number problems from text, by Scragg (1975) that answers questions about food preparation processes and by Bruce (1972) that presents a logic and a system for answering questions about temporal reference. Schank Riesbeck, Goldman and Rieger (1975) have published a significant series of papers on semantic parsing, inference and generation for an English subset concerning fairly ordinary human actions. Hendrix, Slocum and Thompson (1973) describe a system for understanding and generating English about commercial transactions and simple movements. Hendrix (1975) has also developed a set theoretic system of process models for representing natural language meanings. These models are descended from robot problem solving research by Fikes and Nilsson

(1971) and Siklóssy et. al. (1973). Harris (1972) provides a tour de force that uses problem solving, inference and learning methods to teach a robot facts about its microworld. Hobbs (1974) presents an approach to natural language semantics that is shown to apply to several applications, diagrams-to-language, English and Algol-to-Algol, etc.

Much of the most recent work by Abelson (1975), Charniak (1972), Schank and Abelson (1975), Minsky (1975), Winograd (1975), Bobrow and Norman (1975), Collins and Warnock (1975), Rumelhart (1975) has progressed beyond the question of grammar and semantic systems to that of such larger units of semantic organization as Frames, Story grammars, Plans, Schemes, Dremes, etc. Although at this writing most of these formulations still fall short of computational realization, it is clear that the research task of the immediate future is one of formulating and programming structures of organization that will successfully model much more complicated microworlds than those presently achieved. A forthcoming book edited by Collins and Bobrow will present many of these ideas.

LISP is still the language of most frequent choice for these experiments and thanks to the prevalence of virtual memories and virtual LISP, the limitation to in-core implementations has essentially vanished. Many of the programs cited used require from 100 to $300K_{10}$ cells of storage. The system described in subsequent sections resides in 32K on a CDC system, although our most recent additions have caused us to use a virtual memory version of UTLISP that was developed by Mabry Tyson.

III Pictorial Models

Ignoring early work largely lost in the archives of corporate memos, Winograd's language processor is essentially a first reporting of how to map English sentences into diagrammatic pictures. Apart from potential applications, the pictures are of great value in providing a universally understood second language to demonstrate the system's interpretation of the English input. While we are still struggling in early stages of how to compute from English descriptions or instructions, there is much to be gained from studying the subset of English that is picturable. Translation of English into other more general languages such as predicate calculus, LISP, Russian, Basic English, Chinese, etc. can provide the same feedback as to the system's interpretation and must suffice for the unpicturable set of English. But for teaching purposes, computing pictures from language is an excellent instrument.

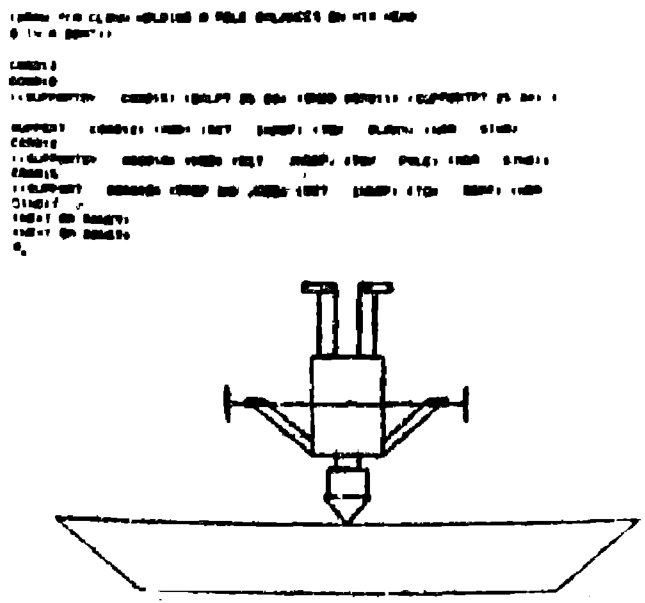
We began with the notion that it should be quite easy to construct a microworld concerning a clown, a pedestal, and a pole. The resulting system could draw pictures for such sentences as:

A clown holding a pole balances on his head in a boat.

A clown on his arm on a pedestal balances a small clown on his head.

Figure 1 shows examples of diagrams produced in response to these sentences.

We progressed then to sentences concerning movement by adding land, water, a lighthouse, a dock and a boat. We were then able to draw pictures such as Figure 2 to represent the meanings of:



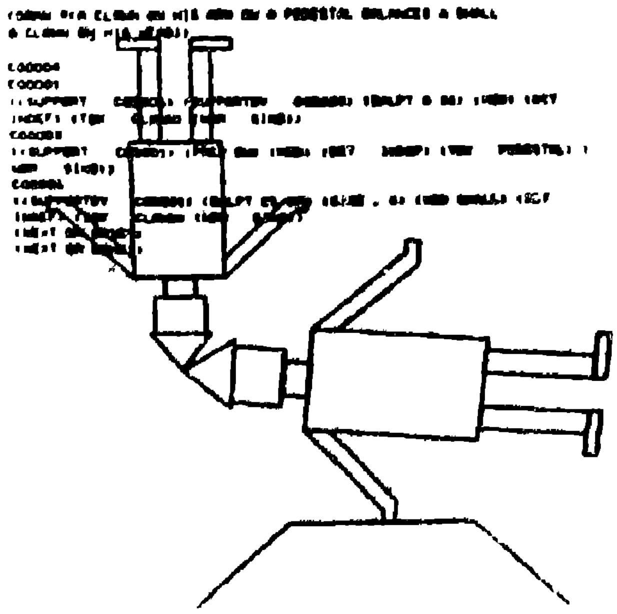


FIGURE 1. STATE VERBS

A clown on his head sails a boat from the dock to the lighthouse.

In the context of graphics, two dimensional line drawings are attractive in their simplicity of computation. An object is defined as a LOGO graphics program that draws it (see Section VI) A scene is a set of objects related in terms of contact points. A scene can be described by a set of predicates

(BOAT ABOVE WATER) (ATTACH $BOAT_{XY}$ $WATER_{XY}$)

(DOCK ABOVE WATER) (DOCK LEFTOF WATER) (BOAT RIGHTOF DOCK)

(ATTACH $DOCK_{XY}$ $WATER_{XY}$) (ATTACH $BOAT_{XY+kY}$ $DOCK_{XY}$)

Orientation functions for adjusting starting points and headings of the programs that draw the objects are required and these imply some trigonometric functions A LISP package of about 650 lines has been developed by Gordon Bennett to provide the picture making capability

What is mainly relevant to the computation of language meanings is that a semantic structure sufficient to transmit data to the drawing package is easily represented as a property list associated with an artificial name for the scene For example, A CLOWN ON A PEDESTAL" results in the following structure

(C1, TOK CLOWN, SUPPORTBY C2, ATTACH(C1 FEETXY C2 TOPXY))

(C2, TOK PEDESTAL, SUPPORT C1, ATTACH(C2 TOPXY C1 FEETXY))

(CLOWN, EXPR(LAMBDA() ,) FEET XY, SIZE 3, STARTPT XY, HEADING A)

(PEDESTAL, EXPR(LAMBDA()) TOP XY, SIZE 3, STARTPT XY, HEADING A)

A larger scene has more objects more attach relations, and may include additional relations such as INSIDE, LEFTOF, RIGHTOF, etc In any case the scene is semantically represented as a set of objects connected by

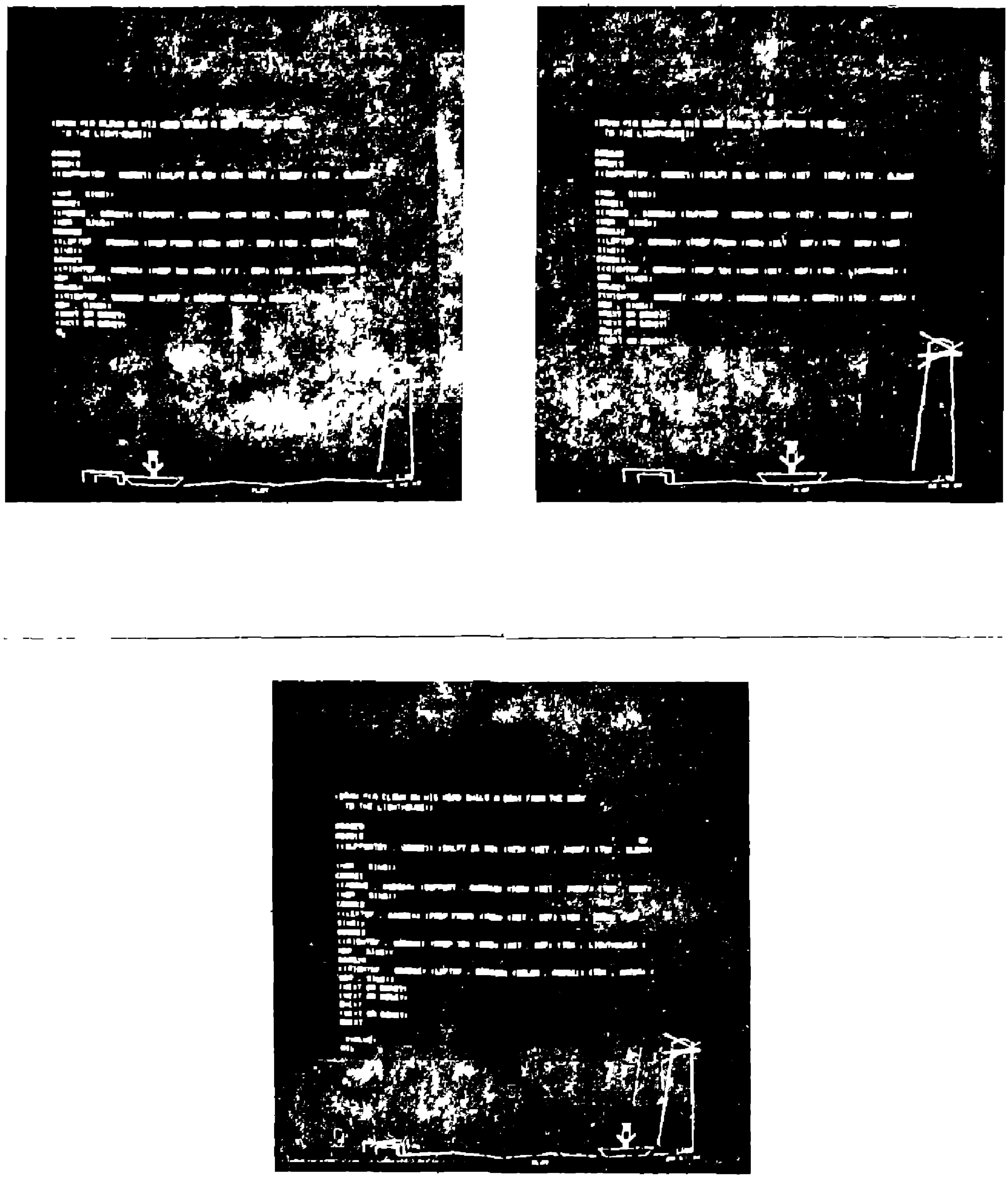


FIGURE 2   A MOTION VERB

relations in a graph (i e a semantic network) that can easily be stored as a property list with references to other objects with property lists

We take "balance" stand' "support "hold ' is on" etc. as state describing verbs in contrast to those such as "sail", 'ride', fly' "buy etc which describe changes of state To model the meaning of state verbs requires only a single diagram to show the state described For change of state verbs a series of pictures' is required and a process model is used to construct a sequence of state descriptions each of which can produce a diagram

IV An English Subset Grammar

We take the Woods ATN as a basic formalism for describing a grammar computationally This system has been well-described by Woods (1970), its application to English semantics by Simmons (1973) and a UTLISP version was programmed by Matousek & Slocum (1972) While generally ignoring theoretical issues in linguistics, we do use such principles as the fact that sentences are composed of constituents, that there are syntactic rules defining acceptable sequences of constituents, and that underlying the English statement there is an idea that can be expressed in some other language by transformations on the English constituents The underlying idea can be expressed in a formal language such as some version of predicate logic, or in a computer data structure or in a language of functions and arguments such as LISP or PLANNER

In presenting the following grammar and semantic system our emphasis is on dealing with the highly variable nature of English embeddings This means that we have been more interested in the many forms of dependent clause--prepositional phrase, relative clause, infinitive, participial phrase, relative conjunctive clause, etc.--than in the fine detail on noun phrase, noun-noun combinations; and the fine grain of verb strings. We have also for the moment ignored ordinary conjunctions in view of the clear treatment offered by Woods, Winograd and Grishman; each of whom points out that an and or an "or" triggers a special subgrammar that attempts to find a structural repetition of a constituent that was just completed. Because of our interest in embeddings we have chosen to consider relative clauses at the top level of the grammar where possible.

The following constituent description defines a very fluid subset of English with great potential for embeddings.

```
   CLAUSE → (NP) + (VP)
          → (DCLAUSE) + CLAUSE
       NP → (ART) + (ADJ*) + N + (DCLAUSE)
          → PRON + (DCLAUSE)
       VP → VG + (NP)
       VG → (AUX*) + (ADV) + V + (ADV)
  DCLAUSE → PP | RELCONJ | RELCLAUSE | VMOD
       PP → PREP* + NP
  RELCONJ → RCONJ + CLAUSE
RELCLAUSE → (RELPRON) + PRONCLAUSE
PRONCLAUSE → VP | NP + VG + (DCLAUSE)
     VMOD → VPAST/VP | VPRESPART/VP | VINF/VP

    VPAST → SUPPORTED  SAILED...
     VINF → TO SUPPORT,...
VPRESPART → SUPPORTING, SAILING...
```

RELPRON → WHO, WHICH, WHAT THAT

RCONJ → BEFORE, AFTER WHILE

AUX → IS WAS; HAS, HAVE, HAD

ADV → HORIZONTALLY, VERTICALLY

V → SUPPORT BALANCE, SAIL

PRON → HE, SHE, IT, THEY ...

ART → A, AN, THE ...

ADJ → LARGE, SMALL, TINY

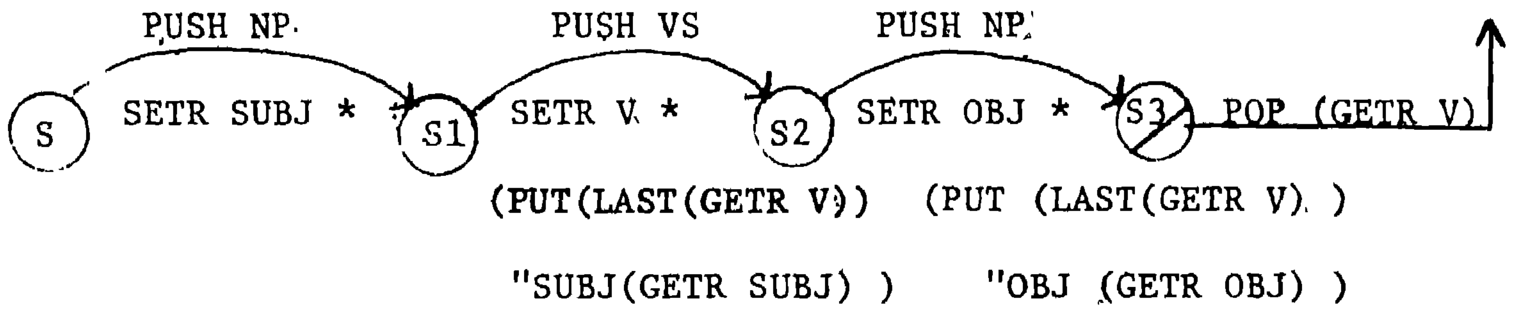
1N → CLOWN, PEDESTAL, BOAT, DOCK, FEET, TOP, SIDE ...

In the above: + means "followed by", (x) means optional x. x* means 1 or more x's, "|" or "," means "or", "..." means etc., and x/y means x is the initia element of y. The arrow → means "defined by".

The form of notation above is a concise recursive description for the ordering of constituents. It shows nothing about the semantics that may be included in the system, and the flow of control for parsing is not at all obvious. Augmented Transition Network graphs following Woods show the conditions on elements of the sentence and the flow of control in terms of directed arcs leaving nodes in a two-dimensioned diagram of the grammar Even more importantly, an ATN provides for the display of semantic operations that are to be undertaken on each constituent. The convention for drawing an ATN is to write conditional statements above the arcs, and operations below.

For example:

S := NP + VS + NP



In this net, if the sentence begins with an NP, the PUSH NP will return the structure of an NP in the * register. At that point the register SUBJect is set to that value. When a VString is analyzed by PUSH VS then V is set to the value VS returned. At this point further structure is built by PUTting on the verb's property list the attribute SUBJ with the value contained in the register SUBJ. Similarly, when an OBJect NP is parsed, it can be added to the structure of V and the value of S can be POPped--i.e. returned--as the register V which will allow access to the property list of the verb on which the values of subject and object can be found by consulting those properties as in (GET (LAST(GETR V)) "SUBJ). The function LAST is used in this example to obtain the last element of a list.

Notice this example illustrates that our general approach to recording semantic information is one of putting detailed information such as the arguments or cases of a verb on the property list of that object. Thus the result of parsing "clowns hold poles" with the above net is:

(HOLD SUBJ CLOWNS, OBJ POLES)

In fact, it is necessary to create new names for each word used in a sentence--to avoid clobbering dictionary information--so the result from actual nets would be:

(C1 TOK HOLD, SUBJ C2, OBJ C3)

(C2 TOK CLOWNS, NBR PL, DET INDEF)

(C3 TOK POLES NBR PL, DET INDEF)

The relation TOK shows that C1 is an instantiation of the lexical item HOLD. In this convention for stating property list values, the first element is the ATOM and each pair separated by commas is an ATTRIBUTE and its VALUE.

The Woods system also stores its past states and provides for backup in the event that no conditional arc succeeds and yet there is still sentence to be scanned. In this event the system recursively consults the state leading to the current node to see if there were arcs that were untried that lead to a successful parsing for the sentence string. The * register has special significance in that ordinarily it contains the sentence element under the scanner, except when a subnet such as NP returns a value, in which case the POP arc sets the value in the * register. The overall flow of control through an ATN is that * is set to the first element of the sentence, then the topmost net, CLAUSE or S, applies the grammar in topdown fashion. Each time a constituent --a word, a phrase, a clause--is recognized and control is passed to another node, the scanner is advanced and parsing proceeds from the new node.

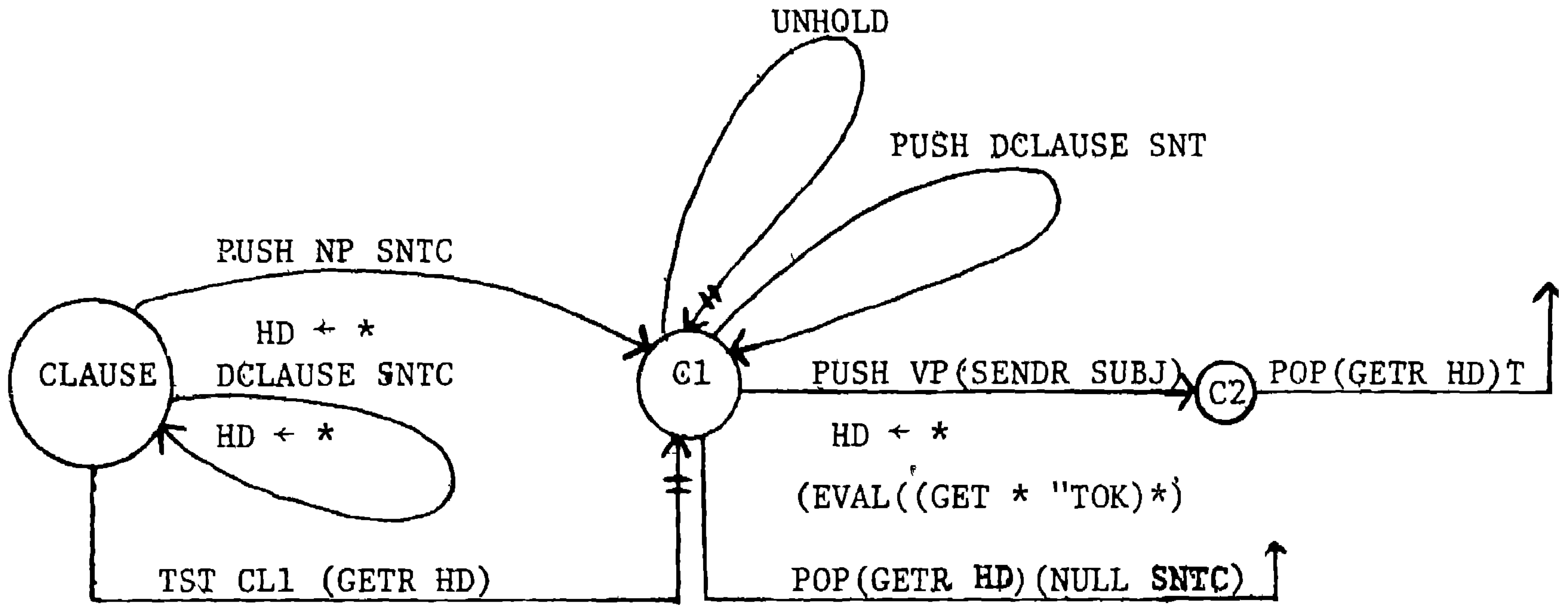
For programming simple grammars without much embedding and without backup capabilities an ATN may be used as a flow chart to design the program. If more complex grammars are required, Woods has provided a complete set of language conventions and an Interpreter with the capability of storing past states and backup.

Lexicon: English words, their word classes and features and other information such as program definitions etc. are recorded on a property list structure for easy access by functions used in the ATN. The following examples illustrate this structure:

```
(CLOWN (N T)(NBR SING)(EXPR (LAMBDA()....))...(FEET XY)(ANIM T))
(BALANCE (V T)(TENSE PRES)(EXPR(LAMBDA(ST)....)) )
(ON (PREP T)(EXPR(LAMBDA(N1 N2)...)) )
(WHO (PRON T)(NBR (SING PL))(PERSON T) (RELPRON T) )
```

The function (PUT X Y Z)--e.g. (PUT "CLOWN "NBR "SING)--will add the pair (Y Z) to the atom X or replace the value of X's attribute Y with the new value Z. The function (GET X Y) will then return the value Z. Such ATN functions as CAT and GETF simply call GET with the first argument set to the value of the word under the sentence scanner.

The EXPR values associated with an English word are semantic functions that are explained later. Many modifications to this simple scheme can be added to provide for morphological variants referring to root forms instead of requiring a definition of their own, and an attribute, FOLLOWEDBY, can be used to collect multiple word terms. The basic property list representation of a dictionary can be expanded to include multiple word senses as well, but it always retains the character of a basic LISP system for storage and retrieval of data associated with an atom.

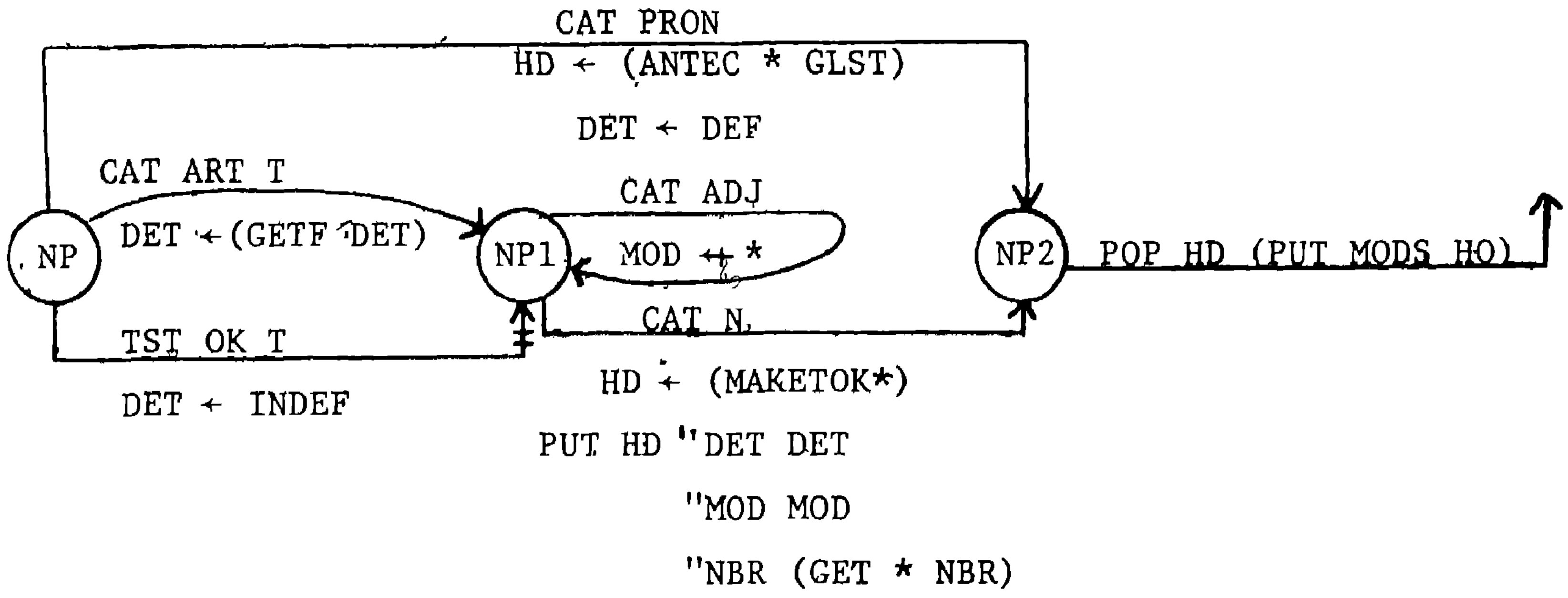


Grammar: This is the toplevel net for the grammar. It is named clause and transfers control to states C1 and C2 each of which can POP a value in the event that the sentence string has been completed or a clause successfully parsed. The barred pointer, ⇸ , indicates a HOP operation which passes control without advancing the sentence scanner or changing the * register.

This net accepts sentences beginning with an NP, a VP or a dependent clause. HD is the name of a register that generally contains the last constituent found. The UNHOLD arc emanating from C1 causes a list, HOLD, to be processed. HOLD contains Dependent Clauses that are missing some element that delays their semantic processing. For example, "on his nose" in "on his nose a clown balances" cannot be semantically processed until "clown" shows up as a following NP. The net is satisfied by a sentence or by a single noun phrase such as "a clown in a boat" or by an imperative, "balance a pedestal". It does not accept question forms; that would require an additional arc from CLAUSE labelled, PUSH QFORM SNTC. The ordinary form of an arc is an arc-label such as CATegory, PUSH, POP, TST followed by its argument, followed by any condition statement. SNTC is simply the variable that contains any remaining séntence string, so the condition SNTC is true except when the string has been exhausted. If SNTC is nil, there is no point in further processing.

The arc PUSH VP (SENDR SUBJ) will send the value of the register SUBJ to the subnet VP.* If VP is successful, the operation under the arc (EVAL((GET * TOK) *)) will call for a function associated with the verb to translate the subject, object and complements of the sentence into the particular semantics of pictorial relations. The verbs SUPPORT, SAIL, and MOVE are defined as semantic functions in section V. Prepositions are also defined as semantic functions in that section.

When HD is popped from C1 or C2 it contains the name of an object on the property list as described earlier. The result of a parse is an atom name whose property list contains labelled references to its arguments which are either symbolic or numeric values, or references to other atoms which have property lists. This of course is a property list representation of a semantic network.



* SENDR is usually signified in the nets by ↓ Thus ↓ SUBJ means (SENDR SUBJ (GETR SUBJ)) X ++ Y means (SETR X (APPEND Y (LIST(GETR X))))

This NP net is operated on the call, PUSH NP T. It allows for a pronoun or a sequence of (art)(adj*) N. Its operation includes some basic semantic transformations on the head noun. If the sentence begins with an ARTicle, the determination is set to DEFINITE or INDEFINITE depending on what feature GETF finds associated with it. A pronoun implies definite determination, and a noun phrase without an article implies indefinite except in the case of proper nouns not considered in this net. Adjectives are appended to a list named MOD.

When the noun head is encountered, MAKETOK creates an atomic name Ci using the LISP function (GENSYM C) and puts on its property list, the pair, TOK WORD. The remaining operations under the CAT N arc add property value pairs to this TOKen of the noun. From NP2 the arc, POP HD (PUTMODS HD), is encountered. PUTMODS is a semantic function that works with adjectives and adverbs in the following fashion:

An adjective, e.g. big, has the following lexical structure:

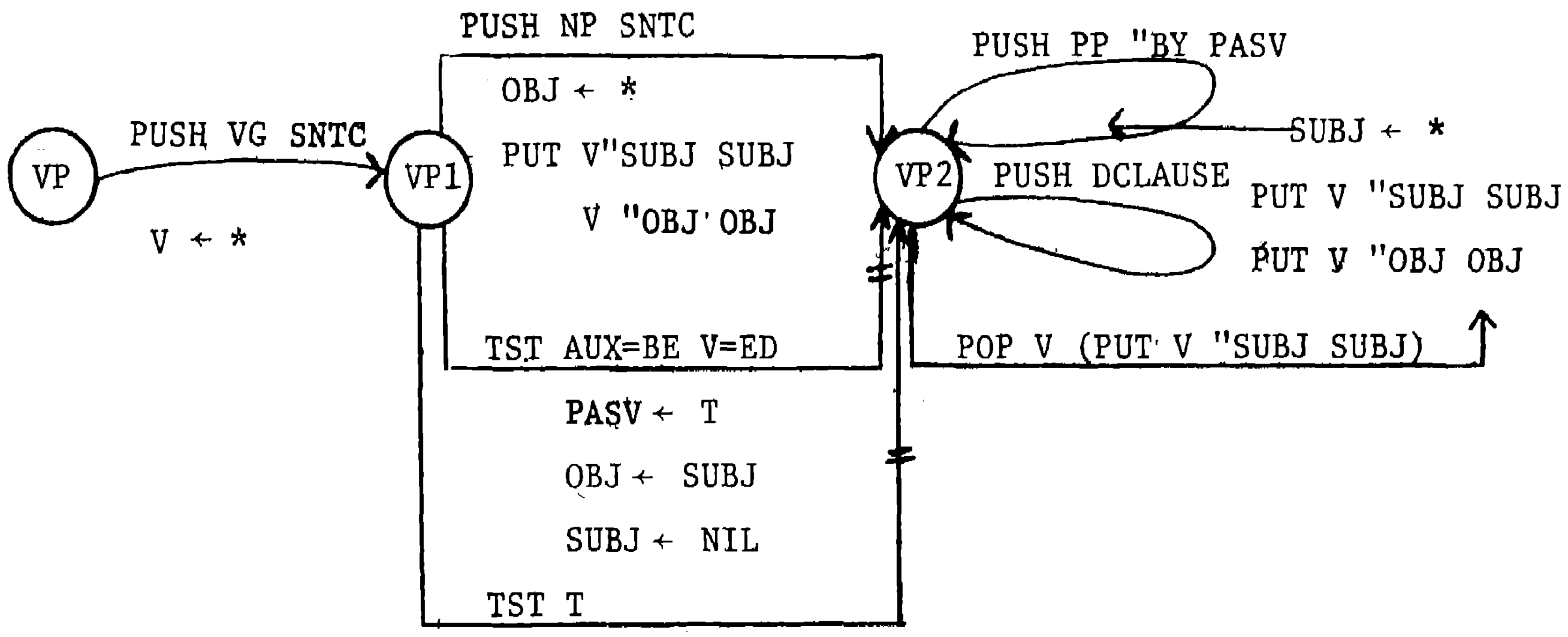
(BIG ADJ T, POS T, TYPE SIZE, VALUE 7)

PUTMODS will for each adjective obtain the TYPE and VALUE and put them on the noun's property list. Thus, "a big red clown" results in:

(C1 TOK CLOWN, DET INDEF, NBR SING, SIZE 7, COLOR 1)

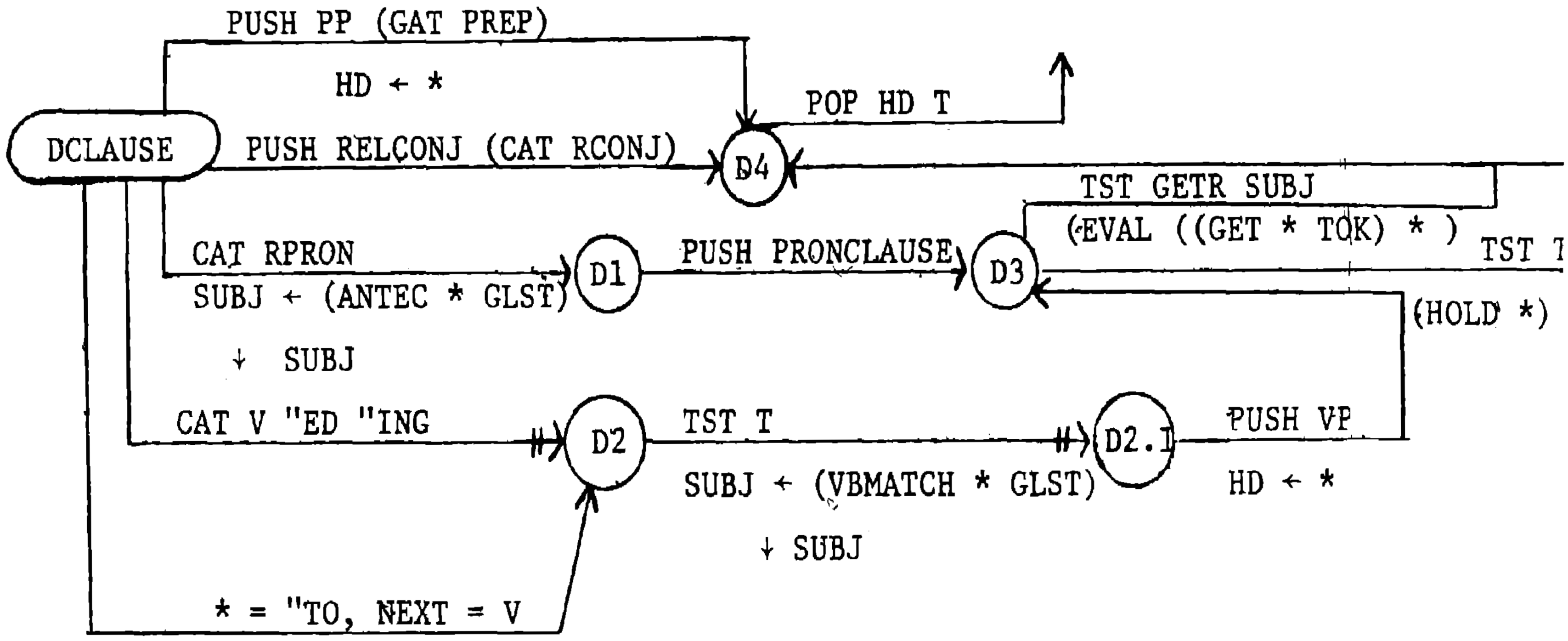
where COLOR 1 assumes that some mechanism for assigning colors likes numbers as inputs, even as the drawing programs require numerical values for SIZE.

The result of parsing a noun phrase with this network is to return the semantic structure of an object as a set of property-value pairs associated with the name Ci which is a token of the word used. The net is not sophisticated as NP definitions go, much more complete grammars of the NP are offered by Winograd and Woods. The lack of a continuation into a modifying clause such as a PP or relative clause is deliberate in that we prefer to return control to the structure calling the NP so that its syntactic-semantic position in the higher sequence can be used by the Dependent Clause net.



This VP net first pushes a VG, verb group. VG is not shown in this discussion, but it scans the sentence string for an acceptable sequence of auxilaries, and adverbs dominated by a verb. It makes a token of the verb and puts its tense and auxiliaries on that token as property value pairs. It returns the token name. In exiting node VP1 we seek an NP as a syntactic OBJect and finding one, add the subject and object as properties of the verb. If no NP follows the verb, the next arc tests to determine whether the verb is a passive form and if so sets the flag PASV, sets object to subject, and subject to nil. If a "by" prepositional phrase follows, it becomes the subject. Additional modifying phrases are picked up by the DCLAUSE loop. No actions are associated with PUSH DCLAUSE arcs because each DCLAUSE calls semantic routines that bind the modifier to the noun or verb it modifies--frequently not the one it immediately follows.

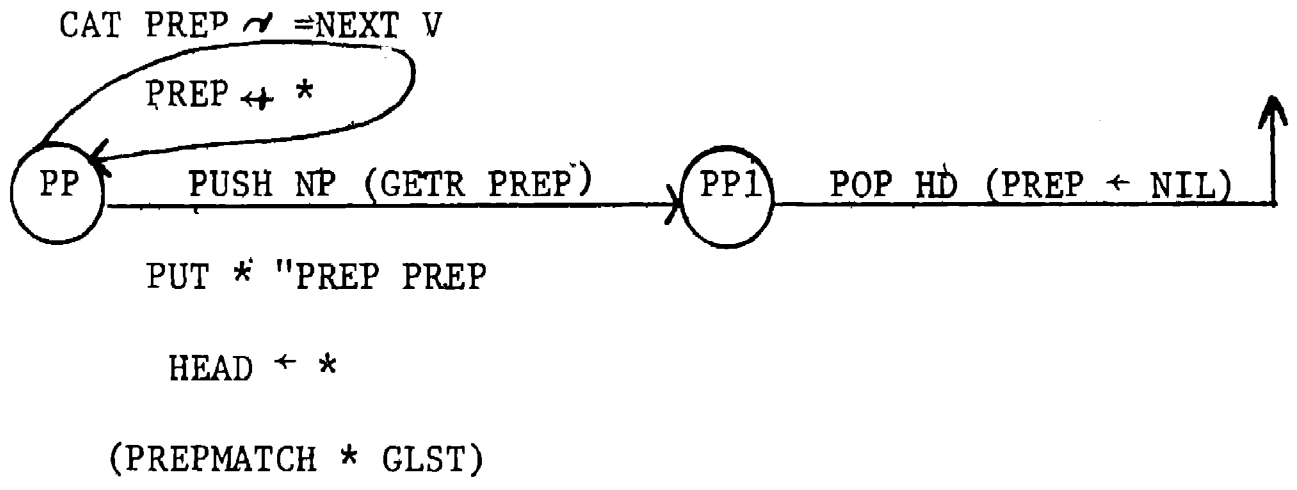
The VP net accepts a verb, a verb group, or a verb group followed by an NP and a string of PPs or other modifying clauses. It lacks the case of two NPs to account for direct and indirect objects.



The DCLAUSE net is fairly intricate in that it accounts for PPs, relative pronoun clauses, infinitive modifiers, participial clauses and clauses introduced by relative conjunctions. A PP is one or more prepositions followed by an NP. A RELCONJ starts with an RCONJ such as "while", "after" etc. and may be followed by a DCLAUSE or a CLAUSE. A relative pronoun clause begins with an optional relative pronoun and is followed by a pronoun clause which is either a VP or an NP followed by a VG and optional DCLAUSES. For the moment we insist for computational economy that a relative clause be introduced by a relative pronoun; actually the form of a pronominal clause is sufficiently well defined that PUSH PRONCLAUSE can identify it without a relative pronoun in most cases.

When a pronoun is found, here or in an NP, the function ANTECedent is called to scan the list of preceding nouns to find the best agreement in person, number, and gender. The function VBMATCH on the exit from node D2 is a function that seeks to find the head that the participial or infinitive phrase is modifying. As in PREPMATCH, the head noun is frequently not the one just preceding the modifying phrase ~~and~~ the particular verb and its ending are used in choosing its head noun or verb. GLST is the name of a list of candidates.

In the event that the DCLAUSE is a relative pronoun or a participial or infinitive construction, the final step is to call the semantic function associated with the verb and evaluate it for the subject, object and complement arguments. DCLAUSE is undefined for adjectival and adverbial clauses that can be used as modifiers. When defined they can be added as additional arcs.



This abbreviated PP net is presented to call attention to its method for accepting a string of prepositions and for accomplishing the semantics by calling PREPMATCH. Although Section IV concerns semantics, it is worth noting that the effect of PREPMATCH is to add information to the semantic structure representing a noun or a verb. For examples:

"a clown on a pedestal on his nose "

(C1 TOK CLOWN, SUPPORTBY #PEDESTAL, BALPT #NOSE)

"...balances on a pedestal on his nose"

(C2 TOK BALANCE, TENSE PRESENT, COMPS(#PEDESTAL #NOSE))

Thus if a verb intervenes between a noun and prepositional phrases that might modify it, the PPs become COMPlements to the verb under the attribute COMPS, and the verb's semantic function has the task of relating it to other elements of the sentence.

V Semantics of the Subset

Parsing a sentence with the ATN grammar just described results in a set of symbols each of which is further characterized by attributes and values on a property list. If no semantic functions were applied--such as those associated with prepositions, modifiers and verbs--the result would be a tree such as the following:

(C1 TOK BALANCE, SUBJ (C2 TOK CLOWN, DET DEF),
OBJ (C3 TOK POLE, DET INDEF),
COMPS (C4 TOK HANDS, POSSBY C2, PREP ON))

he effect of the semantic functions for this sentence is to produce the f llowing:

(C2 TOK CLOWN, SUPPORT C3, SIZE 3, ATTACH ($C2_{xy}$ $C3_{xy}$))
(C3 OK POLE, SUPPORTBY C2, SIZE 3, ATTACH ($C3_{xy}$ $C2_{xy}$))

which is minimally sufficient information for the graphics to produce a single icture to represent the state of affairs the sentence described.

It s perfectly feasible to compute the syntactic form first and then apply the semantics, but as Winograd, Riesbeck and others have found, the early application of semantics can be used to minimize the ambiguities of the syntax. For this reason, as each prepositional phrase is parsed a semantic function is called to determine which noun or verb might be its governor or head. Each time a Verb Phrase is completed, a semantic function is called to translate its syntactic arguments, i.e. SUBJ, OBJ, COMPS, into pictorial relations such as SUPPORT, ATTACH points, etc.

Semantics of Prepositions: After a PP constituent has been identified, a function PREPMATCH is called with a list of the nouns and verbs so far encountered, GLST. Each preposition is associated with a function that examines a candidate head from GLST and the noun object to determine if the candidate can dominate the PP in question. For example "ON" is defined as a LISP function with two arguments. When called with "clown" and "nose", ON returns a structure in which the ATTACH point of the clown is the XY coordinates of his nose. When called with "clown" and "pedestal" it returns a structure in which the pedestal SUPPORTS the clown. If called with "nose" and "pedestal" it returns NIL since nose is neither an independent picturable object nor a part of the pedestal.

PREPMATCH does the book-keeping by calling the preposition function with each candidate from the GLST. If the candidate is a verb that can be modified by that preposition, PREPMATCH adds the PP to the verb's list of COMPS, and the verb semantic function will interpret it. The function BESIDE offers a simple example definition that shows how one preposition can imply another.

```
(BESIDE(LAMBDA(N1 N2)(RIGHTOF N1 N2) ))
(RIGHTOF(LAMBDA(N1 N2)
     (COND((AND(GET N1 "PICT)(GET N2 "PICT))
```

(PUT N2 "RIGHTOF N1)(PUT N1 "LEFTOF N2))

(T NIL))))

Thus. "a beside b" is quite arbitrarily interpreted to mean "b is to the right of a". RIGHTOF requires that its two arguments be picturable objects. "A clown on his nose beside a pedestal" causes PREPMATCH ((NOSE CLOWN) PEDESTAL). PREPMATCH first calls (BESIDE NOSE PEDESTAL) BESIDE calls RIGHTOF which returns NIL because "nose" is not an independent PICTure. Then PREPMATCH calls (BESIDE CLOWN PEDESTAL) and the return is (essentially*) PEDESTAL RIGHTOF CLOWN.

Somewhere else in the forest, the relation RIGHTOF will be interpreted to mean contact between leftside and rightside of two objects. So we quite arbitrarily force a precise meaning--so far sufficient for our purpose--on the geometrically vague term, "beside". In general the prepositional semantics for a microworld model are definable where the number of possible meanings for each preposition are limited by the situation. In the CLOWNS world, "with" "on" and "by" have multiple meanings that are selected in accordance with the conditions described by their semantic functions. In contrast, "from" so far has a single meaning.

Verb Semantics: The English verb is a remarkably complex conceptual object. It may carry several meanings dependent on its arguments and on its larger context. It communicates information about temporal ordering of its process by auxiliaries and its suffix. It implies one or a sequential series of events. Its syntactic position and ending can be used to signal that it is a pre-modifier or a post-modifier for another verb or a noun. It is part of a classification structure and may imply special argument values to some more general verb higher in the classification. For example.

* Where these examples use words the functions are using Ci tokens or words as appropriate.

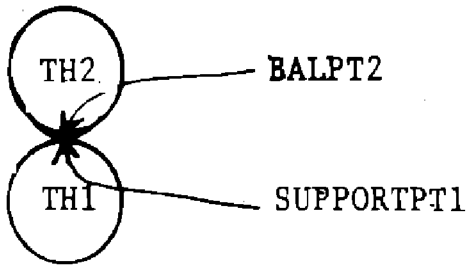
"retort" means "answer sharply" which means "communicate sharply in response to a communication". The verb may imply special arguments in another way; the verb, "sail", implies that "someone caused a vehicle to move through a fluid by a means involving aerodynamics from one place to another" If the sentence omits some of these arguments, the verb semantics implies them. Thus we can sail a boat, a kite, an airplane, a saucer, but hardly a locomotive or a desk. If the arguments are inappropriate we can ascend the classification tree and call the statement a metaphor. In addition, the verb allows its arguments to occupy practically any syntactic position in the clause or sentence and must sort them out on the basis of semantic information.

By analogy, a verb is a dramatic skit with a variable set of characters that successively relates the character roles to one another over a period of time. A verb has a set of arguments, case roles filled by semantic objects; it has an initial state, a set of relations among its characters; a set of intermediate states, one or more sets of relations among its characters; and a final or resulting state similarly characterized. In addition, recent work particularly by Abelson and Schank suggest that in a given culture a verb models a situation that is predictably preceded and followed by more or less typical situations. If a person strikes another person, ,the first one was probably angered by the second, dominates the second, etc. while the second feels pain, may react in anger, etc. So it is reasonable to suppose that our experience is organized in scripts, frames, scenes, dremes, etc. whose component elements include the dynamic skits that verbs signify.

In the CLOWNS world a verb selects an associated semantic function to sort its arguments into typical roles in its picturable dramatic skit and relates them in typical ways for display as initial, intermediate and final conditions. In this fashion, the verb "sail" relates an Agent, a Vehicle, a Medium, a Start point, Intermediate points, a Goal point and possibly a Means of movement. The semantic routine must translate syntactic entities such as Subject Object and Complements into these roles, i.e. bind the variables. It must then relate them in appropriate ways-- AGENT IN VEHICLE. VEHICLE AT STARTPOINT, VEHICLE ON MEDIUM, etc.--for each of its temporal states and call the graphics system to display them.

Support is a verb that describes a static single state of affairs in "The world is supported on a turtle's back". The verbs "balance", "support", "stand" "hold", are each associated with the semantic function SUPPORT1. When a VP constituent using one of these verbs is completed, SUPPORT1 is called to compute a model of the situation described.

SUPPORT1 binds the cases TH1, TH2, SUPPORTPT1, BALPT2. TH stands for THEME and the other two cases for Support Point and Balance Point. The following diagram shows the spatial relations signified by these cases:



Th1 supports TH2 on its BALPT2 on/with/in his SUPPORTPT1. If these four arguments are bound. the support relation is completely defined. If not, means are taken to fill in the missing arguments by a default logic. SUPPORT1 takes as arguments, SUBJ, OBJ, and COMPS where COMPS is a list of complements. SUBJ and OBJ were computed by the VP parser as the subject and object of the ACTIVE form of the clause.

The conditions or rules for transforming these syntactic arguments into semantic roles are as follows:*

SUBJ $\wedge$ OBJ $\rightarrow$ TH1 $\leftarrow$ SUBJ, TH2 $\leftarrow$ OBJ

SUBJ $\rightarrow$ TH2 $\leftarrow$ SUBJ

OBJ $\rightarrow$ TH2 $\leftarrow$ OBJ

For each COMP,

$\sim$TH1 $\wedge$ ON $\wedge$ PICT(COMP) $\rightarrow$ TH1 $\leftarrow$ COMP

$\sim$SUPPORTPT1 $\wedge$ **IN** $\vee$ **ON** $\vee$ WITH $\wedge$ PART(COMP TH1) $\rightarrow$ SUPPORTPT1 $\leftarrow$ COMP

$\sim$BALPT2 $\wedge$ ON $\wedge$ PART(COMP TH2) $\rightarrow$ BALPT2 $\leftarrow$ COMP

T $\rightarrow$ PRINT (LIST "UNDEFINED COLON COMP)

For the following two example sentences, the above rules result in the bindings shown:

```
Ex 1  A clown balances a pedestal on his head on its side
          |                 |              |          |
         TH1               TH2        SUPPORTPT1   BALPT2

Ex 2  A clown balances on a pedestal on its side on his head
          |                    |            |          |
         TH2                  TH1      SUPPORTPT1   BALPT2
```

Additional modifiers may have been present as in the example sentences:

A clown on his hands balances a pedestal on his head, on its side beside a pole.

A clown with a pole in his hands balances on a pedestal...

The earlier action of the preposition semantic functions will have reduced these additional complements to no more than those shown in Examples 1 and 2.

---

*Notes: $x \rightarrow y$	X implies Y	$\wedge$ and
$x \leftarrow y$	SET x to y	$\vee$ or
$\sim x$	Not X	" Quote
F(x)	Evaluate function F of X	

Brief forms such as "A clown balances on his hands" or "A clown holds a pole" result in incomplete bindings from the rules of SUPPORT1. The legitimacy of such brief forms requires a default logic that in the first case assumes that the Ground supports the clown at a point called TOP of the ground. In the second case, the clown's SUPPORTPT1 for the pole is bound to his hands and the BALPT2--for the pole-- is bound to the BOTTOM of the pole. The verb "hold" puts a default value of "hands" on the structure it passes to SUPPORT1 according to the following definition:

```
(HOLD(LAMBDA(ST) (PROG()
        (PUT ST "SUPPORTPT1 "HANDS)
        (RETURN (SUPPORT1 ST))     )))
```

The default logic of the verb seeks these values to bind them appropriately to any empty case arguments. The more general default values of TOP as a missing SUPPORTPT1 and BOTTOM as a missing BALPT2 and the fact that the object on the bottom of the heap must be supported by the GROUND are all supplied just prior to constructing a picture frame.

The result of SUPPORT1 is to create a process model of the following form:

```
(Ci TOK balance, GLOBAL (...),INIT(...),INTER(...)
                 RESULT (...))
```

The value of the attribute, GLOBAL is a quoted set of (PUT X Y Z) which are true at all times in the model. INIT is the set of relations true at the initial state of time in the model, INTER is those for the intermediate states, and RESULT is the set for the final state. When a function PRAG for Pragmatics evaluates one of these attributes, the result is to evaluate these PUT functions to produce a semantic network representing the state of affairs at a given instant of time. The semantic relations are translated to ATTACH 4-tuples which then generate a picture of the state. Successive pictures are obtained by calling PRAG repeatedly for INITial, INTERmediate, and RESULT states.

For the examples of the SUPPORT1 verb, only the GLOBAL attribute is given values as follows:

```
C1...,"GLOBAL(LIST ("PUT TH1 "SUPPORT TH2)
                    ("PUT TH2 "SUPPORTBY TH1)
                    ("PUT TH1 "SUPPORTPT SUPPORTPT1)
                    ("PUT TH2 "BALPT BALPT2) )
```

Initial, Intermediate and Result states are null since the verb simply describes a static state.

The verb MOVE* is more complex and more interesting. Let us assume as input the sentence, "A clown on his head sails from Como to Menaggio" When the parser has completed its VP the semantic structure is as follows: (abbreviated to the portion relevant to this discussion.)

```
(C1 TOK CLOWN, BALPT HEADXY, SIZE 3)
(C3 TOK SAIL, SUBJ C1, COMPS (C4 C5), TENSE PAST)
(C4 TOK COMO,...,PREP FROM)
(C5 TOK MENAGGIO, ..., PREP TO)
```

At this point VP calls (SAIL C3). SAIL is defined as follows:

```
(SAIL(LAMBDA(ST)(PROG()
      (PUT ST "MEDIUM "WATER)
      (PUT ST "VEHICLE "BOAT)
      (RETURN(MOVE* ST))        )))
```

That is, SAIL implies a movement of a boat on water and so passes this information to MOVE* which may have to use it to bind its case roles of MEDIUM and VEHICLE which in fact are not mentioned explicitly in the example sentence.

MOVE* binds the arguments Agent, THeme, VEHICLe, Source, Goal, and MEDium by sorting out the information contained in SUBJ, OBJ and COMPS by the following rules:

ANIM (SUBJ) → A ← SUBJ

FORCE (SUBJ) → I ← SUBJ

VEHIC (SUBJ) → VEHIC ← SUBJ

∿ VEHIC ∧ VEHIC (OBJ) → VEHIC ← OBJ

MEDIUM (OBJ) → MED ← OBJ

OBJ → TH ← OBJ

FOR EACH COMP

∿ MED ∧ IN ∨ ON ∨ THROUGH ∧ MEDIUM(COMP) → MED ← COMP

∿ VEHIC ∧ IN ∨ ON ∨ WITH ∧ VEHIC(COMP) → VEHIC ← COMP

∿ S ∧ FROM ∧ PLACE(COMP) → S ← COMP

∿ G ∧ TO ∧ PLACE(COMP) → G ← COMP

T → PRINT (LIST "UNDEFINED-COMP: COMP)

DEFAULT:

∿ VEHIC → VEHIC ← (GET ST VEHIC); ∿ S → S ← (MAKETOK "POINT)

∿ MED → MED ← (GET ST MEDIUM) ; ∿ G → G ← (MAKETOK "POINT)

This definition of the conditions for MOVE* is still incomplete except for the verb "sail" and will be modified with further experience.

Having bound the role variables, MOVE* creates a process model by assigning to ST, sets of values for the attributes GLOBAL, INITial, INTERmediate, and RESULT.

For GLOBAL conditions,

```
(AND I (PUT S "SUPPORT I)                       (PUT MED "SUPPORT VEHIC)
       (PUT I "SUPPORTBY S) )                   (PUT VEHIC "SUPPORTBY MED)
(AND A TH (PUT A "LEFTOF TH)                    (PUT VEHIC "SUPPORT A)
          (PUT TH "RIGHT OF A))                 (PUT A "SUPPORTBY VEHIC)
(AND A (PUT VEHIC "SUPPORT A))                  (PUT MED "LEFTOF G)
(AND TH (NULL A)(PUT VEHIC "SUPPORT TH))        (PUT MED "RIGHTOF S)
```

For INITIAL,

```
(PUT VEHIC "RIGHTOF S)
(PUT S "LEFTOF VEHIC)
```

For INTERmediate,

```
(REMPROP VEHIC "RIGHTOF)
(REMPROP S "LEFTOF)
(PUT VEHIC "BETWEEN (S G) )
```

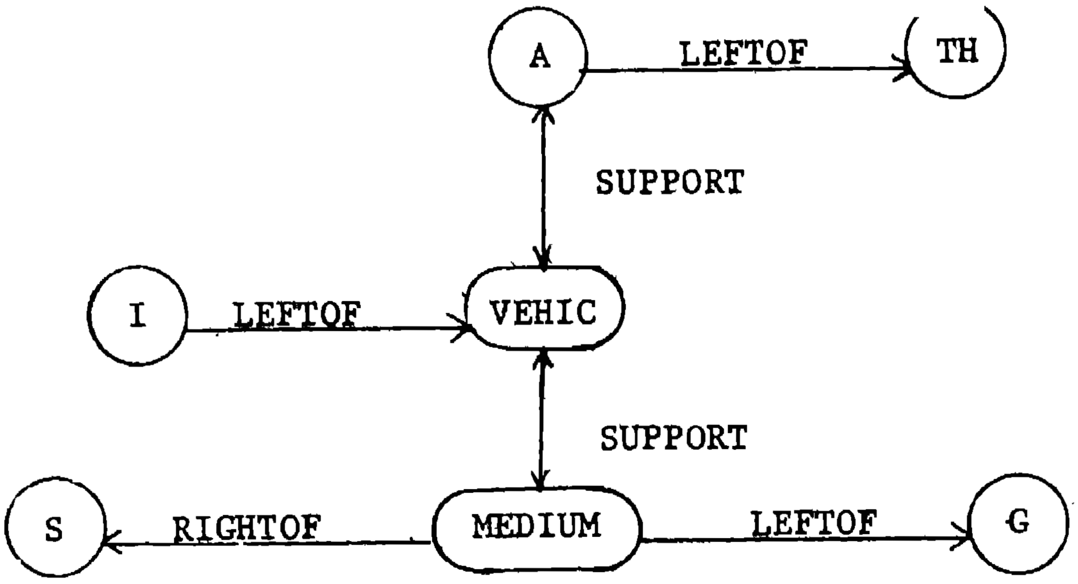
For RESULT,

```
(REMPROP VEHIC "BETWEEN)                        (PUT G "RIGHTOF VEHIC)
(PUT VEHIC "LEFTOF G)
```

Fig. 3 shows these states in the form of a process model.

When this process model, C3, is evaluated, the function PRAG is called with the arguments C3 and either INIT, INTER, or RESULT. PRAG will first interpret the GLOBAL attribute causing the state represented on the property lists for Tokens of clown, boat, etc. to be changed. It will then make the changes indicated by the PUTs which are additions, and the REMPROPs which are deletions. If PRAG is called three times in succession for INIT, INTER, and RESULT, three successive states are created to show the progression of the process from start to finish. After PRAG has been called the support points and balance points are all defaulted as necessary to TOPs and BOTTOMs by the function that calls the GRAPHICS system. This function

GLOBAL:



INIT:

PUT VEHIC "RIGHTOF S

INTER:

REMPROP VEHIC "RIGHTOF

RESULT:

PUT VEHIC "LEFTOF G

Figure 3. PRocess Model For MOVE*

also establishes horizontal contact points for BETWEEN, RIGHTOF and LEFTOF.

VI Semantics of Scenes

A scene is composed of a set of Pictures related to each other by adjacency and support relations including their points of contact. A picture is a LOGO display program that when called with a given start point and heading of the display turtle or cursor will construct a two dimensional line drawing. A square can be drawn by the following sequence of operations. (See Papert 1972.)

PENDOWN, FORWARD 20, RIGHT 90, FORWARD 20, RIGHT 90,
    FORWARD 20, RIGHT 90, FORWARD 20, RIGHT 90, PENUP.

The last "RIGHT 90" restores the cursor to its original heading. FORWARD and BACK are vector making functions that draw a vector from the current xy point of the curser a given number of units in the direction the cursor is aimed. The language uses functions with arguments and may create and call subroutines. Square may be defined as

SQUARE :SIZE: FORWARD :SIZE,.....ETC.

If a triangle has also been defined, we can then define:

HOUSE :SIZE; SQUARE :SIZE; FORWARD :SIZE; TRIANGLE :SIZE;

It is the convenience and simplicity of these LOGO conventions that convinced me that drawing pictures from sentences would not add any great complexity to a basic language analysis system. LOGO offers many additional features as a language for teaching programming skills to non-mathematically oriented users and one of the most important of these may be as a parenthesis-free form of LISP.

In our use of LOGO graphics, we consider that a picture has a name, a program to draw it, a cursor startpoint value, a heading, a size, a frame of minimum and maximum X and Y coordinates, a center of gravity and coordinates associated with any points on it that we need to refer to, such as feet, hands, head, top, bottom, etc.

```
CLOWN EXPR  (LAMBDA () ...)
      SIZE  1
   STARTPT  (XY)
   HEADING  NBR
    PFRAME  (MIN X, MAX X, MIN Y, MAX Y)
        CG  (XY)
      HEAD  (XY)
      FEET  (XY)
        .
        .
        .
    BOTTOM  (XY)
```

All of the XY coordinates designated in a picture structure are relative to the startpoint, heading and size. If we set the startpoint to a given value, say 500, 0; the clown will be drawn from the bottom center of the screen. If we set HEADING to 90, it will be drawn on its side. If we change size to 2 each vector composing the picture will be twice as long.

If we wish to translate the clown to the right 50 units, 50 is added to the X coordinate of the startpoint. IF we wish to move it up, a number is added to the Y coordinate of the startpoint. If we wish to rotate it onto its head with its head at 500,100 life is more difficult. We must use trigonometric functions to compute a heading value and a location of the startpoint that will achieve this result. A function called ORIENT* takes as arguments an object, its balance point, and a reference point.

(ORIENT* CLOWN, HEADXY, (500,100))

This function adjusts the startpoint and heading so that the head of the clown will be at (500,100) with the center of gravity above the point. Similar adjustments are made to the PFRAME values to translate and rotate the imaginary picture frame defined by the XY extremals.

To assemble a set of pictures into a scene, the bottom picture is assigned an XY startpoint and heading. Each picture it supports is translated and rotated to result in adjustments to startpoint, heading and pframe values. Each picture beside it is similarly adjusted until a scene is completed by accounting for all its pictures. At this point, the scene is scaled to the size of the display screen, and the picture drawing programs are executed.

The PFRAME concept developed by Gordon Novak and Mike Smith is very helpful as a computational abbreviation for the program that draws the picture. The PFRAME attribute has a minimum x, maximum x, minimum y, maximum y as four points that define a rectangle that surrounds the extreme points of the picture. When the picture is programmed these are assigned by hand with reference to whatever startpoint and heading were used. The picture as defined is taken as size 1. Whenever the picture is translated or rotated the values of PFRAME, STARTPT and HEADING are adjusted accordingly. As each pair of pictures are combined into a scene, a PFRAME is computed for the scene. The final PFRAME for the entire scene is adjusted to the size of the screen with appropriate scaling of the size values of its component pictures.

A frequent use of PFRAMES is to find default values for TOP, BOTTOM, LEFTSIDE and RIGHTSIDE as contact points between pairs of pictures. Detailed descriptions of these processes are not particularly relevant to this paper's goal of presenting an easily computable syntactic-semantic scheme for subsets of English but will be presented in forthcoming papers by Bennett-Novak and by Michael Smith.

VII CONCLUDING DISCUSSION

In previous sections the terms "process model", "skit", "scene" and "pframe" have been used to describe very limited structures of verb and noun semantics. This usage is in contrast to the much broader ideas associated with "scripts", "frames" etc. which are typically used to describe worlds of vision and belief systems. Example process models for "support" and "move" have been described and applied to the task of organizing images into scenes. Nouns such as "clown", "dock", "pedestal", etc. have been represented as programs that construct line drawings. Adjectives have been used to communicate variations in size, and adverbs to indicate angles. Other nouns, such as "top", "bottom", "edge" etc. are defined as functions that reference particular x-y coordinates of a picture.

Nouns such as "circus", "party", "ballgame" etc. have not yet been attempted. They imply partially ordered sets of process models and are the most exciting next step in this research. More complex verbs like "return" or "make a roundtrip" imply a sequence of interacting process models. Thus. "a clown sailed from the lighthouse to the dock and returned by bus" offers interesting problems in discovering the arguments for MOVE*-return as well as in the design of a higher level process model whose intermediate conditions include the models of MOVE*-sail and MOVE*-return.

We have also noticed that the semantic network that is produced as a result of semantic analysis can be seen as a problem graph by the functions that organize images and it is apparent that as these graphs come to contain larger numbers of images, it will be necessary to develop graph searching strategies along the lines of ordinary problem solvers. Our first experiment in this line will be to semantically analyze the missionaries and cannibals problem and illustrate the solution.

As it stands, the CLOWNS system has served as a vehicle for developing and expressing our ideas of how to construct a tightly integrated language processing system that provides a clearcut syntactic stage with coordinate semantic processing introduced to reduce ambiguity. Two stages of semantic processing are apparent; the first is the use of prepositions and verbs to make explicit the geometric relations of "support", "leftof" etc. among the objects symbolized by the nouns; the second is the transformation of these geometric relations into connected sets of x-y coordinates that can be displayed as a scene. Schank's notion of primitive actions is reflected in our approach to programming high level verbs such as MOVE* to encompass the idea of motion carried in verbs such as "sail", "ride", etc. Woods' ATN approach to syntactic analysis is central to this system and in sharp contrast to the approach of Schank and Riesbeck who attempt to minimize formal syntactic processing. Our process model reflects the ideas developed by Hendrix in his development of a logical structure for English semantics.

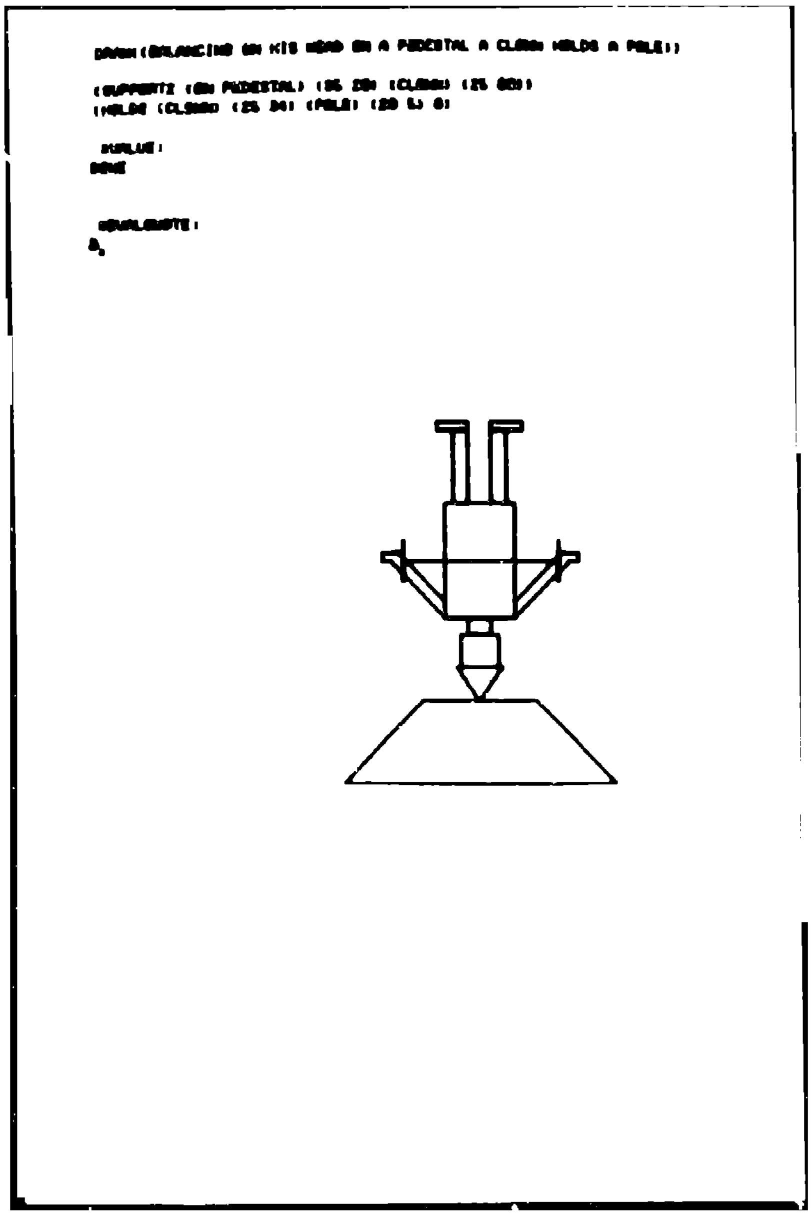
The system is not limited to its present grammar nor to its present vocabulary of images. Picture programs to construct additional objects are easily constructed and the semantic routines for additional verbs and prepositions can be defined fo the system with relative ease. We hope in the near future to illustrate the following sentence:

"One of the first plants to appear on a newly formed volcanic island is the stately and graceful coconut palm." This will involve programming the verbs, "appear", "form", "grow", and programming pictures of plants, coconut palms and islands. Very interesting problems are apparent in understanding and representing the ideas of "first" and "new" as well as in the relation between "plants" and "coconut palms".

The system has been used successfully to communicate methods for natural language computation to graduate students and to undergraduates. It appears to have immediate possibilities for teaching the structure of English, for teaching precision of English expression, and for teaching foreign languages through pictures. Eventually it may be useful in conjunction with very good graphic systems for generating animated illustrations for picturable text.

In my mind CLOWNS shows the power and value of the microworld approach to the study of Artificial Intelligence. By narrowing one's focus to a tiny world that can be completely described, one can define a subset of English in great depth. This is in contrast to the study of text where the situations described are so complex as to forbid exhaustive analysis. The translation into a visualized microworld provides an immediate display in a two-dimensional language of the interpretations dictated by the syntactic and semantic systems and thus a scientific measuring instrument for the accuracy of the interpretation.

Despite the potential for expansion of the system into the world of useful applications, I believe the primary value of this experiment with the CLOWNS world is to show that there exist orderly and straightforward ways of economically computing translations from subsets of English to procedures that do useful work. This is not a new finding but I believe the implementation is considerably simpler than most previous ones.



VIII REFERENCES

1. Abelson, Robert P., "Concepts for Representing Mundane Reality in Plans." In Representation and Understanding: Studies in Cognitive Science, edited by D. Bobrow and A. Collins Academic Press. In Press.

2. Badre, Nagib A., "Computer Learning from English Text." Electronics Research Laboratory, College of Engineering, University of California, Berkeley, 1972.

3. Bobrow, D., and Collins, A., Studies in Cognitive Science, Academic Press, New York 1975. In Press.

4. Bobrow, D.G. and Norman, D.A., "Some principles of cognitive systems." In Bobrow and Collins.

5. Bruce, Bertram C., "A model for temporal references and its application in a question answering program." Artificial Intelligence, 3 1972 pp 1-25.

6. Charniak, Eugene C., "Toward a Model of Children's Story Comprehension." AI TR-266, MIT, Cambridge, Mass., 1972.

7. Collins, A., and Warnock, E., "Reasoning from Incomplete Knowledge." In Bobrow and Collins.

8. Fikes, R.E., & Nilsson, N.J. "STRIPS: A new approach to the application of theorem proving to problem solving." Artificial Intelligence Vol. 2 pp 189-208.

9. Grishman, R., Sager, N., Raze, C. and Bookchin, B., "The Linguistic String Parser." AFIPS, Conference Proceedings Vol. 42; AFIPS Press, Muntvale, N.J. 1973, pp. 427-434.

10. Harris, Larry R., "A Model for Adaptive Problem Solving Applied to Natural Language Acquisition." Thesis MSS, Department of Computer Science, Cornell University Ithaca, N.Y. 1972.

11. Heidorn, George E. "Natural Language Inputs to a Simulation Programming System." NPS-55HD, Naval Post Graduate School, Monterey, Calif. 1972.

12. Hendrix, G., "Preliminary Constructs for the Mathematical Modelling of English Meanings." University of Texas, Department of Computer Sciences, Working Draft, April 1974. (not for distribution)

13. Hendrix, G.G., Thompson, Craig and Slocum, Jonathan. "Language Processing via Canonical Verbs and Semantic Models." Proc. 3rd Int. Jt. Conference on Artificial Intelligence, Stanford Research Institute, Menlo Park, Calif., 1973.

14. Hobbs, Jerry R. "A Model for Natural Language Semantics." Department of Computer Science, Yale University Research Report #36, 1974.

15. Matuszek, D., and Slocum, J., "An Implementation of the Augmented Transition Network System of Woods." Department of Computer Sciences University of Texas, Austin NL-9 1972.

16. Minsky, Marvin, "A Framework for Representing Knowledge," In The Psychology of Computer Vision, edited by P. Winston, McGraw Hill 1975.

17. Papert, S., "Teaching Children to be Mathematicians vs. Teaching About Mathematics." Int. J. Math. Educ. in Science & Tech., New York: Wiley & Sons, 1972; MIT, A.I. Memo # 249, July 1971.

18. Riesbeck, C. K., "The Conceptual analyzer." In Schank, R. Conceptual Information Processing.

19. Rumelhart, David E., "Notes on a Schema for Stories." In Bobrow and Collins.

20. Schank, R. & Abelson, R. "Scripts, Plans and Knowledge." MSS, Yale University, 1975.

21. Schank, Roger, Conceptual Information Processing, North-Holland Publishing Company 1975 (In Press).

22. Scragg, Greg W.,"LUIGI: An English Question Answering Program." Thesis MSS, AP & IS Dept., University of California, San Diego 1973.

23. Siklóssy, L. & Dreussi, J., "An Efficient Robot Planner that Generates its own Procedures." Proceedings 3rd. Int. Jt. Conf. on A.I., pp. 423-430.

24. Simmons, Robert F., "Semantic Networks: Their Computation and Use for Understanding English Sentences:" In Computer Simulation of Cognitive Processes, edited by Schank, R. and Colby, K., Prentice Hall, N.Y. 1973.

25. Walker, Donald E., "Automated Language Processing," in Annual Review of Information Science and Technology Vol. 8, American Society Information Science, Washington, D.C. 1973.

26. Winograd, Terry, "Frame Representations and the Declarative/Procedural Controversy." In Bobrow and Collins.

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

28. Woods, W.A., Kaplan, R.A., & Nash-Webber, B., "The Lunar Sciences Natural Language Information System: Final Report: BBN Report # 2378, June, 1972, Bolt, Beranek & Newman, Inc. Cambridge, Mass.

29. Woods, Wm. A., "Transition Network Grammars for Natural Language Analysis," Comm. ACM, 13, Oct. 1970.

CLOWNS
PROGRAMMED BY R. SIMMONS AND BY G. BENNETT-NOVAK

AFTER SOME CONTROL FUNCTIONS THE PRINTOUT SHOWS THE GRAMMAR, THE LEXICON, THE FUNCTIONS THAT DRAW PICTURES, THE SEMANTIC FUNCTIONS ASSOCIATED WITH WORDS; THEN THE BASIC GRAPHIC FUNCTIONS APPROXIMATING LOGO EQUIVALENTS, AND FINALLY THE DEEP SEMANTIC FUNCTIONS FOR COMBINING AND ASSEMBLING DRAWINGS INTO SCENES.

THE PROGRAM IS IN UTLISP FOR CDC EQUIPMENT AND IS WRITTEN TO INTERFACE WITH AN IMLAC DISPLAY.

```
   THIS IS THE SET OF CONTROL FUNCTIONS, DRAW, PRAG ETC.
  DRAW TAKES THE SENTENCE AS INPUT... LATER PRAG, PREPRAG
  MAKE SEMANTIC NETWORK TO REPRESENT THE MEANING OF THE
  SENTENCE AND CALL THE DEEP SEMANTIC FUNCTIONS  TO DRAW IT ↓
  (IS(LAMBDA(ST) ST))
  ↓A DUMMY FUNCTION FOR THE VERB ≡IS↓
  (GETOK(LAMBDA (ST PROP)(GET(GET ST ≡TOK)PROP)   ))
  ↓AN INDIRECT GET FUNCTION↓

  (DRAW(LAMBDA(SNTC)(PROG(TOKS VTOKS J FOC B VB)
     (COND((NULL(SETQ J(PARSE SNTC)))(RETURN NIL) ))
     (SEPTOKS)
     (PREPRAG VTOKS)
          (SHOW TOKS)
  (PRAG VB ≡INIT)(DRAWPIX TOKS) (PRINT ≡(NEXT OR DONE≥))
     (PRAG VB ≡INTER) (PRINT ≡(NEXT OR DONE≥))
     (COND((EQ(SETQ J (READ)) ≡NEXT)(DRAWPIX TOKS))
         ((EQ J ≡DONE)(RETURN ≡ALLDONE)) )
    (PRAG VB ≡RESULT) (PRINT ≡(NEXT OR DONE≥)).
     (COND ((EQ (SETQ J(READ)) ≡NEXT)(DRAWPIX TOKS))
           ((EQ J ≡DONE)(RETURN ≡ALLDONE)) )
  )))

  (DRAWPIX(LAMBDA(TOKS)(PROG()
  (LINIT)
  (MAKEAR)
  (COMPICS)
  )))
  ↓SEPARATE TOKS INTO VTOKS AND NTOKS:TOKS↓
  (SEPTOKS(LAMBDA()(PROG(TKS)
     (SETQ TKS TOKS)
     (SETQ TOKS NIL)
     A (COND((GET(CAR TKS) ≡TENSE)(SETQ VTOKS(CONS (CAR TKS)
                                           VTOKS)) )
           ((GETOK(CAR TKS)≡PICT)(SETQ TOKS (CONS (CAR TKS)TOKS)) ))
     (COND((SETQ TKS(CDR TKS))(GO A))
           (T(RETURN NIL)) )
  )))
  ↓APPLIES PRAG TO EMBEDDED CLAUSES USING LIST VTOKS↓
  (PREPRAG(LAMBDA(LST)(PROG(TENSE)
     (SETQ LST(EFFACE VB LST))
     (COND((NULL LST)(RETURN NIL) ))
  B  (SETQ TENSE(GET(CAR LST) ≡TENSE))
  (COND((EQ TENSE ≡PRES)(PRAG(CAR LST) ≡INTER))
       ((EQ TENSE ≡PAST)(PRAG(CAR LST) ≡RESULT)) )
  (COND((SETQ LST(CDR LST))(GO B) ))
```

```
)))
↓APPLIES THE PROCESS MODEL OF A VERB TO ITS ARGUMENTS
 PROP IS INIT, INTER, RESULT   ↓
(PRAG(LAMBDA(ST PROP)(PROG (G L)
   (COND((NULL (SETQ G(GET ST PROP)))(RETURN NIL)) )
A  (EVAL (CAR G))
   (COND((SETQ G(CDR G))(GO A) ))
)))


↓ATN GRAMMAR STARTS HERE, MAIN NODES ARE NP,PP,
VG,VP,DCLAUSE AND CLAUSE WHICH IS THE TOP↓
(DEFINE* ≡(
(NP(CAT ART T (SETR DET(GETF DET))(TO NP1))
(CAT PRON T(SETR HD(ANTEC * GLST))(TO NP2))
(TST OK T (SETR DET ≡INDEF)(HOP NP1)) )
(NP1(CAT ADJ T (SETR MOD (APPEND(GETR MOD)(LIST *)))
(TO NP1))
      (CAT N T(SETR HD (MAKETOK *))(SETQ GLST(CONS * GLST))
       (PUT(GETR HD) ≡DET (GETR DET))
         (PUT(GETR HD) ≡MOD (GETR MOD))
        (TO NP2))
     (CAT PPRON T(TO NP1) ))
(NP2(POP (GETR HD)(PUTMODS (GETR HD)) ))

(PP(CAT PREP(NOT(GET(NEXT) ≡V))
       (SETR PREP (APPEND(GETR PREP)(LIST *)))
      (TO PP) )
     (PUSH NP(GETR PREP)(PUT * ≡PREP(GETR PREP))(HOP PP1)))
(PP1(TST TPPV(GETR VCONTROL)
      (SETR VCONTROL NIL)
       (SETR HD *)(HOP PP2) )
    (TST TPP1(SETR J(PREPMATCH * (CDR GLST)))
       (SETR HD *)(TO PP2) ))
(PP2(POP(GETR HD)(OR(SETR PREP NIL) T) ))

(VG(CAT AUX SNTC(SETR AUX(APPEND(GETR AUX)(LIST *)))
       (LIFTR AUX(GETR AUX))(TO VG))
   (CAT ADVB SNTC(SETR VMOD(APPEND(GETR VMOD)(LIST *)))
      (TO VG))
      (CAT V T (HOP VV1))
     (TST VAUX (GETR AUX)
        (SETR V (LAST(GETR AUX)))
      (SETR HD(MAKETOK(GETR V)))
      (SETQ GLST (CONS (GETR V) GLST))
      (HOP VG1) ))
   (VV1(TST VV T(SETR V *)(SETR HD (MAKETOK *))
        (SETQ GLST(CONS * GLST))(TO VG1) ))
   (VG1(CAT ADVB T(SETR VMOD(APPEND(GETR VMOD)(LIST *)))
        (TO VG1))
         (TST OK(OR(AND(SETQ J (GETR AUX))
            (SETR TENSE (GET (CAR J) ≡TENSE)) )
                 (SETR TENSE(GET(GETR V ) ≡TENSE)) T)
        (PUT(GETR HD) ≡TENSE (GETR TENSE))
         (PUT(GETR HD) ≡AUX (GETR AUX))
          (LIFTR AUX(GETR AUX))
          (PUT(GETR HD) ≡VMOD (GETR VMOD))
        (HOP VG2) ))
    (VG2(POP(GETR HD) T))
  (VP(PUSH VG SNTC (SETR V *)(TO VP1)) )
```

```
(VP1(PUSH NP SNTC(SETR OBJ *)(PUT(GETR V) ≡SUBJ (GETR SUBJ))
       (PUT(GETR V) ≡OBJ (GETR OBJ))(TO VP2))
  (TST TVP1(AND(OR(GETR DCL)(GET(LAST(GETR AUX)) ≡BE))
              (GET(GET(GETR V) ≡TOK)≡ED))
          (SETR PASV T)(SETR OBJ (GETR SUBJ))
          (SETR SUBJ NIL)(HOP VP2))
    (TST TVP2 T(HOP VP2) ))
(VP2(PUSH PP(AND(GETR PASV)(GET * ≡BY)(SENDR VCONTROL T))
        (SETR SUBJ *)(PUT (GETR V) ≡OBJ (GETR OBJ))
        (PUT(GETR V) ≡SUBJ (GETR SUBJ))(TO VP2))
    (PUSH DCLAUSE(SENDR DCL T)(TO VP2))
   (POP(GETR V) (PUT(GETR V) ≡SUBJ (GETR SUBJ)) ))

(DCLAUSE(PUSH PP (GET * ≡PREP)(SETR HD *)(TO D4))
       (PUSH RELCONJ (CAT ≡RCONJ)(SETR HD *)(TO D4))
       (CAT RPRON T(SETR SUBJ (ANTEC * GLST))
              (SENDR SUBJ (GETR SUBJ))(TO D1))
     (CAT V(OR(GETF ED)(GETF ING))(HOP D2))
    (CAT PREP(AND (GETF TO)(GET(NEXT) ≡V)(SCANXT))(HOP D2)))
(D2(TST D2 T(SETR SUBJ(VBMATCH * GLST))
        (SENDR SUBJ(GETR SUBJ))(HOP D21) ))
(D21(PUSH VP(SENDR DCL T)(SETR HD *)(HOP D3) ))
(D1(PUSH PRONCLAUSE T(HOP D3) ))
(D3(TST TD3(GETR SUBJ)(SETR HD *)
         ((GET * ≡TOK) *)↓SEMANTICS OF VERB↓
         (PUT(GETR SUBJ) ≡SMOD (CONS *(GET(GETR SUBJ) ≡SMOD)))
       (TO D4) )
   (TST TD31 T(SETQ HOLD(CONS * HOLD))(TO D4) ))
(D4(POP(GETR HD) T))

(PRONCLAUSE(PUSH VP (SENDR DCL T)(SETR HD *)
                 (PUT * ≡SUBJ(GETR SUBJ))(TO PR3) )
       (PUSH NP T(SETR OBJ (GETR SUBJ))(SETR SUBJ *)
                 (TO PR1) ))
(PR1(PUSH DCLAUSE T(SENDR DCL T)(SETR HD *)
          (SETR DCL T)(TO PR1) )
    (PUSH VG T(SETR HD *)(PUT * ≡SUBJ (GETR SUBJ))
                (PUT * ≡OBJ (GETR OBJ))(TO PR2) ))
(PR2(PUSH DCLAUSE T (TO PR2) )
    (TST TPR2 (TO PR3) ))
(PR3(POP(GETR HD) T))

(CLAUSE(PUSH NP SNTC (SETR SUBJ *)
              (SETR HD *)(TO C1) )
      (PUSH DCLAUSE SNTC (SETR HD *)(TO CLAUSE))
      (TST CL1(GETR HD)(HOP C1) ))
(C1(TST CT1 (UNHOLD)(HOP C1) )
   (PUSH DCLAUSE SNTC (TO C1) )
   (PUSH VP (SENDR SUBJ (GETR SUBJ))(SETR HD *)
              ((GET * ≡TOK) *)(TO C2) )
   (POP (GETR HD)(NULL SNTC) ))
(C2(POP(GETR HD) T))

(RELCONJ(CAT RCONJ SNTC (SETR RCONJ *)(TO R1) ))
(R1(PUSH DCLAUSE T(SETR HD *)(HOP R2) )
   (PUSH CLAUSE SNTC (SETR HD *)(HOP R2) ))
(R2(POP(GETR HD)(PUT * ≡RCONJ (GETR RCONJ)) ))

) ≡GR1)
(REMOB ≡TO)

↓A SHORT LEXICON IN THE FORM OF A PROPERTY LIST
```

STRUCTURE EXCEPT THAT (GET ≡THE ≡ART) RETURNS ≡THE↓

```
(LEXICON ≡(
(THE ART (DET DEF)(NBR (SING PL)))
(A ART (DET INDEF)(NBR (SING)))
(AN ART (DET INDEF)(NBR (SING)))
(BIG ADJ (SIZE 7)(CLASS SIZE))
(LARGE ADJ (SIZE 6)(CLASS SIZE))
(LITTLE ADJ (SIZE 3)(CLASS SIZE))
(SMALL ADJ (SIZE 4)(CLASS SIZE))
(IS AUX (TENSE PRES)(BE T))
(WAS AUX (TENSE PAST)(BE T))
(WERE AUX (TENSE PAST)(BE T)(NBR (PL)))
(BY PREP (BY T))
(BESIDE PREP (CANON RIGHTOF))
(TO PREP (TO T))
(IN PREP (CANON ON))
(ON PREP (CANON ON))
(WITH PREP (CANON ON))
(FROM PREP (CANON FROM))
(THROUGH PREP (MEDIUM T))
(ACROSS PREP (MEDIUM T))
(CLOWN N (NBR SING)(HANDS T)(HEAD T)(FEET T)(PERS T)
      (PICT T)(ANIM T)(ARM T)(ARMS T))
(PEDESTAL N (NBR SING)(TOP T) (BASE T)(PICT T))
(TO PREP (G T))
(FROM PREP (S T))
(HEAD N (NBR SING)(PART T))
(NOSE N (NBR SING)(PART T))
(POLE N (PICT T) (NBR SING))
(HEAD N (NBR SING)(PART T))
(FEET N (NBR PL)(PART T))
(ARM N (NBR SING)(PART T))
(ARMS N (NBR PL)(PART T))
(TOP N (NBR SING)(PART T)(NREL T))
(BASE N (NBR SING)(PART T)(NREL T))
(SIDE N (NBR SING)(PART T)(NREL T))
(BALANCE V (TENSE PRES)(INF T)(PREPS (ON WITH IN)) )
(BALANCES V (TENSE PRES)(PREPS (ON WITH IN)) )
(BALANCING V (TENSE PRES)(ING T)(PREPS (ON WITH IN)) )
(BALANCED V (TENSE PAST)(ED T)(PREPS (ON WITH IN)) )
(SUPPORT V (TENSE PRES)(INF T)(PREPS (ON WITH)) )
(SUPPORTS V (TENSE PRES)(PREPS (ON WITH)) )
(SUPPORTED V (TENSE PAST)(ED T)(PREPS (ON WITH)) )
(SUPPORTING V (TENSE PRES)(ING T)(PREPS (ON WITH)) )
(SAILING V (TENSE PRES)(ING T)(PREPS (TO FROM THROUGH ACROSS IN)) )
(SAILED V (TENSE PAST)(ED T)(PREPS (TO FROM THROUGH ACROSS IN)) )
(SAILS V (TENSE PRES)(PREPS (TO FROM THROUGH ACROS IN ACROSS)) )
(SAIL V (TENSE PRES)(PREPS (TO FROM THROUGH ACROSS IN)) )
(HOLD V (TENSE PRES)(INF T)(PREPS (IN WITH)))
(HOLDS V (TENSE PRES)(PREPS (IN WITH)))
(HELD V (TENSE PAST)(ED T)(PREPS (IN WITH)))
(HOLDING V (TENSE PRES)(ING T)(PREPS (IN WITH)))
(WHICH RPRON (SING T)(NBR (SING PL)))
(THAT RPRON (NBR (SING PL)) )
(WHO RPRON (NBR (SING PL))(PERT T))
(IT PRON (NBR (SING)) )
(ITS PPRON (NBR (SING )))
(HIS PPRON (NBR (SING))(PERS T))
(HER PPRON (NBR (SING))(PERS T))
(HE PRON (NBR (SING))(PERS T))
(SHE PRON (NBR (SING))(PERS T))
```

```
(THEY PRON (NBR (PL))(PERS X))
(WHILE RCONJ (TIME SAME))
(BEFORE RCONJ (TIME FIRST))
(POINT N (LOC T)(PICT T))
(WIND N (NBR SING)(PICT T)(FORCE T))
(BOAT N (NBR SING)(PICT T)(VEHIC T)(DMEDIUM WATER))
(DOCK N (NBR SING)(PICT T))
(LIGHTHOUSE N (NBR SING)(PICT T))
(WATER N (NBR SING)(PICT T)(MEDIUM T)(DVEHIC BOAT))
(AFTER RCONJ (TIME LATER))
)))))
(SETQ TOKS NIL)
(SETQ VB NIL)(SETQ FOC NIL)


(DEFINE ≡(
↓THE FOLLOWING FUNCTIONS DEFINE THE PICTURES USED BY
THE SYSTEM ↓
↓
DRAW A PEDESTAL     ↓
(PEDESTAL (LAMBDA (SIZE) (PROG ()
  (PUSHSCALE SIZE)
  (PENDOWN)
  (VECT 20 20) (RIGHT 90) (FORW 30) (VECT 20 20)
  (BACK 70) (LEFT 90)
  (PENUP)
  (POPSCALE)      )))
↓DRAW A POLE↓
(POLE(LAMBDA (SIZE)(PROG ()
     (PUSHSCALE SIZE)
     (PENDOWN)(FORW 10)(BACK 5)(RIGHT 90)(FORW 60)
      (LEFT 90)(FORW 5)(BACK 10)(PENUP)
     (POPSCALE)   )))
↓
INITIALIZE STUFF FOR CLOWN AND PEDESTAL.   ↓
(CLOWNINIT (LAMBDA () (PROG ()
(PUTRELS ≡(POINT STARTPT (0 0) STARTORIENT 0
 PFRAME (0 0 0 0) PSCALE 1 DRAWPROG POINT CG 0 ))
  (PUTRELS ≡(CLOWN STARTPT (14 0) STARTORIENT 0
PFRAME (0 50 0 68) PSCALE 1 BOTTOM (25 0) TOP (25 68)
     RFOOT (14 0) LFOOT (36 0) FEET (25 0)
    DRAWPROG CLOWN
     RARM (0 34) LARM (50 34) HEAD (25 68)
  HANDS (25 34) ARM (0 34) ARMS (25 34)
     CG (25 36)   ))
     (PUTRELS ≡(POLE STARTPT (0 0) STARTORIENT 0
PFRAME (0 60 0 10) PSCALE 1 BOTTOM (30 4.5) TOP (30 5.5)
        BASE (30 4.5) CG (30 5) TIP (0 5) DRAWPROG POLE))
  (PUTRELS ≡(PEDESTAL STARTPT (0 0) STARTORIENT 0
  PFRAME (0 70 0 20) PSCALE 1 BOTTOM (0 35) TOP (35 20)
     BASE (35 0)     CG (35 8)  DRAWPROG PEDESTAL))
 (PUTRELS ≡(BOAT STARTPT (0 0) STARTORIENT 0
  PFRAME (0 150 0 20) PSCALE 1 DRAWPROG BOAT
 BOTTOM (75 0) TOP (75 20) LEFTSIDE (0 10) RIGHTSIDE (150 10)
 CG (75 10) ))

 (PUTRELS ≡(WATER STARTPT (0 0) STARTORIENT 0
 PFRAME (0 500 0 5) PSCALE 1 DRAWPROG WATER
 BOTTOM (250 0) TOP (250 5) LEFTSIDE (0 3) RIGHTSIDE (500 3)
 CG (250 4) ))

 (PUTRELS ≡(DOCK STARTPT (0 0) STARTORIENT 0
```

```
 PFRAME (0 100 0 30) PSCALE 1 DRAWPROG DOCK
 BOTTOM (50 0) TOP (50 30) LEFTSIDE (0 15) RIGHTSIDE (100 15)
 CG (50 25) ))

(PUTRELS =(LIGHTHOUSE STARTPT (0 0) STARTORIENT 0
PFRAME (0 100 0 350) PSCALE 1 DRAWPROG LIGHTHOUSE
BOTTOM (50 0) TOP (50 350) LEFTSIDE (0 175) RIGHTSIDE (100 175)
 CG (50 175) ))
   )))
↓
A PITIFUL EXCUSE FOR A CLOWN, MORE LIKE THE TIN WOODMAN OF
THE WIZARD OF OZ.   ↓
 CLOWN (LAMBDA (SIZE) (PROG ()
  (PUSHSCALE SIZE)
  (PENDOWN)
  (RECT 2 8) (POS 2 4) (RECT 18 4) (POS 18 -2)
  (RECT 28 18) (POS 28 6) (RECT 4 6) (POS 4 -2)
  (RECT 8 10) (POS -52 8) (RECT 2 8) (POS 2 0)
  (RECT 18 4) (POS 58 -8) (LEFT 90) (FORW 1) (RIGHT 90)
  (VECT 8 6) (VECT -8 6) (LEFT 90) (FORW 1) (RIGHT 90)
  (POS -12 -14) (VECT -14 -14) (LEFT 90) (FORW 2)
  (LEFT 90) (FORW 2) (LEFT 90) (FORW 4) (LEFT 90)
  (VECT 12 12) (POS 0 18) (VECT -12 12) (RIGHT 90)
  (FORW 4) (LEFT 90) (FORW 2) (LEFT 90) (FORW 2) (RIGHT 90)
  (VECT 14 -14) (POS -48 -20)
  (PENUP)
  (POPSCALE)
  )))
 (WATER(LAMBDA(SIZE)(PROG()
 (PUSHSCALE SIZE)
 (PENDOWN)(VECT 10 125)(VECT -10 125)(VECT 10 125)(VECT -10 125)
 (PENUP) )))

 (POINT(LAMBDA (SIZE) NIL))
 (BOAT(LAMBDA(SIZE)(PROG()
 (PUSHSCALE SIZE) (FORW 20)
 (PENDOWN)(RIGHT 90)(FORW 150)(VECT -20 20)(RIGHT 180)
 (FORW 110)(VECT 20 20)(RIGHT 90)(PENUP)  )))

 (LIGHTHOUSE(LAMBDA(SIZE)
    (PROG()
 (PUSHSCALE SIZE)
 (PENDOWN)(RIGHT 90)(FORW 100)(VECT -20 -250)(LEFT 90)
 (FORW 50)(LEFT 90)(VECT 25 10)(VECT 25 -10)(LEFT 90)(FORW 50)
 (PENUP)(VECT 0 20)(PENDOWN)(VECT -50 -90)(PENUP)(VECT 25 0)
 (PENDOWN)(VECT 0 90)(PENUP)(VECT -50 0)(PENDOWN)(VECT 50 -90)
 (PENUP)(VECT 0 20)(PENDOWN)(VECT 0 50)
 (VECT 250 20)(RIGHT 180)(PENUP)  )))

 (DOCK(LAMBDA(SIZE)(PROG()  (PUSHSCALE SIZE)
 (PENDOWN)  (FORW 30)
 (RIGHT 90)(FORW 100)(RIGHT 90)(FORW 30)(RIGHT 90)(FORW 15)
 (RIGHT 90)(FORW 20)(LEFT 90)(FORW 55)(LEFT 90)(FORW 20)
 (RIGHT 180)(PENUP) )))

 )))))
 (GRAMMAR =GR1)
 (CLOWNINIT)
 (SETQ LEFTOF =LEFTOF)
 (SETQ RIGHTOF =RIGHTOF)

 ↓THE FOLLOWING FUNCTIONS DEFINE ENGLISH WORDS IN TERMS
```

OF CANONICAL FORMS WHICH ARE THEMSELVES FUNCTIONS ↓

```
(DEFINE ≡(
(RIGHTOF(LAMBDA(N1 N2)
   (COND((AND(GET N1 ≡PICT)(GET N2 ≡PICT))
        (PUT ST ≡RIGHTOF TOK1)(PUT TOK1 ≡LEFTOF ST))
        (T NIL) )
))
(FROM(LAMBDA(NI N2) NIL))
(TO (LAMBDA (N1 N2) NIL))
(LEFTOF(LAMBDA(N1 N2)(RIGHTOF N2 N1) ))
(BESIDE(LAMBDA(N1 N2)(RIGHTOF N1 N2) ))
(HOLDS(LAMBDA(ST)(HOLD ST) ))
(HELD(LAMBDA(ST)(HOLD ST) ))
(HOLDING(LAMBDA(ST)(HOLD ST) ))
(HOLD(LAMBDA(ST)
      (PROG()
   (PUT ST ≡SUPPORTPT1 ≡HANDS)
   (RETURN(SUPPORT1 ST))
)))
(BALANCE(LAMBDA(ST)(SUPPORT1 ST) ))
(BALANCES(LAMBDA(ST)(SUPPORT1 ST) ))
(BALANCED(LAMBDA(ST)(SUPPORT1 ST) ))
(BALANCING(LAMBDA (ST)(SUPPORT1 ST) ))
(SUPPORT(LAMBDA(ST)(SUPPORT1 ST) ))
(SUPPORTS(LAMBDA(ST)(SUPPORT1 ST) ))
(SUPPORTED(LAMBDA(ST)(SUPPORT1 ST) ))
(SUPPORTING(LAMBDA(ST)(SUPPORT1 ST) ))
(BALANCE(LAMBDA (ST)(SUPPORT1 ST) ))
(BALANCES(LAMBDA(ST)(SUPPORT1 ST) ))
(BALANCING(LAMBDA(ST)(SUPPORT1 ST) ))
↓ALWAYS RETURNS TRUE↓
(PUTMODS(LAMBDA(ST)(PROG(J)
   (COND((NULL(SETQ J(GET ST ≡MOD)))(RETURN ST) ))
B  (COND((NULL J)(RETURN ST)) )
   (SETQ K(GET(CAR J) ≡CLASS))
   (PUT ST K (GET(CAR J) K))
   (SETQ J(CDR J))(GO B)
)))
(LAST(LAMBDA (LST)(CAR(REVERSE LST)) ))


↓THIS IS A STATIC VERB THAT IS THE CANONICAL FORM FOR SUPPORT
BALANCE, HOLD ETC.↓

(SUPPORT1(LAMBDA(ST)(PROG(SUBJ OBJ PMOD VMOD TH1 TH2
   BALPT1 BALPT2 SUPPORTPT1 SUPPORTPT2 J TOK COMP)
   (VSET ST)
   (COND((AND SUBJ OBJ)(SETQ TH1 SUBJ)(SETQ TH2 OBJ))
        (SUBJ(SETQ TH2 SUBJ))
        (OBJ(SETQ TH2 OBJ)) )
V1  (COND((NULL VMOD)NIL)
        ((SETQ J(CAR VMOD))(PUT TH2
                 (GET J ≡CLASS)(GET J(GET J ≡CLASS)) )
              (SETQ VMOD (CDR VMOD))(GO V1) ))
P1  (COND((NULL PMOD)(GO DEFAULT)))
        (SETQ COMP(CAR PMOD))(SETQ PMOD(CDR PMOD))
          (COND((SETQ PREP(GET COMP ≡PREP))(SETQ PREP(CAR PREP)) ))
         (SETQ TOK(GET COMP ≡TOK))
   (COND((AND (NULL TH1)(GET TOK ≡PICT)(MEMBER PREP (LIST ≡ON ≡IN)))
            (SETQ TH1 COMP) )
       ((AND OBJ (NULL SUPPORTPT2)(MEMBER PREP(LIST ≡IN ≡ON ≡WITH))
```

```
          (SETQ SUPPORTPT2(GETOK TH2 TOK)) )
               (PUT TH2 =SUPPORTPT SUPPORTPT2) )
        ((AND (NULL OBJ)(NULL BALPT2)(MEMBER PREP(LIST =ON =WITH))
          (SETQ BALPT2(GETOK TH2 TOK)) )
               (PUT TH2 =BALPT BALPT2) )
        ((AND (NULL BALPT1)(MEMBER PREP(LIST =ON =BY))
          (SETQ BALPT1(GETOK TH1 TOK)) )
               (PUT TH1 =BALPT BALPT1) )
       (T(PRINT(CONS =(UNACCOUNTED PMOD:::)COMP)) ))
(GO P1)
DEFAULT
   (COND((AND TH1 TH2)(PUT TH1 =SUPPORT TH2)
                           (PUT TH2 =SUPPORTBY TH1) ))
   (COND((AND(NULL SUPPORTPT1)(SETQ SUPPORTPT1(GET ST =SUPPORTPT1)))
                (SETQ SUPPORTPT1(GETOK TH1 SUPPORTPT1))
                 (PUT TH1 =SUPPORTPT SUPPORTPT1) ))
*NOTE THIS IS A STATE VERB SO NO PROCESS MODEL IS CONSTRUCTED*
)))
(VSET(LAMBDA(ST)(PROG()
      (SETQ SUBJ(GET ST =SUBJ))
      (SETQ OBJ (GET ST =OBJ))
      (SETQ PMOD (GET ST =PMOD))
      (SETQ VMOD (GET ST =VMOD))

)))
(SAIL(LAMBDA(ST)(PROG()
(PUT ST =MEDIUM =WATER)(PUT ST =VEHIC =BOAT)
(RETURN(MOVE* ST)) )))

(SAILS(LAMBDA(ST)(SAIL ST) ))
(SAILED(LAMBDA(ST)(SAIL ST) ))
(SAILING(LAMBDA(ST)(SAIL ST) ))

* THIS IS THE CANONICAL VERB OF MOTION FOR THE SYSTEM *

(MOVE*(LAMBDA(ST)(PROG(SUBJ OBJ COMP COMPS A TH  PMOD
                     I VEHIC MEDIUM S G J)
*SET SUBJ OBJ COMPS WITH VSET *
(VSET ST)
(COND((GETOK SUBJ =FORCE)(SETQ I SUBJ))
     ((GETOK SUBJ =ANIM)(SETQ A SUBJ))
     ((GETOK SUBJ =VEHIC)(SETQ VEHIC SUBJ)) )
(COND((AND (NOT VEHIC)(GETOK OBJ =VEHIC))(SETQ VEHIC OBJ))
     ((GETOK OBJ =MEDIUM)(SETQ MEDIUM OBJ))
     (OBJ(SETQ TH OBJ)) )
*FOR EVERY COMPLEMENT*
P1 (COND((NULL PMOD)(GO DEFAULT) ))
  (SETQ COMP (CAR PMOD))
  (SETQ PREP (CAR(GET COMP =PREP)))
  (SETQ NHD (GET COMP =TOK))
 (COND((AND(NULL VEHIC)(MEMBER PREP (LIST =IN =ON))
          (GET NHD =VEHIC))(SETQ VEHIC COMP))
      ((AND(NULL MEDIUM)(MEMBER PREP(LIST =ON =THROUGH  =ACROSS
          =IN))(GET NHD =MEDIUM))(SETQ MEDIUM COMP))
      ((AND(NULL S)(MEMBER PREP(LIST =FROM =OUT =OFF))
            (GET NHD =PICT))(SETQ S COMP))
      ((AND(NULL G)(MEMBER PREP(LIST =TO =FOR))
            (GET NHD =PICT))(SETQ G COMP))
      (T(PRINT(LIST =UNDEFINED: COMP)) ))
 (COND((SETQ PMOD (CDR PMOD))(GO P1) ))
 DEFAULT
 *I LOOK ON NPS FOR VEHIC, MEDIUM, S , G,(NOT DONE YET)
```

```
 2 LOOK ON VB ST TO SEE IF VEHIC AND MEDIUM HAVE BEEN PASSED UP
 3 LOOK IN DICTIONARY FOR NORMAL MEDIUM OR VEHIC GIVEN ONE
 4 DEFAULT S AND G TO LEFT AND RIGHT SCREEN
 5 DEFAULTS TO TOP, BOTTOM , LEFT, AND RIGHTSIDE OCCUR IN
  MAKEAR AND COMPICS
↓
(COND((AND(NULL VEHIC)(SETQ J(GET ST ≡VEHIC)))
          (SETG VEHIC (MAKETOK J)) ))
(COND((AND(NULL MEDIUM)(SETQ J(GET ST ≡MEDIUM)))
       (SETQ MEDIUM (MAKETOK J)) ))
(COND((NULL S)(SETQ S(MAKETOK ≡POINT)) ))
(COND((NULL G)(SETQ G(MAKETOK ≡POINT)) ))
(COND((AND MEDIUM (NULL VEHIC)(SETQ J(GETOK MEDIUM ≡DVEHIC)))
       (SETG VEHIC (MAKETOK J)) ))
(COND((AND VEHIC (NULL MEDIUM)(SETQ J(GETOK VEHIC ≡DMEDIUM)))
       (SETG MEDIUM(MAKETOK J)) ))
↓PROCESS MODEL
     PUTS GLOBAL CONDITIONS ON SEMANTIC NET, PUTS INITIAL,
 INTERMEDIATE AND RESULT CONDITIONS ON ST WHERE PREPRAG AND PRAG
CAN BRING THEM ONTOTHE NET TO COMPOSE A PICTURE
↓
GLOBAL
(AND I (PUT S ≡SUPPORT I)(PUT I ≡SUPPORTBY S))
(AND A TH (PUT A ≡LEFTOF TH)(PUT TH ≡RIGHTOF T))
(AND A (PUT VEHIC ≡SUPPORT A)(PUT A ≡SUPPORTBY VEHIC))
(AND TH (NULL A)(PUT VEHIC ≡SUPPORT TH)(PUT TH ≡SUPPORTBY VEHIC))
(PUT VEHIC ≡ABOVE MEDIUM)
(PUT MEDIUM ≡BELOW VEHIC)
(PUT MEDIUM ≡LEFTOF G)(PUT G ≡RIGHTOF MEDIUM)
(PUT MEDIUM ≡RIGHTOF S)(PUT S ≡LEFTOF MEDIUM)
INIT
(PUT ST ≡INIT (LIST(LIST ≡REMPROP VEHIC ≡LEFTOF)
                        (LIST ≡PUT VEHIC ≡RIGHTOF S)))
INTER
(PUT ST ≡INTER(LIST(LIST ≡REMPROP VEHIC ≡RIGHTOF)))
RESULT
(PUT ST ≡RESULT(LIST(LIST ≡PUT VEHIC ≡LEFTOF G)))
(RETURN ST)
)))

↓HERE IS THE PROCRUSTEAN BED FOR PREPOSITIONS ↓

(IN(LAMBDA(N1 N2)(ON N1 N2) ))
(BY(LAMBDA (N1 N2)(LEFTOF N1 N2) ))

(WITH(LAMBDA(N1 N2)
    (COND((NULL(GET N1 ≡PICT))NIL)
         ((PUT TOK1 ≡SUPPORT ST)(PUT ST ≡SUPPORTBY TOK1))
         (T NIL))
))
(ON(LAMBDA(N1 N2)
   (COND((NOT(GET N1 ≡PICT))NIL)
     ((AND(SETQ J(GET N1 N2))(GET TOK1 ≡SUPPORT))
             (PUT TOK1 ≡SUPPORTPT J) )
     (J(PUT TOK1 ≡BALPT J))
        ((GET N2 ≡PICT)(PROG2(PUT ST ≡SUPPORT TOK1)
                             (PUT TOK1 ≡SUPPORTBY ST)) )
        (T NIL) )
))


 ↓THIS IS THE CONTROL FUNCTION THAT SETS UP CALLS TO
```

PREPOSITIONS AND THEIR ARGUMENTS TO DETERMINE WHICH WORD BEST QUALIFIES AS THE HEAD THAT IS TO BE MODIFIED ↓
↓(P CAND N) IS THE CALL WHICH EVALS P WITH THE ARGUMENTS CAND AND N. THIS MODE IS ALSO USED TO CALL A VERB IN THE GRAMMAR↓

```
(PREPMATCH(LAMBDA(ST LST)(PROG(N CAND P TOK1 TOK2)
   (COND((SETQ P(GET ST ≡PREP))(SETQ P(CAR P)) ))
   (SETQ N (GET ST ≡TOK))
A  (COND((NULL LST)(RETURN NIL) ))
   (SETQ CAND(CAR LST))
   (SETQ TOK1 (CAR(GET CAND ≡U/I)))
   B (COND((AND(GET CAND ≡V)(MEMBER P(GET CAND ≡PREPS)) )
          (RETURN(PUT TOK1 ≡PMOD(CONS ST(GET TOK1 ≡PMOD)))) )
      ((EQ ST TOK1)(SETQ TOK1(CADR(GET CAND ≡U/I)))(GO B) )
        ((P CAND N)(RETURN TOK1))↓DO SEMANTICS OF PREP↓
        (T(SETQ LST(CDR LST))(GO A) ))
)))
```

↓THIS USE OF HOLD AND UNHOLD IS APPROXIMATELY EQUIVALENT TO THE WOODS VERT ARC ↓

```
(UNHOLD(LAMBDA()(PROG(J)
A (COND((NULL HOLD)(RETURN NIL))
        ((GET(CAR HOLD) ≡PREP)(GO P1))
        ((NULL(GET(CAR HOLD) ≡SUBJ))(PUT (CAR HOLD) ≡SUBJ
                        (GETR ≡SUBJ))
↓DO VB SEMANTICS↓
             ((GET(CAR HOLD) ≡TOK)(CAR HOLD))(GO P2) )
       ((NULL(GET(CAR HOLD) ≡OBJ))(PUT (CAR HOLD) ≡OBJ
                        (GETR ≡SUBJ))
↓DO VB SEMANTICS↓
             ((GET(CAR HOLD) ≡TOK)(CAR HOLD))(GO P2) ))
P1  (COND((PREPMATCH(CAR HOLD)(LIST(GETR SUBJ)))T)
          ((PREPMATCH(CAR HOLD)GLST)T)
          (T NIL))
P2 (SETQ HOLD(CDR HOLD))(GO A)
)))

(PARSE(LAMBDA(SNTC)(PROG(*    NOSEF HD HOLD GLST LEV)
↓PARSE CALLS THE ATN BY SETTING LEV-EL TO ZERO AND
PUSHING TO CLAUSE↓
      (CLEARREGS REGLIST)
      (SETQ LEV 0)
      (SETQ * (CAR SNTC))
      (PRINT(PUSH ≡CLAUSE))
   (COND((GET * ≡DET)(SETQ FOC *)(SETQ VB ≡IS))
        ((SETQ FOC(GET * ≡SUBJ))(SETQ VB *))
        ((SETQ FOC(GET * ≡OBJ))(SETQ VB *)) )
          (RETURN (CAR TOKS))
)))

(SHOW(LAMBDA(TOKS)
     (COND((NULL TOKS)≡DONE)
          ((PRINT(CAR TOKS))(PRINT(PPROP(CAR TOKS)))
                            (SHOW  (CDR TOKS)) ))
))
(VBMATCH(LAMBDA(VB LST)

↓SELECTS A NOUN WHICH IS NOT A PART AS AN ARGUMENT
FOR A VERB↓
    (COND((NULL LST)NIL)
         ((GET (CAR LST) ≡PART)(VBMATCH VB (CDR LST)) )
         (T(CAR(GET (CAR LST) ≡U/I))) )
```

```
))
(MAKETOK(LAMBDA(WD)(PROG(J)
   (SETQ J(INTERN(GENSYM C)))
   (SETQ TOKS (CONS J TOKS))
(SET J J)
   (PUT WD ≡U/I(CONS J(GET WD ≡U/I)))
   (PUT J ≡NBR(GET WD ≡NBR))
   (RETURN (PUT J ≡TOK WD))
)))

↓FINDS ANTECEDENTS FOR PRONOUNS BY CHECKING PERSON
AND NUMBER...CLOWNS ARE SEXLESS ↓
(ANTEC(LAMBDA(PRON LST)(PROG(CAND)
   (COND((NULL LST)(RETURN NIL)) )
   (SETQ CAND (CAR LST))
   (COND((AND(EQ(GET PRON ≡PERS)(GET CAND ≡PERS))
      (MEMBER(GET CAND ≡NBR)(GET PRON ≡NBR)))
           (RETURN(CAR(GET CAND ≡U/I))))
         (T(RETURN(ANTEC PRON (CDR LST)))) )
)))

(SCANXT(LAMBDA()
    (COND((NULL SNTC)NIL)
          ((SETQ SNTC(CDR SNTC))(SETQ *(CAR SNTC)))
          (T NIL) )
))
(NEXT(LAMBDA()
   (COND(SNTC(CAR SNTC))
        (T NIL) )
))
(PPROP(LAMBDA(X)(OUTPSET(CSR X)) ))

↓LEXICON AND LEX1 FORM A PROPERTY LIST STRUCTURE FROM
THE FORM SHOWN IN (LEXICON (...))
(LEXICON(LAMBDA(L)
   (MAP L(FQUOTE(LAMBDA (L)(PROG2
    (PUT(CAAR L)(CADAR L)(CAAR L))
    (LEX1(CAAR L)(CDDAR L)) )))
)))
(LEX1(LAMBDA(W LP)
   (COND((NULL LP)T)
        ((PUT W (CAAR LP)(CADAR LP))
           (LEX1 W (CDR LP)) ))
))
)))))

↓  THIS FILE IS A SET OF LISP FUNCTIONS TO SIMULATE
SOME OF THE PRIMITIVE FUNCTIONS OF THE M.I.T. LOGO
LANGUAGE AND INTERFACE TO THE GESYS SOFTWARE FOR THE
IMLAC DISPLAY TERMINAL.

WRITTEN BY GORDON NOVAK ON 29 MAY 74.      ↓
↓
GLOBAL INITIALIZATION.  ENTER (LOGO) TO START THE
SYSTEM AND RETURN TO MAINLOOP      ↓
(LOGO (LAMBDA () (PROG (IPEN THETA STHETA CTHETA
     GLOBALSIZE SSCALE CSCALE XTOTAL YTOTAL
      THETAI SCREENXMAX SCREENYMAX CSIZE PSIZE  ITEMAD
      IXTOTAL IYTOTAL PI180
      UNIVOD PICSCL *TRACE* MASSSCL
     )
  (LINIT)
```

```
  (MAINLOOP)   )))
↓
INITIALIZATION.  ENTER (LINIT) TO RE-INITIALIZE AND
START A NEW PICTURE.    ↓
(LINIT (LAMBDA () (PROG ()
  (SETQ GLOBALSIZE 1.0)
  (CSETQ PI180 (QUOTIENT 3.1415926535898 180.0))
  (SETQ CSIZE 1.0)
  (SETQ PSIZE NIL)
  (SETQ SCREENXMAX 1023.0)
  (SETQ SCREENYMAX 1023.0)
  (TEREAD)
  (PENUP)
  (HEADING 0.0)
  (RETURN)       )))
↓
NEWFRAME SENDS COMMANDS TO THE IMLAC TO ERASE THE SCREEN
AND RECREATES THE FRAME =LOGO.          ↓
(NEWFRAME (LAMBDA () (PROG ()
(GOUT =ER NIL)
(SETQ ITADDER 2048)

      )))
↓
E IS A SHORT FORM OF ERASE TO ERASE THE SCREEN.    ↓
(E (LAMBDA () (ERASE)))
↓
ERASE THE SCREEN AND ALL TOKENS IN UNIVERSE OF DISCOURSE ↓
(ERASE (LAMBDA () (PROG (ATM)
A (COND ((NULL UNIVOD) (GO B)))
  (SETQ ATM (CAR UNIVOD))
  (PUT ATM =IMLACITEM NIL)
  (DELITEM ATM)
  (GO A)
B (NEWFRAME)
  (RETURN)
)))
↓
LIST PROPERTY LIST RELATIONS OF AN ATOM, EXCEPT PNAME,
INFO, AND EXPR.    ↓
(LISTREL (LAMBDA (ATM) (PROG (X Y)
 (PRINT BLANK)
 (PRINT ATM)
 (SETQ X (CSR ATM))
A (COND ((NULL X) (RETURN)))
  (SETQ Y (CSR X))
  (COND ((OR (EQ Y =PNAME) (EQ Y =INFO) (EQ Y =EXPR))
         (GO B)))
  (PRIN1 BLANK) (PRIN1 BLANK) (PRIN1 Y) (PRIN1 COLON)
  (PRIN1 BLANK)
  (PRINT (CAR X))
B (SETQ X (CDR X))
  (GO A)
)))
↓
TURN TURTLE HEADING TO THE RIGHT.   ↓
(RIGHT (LAMBDA (N) (HEADING (PLUS N THETA))))
↓
TURN TURTLE HEADING TO THE LEFT.    ↓
(LEFT (LAMBDA (N) (HEADING (DIFFERENCE THETA N))))
↓
ESTABLISH TURTLE HEADING.  ARGUMENT IS HEADING IN
```

```
DEGREES CLOCKWISE FROM NORTH.      ↓
(HEADING (LAMBDA (TH) (PROG ()
  (COND ((OR (GREATERP TH 3600.0) (LESSP TH -3600.0))
      (ERROR =(ARG OF HEADING TOO BIG))))
  (SETQ THETA TH)
A (COND ((GREATERP THETA -0.00000001) (GO B)))
  (SETQ THETA (PLUS THETA 360.0))
  (GO A)
B (COND ((LESSP THETA 360.0) (GO C)))
  (SETQ THETA (DIFFERENCE THETA 360.0))
  (GO B)
C (SETQ THETAI (TIMES (DIFFERENCE 90.0 THETA) PI180))
  (SETQ STHETA (SIN THETAI))
  (SETQ CTHETA (COS THETAI))
  (SETQ SSCALE (TIMES STHETA CSIZE))
  (SETQ CSCALE (TIMES CTHETA CSIZE))
  (RETURN)    )))
↓
PICK THE TURTLE*S PEN UP.    ↓
(PENUP (LAMBDA () (SETQ IPEN NIL)))
↓
PUT THE TURTLE*S PEN DOWN.    ↓
(PENDOWN (LAMBDA () (SETQ IPEN T)))
↓
MOVE THE TURTLE BACKWARDS     ↓
(BACK (LAMBDA (W) (FORW (MINUS W))))
↓
MOVE THE TURTLE BY A SIGNED AMOUNT    ↓
(MOVE (LAMBDA (W) (FORW W)))
↓
GENERATE OUTPUT COMMANDS TO THE IMLAC GIVEN THE COMMAND
WORD AND A LIST OF ARGUMENTS.  ↓
(GOUT (LAMBDA (COMMAND PLIST) (PROG ()
  (PRIN1 DARROW)
  (PRIN1 COMMAND)
A (COND ((NULL PLIST) (TERPRI) (RETURN)))
  (PRIN1 BLANK)
  (PRIN1 (CAR PLIST))
  (SETQ PLIST (CDR PLIST))
  (GO A)     )))
↓
FORW MOVES THE TURTLE BY A SIGNED AMOUNT IN THE CURRENT
DIRECTION.  IF THE PEN IS DOWN (IPEN = T), A VECTOR WILL
BE DRAWN.  ↓
(FORW (LAMBDA (W) (PROG (X Y IX IY XP YP)
  (SETQ X (TIMES CSCALE W))
  (SETQ Y (TIMES SSCALE W))
  (SETQ XP (PLUS XTOTAL X))
  (SETQ YP (PLUS YTOTAL Y))
  (COND ((OR (LESSP XP 0) (LESSP YP 0) (GREATERP XP SCREENXMAX)
            (GREATERP YP SCREENYMAX)) (ERROR =(MOVE WOULD GO OFF
                  SCREEN)) (RETURN)))
  (SETQ IX (IROUND (DIFFERENCE XP IXTOTAL)))
  (SETQ IY (IROUND (DIFFERENCE YP IYTOTAL)))
  (SETQ IXTOTAL (PLUS IX IXTOTAL))
  (SETQ IYTOTAL (PLUS IY IYTOTAL))
  (SETQ XTOTAL XP)
  (SETQ YTOTAL YP)
  (COND (IPEN (GOUT =LI (LIST IX IY)))
        (T (GOUT =MO (LIST IX IY))))
      )))
```

```
IROUND ROUNDS A NUMBER TO THE CLOSEST INTEGER.  ↓
(IROUND (LAMBDA (X)
  (COND ((MINUSP X) (FIX (DIFFERENCE X 0.5)))
        (T (FIX (PLUS X 0.5))) )  ))
↓
POSITION SETS THE CURRENT POSITION OF THE TURTLE TO A SPECIFIED
VECTOR POSITION, SUBJECT ONLY TO THE SIZE FACTOR GLOBALSIZE.
A VECTOR IS A LIST OF THE (X Y) COORDINATES.   ↓
(POSITION (LAMBDA (V) (PROG (IX IY)
  (SETQ IX (TIMES (CAR V) GLOBALSIZE))
  (SETQ IY (TIMES (CADR V) GLOBALSIZE))
  (COND ((OR (LESSP IX 0) (LESSP IY 0) (GREATERP IX SCREENXMAX)
             (GREATERP IY SCREENYMAX))
                (ERROR ≡(POSITION IS OFF
               SCREEN))(RETURN)))
  (SETQ XTOTAL IX)
  (SETQ YTOTAL IY)
  (SETQ IXTOTAL (IROUND IX))
  (SETQ IYTOTAL (IROUND IY))
  (GOUT ≡MT (LIST IXTOTAL IYTOTAL))      )))
↓
SCALE SETS A LOCAL SCALE (IN ADDITION TO GLOBALSIZE), AND MAY
BE USED TO SET THE SIZE OF SOMETHING TO BE DRAWN WITHOUT
MULTIPLYING EVERYTHING OUT.     ↓
(SCALE (LAMBDA (S) (PROG ()
  (SETQ CSIZE (TIMES S GLOBALSIZE))
  (SETQ SSCALE (TIMES STHETA CSIZE))
  (SETQ CSCALE (TIMES CTHETA CSIZE))
)))
↓
PUSHSCALE PUSHES DOWN THE CURRENT SCALE AND SETS THE CURRENT
SCALE FACTOR TO THE SPECIFIED VALUE.  THE COMPLEMENTARY
ROUTINE POPSCALE WILL RESTORE THE SCALE TO THE PREVIOUS
VALUE.    ↓
(PUSHSCALE (LAMBDA (S) (PROG ()
  (SETQ PSIZE (CONS CSIZE PSIZE))  (SCALE S)  )))
↓
POPSCALE WILL RESTORE THE SCALE FACTOR TO THE PREVIOUS
VALUE SAVED BY PUSHSCALE.    ↓
(POPSCALE (LAMBDA () (PROG ()
  (COND ((NULL PSIZE) (SCALE 1.0) (RETURN)))
  (SETQ CSIZE (CAR PSIZE))
  (SETQ SSCALE (TIMES STHETA CSIZE))
  (SETQ CSCALE (TIMES CTHETA CSIZE))
  (SETQ PSIZE (CDR PSIZE))      )))
↓
A VECTOR, FOR PURPOSES OF THE FOLLOWING VECTOR ROUTINES,
IS A LIST OF TWO VALUES, THE X AND Y COORDINATES.

VSUM FORMS THE SUM OF TWO VECTORS AS AN OUTPUT VECTOR.  ↓
(VSUM (LAMBDA (V1 V2) (LIST (PLUS (CAR V1) (CAR V2))
        (PLUS (CADR V1) (CADR V2)) ) ))
↓
VDIFF FORMS THE DIFFERENCE OF TWO VECTORS   ↓
(VDIFF (LAMBDA (V1 V2) (LIST (DIFFERENCE (CAR V1) (CAR V2))
           (DIFFERENCE (CADR V1) (CADR V2)) ) ))
↓
VSCALE SCALES A VECTOR BY A SCALAR.    ↓
(VSCALE (LAMBDA (V S) (LIST (TIMES (CAR V) S)
           (TIMES (CADR V) S)) ))
↓
VROT ROTATES A VECTOR BY A GIVEN ANGLE (IN DEGREES)    ↓
```

```
(VROT (LAMBDA (V TH) (PROG (STH CTH)
  (COND ((LESSP (ABS TH) 0.0001) (RETURN V)))
  (SETQ STH (SIN (TIMES TH PI180)))
  (SETQ CTH (COS (TIMES TH PI180)))
  (RETURN (LIST
     (DIFFERENCE (TIMES (CAR V) CTH) (TIMES (CADR V) STH))
     (PLUS (TIMES (CAR V) STH) (TIMES (CADR V) CTH))))
 )))
↓
VMAG RETURNS THE MAGNITUDE OF A VECTOR.     ↓
(VMAG (LAMBDA (V) (SQRT (FLOAT (PLUS
          (TIMES (CAR V) (CAR V))
          (TIMES (CADR V) (CADR V))  ))) ))
↓
VANG RETURNS THE ANGLE IN DEGREES OF A GIVEN VECTOR. ↓
(VANG (LAMBDA (V) (QUOTIENT (ATAN2
     (FLOAT (CADR V)) (FLOAT (CAR V)) ) PI180)))
↓
RECT DRAWS A RECTANGLE FROM THE CURRENT ORIENTATION AND
POSITION.  THE ARGUMENTS ARE THE NUMBER OF UNITS FORWARD
AND THE NUMBER OF UNITS TO THE RIGHT TO BE MOVED IN MAKING
THE RECTANGLE.
(RECT (LAMBDA (FW RT) (PROG ()
(FORW FW) (RIGHT 90) (FORW RT) (RIGHT 90) (FORW FW)
(RIGHT 90) (FORW RT) (RIGHT 90)  )))
↓
POS POSITIONS THE TURTLE RELATIVE TO ITS PRESENT POSITION
WITHOUT DRAWING A LINE.  THE ARGUMENTS ARE THE NUMBER OF
UNITS TO MOVE FORWARD AND THE NUMBER OF UNITS TO MOVE TO THE RIGHT.
THE ORIENTATION IS LEFT AS BEFORE THE CALL.   ↓
(POS (LAMBDA (FW RT) (PROG (SPEN)
  (SETQ SPEN IPEN) (PENUP)
  (SETQ ANG (VANG (LIST FW RT)))
  (RIGHT ANG)
  (FORW (VMAG (LIST FW RT)))
  (LEFT ANG)
  (SETQ IPEN SPEN)  )))
↓
VECT DRAWS A VECTOR WHICH WILL GO FROM THE CURRENT POSITION
AND ORIENTATION BY A SPECIFIED AMOUNT FORWARD AND A SPECIFIED
AMOUNT TO THE RIGHT.  THIS IS NEEDED BECAUSE IT IS USUALLY
THE CASE THAT EITHER THE LENGTH OR THE ANGLE IS A NASTY
NUMBER.  ↓
(VECT (LAMBDA (FW RT) (PROG (ANG)
  (SETQ ANG (VANG (LIST FW RT)))
  (RIGHT ANG)
  (FORW (VMAG (LIST FW RT)))
  (LEFT ANG)   )))
↓
DISPLAY CLOSES THE CURRENT ITEM AND ADDS IT TO THE CURRENT
FRAME SO IT WILL BE DISPLAYED.    ↓
(DISPLAY (LAMBDA () (PROG ()
↓                 SEND A COMMAND TO THE IMLAC TO CLOSE THE CURRENT
                   ITEM.    ↓
  (GOUT =CL NIL)
  (RETURN)   )))
↓
DELITEM DELETES AN ITEM BOTH FROM THE LISP DATA STRUCTURE AND
FROM THE DISPLAY.   ↓
(DELITEM (LAMBDA (TOKNAME) (PROG ()
(COND ((GET TOKNAME =IMLACITEM)
  (PRIN1 =DI) (PRIN1 BLANK) (PRIN1 EQUIV) (PRIN1 TOKNAME)
```

```
    (TERPRI) ))
    (DELREL (GET TOKNAME ≡TOK) ≡TOKENS TOKNAME)
    (SETQ UNIVOD (REMLIST TOKNAME UNIVOD))
    (REMOB TOKNAME)
    (RETURN)   )))
↓
DELREL REMOVES AN ENTRY UNDER A PROPERTY LIST INDICATOR
WHOSE LEFTMOST ATOM IS AS SPECIFIED.     ↓
(DELREL (LAMBDA (ATOM REL VALUE) (PROG (PROPS )
  (COND ((NULL (SETQ PROPS (GET ATOM REL))) (RETURN))).
  (PUT ATOM REL (REMLIST VALUE PROPS))
  (RETURN)
)))
↓
REMOVE A SPECIFIED ITEM FROM THE TOP LEVEL OF A LIST   ↓
EMLIST (LAMBDA (VAL LST) (PROG (TMP)
(COND ((NULL LST) (RETURN))
      ((EQ (CAR LST) VAL) (RETURN (CDR LST)) ))
(SETQ TMP LST)
(COND ((NULL (CDR LST)) (RETURN TMP))
      ((EQ (CADR LST) VAL) (RPLACD LST (CDDR LST))
       (RETURN TMP)))
(SETQ LST (CDR LST))
         (GO A)
)))
↓
CARATOM KEEPS TAKING THE CAR OF THE INPUT UNTIL IT FINDS AN
ATOM.   ↓
(CARATOM (LAMBDA (X) (COND ((ATOM X) X)
                           (T (CARATOM (CAR X)))))))
↓
ABSVAL RETURNS THE ABSOLUTE POSITION OF A POINT IN RELATIVE
COORDINATES ON AN OBJECT WHICH HAS BEEN POSITIONED BY
ORIENT1.     ↓
(ABSVAL (LAMBDA (OBJECT RELPT) (PROG (MODEL VDEL ROT)
  (SETQ MODEL (GET OBJECT ≡TOK))
  (SETQ VDEL (VSCALE (VDIFF RELPT (GET MODEL ≡STARTPT))
                 (GET OBJECT ≡SIZE)) ).
  (COND ((SETQ ROT (GET OBJECT ≡ORIENTATION))
         (SETQ VDEL (VROT VDEL ROT)) ))
  (RETURN (VSUM (GET OBJECT ≡STVAL) VDEL))
)))
↓
PUTRELS PUTS A STRING OF THINGS ON AN ATOM≠S PROPERTY LIST.
THE ARGUMENT IS A LIST OF THE ATOM, FOLLOWED BY INDICATOR
AND VALUE PAIRS.    ↓
(PUTRELS (LAMBDA (L) (PROG (ATOM REL VALUE)
  (SETQ ATOM (CAR L))
A (COND ((NULL (SETQ L (CDR L))) (RETURN)))
  (SETQ REL (CAR L))
  (COND ((NULL (SETQ L (CDR L))) (RETURN)))
  (SETQ VALUE (CAR L))
  (PUT ATOM REL VALUE)
  (GO A)    )))
↓       ABSOLUTE VALUE     ↓
(ABS (LAMBDA (X) (COND ((MINUSP X) (MINUS X)) (T X))))
)))))


↓DISKOUT IS THE FUNCTION BY MABRY TYSON THAT GIVES US
VIRTUAL MEMORY FOR FUNCTIONS.
```

```
THE FUNCTIONS IN THIS SECTION ARE VERY DIFFICULT TO
UNDERSTAND IN DETAIL BUT ESSENTIALLY THEY COMBINE PICTURE
ELEMENTS TO FORM AND SCALE A COMPLETE PICTURE ↓

(DISKOUT 4000 7000 NIL)
(DEFINE ≡(

(DEFINE(LAMBDA(X)
 (DISKOUT //CODEMIN //CODEMAX ((GET ≡DEFINE ≡SUBR) X)) ))
    ))
(DEFINE ≡(
↓
SETREL ADDS THE ITEM -VALUE- TO THE PROPERTY LIST RELATION
-REL- OF THE ATOM -ATOM-.  IF -VALUE- IS ALREADY THERE,
IT IS NOT ADDED AGAIN.     ↓
(SETREL (LAMBDA (ATOM REL VALUE) (PROG (X)
  (COND ((NULL (SETQ X (GET ATOM REL)))
           (RETURN (PUT ATOM REL (LIST VALUE)))))
A (COND ((EQUAL (CAR X) VALUE) (RETURN))
        ((NULL (CDR X)) (RETURN (RPLACD X (LIST VALUE)))))
  (SETQ X (CDR X))
  (GO A)      )))
↓
MAKE ATTACHMENT RELATIONS FROM THE SEMANTIC NETWORK
20 JAN 75     ↓
(MAKEAR (LAMBDA () (PROG (TKS TK ST ROT)
  (SETQ TKS TOKS)
A (SETQ TK (CAR TKS))
  (COND ((SETQ ST (GET TK ≡SUPPORT))
           (DEFAULT TK ≡SUPPORTPT ≡TOP)
           (DEFAULT ST ≡BALPT ≡BOTTOM)
           (MAKEATT
          TK (LIST ≡GETLOC (LIST TK (GET TK ≡SUPPORTPT)))
          ST (LIST ≡GETLOC (LIST ST (GET ST ≡BALPT))) ) ))
  (COND ((OR (SETQ ROT (GET TK ≡ANGLE))
           (AND (GET TK ≡BALPT) (SETQ ROT (DIFFERENCE
              90.0 (VANG (VDIFF (GETLOCK TK ≡CG) (GET TK ≡BALPT)
              )))) (GREATERP (ABS ROT) 0.00001)))
          (PUT TK ≡ORIENTATION ROT)))
  (COND ((SETQ TKS (CDR TKS)) (GO A)))
)))
↓IF ST HAS PROP1 RETURNS T ELSE PUTS PROP2 AS VALUE OF
 PROP1 FOR ST↓
(DEFAULT(LAMBDA(ST PROP1 PROP2)
     (COND((GET ST PROP1)T)
          (T(PUT ST PROP1 (GETOK ST PROP2))) )
))
↓GETOK PRONOUNCED GET-TOK - GETS THE VALUE OF PROP
 FROM THE OBJECT ST IS A TOKEN OF↓
↓
COMBINE PICTURE ELEMENTS TO FORM A COMPLETE PICTURE
THIS ROUTINE GOES THROUGH THE LIST TOKS AND MAKES PICTURE
FRAMES UNTIL ALL THE OBJECTS IN TOKS ARE INCLUDED IN SOME
PICTURE FRAME.  EACH OF THESE FRAMES WILL BE BASED ON STRONG
CONNECTIONS SUCH AS ↑SUPPORT↑.  THEN THESE PICTURE FRAMES
ARE COMBINED ON THE BASIS OF WEAK RELATIONS SUCH AS ↑RIGHTOF↑
AMONG MEMBERS OF THE PICTURE FRAMES.
VERSION 2      19 MARCH 75   G B-N     ↓
↓
NEW VERSION OF COMPICS, COMPOSED HURRIEDLY ON 20 MARCH 75.
THIS VERSION ALLOWS THE RELATIONS (LEFTOF, RIGHTOF, ABOVE,
BELOW, SUPPORT, SUPPORTBY).  ABOVE AND BELOW MAY BE COMBINED
```

WITH LEFTOF AND RIGHTOF TO PRODUCE A PICTURE WITH BOTH RELATIONS SATISFIED.
NOTE THAT THIS VERSION DOES NOT USE THE PROGRAM -DIAGRAM-.

```
↓
(COMPICS (LAMBDA () (PROG (AFRAME DIFFRAME TKS TK PFA PFB
          ETWAS TKB TKC ATTP ATTPOB YA YB TMP)
↓     MAKE A PF FOR EACH TOKEN AND PUT ON THE PROPERTY LIST
      UNDER THE NAME PF.    ↓
  (MAP TOKS (FQUOTE (LAMBDA (X)
          (PUT (CAR X) ≡PF (MAKEPF (CAR X))) )))
↓     GO THROUGH AND SEE WHAT WE CAN COMBINE.     ↓
A (SETQ TKS TOKS)
  (SETQ DIFFRAME NIL)
  (SETQ ETWAS NIL)
  (SETQ AFRAME (GET (CAR TOKS) ≡PF))
B (SETQ TK (CAR TKS))
  (SETQ PFA (GET TK ≡PF))
  (COND ((NOT (EQ PFA AFRAME)) (SETQ DIFFRAME T)))
↓     TEST FOR UNSATISFIED RELATIONS     ↓
  (COND ((UNSATR TK ≡LEFTOF) (GO L))
        ((UNSATR TK ≡RIGHTOF) (GO R))
        ((UNSATR TK ≡SUPPORT) (GO S))
        ((UNSATR TK ≡BELOW) (GO W)))
↓     END OF THE INNER LOOP     ↓
K (COND ((SETQ TKS (CDR TKS)) (GO B))
        ((NULL DIFFRAME) (DRAWPICS (GET (CAR TOKS) ≡PF))
              (RETURN))
        (ETWAS (GO A)))
  (PRINT ≡(COMPICS FAILED))
  (RETURN)
↓     UNSATISFIED LEFTOF RELATION     ↓
L (SETQ TKB (GET TK ≡LEFTOF))
  (SETQ PFB (GET TKB ≡PF))
  (COND ((ANYUNSAT TKB ≡(RIGHTOF)) (GO K)))
  (COALESCE (LIST (CADAR PFA) (CADDAR PFA))
            (LIST (CAAR PFB) (CADDAR PFB)))
  (GO K)
↓     UNSATISFIED RIGHTOF RELATION     ↓
R (SETQ TKB (GET TK ≡RIGHTOF))
  (SETQ PFB (GET TKB ≡PF))
  (COND ((ANYUNSAT TKB ≡(LEFTOF)) (GO K)))
  (COALESCE (LIST (CAAR PFA) (CADDAR PFA))
            (LIST (CADAR PFB) (CADDAR PFB)))
  (GO K)
↓     UNSATISFIED SUPPORT RELATION     ↓
S (SETQ TKB (GET TK ≡SUPPORT))
  (SETQ PFB (GET TKB ≡PF))
  (COND ((ANYUNSAT TKB ≡(SUPPORTBY)) (GO K)))
  (SETQ TMP (CAR (GET TK ≡ATTACH)))
  (SETQ ATTP (EXECLOC (CAR TMP)))
  (SETQ ATTPOB (EXECLOC (CADR TMP)))
  (COND ((EQ (CAAR TMP) TKB) (SETQ TMP ATTP)
          (SETQ ATTP ATTPOB) (SETQ ATTPOB TMP)))
  (COALESCE ATTP ATTPOB)
  (GO K)
↓     UNSATISFIED BELOW RELATION     ↓
W (SETQ TKB (GET TK ≡BELOW))
  (SETQ PFB (GET TKB ≡PF))
  (SETQ YA (PLUS (CADR (GET TK ≡STVAL))
                 (DIFFERENCE (CADDDR (GET TK ≡ORIGPF))
                    (CADR (GET TK ≡ORIGST)) )))
  (SETQ YB (PLUS (CADR (GET TKB ≡STVAL))
```

```
                 (DIFFERENCE (CADDR (GET TKB ≡ORIGPF))
                     (CADR (GET TKB ≡ORIGST)) )))
  (COND ((ANYUNSAT TKB ≡(ABOVE)) (GO U)))
↓      SIMPLE BELOW, NO LEFT OR RIGHT.  CENTER PFS.   ↓
  (COALESCE (LIST (TIMES (PLUS (CAAR PFA) (CADAR PFA)) 0.5)
               YA) (LIST (TIMES (PLUS (CAAR PFB) (CADAR PFB))
                0.5) YB) )
  (GO K)
↓      BELOW WITH LEFT OR RIGHT       ↓
U (COND ((ANYUNSAT TKB ≡(ABOVE LEFTOF RIGHTOF)) (GO K))
        ((UNSATR TKB ≡LEFTOF) (GO V)))
↓      BELOW WITH RIGHTOF        ↓
  (SETQ TKC (GET TKB ≡RIGHTOF))
  (COND ((NOT (MEMBER TKC (CDR PFA))) (GO K)))
  (COALESCE (LIST (CAAR PFA) YA) (LIST (CADAR PFB) YB))
  (GO K)
↓      BELOW WITH LEFTOF       ↓
V (SETQ TKC (GET TKB ≡LEFTOF))
  (COND ((NOT (MEMBER TKC (CDR PFA))) (GO K)))
  (COALESCE (LIST (CADAR PFA) YA) (LIST (CAAR PFB) YB))
  (GO K)
)))
↓
TEST FOR UNSATISFIED RELATION      ↓
(UNSATR (LAMBDA (TOK REL)
  (AND (GET TOK REL) (NOT (MEMBER (GET TOK REL)
          (CDR (GET TOK ≡PF)) ))) ))
↓
TEST FOR ANY UNSATISFIED RELATION EXCEPT FOR THE GIVEN LIST ↓
(ANYUNSAT (LAMBDA (TOK LST) (PROG (FLG)
  (MAP ≡(LEFTOF RIGHTOF ABOVE BELOW SUPPORT SUPPORTBY)
     (FQUOTE (LAMBDA (X)
       (COND ((AND (NOT (MEMBER (CAR X) LST))
                  (UNSATR TOK (CAR X))) (SETQ FLG T))) )))
  (RETURN FLG)
)))
↓
COALESCE TWO FRAMES, PFA AND PFB, AT THE GIVEN POINTS   ↓
(COALESCE (LAMBDA (PA PB) (PROG (PF)
  (SETQ ETWAS T)
  (SETQ PF (COMPFRM PFA PA PFB PB))
  (MAP (CDR PF) (FQUOTE (LAMBDA (X)
       (PUT (CAR X) ≡PF PF) )))
)))
↓
SUBROUTINE OF COMPICS TO SEE IF THERE IS AN OBJECT WITH
THE RELATION REL AND SET PFB TO ITS PICTURE FRAME SET  ↓
(FNDCPC (LAMBDA (REL) (PROG (PFT)
↓      SEE IF COB HAS SOMETHING IN THIS RELATION     ↓
  (COND ((NULL (SETQ OB (GET COB REL))) (RETURN)))
↓      SEE IF ONE OF THE OTHER PF SETS HAS OB IN IT     ↓
  (SETQ PFT PFS)
A (COND ((NULL (SETQ PFT (CDR PFT))) (RETURN))
        ((MEMBER OB (CDAR PFT)) (SETQ PFB (CAR PFT))
            (RETURN T))
        (T (GO A)))
)))
↓
TEST WHETHER AN ATOM OCCURS IN A STRUCTURE     ↓
(OCCURS (LAMBDA (ATM STR)
  (COND ((NULL STR) NIL)
        ((EQ ATM STR) T)
```

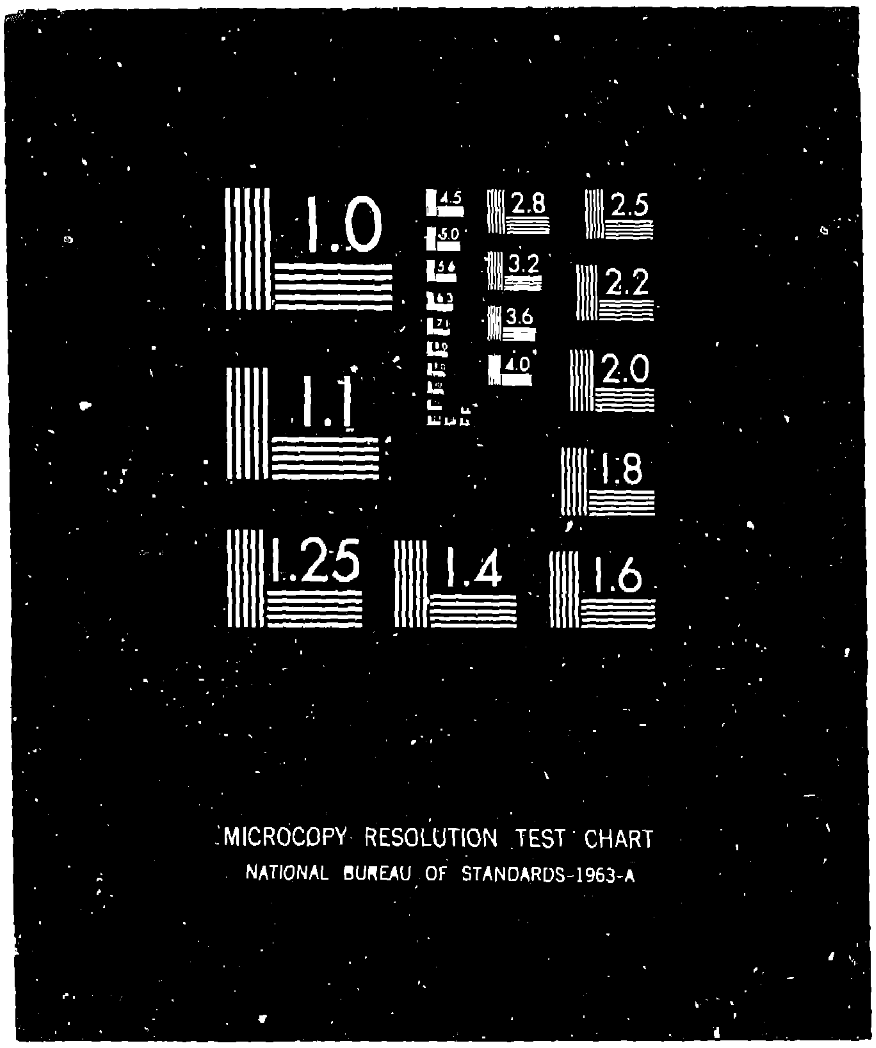
```
        ((ATOM STR) NIL)
        (T (OR (OCCURS ATM (CAR STR)) (OCCURS ATM (CDR STR)) ))) ))
↓
MAKE AN ATTACHMENT RELATION    ↓
(MAKEATT (LAMBDA (A AP B BP) (PROG (TMP ATTRS ATTR TMPB)
  (SETQ TMP (LIST (LIST A AP) (LIST B BP)))
  (COND ((NULL (SETQ ATTRS (GET A ≡ATTACH))) (GO C)))
D (SETQ ATTR (CAR ATTRS))
  (COND ((NOT (EQ (CAR (OTHER ATTR A)) B)) (GO E)))
  (COND ((NOT (EQ (CAAR ATTR) A)) (SETQ TMPB AP)
          (SETQ AP BP) (SETQ BP TMPB)))
  (COND ((AND AP (OR (NOT (EQ (CAR AP) ≡DEFAULTLOC))
           (NULL (CADAR ATTR))))
          (RPLACA (CDAR ATTR) AP)))
  (COND ((AND  BP (OR (NOT (EQ (CAR BP) ≡DEFAULTLOC))
           (NULL (CADADR ATTR))))
          (RPLACA (CDADR ATTR) BP)))
E (COND ((SETQ ATTRS (CDR ATTRS)) (GO D)))
C (SETREL A ≡ATTACH TMP)
  (SETREL B ≡ATTACH TMP)
  (RETURN)
)))
↓
OPEN AN ITEM ON THE IMLAC    ↓
(OPENITEM (LAMBDA (TOKNAME) (PROG ()
 (GOUT ≡IT (LIST ITADDER 200 0 32831))
 (GOUT ≡DI (LIST ITADDER))
 (SETQ ITADDER (PLUS ITADDER 200))
  (PUT TOKNAME ≡IMLACITEM T)
  (RETURN)     )))
↓
COMBINE TWO PICTURE FRAMES.  A PICTURE FRAME IS OF THE FORM
  ((XMIN XMAX YMIN YMAX) TOKEN ... TOKEN)
PF2 IS COMBINED INTO PF1 SO THAT THE POINT P2 IN PF2
IS THE SAME AS P1 IN PF1.    ↓
(COMPFRM (LAMBDA (PF1 P1 PF2 P2) (PROG (XA YA XP YP STPT
      XMIN XMAX YMIN YMAX TMP PF2P SP)
  (SETQ XA (CAR P1)) (SETQ YA (CADR P1))
  (SETQ XP (CAR P2))  (SETQ YP (CADR P2))
  (SETQ XMIN (CAAR PF1))  (SETQ XMAX (CADAR PF1))
  (SETQ YMIN (CADDAR PF1)) (SETQ YMAX (CADDDAR PF1))
  (COND ((LESSP (SETQ TMP (NEWX (CAAR PF2))) XMIN)(SETQ XMIN TMP))
  (COND ((GREATERP (SETQ TMP (NEWX (CADAR PF2))) XMAX)
          (SETQ XMAX TMP)))
  (COND ((LESSP (SETQ TMP (NEWY (CADDAR PF2))) YMIN)
          (SETQ YMIN TMP)))
  (COND ((GREATERP (SETQ TMP (NEWY (CADDDAR PF2))) YMAX)
          (SETQ YMAX TMP)))
  (RPLACA PF1 (LIST XMIN XMAX YMIN YMAX))
  (SETQ PF2P (CDR PF2))
A (SETQ PF2 (CDR PF2))
  (COND ((NULL PF2) (RETURN (NCONC PF1 PF2P))))
  (SETQ TMP (CAR PF2))
  (SETQ SP (GET TMP ≡STVAL))
  (SETQ STPT (LIST (NEWX (CAR SP)) (NEWY (CADR SP))))
  (PUT TMP ≡STVAL STPT)
  (GO A)
)))
↓    COORDINATE TRANSFORMATION FOR COMPFRM    ↓
(NEWX (LAMBDA (X) (PLUS XA (DIFFERENCE X XP))))
(NEWY (LAMBDA (Y) (PLUS YA (DIFFERENCE Y YP))))
```

```
DRAW A PICTURE FRAME SET      ↓
(DRAWPICS (LAMBDA (PF) (PROG (SIZE TMP XMIN YMIN BASEV
      STPT OBJECT ROT MODEL)
  (NEWFRAME)
↓     COMPUTE MAX FRAME DIMENSION AND SCALE TO THE CRT  ↓
  (SETQ SIZE (DIFFERENCE (CADAR PF) (CAAR PF)))
  (SETQ TMP (DIFFERENCE (CADDDAR PF) (CADDAR PF)))
  (COND ((GREATERP TMP SIZE) (SETQ SIZE TMP)))
↓     SAFETY IN CASE WE TRY TO DRAW A ZERO SIZE OBJECT   ↓
  (COND ((ZEROP SIZE) (SETQ SIZE 1.0)))
  (SETQ GLOBALSIZE (TIMES 0.9
          (QUOTIENT SCREENXMAX SIZE) ))
↓     FIX 19 MARCH 75 TO KEEP FROM INCREASING PICTURE SIZE    ↓
  (COND ((GREATERP GLOBALSIZE 5.0) (SETQ GLOBALSIZE 5.0)))
  (SETQ XMIN (DIFFERENCE (CAAR PF) (TIMES 0.05 SIZE)))
  (SETQ YMIN (DIFFERENCE (CADDAR PF) (TIMES 0.05 SIZE)))
  (SETQ BASEV (LIST XMIN YMIN))
A (SETQ PF (CDR PF))
  (COND ((NULL PF) (RETURN)))
  (SETQ OBJECT (CAR PF))
  (OPENITEM OBJECT)
  (SETQ STPT (VDIFF (GET OBJECT ≡STVAL) BASEV))
  (SETQ ROT (GET OBJECT ≡ORIENTATION))
  (SETQ SIZE (GET OBJECT ≡SIZE))
  (POSITION STPT)
  (COND (ROT (HEADING (MINUS ROT)))
        (T (HEADING 0.0)))
↓      DRAW THE OBJECT    ↓
  (SETQ MODEL (GET OBJECT ≡TOK))
  ((GET MODEL ≡DRAWPROG) SIZE)
  (DISPLAY)
  (GO A)
)))
↓
PICK THE PART OF A PAIR WHOSE CAR IS NOT THE GIVEN ATOM.  ↓
(OTHER (LAMBDA (PAIR VALUE)
  (COND ((EQ (CAAR PAIR) VALUE) (CADR PAIR))
        (T (CAR PAIR)))       ))
↓
DEFINE OVERALL PICTURE SCALE BY FINDING THE LENGTH OF
THE BIGGEST OBJECT FOR WHICH A LENGTH IS SPECIFIED.   ↓
(PICSCALE (LAMBDA (UOD) (PROG (SCL ATM SCLP MSCL)
A (COND (UOD (GO C)))
  (COND (SCL (SETQ PICSCL SCL))
        (T (SETQ PICSCL 1.0)))
  (COND (MSCL (SETQ MASSSCL MSCL))
        (T (SETQ MASSSCL 1.0)))
  (RETURN)
C (SETQ ATM (CAR UOD))
  (SETQ UOD (CDR UOD))
  (COND ((AND (SETQ SCLP (CAR (GET ATM ≡LENGTH)))
           (OR (NULL SCL) (GREATERP SCLP SCL)))
          (SETQ SCL SCLP)))
  (COND ((AND (SETQ SCLP (CAR (GET ATM ≡MASS)))
           (OR (NULL MSCL) (GREATERP SCLP MSCL)) )
          (SETQ MSCL SCLP)))
  (GO A)
)))
↓
CONSTRUCT A DIAGRAM FOR THE PHYSICS PROBLEM.  THE ARGUMENT
IS THE OBJECT TO START THE DIAGRAM WITH.       ↓
(DIAGRAM (LAMBDA (OBJ) (PROG (OBJL PICSCL PF OB OBPF
```

```
    INPIC ATTRS ATTP ATTPOB ATTR TMP OBB)
  (SETQ OBJL (LIST OBJ))
  (SETQ PF (LIST (LIST 0.0 0.0 0.0 0.0)))
A (COND ((NULL OBJL) (RETURN PF)))
  (SETQ OB (CAR OBJL))   (SETQ OBJL (CDR OBJL))
  (SETQ INPIC (CONS OB INPIC))
  (SETQ OBPF (MAKEPF OB))
  (SETQ ATTRS (GET OB ≡ATTACH))
B (COND ((NULL ATTRS) (SETQ ATTP (LIST 0 0))
        (SETQ ATTPOB (LIST 0 0)) (GO C)))
  (COND ((MEMBER (CAR (OTHER (CAR ATTRS) OB)) INPIC)
         (GO D)))
  (SETQ ATTRS (CDR ATTRS))
  (GO B)
D (SETQ ATTR (CAR ATTRS))
  (SETQ ATTP (EXECLOC (CAR ATTR)))
  (SETQ ATTPOB (EXECLOC (CADR ATTR)))
  (COND ((EQ (CAAR ATTR) OB)  (SETQ TMP ATTP)
         (SETQ ATTP ATTPOB) (SETQ ATTPOB TMP)))
C (COMPFRM PF ATTP OBPF ATTPOB)
  (SETQ ATTRS (GET OB ≡ATTACH))
E (COND ((EQ (CAR ATTRS) ATTR) (GO G)))
  (SETQ TMP (CAR ATTRS))
  (SETQ OBB (CAR (OTHER TMP OB)))
  (COND ((MEMBER OBB INPIC)
         (PRINT ≡(DRAWING OVERSPECIFIED)) (GO G))
        ((NOT (MEMBER OBB OBJL))
         (SETQ OBJL (CONS OBB OBJL)) ))
G (COND ((SETQ ATTRS (CDR ATTRS)) (GO E)))
  (GO A)
)))
↓
MAKE A PICTURE FRAME FOR A SINGLE OBJECT     ↓
(MAKEPF (LAMBDA (OBJ) (PROG (LNG SPT SIZ BASEV PF ROT MODEL)
  (COND ((NULL (SETQ SIZ (GET OBJ ≡SIZE))) (SETQ SIZ 5.0)))
  (SETQ MODEL (GET OBJ ≡TOK))
  (SETQ PF (GET MODEL ≡PFRAME))
  (SETQ BASEV (VSCALE (LIST (CAR PF) (CADDR PF)) SIZ))
  (COND ((SETQ ROT (GET OBJ ≡ORIENTATION))
         (SETQ PF (ROTPF PF ROT)) ))
  (SETQ PF (MAPLIST PF (FQUOTE (LAMBDA (X)
              (TIMES (CAR X) SIZ) ))))
  (PUT OBJ ≡ORIGPF PF)
  (SETQ SPT (VSCALE (GET MODEL ≡STARTPT) SIZ))
  (COND (ROT (SETQ SPT (VSUM (VROT (VDIFF SPT BASEV)
              ROT) BASEV))))
  (PUT OBJ ≡SIZE SIZ)
  (PUT OBJ ≡STVAL SPT)
  (PUT OBJ ≡ORIGST SPT)
  (RETURN (LIST PF OBJ))
)))
↓
COMPUTE FRAME FOR A ROTATED PICTURE     14 JAN 75  ↓
↓     THE ARGUMENTS ARE (XMIN XMAX YMIN YMAX) AND THE ANGLE ↓
(ROTPF (LAMBDA (PF THETA) (PROG (DX DY STH CTH XS YS)
  (SETQ DX (DIFFERENCE (CADR PF) (CAR PF)))
  (SETQ DY (DIFFERENCE (CADDDR PF) (CADDR PF)))
  (SETQ STH (SIN (TIMES THETA PI180)))
  (SETQ CTH (COS (TIMES THETA PI180)))
  (SETQ XS (LIST (VRX 0 0) (VRX DX 0) (VRX 0 DY) (VRX DX DY)))
  (SETQ YS (LIST (VRY 0 0) (VRY DX 0) (VRY 0 DY) (VRY DX DY)))
  (RETURN (LIST (PLUS (LSMIN XS) (CAR PF))
```

```
                (PLUS (LSMAX XS) (CAR PF))
                (PLUS (LSMIN YS) (CADDR PF))
                (PLUS (LSMAX YS) (CADDR PF)) ))
)))
↓
MIN AND MAX OVER LISTS      19 MARCH 75     ↓
(LSMIN (LAMBDA (L)
  (COND ((NULL (CDR L)) (CAR L))
        (T (MIN (CAR L) (LSMIN (CDR L)) ) )) ))
(LSMAX (LAMBDA (L)
  (COND ((NULL (CDR L)) (CAR L))
        (T (MAX (CAR L) (LSMAX (CDR L)) ) )) ))
↓    VECTOR ROTATIONS FOR X AND Y   USED IN ROTPF     ↓
(VRX (LAMBDA (X Y) (DIFFERENCE (TIMES X CTH) (TIMES Y STH))))
(VRY (LAMBDA (X Y) (PLUS (TIMES X STH) (TIMES Y CTH))))
↓
EXECUTE THE FUNCTION TO GET A LOCATION     ↓
(EXECLOC (LAMBDA (L) (PROG ()
  (SETQ L (CADR L))
  (COND ((CADDR L) (RETURN ((CAR L) (CADR L) (CADDR L)))))
        (T (RETURN ((CAR L) (CADR L))))) )
)))
↓
DEFAULT LOCATION, SAME AS GETLOC IN EXECUTION   ↓
(DEFAULTLOC (LAMBDA (L) (GETLOC L)))
↓
GET A LOCATION IN OBJECT COORDINATES    ↓
(GETLOC (LAMBDA (L) (PROG (OBJ MODEL LOC )
  (SETQ OBJ (CAR L))
  (SETQ MODEL (GET OBJ ≡TOK))
  (SETQ LOC (CADR L))
  (RETURN (ABSVAL OBJ LOC))
)))
↓
GET A RELATIVE LOCATION IN OBJECT COORDINATES     ↓
(FROMLOC (LAMBDA (L D) (PROG (OBJ MODEL LOC TMP)
  (SETQ OBJ (CAR L))
  (SETQ MODEL (GET OBJ ≡TOK))
  (SETQ LOC (VSUM (GET MODEL (CADR L))
             (VSCALE (SETQ TMP (VDIFF (GET MODEL ≡CG)
             (GET MODEL (CADR L)))) (QUOTIENT (TIMES
             (QUOTIENT (FLOAT (CAR D)) PICSCL)
             (GET MODEL ≡PSCALE))
             (VMAG TMP)))))
  (RETURN (ABSVAL OBJ LOC))
)))
↓
ERROR TRAP ROUTINE         ↓
(ERR (LAMBDA (MSG) (PROG (REX)
  (PRIN1 ≡ERROR:) (PRIN1 MSG) (TERPRI)
  (MAINLOOP) )))
↓
PRINT ERROR MESSAGE AND SUBSTITUTE WORDS FOR *S   ↓
(PRINTERR (LAMBDA (MSG ARGS) (PROG ()
A (COND ((EQ (CAR MSG) ≡*) (PRIN1 (CAR ARGS))
           (PRIN1 BLANK) (SETQ ARGS (CDR ARGS)))
        (T (PRIN1 (CAR MSG)) (PRIN1 BLANK)))
  (COND ((SETQ MSG (CDR MSG)) (GO A))
        (T (TERPRI) (RETURN)))
)))
```

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
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A