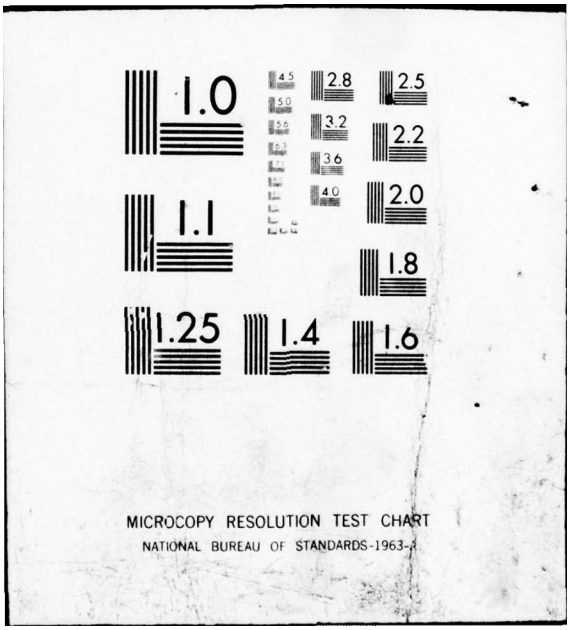
AD-A057 641 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM.(U)
OCT 77 B A GOODMAN N00014-67-A-0305-0026
UNCLASSIFIED R-798 NL

1 OF 2
AD A057641


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
5.6
6.3
7.1
8.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A057641

LEVEL II

REPORT R-798 OCTOBER, 1977 UILU-ENG 77-2245

CSL COORDINATED SCIENCE LABORATORY

A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM

BRADLEY ALAN GOODMAN

AD No.
DDC FILE COPY

DDC
RECEIVED
AUG 18 1978
D

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

78 15 08 065

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM.

5. TYPE OF REPORT & PERIOD COVERED

Technical Report

6. PERFORMING ORG. REPORT NUMBER

R-798, UILU-ENG-77-2245

7. AUTHOR(s)

Bradley Alan Goodman

8. CONTRACT OR GRANT NUMBER(s)

N00014-67-A-0305-0026

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Office of Naval Research
Arlington, Virginia

12. REPORT DATE

October, 1977

13. NUMBER OF PAGES

108

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

Master's thesis, 116p.

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

78 15 08 065

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Natural Language
Question Answering Systems
Relational Data Bases
Very Large Data Bases
Augmented Transition Networks
Semantic Grammar
Natural Language Front Ends
English Dialogue
Query Generator

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A system called PLANES (for Programmed LANguage-based Enquiry System) is under development at the University of Illinois Coordinated Science Laboratory. The primary objective of PLANES is to allow a non-programmer to obtain information from a large data base with minimal prior training or experience. Such a system must understand a substantial degree of a user's natural language and guide and educate him to formulate requests in a form the system can understand. The system must handle complex syntactic structures, abbreviations, pronoun reference and ellipsis. The system must also give back explicit answers and →over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

LB UNCLASSIFIED 097 700

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (continued)

not just retrieve a file. Minor errors--such as spelling and grammatical errors--should be tolerated. The system should be interactive and on-line, provide clarifying dialogues, operate fairly rapidly, and should be relatively easy to extend. See also AD-A028 316.

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

LEVEL II

12

UILU-ENG-77-2245

A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM

by

Bradley Alan Goodman

This work was supported by the Office of Naval Research under Contract N00014-67-A-0305-0026.

Reproduction in whole or in part is permitted for any purpose of the United States Government.

Approved for public release. Distribution unlimited.

DDC
RECEIVED
AUG 18 1978
D

A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM

BY

BRADLEY ALAN GOODMAN

B.S., Carnegie-Mellon University, 1975

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1978

Thesis Adviser: Professor David L. Waltz

Urbana, Illinois

ACKNOWLEDGEMENT

I would like to express my appreciation to Professor David Waltz for his helpful suggestions, encouragement, and patience during my work on this thesis.

TABLE OF CONTENTS

Page

1. INTRODUCTION 1

   1.1. Introduction 1
   1.2. An Overview of this Thesis 3

2. HISTORICAL PERSPECTIVE 4

   2.1. SHRDLU 4
   2.2. LUNAR 6
   2.3. SOPHIE 9
   2.4. Improving Past Systems 11

3. PLANES 13

   3.1. The PLANES System 13
   3.2. The PLANES World 14
   3.3. The PLANES Data Base 14
   3.4. Helpful Factors in the PLANES World 14
   3.5. Not-So-Helpful Factors in the PLANES World 16

4. THE BACKBONE OF THE PLANES SYSTEM 17

   4.1. Use of Augmented Transition Networks in PLANES 17
   4.2. An Example of a Sub-Network 21
   4.3. Context Registers and Subnets 21
   4.4. Pronoun Reference and Ellipsis 24
   4.5. Query Language 26
   4.6. Answer Formating 28

5. PLANES--TWO IMPLEMENTATIONS 29

   5.1. PLANES I 29
      5.1.1. Operation 29
      5.1.2. Parsing 32
      5.1.3. Putting Words in Canonical Form and Correcting Spelling 32
      5.1.4. Matching Against Prestored Request Patterns and Setting Context Registers 33
      5.1.5. Implementation of Prestored Request Patterns and Subnets 39
      5.1.6. Translation into Query Language . . 41
      5.1.7. Handling of Pronoun Reference and Ellipsis 42

5.2. PLANES II 47
5.2.1. Why PLANES II? 47
5.2.2. Concept Case Frames to the Rescue . 48
5.2.3. The System 49
5.2.4. Parsing 49
5.2.5. Concept Case Frames 50
5.2.6. Resolving Pronoun Reference and Ellipsis 54
5.2.7. Generating the Query 55

6. A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM . 58

6.1. Language Processing Model for Data Base Requests 58
6.2. What Types of Problems is the Model Suited For? 58
6.3. Central Assumption 60
6.4. Concept Case Frames and Context Registers 61
6.5. Query Generator 62
6.5.1. What Fields to Return and Operations to Perform on Them . . . 62
6.5.2. Sorting the Output 65
6.5.3. Predicate Generation 65
6.6. Constructing a Paraphrase 66
6.7. Embedded Clauses 67
6.8. Adding New Questions 68

APPENDICES 70

A Sample Queries of PLANES I 70

B Sample Queries of PLANES II 81

LITERATURE CITED 106

INTRODUCTION

1.1 Introduction

A system called PLANES (for Programmed LANguage-based Enquiry System) is under development at the University of Illinois' Coordinated Science Laboratory [Waltz, et al., 1976a]. The primary objective of PLANES is to allow a non-programmer to obtain information from a large data base with minimal prior training or experience. Such a system must understand a substantial degree of a user's natural language and guide and educate him to formulate requests in a form the system can understand. The system must handle complex syntactic structures, abbreviations, pronoun reference and ellipsis. The system must also give back explicit answers and not just retrieve a file. Minor errors--such as spelling and grammatical errors--should be tolerated. The system should be interactive and on-line, provide clarifying dialogues, operate fairly rapidly, and should be relatively easy to extend.

Two functioning versions of PLANES were developed during 1975 and 1976--PLANES I and PLANES II. Our overall goal during their development was to generate and develop ideas which will allow a casual user to use a computer effectively and confidently, with a minimum of training prior to use and a minimum of frustration during use. We approached this goal in two ways. First, PLANES was built to give a casual user access to a large data base of aircraft flight and maintenance data via the typing of natural English requests. Second, we undertook basic research on more general techniques for representing the meaning of natural language and on procedures for extracting particular meanings

from English passages.

The language processing model for the PLANES system for natural language access to a large data base is described in detail below. Key concepts and novel ideas include:

(1) the central role of ATN subnets [Woods 1970] for recognizing phrases and groups of phrases with specific meaning; in particular this allows the recognition of requests which are not grammatically correct;

(2) the use of concept case frames for checking meaningfulness of requests, for generating dialogue to find missing elements, for aiding in pronoun reference and ellipsis resolution, and for aiding translation into the query language;

(3) the use of context registers as history keepers for many purposes including especially the resolution of pronoun reference and ellipsis and the efficient answering of follow-up questions;

(4) the use of relational calculus as a convenient language into which to translate English, and a relational data base organization as appropriate to our record-based data base[Codd 1970, Palermo 1972];

(5) the programming of a powerful query generator, which takes a number of semantic features of a user's request supplemented by dialogue context information, and generates the corresponding data base query; the fact that this can be done for a wide range of requests by a single program is a central remarkable result of our research;

(6) the attack on a broad front of many practical problems including operating speed, dialogue and paraphrase generation, spelling correction and error recovery, browsing and answering of vague questions, answer generation, automatic HELP files, etc.[Codd 1974]

1.2 An Overview of this Thesis

This thesis describes the PLANES I and II systems, their design considerations, particular implementations, and so forth. Throughout their description characteristics that can be generalized for use with any relational data base are pointed out. The thesis ends with the development of a model for a language processing system. The appendices contain sample sessions with the two PLANES system. Before all this is presented, we will examine past natural language systems.

2

HISTORICAL PERSPECTIVE

2.1 SHRDLU

Systems that retrieve information requested in English from a data base have been an interest of computer scientists for many years. For example, a program called SHRDLU, completed in 1971 and written by Terry Winograd at M.I.T., works on a small data base describing a world of geometric solids such as rectangular blocks, wedges and pyramids [Winograd 1973]. SHRDLU can display this "micro-world" on a CRT screen and actually simulate the movement of members of the world with its own imaginary hand. The user can request SHRDLU to perform certain manipulations of the blocks and to answer questions about the current scene. In addition SHRDLU can comprehend declarative sentences (e.g. "The blue pyramid is nice.") and imperative sentences (e.g. "Pick up a big red block.") as well as procedural statements (e.g. "A steeple is a stack which contains two green cubes and a pyramid.").

SHRDLU consists of a set of recursive procedures that can profitably describe natural language grammars and parsers. It uses a fairly comprehensive grammar of English. The parser is organized around syntactic units, which play a primary role in determining meaning. For each syntactic unit, there exists a program (written in the language PROGRAMMAR also developed by Winograd at M.I.T.) which operates on the input string to see if it can represent that type of a unit. In the process of doing this, it calls on other syntactic programs (and even possibly recursively on itself). These programs incorporate descriptions of the possible orderings of words and other units. When the

parser finds a syntactically acceptable phrase, it performs a semantic analysis on it to decide whether to continue along the current line of parsing.

Winograd's system is based on a theory by Halliday (1970) called Systemic Grammar which incorporates both syntactic and semantic information. Instead of relying on a "deep structure" tree, it describes the interaction and dependency of different features on each other. It is concerned more with the way language is organized into units, each of which has a special role in conveying meaning. Systemic grammar uses the WORD as its basic building block. Classes of words such as "noun", "verb", and "adjective" are used.

The next unit above WORD is GROUP. Such groups include noun groups, which describe objects; verb groups, which convey messages about time and modality; prepositional groups, which describe simple relationships; and adjective groups which convey other types of relationships and descriptions of objects [M.I.T. A.I. Progress Report 1974]. Each of the groups has "slots" for the words of which it is composed. For example, a noun group has slots for the "determiner", "numbers", "adjectives", "classifiers", and a "noun."

The most complex unit of the language is the CLAUSE. It is used to express relationships and events that involve time, place, and manner. A clause can be a QUESTION, a DECLARATIVE, or an IMPERATIVE, it can be in the "passive" or "active" form, it can be a YES-NO or WH-question, and so forth. Clauses can be made up of other clauses and they can be used as parts of groups in many ways.

The interpretation of a request by SHRDLU is done by making use of a detailed world model that describes both the current state of the blocks and its knowledge of procedures that allow it to change that state. The model is a symbolic representation that shows those aspects of the world that are relevant

to the operations needed to work with it and to discuss it. At any specific time, a set of facts is stored in the data base that describes the condition of the blocks world by simply indicating relationships between objects (e.g. "(ABOVE B1 B2)"). These facts are created by a dozen or so programs that are called "semantic specialists." The specialist programs are experts at unraveling particular syntactic structures into meaningful forms. They inspect both the syntactic structures and the meaning of each word to build up "theorems" that can be used by the deductive systems to perform actions or to answer questions, or for the syntactic system itself to decide if a proposed noun group makes sense.

2.2 LUNAR

The Lunar Sciences Natural Language Information System (LSNLIS) contains information about samples of lunar rocks and soils that were returned by the Apollo moon missions. The system, called LUNAR, was developed by William Woods and other co-workers at Bolt, Beranek and Newman, Inc. It is an experimental question-answering system that was designed to help geologists access, compare, and evaluate the data. It is able to accept grammatically complex sentences, involving nested dependent clauses, comparative and superlative adjective forms and some types of anaphoric reference. LUNAR performed well in its domain of geology. In fact, at a geology conference in 1971, it was able to answer 78% of the questions solicited from geologists, and was judged capable of answering 90% if simple changes or additions to the system were made.

The syntactic component of LUNAR is an augmented transition network grammar. The grammar is implemented by an augmented transition network (ATN). An ATN is a set of recursive procedures that can efficiently describe natural language grammars and parsers. It consists of sets of nodes and branches emanating from the nodes. Each branch is a labeled directed arc that is allowed to specify a condition and a sequence of actions to be taken if the condition is met. ATNs enable one to try out different kinds of parsing strategies on variably large phrases in a sentence, to store information relating to the success of those strategies as they are being carried out, and to recognize whenever a given strategy has failed so that a new strategy can be tried. In particular, ATN parsers employ a depth-first backtracking algorithm as they attempt to traverse a path of nodes and arcs leading from the starting state to some accepting state. The value of the input string and the tests applied to it determine which paths are taken in the ATN.

LUNAR's ATN parser attempts to map an input request into a deep-structure-like representation. As transitions occur in the nets, the parser builds up parts of a deep-structure tree and stores them in "registers" until they can be combined into larger groups, and, ultimately, into a complete representation of the input. LUNAR defines a sentence as consisting of a subject noun-phrase; an auxiliary-verb component that specifies the tense, modality, and aspect of the sentence; a verb-phrase containing the main verb; the direct and indirect objects; and possible adverbial and prepositional phrase modifiers [Woods, et al. 1972]. The basic approach of the parser is as follows: at a sentence level, it tries to determine whether the input string is declarative (e.g. "John needs money.") or a interrogative. At the next lower level, the parser attempts to find the subject noun-phrase, the verb phrase, the objects, etc. Each attempt at parsing those constituents requires descending

into lower levels looking for such forms as adjectives, determiners, prepositions, etc. In the process it is possible for recursive calls to be made to some of the nets.

Semantic interpretation in LUNAR involves translating the parse tree of the sentence into a program in a formal query language that can be executed to retrieve or compute the answer. A formal query is essentially a retrieval program that computes the truth values of propositions or carries out commands. The formal query language in LUNAR consists of primitive commands, functions, and predicates which may be combined and quantified. The actual interpretation of a sentence into the query occurs in two phases.

The first phase looks to see if there are any operators or commands such as NOT, TEST, etc., that govern the sentence. This phase is performed before actual examination of the input sentence itself (and is thus a preprocessing phase). The phase consists of a search for rules which match anything in the input. The handling of compound sentences, declarative sentences, imperative sentences, and questions begins here. The second phase uses the main verb in the sentence and rules associated with that verb to interpret more of the sentence.

The semantic interpretation of a sentence normally requires the interpreting of one or more noun phrases. First the determiner structure of the noun phrase is examined to decide what kind of quantifier should govern it. This quantifier structure is used during the rest of the analysis when the noun of the phrase and relative clauses are handled. Other noun phrases that consist of topic descriptions are treated differently. In their case a set of rules is used to translate the syntax trees into Boolean combinations of important phrases.

2.3 SOPHIE

SOPHIE is a computer system developed by Brown, Burton and Bell (1974) at Bolt, Beranek and Newman, Inc. SOPHIE takes advantage of the symbol manipulation capabilities of a computer while providing a new kind of educational environment for a user. The designers of SOPHIE endeavored to create a CAI (Computer-Aided Instruction) system that had sufficient symbolic knowledge, problem solving strategies, and natural language competence so that it could duplicate some of the capabilities of a human tutor. They wanted SOPHIE to be able to respond, on its own, to a student's questions, to evaluate his hypotheses, and to give detailed feedback about his answers. This meant that it had to demonstrate a sense of "understanding" about the subject area it was teaching.

The subject area chosen for SOPHIE was that of electronic troubleshooting. SOPHIE gives a student a controlled environment in which common sense reasoning can be exercised. This is very beneficial considering that a typical laboratory used for learning troubleshooting is severely limited since it is difficult to introduce faults into a circuit. It was also possible for powerful inference procedures to be designed to handle questions that occur in this domain.

SOPHIE uses several representations of knowledge including semantic/conceptual networks. Several inferencing procedures are associated with each of these representations. SOPHIE is capable of limited context switching in its environment. Its linguistic abilities are designed specifically for its subject domain and are not at the level to handle most of the English language.

SOPHIE employs a "semantically driven fuzzy parser." This parsing scheme takes advantage of the powerful constraints that exist in the relationships between the various semantic/conceptual constituents that make up a question. The refinement of the usual syntactic categories into relevant semantic/conceptual categories allows the parser to utilize the high degree of semantic predictability inherent in the domain. The parser is very efficient due to its top down organization that is driven on the basis of semantics rather than syntax. It incorporates a certain amount of "fuzziness." Should the parser be searching down a particular path due to the current semantics, and the current word does not fit this instantiation, the parser can skip over that word and continue searching along that path. This allows the parser to ignore words. Other features SOPHIE exhibits are the expansion of compounds before the parse and the correction of spelling.

Semantic interpretation of a query occurs during the parsing phase. When a phrase is parsed, the structural representation of the phrase returned by a grammar rule is a piece of LISP code which, upon evaluation, represents the meaning of the phrase. Associated with each semantic category in the grammar is a class of descriptions of various ways of using the category. A set of functions are attached to these categories to fill in the information that characterizes the category from the current phrase.

To extend the domain of the natural language portion requires two types of changes which must be made to the grammar. When a new concept is to be added to the system, a new rule must be written that will accept the various ways of stating the concept. Allowing the grammar to accept a new way of accepting something that it already handles requires rewriting existing rules. In either case, one must be careful that no undesirable interactions between rules yields unwanted side-effects.

2.4 Improving Past Systems

The approach of previous natural language systems have hinged heavily on the use of syntax to map sentences into some deep-structure form that can be semantically interpreted. We contend that the syntactic component of the system is of limited value in systems that completely separate the syntactic and semantic interpretation sections. A better approach would be one that incorporated both syntactic and semantic analysis together. The SOPHIE system, as previously described, demonstrates how well such an approach works. The whole task of parsing was performed more efficiently because SOPHIE utilized the powerful constraints inherent in the semantic/conceptual entities that make up a question.

Such a grammar, that incorporates both syntactic and semantic information, is called a "semantic grammar" [Brown and Burton 1976]. In a semantic grammar, the usual syntactic categories such as noun phrase, verb phrase, adjective, etc. are not used but instead semantic categories that encompass the domain of expertise of the system are employed. The grammar resulting from such a formalization must give specifications of constraints between concepts. These specifications expound the ways of expressing a particular concept in terms of its constituent concepts. Such a grammar will only accept those sentences that are semantically well-formed (i.e. meaningful) with respect to the system.

Other changes in past natural language systems are necessary to achieve a system that is useable by non-technical people. Codd (1974) made many recommendations for the implementation of such a system. He recommended the selection of a simple data model such as a relational data base. This would

make it easier to relate semantic concepts to the actual data. To enable the system to detect ambiguity, incompleteness, or incompatibility, Codd suggested that a user's query be translated into a very precise target language. His data sublanguage ALPHA is an example of such a language [Codd 1971b]. When the system had trouble comprehending the intent of a user's request, Codd felt that it would be necessary to enter into a clarification dialog with the user. To make such a dialog workable, he believed that the system must force the user to confine his responses to a small domain. When a response was understood by the system, a paraphrase of the user's query should be fed back to the user so that he or she may confirm that the system correctly comprehended the request. If the request was not correctly grasped, the user could enter into a clarifying dialog with the system. When, and only when, the user and system were in complete agreement, should the data base be searched. This could save time and money due to unnecessary searches. When all else failed, Codd suggested that the system fall back to a multiple choice sequence that allowed the user to enter his or her request.

3

PLANES

3.1 The PLANES System

Like LUNAR, PLANES is designed to operate on a real data base, but PLANES operates on a much larger data base with a wider universe of discourse. In addition, PLANES is capable of performing complex operations on the data and employing strategies in the search operations.

To give an idea of the current capabilities of PLANES and some of our future goals, a sample exemplified scenario is presented below.

```
User:  How many flight hours did plane 3 log in Jan 73?
PLANES:  ((SUM FLIGHT HOURS) = 33.)
User:  In March?
PLANES:  ((SUM FLIGHT HOURS) = 0.)
User:  For plane 4?
PLANES:  ((SUM FLIGHT HOURS) = 5.)
User:  What maintenances were performed on plane 3
       during that period?
PLANES: Do you mean
          1.  in March 73
          2.  in Jan 73
          3.  in Jan and March 73
          4.  another time period?
User:  Three.
PLANES:     DATE                HOW MALFUNCTIONED

            Jan 1                  fuses blown
            Jan 5               internal engine noise
            Jan 17                 engine removed
            Mar 12                 power supply blown
User:  Which of those were performed by maintenance group
       HEBRON FMFPAC?
PLANES:     DATE                HOW MALFUNCTIONED

            Jan 5               internal engine noise
            Jan 17                 engine removed
User:  Was it able to make any flights in February?
PLANES:  ((TOTAL FLIGHTS) = 0.)
```

3.2 The PLANES World

This section describes the general environment in which PLANES operates. This environment is composed of the data base, the user, and the user's range of queries.

3.3 The PLANES Data Base

We have obtained a data base (on the order of 10^8 bits) from the Navy 3-M (Maintenance and Material Management) Data Base for Aircraft, Mechanicsburg, PA., consisting of complete records of aircraft maintenance and flight information for 48 A7 and F4 aircraft, extending over a period of two years. Every time a plane is serviced, a record is made that includes such information as the time and duration of the maintenance, who performed it, what action was taken, which parts were removed, replaced, or installed, whether or not the service was scheduled or unscheduled, and so on. Records on the number of flights and the number of hours in the air are also kept for each plane. There are approximately forty different record formats which occur in the data base, each containing ten and twenty separate fields, where each field encodes information like the date of the action, type of aircraft, serial number of the aircraft, type of malfunction, component serviced, or the work center performing maintenance and so on.

3.4 Helpful Factors in the PLANES World

In designing PLANES we have followed a more or less engineering approach that falls outside the realm of traditional linguistics. The dictionary entries are simplified or eliminated for many words since words are defined implicitly within patterns stored in a network of phrases. Non-grammatical sentences can

be accepted as meaningful to allow the system to be tolerant of common grammatical errors. These and other user-oriented approaches are possible because a number of factors contribute to making our problem much easier to solve than the general problem of understanding unconstrained natural language.

(1) Lack of ambiguity- Relatively few words and sentences in the PLANES world are ambiguous. This implies that if PLANES can find any interpretation at all for a request, it is in all likelihood the correct interpretation.

(2) Small vocabulary- Our current system has about 1100 words. We estimate that 2000 words will cover 90% or more of all requests made by users with at least some prior experience with PLANES.

(3) Only two modes- PLANES is either answering a question or trying to aid a user in formulating a request in terms it can understand.

(4) People do not type complex sentences- The increasing likelihood of making typing errors in lengthy requests, the increasing likelihood that long requests will puzzle a program in some respect, and general laziness all contribute to keeping input requests short and simple in construction. Malhotra (1975) performed an experiment in which non-programmers thought they were communicating with an intelligent program, when in fact they were interacting with another person who would respond appropriately to any input. He found that 10 simple sentence types covered 78% of all input requests, and that another 10 would handle all but 10% of the requests.

(5) Less than 100% answer rate is acceptable- We feel that a 90% answer rate without the need of rephrasing the request would be adequate to keep a user's interest and provide a practical and useful system. Even a lower rate might be reasonable.

3.5 Not-So-Helpful Factors in the PLANES World

Other factors in the PLANES world would make our task more difficult.

(1) The system must contain a great deal of specialized knowledge- one of our major realizations has been that a small number of general rules cannot suffice to "translate" natural English requests into data base queries. Consider the following sentences:

(S1) "Which A7 has the worst maintenance record?"

(S2) "Find any common factors of plane numbers 37 and 78."

Clearly the system must contain special programs to assemble a "maintenance record", and special knowledge to judge its "goodness"; the system must know that having the same digit as an element of their serial number does not constitute a "common factor", but that similar event sequences are important. We must also beware that "goodness" and other such factors may have two different meanings to two different users.

(2) Each request may be expressed in a great many different ways- Clearly if users are encouraged to use the system with little or no prior training, and if the system is expected to understand enough of a user's input to keep his interest, and perform useful actions from the beginning, then the system must be able to make some sense of a large number of types of queries, and a wider range of syntactic constructions.

4

THE BACKBONE OF THE PLANES SYSTEM

4.1 Use of Augmented Transition Networks in PLANES

We utilize ATNs in a manner different than the way they have previously been used. In all versions of PLANES created so far, the ATNs have been employed to store the actual words used to form phrases, which in turn one puts together to form requests, instead of storing instructions necessary to perform a syntactic parse [Finin 1977]. This allows us to forget about syntactic information and work only with semantic concepts. In other words, we have chosen to implement a "semantic grammar" as described earlier.

For each semantic concept that forms the grammar, we have a specialized sub-network to recognize it. In all we have a total of twelve specialized sub-networks which recognize such constituents as time specifications, plane specifications, etc. Each specialist is called in when the parser suspects that the current constituent is of a particular semantic category. The subnet, if it successfully parses the constituent, returns a representation of what it has found.

The following gives a brief description of the twelve specialist networks and some examples of typical phrases they recognize.

PLANETYPE: Recognizes noun phrases which refer to a plane or group of planes. Examples of the phrases it recognizes are:

"Skyhawks that had engine damage"

"The F4 with tail number 00048"

"Plane 051"

"BUSER number 016"

MAINTENANCE: Recognizes phrases referring to maintenance, either by their work-unit-code, action-taken-code, or type-maintenance-code. Some of the constructions recognized are:

"unscheduled maintenance on ..."

"scheduled work with ..."

MAINTAT: Recognizes maintenance actions when referred to by their action-taken codes. Some examples of phrases are:

"action taken code Q"

"AT 6"

MAINTTM: Recognizes maintenance actions when specified by a legal type-maintenance code. For example:

"TM Y"

"type maintenance code 10"

MAINTWUC: Recognizes maintenance when referred to by a legal work-unit code. Examples of accepted phrases are:

"system work unit code 46"

"WUC 91"

"work unit 97"

DAMAGETYPE: Recognizes a specification of damage to aircraft. The damage can be to a general area or system, or can refer to a particular howmal-malfunctioned code or codes. For example:

"damage to electronic systems"

"damage due to howmal code 53"

"launch damage"

"mechanical system damage"

"HOWMAL 45 damage"

TIMEPP: Recognizes specifications of time periods, either as prepositional phrases or adverbial phrases. A time period can be a day, a month, a year, or a span of time between two days, months, or years. For example, the following are accepted:

"last month"

"before April 1973"

"between January 1 and June 31, 1975"

"during the month of August"

"on 10 Dec. 75"

DATE: Recognizes a time period which is a date. A date can contain a value for the month, day and year. Some strings which are accepted include:

"June 29 73"

"1 January"

"July fourth, 1976"

"5 28 77"

PLACE: Recognizes a phrase denoting a location. Our current implementation accepts locations in a coded form, although a few common names are included. Some examples are:

"P9R"

"Midway"

"VMFA-122"

CODETYPES: Recognizes phrases which refer to one of several classes of codes used in the data base. These include the action-taken codes, work-unit codes, etc. Some examples are:

"when discovered codes"

"malfunction codes"

ABBREV: Recognizes a number of phrases which are mapped into the appropriate abbreviations as well as accepting abbreviations and producing the appropriate phrase.

AMOUNT: Recognizes phrases used as numerical quantifiers. It produces a list of predicates which represent the phrase. Examples are:

"three or more times"

"more than 6"

"between 100 and 200"

"greater than 100 but less than 1000"

"30"

"any"

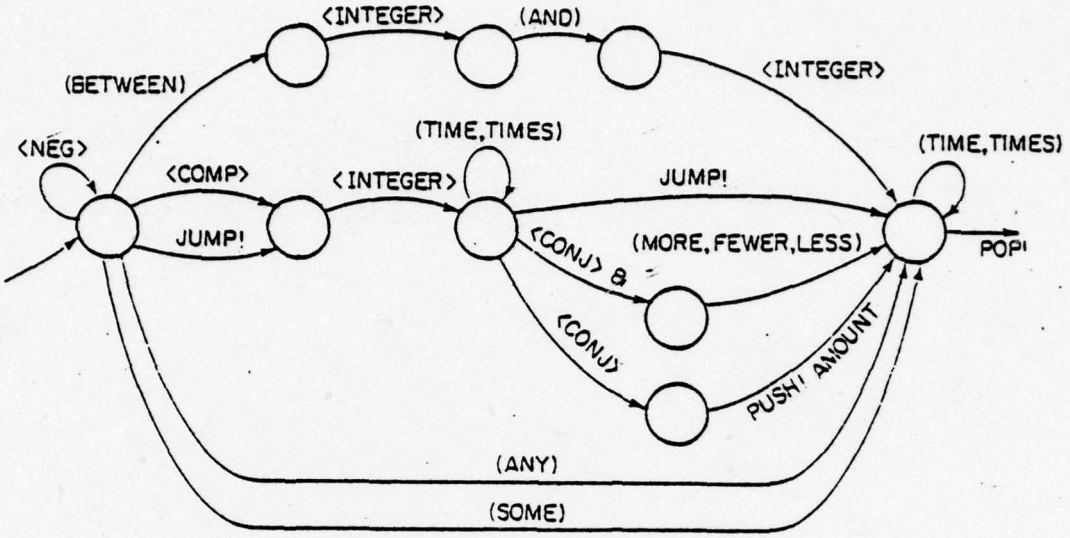
4.2 An Example of a Sub-Network

The AMOUNT net has the responsibility of recognizing phrases which are used as numerical quantifiers and to produce a list of predicates that represents the phrase. It accepts such expressions as "between 10 and 20", "greater than 13", "exactly twelve", and "any."

Figure 1 shows a representation of the subnet in graphical form. The following conventions are used in the figure. Any arc labeled with a word or list of words in parentheses is taken if and only if the current input word is equal to that word or is a member of the list. Arcs with a word in angle brackets are taken only if the current word is of that lexical category. For example, the arc labeled <INTEGER> is taken only if the current word is an integer. An arc labeled JUMP! can always be taken, but the input word is left the same and is not advanced. An arc labeled PUSH! makes a recursive call to another subnet before it is taken.

4.3 Context Registers and Subnets

In the PLANES system we have tried to combine as many of the syntactic and semantic processes together as possible. We felt that as the parsing task occurs, useful semantic information is also being generated concurrently. To gain access to this information, much of our parser was concerned with parsing specific entities known to be required for inquiries about information in the PLANES data base. Such necessary information includes the types of planes, the damage types, the maintenance types, the time period in question, etc. As each of these entities is matched, it is saved--either in some coded form or exactly as it appears in the sentence--in a register. We call these registers "context



<COMP>	<CONJ>	<INTEGER>	<NEG>
MORE THAN	AND	1,2,3...	NO
AT LEAST	BUT	ONE	NONE
EXACTLY	OR	TWO	NOT
APPROXIMATELY		THREE	
AS MANY AS			

FD-1042

FIGURE 1 The AMOUNT net

registers."

The context registers are normally filled with information that has been returned from the subnets of the augmented transition network. These subnets are designed to parse specific parts of the user's request. When a parse of a specific entity occurs in the subnet, a list is generated that contains the "essence" of the entity. Some of the information coded in the lists includes such information as whether or not a pronoun was parsed, specific code numbers, or dates. A list of subnets and what they return to the upper level net (a process called "popping") follows. The symbol "+" used below represents information that is filled in by the subnets from the current request.

*planetype: Parses a name of a plane. It pops up
(plane (type +) (buser +)).

*mainttype: Parse a maintenance-type, maintenance-action, or work unit code. It pops up
(maint (type-maint +) (action-taken +) (workunitcode +)).

*maintat: Parse an action-taken (at) code that gives the current status of a maintenance task. It pops up
(<at>).

*mainttm: Parse a maintenance type (tm) code. It pops up
(<tm>).

*mainwuc: Parse a work unit code (wuc). It pops up
(<wucsys>).

*damagetype: Parse a how malfunctioned code (howmal). It pops up
(<howmal>).

*timepp: Parse a time range. It pops up
(time <date 1> <date 2>).

*date: Parse a date. It pops up

(date (month +) (day +) (year +)).

*place: Parse a place. It pops up

(place +).

*codetypes: Parse a codetype (describes the information available in the data base). It pops up

(<code>).

*abbrev: Parse an abbreviation or phrase to be abbreviated. It pops up

(abbrev <phrase> <abbrev>).

*amount: Parse a quantifier. It pops up

((<relation> <no.>) (<relation> <no.>)).

The information that is popped from the subnets is stored directly in the context registers.

4.4 Pronoun Reference and Ellipsis

Pronoun reference is one of the more difficult tasks in natural language understanding. It requires using the current context of a sentence to find a set of possible elements that the pronoun may stand for from the immediate past history of inquiries to the system. After weighing the merits of each member of the set, the system must select one as the "best" choice to replace the pronoun. Ellipsis is a very similar problem that occurs when a request does not express a complete thought. It involves filling in missing information from a user's request with information from the immediate past. As an example a user might follow the question "How many maintenance actions did plane 004 have in May 1974?"

with "In June?".

Winograd (1972) handles pronoun reference at the time a pronoun is parsed. He utilizes a set of programs that are designed to handle specific pronouns. For example, the definitions of "it" and "they" utilize a special heuristic program which looks into the past for anything that the pronoun might refer to. For each possibility, a value is assigned which represents the likelihood that the phrase could be represented by the pronoun. When more than one phrase is possible, all are carried along through the rest of the analysis of the sentence. At the end, an ambiguity mechanism is applied to decide which is the best choice. If no choice can be made, the user is asked for clarification.

Woods' (1972 et al.) routine analyzes pronouns in the semantic phase. When a pronoun is found as a node in the syntax tree, a function is applied to the node to try to determine what the pronoun stands for. This determination is made by searching through a list of antecedent noun phrases for one that has a syntactic and semantic structure parallel to the given node. The node is then replaced by the antecedent noun phrase that was selected.

The SOPHIE [Brown and Burton 1975] natural language processor handles ellipsis in a novel way. When it deduces that information is missing from the current request, SOPHIE looks for a previously mentioned use of a currently specified object. The semantic grammar is used to identify the type of concept that is involved. A context mechanism then searches the past history for a specialist in a previous parse that will accept the given concept as an argument. Once this is found, the phrase can be substituted into the current request.

A description of the handling of pronoun reference and ellipsis for the two working versions of PLANES--PLANES I and PLANES II--can be found in the sections that describe each of them in chapter 5.

4.5 Query Language

A formal query language has been developed for our own data base [Green 1976]. We are using Codd's notion of a relational view of data [Codd 1970]. This data model has its foundation in the mathematical theory of relations. In this model the data is viewed as being divided into "relations" which correspond to files in conventional data base terminology. Each relation contains a collection of "tuples" which correspond to records, and each tuple contains one or more "domains" or fields. One can think of a relation as a table with each row being a tuple and each column a domain. The query language is an ALPHA sublanguage [Codd 1971b]. It turns the requests into procedures, retrieves the relations, searches, returns "hits" and stores temporary relations. Optimization is also performed.

Example Consider the query language expression in Figure 2.

The meaning of the expression is:

(1) "FIND 'ALL" means return all items that satisfies the requirements. The "ALL" could be replace by a specific number so that only so many "hits" are returned.

(2) V is a variable name that becomes bound to the relation.

(3) M specifies a relation name. The M relations are the set of daily maintenance files.

(4) '((V BUSER) (V ACTDATE) (V HOWMAL)) is the list of fields whose values are to be returned. A table can be generated from the output with columns BUSER (BUreau

SERial number--a plane identification number), ACTDATE (a five digit number that represents the Julian DATE that the ACTion took place), and HOWMAL (HOW MALfunctioned--a damage code that describes what malfunction occurred).

```
(FIND 'ALL
      '((V M))
      '((V BUSER) (V ACTDATE) (V HOWMAL))
      '(AND (EQU (V PLANETYPE) A7)
            (AND (GEQ (V ACTDATEDAY) 1.)
                 (LEQ (V ACTDATEDAY) 60.)
                 (EQU (V ACTDATEYR) 2.))
            (OR (EQU (V HOWMAL) 1.)
                (EQU (V HOWMAL) 2.)))
      '(BUSER DOWN))
```

FIGURE 2 Query Language expression representing "Which A7s had either malfunction 1 or 2 damage between Jan 1 and Mar 1 1972? Sort in descending order by serial number."

(5) The six lines beginning with '(AND... specify the predicate that must be true for each entry in the answer table, namely the plane type must be an A7, the day must be between the 1st and 60th day, the year must be 2 (1972), and the malfunction code must be either a "1" or a "2".

(6) '(BUSER DOWN) means the output table should be sorted in descending order by BUSER (plane serial number).

4.6 Answer Formating

Once the data has been retrieved, the results are passed to an output module, which decides on an appropriate display format for the data. If possible it attempts to produce a graph. This can only be done if (1) pairs of items are returned, (2) one item is numerical, (3) the number of items returned is small enough (but not too small) to produce a reasonable graph which will fit on a CRT screen. If a graph is not possible, the system will produce a list or table; if there is too much data to fit on the screen, the results will be automatically output to the line printer.

5

PLANES--TWO IMPLEMENTATIONS

Two working versions of PLANES have been developed. Both versions rely heavily on ATN subnets and context registers, yet their implementations vary greatly. We describe in the next several sections their respective implementations, their strengths and weaknesses, and so on. The culmination of the work involved in developing PLANES I and II has been the development of a model for processing English requests for information from a relational data base. The description of this model appears in the next chapter.

5.1 PLANES I

5.1.1 Operation

The processing of a user's request is divided into four main phases: parsing, interpretation, evaluation, and response (see Figure 3).

(1) The first phase, the parsing, is accomplished by matching the input against request patterns stored in a large ATN [Woods 1970]. This network defines a "semantic grammar" [Brown and Burton 1975] which maps the user's request into an internal representation, the paraphrase language. The grammar is "semantic" in that it incorporates semantic and pragmatic knowledge as well as syntactic knowledge.

(2) In the interpretation phase, the paraphrase language representation of the user's request is translated into a 'program' to generate the data to answer the request. This stage contains specific knowledge about the data base. It knows, for example, which sub-parts of the data base contain certain relations

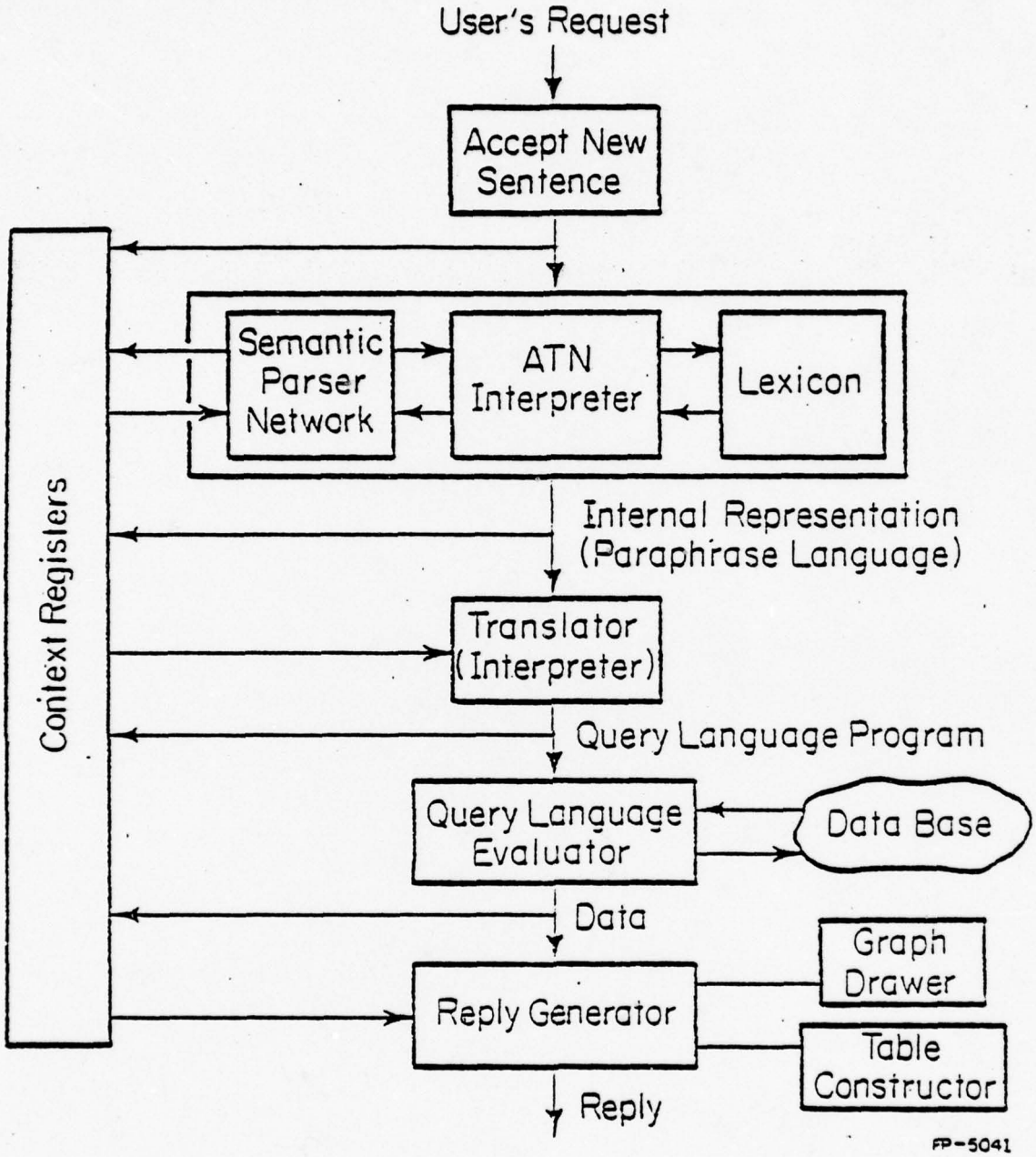


FIGURE 3 An overview of the PLANES system

and the particular codes used to represent certain facts. The program is constructed out of primitives of the query language.

(3) The evaluation phase uses the program generated in the previous state to search the data base and construct an answer. This portion is based on a relational data model [Codd 1970, Date 1975], and is implemented along lines described by Palermo (1972).

(4) The evaluation portion passes the resulting data to the response generator. This module can display the answer in one of three forms: as a simple number or list, as a graph, or as a table. The choice of output form can be determined by the user through a direct request (e.g. "Draw a graph of...) or by a set of heuristics which attempt to find the "most" natural form. A graph, for example, will be generated only if the data consists of a set of tuples which can be interpreted as a function of two variables. Furthermore, the number of tuples must lie within certain bounds, so that the entire result will fit on a CRT screen.

At each stage of the process, the results are sent to the history keeper which manages a set of stacks of relevant information. These stacks contain the results of each stage (e.g. user's request, paraphrase, etc.), syntactic components (e.g. subject, object, etc.), and semantic/contextual information (e.g. time specifications, plane specifications, etc.). This information is made available for resolving anaphoric reference, supplying phrases deleted through ellipsis, and generating responses.

The entire process can be aborted by any of the major stages. If this is done, a suitable error message is generated and the user is asked to restate either part or all of his request.

Each of the main phases is described in detail in the following sections. For greater understanding, we will trace a request through all phases of processing. Assume that the system has already successfully answered the question:

(Q1) "Which A7s logged less than 5 flight hours in February 1973?"

We then give PLANES the following request:

(Q2) "Which ones logged betwen 10 and 20 flight hours in Jan?".

5.1.2 Parsing

In the PLANES system, "parsing" involves three operations:

(1) putting all words and phrases in canonical form and correcting spelling;

(2) matching against prestored request patterns and setting context registers (history keepers); and

(3) constructing a paraphrase of the input request. These operations are explained in more detail in the next several sections.

5.1.3 Putting Words in Canonical Form and Correcting Spelling

The parser first checks to make sure that each word of the input is known by the system. Roots and inflection markers are substituted for inflected words, canonical words are substituted for synonyms, and single words are substituted for certain phrases (e.g. "USA" for "the United States of America"). If a given input word cannot be found in the dictionary, then the spelling correction module is called. This module attempts to find dictionary entries "close" to the input word; if one or more candidates are found, the

user is given an appropriate message, and if one of the system's guesses is correct, it is inserted in place of the misspelled word. If no candidates are found, or if none of the candidates is the properly spelled word, the system calls a word adding module to try to add the user's word to the dictionary.

Example 1 Given our question:

(Q1) "Which ones logged betwen 10 and 20 flight hours in Jan?"

the parser first notes that "betwen" is not in the dictionary; the spelling corrector finds that "between" is the most similar dictionary entry, and types back:

(A1) Is betwen a misspelling of between? Type y or n.

Once the user types "y", the system substitutes "between" for "betwen", and proceeds. "(Log past)" is substituted for "logged", "flighthours" is substituted for "fight hours", and "January" is substituted for "Jan." Thus the output of this first operation is:

(Q2) Which ones (log past) between 10 and 20 flighthours in January?

5.1.4 Matching Against Prestored Request Patterns and Setting Context Registers

This phase comprises the heart of the language understanding process. It is here that pronoun reference and ellipsis are resolved, and here too that much of the overall programming effort for the system has been expended.

These are three large program portions of interest here:

(1) the prestored request network;

(2) the subnets; and

(3) the context registers.

The prestored request network is made up of pattern-action pairs, some simplified examples of which are shown in Figure 4. If the input matches a pattern, then the action corresponding to that pattern is executed. The action can be a data base search, the selection of a HELP file, or any other program we wish to associate with an input pattern. Many different patterns can point to a single action (as in the case with A1 in figure 4).

Each of the starred pattern elements in Figure 4 (e.g. *planetype) must match a noun, verb or prepositional phrase which has a specific meaning (e.g. a type of aircraft, a period of time, etc.). Pattern elements preceded by a number sign (#log) match any synonymous word or phrase (e.g. logged, had, recorded, etc.).

The matching of starred elements of the patterns with phrases in the input is handled by subnets. Each subnet is a phrase parser which matches only phrases with a specific meaning. Some examples of phrases which the subnet for *planetype would match are: "A7", "phantom", "phantom or skyhawk", "ones" (where some type of aircraft is an appropriate referent of "ones") and "A7s which crashed in May."

Some of the sentences (besides Q1) which match pattern P1 of Figure 4 are:

(Q3) Find which phantoms had fewer than 15 flight hours in Feb. 1974.

(Q4) Please tell me which skyhawks and F4s logged no flight hours.

(Q5) Which A7s that crashed in May had 50 or more flight hours in April?

Pattern	Matching Sentence	Action
(P1) Which *planetype #log *flighthours	(S1) Which A7s logged between 10 and 20 flight hours in Jan 1973?	(A1) Generate paraphrase from context register values
(P2) Which *planetype #log *flights	(S2) Please tell me which planes had 20 or more flights?	(A1) (A1)
(P3) How many *flighthours #log by *planetype	(S3) Find the number of flight hours logged by plane 76 during Feb. 1974	(A1)
(P4) Which kinds of *data-items #exist	(S4) Which kinds of planes do you know about?	(A2) List *data-items from HELP files.

FIGURE 4 Sample pattern-action pairs

Whenever a subnet matches a phrase, it sets the values of its corresponding context register, which acts as a history keeper. Context registers are used for pronoun reference and ellipsis; if some item(s) in a request have been left unspecified or replaced by pronouns, the context register values from the previous request are used to supply the missing information or the referent of a pronoun (see example 3). There are also context registers for the last request, last paraphrase, last query language form, and last answer. Context registers are implemented as stacks which are pushed down with each new request. While we have not yet done so, we could conceivably modify PLANES to retrieve an earlier context by saying something like:

(Q6) Earlier we were talking about skyhawks.

Time is handled in a special manner. Since time phrases can occur in any question, time phrases are not considered part of the patterns (see Fig. 4) but are searched at the beginning or end of a request, or after the verb. There are special subnets and context registers for time phrases.

Quantifiers (e.g. "first", "rest", "more than", "largest", etc.) are handled by a special subnet as are qualifiers (e.g. the underlined words in "A7s which crashed in May").

"Noise words" are matched and essentially discarded. By "noise words", we mean phrases like "please tell me", "can you tell me", "would you let me know", "could you find", etc. The next section includes a description of the processing of such phrases.

A paraphrase of the first pattern found which matches the input is fed back to the user for his approval, and if the match is correct, the action associated with this pattern is executed. If the first match is incorrect, PLANES currently fails; eventually a more sophisticated presentation to the user of alternative interpretations will be implemented.

Example 3

Let us now follow our example through; the input to the matching portion from the previous portion is:

(Q2) Which ones (log past) between 10 and 20 flighthours in January?

This request matches pattern P1 in Figure 4, and causes context registers to be set as follows:

register	value
*oper	FINDALL (due to which)
*planetype	(pronoun ones)
*obj	flighthours
*quant-obj	(>10,<20)
*time-period	January

There is an important thing to note here: the system has solved the pronoun reference problem for the phrase "which ones" by finding the phrase type (i.e. *planetype) which makes sense in the context of the given sentence. There is only one pattern which matches "which ?X log flighthours": the pattern "which *planetype log *flighthours." This allows us to conclude that ?X refers to *planetype. (While pilots could log flight hours, there is no pilot data in our world.) Each pattern can thus be viewed as representing a simple meaningful "concept" which the system knows. Thus whenever constituents of a sentence are missing (as in ellipsis) or replaced by pronouns or referential phrases, the system is able to suggest what type of phrase is necessary to complete the concept by finding all the patterns which match the rest of the sentence. If only one matches, we are done; if more than one matches, then reference to which constituents were present in the previous sentence is usually adequate to decide; otherwise, the user can be given a set of possibilities from which to choose the appropriate referents for each phrase. The system can conclude that sentences like "How many malfunctions logged more that 10 flight hours." are meaningless because all phrases are recognized but no matching pattern exists.

Any system for answering questions which is to distinguish meaningful from meaningless requests must store information about which concepts are possible, whether by patterns (as in PLANES or in PARRY [Colby, et al. 1974]), by verb

meaning structures as in case grammars [Fillmore 1968, Celce-Murcia 1972, Bruce 1975], by conceptual dependency diagrams [Schank 1973] or by preference semantics systems [Wilks 1975].

Continuing with the example, once pattern P1 has been matched, the action A1 associated with P1 is retrieved. A1 in this case merely says to construct a paraphrase expression from the context register values, using the translator. The result is shown in figure 5.

```
(FINDALL (PLANE (PRONOUN T)
                (TYPE A7)
                (BUSER NIL))
         (NEG NIL)
         (FLY HOURS ((<20.)(>10.)))
         (TIME (DATE (MONTH (1. 0. 0.))
                     (DAY NIL)
                     (YEAR 73.))))
```

FIGURE 5 Paraphrase language representation of Q1.

This expression can be read as follows:

(1) FINDALL is a search function which means "find all items in the data base satisfying the specifications following." The context register *oper is set to FINDALL when the word "which" is matched in the original question.

(2) The three lines beginning with "PLANE ..." specify the item to be searched for and returned as a value, in this case A7s. (PRONOUN T) means that pronoun

(or phrase) reference occurred in the request. Note that the value "A7" has been inserted for plane type. This value was obtained from the context register values from the previous sentence, and thus completes the handling of pronoun reference for the phrase "which ones". (BUSER NIL) means that no specific serial numbers of planes appeared in the request.

(3) (NEG NIL) means that the sentence is not negated.

(4) The three lines beginning "(TIME ...)" specify that the entire first month of 1973 is to be considered. Here again the value of the year (1973) was not specified in the request, and so had to be obtained from the *time-period context register value of the previous request. This is another example of the handling of ellipsis (or automatic filling in of information understood in the context of the question) where the relevant context here is the preceding dialogue. It should be clear that similar mechanisms are used to handle both ellipsis and pronoun reference problems.

In our current system, the entire "paraphrase" is fed back to the user. In the near future, we intend to feed back an English version of this information, e.g. "Find all planes which had less than 20 and more than 10 flight hours during January 1973."

5.1.5 Implementation of Prestored Request Patterns and Subnets

Noun phrase parsing is based on Winograd's (1972) analysis of noun phrases. Both prestored request networks and subnets are stored as ATNs.

Instead of storing and matching against a list of patterns (like the arrangement shown in Figure 4), the patterns are actually organized into a network structure, a portion of which is shown in Figure 6. The network

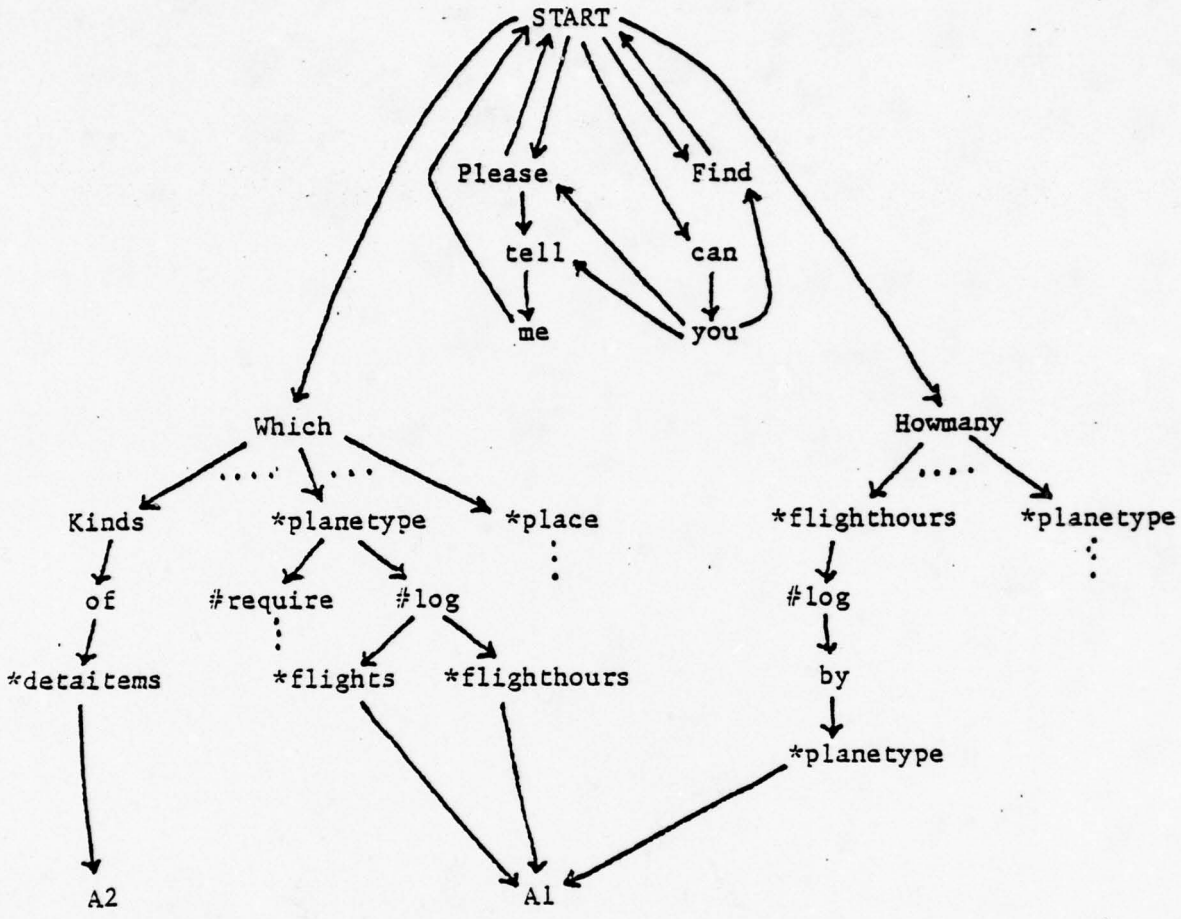


FIGURE 6 Simplified portion of the prestored request network.

structure (most of which is a tree) allows the initial portions of similar patterns to be merged; at the point where two patterns differ, the network branches into separate paths. The ends of these paths point to the appropriate actions.

The network portion of which Figure 6 is a part, is called the SENTENCE network. It guides the overall processing and calls the various subnets as specialists on various phrase types. Currently the entire network consists of approximately 700 states.

5.1.6 Translation into Query Language

The paraphrase language expression is next translated into a formal query expression for use with our relational data base system. The translation involves:

(1) Deciding which card types to look at to retrieve the information necessary for answering the user's request;

(2) Deciding what data fields to return from the cards which are searched. (In general, more fields are returned than are actually asked for. For example, if asked about which planes had engine maintenance during some time period, the system returns not only the plane identification numbers, but also the dates of maintenance and codes for the exact type of maintenance.);

(3) Deciding how to arrange the output data. Typical orderings are by increasing or decreasing size of some field value (like number of hours down time) or sequentially by date.

(4) Deciding which operations should be performed on the fields returned. Examples of operations include list, count, average, sum, and find largest;

(5) Translating field values (e.g. for dates, plane types, or actions) into internal data base codes;

(6) Most important, organizing all this material into an expression in the relational calculus [Codd 1971b, Date 1975, Green 1976] which can be used to implement the actual data base search.

5.1.7 Handling of Pronoun Reference and Ellipsis

PLANES I is able to handle pronoun references to the previous request only. This is accomplished relatively easily and cheaply by simply saving the card images found during the last query of the data base in a temporary file. Thus, when a pronoun is parsed, the query language is generated to search the temporary file only. Thus, for example:

(1) "How many flights did plane 48 make during Dec. 1969?"

Paraphrase generated:

```
(COUNT (plot NIL)
       FLIGHTS
       NIL
       (neg NIL)
       (FLY (plane (pronoun NIL) (type PLANE)
                   (buser 48.))
            NIL)
       (time (date (month (12. 334. 335.))
                   (day NIL) (year 69.))
             NIL)
       (NIL NIL))
```

Query language generated:

```
(FIND 'ALL
      '((V 0))  ["0" is the type of file to search]
      '((SUM (V TOTFLTS)))
      '(AND (EQU (V ACTDATE) 912.)
            (EQU (V BUSER) 48.))
      'NIL)
```

(2) "How many flight hours did it log?"

(Thus, "it" refers to "plane 48" in (1).)

Paraphrase generated:

```
(COUNT (plot NIL)
       TIME
       NIL
       (neg NIL)
       (FLY (plane (pronoun IT)
                   (type NIL) (buser NIL))
            NIL)
       (time (date (month (12. 334. 335.))
                   (day NIL) (year 69.))
             NIL)
       (NIL NIL))
```

Query language generated:

```
(FIND 'ALL
      '((V 00031))   [this causes a search of only the
                      temporary file 00031 to take
                      place.]
      '((SUM (V TOTHRS)))
      '(AND (EQU (V ACTDATE) 912.)
            (EQU (V BUSER) 48.))
      'NIL)
```

One can see that the use of temporary files in handling pronoun references can be very efficient since it saves searching through a great number of cards that had been previously searched. However two limitations occur in the current implementation: (1) pronouns can only reference noun groups in the previous request; and (2) should a request use a pronoun reference to the last sentence but ask for information that is contained in a different set of cards than those in the temporary file, no answer could be found. As an example, consider

(1) "How many flights did plane 48 make in April 1973?"

This would form a temporary file of the monthly summary card for April 1973 for plane 48.

If we followed with:

(2) "Which damage occurred to it?"

This could not be answered since the temporary file <u>does not</u> contain damage

codes.

We wish to solve these problems without giving up the time advantage introduced by the temporary files. It should be relatively easy to solve the second problem. We can simply construct a table of fields present on each type of data card. Then if the field we must key on due to the user's request is not present on the cards in the temporary file, we will know that we must search the data base directly.

The solution to the first problem will be somewhat harder. It will require the saving of the past history of all paraphrases, queries, and temporary files. These can be saved on a stack (possibly of fixed size) to allow us to examine the most recent entries first. To hypothesize what the pronoun refers to, we must decide what category it is applicable to (simply examine the current paraphrase for this), and then find the last non-pronoun entry that was made in this category. Further examination of the paraphrase and its query language request can be used to either reinforce or deny our hypothesis. If too many conflicts appear, we may search further back, otherwise we will assume that we have found the correct pronoun referent. When the conflicts cannot be resolved, we can ask the user to tell us which one of a number of possible choices the pronoun stands for. Once a reference point has been found to an object in the past, we can examine all temporary files back to that point to see if they can be used to answer our request. If not, we can still search the data base directly.

This method will also give us another advantage. We should be able to change the current context back to an earlier environment; e.g. by stating:

"Earlier we were talking about Skyhawks."

We can search back through the paraphrases, query requests, and temporary files

for a reference to "Skyhawks", and then be ready to use previous information over again.

Storing the context registers in a stack makes the saving of previous paraphrases relatively easy. It also makes ellipsis relatively simple to handle. We simply fill in any missing--but required--information by finding the last non-nil entry of a particular context register. Thus if given the sentences:

(1) "Tell me all the malfunctions found on the A7 with buser 12 on Feb. 15, 1970."

(2) On Feb. 16, 1970."

the system can generate a paraphrase for the second request by copying all context registers from the last request except the one for the time period. The new period is then used. As another example, if given

(3) "How many A7's had greater than 20 NOR hours on Feb. 15, 1970?" and

(4) "How many had greater than 30?"

all context registers for the first request are copied except the ones used to quantify NOR hours. This number is filled in from the new request.

The detection of missing information occurs by determining if a sentence has run out of words prematurely or if a verb appears where a noun phrase is expected. The beauty of the transition network is that even if a mistaken

ellipsis is made, the parsing process continues, and the mistake will most likely be detected further into the parsing process.

Please refer to the appendix for numerous examples of queries that PLANES I can handle.

5.2 PLANES II

5.2.1 Why PLANES II?

Our first approach to the problem of isolating a user's query and translating it into a reasonable predicate calculus query language was based on a naive fact. We felt that with the proper use of subnets, we could build a tree structure using ATN machinery that represented most of the paraphrases of questions anyone cared to ask about the Navy's 3-M data base. Our hope was that all paraphrases of a particular question could be mapped in such a way that they merged at the bottom of the tree. By collecting useful information--and storing it in context registers--we hoped that a query could be generated by a program stored at the node where all the paraphrases merged. The system we built worked well and we even developed a program that could automatically add sentences to the network [Hadden 1977]. The problem we ran into was that our overall net began growing enormously in size, while at the same time we found, to our chagrin, that our understanding system had trouble answering even up to 25% of an unsophisticated users questions.

5.2.2 Concept Case Frames to the Rescue

The problem that confronted us was clear--must we give up this semantic approach, utilizing the question forms as its base, for a more classical syntactic approach. The syntactic approach offered the ability to forget about enumerating the kinds of questions and to approach the problem in a more general way. However it violated the basic engineering approach that we had vowed to follow--namely to develop a strategy for allowing relatively unrestricted natural language queries of a data base while at the same time making it readily transportable to other data bases. We already had a general purpose predicate calculus language based upon Codd's work that could easily be moved between different relational data bases. The rigid nature of a syntactic parse would not allow a user the freedom to be sloppy, to abbreviate ideas, and utilize ellipsis and pronoun reference in a non-trivial manner. Our network of sentences allowed this freedom but only on those abbreviated queries that we foresaw to code into the net.

The information we needed to build a query was there, hidden in a user's abridged request. What we needed was a means for extracting the useful information required for generating a query and filling in any other necessary information. Extracting the useful information seemed easy. The remnants of our first system contained numerous ATN subnets used for parsing descriptions of planes, various maintenance actions, damages, etc. These same subnets could be employed to remove useful information from a user's input by matching against phrases with specific meanings. However what we are left with after parsing the various phrases are several unconnected semantic features and values. Concept case frames provide a connection. Concept case frames are patterns which describe the information that is stored in the data base and the legal requests

about the data. They can be used for judging meaningfulness of questions, generating dialogue for clarifying partially understood questions, resolving ellipsis and pronoun reference, and providing a pointer to special purpose interpretation routines for certain types of questions.

5.2.3 The System

5.2.4 Parsing

The generation of an answer to a user's query follows through several phases. The first phase is a parsing phase. Here the phrase parsers each sequentially scan from an anchored position at the left end of a string to the right to see if they can find a match. Once a match is found, the result is stored in some internal form, and the anchor is then moved further into the string. If no match is found, the current word is tagged as unparsed and the anchor is moved forward one word. This process is repeated until the end of the sentence is reached. At the end of this phase we are left with a set of representations of the semantic contents of the phrases that comprise the sentence. These categories include plane types, maintenance actions, damages, actions(verbs), quantifiers(which are attached to a particular noun phrase), and time periods.

With this scheme one can see that the order of the phrases in the sentence is relatively unimportant. For this reason pronoun reference and ellipsis become a difficult task. The parsing of a pronoun does not result in its attachment to a particular semantic category. Ellipsis may be taking place but it is hard to be sure. (One clue for ellipsis is finding a question word

followed by an action. This sometimes indicates that it is occurring--namely the noun phrase normally expected in that position is missing. The probable ellipsis can be tagged so that further investigation can be taken on later.)

5.2.5 Concept Case Frames

At this point we have a set of representations of the semantic contents of phrases, some pronouns, and tags for possible ellipsis. Somehow we need to tie these together in such a way that we can verify that we have a meaningful request and to enable us to resolve the pronoun and ellipsis references. This is where our concept case frames come into play. They provide the rigorous framework that we can use to extract meaning from a user's request. By inspection of the case frame matching the query best, it is possible to find entities that could be referred to by pronouns and/or ellipsis and the relationships between these elements.

Each concept case frame consists of the act (typically related to the verb) and a list of noun phrases (referred to by subnet/context register name) which can occur meaningfully with the act. Unlike ordinary case frames [Bruce 1975], we do not store information about the role (e.g. agent, patient, object) played by various phrases and each act covers a number of related verbs. Each concept case frame is a template representing a whole series of questions about the data base. Queries addressed to the system are mapped to these templates so that meaning can be extracted. The templates are used as tools to "build" a legal query by filling in the "slots" of the template with information extracted from the current request, earlier requests, world knowledge or default values.

The concept case frame is divided into two distinct sections--one for use as a template for comparison with user's queries and the other for translating the queries into our query language. The template part is constructed as follows:

i) Take a question form that classifies a group of questions about the data base

ii) Find the actions(from our group of legal actions) that are used in the sentences

iii) Enter in the case frames of those actions a list of the semantic groups that are in the template

iv) Arrange the semantic groups to show the relationships between different elements.

A sample template for the action REPAIR is shown below.

```
(REPAIR
        ( ((N1 (TIMES DATES)) (N2 (FAILURE SYSTEMS)))
          (V2 (REPAIR))
          *PLANETYPE
          (?opt (*PLANETYPE))
          (V1 (REPAIR))
          *TIMEPERIOD )
        (<a mapping by question word into a table of query skeletons>)).
```

The list can be interpreted as follows:

i) The list ((N1 (TIMES DATES)) (N2 (FAILURE SYSTEMS))) represents the kinds of noun phrases that can go into the first slot in the template. The markers N1 and N2 are used to classify the different structures of the

query that can result depending on which noun appears.

ii) (V2 (REPAIR)) means that if any noun from N2 appears, the action REPAIR must be placed in this slot. Otherwise this slot is left empty.

iii) *PLANETYPE means a description of a plane must be filled in.

iv) (?opt (*PLANETYPE)) means that an optional plane description could be inserted here. This usually is a pronoun used in cases like "For plane 3, which engine systems were repaired on it during June 1973?".

v) (V1 (REPAIR)) means that if any noun from N1 appears, the action REPAIR must be placed in this slot. Otherwise the slot is left empty.

vi) *TIMEPERIOD means a time period over which the query is specified must be put into the slot here.

The template for REPAIR thus represents the following sentence types:

```
(TIMES *PLANETYPE (V REPAIR) *TIMEPERIOD)
(TIMES *PLANETYPE (V REPAIR) *PLANETYPE *TIMEPERIOD)
(DATES *PLANETYPE (V REPAIR) *TIMEPERIOD)
(DATES *PLANETYPE (V REPAIR) *PLANETYPE *TIMEPERIOD)
(FAILURE (V REPAIR) *PLANETYPE *TIMEPERIOD)
(FAILURE (V REPAIR) *PLANETYPE *PLANETYPE *TIMEPERIOD)
(SYSTEMS (V REPAIR) *PLANETYPE *TIMEPERIOD)
(SYSTEMS (V REPAIR) *PLANETYPE *PLANETYPE *TIMEPERIOD)
```

The query "For A7s, which engine systems were repaired on them during June 1973?" would be filled in something like:

```
(     (SYSTEMS (ENGINE))
      (REPAIR (PAST))
      (PLANE (SERIALNO NIL) (TYPE A7))
      (PRONOUN THEM)
      (NIL)
      (TIME ((MONTH 6.)) (DAY NIL) (YEAR 73.)))   ).
```

"For the engine systems, which A7s were repaired on them during June 1973?" would be mapped into the same structure. While the meaning of the sentences are not the same, the queries to the data base have the same intent, namely: DISPLAY "engine systems and A7s" from the repair files for June 1973. Hence they are mapped together.

However there are some cases where the meanings are much different. E.g. "How many maintenances were performed on the A7s in 1974?" and "How many A7s had maintenance performed in 1974?". In the first case we want to count the number of maintenances while in the second case the number of A7s. With the case frames as they stand we would have to require both questions to be answered. This actually adds no further difficulty since each question must search the same set of files and will have "hits" on the same records. It is possible to resolve conflicts like this by requiring stricter standards of input but, since no extra searching is involved, it probably is not worth it. We would also like to allow the user to input abbreviated requests like "TOTAL MAINTENANCE, A7s, 1974" even at the expense of producing a little extra information.

5.2.6 Resolving Pronoun Reference and Ellipsis

After the best concept case frame (i.e. the longest one that matches the most semantic categories of the phrases) has been chosen and as many slots in the template as possible are filled in, pronoun reference and ellipsis must be resolved. If all mandatory slots are filled and no pronouns occur, everything is already resolved. Pronouns are resolved by noting which semantic category slots are left over for them. Pronouns can be replaced with items that fit that same semantic category by scanning backwards through the context register values for earlier sentences. When ellipsis and pronouns occur at the same time, it is impossible to decide which of two or more slots to put a pronoun into. If the frame contains no optional slots, it makes no difference where the pronoun is put because the system will have to fill in any other empty slots before proceeding. Should optional slots be included, the task becomes more difficult because the pronoun may go in a required slot or an optional slot. If it should go in an optional slot, then ellipsis must be occurring for the required slot. If it fills the required slot, then ellipsis may or may not be occurring for the optional slot. The second case is the one that causes a problem--we must know if ellipsis is occurring to be able to extract the full meaning of the query. We resolve this problem by assuming that ellipsis is occurring and scanning the very recent set of past queries (say the last one or two sentences) to see if the context built up confirms our hypothesis (i.e. see if can find anything to fill the optional slot). If nothing fits the semantic category of the slot, the ellipsis is overruled and the hypothesis dropped. Otherwise we fill in the slot with the information found. If all else fails, we can ask the user to select the appropriate interpretation from among a set of hypotheses. (Being able to choose from among a small set of possibilities is still much simpler for the user than rephrasing!)

Ellipsis alone is a little easier to handle. Any time information is missing from required slots we know that ellipsis is occurring. The past queries can be scanned backwards for information to fill the slots. If we do not find enough information to fill all the slots we ask the user for the required information. When ellipsis of optional slots is occurring, we must use a set of heuristics to give us a clue that the omission of a phrase has occurred. Earlier we mentioned an example of such a heuristic--namely finding a question word followed by a verb other than the verb to be without any intervening noun phrase.

5.2.7 Generating the Query

The filled in concept case frame is next translated into a formal query expression for use with a relational data base system. The translation involves deciding which relations to search, what domains from those relations to return, how to arrange the output data, what operations to apply to the fields returned, and translating the field values into internal data base codes.

The general process for a simple query (one involving no joins, comparatives, listing by special grouping (e.g. by month, plane serial number, repair location, etc.), and no special quantification) is carried out by (a) finding the question phrase (e.g. which planes in "which planes flew more than 10 times?"), (b) inserting it in the general query skeleton pointed to by the case frame, (c) interpreting other semantic phrases and values as predicates, (d) inserting these in the general skeleton (e.g. adding (GT FLIGHTS 10.) to the list of predicates, given the question above), (e) completing the answer-description portion of the skeleton with other field values, using both

heuristic knowledge about meaningful answer forms and any special user instructions (e.g. "Plot...", "List...", etc.), and (f) deciding upon and filling in a answer sort specification (e.g. by increasing serial number, in time sequence, by groups according to portion of the plane repaired, etc.).

The query construction is more complex for quantified expressions (e.g. "Find all repairs for the 5 planes with the most flight hours), comparatives (e.g. "Did plane 5 log more flight hours than plane 3?"), questions with embedded clauses (e.g. "Which planes that crashed in May had engine maintenance in April?") and requests involving special operations (e.g. "Which plane flew the most hours in May?").

Of particular interest is the method by which relations to be searched are selected. The system looks at each phrase separately, and notes which relations the phrase could possibly belong to. Some phrases (like plane type and date) are not very useful for this process, since they appear in most relations, but others (like flighthours) appear in only one or two relations.[1] All the relations possible for a clause are then intersected, and if a single relation is selected, the process of selecting a relation is complete for the phrase. Clauses are then considered in pairs, and so on. If more than one relation or a set of relations remains, then the request is ambiguous, and a dialogue is necessary to select the appropriate relation. If no relations remain at any intersection step, then more than one relation must be searched to answer the request, and the results of these searches must then be combined via the

---

[1]Some phrases occur in many relations (e.g. "maintenance" could refer to fields WUC (Work Unit Code), ML (Maintenance Level), TM (Type Maintenance), and AT (Action Taken)) and other phrases refer to our own summations of several fields (e.g. NOR (Not Operationally Ready) which is the summation of NORS (NOR due to Supply), NORMSCH (NOR due to SCHeduled Maintenance) and NORMUNS (NOR due to UNScheduled Maintenance.)

relational operation called joining (constructing a single relation from two different relations). As an example, the request: "Find all planes which had engine maintenances on the same day as a flight" would require searching the maintenance and flight relations, and then joining these relations via the date and plane domains by intersecting the sets of tuples for maintenance and flight and retaining tuples with identical planes and dates.

Please refer to the appendix for numerous examples of queries that PLANES II can handle.

6

A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM

The development of the PLANES I and II systems has given us great insight into the problems of language processing for data base requests. We are particularly interested in those aspects of the system that could be easily transferred to other relational data bases. In this chapter we will tie those aspects in with a model for such a language system.

6.1 Language Processing Model for Data Base Requests

We have developed a model (described in [Waltz and Goodman 1977]) for converting a user's English request into a formal query language request for operation on a relational data base. The model has as its main steps (a) locating semantic constituents of a request; (b) matching those constituents against larger templates called concept case frames; (c) filling in the selected concept case frame using information from the user's request, from the dialogue context and from the user's responses to questions posed by the system; and (d) generating a formal data base query using the collected information. We have also suggested methods for constructing the components of the model for an arbitrary relational data base.

6.2 What Types of Problems is the Model Suited For?

The model is designed to handle requests by real, casual users, whose only programming language is English, but who have some knowledge of the material in

the data base.[1] We have assumed that users will ask questions which are often ungrammatical, which include abbreviations, both standard and non-standard, and which use ellipsis and pronouns extensively, etc. (see Malhotra (1975) for ideas about the types of things users are likely to type in).

The model is designed to work with a relational data base, in the relational model [Codd 1970, Date 1975] data is viewed as being divided into relations which correspond to files or sets of files in conventional data base terminology. Each relation contains a collection of tuples which correspond to records; each tuple contains one or more domains or fields. A relation can conveniently be thought of as a table, with each row being a tuple and each column a domain. There are two important reasons for using the relational approach.

(1) The relational approach stresses data independence. This means that the user and front end programs are effectively isolated from the actual data base organization. We are now working with only a small subset (approx. 10^8 bits) of a much larger (approx. 10^{11} bits) database; if we were to use our front end with the entire data base, the data accessing programs would have to be modified, but, using the relational model, these changes need not affect the "data model" seen by users and the natural language front end.

(2) Many data bases are already internally organized in a tabular form, and are thus suited to a relational data model.

Most of the examples given in this chapter are drawn from the world of the PLANES system.

---

[1] In the PLANES systems, we have provided easily accessible HELP files to bring a user without data base knowledge to a point where he can use the rest of the system.

6.3 Central Assumption

The central novel assumption underlying the model is that a data base request is uniquely determined by the set of semantic constituents in a clause, independent of the order of the constituents.[1] We need only sufficient grammatical correctness to recognize phrase boundaries of semantic constituents and to recognize clause boundaries (if any). Thus the model handles grammatical English, "pidgin English", or ungrammatical lists of semantic constituents with equal ease. To our knowledge the use of a method such as this for data base queries is original with this project.

Given our central assumption, the processing of a user's query involves primarily identifying all the semantic constituents, e.g. (for our data base) time period, plane type, serial numbers, maintenance types, etc. Each of these semantic constituents can be a variable (i.e. the name of a field for which the system is to find values) or a constant (i.e. a value for a particular field which a data item must satisfy), or a set of constants (i.e. a set of values or range of values). Thus, in "Which planes had engine maintenance in May 1973?" planes is a variable and engine maintenance and May 1973 are constants. An operation on a variable (e.g. "sum of flight hours") functions as a constant. The identification of semantic constituents is handled by a group of ATN subnets, each of which is an "expert" on recognizing different ways of expressing one type of semantic constituent; thus, there are subnet experts for time period, planetype, etc.

---

[1] There are important exceptions, for example comparative constructions such as "...did plane 3 have more flights than plane 2..." where the order of "plane 3" and "plane 2" in the sentence is important.

6.4 Concept Case Frames and Context Registers

A second major set of ideas is necessary to handle the following problem: often a user's request will omit information necessary to form an adequate query. This can happen because (a) a user leaves out information meant to be understood in context (this is called ellipsis), (b) a user uses a pronoun in place of a named constituent, or (c) a user simply neglects to include all the necessary information. To handle these situations we use Context registers together with concept case frames. Context registers are history keepers; they store all semantic constituents of requests along with answers to earlier questions and other information. A set of concept case frames enumerates the sets of semantic constituents which constitute meaningful queries. If the set of semantic constituents found in a user's request does not match a concept case frame exactly, we can look back through past context register values to fill in missing elements (as with ellipsis and pronoun reference), ask the user to pick the appropriate meaning if there is more than one possible way of filling in the concept case frame, or note that a join (combining of more than one relation) is necessary if there are elements left over after matching any single concept case frame. Certain concept case frames correspond to requests for HELP files or note that declarative information is being input (e.g. "From now on consider only plane 3."), or note that the request requires special processing, as in the case of a vague question or one requiring browsing functions. A good way to begin enumerating concept case frames is to look at the domains (field names) for each relation.

6.5 Query Generator

After the concept case frames have been used to resolve pronoun reference and ellipsis and to verify the semantic integrity of the request, it is necessary to generate a 'program' in our data base query language that can retrieve the actual data to answer the question. A valid query must include the set of relations to be searched, which fields to return, what operations to perform on those fields, how to sort the output, and a set of predicates expressing the semantic information in internal data base codes. We have already covered in a previous chapter how to select which relations to search. We will now consider how to construct the other parts of the query.

6.5.1 What Fields to Return and Operations to Perform on Them.

PLANES II uses the question word (e.g. which, what, etc.) of the request to index into a table of query skeletons pointed to by the concept case frame fitting the request. The entry in the table contains a list of fields to return for the request and the operations that are to be applied to those fields (e.g. SUM, COUNT, MIN, MAX, etc.). The entries are indexed by the qword (i.e. the question word) because it determines what form the output data should be.[1]

This method is convenient because it allows the flexibility of applying a set of user-created functions to fields on top of those built into the system. This permits one to "create fields" that do not really exist by doing arithmetic

---

[1] E.g. "How many <noun phrase> ..." suggests that either a summation over a field or the counting of unique "hits" should be returned. This is in contrast to "Which <noun phrases> ..." where you wish to return a list of the actual occurrences.

and Boolean combinations of entries already in the files.[1] However the use of a query skeleton table lacks any generality and requires user interaction--namely the creation of new entries (with the need to know a little about the query language itself), etc. We contend that it is possible to eliminate the table in favor of a more automated approach. Analysis of the entries in the table and the concept case frames that referred to them revealed that the noun phrase that follows the question word usually expresses some of the information that should be returned by the query. With the use of an association list composed of nouns and sets of fieldnames, we were able to select the fields to return by simply inspecting the noun phrase that follows the question word.[2] We also noted that the set of constituents to be returned are normally those which are variables or sets of variables. Those that are constants or sets of constants can be used to form the predicates.

The application of functions to the fields proposes a more difficult task. Quantifiers and qualifiers modify the "sense" of the noun in a noun phrase. Their presence can suggest the need for special processing of the fields that are to be returned. We currently parse out phrases such as quantifiers and qualifiers separately from the nouns they modify. We feel that by associating the phrases found together (i.e. quant+NP, NP+qual, qword+NP), we can extract enough meaning to know which fields require operations to be performed on them

---

[1] E.g. NOR (Not Operationally Ready) is the summation of several different fields--NORS (NOR due to Supply), NORMSCH (NOR due to SCHeduled Maintenance), and NORMUNS (NOR due to UNScheduled Maintenance).

[2] E.g. The general term "maintenance" could have associated with it the set of fields "WUC (Work Unit Code), ML (Maintenance Level), TM (Type Maintenance), and AT (Action Taken)."

and which special functions must be applied to those fields.

Phrases like "the first", "the largest", "the last", "the least", "the total", "the maximum", etc. are general enough to be translated into functions that can be applied to any field in the data base that contains numerical values. However phrases like "the worst", "the worst five", "the best", etc. can impose different criteria depending on the field they refer to (and also depending on the intention of the user). These could be managed by a class of special functions. However it would seem more appropriate to pass these questions onto either a more complex information system (which we call BROWSER [Conrad 1976]) or to a HELP system so that the user can define what he means in his own terms. Such actions could be triggered by a message attached to the concept case frame. This method could also apply other exceptions to the general routine.

Examples

1. "Find the total number of flights flown by the A7s in 1974 by month."

   {"the total number of"} {"flights"} implies that the flights field should be summed up over the time period. "By month" requests that this summation occur separately for each month.

2. "For the worst five planes ..."

   {"the worst five"} {"planes"}

   The "worst" trait has a different meaning depending on the field it is applied to ("plane" here) and the aim of the current user of the system. This question should be handled by a BROWSER-like system and HELP routines which allow the user to model his understanding of the term "worst".

6.5.2 Sorting the Output

It should be relatively easy to develop a set of heuristics to decide which field to sort.

a. If one is looking for the "maximum", "minimum", "first five", etc. of a particular field, then that field should be chosen to sort over.

b. Should the date field be returned, sort over it.

c. If plane serial numbers are returned, they can be sorted.

d. The user should be able to override the heuristics by explicitly stating which field to sort over.

6.5.3 Predicate Generation

The predicates (which are used to decide which instances are "hits") are generated by the query generation routines from the noun phrases and the time period context registers. One needs a set of LISP functions that have knowledge of specific field names and of the domain of items that go in those fields, and an association list of field names and English description of those fields. When specific values or a range of values are given for a particular field, a predicate is generated to depict this. Relationships between different entities in the query are also represented by predicates.

Example "total flights of plane 7 greater than total flights of plane 8":

Let V1 be a relation representing plane 7 and V2 be a relation representing plane 8. Then

```
(AND (EQU (V1 BUSER) 7)
     (EQU (V2 BUSER) 8)
     (GT (V1 TOTFLTS) (V2 TOTFLTS)))
```

represents the essence of the English phrase.

6.6 Constructing a Paraphrase

An important part of any query system's operation is allowing a user to verify whether or not the system has correctly understood his request. To this end, the system feeds back its understanding of the request, with pronoun reference and ellipsis resolved, for the user's approval. The paraphrase is straightforwardly constructed from the query, and any special information associated with the matched concept case frame. "Special information" includes query language skeletons, calls to HELP files, and special functions, such as statistical comparison functions, needed to answer complex questions.

If the user does not approve of the interpretation of his request, he can enter into a clarifying dialogue with the system [Codd 1974]. The system asks whether the user wants this query executed on the data base, if he wishes to continue with the current sentence as context (this is useful for correcting minor errors, e.g. typing the wrong year), or with the previous sentence as context. As a simple example, suppose a user wanted data for January 1972, but typed "...January 1973" instead. It is simple to correct this by typing "n" when asked by the system "Shall I execute this query on the data base...y or n?", and then simply typing "1972." The system will recognize this as a year, substitute it for 1973, and the user can then have the corrected query executed. Clearly, minimum typing and no remembering of special commands is required for

this sort of error correction.

6.7 Embedded Clauses

Qualifying phrases or qualifiers (see Winograd [1972]), constitute the most common type of dependent clause. Examples of qualifiers are the underlined parts of "planes which crashed in May", "maintenances performed on A7s", and "planes with poor maintenance records." Qualifiers appear after the main noun in a noun phrase, and are often introduced by relative pronouns such as "which" or "that", or by verb forms ending in -ed or -ing. Prepositional phrases can also serve as qualifiers.

Qualifiers can be found by applying a qualifier subnet to the portion of a request following the main noun of a noun phrase. Because qualifier syntax is fairly restrictive, in many cases merely examining the single word after the main noun may suffice to preclude the presence of a qualifier. If a qualifying phrase or clause may be present, the following actions are taken:

(1) A syntactic parser (based on Woods' LSNLIS parser [Woods, et al. 1972]) is invoked to find if a qualifier is present, and if so what its boundaries are.

(2) If the embedded clause is not grammatical, heuristics are invoked to attempt to bracket the clause.

(3) Once a qualifier is found to be present, processing is suspended on the current clause, and the current context register values are pushed down.

(4) The main noun from outside the qualifying phrase is substituted for the relative pronoun (if any) or is inserted as a phrase element in the qualifier.

(5) The qualifying phrase or clause is processed like a normal request, with the main noun from the clause above serving the role of the requested item.

Note that verb forms get changed to a root plus an inflection, so that the exact verb form does not affect this processing. Prepositions as well as verbs can refer to certain case frames, so that, for example "planes with poor maintenance records" has the same meaning to the system as "planes having poor maintenance records."

(6) The query corresponding to the qualifier must be integrated with the query corresponding to its surrounding clause to form the overall query. The ordinary meaning of qualifiers seems to suggest that the qualifier query be evaluated on the data base first, and its result should then be used as the scope of search for the other clauses. In fact, either search can in general be performed first.[1]

Notice that requests can involve searching more than one relation, even if there are no qualifiers or dependent clauses, as in the sentence: "Did any planes have a flight on the same day as an engine maintenance?" In this case the relation for flights and the relation for maintenances must both be searched, and the results joined with respect to plane and date values.

6.8 Adding New Questions

Extending system competence within the model is particularly easy. To add the ability to handle a new type of sentence, one must only add a concept case frame which expresses that sentence. Variations on this sentence, including

---

[1]PLANES estimates temporary storage required for each query, and selects the query with minimum requirements to search first. The storage estimates are made on the basis of statistical information stored for each file. Query construction is discussed in more detail in [Waltz, et al. 1976] and [Green 1976].

active and passive forms, ellipsis, different phrase orderings and the addition of noise words can all be handled with no additional machinery, provided that the proper relation (or relations) is automatically selected by the translation mechanisms discussed above. This can be handled in ordinary user mode. If a novel request is encountered (i.e. one which does not match any concept case frame), the query generator can still attempt to handle the request, and feed back a paraphrase to the user. If the user approves of the paraphrase, the request could be added to the concept case frame list. The request can be added in a general way only if there were no pronoun reference and no ellipsis in the request--if there were, the concept case frame generated would be incomplete. If special instructions are necessary, these are attached to the concept case frame, but this process probably will involve a programmer for the foreseeable future.

Extending the subnets is also easy; we have written a net editor (described in [Hadden 1977]) which takes a phrase and adds the states and arcs to match this phrase to a specified subnet, using a minimum number of new arcs and states. Once a new phrasing has been added to a subnet's repertoire, the phrase can of course be matched in any sentence context. To add a new phrase, in user mode, the new phrase must be synonymous with some already known phrase.

APPENDIX A
Sample Queries for PLANES I

```
; Archive file of 7/14/77 at 11:0:26

Please enter your question.....
  >> how many flights did plane 48 make during dec 1969?


parsing......
 {cpu time was 1.86 seconds, real time was 3.65 seconds.
I have understood your request as:
     (COUNT (PLOT NIL)
      FLIGHTS
      NIL
      (NEG NIL)
      (FLY (PLANE (PRONOUN NIL)
                  (TYPE PLANE)
                  (BUSER 48.)
                  (PLNEG NIL)
                  (PLDAM NIL)
                  (PLMAI NIL))
           NIL)
      (TIME (DATE (MONTH (12. 334. 335.)) (DAY NIL) (YEAR 69.))
            NIL)
      (NIL NIL))

Interpreting.....
 {cpu time was 0.47 seconds, real time was 1.4 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((SUM (V TOTFLTS)))
     '(AND (EQU (V ACTDATE) 912.) (EQU (V BUSER) 48.))
     'NIL)

Evaluating.....
 {cpu time was 4.33 seconds, real time was 6.31 seconds.
       ((SUM TOTFLTS) = 24.)
```

```
; Archive file of 7/14/77 at 11:7:42

Please enter your question.....
  >> how many flighthours did it log?


parsing......
 {cpu time was 3.92 seconds, real time was 19.23 seconds.
I have understood your request as:
     (COUNT (PLOT NIL)
      TIME
      NIL
      (NEG NIL)
      (FLY (PLANE (PRONOUN IT)
                  (TYPE NIL)
                  (BUSER NIL)
                  (PLNEG NIL)
                  (PLDAM NIL)
                  (PLMAI NIL))
           NIL)
      (TIME (DATE (MONTH (12. 334. 335.)) (DAY NIL) (YEAR 69.))
            NIL)
      (NIL NIL))

Interpreting.....
 {cpu time was 0.52 seconds, real time was 1.96 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 00260))
     '((SUM (V TOTHRS)))
     '(AND (EQU (V ACTDATE) 912.))
     'NIL)

Evaluating.....
 {cpu time was 0.85 seconds, real time was 2.93 seconds.
       ((SUM TOTHRS) = 42.)
```

```
; Archive file of 7/14/77 at 11:18:31

Please enter your question.....
  >> did it have > 50 nor hours ?


parsing......
 {cpu time was 3.19 seconds, real time was 15.83 seconds.
I have understood your request as:
     (FINDONE (PLOT NIL)
        (PLANE (PRONOUN IT)
               (TYPE NIL)
               (BUSER NIL)
               (PLNEG NIL)
               (PLDAM NIL)
               (PLMAI NIL))
        NIL
        (NEG NIL)
        (NOR HOURS ((> 50.)))
        (TIME (DATE (MONTH (12. 334. 335.)) (DAY NIL) (YEAR 69.))
              NIL)
        (NIL NIL))

Interpreting.....
 {cpu time was 0.62 seconds, real time was 1.83 seconds.

I have interpreted your request as follows:
     (FIND '1.
     '((V 00260))
     '((V ACTDATE) (V (NOR)))
     '(AND (EQU (V ACTDATE) 912.) (GT (V (NOR)) 50.))
     'NIL)

Evaluating.....
 {cpu time was 1.5 seconds, real time was 25.18 seconds.
  gc time was 6.64 seconds.

ACTDATE NOR
--------
   912. 100.
```

```
; Archive file of 7/14/77 at 11:23:21

Please enter your question.....
  >> how many flights did plane 3 make in Feb. 73 ?



parsing......
 {cpu time was 2.14 seconds, real time was 4.61 seconds.
I have understood your request as:
     (COUNT (PLOT NIL)
      FLIGHTS
      NIL
      (NEG NIL)
      (FLY (PLANE (PRONOUN NIL)
                  (TYPE PLANE)
                  (BUSER 3.)
                  (PLNEG NIL)
                  (PLDAM NIL)
                  (PLMAI NIL))
           NIL)
      (TIME (DATE (MONTH (2. 31. 31.)) (DAY NIL) (YEAR 73.)) NIL)
      (NIL NIL))

Interpreting.....
 {cpu time was 0.59 seconds, real time was 1.8 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((SUM (V TOTFLTS)))
     '(AND (EQU (V ACTDATE) 302.) (EQU (V BUSER) 3.))
     'NIL)

Evaluating.....
 {cpu time was 2.91 seconds, real time was 12.13 seconds.
(((SUM TOTFLTS)) = (1.0))
```

```
Please enter your question.....
  >> which a7s had > 100 normu hours between april 73 and may 73 ?


parsing......
 {cpu time was 1.54 seconds, real time was 2.23 seconds.
I have understood your request as:
     (FINDALL (PLOT NIL)
        (PLANE (PRONOUN NIL)
               (TYPE A7)
               (BUSER NIL)
               (PLNEG NIL)
               (PLDAM NIL)
               (PLMAI NIL))
        NIL
        (NEG NIL)
        (NORMU HOURS ((> 100.)))
        (TIME (DATE (MONTH (4. 90. 91.)) (DAY NIL) (YEAR 73.))
              (DATE (MONTH (5. 120. 121.)) (DAY NIL) (YEAR 73.)))
        (NIL NIL))

Interpreting.....
 {cpu time was 0.61 seconds, real time was 1.84 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((V ACTDATE) (V BUSER) (V NORMUNS))
     '(AND (GEQ (V ACTDATE) 304.)
           (LEQ (V ACTDATE) 305.)
           (EQU (V TEC) (QUOTE AAFF))
           (GT (V NORMUNS) 100.))
     '(ACTDATE DOWN))

Evaluating.....
 {cpu time was 16.46 seconds, real time was 62.73 seconds.

ACTDATE BUSER NORMUNS

------------------------
   305.    4.    162.
   304.   11.    133.
   304.   10.    136.
   304.    5.    213.
   304.   43.    118.
```

```
; Archive file of 7/14/77 at 11:31:9

Please enter your question.....
  >> where was plane 3 repaired between november 1 and december 30 1972 ?

parsing......
 {cpu time was 2.01 seconds, real time was 9.71 seconds.
I have understood your request as:
     (WHERE (PLOT NIL)
      (PLANE (PRONOUN NIL)
             (TYPE PLANE)
             (BUSER 3.)
             (PLNEG NIL)
             (PLDAM NIL)
             (PLMAI NIL))
      NIL
      (NEG NIL)
      (REPAIRED NIL NIL)
      (TIME (DATE (MONTH (11. 304. 305.)) (DAY 1.) (YEAR 72.))
            (DATE (MONTH (12. 334. 335.)) (DAY 30.) (YEAR 72.)))
      (NIL NIL))
Interpreting.....
 {cpu time was 0.69 seconds, real time was 1.83 seconds.
I have interpreted your request as follows:
     (FIND 'ALL
     '((W F))
     '((W ACTDATE) (W ACTWC))
     '(AND (GEQ (W ACTDATE) 2306.)
           (LEQ (W ACTDATE) 2365.)
           (EQU (W BUSER) 3.))
     '(ACTDATE DOWN))

Evaluating.....
 {cpu time was 5.78 seconds, real time was 19.16 seconds.

ACTDATE ACTWC
--------------
  2363.   210
  2361.   210
  2359.   615
  2357.   626
  2355.   631
  2353.   220
  2352.   615
  2340.   626
  2336.   140
  2335.   649
  2314.   630
  2313.   630
```

```
; Archive file of 7/14/77 at 11:37:11

Please enter your question.....
  >> what are all of the malfunctions found on the a7 with tail 3 between
     dec 1 and dec 10 1972 ?



parsing......
 {cpu time was 2.26 seconds, real time was 5.13 seconds.
I have understood your request as:
     (FINDALL (PLOT NIL)
        MALFUNCTION
        NIL
        (NEG NIL)
        (ON (PLANE (PRONOUN NIL)
                   (TYPE A7)
                   (BUSER 3.)
                   (PLNEG NIL)
                   (PLDAM NIL)
                   (PLMAI NIL))
            NIL)
        (TIME (DATE (MONTH (12. 334. 335.)) (DAY 1.) (YEAR 72.))
              (DATE (MONTH (12. 334. 335.)) (DAY 10.) (YEAR 72.)))
        (NIL NIL))

Interpreting.....
 {cpu time was 0.65 seconds, real time was 2.16 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((W F))
     '((W ACTDATE) (W BUSER) (W HOWMAL))
     '(AND (GEQ (W ACTDATE) 2336.)
           (LEQ (W ACTDATE) 2345.)
           (EQU (W TEC) (QUOTE AAFF))
           (EQU (W BUSER) 3.))
     '(ACTDATE DOWN))

Evaluating.....
 {cpu time was 4.3 seconds, real time was 11.8 seconds.

ACTDATE BUSER HOWMAL
-------------------
  2340.    3.    80.
  2340.    3.   169.
  2340.    3.   450.
  2340.    3.   900.
  2336.    3.   242.
```

```
; Archive file of 7/14/77 at 11:51:35

Please enter your question.....
  >> how many flights did plane 3 fly in 1973?



parsing......
 {cpu time was 1.51 seconds, real time was 7.15 seconds.
I have understood your request as:
     (COUNT (PLOT NIL)
      FLIGHTS
      NIL
      (NEG NIL)
      (FLY (PLANE (PRONOUN NIL)
                  (TYPE PLANE)
                  (BUSER 3.)
                  (PLNEG NIL)
                  (PLDAM NIL)
                  (PLMAI NIL))
           NIL)
      (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 73.)) NIL)
      (NIL NIL))

Interpreting.....
 {cpu time was 0.32 seconds, real time was 3.13 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((SUM (V TOTFLTS)))
     '(AND (GEQ (V ACTDATEMON) 1.)
           (LEQ (V ACTDATEMON) 12.)
           (EQU (V ACTDATEYR) 3.)
           (EQU (V BUSER) 3.))
     'NIL)

Evaluating.....
 {cpu time was 3.22 seconds, real time was 13.88 seconds.
(((SUM TOTFLTS)) = (39.0))
```

```
; Archive file of 7/14/77 at 12:5:23

Please enter your question.....
  >> between jan and feb



parsing......
 {cpu time was 1.4 seconds, real time was 6.49 seconds.
I have understood your request as:
     (FINDALL (PLOT NIL)
        (PLANE (PRONOUN NIL)
               (TYPE A7)
               (BUSER NIL)
               (PLNEG NIL)
               (PLDAM NIL)
               (PLMAI NIL))
        NIL
        (NEG NIL)
        (NOR HOURS ((> 10.)))
        (TIME (DATE (MONTH (1. 0. 0.)) (DAY NIL) (YEAR 72.))
              (DATE (MONTH (2. 31. 31.)) (DAY NIL) (YEAR 72.)))
        (NIL NIL))

Interpreting.....
 {cpu time was 0.59 seconds, real time was 3.81 seconds.

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((V ACTDATE) (V BUSER) (V (NOR)))
     '(AND (GEQ (V ACTDATE) 201.)
           (LEQ (V ACTDATE) 202.)
           (EQU (V TEC) (QUOTE AAFF))
           (GT (V (NOR)) 10.))
     '(ACTDATE DOWN))

 {cpu time was 21.93 seconds, real time was 123.96 seconds.
(OUTPUT SCHEDULED , THERE WERE 53. ITEMS)
```

```
; Archive file of 7/14/77 at 12:12:44

Please enter your question.....
  >> what kinds of codes are in the data base?


parsing......

HELP:

  THE FOLLOWING FILES CAN BE ACCESSED FOR HELP:

   FILE                              DESCRIPTION

------------      ------------------------------------------------------

ACTORG.DEF          GIVES THE ORGANIZATION CODES (ACTORG)
ACTWC.DEF GIVES THE ACTION WORK CENTERS (ACTWC)
AT.DEF              GIVES THE ACTION TAKEN CODES (AT)
BATTLE.DAM          LISTS THE HOWMAL CODES PERTAINING TO BATTLE DAMAGE
BIRD.DAM  LISTS THE HOWMAL CODES PERTAINING TO BIRD STRIKE DAMAGE
DAM.LSP             LISTS THE TWELVE DAMAGE AREAS IN THE HOWMAL CODES
ELECTRICAL.DAM      LISTS THE HOWMAL CODES PERTAINING TO ELECTRICAL DAMAGE
TRONIC.DAM          LISTS THE HOWMAL CODES PERTAINING TO ELECTRONIC DAMAGE
ENGINE.DAM          LISTS THE HOWMAL CODES PERTAINING TO ENGINE DAMAGE
FOREIGN.DAM         LISTS THE HOWMAL CODES PERTAINING TO FOREIGN OBJECT DAMAGE
GEN.DEF             GIVES THIS MESSAGE
GUNNERY.DAM         LISTS THE HOWMAL CODES PERTAINING TO GUNNERY DAMAGE
HARD.DAM  LISTS THE HOWMAL CODES PERTAINING TO HARD LANDING DAMAGE
HOWMAL.DEF          LISTS ALL THE HOWMAL CODES (ALL TWELVE DAMAGE AREAS)
LAUNCH.DAM          LISTS THE HOWMAL CODES PERTAINING TO LAUNCH(TAKE-OFF) DAMAGE
MECHANICAL.DAM      LISTS THE HOWMAL CODES PERTAINING TO MECHANICAL DAMAGE
MISC.DAM  LISTS THE HOWMAL CODES PERTAINING TO MISCELLANEOUS DAMAGE
PLN.DEF             LISTS ALL THE PLANES COVERED IN THE FILE
TIM.DEF             EXPLAINS HOW DATES ARE ENCODED
TM.DEF              GIVES THE TYPE-MAINTENANCE (MT) CODES
WEATHER.DAM         LISTS THE HOWMAL CODES PERTAINING TO WEATHER DAMAGE


  TO ACCESS THE ABOVE FILES:

          (1) CAN SIMPLY ASK A QUESTION CONCERNING THE

  TYPES, CODES, ETC. THAT IS BEING SOUGHT; OR


          (2) CAN PRINT THE FILES OUT DIRECTLY IN MONITOR

  MODE BY EXECUTING PRINT <NAME>.<EXT>    .

END-OF-FILE

 {cpu time was 14.75 seconds, real time was 46.83 seconds.
```

```
; Archive file of 7/14/77 at 12:19:7

Please enter your question.....
  >> what does howmal code 80 mean?


parsing......
(BURNED OUT LIGHT BULBS OR FUSES)
 {cpu time was 1.96 seconds, real time was 3.44 seconds.

Please enter your question.....
  >> what does howmal code 450 stand for?


parsing......
(OPEN)
 {cpu time was 1.93 seconds, real time was 3.24 seconds.

Please enter your question.....
  >> what does howmal code 242 stand for?


parsing......
(FAILED TO OPERATE OR FUNCTION /- SPECIFIC REASON UNKNOWN)
 {cpu time was 1.7 seconds, real time was 2.91 seconds.

Please enter your question.....
  >> what is the abbreviation for tail number?


parsing......
THE ABBREVIATION FOR BUREAU NUMBER/SERIAL NUMBER IS BUSER
 {cpu time was 2.49 seconds, real time was 6.68 seconds.

Please enter your question.....
  >> do colorless green ideas sleep furiously?


parsing......
 {cpu time was 0.6 seconds, real time was 1.68 seconds.
I could not understand your request Please rephrase it.
```

APPENDIX B
Sample Queries from PLANES II

```
; Archive file of 6/15/77 at 10:36:8
T
(TALK)

Please enter your question.....
  >> how many a7s flew > 10 flights in 1973?


parsing......
(*QWORD *NPHR1 *ACT1 *NPHR2 *TIMEPP)
 {cpu time was 5.32 seconds, real time was 46.1 seconds.}
I have understood your request as:
     ((HOWMANY)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE A7) (BUSER NIL)) NIL)
    (FLIGHTS ((> 10.)))
    (NIL NIL))
      (NEG NIL)
      (ACT (FLY NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 73.)) NIL)
      (NIL NIL))

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 1.72 seconds, real time was 21.41 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((COUNT (V BUSER)) (SUM (V TOTFLTS)))
     '(AND (GEQ (V ACTDATEMON) 1.)
           (LEQ (V ACTDATEMON) 12.)
           (EQU (V ACTDATEYR) 3.)
           (EQU (V PLANETYPE) A7)
           (GT (V TOTFLTS) 10.))
     'NIL)

Should I evaluate the query?
  type Y or N >> y

(((COUNT BUSER) (SUM TOTFLTS)) = (7. 971.0))
```

The next example demonstrates simple ellipsis by answering the last question for a new date.

```
Please enter your question.....
  >> in 1972?


parsing......
 {cpu time was 1.06 seconds, real time was 2.44 seconds.}
I have understood your request as:
     ((HOWMANY)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE A7) (BUSER NIL)) NIL)
     (FLIGHTS ((> 10.)))
     (NIL NIL))
      (NEG NIL)
      (ACT (FLY NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 72.)) NIL)
      (NIL NIL))

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 1.95 seconds, real time was 5.31 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((COUNT (V BUSER)) (SUM (V TOTFLTS)))
     '(AND (GEQ (V ACTDATEMON) 1.)
           (LEQ (V ACTDATEMON) 12.)
           (EQU (V ACTDATEYR) 2.)
           (EQU (V PLANETYPE) A7)
           (GT (V TOTFLTS) 10.))
     'NIL)

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 9.01 seconds, real time was 115.9 seconds.}
 {gc time was 27.76 seconds.}
(((COUNT BUSER) (SUM TOTFLTS)) = (32. 5894.0))
```

The pronoun reference and ellipsis routines that utilize the concept case frames are turned on. The next example demonstrates the resolution of pronoun references.

```
(SETQ SOLVE-ELLIP-PRON T)

Please enter your question.....
  >> how many did they fly in 1970?


parsing......
(*QWORD *NPHR1 *ACT1 *NPHR1 *ACT2 *TIMEPP)
 {cpu time was 3.42 seconds, real time was 8.2 seconds.}
I have understood your request as:
     ((HOWMANY)
      (PLOT NIL)
      (NP (ELLIPSIS NIL) ((PRONOUN THEY) NIL) (NIL NIL))
      (NEG NIL)
      (ACT (DO NIL) (FLY NIL) (NIL NIL))
      (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 70.)) NIL)
      (NIL NIL))
 {cpu time was 1.03 seconds, real time was 2.23 seconds.}
AFTER ELLIPSIS PRONOUN REF:
     ((HOWMANY)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE A7) (BUSER NIL)) NIL)
    (FLIGHTS NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (DO NIL) (FLY NIL) (NIL NIL))
      (TIME (DATE (MONTH NIL) (DAY NIL) (YEAR 70.)) NIL)
      (NIL NIL))

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 1.59 seconds, real time was 8.0 seconds.}
I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((COUNT (V BUSER)) (SUM (V TOTFLTS)))
     '(AND (GEQ (V ACTDATEMON) 1.)
           (LEQ (V ACTDATEMON) 12.)
           (EQU (V ACTDATEYR) 0.)
           (EQU (V PLANETYPE) A7))
     'NIL)

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 2.26 seconds, real time was 10.6 seconds.}
(((COUNT BUSER) (SUM TOTFLTS)) = (4. 176.0))
```

```
; Archive file of 6/15/77 at 11:12:29
T
(TALK)

Please enter your question.....
  >> which a7s flew > 12 flights during jun 72?


parsing......
(*QWORD *NPHR1 *ACT1 *NPHR2 *TIMEPP)
 {cpu time was 4.91 seconds, real time was 10.54 seconds.}
I have understood your request as:
     ((WHICH)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE A7) (BUSER NIL)) NIL)
     (FLIGHTS ((> 12.)))
     (NIL NIL))
       (NEG NIL)
       (ACT (FLY NIL) (NIL NIL) (NIL NIL))
       (TIME (DATE (MONTH (6. 151. 152.)) (DAY NIL) (YEAR 72.)) NIL)
       (NIL NIL))

 {cpu time was 0.65 seconds, real time was 2.2 seconds.}

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 2.05 seconds, real time was 6.25 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V 0))
     '((V BUSER) (V TOTFLTS))
     '(AND (EQU (V ACTDATEMON) 6.)
           (EQU (V ACTDATEYR) 2.)
           (EQU (V PLANETYPE) A7)
           (GT (V TOTFLTS) 12.))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 8.05 seconds, real time was 35.8 seconds.}

BUSER TOTFLTS
-------------
   6.     34.
   9.     17.
  10.     24.
  14.     20.
  15.     34.
  17.     21.
  18.     30.
  19.     13.
  20.     22.
```

Example demonstrates simple ellipsis. It repeats the last question for a different plane series.

```
Please enter your question.....
  >> which f4s?


parsing......
(*QWORD *NPHR1)
 {cpu time was 3.02 seconds, real time was 7.8 seconds.}
I have understood your request as:
     ((WHICH)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE F4) (BUSER NIL)) NIL)
    (FLIGHTS ((> 12.)))
    (NIL NIL))
      (NEG NIL)
      (ACT (FLY NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (6. 151. 152.)) (DAY NIL) (YEAR 72.)) NIL)
      (NIL NIL))
 {cpu time was 0.34 seconds, real time was 0.8 seconds.}
AFTER ELLIPSIS PRONOUN REF:
     ((WHICH)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE F4) (BUSER NIL)) NIL)
    (FLIGHTS ((> 12.)))
    (NIL NIL))
      (NEG NIL)
      (ACT (FLY NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (6. 151. 152.)) (DAY NIL) (YEAR 72.)) NIL)
      (NIL NIL))
Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 2.39 seconds, real time was 6.4 seconds.}
I have interpreted your request as follows:
     (FIND 'ALL
     '((V O))
     '((V BUSER) (V TOTFLTS))
     '(AND (EQU (V ACTDATEMON) 6.)
           (EQU (V ACTDATEYR) 2.)
           (EQU (V PLANETYPE) F4)
           (GT (V TOTFLTS) 12.))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 2.61 seconds, real time was 6.58 seconds.}

BUSER TOTFLTS

  40.     17.
  42.     15.
```

```
Please enter your question.....
  >> did plane 48 fly during may 1969?


parsing......
(*QWORD *NPHR1 *ACT1 *TIMEPP)
 {cpu time was 5.53 seconds, real time was 86.68 seconds.}
 {gc time was 15.63 seconds.}
I have understood your request as:
     ((DO)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (FLY NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY NIL) (YEAR 69.)) NIL)
      (NIL NIL))

 {cpu time was 0.54 seconds, real time was 1.55 seconds.}

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 2.01 seconds, real time was 6.01 seconds.}

I have interpreted your request as follows:
     (FIND '1.
     '((V O))
     '((V BUSER) (V ACTDATE) (V TOTFLTS))
     '(AND (EQU (V ACTDATEMON) 5.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     'NIL)

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 1.86 seconds, real time was 4.3 seconds.}

BUSER ACTDATE TOTFLTS

  48.    905.     24.
```

The example demonstrates pronoun reference. It is also used to build up a context for further queries to work in.

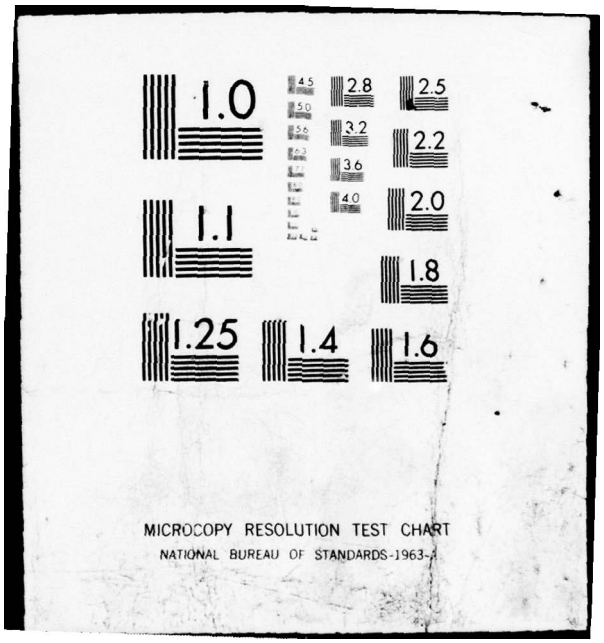
```
Please enter your question.....
  >> what maintenances were performed on it between may 16 and 17 1969?


parsing......
(*QWORD *NPHR1 *ACT1 *ACT2 *NPHR2 *TIMEPP)
 {cpu time was 9.86 seconds, real time was 72.81 seconds.}
 {gc time was 16.88 seconds.}
I have understood your request as:
     ((WHAT)
      (PLOT NIL)
      (NP ((MAINT (PRONOUN NIL)
            (TYPE-MAINT NIL)
            (ACTION-TAKEN NIL)
            (WORKUNITCODE NIL))
      NIL)
     ((PRONOUN IT) NIL)
     (NIL NIL))
       (NEG NIL)
       (ACT (PERFORM NIL) (ON NIL) (NIL NIL))
       (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.))
       (DATE (MONTH (5. 120. 121.)) (DAY 17.) (YEAR 69.)))
       (NIL NIL))

 {cpu time was 0.76 seconds, real time was 1.68 seconds.}

AFTER ELLIPSIS PRONOUN REF:
     ((WHAT)
      (PLOT NIL)
      (NP ((MAINT (PRONOUN NIL)
            (TYPE-MAINT NIL)
            (ACTION-TAKEN NIL)
            (WORKUNITCODE NIL))
      NIL)
     ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
     (NIL NIL))
       (NEG NIL)
       (ACT (PERFORM NIL) (ON NIL) (NIL NIL))
       (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.))
       (DATE (MONTH (5. 120. 121.)) (DAY 17.) (YEAR 69.)))
       (NIL NIL))

Should I translate this?
  type Y or N >> y
```

```
Interpreting.....
RELATIONS M F ALL POSSIBLE
WILL SELECT M
 {cpu time was 3.48 seconds, real time was 9.78 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V M))
     '((V BUSER) (V WUC) (V ML) (V TM) (V AT))
     '(AND (GEQ (V ACTDATEDAY) 136.)
           (LEQ (V ACTDATEDAY) 137.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 7.63 seconds, real time was 73.21 seconds.}
 {gc time was 17.12 seconds.}

BUSER      WUC ML           TM           AT
-------------------------------------------
  48. 6712300. 2. UNSCHED MAIN WSTOP PARTS
  48.  4511C00 1. UNSCHED MAIN REP LINE
  48. 9115200. 1. UNSCHED MAIN OEM REM REIN
  48.  6712F00 1. UNSCHED MAIN OEM REM REP
  48. 1221500. 1. UNSCHED MAIN OEM REM REIN
  48. 6712100. 2. UNSCHED MAIN REP
  48. 6712300. 1. UNSCHED MAIN OEM REM REP
  48. 7432200. 2. UNSCHED MAIN WSTOP PARTS
  48. 7432000. 1. UNSCHED MAIN REP
  48. 1333100. 1. UNSCHED MAIN OEM REM REP
  48.  1121C00 1. UNSCHED MAIN REP
```

AD-A057 641 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/2
A MODEL FOR A NATURAL LANGUAGE DATA BASE SYSTEM.(U)
OCT 77 B A GOODMAN N00014-67-A-0305-0026
UNCLASSIFIED R-798 NL

2 OF 2
AD
A057641

END
DATE
FILMED
9-78
DDC


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
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
     Demonstrates pronoun reference and simple ellipsis.

Please enter your question.....
  >> what damages occurred to it?



parsing......
(*QWORD *NPHR1 *ACT1 *NPHR2)
 {cpu time was 4.86 seconds, real time was 80.04 seconds.}
 {gc time was 16.53 seconds.}
I have understood your request as:
     ((WHAT)
      (PLOT NIL)
      (NP ((DAMAGE (PRONOUN NIL) (AREA ANY) (HOWMAL ANY)) NIL)
    ((PRONOUN IT) NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (OCCUR NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.))
      (DATE (MONTH (5. 120. 121.)) (DAY 17.) (YEAR 69.)))
      (NIL NIL))

 {cpu time was 0.59 seconds, real time was 1.18 seconds.}

AFTER ELLIPSIS PRONOUN REF:
     ((WHAT)
      (PLOT NIL)
      (NP ((DAMAGE (PRONOUN NIL) (AREA ANY) (HOWMAL ANY)) NIL)
    ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (OCCUR NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.))
      (DATE (MONTH (5. 120. 121.)) (DAY 17.) (YEAR 69.)))
      (NIL NIL))

Should I translate this?
  type Y or N >> y
```

```
Interpreting.....
RELATIONS M F R I ALL POSSIBLE
WILL SELECT M
 {cpu time was 2.71 seconds, real time was 7.44 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V M))
     '((V BUSER) (V HOWMAL))
     '(AND (GEQ (V ACTDATEDAY) 136.)
           (LEQ (V ACTDATEDAY) 137.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 6.04 seconds, real time was 12.46 seconds.}

BUSER          HOWMAL
_____________________
  48.            782.
  48. ADJ IMPROP
  48. NO DEF OTHER
  48. NOISY
  48. FAILED TO OPER
  48.            901.
  48. LOOSE
  48. SHORTED
  48. DIRTY
  48. LOCK MALF
```

Demonstrates pronoun reference and simple ellipsis.

```
; Archive file of 6/15/77 at 11:31:19
T
(TALK)

Please enter your question.....
  >> when was it repaired?



parsing......
(*QWORD *ACT1 *NPHR1 *ACT2)
 {cpu time was 5.27 seconds, real time was 16.35 seconds.}
I have understood your request as:
     ((WHEN)
      (PLOT NIL)
      (NP ((PRONOUN IT) NIL) (NIL NIL) (NIL NIL))
      (NEG NIL)
      (ACT (BE NIL) (REPAIRED NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.))
      (DATE (MONTH (5. 120. 121.)) (DAY 17.) (YEAR 69.)))
      (NIL NIL))

 {cpu time was 0.81 seconds, real time was 2.46 seconds.}

AFTER ELLIPSIS PRONOUN REF:
     ((WHEN)
      (PLOT NIL)
      (NP ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (BE NIL) (REPAIRED NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.))
      (DATE (MONTH (5. 120. 121.)) (DAY 17.) (YEAR 69.)))
      (NIL NIL))

Should I translate this?
  type Y or N >> y
Interpreting.....
RELATIONS M F R I ALL POSSIBLE
WILL SELECT M
 {cpu time was 1.98 seconds, real time was 5.78 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V M))
     '((V BUSER) (V ACTDATE) (V AT))
     '(AND (GEQ (V ACTDATEDAY) 136.)
           (LEQ (V ACTDATEDAY) 137.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     '(BUSER UP))
```

```
Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 5.97 seconds, real time was 12.71 seconds.}
BUSER ACTDATE           AT

-------------------------------

  48.   9136. REP LINE
  48.   9136. WSTOP PARTS
  48.   9136. OEM REM REP
  48.   9137. REP
  48.   9136. REP
  48.   9137. WSTOP PARTS
  48.   9136. OEM REM REIN
```

```
Please enter your question.....
  >> how many parts failed on plane 48 on may 16 69?


parsing.....
(*QWORD *NPHR1 *ACT1 *NPHR2 *TIMEPP)
 {cpu time was 6.27 seconds, real time was 70.08 seconds.}
 {gc time was 17.96 seconds.}
I have understood your request as:
     ((HOWMANY)
      (PLOT NIL)
      (NP (PARTS NIL)
    ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (FAIL NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.)) NIL)
      (NIL NIL))

 {cpu time was 0.54 seconds, real time was 1.28 seconds.}

AFTER ELLIPSIS PRONOUN REF:
     ((HOWMANY)
      (PLOT NIL)
      (NP (PARTS NIL)
    ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (FAIL NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.)) NIL)
      (NIL NIL))

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 0.76 seconds, real time was 2.31 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V F))
     '((COUNT (V JCN)))
     '(AND (EQU (V ACTDATEDAY) 136.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     'NIL)

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 3.34 seconds, real time was 57.86 seconds.}
 {gc time was 15.22 seconds.}
(((COUNT JCN)) = (1.))
```

The next several examples demonstrate simple ellipsis.

```
Please enter your question.....
  >> which parts?


parsing......
(*QWORD *NPHR1)
 {cpu time was 2.94 seconds, real time was 8.0 seconds.}
I have understood your request as:
     ((WHICH)
      (PLOT NIL)
      (NP (PARTS NIL)
    ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (FAIL NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.)) NIL)
      (NIL NIL))
 {cpu time was 0.43 seconds, real time was 1.01 seconds.}

AFTER ELLIPSIS PRONOUN REF:
     ((WHICH)
      (PLOT NIL)
      (NP (PARTS NIL)
    ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 48.)) NIL)
    (NIL NIL))
      (NEG NIL)
      (ACT (FAIL NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (5. 120. 121.)) (DAY 16.) (YEAR 69.)) NIL)
      (NIL NIL))

Should I translate this?
  type Y or N >> y
Interpreting.....
 {cpu time was 2.09 seconds, real time was 6.04 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V F))
     '((V BUSER) (V PARTNO))
     '(AND (EQU (V ACTDATEDAY) 136.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 3.28 seconds, real time was 118.03 seconds.}
 {gc time was 13.63 seconds.}

BUSER  PARTNO
------------
  48. AC900E6
```

Can tell the system to turn off the display some of the information.

```
; Archive file of 6/15/77 at 11:57:36
T
(TALK)

Please enter your question.....
  >> turn off paraphrase



parsing......
 {cpu time was 0.31 seconds, real time was 0.78 seconds.}


Please enter your question.....
  >> which parts were removed?



parsing......
(*QWORD *NPHR1 *ACT1)
 {cpu time was 6.18 seconds, real time was 67.31 seconds.}
 {gc time was 16.45 seconds.}
 {cpu time was 0.43 seconds, real time was 1.03 seconds.}
Interpreting.....
 {cpu time was 2.53 seconds, real time was 73.84 seconds.}
 {gc time was 16.37 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V R))
     '((V BUSER) (V PARTNO))
     '(AND (EQU (V ACTDATEDAY) 136.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 3.38 seconds, real time was 9.33 seconds.}

BUSER       PARTNO
-----------------
  48. 545-8186-005
  48.      585-340
  48.    219A556-2
```

```
Please enter your question.....
  >> parts installed?


parsing......
(*NPHR1 *ACT1)
 {cpu time was 5.09 seconds, real time was 61.28 seconds.}
 {gc time was 16.83 seconds.}
 {cpu time was 0.68 seconds, real time was 1.96 seconds.}
Interpreting.....
 {cpu time was 2.58 seconds, real time was 6.7 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V I))
     '((V ACTDATE) (V BUSER) (V PARTNO))
     '(AND (EQU (V ACTDATEDAY) 136.)
           (EQU (V ACTDATEYR) 9.)
           (OR (EQU (V BUSER) 48.)))
     '(ACTDATE UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 4.18 seconds, real time was 56.13 seconds.}
 {gc time was 16.18 seconds.}

ACTDATE BUSER       PARTNO
-------------------------
  9136.   48.       585-340
  9136.   48.     219A556-2
  9136.   48. 545-8186-005
  9136.   48.     218A556-2
```

The next group of examples demonstrates user-oriented routines.

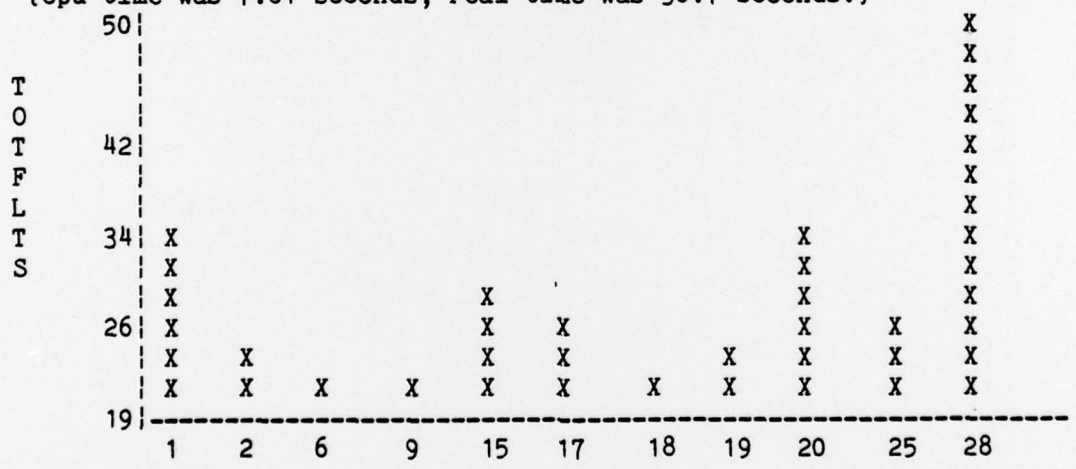
```
; Archive file of 6/15/77 at 12:5:10
T
(TALK)

Please enter your question.....
  >> what time is it?


parsing......
12. PM 10. MIN 52. SEC
 {cpu time was 0.94 seconds, real time was 2.21 seconds.}

Please enter your question.....
  >> what is the present date?


parsing......
I don't know the meaning of PRESENT
Perhaps it's miss-spelled?
Select one of the following:
   1. REPRESENT
   2. PRINT
   3. none of the above
>> 3

I don't know the word PRESENT
Select an option:
   1. Enter a replacement string for this word.
   2. Define this word
   3. Ignore this word
>> 2

I don't know the word PRESENT
Would you like to try to define it?
  type Y or N >> y
What lexical catagory should word PRESENT be in?
   1. noun
   2. verb
   3. adjective or adverb
   4. NONE OF THE ABOVE
>> 3

Would you like to try to find a synonym for PRESENT that I know?
  type Y or N >> y
Please enter a word or phrase which is roughly synonomous with PRESENT .
If I don't understand, We'll try again.
Whenever you feel like giving up, just type a carriage-return.
  >> current
Great, I know this one.
Henceforth, I will substitute CURRENT whenever I see PRESENT .
6. 15. 77.
 {cpu time was 3.63 seconds, real time was 94.51 seconds.}
 {gc time was 17.67 seconds.}
```

```
Please enter your question.....
  >> what does nor stand for?


parsing......
NOT OPERATIONALLY READY is the same as NOR
 {cpu time was 0.74 seconds, real time was 1.66 seconds.}


Please enter your question.....
  >> what kinds of planes does the data base know about?


parsing......

THE FOLLOWING PLANES WILL BE RECOGNIZED BY THE SEMANTIC NET:

  (1) A3

  (2) A7

  (3) F4

  (4) F11

  (5) CHEROKEE

  (6) PHANTOM

  (7) SKYHAWK

THE PLANES ABOVE MAY BE FURTHER SPECIFIED BY GIVING THE TAIL(BUSER,

BUREAU SERIAL NUMBER, BUNO, BUREAU NUMBER) ALONG WITH THE NAME OF THE

PLANE.

END-OF-FILE

 {cpu time was 5.28 seconds, real time was 65.46 seconds.}
 {gc time was 16.73 seconds.}
```

```
Please enter your question.....
  >> how big was it?


parsing......
WHO DO YOU THINK YOU ARE--ED MCMAHON
 {cpu time was 0.47 seconds, real time was 1.11 seconds.}
```

```
; Archive file of 6/15/77 at 12:14:56
T
(TALK)

Please enter your question.....
  >> for planes 3 and 5, how many catapult flights were flown between
     jan 1 and jun 30 74?



parsing......
(*NPHR1 *QWORD *NPHR2 *ACT1 *TIMEPP)
 {cpu time was 8.54 seconds, real time was 200.0 seconds.}
 {gc time was 32.34 seconds.}
 {cpu time was 0.48 seconds, real time was 1.23 seconds.}
Interpreting.....
 {cpu time was 2.71 seconds, real time was 58.85 seconds.}
 {gc time was 15.77 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V A))
     '((COUNT (V BUSER)) (SUM (V NCAT)))
     '(AND (GEQ (V ACTDATEDAY) 1.)
           (LEQ (V ACTDATEDAY) 181.)
           (EQU (V ACTDATEYR) 4.)
           (OR (EQU (V BUSER) 5.) (EQU (V BUSER) 3.)))
     'NIL)

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 4.41 seconds, real time was 10.09 seconds.}
 {gc time was 17.61 seconds.}
(((COUNT BUSER) (SUM NCAT)) = (1. 1.0))
```

```
Please enter your question.....
  >> which a7s flew > 15 flights in jan 72?


parsing......
(*QWORD *NPHR1 *ACT1 *NPHR2 *TIMEPP)
 {cpu time was 7.49 seconds, real time was 73.03 seconds.}
 {gc time was 15.42 seconds.}
 {cpu time was 0.44 seconds, real time was 1.36 seconds.}
Interpreting.....
 {cpu time was 3.12 seconds, real time was 52.13 seconds.}
 {gc time was 15.66 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V O))
     '((V BUSER) (V TOTFLTS))
     '(AND (EQU (V ACTDATEMON) 1.)
           (EQU (V ACTDATEYR) 2.)
           (EQU (V PLANETYPE) A7)
           (GT (V TOTFLTS) 15.))
     '(BUSER UP))

Should I evaluate the query?
  type Y or N >> y
Evaluating.....
 {cpu time was 7.81 seconds, real time was 36.7 seconds.}
```




```
          MIN = 19.0      AVERAGE = 27.4545455      MAX = 50.0
```

The next group of examples demonstrate some of the advance features currently being added to PLANES. These include the use of a query generator, the handling of compound noun phrases, the generation of a paraphrase of the request and the answering of questions involving comparisons.

```
;Archive file of 10/18/77 at 10:46:4

Please enter your question.....
  >> what maintenance occurred to plane 3 on jan 3 1973?

parsing......
(*QWORD *NPHR1 *ACT1 *NPHR2 *TIMEPP)
 {cpu time was 13.08 seconds, real time was 23.02 seconds.}
I have understood your request as:
     ((WHAT)
      (PLOT NIL)
      (NP ((MAINT (PRONOUN NIL)
            (TYPE-MAINT NIL)
            (ACTION-TAKEN NIL)
            (WORKUNITCODE NIL))
      NIL)
     ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 3.)) NIL)
     (NIL NIL)
     NIL
     ((WHAT)
      (MAINT (PRONOUN NIL)
            (TYPE-MAINT NIL)
            (ACTION-TAKEN NIL)
            (WORKUNITCODE NIL))))
       (NEG NIL)
       (ACT (OCCUR NIL) (NIL NIL) (NIL NIL))
       (TIME (DATE (MONTH (1. 0. 0.)) (DAY 3.) (YEAR 73.)) NIL)
       (NIL NIL)
       NIL)
```

```
Should I translate this?
  type Y or N >>y
Interpreting.....
RELATIONS M F ALL POSSIBLE
WILL SELECT M
 {cpu time was 5.09 seconds, real time was 13.22 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V M))
     '((V BUSER) (V ACTDATE) (V WUC) (V ML) (V AT) (V TM))
     '(AND (EQU (V ACTDATEDAY) 3.)
           (EQU (V ACTDATEYR) 3.)
           (EQU (V BUSER) 3.))
     '(ACTDATE UP))

A paraphrase of the query is:
PLANES SEARCHES THE DAILY MAINTENANCE ACTION SUMMARIES AND RETURNS:::

THE VALUES ((SERIAL#) ACTDATE (WORK UNIT CODE 7. DIGITS) ML (ACTION TAKEN)
            (TYPE MAINTENANCE))
SATISFYING:: (DATE = (ON JANUARY 3. , 73.)) ((SERIAL#) = 3.)

Should I evaluate the query?
  type Y or N >>y
Evaluating.....
 {cpu time was 4.0 seconds, real time was 9.47 seconds.}

SERIAL# ACTDATE WORK UNIT CODE 7.    ML ACTION TAKEN TYPE MAINTENANCE
                     DIGITS
-------------------------------------------------------------------
     3.   3003.      6315900.       1. REP          UNSCHED MAIN
     3.   3003.      7489500.       1. REP          UNSCHED MAIN
```

Example demonstrates compound noun phrases and their handling by the query generator.

```
Please enter your question.....
  >> how many flights and flight hours were flown by plane 3 in jan 73?

parsing......
(*QWORD *NPHR1 *ACT1 *NPHR2 *TIMEPP)
 {cpu time was 13.94 seconds, real time was 26.37 seconds.}
I have understood your request as:
     ((HOWMANY)
      (PLOT NIL)
      (NP (*COMPOUND NIL)
    ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 3.)) NIL)
    (NIL NIL)
    ((FLIGHTS NIL) (FLIGHTHOURS NIL))
    ((HOWMANY) (*COMPOUND NIL)))
      (NEG NIL)
      (ACT (FLY NIL) (NIL NIL) (NIL NIL))
      (TIME (DATE (MONTH (1. 0. 0.)) (DAY NIL) (YEAR 73.)) NIL)
      (NIL NIL)
      NIL)

Should I translate this?
  type Y or N >>y
Interpreting.....
Could not translate this query.
Would you like the query generator to try to translate it?
  type Y or N >>y
 {cpu time was 4.28 seconds, real time was 12.97 seconds.}

I have interpreted your request as follows:
     (FIND 'ALL
     '((V1 O))
     '((SUM (V1 TOTFLTS)) (SUM (V1 TOTHRS)))
     '(AND (EQU (V1 BUSER) 3.)
           (EQU (V1 ACTDATEMON) 1.)
           (EQU (V1 ACTDATEYR) 3.))
     'NIL)

A paraphrase of the query is:
PLANES SEARCHES THE MONTHLY FLIGHT AND MAINTENANCE
SUMMARIES AND RETURNS:::

THE SUM OF THE VALUES ((TOTAL FLIGHTS) (TOTAL HOURS))
SATISFYING:: ((SERIAL#) = 3.) (DATE = (IN JANUARY 73.))

Should I evaluate the query?
  type Y or N >>y
Evaluating.....
 {cpu time was 1.6 seconds, real time was 3.82 seconds.}
(((SUM TOTFLTS) (SUM TOTHRS)) = (17.0 33.0))
```

This example demonstrates the handling of questions involving comparisons.

```
Please enter your question.....
  >> did plane 3 have more flights in march than in april 1973?


parsing......
THE TARGET FOR COMPARISON HAS BEEN INTERPRETED AS FOLLOWS:
((HOWMANY)
 (PLOT NIL)
 (NP (FLIGHTS NIL)
     ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 3.)) NIL)
     (NIL NIL))
 (NEG NIL)
 (ACT (HAD NIL) (NIL NIL))
 (TIME (DATE (MONTH (4. 90. 91.)) (DAY NIL) (YEAR 73.)) NIL)
 (NIL NIL)
 NIL)

(((COUNT BUSER) (SUM TOTFLTS)) (1. 8.0))
8.0

THE OUTER COMPARISON HAS BEEN INTERPRETED AS:
((DO)
 (PLOT NIL)
 (NP ((PLANE (PRONOUN NIL) (TYPE PLANE) (BUSER 3.)) NIL)
     (FLIGHTS ((> 8.0)))
     (NIL NIL))
 (NEG NIL)
 (ACT (HAD NIL) (NIL NIL))
 (TIME (DATE (MONTH (3. 59. 60.)) (DAY NIL) (YEAR 73.)) NIL)
 (NIL NIL)
 NIL)

((BUSER ACTDATE TOTFLTS) ((3. 303. 13.)))
 {cpu time was 29.26 seconds, real time was 227.5 seconds.}
 {gc time was 43.05 seconds.}
```

LITERATURE CITED

[Brown, Burton and Bell 1974] Brown, John S., Richard R. Burton, and Alan G. Bell, SOHPIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI), BBN Report No. 2790, Bolt, Beranek and Newman, Inc., March 1, 1974.

[Brown and Burton 1975] Brown, John S. and Richard R. Burton, "Multiple Representation of Knowledge for Tutorial Reasoning," in Bobrow and Collins (eds.) Representation and Understanding, Academic Press, Inc., New York, 1975, pp. 311-50.

[Bruce 1975] Bruce, B., "Case Systems for Natural Language," Artificial Intelligence, 6 (1975), pp. 327-60.

[Celce-Muria 1972] Celce-Murcia, M., "Paradigms for Sentence Recognition," UCLA Dept. of Linguistic Memo.

[Codd 1970] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Comm. ACM, Vol. 6, No. 3, June 1970.

[Codd 1971a] Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," Proc. ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Nov. 11-12, 1971.

[Codd 1971b] Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," Proc. ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego, Nov. 11-12, 1971.

[Codd 1974] Codd, E. F., "Seven Steps to Rendezvous with the Casual User," Proc. IFIP TC-2 Working Conf. on Data Base Management Systems, Cargese, Corsica, April 1974, North-Holland Publ. Co., Amsterdam.

[Colby et al 1974] Colby, K. M., Faught, B., and Parkinson, R., "Pattern-Matching Rules for the Recognition of Natural Language Dialogue Expressions," Stanford AI Lab, Memo AIM 234, June 1974.

[Conrad 1976] Conrad, F., "Browser: A User Oriented Information Retrieval System," M.S. Thesis, Dept. of Computer Science, University of Illinois, Urbana, October 1976.

[Date 1975] Date, C. J., An Introduction to Database Systems, Addison-Wesley, Reading, MA., 1975.

[Fillmore 1968] Fillmore, C., "The Case for Case," in Bach and Harms (eds.) Universals in Linguistic Theory, Holt, Rinehart and Winston, New York, 1968.

[Finin 1977] Finin, T., "An Interpreter and Compiler for Augmented Transition Networks", M.S. Thesis, Dept. of Computer Science, University of Illinois, Urbana, May 1977.

[Green 1976] Green, F., "Implementation of a Query Language Based on the Relational Calculus," M.S. Thesis, Dept. of Computer Science, University of Illinois, Urbana, October 1976.

[Hadden 1977] Hadden, G., "NETEDI: An Augmented Transition Network Editor," M.S. Thesis, Dept. of Electrical Engineering, University of Illinois, Urbana, May 1977.

[Halliday 1970] Halliday, M. A. K., "Functional Diversity in Language as Seen from a Consideration of Modality and Mood in English," Foundations of Language, Vol. 6, 1970, pp. 322-61.

[Malhotra 1975] Malhotra, "Knowledge-Based English Language Systems for *Management Support: An Analysis of Requirements,"* 4IJCAI, Sept. 1975, pp. 842-47.

[M.I.T. Prog. Report 1974] M.I.T. A.I. Progress Report, Mass. Institute of Technology, Boston, 1974.

[Palermo 1972] Palermo, F. P., "A Data Base Search Problem," 4th Intl. Symp. on Computer and Infor. Tech., Dec. 1972, Plenum Press, N.Y., 1972, pp. 67-101.

[Schank 1973] Schank, R. C., "Identification of Conceptualizations Underlying Natural Language," in Schank and Colby (eds.) Computer Models of Thought and Language, Wott, Freeman, San Francisco, CA, 1973, pp. 187-247.

[Waltz et al 1976] Waltz, D. L., T. Finin, F. Green, F. Conrad, B. Goodman, and G. Hadden, "The PLANES System: Natural Language Access to a Large Data Base," Annual Report (July 1976), Coordinated Science Laboratory, University of Illinois, Urbana, Tech. Rpt. No. T-34.

[Waltz and Goodman 1977] Waltz, D. L. and B. A. Goodman, "Writing a Natural Langauge Data Base System," 5IJCAI, August 1977.

[Wilks 1975] Wilks, Y., "A Preferential, Pattern-Seeking, Semantics for Natural Language Inference," Artificial Intelligence, Vol. 6, No. 1, Spring 1975, pp. 53-74.

[Winograd 1972] Winograd, T., Understanding Natural Language, Academic Press, New York, 1972.

[Winograd 1973] Winograd, T., "A Procedural Model of Language Understanding," in Schank and Colby (eds.) Computer Models of Thought and Language, W. H. Freeman and Co., San Francisco, CA, 1973, pp. 152-186.

[Woods 1970] Woods, W. A., "Transition Network Grammars for Natural Language Analysis," Comm. ACM, Vol. 13, No. 10, Oct. 1970, pp. 591-606.

[Woods et al 1972] Woods, W. A., R. M. Kaplan, and B. Nash-Webber, "The Lunar Sciences Natural Language System: Final Report," Bolt, Beranek and Newman, Inc., Cambridge, MA, Report No. 2378, 1972.