RJ2144(29407)1/26/78
Computer Science

RENDEZVOUS VERSION 1: AN EXPERIMENTAL ENGLISH-LANGUAGE QUERY FORMULATION SYSTEM FOR CASUAL USERS OF RELATIONAL DATA BASES

E. F. Codd
R. S. Arnold
J-M. Cadiou
C. L. Chang
N. Roussopoulos

IBM Research Laboratory
San Jose, California 95193

ABSTRACT: This system, which became operational in April 1977, allows a user to express data base queries of the existential conjunctive type (with local negation if desired) in any English (grammatically correct or not) that he chooses. The user may enter his query in fragments if he happens to conceive it in that manner.

The system engages the user in clarification dialog when it encounters unfamiliar words or phrases, ambiguities, or errors of various kinds. Before retrieving data from the data base, the system generates a precise English version of its present understanding of the user's query. If the user finds that his intent has been misunderstood or only partially captured, he can edit his original query or edit the system's version or provide additional information through a menu-driven process. Experience with this system includes tests with more than 30 subjects (some computer-naive) using a data base dealing with suppliers, parts, project, and shipments.

CONTENTS

1 Introduction
2 System Behavior
2.1 Sample Data Base
2.2 Major Components of Version 1
2.3 Logical Coverage
2.4 Dialog Examples
2.5 Summary of Types of Dialog
3 Analyzer
3.1 Analyzer Goals
3.2 Analyzer Structures
3.2.1 Lexicon
3.2.2 Query Representation
3.2.3 Catalog
3.2.4 Value Tables
3.2.5 Denotation Tables
3.3 Analyzer Components
3.3.1 Word Normalizer
3.3.2 Phrase Transformer
3.3.3 Rule Class 100 Control
3.3.4 Goal Checker
3.4 Discussion of Analyzer
4 Menu Driver
4.1 Possible Missing Information
4.2 Discovering Missing Information
4.3 Joining Query Fragments
5 Generator
5.1 Goals of Generator
5.2 Knowledge Used by Generator
5.3 General Description of Generator
5.4 Examples showing Specific Features
5.4.1 Comparators
5.4.2 Key specification
5.4.3 Dates
5.4.4 Combination of Attribute Phrases
5.4.5 Implicit Entities
5.4.6 Focus in Case of Truth Test
5.5 Present Coverage
5.6 Related Work in Generation
5.7 Discussion of Generator
6 Menu-Driven Editor
7 Data Base Retriever
7.1 Compilation of DEDUCE
7.2 Query Execution
7.3 Answer Generation
8 Display Support
8.1 Design Approach
8.2 Main Components of Display Support
8.2.1 Formatted Screen Display
8.2.2 Adapted Keyboard
8.2.3 Self-Documentation
8.2.4 History Keeper
8.3 Summary of Display Support

9 Helper
10 Supervisor
11 Conclusion
11.1 Experience with Version 1
11.2 Comments on Future Directions
12 Acknowledgment
13 References
14 Appendix

RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases

1 INTRODUCTION

During the seventies several experimental systems have been developed that permit a user to express a query on a formatted data base in a more or less rich subset of a natural language such as English. We can loosely classify these systems into two generations. First-generation systems parsed the input query syntactically employing a natural language grammar (possibly involving some semantic features) and then -- in a separate processing stage -- used semantic information (usually expressed as a set of procedures) to convert the parse into high-level or low-level retrieval programs. Systems of this generation include CONVERSE [1], REL [2], LUNAR ROCKS [3], and TQA (formerly called REQUEST) [4]. In the second-generation systems the data base, which contains extensional information about some micro-world, is augmented by a knowledge base that provides intensional information about that same micro-world. This knowledge base provides a means of integrating semantic analysis with the syntactic parsing to such an extent that, in some second generation systems, the former completely dominates the latter. Examples of second generation systems are DILOS [5], PLANES [6], LIFER [7], and TORUS [8]. Of these systems PLANES is probably closest in spirit to RENDEZVOUS.

An important additional feature that distinguishes these two generations is the extent to which they provide the user with feedback concerning the system's understanding of his query prior to extracting an answer from the data base. First-generation systems provided virtually no feedback of this kind. For example, if a first-generation system discovered two or more semantically distinct interpretations of the user's query, it would generate and then execute two or more distinct retrieval programs; consequently, multiple answers would be delivered to the user. To make matters worse, such a system did not tell the user which answer corresponded to which interpretation. Second-generation systems, on the other hand, offer to the user a representation of each of the distinct meanings and ask the user to choose one before the system dips into the data base to obtain answers. Unfortunately, the user feedback in these second-generation systems is severely limited in scope and is expressed in a formal language that comparatively few users would understand.

In [9] we outlined the architecture and partial implementation of a natural language query system RENDEZVOUS that would provide feedback to the user in English and would therefore require two-way translation (from English to a formal language and vice versa). Furthermore, this system

would enter into clarification dialog with the user not only in situations in which the system encountered ambiguity, but also when it encountered something erroneous or incomprehensible in the user's query. In each case, the system initiates clarification dialog at the earliest opportunity to avoid the compounding of ambiguities and misunderstandings. This system, RENDEZVOUS Version 1, became fully operational in April 1977.

The goals of the RENDEZVOUS project can be summarized as follows:

1) to serve that segment of the general public (especially casual users) who need information from a formatted data base and can handle keyboard input;

2) to accept queries stated in <u>any</u> English, grammatically correct or not, rejecting only those that are clearly outside the domain of discourse supported by the data base at hand;

3) to attempt to understand as much as possible of each query on the supported domain of discourse and, where necessary, conduct dialog to arrive at a precise, unambiguous query that the user will accept as correctly interpreting his intent;

4) to extract the answer to this approved query from the data base.

An additional goal is to develop a query formulation system that is readily adaptable by a linguistic engineer to any relational data base. To attain this goal, the actual code should be as independent as possible of the domain of discourse, and the system should embody as much general knowledge of logic and relational theory as is common to all relational data base retrieval.

Version 1 is a pilot system working on a logically limited class of queries, but exhibiting very robust behavior linguistically and conversationally. For the sample data base and within the logically prescribed class of queries, we believe it meets goals 2), 3), and 4) above. For reasons cited in the Conclusion we do not believe Version 1 meets goal 1), although it represents a significant step in that direction. This claim is based primarily on the support Version 1 provides for 1) clarification dialog; 2) restatement in English of the system's understanding; and 3) <u>incremental query formulation</u>, which allows a user to conceive his query in steps that suit his mental capabilities and allows him to make mistakes and recover from them along the way (see Example B below).

In 1974 we could find no record of any previous system supporting clarification dialog (during linguistic analysis of the user's query). Moreover, it appeared to be very difficult to incorporate such dialog into the first-generation systems which existed then. Accordingly, the analyzer of Version 1 had to be designed with very little guidance from past experience.

2 SYSTEM BEHAVIOR

2.1 Sample Data Base

The behavior of the system will be illustrated using a simple data base concerning water projects in California, parts for these projects (such as pipes and valves), suppliers located in major cities of the United States, and shipments of parts from the suppliers to the projects. The corresponding four relations are:

PROJECT (JNO JNAME JLOC)

PART (PNO PNAME QOH QOO PTYPE)

SUPPLIER (SNO SNAME SLOC RATE)

SHIP (SNO PNO JNO DATE QSHIP)

For a given project, JNO denotes its unique serial number, JNAME its name (also unique), JLOC the city in which it is located (a project is located in exactly one city, but a given city may be the location of one or more projects). For a given kind of part, PNO denotes its unique serial number, PNAME its not necessarily unique name, QOH the quantity on hand, QOO the quantity on order, and PTYPE a broad classification of type (admissible values are INTERNAL, EXTERNAL, and PROCURED). For a given supplier, SNO denotes the unique serial number, SNAME the not necessarily unique name, SLOC the city in which the supplier is located (a supplier is located in exactly one city, but a city may be the location of one or more suppliers), and RATE denotes the supplier rating (admissible values are EXCELLENT, GOOD, FAIR, and POOR). For a given shipment, SNO denotes the supplier sending the shipment, PNO denotes the kind of part being shipped (only one kind per shipment), JNO denotes the project receiving the shipment, DATE denotes the date shipped from the supplier, and QSHIP denotes the quantity of parts of the specified kind in the shipment. As usual, the primary key of each relation is underlined.

2.2 Major Components of Version 1

To make the behavioral description of Version 1 more comprehensible, we shall now present a block diagram of this system (Fig. 1) plus a brief description of the services provided by each major component.

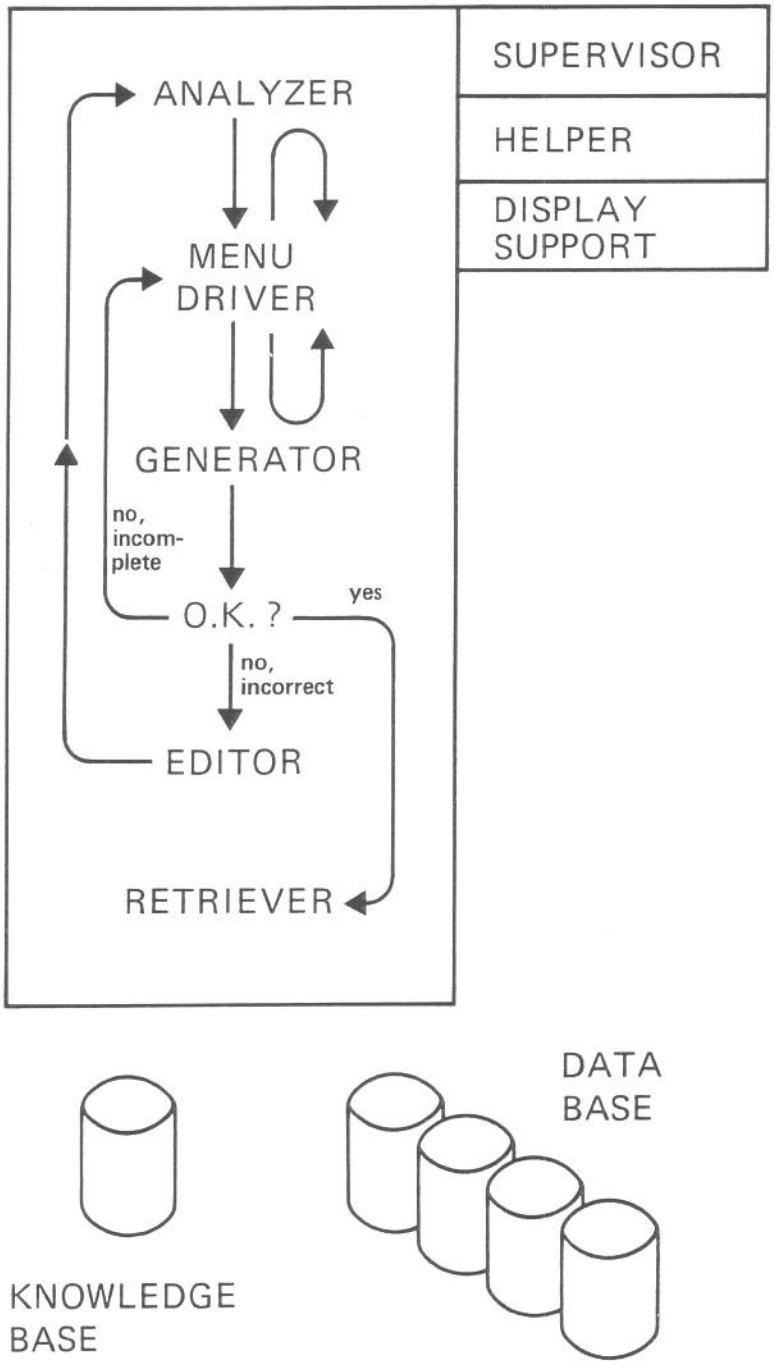

Figure 1. RENDEZVOUS SYSTEM

The analyzer attempts to translate the user's query into a simplified version of the data sublanguage DEDUCE [10]. It also supports two kinds of dialog: clarification dialog which consists of posing questions to the user about his intended query in the context of the query he actually entered (we do not assume that the query entered is a faithful representation of the query intended); and continuation dialog which allows a user to formulate a new query by specifying (perhaps very indirectly) small changes to be applied to his immediately preceding query.

The menu-driver takes the analyzer output, tests its logical completeness, and poses only those questions to the user that will yield a logically complete formal query. This menu-driven dialog is conducted without reference to the query actually entered by the user, and this is what distinguishes it from clarification dialog.

The generator translates the logically complete formal query (output by the menu-driver) into a precise English re-statement. It then asks the user if he is satisfied that this re-statement captures his intent.

The retriever is invoked only when the user has approved the English re-statement. It then takes the logically complete formal query from the menu-driver and retrieves data from the data base in order to generate an answer to that query. It also supports output dialog -- that is, dialog with the user about the answer from the data base.

The query editor provides a menu-oriented means for the user to change his query or the system's version thereof. The helper provides a menu-oriented means for the user to obtain general information about the kinds of data stored in the data base. The display support provides a dynamic partitioning of the display terminal screen to facilitate user interaction. It also keeps a history of the user's interactions and allows the user to examine this history whenever the system is awaiting a response from him. The supervisor is responsible for invoking the other components whenever appropriate.

2.3 Logical Coverage

Version 1 supports queries whose DEDUCE representation is a conjunction of relational clauses wherein each attribute can be given a value along with a connective ($<, \leq, =, \geq, >, \neq$), or it may be starred (for printing or counting), or it may be equated to a join variable. Existential quantification is assumed where necessary (no other quantification is permitted). The only functions supported in Version 1 are COUNT and EXIST. Although these limitations may seem highly restrictive and certainly must be removed in Version 2, the

majority of queries encountered in practice probably fall within the class supported by Version 1.

2.4 Dialog Examples

The examples shown below are transcripts of actual interactions with Version 1. The user interacts via a display terminal and keyboard. For his convenience, the top portion of the screen is used (amongst other things) to remind him of the latest source query he entered. This continual reminder has been edited out of the transcript, and some explanatory comments have been inserted in the left margin. Each time the system displays Sxxx, it poses a question to the user, and the question is uniquely identified by the 3-digit number xxx. It then displays Uxxx and awaits the user's reply, which begins on the same line immediately after the Uxxx identifier. Besides posing questions to the user, the system provides him with information that does not require a response. Such pieces of advice are prefixed by the word NOTE.

Example A.

```
                S 001  PLEASE TYPE IN YOUR QUERY:
                U 001  I WANT TO FIND CERTAIN PROJECTS.  PIPES WERE SENT
                       TO THEM IN FEB. 1975.

no clarifi-     NOTE   THIS IS WHAT THE SYSTEM UNDERSTANDS YOUR QUERY TO BE:
cation dialog          PRINT THE NAME OF EVERY PROJECT TO WHICH A SHIPMENT
                       OF A PART NAMED PIPE WAS SENT DURING FEBRUARY 1975.

verification    S 002  IS THE SYSTEM'S UNDERSTANDING
of single                    1  CORRECT AND COMPLETE
rephrasing                   2  NOT YET COMPLETE
                             3  INCORRECT
                       SELECT EXACTLY ONE ITEM BY NUMBER
                U 002  1


                * ***  THE ANSWER TO YOUR QUERY IS:
                       -----------
                         PROJECT
                           NAME
                       -----------
                         SAN LUIS
                          HYDRO
                          SONOMA
                       -----------


                S 003  IN REGARD TO YOUR PRESENT QUERY, WHAT ADDITIONAL
                       INFORMATION WOULD YOU LIKE TO HAVE?  IF NONE, JUST
                       HIT THE 'SEND' KEY.
target ellipsis U 003  GIVE ME THEIR LOCATIO ALSO


spelling        S 004  IS THE WORD 'LOCATIO' INTENDED TO BE: LOCATION?
                       (YES OR NO)
correction      U 004  YES


                NOTE   THIS IS WHAT THE SYSTEM UNDERSTANDS YOUR QUERY TO BE:

                       PRINT THE NAME AND LOCATION OF EVERY PROJECT TO WHICH
                       A SHIPMENT OF A PART NAMED PIPE WAS SENT DURING
                       FEBRUARY 1975.

                S 005  IS THE SYSTEM'S UNDERSTANDING
                             1  CORRECT AND COMPLETE
                             2  NOT YET COMPLETE
                             3  INCORRECT
                       SELECT EXACTLY ONE ITEM BY NUMBER
                U 005  1
```

* *** THE ANSWER TO YOUR QUERY IS:

PROJECT NAME	PROJECT LOCATION
SAN LUIS	FRESNO
HYDRO	SACRAMENTO
SONOMA	SAN FRANCISCO

S 006 IN REGARD TO YOUR PRESENT QUERY, WHAT ADDITIONAL INFORMATION WOULD YOU LIKE TO HAVE? IF NONE, JUST HIT THE 'SEND' KEY.

qualification ellipsis

U 006 HOW ABOUT RECENTLY?

fuzzy value

resolution

S 007 BY THE WORD RECENT(LY), DO YOU INTEND ONE OF THE FOLLOWING:

1 WITHIN THE LAST FEW YEARS
2 WITHIN THE LAST YEAR
3 WITHIN THE LAST FEW MONTHS
4 WITHIN THE LAST MONTH
5 NONE OF THE ABOVE

SELECT EXACTLY ONE ITEM BY NUMBER

U 007 4

NOTE THIS IS WHAT THE SYSTEM UNDERSTANDS YOUR QUERY TO BE:

PRINT THE NAME AND LOCATION OF EVERY PROJECT TO WHICH A SHIPMENT OF A PART NAMED PIPE WAS SENT DURING JULY 1977 OR AFTER.

S 008 IS THE SYSTEM'S UNDERSTANDING

1 CORRECT AND COMPLETE
2 NOT YET COMPLETE
3 INCORRECT

SELECT EXACTLY ONE ITEM BY NUMBER

U 008 1

* *** NO OBJECTS IN THE DATABASE QUALIFY AS AN ANSWER TO YOUR QUERY.
NOTE THAT IN THIS DATABASE
THERE IS NO SHIPMENT SENT DURING JULY 1977 OR AFTER

explanation of empty result

Example A exhibits a user query, which the analyzer was able to analyze without resorting to clarification dialog. Note, incidentally, that the input consists of two sentences.

The generator tells the user what it understands his query to mean by constructing a precise English sentence from the formal query computed by the analyzer, and then asks for the user's approval (giving him two disapproval options in case he is dissatisfied). The user approves, and the retriever therefore proceeds to extract the requested information from the data base. Note that the user failed to say specifically what information he wanted displayed about the pertinent projects. The analyzer accordingly took the project name from a table of defaults.

Immediately after retrieval of the data, the system poses a question (S003) that is intended solely as a psychological stimulus to the user -- subsequent system behavior is only trivially affected by the user's answer. The aim of this stimulus is to encourage the user to employ, wherever possible, very terse elliptical references to his previous query in order to amplify or modify it, rather than ponderously repeating much of it again. U003 is an excellent example of this. Since in this case the user is amplifying the target list of attributes to be displayed, we call this target ellipsis.

The user makes a keyboard error in entering the word LOCATION. Since the word LOCATIO which he entered is close to just one word LOCATION in the lexicon, the analyzer asks him if this is what he intended (S004) and he replies YES (U004). The final output now includes the location as well as the name of the pertinent projects. Stimulus S006 again induces the user to employ ellipsis, but this time it is not an amplification of the target list. Instead, it is a modification of the qualification and is therefore called qualification ellipsis. Version 1 allows only one new condition clause to be introduced or one existing condition to be altered in qualification ellipsis. In the present example, one existing condition clause involving SHIPDATE is modified from FEBRUARY 1975 to RECENTLY.

S007 is an attempt by the analyzer to resolve the meaning of RECENTLY in this particular query (the user's intended meanings for such fuzzy words could change from query to query). Incidentally, the system's restatement specifies JULY 1977, because it was run in that month and the user selected option 4 in response U007.

Following the user's approval (U008) of the system's understanding, the retriever finds there are no projects satisfying the specified conditions. Where possible without exploring too many subconditions of the specified condition, the retriever attempts to provide an explanation of such an

empty result. In this case, it finds that there is no shipment of any kind whatsoever in July 1977 or after. Such advice to the user will often relieve him of the burden of asking many closely related queries to pin down the cause of an exceptional answer from the data base.

Example B

```
S 001  PLEASE TYPE IN YOUR QUERY:
U 001  NONEXCELLENT NON DETROIT SUPPLIERS SHIPPING PART 37.
```

value ambiguity resolution

```
S 002  WHEN YOU USE THE PHRASE  PART 37', WHICH OF THE
       FOLLOWING PROPERTIES OF PARTS DO YOU INTEND BY
       THE WORD '37'?
             1  PART PART-SERIAL-NUMBER
             2  PART QUANTITY-ON-HAND
             3  PART QUANTITY-ON-ORDER
             4  NONE OF THE ABOVE
       SELECT ONE ITEM OR NONE BY NUMBER
U 002  1
```

value inconsistency notification

```
NOTE   BUT 37 IS INCORRECT AS A PART-SERIAL-NUMBER OF A PART;
       THE PART-SERIAL-NUMBER OF A PART MUST CONTAIN AT LEAST
       ONE DIGIT AND AT LEAST ONE LETTER

S 003  DO YOU WISH TO CHANGE YOUR QUERY? (YES OR NO)
```

pressed HELP key

```
U 003  <HELP>
```

help mode

```
S 004  WOULD YOU LIKE
             1  INFORMATION ON THE DATABASE
             2  HELP WITH THE KEYBOARD
             3  TO MAKE CHANGE(S) IN YOUR QUERY
             4  TO ENTER A NEW QUERY AND RESTART
             5  TO RESUME ACTION ON YOUR QUERY
             6  TO END THE SESSION ALTOGETHER
       SELECT EXACTLY ONE ITEM BY NUMBER
U 004  1


S 005  ON WHICH OF THE FOLLOWING DO YOU NEED EXPLANATION:
             1  THE WHOLE DATABASE
             2  SUPPLIERS
             3  PROJECTS
             4  PARTS
             5  SHIPMENTS
       SELECT ONE OR MORE ITEMS BY NUMBER
U 005  4
```

database information on parts

```
PARTS HAVE A SERIAL NUMBER AND A NAME.
THE SERIAL NUMBER OF A PART IDENTIFIES ITS TYPE UNIQUELY.
EXAMPLES OF PART SERIAL NUMBERS: P1, P13, P25, ETC.
PART NAMES DO NOT NECESSARILY DETERMINE THEIR PART SERIAL NUMBER.
EXAMPLES OF PART NAMES: SCREW, BOLT, PIPE, PUMP, ETC.
QUANTITY ON HAND SPECIFIES THE TOTAL QUANTITY OF A PART TYPE
     IN STOCK.
QUANTITY ON HAND IS GIVEN IN NUMBERS, E.G. 5, 30, 100, 250, ETC.
```

```
          QUANTITY ON ORDER SPECIFIES THE TOTAL QUANTITY OF A PART TYPE
               ORDERED FROM ALL SUPPLIERS.


          S 006  WOULD YOU LIKE
                       1  INFORMATION ON THE DATABASE
                       2  HELP WITH THE KEYBOARD
                       3  TO MAKE CHANGE(S)  IN YOUR QUERY
                       4  TO ENTER A NEW QUERY AND RESTART
                       5  TO RESUME ACTION ON YOUR QUERY
                       6  TO END THE SESSION ALTOGETHER
                  SELECT EXACTLY ONE ITEM BY NUMBER
          U 006  3

iriven    S 007  DO YOU WISH TO
ıg                     1  INSERT A WORD OR PHRASE BEFORE A WORD IN THE QUERY
                       2  INSERT A WORD OR PHRASE AFTER A WORD IN THE QUERY
                       3  DELETE A WORD OR PHRASE FROM THE QUERY
                       4  REPLACE A WORD OR PHRASE IN THE QUERY
                  SELECT EXACTLY ONE ITEM BY NUMBER
          U 007  4

          S 008  ENTER WORD OR PHRASE TO BE REPLACED; (OR JUST HIT
                 'SEND' TO SKIP THIS REQUEST AND CONTINUE)
          U 008  37

eplacement S 009 ENTER WORD OR PHRASE TO REPLACE 37; (OR JUST HIT
                 'SEND' TO SKIP THIS REQUEST AND CONTINUE)
          U 009  P37

          S 010  ANY OTHER CHANGE?
          U 010  NO

type      S 011  WHAT KIND OF RESPONSE TO YOUR QUERY ARE YOU EXPECTING?
tion                   1  A LIST OF ITEMS OF INFORMATION TO BE PRINTED
                          FROM THE DATA BASE
                       2  A YES/NO OR TRUE/FALSE ANSWER
                       3  A COUNT OF ITEMS FROM THE DATA BASE
                  SELECT EXACTLY ONE ITEM BY NUMBER
          U 011  1

resolu-   S 012  ON WHICH OF THE FOLLOWING ITEMS DO YOU WANT INFORMATION
                 PRINTED?
                       1  SUPPLIERS
                       2  SHIPMENTS
                       3  PROJECTS
                       4  PARTS
                       5  NONE OF THE ABOVE
```

SELECT ONE OR MORE ITEMS BY NUMBER
U 012 1

attribute resolution

S 013 WHICH OF THE FOLLOWING ITEMS OF INFORMATION ON SUPPLIERS DO YOU WANT PRINTED?
1 SUPPLIER NUMBER
2 SUPPLIER NAME
3 SUPPLIER LOCATION
4 SUPPLIER RATING
5 NONE OF THE ABOVE
SELECT ONE OR MORE ITEMS BY NUMBER
U 013 1,2,3,4

NOTE THIS IS WHAT THE SYSTEM UNDERSTANDS YOUR QUERY TO BE:

PRINT THE SERIAL NUMBER, NAME, LOCATION AND RATING OF EVERY SUPPLIER NOT RATED EXCELLENT AND NOT LOCATED IN DETROIT WHO SENT A SHIPMENT OF PART NUMBER P37 .

S 014 IS THE SYSTEM'S UNDERSTANDING
1 CORRECT AND COMPLETE
2 NOT YET COMPLETE
3 INCORRECT
SELECT EXACTLY ONE ITEM BY NUMBER
U 014 2

S 015 DOES THE INTERPRETATION FAIL TO MENTION SOMETHING YOU WANT PRINTED?
U 015 NO

S 016 DOES THE INTERPRETATION FAIL TO MENTION ANY SPECIFICATION(S) ON SUPPLIERS, PROJECTS, PARTS OR SHIPMENTS ?
U 016 YES

S 017 WHAT INFORMATION DO YOU WANT TO SPECIFY FOR (OR ADD TO) YOUR QUERY?
1 SUPPLIER(S) (E.G. ONE NAMED ACME, ETC.)
2 PART(S) (E.G. ONE WITH QUANTITY ON HAND 10, ETC.)
3 PROJECT(S) (E.G. ONE LOCATED IN HOUSTON, ETC.)
4 SHIPMENT(S) (E.G. ONE SENT ON JUNE 23,1976, ETC.
5 NONE OF THE ABOVE
SELECT ONE OR MORE ITEMS BY NUMBER
U 017 4

S 018 WHICH PARTICULAR ITEM(S) OF THE FOLLOWING INFORMATION ON SHIPMENTS DO YOU WANT TO SPECIFY?
1 SERIAL NUMBER OF PARTS SHIPPED

```
                        2  SERIAL NUMBER OF SUPPLIERS MAKING SHIPMENT
                        3  SERIAL NUMBER OF PROJECTS RECEIVING SHIPMENT
                        4  QUANTITY SHIPPED
                        5  DATE SHIPPED
                        6  NONE OF THE ABOVE
                    SELECT ONE OR MORE ITEMS BY NUMBER
                U 018  5


                S 019  PLEASE ENTER HERE THE DATE OF SHIPMENT YOU WISH TO SPECIFY:
variant input   U 019  AUG75
syntax for dates

                NOTE   THIS IS WHAT THE SYSTEM UNDERSTANDS YOUR QUERY TO BE:

                       PRINT THE SERIAL NUMBER, NAME, LOCATION AND RATING OF
                       EVERY SUPPLIER NOT RATED EXCELLENT AND NOT LOCATED IN
                       DETROIT WHO, DURING AUGUST 1975, SENT A SHIPMENT OF
                       PART NUMBER P37.

                S 020  IS THE SYSTEM'S UNDERSTANDING
                        1  CORRECT AND COMPLETE
                        2  NOT YET COMPLETE
                        3  INCORRECT
                    SELECT EXACTLY ONE ITEM BY NUMBER
                U 020  1


                * ***  NO OBJECTS IN THE DATABASE QUALIFY AS AN ANSWER TO
                       YOUR QUERY
```

The query U001 is not a complete English sentence. In addition, it contains a value error, and, as we shall see, represents only a small part of the user's total intent -- possibly because of the user's inability to express his entire intent in a single utterance. The interaction S002-U002 shows that the user intended 37 to be a part serial number. In the immediately following NOTE, the analyzer informs the user of this error. Accordingly, he calls for the helper to provide information about what kinds of data are available concerning parts. He then uses the query editor to change 37 to P37 (S007-U010) The analyzer produces a formal query which is incomplete, and the menu-driver acquires the missing information (S011-U013). By now the system's version of the query is quite complicated, but still does not satisfy the user (S014-U014). He proceeds to request a further condition (S016-U019) and forgets to type a blank between AUG and 75 (U019). Finally (U020), he accepts the system's version, and we see how the system has helped the user to advance step-by-step from the original incomplete and erroneous input to a rather complicated, but unambiguous, error-free query. This is an example of incremental query formulation, which we believe is vital to successful support of casual users.

Example C

two kinds of ambiguity

```
S 001  PLEASE TYPE IN YOUR QUERY:
U 001  HOW MANY WHITNEY SHIPMENTS HAVE A SHIPDATE 6/10/1975

S 002  IS 6/10/1975 INTENDED TO BE A DATE? (YES OR NO)
U 002  YES

S 003  WE ARE TRYING TO INTERPRET 6/10/1975 AS A DATE.
       WHICH OF THE FOLLOWING DATES DO YOU INTEND:
             1  JUNE 10
             2  OCTOBER 6
             3  NONE OF THE ABOVE
       SELECT ONE ITEM OR NONE BY NUMBER
U 003  1

S 004  IS THE WORD 'WHITNEY' CORRECTLY TYPED? (YES OR NO)
U 004  YES

S 005  IS 'WHITNEY' THE NAME OF A PART? (YES OR NO)
U 005  NO
```

association dialog

```
S 006  THE WORD 'WHITNEY' IS UNFAMILIAR.  IS IT ONE OF THE
       FOLLOWING:
                1  SUPPLIER NAME          5  PROJECT LOCATION
                2  SUPPLIER LOCATION      6  PART NAME
                3  SUPPLIER RATING        7  PART TYPE
                4  PROJECT NAME           8  NONE OF THE ABOVE
       SELECT BY NUMBER THE CLOSEST CATALOG ITEM
U 006  1

NOTE   THE SYSTEM HAS 2 WAYS TO INTERPRET YOUR QUERY. PLEASE
       INDICATE IF ANY OF THEM IS WHAT YOU INTENDED.
```

multiple rephrasings

```
    1  COUNT THE NUMBER OF SUPPLIERS NAMED WHITNEY WHO, ON
       JUNE 10, 1975, SENT A SHIPMENT .

    2  COUNT THE NUMBER OF SHIPMENTS SENT ON JUNE 10, 1975 BY
       A SUPPLIER NAMED WHITNEY .

S 007  WHICH OF THE INTERPRETATIONS IS CORRECT:
             1  FIRST
             2  SECOND
             3  NONE OF THE ABOVE
       SELECT EXACTLY ONE ITEM BY NUMBER
U 007  2

* ***  THERE IS NO SHIPMENT IN THE DATABASE SATISFYING YOUR QUERY.
```

The ambiguities in query U001 center on the phrase WHITNEY SHIPMENTS and the date 6/10/75. WHITNEY SHIPMENTS could mean amongst other things shipments from a supplier named WHITNEY, shipments of a part named WHITNEY, or shipments to a project named WHITNEY. For reasons that will be explained later, the query formulation front-end does not have the knowledge that WHITNEY is a supplier name. In fact, the word WHITNEY does not appear in its lexicon at all. The date 6/10/75 could mean June 10 (American style) or 6 October (European style). The at least 7-fold ambiguity in WHITNEY SHIPMENTS together with the independent 2-fold ambiguity in the date yield a 14-fold ambiguity in the overall query. By discussing these ambiguities with the user separately from one another, the analyzer keeps their resolution down to an additive, rather than a multiplicative, scale.

The interaction S006-U006 illustrates association dialog, which is applied by the analyzer to each unfamiliar word after it has unsuccessfully tried to induce its semantic category from context and from simple questions to the user (such as S005). The attributes (and associated relation names) listed as options in S006 are those having values which consist of letters only (no digits).

Two restatements in English are offered to the user, because of the system's inability in this query to precisely determine the focus of the COUNT request. These restatements represent two logically distinct queries. Thus, the system's subsequent behavior is materially affected by the user's response (U007). As we shall see in Example F below, multiple restatements are sometimes generated when the logical query has been completely determined.

Example D

use of menu-driver as independent query formulator

```
                S 001  PLEASE TYPE IN YOUR QUERY:
null query      U 001

query type      S 002  WHAT KIND OF RESPONSE TO YOUR QUERY ARE YOU EXPECTING?
resolution                   1  A LIST OF ITEMS OF INFORMATION TO BE PRINTED
                                FROM THE DATA BASE
                             2  A YES/NO OR TRUE/FALSE ANSWER
                             3  A COUNT OF ITEMS FROM THE DATA BASE
                           SELECT EXACTLY ONE ITEM BY NUMBER
                U 002  2

relation        S 003  WHAT INFORMATION DO YOU WANT TO SPECIFY FOR (OR ADD TO)
specification          YOUR QUERY?
                             1  SUPPLIER(S)  (E.G. ONE NAMED ACME, ETC.)
                             2  PART(S)      (E.G. ONE WITH QUANTITY ON HAND 10,
                                             ETC.)
                             3  PROJECT(S)   (E.G. ONE LOCATED IN HOUSTON, ETC.)
                             4  SHIPMENT(S)  (E.G. ONE SENT ON JUNE 23,1976, ETC.)
                             5  NONE OF THE ABOVE
                           SELECT ONE OR MORE ITEMS BY NUMBER
                U 003  1

attribute       S 004  WHICH PARTICULAR ITEM(S) OF THE FOLLOWING INFORMATION
resolution             ON SUPPLIERS DO YOU WANT TO SPECIFY?
                             1  SERIAL NUMBER OF SUPPLIER
                             2  NAME OF SUPPLIER
                             3  CITY IN WHICH SUPPLIER IS LOCATED
                             4  RATING OF SUPPLIER
                             5  NONE OF THE ABOVE
                           SELECT ONE OR MORE ITEMS BY NUMBER
                U 004  3 4

value           S 005  PLEASE ENTER HERE THE SUPPLIER LOCATION YOU WISH TO SPECIFY:
specification   U 005  LOS ANGELES

                S 006  PLEASE ENTER HERE THE SUPPLIER RATING YOU WISH TO SPECIFY:
                U 006  GOOD

comparator      S 007  DO YOU INTEND THAT SUPPLIER(S) IS RATED
specification                1  GOOD
```

```
                    2  OTHER THAN GOOD
                    3  WORSE THAN GOOD
                    4  GOOD OR WORSE
                    5  BETTER THAN GOOD
                    6  GOOD OR BETTER
                    7  NONE OF THESE ?
               SELECT EXACTLY ONE ITEM BY NUMBER
          U 007  6


          NOTE   THIS IS WHAT THE SYSTEM UNDERSTANDS YOUR QUERY TO BE:

                 IS THERE ANY SUPPLIER RATED GOOD OR BETTER AND LOCATED
                 IN LOS ANGELES?

          S 008  IS THE SYSTEM'S UNDERSTANDING
                    1  CORRECT AND COMPLETE
                    2  NOT YET COMPLETE
                    3  INCORRECT
               SELECT EXACTLY ONE ITEM BY NUMBER
          U 008  1


          * ***  YES.  THERE ARE EXACTLY 2 SUPPLIERS IN THE DATABASE
                 SATISFYING YOUR QUERY
```

question whether counted relation should be displayed

```
          S 009  DO YOU WANT THEM LISTED?
          U 009  YES
```

SUPPLIER SERIAL	SUPPLIER NAME	SUPPLIER LOCATION	SUPPLIER RATING
S6	PACIFIC	LOS ANGELES	GOOD
S8	VALLEY	LOS ANGELES	EXCELLENT

This example illustrates the behavior of the menu-driver when it receives no information at all from the analyzer. It first determines the type of query the user has in mind. It then seeks the principal topics of the query. The sample data base has such a few topics that they can be directly identified with the 4 base relations. In a data base with more topics, it might be necessary to use two or more steps of interrogation to find out which base relations are involved. In any event the menu-driver does not assume that the user's response is complete. For each base relation, the menu-driver probes depth-first which attributes are relevant and what values and comparators, if any, are to be associated with these attributes.

Example E

```
S 001  PLEASE TYPE IN YOUR QUERY:
U 001  WHO ARE THE MANAGERS OF MONTEREY PROJECTS

NOTE   WE HAVE NO INFORMATION ON THE FOLLOWING TOPIC(S):
       'MANAGERS'

S 002  WOULD YOU LIKE
              1  INFORMATION ON THE DATABASE
              2  HELP WITH THE KEYBOARD
              3  TO MAKE CHANGE(S) IN YOUR QUERY
              4  TO ENTER A NEW QUERY AND RESTART
              5  TO RESUME ACTION ON YOUR QUERY
              6  TO END THE SESSION ALTOGETHER
       SELECT EXACTLY ONE ITEM BY NUMBER
U 002  6
```

semantic overshoot

When a user learns that the data base contains information about a certain topic (e.g., projects), he is likely to guess that the data base contains information about a closely related topic (e.g., managership of projects). When such a guess is wrong (and in this example it is), the user may enter a query that is outside the domain of discourse supported by the data base. Such a query is said to contain a semantic overshoot. The analyzer can handle some of the more frequently occurring semantic overshoots. Example E illustrates how the system reacts in such a case. Immediately after the NOTE advising the user that a certain topic is not supported, the system invokes the HELP component and thus offers the user (in S002) a variety of options for his subsequent interaction.

Example F

```
S 001  PLEASE TYPE IN YOUR QUERY:
U 001  ARE THERE ANY SUPPLIERS SHIPPING TO BOTSON


S 002  IS THE WORD 'BOTSON' INTENDED TO BE: BOSTON? (YES OR NO)
U 002  YES


NOTE   SHIPPING TO BOSTON ?
       WE DO NOT HAVE PROJECTS WITH LOCATION BOSTON BUT WE DO
       HAVE PROJECTS WITH LOCATION BAKERSFIELD FRESNO
       LOS-ANGELES MONTEREY OAKLAND SACRAMENTO SAN-DIEGO
       SAN-FRANCISCO SAN-JOSE STOCKTON. IN ADDITION, WE HAVE
       SUPPLIERS WITH LOCATION BOSTON

S 003  DO YOU WISH TO CHANGE YOUR QUERY? (YES OR NO)
U 003  HELP


S 004  WOULD YOU LIKE
             1  INFORMATION ON THE DATABASE
             2  HELP WITH THE KEYBOARD
             3  TO MAKE CHANGE(S) IN YOUR QUERY
             4  TO ENTER A NEW QUERY AND RESTART
             5  TO RESUME ACTION ON YOUR QUERY
             6  TO END THE SESSION ALTOGETHER
       SELECT EXACTLY ONE ITEM BY NUMBER
U 004  3


S 005  DO YOU WISH TO
             1  INSERT A WORD OR PHRASE BEFORE A WORD IN THE QUERY
             2  INSERT A WORD OR PHRASE AFTER A WORD IN THE QUERY
             3  DELETE A WORD OR PHRASE FROM THE QUERY
             4  REPLACE A WORD OR PHRASE IN THE QUERY
       SELECT EXACTLY ONE ITEM BY NUMBER
U 005  4


S 006  ENTER WORD OR PHRASE TO BE REPLACED; (OR JUST HIT 'SEND'
       TO SKIP THIS REQUEST AND CONTINUE)
U 006  TO


S 007  ENTER WORD OR PHRASE TO REPLACE TO; (OR JUST HIT 'SEND'
       TO SKIP THIS REQUEST AND CONTINUE)
U 007  FROM


S 008  ANY OTHER CHANGE?
U 008  NO
```

```
NOTE    THE SYSTEM HAS 2 WAYS TO INTERPRET YOUR QUERY. PLEASE
        INDICATE IF ANY OF THEM IS WHAT YOU INTENDED.

     1  IS THERE ANY SUPPLIER LOCATED IN BOSTON WHO SENT A
        SHIPMENT ? .

     2  IS THERE ANY SHIPMENT FROM A SUPPLIER LOCATED IN BOSTON ? .

S 009   WHICH OF THE INTERPRETATIONS IS CORRECT:
              1  FIRST
              2  SECOND
              3  NONE OF THE ABOVE
        SELECT EXACTLY ONE ITEM BY NUMBER
U 009   1


* ***   YES.  THERE IS EXACTLY 1 SUPPLIER IN THE DATABASE
        SATISFYING YOUR QUERY.
S 010   DO YOU WANT IT LISTED?
U 010   YES
```

SUPPLIER SERIAL	SUPPLIER NAME	SUPPLIER LOCATION	SUPPLIER RATING
S18	ATLANTIC	BOSTON	POOR

Example F illustrates the kind of dialog that arises when a user's query involves a false presupposition about the value of an attribute (project location JLOC) that happens to be cataloged. In the NOTE immediately prior to S003, the system not only tells the user that his presupposition is wrong, but provides some positive guidance to enable him to correct his query (which he immediately proceeds to do).

In the note immediately preceding S009, the system provides two rephrasings of the user's query that are logically equivalent, but not psychologically equivalent (1 focuses on suppliers as the principal object, while 2 focuses on shipments). Whichever option is selected (1 or 2), the same DEDUCE query is interpreted by the RETRIEVER.

2.5 Summary of Types of Dialog

Version 1 of the RENDEZVOUS system supports 6 major types of dialog: clarification dialog (analyzer), menu-driven completion of a logically incomplete query (menu driver); verification dialog involving one or more re-statements in English (generator); editing dialog (menu-driven editor); output dialog concerning the answer to the user's query (retriever); and continuation dialog (analyzer).

This concludes the discussion of the system's behavior. Now, we turn our attention to the internal structure of the system, and treat each of the major components one by one.

3 RENDEZVOUS ANALYZER

3.1 Analyzer Goals

The principal function of the analyzer is that of translating as much as possible of the user's query into the data sublanguage DEDUCE. Remember that the user's query may be expressed in any English words whatsoever and these may be mis-spelled or mis-typed. The query need not conform to accepted English grammar and certainly need not be a whole sentence. One user of Version 1 entered the query PARTS INVENTORY when attempting to find the serial numbers of parts having a quantity on hand greater than 50. In such cases, the analyzer produces DEDUCE fragments and passes them on to the menu driver for completion of the formulation. It is important to note that the analyzer does not stop its analysis at the first difficulty encountered, but presses on to see what can be done with other parts of the query.

From the very beginning the RENDEZVOUS analyzer has been designed to support clarification dialog during the process of analysis. When it is necessary for the system to ask the user a clarification question, it is important to phrase the question in terms which the user will understand, avoiding abstract concepts in logic, grammar, and the theory of relations. This, in turn, implies that, however deep the analysis goes, the system must always be able to relate any difficulty which arises to its origin in some portion of the source query entered by the user. RENDEZVOUS Version 1 supports this requirement by dynamically maintaining for each unit of analysis (normally an English word or system-generated symbol) a tagpair i.e., a pair of integers, which specify the interval in source query word numbers of that portion of the source query giving rise to that unit.

To illustrate the use of these tagpairs, suppose the analyzer needs to ask the user a question concerning a phrase in the intermediate text. It cannot use this phrase itself in communicating with the user, because the words may be different from those he entered (due to word normalization), they may be differently ordered (due to shuffling by previously applied rules), and there may be system symbols (introduced by previously applied rules). However, from the tagpairs of all the words or symbols appearing in the intermediate phrase, the analyzer can compute a word number pair consisting of the least tag and the greatest tag appearing in these tagpairs. This word number pair defines that segment of source text which gave rise (perhaps through many steps) to the intermediate phrase. The analyzer can therefore use this pair to locate and display the source segment as a preface to its question for the user.

The analyzer has 3 major components: the word normalizer, the phrase transformer, and the goal checker. We shall now describe the more important structures which the analyzer uses, and then describe the processes which exploit these structures.

3.2 Analyzer Structures

It is incorrect to say, as in [8], that this system does not have a knowledge base. Knowledge happens to be represented in tables, functions, and certain classes of rules, but not in semantic network form. The principal structures used by the analyzer (and certain other parts of the system) to retain linguistic, semantic, and relational information for the pertinent domain of discourse and data base are the lexicon, the various catalog tables, value tables, denotation tables, query tables, and phrase transformation rules. We now describe each of these structures in turn, with the exception of the rules, which are best postponed to Section 3.3 on the phrase transformer.

3.2.1 Lexicon

The lexicon holds numerous English words plus internal system words (these are primarily related to the data base schema and the DEDUCE language). It also holds certain phrases needed by the generator. Associated with every entry in the lexicon is a pair of integers (called the lexclass pair) which identify the lexical classes to which the entry belongs. For example, the word WITH has the lexclass pair (17,15) which means that it belongs to the class of words that can begin a specification clause in English (class 17) and also to the class of prepositions (class 15). The membership of certain lexclasses is clearly dependent upon the domain of discourse (e.g., class 7, the relation names). The other lexclasses are of general utility (e.g., class 15, prepositions). Thus, classification of words by means of these lexclasses provides, among other things, a reasonably clear dichotomy between that part of the lexicon which is dependent on the domain of discourse and that part which is independent.

The lexicon is organized by word length. For word length j ($1 \leq j \leq 15$) there are 3 tables, one holding the actual words, another the lexclass pairs, and the third holding integer identifiers (one per word, uniquely distinguishing that word from all other words). Three similar tables are provided to accommodate all words or phrases whose length exceeds 15 characters. Fig. 2 shows the 3 tables for words of length 4 characters. Fig. 3 shows what each lexical class signifies.

LEX04	LID04	LCLASS04	
PART	104	7	20
SLOC	204	6	14
JLOC	304	6	14
DATE	404	6	49
NAME	504	9	14
TYPE	604	9	49
GOOD	704	5	2
CITY	804	22	14
ON-*	904	-1	-1
FROM	1004	15	14
*-OF	1204	-1	-1
USED	1304	-1	-1
COMP	1404	-1	-1
SHIP	1504	7	21
SENT	1604	9	50
FAIR	1704	5	2
BACK	1804	53	14
CASE	1904	24	14
DATA	2004	24	14
DEAL	2104	17	15
DOES	2204	27	14
DOWN	2304	15	14
DROP	2404	32	14
EACH	2504	16	14
EVER	2604	14	14
FACT	2704	24	14
FAST	2804	53	14
FILE	2904	24	14
FIND	3004	28	14
FIVE	3104	22	14
FOUR	3204	22	14
GIVE	3304	28	14
HAND	3404	9	14
HAVE	3504	17	15
HERS	3604	19	14
JULY	3704	46	14
JUNE	3804	46	14
KIND	3904	9	14
LAST	4004	35	14
LESS	4104	12	41
LIST	4204	28	14
LOOK	4304	28	14
MADE	4404	27	14
MAKE	4504	28	14
MANY	4604	23	58
MEAN	4704	8	26
MILE	4804	29	14
MORE	4904	12	43
NONE	5004	22	14

Fig. 2 Part of Lexicon

01	NUM VALUE	31	REPLACE
02	ALPHA VALUE	32	DELETE
03	MIXED VALUE	33	TEXT
04	NON-CATALOGED VALUE	34	BEGIN
05	CATALOGED VALUE	35	NTH
06	ATTRIBUTE	36	MAJOR
07	RELATION	37	INTERMEDIATE
08	FUNCTION	38	MINOR
09	CATSYNONYM	39	FREE
10	IS	40	TEXT COMMAND
11	BOOLEAN	41	<
12	COMPARATOR	42	=
13	LOCATION	43	>
14	NOISE	44	PARENDS
15	PREPOSITION	45	SYSTEM WORDS
16	QUANTIFIER	46	MONTH
17	SIG	47	YEAR
18	TIME	48	DATE
19	PRONOUN	49	DOMAIN
20	NL-RELATION	50	TRANSFER
21	L-RELATION	51	FIRST BARREL
22	GENSYNONYM	52	SECOND BARREL
23	FUZZY	53	NOT-SUPPORTED
24	CASE	54	MEAGERLY-SUPPORTED
25	ARTICLE	55	CATALOG
26	WANT	56	FUZZY LOW
27	DO	57	FUZZY MEDIUM
28	FIND	58	FUZZY HIGH
29	UNITS	59	FUZZY MODIFIER
30	PUT		

Fig. 3 Lexical Class Numbers and Meaning

3.2.2 Query Representation

The user's query is initially stored as a string of words, some of which may be partially or entirely numeric. The query is immediately cast into a representation used throughout the analyzer for many purposes. This representation consists of 3 tables: one called the text table holds the words of the query, another called the lex table holds the lexclass pairs of those words, and a third called the tag table holds the tag pairs of those words. During initialization, as each word is appended to the text table, its lexclass pair is obtained from the lexicon if possible, else it is computed. This pair is then appended to the lex table. The initial tag pair assigned to word j is (j,j), and this is appended to the tag table.

Eight query stages are stored, each using this structure, and they are used for the following 8 purposes:

1) the user's source query

2) the word-normalized query

3) the current version of the query for matching attempts by the phrase transformer

4) the phrase generated by the phrase transformer

5) the new version of the query obtained by substituting the generated phrase (4) in place of the matched phrase in (3)

6) the list of words plus associated lexpairs and tagpairs tentatively dropped by the goal checker

7) a copy of the current query version from (3) when the analyzer has been temporarily diverted from analyzing the current query to analyzing the latest response from the user

8) a copy of the partially generated phrase from (4) when such a temporary diversion is taking place.

3.2.3 Catalog

The principal table ACATDOT in the catalog lists each of the attributes for each relation along with an identifier for the underlying domain, the syntactic type of the values taken by the attribute (A = alphabetic, N = numeric, D = date, M = alphanumeric), whether or not the attribute participates in the primary key of the relation, whether or not a meaningful ordering is associated with the values of the attribute, and whether or not the attribute has all of its values in a table ACATV3. Other information stored in

this table assists the analyzer in making guesses concerning which attributes of a given relation are implied when the user fails to specify them in certain kinds of contexts.

3.2.4 Value Tables

Some attributes have alphabetic values. These may be proper names (e.g., MARIN as a value of the project name JNAME) or English adjectives (e.g., EXCELLENT as a value of the supplier rating RATE). When such values occur in a query, the analyzer must have some way to determine that they are indeed values which may appear in the data base. The usual approach adopted in natural language processing is to build a lexicon that contains a copy of all the distinct alphabetic values in the data base or else to use the data base itself as an extension of the lexicon. However, RENDEZVOUS is intended to be a front-end query formulator for very large back-end data bases (possibly remotely located). We therefore do not find either of the above approaches attractive. Instead, we divide attributes into 2 types, those with sufficiently few distinct values that all such values can be accommodated in the front end, and those with too many distinct values. The former are called cataloged attributes and their values are called cataloged values. The latter are called cached attributes, because only the most frequently used values of these attributes are recorded in the front end. Those values of cached attributes which are recorded in the front end are called cached values.

Given any alphabetic query word, the system can always determine whether or not that word is a value of a cataloged attribute by searching for it in the table ACATV3. If found, an associated pointer to ACATDOT indicates the cataloged attribute of which it is a value. If not found, the system can conclude that the word is not the value of a cataloged attribute. In contrast, the system cannot always determine by itself whether such a word is a value of a cached attribute. It would first search the table of cached values ACATVB. If the word is found, then an associated pointer to ACATDOT again indicates the attribute of which it is a value. If not found, however, the word may or may not be a non-cached value of some cached attribute. If immediate resolution of this question is needed, the system puts a suitably concrete question to the user (for example: IS FOTHERINGAY THE NAME OF A SUPPLIER?).

3.2.5 Denotation Tables

The set of all words that may be employed by the set of all potential users may be called the source vocabulary. For economy in translation machinery, it is desirable to translate as many user words as possible from the source vocabulary into words from a much smaller vocabulary, while

preserving the original meaning. When two or more words have the same denotatien, one of them (or some system abbreviation) may be chosen to replace all the others. For example, RENDEZVOUS Version 1 replaces every occurrence of the words ZERO, NONE, NIL, NOUGHT by the numeral 0. It also replaces PLACE, CITY, LOCATION by the system abbreviation LOC. Like the source words PLACE, CITY, and LOCATION, the abbreviation LOC may denote (in the example domain of discourse) either supplier location SLOC or project location JLOC.

There are two tables ACATSYN and ACATGSYN which store this kind of information. ACATSYN is the table that is consulted first at word-normalization time. Each row contains a source word or system abbreviation along with a pointer identifying some other table together with a row number within that table. The words SEND and SUPPLY, for example, occur in ACATSYN with the same pointer R03, which points to the 3rd row of the table ACATR of relation names, and we find in this row the relation name SHIP. Similarly, LOCATION and CITY occur in ACATSYN with the same pointer G03, which points to the 3rd row of the table ACATGSYN, and here we find the entry LOC. The word LOC also appears in ACATSYN twice, once with a pointer A06 to the 6th row of ACATDOT (i.e., to the attribute JLOC), and once with a corresponding pointer A02 (to the attribute SLOC).

3.3 Analyzer Components

3.3.1 Word Normalizer

The principal objective of the word normalizer is to reduce as many words as possible in the source query to standard internal counterparts, so as to permit significant simplification in the phrase transforming rules that are invoked later (see next section). The services provided by the word normalizer include:

1) replacement of punctuation marks by blanks; removal of parentheses, quotation marks, and certain other special characters (because we have found that casual users tend to misuse them and mislead the system);

2) stripping of suffixes, providing the result of stripping is found in the lexicon (e.g., SHIPPED is reduced to SHIP);

3) separation of prefixes, providing the stem is found in the lexicon (e.g., NONDETROIT is reduced to NOT DETROIT);

4) recognition and hyphenation of compound names (e.g., SAN JOSE is reduced to SAN-JOSE);

5) conversion of each synonym to a unique internal representative where such exists;

6) correction of spelling (subject of course to user approval) provided there is at least one lexicon word close to the source word or a possible stem of the source word;

7) correction of typographical errors (subject to user approval), such as omitted blanks, the letter O used for the digit 0, and the letter L being used for the digit 1;

8) recognition of many different ways of expressing dates and conversion to internal standard form (sometimes with and sometimes without user dialog);

9) partial conversion of spelled-out numbers to digital form (e.g., TWENTY and THIRD are separately reduced to 20 and 3 in preparation for later phrase transformation of 20 3 to 23);

10) recognition of words representing topics not supported in the data base and generation of a warning to the user -- of course, this is limited to those words pre-stored in the lexicon with a certain class identifier (NOT-SUPPORTED coded 53);

11) recognition of words representing topics that are only meagerly supported in the data base and generation of a suitable warning to the user -- again, this is limited to those words pre-stored in the lexicon with a certain class identifier (MEAGERLY-SUPPORTED coded 54);

The spelling corrector's measure of closeness of two words is based on a combination of the following possibilities: two adjacent letters may have been interchanged or letters may have been dropped, introduced, or altered. For words longer than 3 letters, the permitted extent of these changes depends directly on the lengths of the two words involved -- the longer the words, the more aberrations are permitted. For words of 2 or 3 letters, only the interchange of 2 letters is allowed.

3.3.2 Phrase Transformer

The second component of the analyzer to be activated is the phrase transformer. By means of phrase transformation rules, applied in a deterministic manner, it attempts to recognize certain types of phrases in the text, and replace these by other phrases that are syntactically closer to the form required by the target language DEDUCE. The syntax of DEDUCE is tabulated in the Appendix A.

The rules are designed to home the user into the domain of discourse supported by the data base. In spite of this, about half of the rules are independent of the particular domain of discourse. The majority of the "domain-independent" ones are, however, dependent upon the data base being of the n-ary relational type.

For reasons of modularity, the phrase transformation rules are partitioned into classes each of which has a specific purpose. In Version 1 the rule classes are as follows (listed in order of initial invocation):

class	purpose
1	discover type of query
2	process dates and date phrases
3	process general idioms of English
4	process special idioms of domain of discourse
5	handle local negation and comparators
100	process RAVp patterns (i.e., patterns of relation names, attributes, values, parends)
10	introduce join terms
11	handle left-over attributes and values

The phrase transformation rules of RENDEZVOUS are multi-purpose. They always specify conditional replacement of one kind of phrase (if it occurs in the text being examined) by another kind (possibly empty). However, many of the rules call for one or more of the following sources to be consulted: the static knowledge base, the dynamic knowledge base, and the user. When the user is questioned by the phrase transformer, the dialog parameters are either directly or indirectly specified within the rules themselves. These sources of information must be tapped to induce or directly acquire missing information (e.g., attributes, relation names, prepositions) and to resolve ambiguities, apparent inconsistencies, etc.

To explain how the rules work, we need to look at them both individually and collectively. The syntax of the rule language is given in the Appendix B. In [21] Finin discusses the problems encountered in converting the RENDEZVOUS analyzer rules into augmented transition network form.

The left-hand side (LHS) of each rule specifies a pattern to be matched and may include the invocation of truth-valued functions. The right-hand side (RHS) specifies one or more replacement phrases to be generated, and may include the invocation of functions which extract information from the knowledge base or the user or both. If two or more replacement phrases are specified, the user will be called upon to select one or none -- of course, he is presented with the various options in an external form that is easy for him to understand.

The query cursor points to the query word (in the present version of the query) that is the first to be matched in the next matching attempt. After a rule has been successfully applied, the normal or default action on the cursor is to advance it to the first query word immediately following the replacement phrase resulting from the rule which has just succeeded. However, by use of a control code at the end of a rule, the cursor can be advanced a specified number of words to the right or left (+n means advance n words to the right, -n means the same number of words to the left, but not beyond the first query word); or it can be set back to the first word in the present query (% denotes this option). If there is no rule of the presently active rule class that can succeed at the present cursor position, the cursor is normally advanced one word to the right. The only exception to this occurs in the processing of class 100 rules dealing with RAVp patterns, wherein the cursor is advanced to the next occurrence of a relation, attribute, value, or left parend. As soon as the cursor points to the end-of-query symbol, the presently active class is deactivated, and the next class is activated.

In the examples which follow we give the rule identifier, then the rule itself, and then the informal meaning of the rule.

```
AX0131   HOW MANY -> C
```

The effect of this rule is to replace the phrase HOW MANY, whenever it is found at the query cursor, by the internal symbol C, which denotes a count-type query. This is the simplest kind of rule: no lexical class numbers, no rule-invoked functions, and no dialog are involved.

```
AX0151   10 THERE (NOT,•) (25,•) -> E
```

The first term on the left-hand side (LHS) of this rule matches the word at the query cursor if that word belongs to lexclass 10 (i.e., if it is one of the forms of the verb TO BE, such as IS, ARE, WAS, WERE, BE). The third term will match the query word NOT if it is in position 3 relative to the query cursor, but the middle dot • indicates that its appearance is optional. Likewise, the 4th term permits a word of lexclass 25 (an article) to be present, but does not require it. Accordingly, this one rule is equivalent to the set of rules:

```
10 THERE NOT 25 -> E
10 THERE NOT -> E
10 THERE 25 -> E
10 THERE -> E
```

The effect, whenever a match is obtained, is to replace the

entire matched phrase by E, which stands for the exist-type query. For example, IS THERE A PROJECT is replaced by E PROJECT. The same phrase replaces ARE THERE ANY PROJECT (note that the word normalizer has dropped the S from PROJECTS, if the user actually employed the plural).

Incidentally, throughout the attempted application of rules of class 1 (including AX0131, AX0151), the cursor is kept fixed and points to the first query word. This restriction applies only to rule class 1.

AX0235 46 1 1 -> D213 -3

A phrase from the query matches the LHS of this rule if it consists of a month followed by two integers (e.g., NOVEMBER 23 75). The first term D213 on the right-hand side (RHS) invokes an APL function AHD with 3 arguments obtained successively from the query words denoted by LHS terms numbered 2,1,3 (i.e., the first integer, then the month, then the second integer). AHD attempts to make a complete internal date from these arguments without dialog if possible, else with dialog (in the cited example, dialog would not be necessary and AHD would yield the internal standard European-style date 23/11/75). The final term -3 specifies that, whenever the rule is successfully applied, the query cursor is to be moved 3 words to the left (or to the first word of the current query, if there are less than 3 words to the left).

With the present knowledge base, if the integer specifying the year were outside the ranges 50 through 80 and 1950 through 1980, the user would be asked if he intended that integer to be a year. An affirmative response from the user elicits advice from the system as to the range of dates supported in the data base, plus the invitation to change his query accordingly. A negative response would cause this application of rule AX0235 to be quashed. The origin would then be advanced to the right by one word.

AX0318 NEITHER *N NOR -> NOT 2 NOT +1

A phrase from the query matches the LHS of this rule if it begins with the word NEITHER, ends with the word NOR, and the sequence of words in between is non-empty and does not contain NOR. The symbol * denotes the shortest phrase between an occurrence of NEITHER and a subsequent occurrence of NOR; the symbol N after * requires this phrase to be non-empty. For example, the phrase NEITHER IN DETROIT NOR POOR is replaced by NOT IN DETROIT NOT POOR.

AX0451 SHIP (25,•) 2 -> 1 PART (PNAME = 3?6?8) +0

A phrase from the query matches the LHS of this rule if it begins with the word SHIP, which is optionally followed by

an article, and ends with an alphabetic word not belonging to the lexicon. The phrase to be constructed is:

SHIP PART (PNAME = <value>)

where <value> is the denotation of rule term 3 (i.e., the alphabetic word not in the lexicon). However, rule term 3?6?8 specifies that, unless this alphabetic word is a cached value, user confirmation is required, and that the dialog arguments for constructing the question to the user are the value (denotation of rule term 3), PART (rule term 6), and PNAME (rule term 8). For example, the phrase SHIP A WIDGET yields the question IS WIDGET THE NAME OF A PART? and an affirmative answer from the user yields the phrase:

SHIP PART (PNAME = WIDGET)

in place of the given phrase.

AX2307 5 7 -> 2 (A201 = 1) %

This is one of the rules in class 23. Class 23 is one of several disjoint rule classes that together constitute class 100. While class 100 as a whole deals with various patterns in which relation names R, attributes A, values V, and parenthesized expressions p may appear, class 23 deals only with those cases in which a cataloged value is followed by a relation name either immediately or without any intervening R, A, V, or p.

The LHS of this rule is matched whenever at the cursor there is a cataloged value immediately followed by a relation name (e.g., INTERNAL PART or DETROIT PART). The RHS specifies the construction of a replacement phrase consisting of the relation name (rule term 2), a left parend, an attribute from the catalog, the equal sign, the cataloged value (rule term 1), and a right parend. To obtain an appropriate attribute from the catalog, the term A201 is decoded as follows:

A designates that an attribute is to be obtained;

2 denotes the query word matching rule term 2
i.e., a relation name, R say (lexclass is 7);

0 denotes a null argument (lexclass is 0);

1 denotes the query word matching rule term 1
i.e., a cataloged value, V say (lexclass is 5).

To decide which APL function to invoke, the code A201 is converted per the corresponding lexclass numbers to A705, and a prefix AH is attached. Thus, the function AHA705 is invoked with the arguments R and V. This function searches the catalog for all the attributes of R for which V is an admissible value. If only one such attribute is found, it is immediately incorporated into the new phrase and phrase replacement can proceed without dialog. If two or more such

attributes are found, the ambiguity is resolved by constructing a multiple choice question for the user, together with a preamble which tells him which part of his query is in doubt and that just one of the displayed options should be selected (note that the option NONE OF THE ABOVE is included in most multiple choice questions). If, however, no attribute of R has V as an admissible value, two cases are distinguished:

1) R has an attribute, A say, defined on a domain D and V belongs to D, but not A;

2) R has no such attribute.

In case 1) a message is computed telling the user that the objects (e.g., projects) denoted by the relation name R do not have value V (e.g., Boston) as an admissible value for the attribute A (e.g., location). The message is, of course, cast in very concrete form: WE DO NOT HAVE PROJECTS IN BOSTON. The system goes on to advise the user where projects are presently located, and also what objects other than projects are located in Boston. In case 2) the user will be advised, again in a very concrete way, that the objects denoted by the relation name do not have V as the value of any of their attributes.

This rule can bring about silent replacements (i.e, those not entailing dialog). For example, POOR SUPPLIER is replaced by SUPPLIER (RATE = POOR).

```
AX1001   PART *P * SHIP_X1 *P
      -> 1 ( 2 PNO = X ) 3 4 ( 5 PNO = X ) +6
```

Term 1 of this rule requires the word PART to appear at the cursor. Term 2, *P, requires a parenthesized expression or null. Term 3, *, requires a sequence of words, possibly empty. Term 4, SHIP_X1, requires that the phrase matched by term 3 be followed by the word SHIP and that there be no join term joining SHIP with PART (the denotation of term 1). Term 5, *P, requires a parenthesized expression or null. The RHS of this rule calls for the incorporation of join terms to join PART with SHIP via the common attribute PNO. Note that the digit 2 on the RHS designates the expression found within the parends (if any) following PART; else null. Similarly, the digit 5 denotes the expression found within the parends (if any) immediately following SHIP. The two occurrences of X on the RHS denote a common DEDUCE variable obtained from a dispenser of such variables (e.g., x2). As an example, consider the phrases:

1) PART (PNAME = WIDGET) SHIP BY
2) PART (PNO = x1 PNAME = WIDGET) SHIP (PNO = x1) BY

Rule AX1001 causes phrase 1) to be replaced by phrase 2).

AX0406 C PART * SHIP -> ?F?C PART 3 ?QSHIP?PNO 4 +2

This rule is intended to resolve the type-instance ambiguity in source phrases like COUNT THE PARTS SHIPPED TO ..., which could mean either:

1) count the distinct types of parts shipped -- a request to count the distinct values of PNO; or

2) find the quantity of parts of a particular type shipped -- a request to find the attribute QSHIP in the relation SHIP.

Appearance of the symbol ? at the beginning of terms on the RHS of a rule calls for the user to choose between two or more distinct replacement phrases (translated, of course, into an external form comprehensible to the user). In this case, the RHS designates the following 2 alternative phrases:

1) F PART <phrase> QSHIP SHIP
2) C PART <phrase> PNO SHIP

where <phrase> designates the query phrase denoted by the * on the LHS of this rule. For each of these phrases the system constructs an external English version that should be comprehensible to the casual user. It then adds the option NONE OF THE ABOVE, and presents the resulting three options to the user. Note that, if the user selects the first option, the query type is immediately changed from COUNT to FIND. For example, with this option the phrase C PART SHIP is replaced by F PART SHIP QSHIP.

3.3.3 Rule Class 100 Control

We have already mentioned that rule class 100 is actually composed of several (in fact 23) disjoint classes which collectively process the various RAVp patterns in the query resulting from the application of rule classes 1 through 5. Unlike the other rule classes, those which belong to class 100 are not processed in a pre-determined sequence. Instead, their invocation is determined by an additional higher level of pattern matching that is executed on an abbreviated version of the present query, namely one obtained by retaining with order preserved all occurrences of words of type R, A, and V together with parenthesized expressions, and by dropping all other words. For example, the sequence RpAV triggers the invocation of class 71. The regular pattern matching procedure is then executed on the unabbreviated present version of the query and it determines if any rule in class 71 can be applied at the present cursor position. As soon as one is found, it is applied in the

usual manner, except that the default cursor advancement is determined by a higher level search for the beginning of the next RAVp pattern.

3.3.4 Goal Checker

The third and final component of the analyzer is the goal checker. It is activated immediately after each pass through the rule classes 2 through 11. Its first action is to check to see whether the transformed query has reached a completely formal (if not logically complete) state. To be completely formal, the query must consist of a query type symbol (w = undefined, F = find, E = exist, T = truth test other than exist, C = count) followed by zero or more relation names, each of which may be immediately followed by a parenthesized expression, and terminated by the symbol w. If the query is in this state, analysis is terminated and control is passed to the menu-driver.

If the query is still not completely formal and the rule pass just completed is the first on the given source query or the first since a user-initiated or system-initiated change in the source query, the goal checker tentatively drops all query words belonging to non-essential lexical classes (e.g., prepositions, articles, etc. -- in fact, all classes except 1,3,5,6,7,8,9,12 and 41 through 49). It then passes control back to the phrase transformer to make another pass over the rules (but omitting class 1). With non-essential words dropped, one or more rules may become effective, and these in turn may trigger a host of other changes. Notice that class 2 words (alphabetic, not in lexicon) are included in those tentatively dropped. This action tends to compensate for inevitable deficiencies in both the lexicon and rules, although the associated risks are justified only by the system's support for menu-driven completion of logically incomplete queries, and by the English re-statement to check intent.

If a second pass through the rules fails to yield a completely formal query and if one or more class 2 words were, in fact, tentatively dropped from the query, then the goal checker tries association dialog for these words. In Version 1 all the attributes which have purely alphabetic values are displayed with the corresponding relation names. The usual escape option NONE OF THE ABOVE is appended. If the user selects an attribute (e.g., supplier name), the source query is modified by the system to include this attribute (along with the relation name) immediately prior to each occurrence of the unrecognized alphabetic value. If the user selects NONE OF THE ABOVE, the word being questioned is dropped. Thus, this response causes the unknown word HELL to be dropped from the query WHAT THE HELL DOES JONES SUPPLY? and the resulting query WHAT THE DOES JONES SUPPLY? is one which the phrase transformer can handle

directly. Note that this word-dropping tactic would normally fail if the rest of the analyzer required strictly grammatical English as input.

To sum up, the goal checker tries several tactics if the output of the phrase transformer is not completely formal. These tactics include 1) tentative dropping of non-essential words with re-invocation of rule classes 2 through 11 and 2) association dialog followed by system modification of the user's source query and possible re-invocation of all rule classes.

3.4 Discussion of Analyzer

The outstanding feature of the Version 1 analyzer is its robustness: i.e., its ability to produce a complete, or reasonably complete, analysis of a large class of sentences whose structures have not been anticipated in the design of the rules and content of the lexicon. The principal contributors to this robustness are the word-dropping tactics, clarification dialog, and the class-by-class shift in emphasis of the rule classes from concern with English language aspects of the input to the essential relational and logical ingredients.

Nevertheless, in several ways the present analyzer is too simple for the job it has to perform. The next version will have to have both increased syntactic knowledge of English and increased semantic knowledge of the domain of discourse if it is to reduce the dialog burden on the user, handle a richer class of queries in the logical sense, and more astutely recognize excursions by the user outside of the domain supported by the data base itself. The robustness features of Version 1 will, of course, be retained in Version 2.

4 MENU DRIVER

The menu driver receives as input whatever formal output the analyzer has generated from the user's query and from any subsequent clarification dialog. The output of the analyzer may be a complete DEDUCE query, or an incomplete DEDUCE query consisting of fragmentary DEDUCE expressions. If it is complete, the menu driver passes it to the English language generator to translate it back into an unambigiuous English restatement of the user's query. If the output from the analyzer is incomplete, the menu driver tries to complete it by engaging the user in a dialog which depends on the partial formal query alone: at this point no use is made of the user's source query. The dialog is specific and is kept closely tied to the data base description. The menu driver tries to find out the user's intended query through his responses to multiple choice questions.

Depending upon what kinds of things are missing from the incomplete DEDUCE query, different questions will be presented to the user. Note that the output of the analyzer may be empty. This could happen when the user types in a null query such as U001 of Example D, or when the analyzer completely fails to analyze the user's query. In this case, the menu driver provides a means whereby the user can formulate his query through multiple choice questions alone (see Example D).

The menu driver is abnormally invoked whenever the user expresses dissatisfaction with the system's restatement of his query and requests <u>augmentation</u> of the system's version. Such augmentation is achieved through multiple choice questions which are presented in a sequence similar to that used for the null query (see S016 through U019 of Example B). There are no elements that can be detected as missing on purely logical grounds. Hence, such elements cannot be used to guide the questioning.

We shall now discuss the kinds of information that can be missing from the incomplete DEDUCE query, and later the strategy for discovering them. Finally, we briefly discuss how query fragments are pieced together to form a complete DEDUCE query.

4.1 Possible Missing Information

The following types of information may be missing from an incomplete DEDUCE query.

4.1.1 Query Type

The three types of queries recognized by the menu driver are distinguished by the kind of response expected. The response can be a list of items of information, a YES/NO

answer, or a count of items from the data base. If the query type is missing, the user will be asked a multiple choice question like S002 of Example D.

4.1.2 Query Focus

For the menu driver the query focus is defined for print type queries only and is the list of attributes to be printed. For instance, in Example B after the response U011, the system already knows the user wants some items of information printed. To find out exactly what information is to be printed, questions S012 and S013 are presented to the user. The responses indicate that the user wants to print the serial numbers, names, locations and ratings of suppliers.

4.1.3 Relation names

If the incomplete DEDUCE query contains an isolated value, but does not specify with which relation and attribute the value has to be associated, then the menu driver has to determine from the user the missing relation. For example, Houston may be the isolated value. In this case, the menu driver has to find out whether the user intends Houston to be a location of a supplier or a project. Another example is an isolated word such as 10. The menu driver has to find out whether 10 is used to specify the quantity of a part on hand, quantity on order, quantity of a part shipped, or even a date of shipment. Even if there are no isolated values, the menu driver may ask the user to select one or more relations: for example, if the user has indicated a desire to augment his query (see U003 in Example D).

4.1.4 Attribute names

Given an isolated value, the menu driver not only has to find a relation, but also an attribute of that relation with which the value should be associated. If the analyzer outputs a relation name without any corresponding attributes or if the user wants to specify a relation for any reason, the system asks him which attributes of that relation are relevant to his needs. An example is shown in S004 of Example D.

4.1.5 Values

If the incomplete DEDUCE query contains an incomplete fragment R(A), where R is a relation and A is an attribute of R, then the menu driver will ask the user if he wants to print or specify A. If he wants A to be printed, R(A) will be changed to R(*A). If he wants to specify A, he will be asked to enter a value for A such as shown in S005 and S006 of Example D. After the user has entered a value for A, the menu driver uses the analyzer function AUNORM to normalize

the value, and R(A) is changed to R(A V), where V is the normalized value. The menu driver also allows the user to enter a complete specification for the attribute A. For instance, in Example D, if the user had entered GOOD OR BETTER in U006, analysis by AUNORM would have produced ≥ GOOD. In this case, S007 would have been skipped.

4.1.6 Comparators

Given an incomplete fragment R(A V), the menu driver will seek a comparator to connect A and V. Whenever the domain of an attribute A (such as QOH, QOO, QSHIP, DATE, or RATE) possesses a total ordering, the comparators =,≠,<,≤,≥,> can be used to connect A and V. Otherwise, the eligible comparators are =,≠ only. For instance, in Example D after U006, the system has an incomplete fragment

SUPPLIER (RATE GOOD)

and tries to find out which one of the following is intended by the user:

SUPPLIER (RATE = GOOD)
SUPPLIER (RATE ≠ GOOD)
SUPPLIER (RATE < GOOD)
SUPPLIER (RATE ≤ GOOD)
SUPPLIER (RATE > GOOD)
SUPPLIER (RATE ≥ GOOD)

The options listed in question S007 are essentially generated from the above terms. Clearly, the English phrase for a comparator will be different for different attributes. For example, the comparator > will be phrased as GREATER THAN, BETTER THAN, and AFTER according as it is applied to QOH, RATE and DATE respectively. For the attribute DATE, the menu driver invokes the generator function GDATEPHRASE to generate the options presented to the user. In Version 2 we expect to employ the generator more uniformly to all attributes for the generation of options.

4.1.7 Joins

After the missing information of types 4.1.1 through 4.1.6 has been obtained, each term in the DEDUCE query will be in one of three forms:

*A, A = xn, A C V

where A, C, xn and V are attribute, comparator, join variable, and value respectively. Still, the DEDUCE query may not be complete, because clauses in the DEDUCE query are not properly joined. For example, we may have an incomplete DEDUCE query, FIND SUPPLIER(*SNAME) & PART(PNAME=PIPE). In this case, clauses SUPPLIER(*SNAME) and PART(PNAME=PIPE) may be joined relationally in any one of the following ways:

```
1)  FIND  SUPPLIER(*SNAME, SNO=x1) &
          SHIP(SNO=x1, PNO=x4) &
          PART(PNAME=PIPE, PNO=x4).
```

This complete query may be rephrased as: Print the name of every supplier who sent a shipment of a part named pipe.

```
2)  FIND  SUPPLIER(*SNAME, SNO=x1) &
          SHIP(SNO=x1, QSHIP=x7) &
          PART(PNAME=PIPE, QOO=x7).
```

This query is: Print the name of every supplier who sent a shipment having a quantity identical to the quantity on order for a part named pipe.

```
3)  FIND  SUPPLIER(*SNAME, SLOC=x2) &
          PROJECT(JLOC=x2, JNO=x3) &
          SHIP(JNO=x3, PNO=x4) &
          PART(PNAME=PIPE, PNO=x4).
```

This query is: Print the name of every supplier who has the same location as a project which receives a shipment of a part named pipe.

The menu driver now asks the user which of 1), 2) and 3) represents his intent. Note that 1), 2), and 3) may be presented together to the user, or one by one. Actually, in Version 1 of RENDEZVOUS the menu driver can find 1), 2), and 3). However, the present implementation of the generator cannot rephrase 2) and 3) in English. Therefore, only the restatement of 1) is presented to the user for verification.

4.2 Discovering Missing Information

The strategy of the menu driver is to discover any missing information of types 4.1.1 through 4.1.6 through dialog, and then systematically find feasible joins for piecing fragments together. In this section, we shall discuss techniques for finding missing information in incomplete fragments. The incomplete fragments that we encountered most often have the following forms: V, A, AV, R, R(V), R(A), and R(A V), where R, A and V are relation, attribute and value respectively. Given an incomplete DEDUCE query passed on from the analyzer, the menu driver will first find all the incomplete fragments, order them in the order of R(A V), R(V), V, AV, A, R(A), and R, and then try to complete them one by one. For each incomplete fragment, the menu driver first finds out what parts are missing from it, and then proceeds to discover these parts through dialog in the order: query type, query focus, relation, attribute, value, comparator. For each missing part, there is a multiple choice question which is generated from a question template stored in the system. Each template consists of a preamble, a list of options and actions associated with the options,

and a postscript. The menu driver will first display the preamble, options, and postscript as shown in Fig. 4.

```
WHICH PARTICULAR ITEM(S) OF THE FOLLOWING
INFORMATION ON SUPPLIERS DO YOU WANT TO          preamble
SPECIFY?
      1  SERIAL NUMBER OF SUPPLIER
      2  NAME OF SUPPLIER
      3  CITY IN WHICH SUPPLIER IS LOCATED          options
      4  RATING OF SUPPLIER
      5  NONE OF THE ABOVE
SELECT ONE OR MORE ITEMS BY NUMBER               postscript
```

Fig. 4 Typical Multiple Choice Question

For each option that the user selects, the corresponding action is invoked. Wordings of the preamble, options, and postscript in a multiple choice question are obtained from its template by combining prestored phrases with phrases extracted from the incomplete fragment the menu driver is currently working on, and from the data base. We store every multiple choice template in a table. For our sample data base, we have the following multiple choice templates:

TEMPLATE NAME	NO. OF OPTIONS	ITEMS TO BE DISCOVERED
M0T	3	query type
M0PT	4	relations to print
M1PJ	3	attributes of PROJECT to print
M1PP	5	attributes of PART to print
M1PS	4	attributes of SUPPLIER to print
M1PZ	5	attributes of SHIP to print
M1V	4	relations to specify
M2VJ	3	attributes of PROJECT to specify
M2VP	5	attributes of PART to specify
M2VS	4	attributes of SUPPLIER to specify
M2VZ	5	attributes of SHIP to specify
M3CS.SNO	2	comparator for SNO of SUPPLIER
M3CS.SNAME	2	comparator for SNAME of SUPPLIER
M3CS.SLOC	2	comparator for SLOC of SUPPLIER
M3CS.RATE	6	comparator for RATE of SUPPLIER
M3CJ.JNO	2	comparator for JNO of PROJECT
M3CJ.JNAME	2	comparator for JNAME of PROJECT
M3CJ.JLOC	2	comparator for JLOC of PROJECT
M3CP.PNO	2	comparator for PNO of PART
M3CP.PNAME	2	comparator for PNAME of PART
M3CP.QOH	6	comparator for QOH of PART
M3CP.QOO	6	comparator for QOO of PART
M3CP.PTYPE	2	comparator for PTYPE of PART
M3CZ.SNO	2	comparator for SNO of SHIP
M3CZ.JNO	2	comparator for JNO of SHIP
M3CZ.PNO	2	comparator for PNO of SHIP
M3CZ.QSHIP	6	comparator for QSHIP of SHIP
M3CZ.DATE	6	comparator for DATE of SHIP

The above multiple choice templates are organized by a graph called a tactical net as shown in Fig. 5. When the menu driver tries to complete an incomplete fragment, it first finds in the tactical net a node (or a set of nodes) that has one or more multiple choice questions for discovering the information missing in the incomplete fragment, then starting with the node, it presents to the user multiple choice questions one by one by traversing the tactical net.

4.3 Joining Query Fragments

When all missing information of types 1) through 6) has been discovered each term will be complete. However, as discussed in 4.1.7, some joins may still be missing. In general, there will be many alternative sets of join terms that will make the query logically complete. The techniques used to select an appropriate set of join terms are discussed in [11].

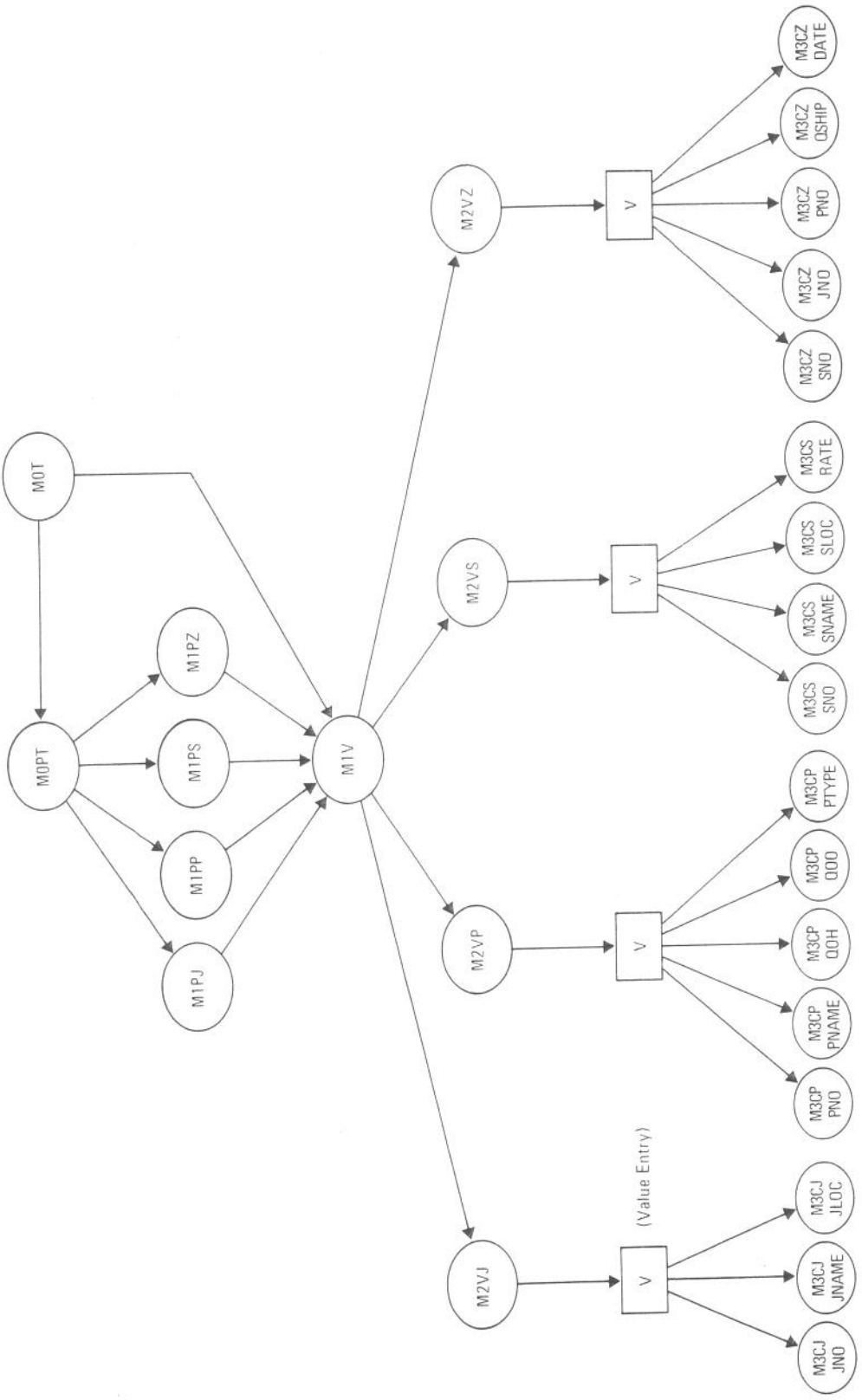

Figure 5. Tactical Net

5 GENERATOR

5.1 Goals of Generator

The goal of the generator is to provide the user with an English version of the formal query developed by the analyzer and menu driver. To the user this English version represents the system's present understanding of his intended query. Actually, however, it represents the system's understanding of the query he entered together with any modifications he made subsequently through dialog. In order for the user to verify that the intent of his query has been captured exactly by the system, the generated English should, as far as possible, be:

1) unambiguous
2) easy to understand
3) discriminating enough
4) not misleading.

It is a non-trivial task to satisfy all of these requirements simultaneously. For example, lack of ambiguity does not necessarily imply ease of understanding, as shown by the following sentence: IS IT THE CASE THAT, FOR EVERY PAIR OF SUPPLIERS x AND y, IF x IS LOCATED IN DETROIT, AND IF y IS LOCATED IN DETROIT, THEN x AND y ARE THE SAME. Such English may be perfectly unambiguous (at least for a mathematician), but would probably be incomprehensible to casual users. The sentence: IS THERE AT MOST ONE SUPPLIER LOCATED IN DETROIT? captures essentially the same idea and is far easier to understand.

The discrimination requirement 3) means that semantically close queries must be expressed in strikingly distinctive ways, so that the user does not mistakenly accept or reject whichever one happens to be generated. For example, in certain contexts, a user might ask about EMPLOYEES AGED 50 whereas he actually means EMPLOYEES OF AGE 50 OR OVER. Let us suppose that the analyzer understands the former phrase literally and produces the clause AGE = 50. It is then up to the generator to rephrase that part of the formal query so that the user realizes the system's misunderstanding. This means the rephrasings for AGE = 50 and AGE ≥ 50 must be sufficiently different to alert the user to the misunderstanding. Rephrasing AGE = 50 as EMPLOYEES AGED EXACTLY 50 considerably amplifies the distinctiveness. Whether or not this simple kind of amplification is sufficient for most users has to be experimentally tested.

As another example of the discrimination need, let us consider the following three queries:

Q1: PARTS SUPPLIED BY SUPPLIER s TO PROJECT j
Q2: PARTS SUPPLIED BY SUPPLIER s AND TO PROJECT j

Q3: PARTS SUPPLIED BY SUPPLIER s OR TC PROJECT j

These three queries are quite different, but in a non-trivial way. The above sentences are again not sufficiently different from one another to alert a user to a possible system misunderstanding. More discriminating rephrasings might be, for example:

Q1: LIST PARTS SHIPPED FROM SUPPLIER s TO PROJECT j
Q2: LIST PARTS WHICH ARE SHIPPED FROM SUPPLIER s AND ARE ALSO SHIPPED TC PROJECT j, BUT NOT NECESSARILY FROM s
Q3: LIST PARTS SHIPPED FROM SUPPLIER s;
ALSO LIST PARTS SHIPPED TO PROJECT j.

Incidentally, Q2 is outside of the logical/semantic coverage of RENDEZVOUS Version 1, because there are two semantically distinct occurrences of a relation (in this case, SHIP). Q3 actually consists of two successive but separate queries; and would have to be entered as two to avoid a misunderstanding by Version 1.

Requirement 4) is that the rephrasing should not mislead the user. The sentence: LIST THE POPULATION OF EVERY CAPITAL OF THE STATE OF CALIFORNIA is misleading, because it implies that there are several capitals of California. Given this rephrasing, a naive user might wonder what these capitals are (for example, are they county seats?) or else he may get the impression that the system is either confused or is working with false knowledge.

Like the rest of the RENDEZVOUS system, the generator is intended to be as independent as possible of the data base. That is, the knowledge about the particular data base must be in the data used by the generator, not in the generator's code itself. This objective is not yet fully achieved. To adapt Version 1 to new data bases, new data structures would be required to cope with new semantic and linguistic features not yet covered in the present implementation. Also, some sentence fragments (like the header -- see Section 5.3), are currently generated directly in the code, because they seem common to many data bases.

5.2 Knowledge used by the Generator

There are currently three types of knowledge used by the generator:

1) relational
2) semantic
3) linguistic.

The generator's relational knowledge consists of information concerning the underlying domains on which the data base is

defined, the relations defined on these domains, the attributes, and which of the attributes are keys. This basic knowledge is needed to support retrieval operations using a formal query language.

The generator's semantic knowledge consists of further information needed to support natural language query understanding -- including information such as whether or not a relation is linking, whether or not a domain has an ordering, etc.

The generator's linguistic knowledge consists of the English words and English sentence fragments needed to compose data base queries in English. The three types of knowledge are currently represented in relational tabular form, as follows:

1) The relational knowledge is contained in Table GRAR shown in Appendix C.1, which, for each relation of the data base and for each attribute of that relation, records the domain from which the attribute draws its values, whether or not the attribute is a key (or part of a key) of that relation, and, if the attribute is a linking key, what relation is its <u>home base</u>. An attribute of relation R is linking if it is the primary key of another relation, which is then called its home base. An example of a linking attribute is SNO in the SHIP relation. The home base of SNO is the SUPPLIER relation.

2) The semantic knowledge currently consists of Tables GDS and GRS given in Appendix C.2 and C.3 respectively: Table GDS indicating whether or not a particular domain has an ordering (the date domain is ordered, the supplier-serial-number domain is not); Table GRS indicating whether or not a particular relation is an entity relation or a linking relation. A relation can, of course, be both an entity relation and a linking relation (SHIP is an example).

3) The linguistic knowledge is contained in a variety of different tables (seven in the current implementation as shown in Appendix C.4 through C.10). One of them contains the external name associated with each entity relation. Another contains the external name and the preposition associated with each attribute of each relation (e.g. NAME OF). Still another contains external templates for comparator phrases. Two of the most important ones are the pattern tables: one is for pure entity relations and the other is for linking relations. Their description is better understood in conjunction with the kind of processing performed on them, so this will be deferred until the next section.

5.3 General Description of the Generator

The generator essentially takes as input a formal query on the relational data base and produces as output a rephrasing of that query in English. Input to the generator may be either DEDUCE character strings or an encoded tabular form of DEDUCE, very similar to that produced by the menu driver. There is a simple compiler which translates DEDUCE character strings into the internal tabular format.

Normally, the generator develops its output sentence from left to right: i.e., starting out with the null string, it successively appends text fragments on the right end of the partial sentence. In some cases non-trivial rearrangements of phrases occur in a post-processing phase, but this is abnormal. Consider the following simple example:

Q4: SUPPLIER (SLOC=HOUSTON, *SNO, SNAME=JONES)

The output of the generator for query Q4 will be the sentence:

PRINT THE SERIAL NUMBER OF EVERY SUPPLIER NAMED JONES AND LOCATED IN HOUSTON

There are three different parts of this sentence which are generated separately (but not independently):

1) the header: PRINT THE ... EVERY
2) the target list: SERIAL NUMBER OF
3) the body: SUPPLIER NAMED JONES AND LOCATED IN HOUSTON

The header depends on the query type. The generator currently handles the same query types as the menu driver: namely, FIND, EXIST, and COUNT. For FIND type queries, the header is most frequently:

PRINT THE ... EVERY

The word EVERY is dropped when the entity which follows is known to be unique. This is the case when the key of the corresponding entity relation is specified with equality (e.g., SUPPLIER NUMBER S14). For EXIST type queries, the header is: IS THERE ANY. For COUNT type queries, the header is: COUNT THE NUMBER OF.

The target list specifies the attributes to be printed. It only appears for PRINT type queries. The English names for the attributes and their prepositions are taken from the linguistic knowledge of the generator. The attributes in the list are separated by commas, except for the last two which are separated by AND. The format is A1, A2, ... A(n-1) AND An, for n > 2; A1 AND A2 for n = 2; and A1 for n = 1 . This sentence structure, where the target list

appears at the beginning, is appropriate in many cases. However, we shall see later some examples where it is more convenient to have the target list appear at the end of the sentence.

The body is normally the main part of the rephrased query. It is a sentence fragment expressing the specifications (or conditions) imposed on the entities that appear in the query, together with the various relationships between those entities. The generation of such a sentence is driven by the various patterns contained in the linguistic knowledge of the generator. Let us go back to query Q4:

Q4: SUPPLIER (SLOC=HOUSTON, *SNO, SNAME=JONES)

We have already considered the generation of the fragment: PRINT THE SERIAL NUMBER OF EVERY (i.e., the header and target list). We now consider the generation of the rest of the sentence using the SUPPLIER pattern table: (Patterns for the entity relations SUPPLIER, PROJECT and PART are given in Appendix C.9.)

condition	action
*	*
SNAME	NAMED-*
RATING	RATED-*
SLOC	LOCATED-IN-*

The routine which processes the entity relation patterns essentially proceeds from the top of the table down, one row at a time. Each row is of one of three types:

1)	*	*
2)	*	<English-Phrase>
3)	<Attribute>	<Attribute-English-Phrase>

Only types 1) and 3) appear in the supplier pattern table above. A row of type 1) indicates that the English name for the entity being processed should now be unconditionally appended to the partial sentence -- a routine is called which finds that name in a separate table. This routine determines from the query type whether the entity should be in the singular or in the plural. In the latter case, a table of irregular plurals is first looked up. If the entity name is not there, then it is regular and an S is appended to it. In the current implementation, entities in the plural appear in COUNT type queries only.

When a row of type 2) is encountered, the action specified by the second element of that row is immediately and unconditionally invoked. This type of row is used to introduce various glue words and phrases (such as FOR WHICH).

When a row of type 3) is encountered, an English fragment is generated only if the attribute of the row is specified in the query. In the case of an A = V specification, the English fragment is obtained by substituting the value V as specified in the query for the asterisk in the English phrase contained in the action component of the row. For example in processing the specification SLOC = HOUSTON in query Q4, the SLOC row of the SUPPLIER pattern table gives rise to the fragment LOCATED-IN-HOUSTON, which is then appended to the partial sentence.

The order in which the sentence fragments are generated is governed by the order in which the rows appear in the pattern, not by the order of the attribute clauses in the query. This generation order is crucial, and the pattern design must be done very carefully to avoid possible ambiguities or unwanted changes of meaning when clauses are added or suppressed. To illustrate this point, consider the following sentence: PRINT THE NAME OF EVERY EMPLOYEE AGED 60 WHO HAS A SECRETARY. Permuting the two attribute phrases radically changes the meaning of this sentence, since it yields: PRINT THE NAME OF EVERY EMPLOYEE WHO HAS A SECRETARY AGED 60.

The following more complicated example illustrates how elementary noun phrases like the body of the previous rephrasing are combined together.

```
Q5:     SUPPLIER ( *SNAME, SNO=x1, SLOC=HOUSTON )
        & SHIP ( SNO=x1, JNO=x2 )
        & PROJECT ( JNO=x2, JLOC=OAKLAND )
```

This query is rephrased as:

PRINT THE NAME OF EVERY SUPPLIER LOCATED IN HOUSTON WHO SENT A SHIPMENT TO A PROJECT LOCATED IN OAKLAND

Query Q5 contains two entity relation clauses, the SUPPLIER clause and the PROJECT clause, linked together by a linking relation clause, the SHIPMENT clause. The header PRINT THE ... EVERY and the target list NAME OF are generated as in the previous example. So are the two noun phrases corresponding to the SUPPLIER and PROJECT clauses: SUPPLIER LOCATED IN HOUSTON and PROJECT LOCATED IN OAKLAND. The SHIP relation links these two noun phrases together per the following specification:

<supplier phrase> WHO SENT A SHIPMENT TO A <project phrase>.

In this example, the SHIPMENT pattern is activated from the SUPPLIER relation via the linking key SNO. The SHIP relation actually has four different patterns, one for each linking key (SNO,PNO,JNO) plus one for the other attributes

(DATE and QSHIP). These four patterns can be sketched as follows:

<supplier> WHO SENT A SHIPMENT OF A <part> TO A <project>

<part> SHIPPED FROM A <supplier> TO A <project>

<project> TO WHICH A SHIPMENT OF A <part> WAS SENT BY A <supplier>

SHIPMENT OF A <part> FROM A <supplier> TO A <project>

The actual patterns are slightly more complicated, due to the attributes DATE and QSHIP (see Appendix C.10).

The design of adequate patterns is the critical aspect of applying the generator to any selected data base. Each pattern is a schema which can be instantiated into many different sentences: a pattern corresponding to a relation with n non-key attributes will give rise to 2**n different sentence structures depending on whether each attribute is specified or not in the query.

Care must be taken to ensure proper quality of all the possible sentences generated. Roughly speaking, the sentences generated must grow or shrink nicely, as attribute phrases are added or deleted. As an example of a sentence which does not shrink well, consider the following:

SUPPLIER SENDING PART NUMBER B15 TO A PROJECT LOCATED IN HOUSTON

If we remove the specification PNO = B15, the sentence becomes:

SUPPLIER SENDING TO A PROJECT LOCATED IN HOUSTON

which, although not technically ambiguous, is somewhat confusing, and as such should be avoided.

The patterns must also accommodate the variety of connectives which can be used in the attribute specifications: $=, \neq, \geq, >, <, \leq$. For example, the fragment A RED CAR could be a satisfactory rephrasing for a relational clause CAR (COLOR = RED), but the fragment A NOT RED CAR is not acceptable for the clause CAR (COLOR $\neq$ RED).

Finally the sentences produced must be combinable to form more complex sentences without creating ambiguities. The following is an example of an ambiguity created by embedding one sentence in another:

PART SHIPPED TO A <project phrase> DURING 1975

where <project phrase> is instantiated by PROJECT LOCATED IN HOUSTON. The combination PART SHIPPED TO A PROJECT LOCATED IN HOUSTON DURING 1975 is ambiguous, since the date phrase DURING 1975 may now be interpreted as if it applied to the clause LOCATED IN HOUSTON. Note that this problem arises when an entity (in this case PROJECT) is qualified by a space-oriented condition that is, or could be, time-sensitive. The fact that PART SHIPPED TO PROJECT NUMBER J12 DURING 1975 is unambiguous shows that appending a time-oriented clause does not necessarily give rise to a problem. However, fluency considerations would not permit a reordering of the clauses NUMBER J12 and DURING 1975.

Since dates can bear on many attributes, it is usually a good idea to have the date phrase as close as possible to the entity it qualifies, without however displacing a uniquely (or near-uniquely) identifying phrase such as NUMBER J12. In the case under consideration, the relevant SHIPMENT pattern is organized as follows:

PART SHIPPED <date phrase> TO A <project phrase>.

There are cases where the sentence structure so far discussed, <header><target list><body>, is not adequate for query rephrasing. This happens, for example, when the target list contains attributes of several distinct entities or when there is no noun form for a relation referenced by the target list. Let us take, as an example of the first category, the following query:

```
Q6:     SUPPLIER ( SNO=x1, *SLOC )
        & SHIP ( SNO=x1, *PNO, JNO=x2 )
        & PROJECT ( JNO=x2, JLOC=OAKLAND )
```

The target list in this query consists of the location of the supplier and the serial number of the part. To avoid problems with forward referencing, we generate the body of the sentence first, and then generate the target list with backward references. The sentence generated from this formal query is:

IN EACH CASE IN WHICH THERE IS A SHIPMENT OF A PART FROM A SUPPLIER TO A PROJECT LOCATED IN OAKLAND, PRINT THE SERIAL NUMBER OF THE PART AND THE LOCATION OF THE SUPPLIER.

Notice that the <body> of this sentence, i.e., SHIPMENT OF A PART FROM A SUPPLIER TO A PROJECT LOCATED IN OAKLAND is the same as that which would have been generated for the simpler sentence structure where the target list is at the beginning.

5.4 Examples showing specific features

5.4.1 Comparators

In a specification A C V, where A is an attribute, C a comparator, and V a value, the comparator C may be drawn from either the class =,≠,<,≤,≥,> or the class =,≠ depending on whether there is or is not an ordering associated with A.

The English phrases to express these comparators are contained in a table called GPRINTCOMP (see Appendix C.6). Each column of this table corresponds to an attribute, and each row to a comparator. A reference to the appropriate column number in that table is contained in the attribute row of the pattern, to be used when the attribute phrase is generated. This gives added flexibility in the output design. For example, QSHIP > 50 in a SHIPMENT clause will be rephrased as:
... SHIPMENT CONTAINING MORE THAN 50 OF ...
whereas QOO > 50 in a PART clause will be rephrased as:
... PART WITH A QUANTITY ON ORDER GREATER THAN 50 ...
The corresponding phrases in the GPRINTCOMP table are MORE THAN and GREATER THAN respectively.

The ≠ comparator receives somewhat special treatment. In many cases, a good rephrasing is obtained by simply pre-negating the entire attribute phrase: for example, SLOC ≠ HOUSTON can be rephrased as NOT LOCATED IN HOUSTON. The information that pre-negation is appropriate for an attribute is contained in the pattern used for the clause generation. If pre-negation is not appropriate, then the phrase to be used is found in the GPRINTCOMP table, as in the ordinary comparator case. For example QOH ≠ 50 will be rephrased as WITH A QUANTITY ON HAND OTHER THAN 50.

5.4.2 Key specification

Keys of entity relations receive a special treatment, due to the fact that they uniquely determine the entity. As pointed out in the CAPITALS OF CALIFORNIA example, the usual EVERY constuction may mislead the user. In this case, we suppress EVERY, and use special attribute phrases to be found in table GKP. For example SUPPLIER (SNO=S14, *SLOC) gets rephrased as: PRINT THE LOCATION OF SUPPLIER NUMBER S14, instead of: PRINT THE LOCATION OF EVERY SUPPLIER WITH SERIAL NUMBER S14.

Rephrasing key attribute phrases when the comparator is ≠ also receives special treatment. The formal query SUPPLIER (SNO≠S14, *SLOC) is rephrased as: PRINT THE LOCATION OF EVERY SUPPLIER OTHER THAN SUPPLIER S14, instead of: PRINT THE LOCATION OF EVERY SUPPLIER WITH A SERIAL NUMBER OTHER THAN S14, since the former carries more information.

5.4.3 Dates

Attributes taking their values in the date domain pose special problems. Dates can be of three types:

Day / Month / Year
Month / Year
Year

The English phrase to be generated differs according to the type of the date value. For example, SDATE = 14/5/1976 in a shipment clause can be rephrased as SENT ON MAY 14, 1976, whereas SDATE = 1976 cannot be rephrased as SENT ON 1976. A correct rephrasing for the latter attribute clause is SENT DURING 1976. In this case, the preposition generated depends on the value type. This is implemented by a small decoding routine which returns both the English form of the date and its type. The attribute phrases to be used are contained in a table called GDATECOMP, analogous to GPRINTCOMP described in 5.4.1. The date type determines which column of GDATECOMP should be used for generating the appropriate date phrase (see Appendix C.7).

5.4.4 Scope of negation

The combination of attribute phrases, one or more of which contains a local negation can result in unwanted extension of the scope of negation and possible ambiguity. As an example, consider: SUPPLIER (RATE≠GOOD, SLOC=DETROIT). Suppose this is rephrased as: SUPPLIER NOT RATED GOOD AND LOCATED IN DETROIT. Does the NOT apply to RATED GOOD only, or does it apply to LOCATED IN DETROIT also? To avoid this possible source of confusion, the generator yields:
SUPPLIER NOT RATED GOOD BUT LOCATED IN DETROIT.

5.4.5 Implicit entities

Sometimes it is necessary to generate explicit sentence fragments corresponding to entities which are only implicitly present in the formal query. In our present data base this occurs for the QSHIP attribute of the SHIP relation. This attribute refers to the quantity of a part shipped in the shipment. The implicit entity in this case is the part, which need not be explicitly specified or joined in the query, and yet is always implicitly present since every shipment is a shipment of some part. To illustrate the effect on generation, consider the query:

Q7: SHIP (*DATE, QSHIP≥50)

Rephrasing this query is not easy without explicitly refering to the part being shipped. The generator rephrases it as: PRINT THE DATE OF EVERY SHIPMENT CONTAINING 50 OR MORE OF SOME PART. Generation of the phrase SOME PART is triggered by the appearance of QSHIP in the formal query;

note that neither PART nor PNO appears in Q7 at all. The information necessary to support the conditional introduction of the implicit entity is contained in the pattern for the SHIP relation, in the row corresponding to the attribute PNO.

5.4.6 Focus in truth test queries

Consider the two sentences:

1) IS THERE ANY SHIPMENT TO A PROJECT LOCATED IN OAKLAND?
2) IS THERE ANY PROJECT LOCATED IN OAKLAND TO WHICH A SHIPMENT WAS SENT?

Given the semantics of the sample data base, these two sentences are logically equivalent -- the answer to 1) will be yes if and only if the answer to 2) is yes. However, our experiments have shown that some users do not perceive them to be logically equivalent, and it must be admitted that it takes a small amount of reasoning for someone to convince himself that they are. Now, if a user is conceiving his query in form 1) focusing on shipments, and if the system rephrases it in form 2) focusing on projects, there is a good chance that the user may be confused and incorrectly reject the system's version of his query.

We do not have a totally satisfactory answer to this problem. Version 1 presents to the user all of the possible rephrasings, corresponding to the focus being successively placed on all of the relational clauses in the query. The advantage of this approach is that, unless there is some other problem, at least one of the versions should correspond to the user's intent. There are however two disadvantages: for one thing the amount of material given to the user to read can become too large and confuse the user. Secondly, some careful users are worried by the apparent differences between the several versions and wonder what the actual differences are.

5.5 Present coverage

The class of queries presently supported by the generator is a subset of the so-called conjunctive queries (see [14]). These queries consist of ANDed relational clauses with join variables which are implicitly existentially quantified. The attribute clauses within the relational clauses are restricted to be ANDed only. Negation occurs only at the level of the attribute clauses, and only through the infix specification connectives ($<, \leq, =, \geq, >, \neq$). Joins are restricted to be equi-joins only.

A semantic restriction of the generator is that no more than one linking relation may appear in the formal query. Fully relaxing this restriction will require a good understanding

of how to decompose a query into smaller pieces. While this is easy from a logical standpoint, it is far from obvious from a linguistic and psychological point of view. Another semantic restriction of the generator is that it permits linking joins only. These are joins between a foreign key component of the key of a linking relation and its home base relation. Examples of joins which are not in that category are:

```
Q8:     PART ( *PNO, QOO=x1, QOH=x1 )

Q9:     SUPPLIER ( *SNAME, SLOC=x5 )
        & PROJECT ( JLOC=x5 )
```

There are also limitations of a linguistic nature. For example, the generator currently does not handle the case where the target list contains an attribute of a relation which has no noun form. An example of such a relation would be

```
COMP ( SUPPART  SUBPART  QUANT )
```

with the meaning that it takes the amount QUANT of the SUBPART to make one instance of the SUPPART. Such a relation does not have a noun form, and it is therefore more difficult to express in English the notion of the QUANT of a certain COMP -- contrast this with the notion of the DATE of a certain SHIPMENT. Note that this difficulty is mainly a question of output design, not of implementation.

A further restriction is that the target list may not contain attributes belonging to two different entities drawn from a common domain. An example of such a query would be:

```
Q10:    PART ( PNO=x1, *PNAME )
        & COMP ( SUBPART=x1, SUPPART=x2 )
        & PART ( PNO=x2, *QOO )
```

The generation difficulty here is to appropriately establish references to the correct PART, since the query refers to the name of one and to the quantity on order of the other. One way to do this is to name them locally: e.g., THE SUBPART and THE SUPERPART. Another is to refer to them by their order of appearance in the sentence: e.g., THE FIRST PART and THE SECOND PART.

5.6 Related work in generation

There has been little work reported on the subject of natural language generation, and none that we are aware of on the specific topic of natural language rephrasing of data base queries. Simmons and Slocum [16] and Slocum [17] discuss the generation of discourse from semantic nets using patterns and templates. Goldman [18] and Herskovits [19] deal with generation of natural language (both English and French) from deep conceptual structures. Heidorn [20]

provides general low level data structures and an interpreter for rules which operate on these data structures. His framework can be used both for analysis and for generation.

None of this work addresses the questions of output design, which we have found to be the most difficult part: questions of precision, prevention of ambiguity, discrimination, scope of negation, and fluency. Moreover, the generation mechanisms described do not cope with the specialized problems, features, and knowledge that are peculiar to data base query rephrasing (such as the special treatment needed for keys, target lists, condition clauses, join terms, etc.).

5.7 Discussion of Generator

The experiments we have conducted with subjects show that the current rephrasings have been clearly understood by them. There have been some instances where a subject incorrectly indicated the system's version to be identical in meaning to his own when in fact it was not. The cases when that happened fell into three categories:

1) the subject was not too sure what he wanted anyway and was happy with what the system was offering;

2) the subject was not actually reading the system's rephrasing;

3) the subject assumed that the system was only giving a partial rephrasing.

The last case occurred only once and came as a surprise to us. The subject was asking about suppliers in Los Angeles. Owing to the subject's manner of phrasing her query coupled with the subsequent dialog, the system dropped LOS ANGELES from its formal query. The system's restatement in English accordingly omitted any mention of Los Angeles. Yet the subject accepted the query as correct. When asked why she did this, she pointed out that the system had mentioned Los Angeles at some point in its dialog with her and therefore she assumed that the rephrasing was intended to be interpreted by her in the context of Los Angeles. We immediately modified the introductory advice displayed for the user when he logs on to Version 1. He is now advised to make sure that the system's restatement captures not merely the intent of his query, but the whole intent and nothing but the intent. Case 3) has not occurred again.

A number of extensions to the generator are needed as the system is extended overall to cope with a richer class of formal queries and an increased variety of types of data base relations. Some of these have already been discussed

in Section 5.5. Let us mention two more. The linear string representation of the generator's present output may prove to be difficult to read and understand in cases where the query contains many conditions. In such cases one might resort to a two-dimensional representation, such as:

```
PRINT THE ...  OF EVERY ... SATISFYING ALL  OF THE FOLLOWING
CONDITIONS:
            <condition 1>
            <condition 2>
                ...
            <condition n>
```

There is also a need for more feedback to the user during the analyzer's clarification dialog and the menu-driven dialog. With improved context feedback the user should find it easier to respond correctly to some of the system's questions. To support this, the generator must be able to translate partially developed formal queries into comprehensible English phrases. The generator in Version 1 has a very limited capability in this respect: for example, it will rephrase the clause SUPPLIER (SLOC=undefined) by SUPPLIER LOCATED IN ??

6 MENU DRIVEN EDITOR

This component allows the user to edit a query (whether it is his own or a system-generated paraphrase). It is necessary to provide casual users with editing facilities for several reasons. One reason is to allow a user to change his mind during the processing of his query. As the system engages the user in clarification dialog and menu-driven dialog, it presents a great deal of information about the data base, and this in turn may suggest to the user a different course of action to get what he needs. A second reason is to allow the user to change a portion of his query or the system's paraphrase when the system has misunderstood his intention.

To avoid burdening the user with having to learn editing commands, the query editor is based on a menu selection approach similar to that used in the analyzer and menu driver. To assist the user during the editing, the editor displays in the query viewport the query to be edited and, of course, updates this viewport as the query is modified. He can review his past editing by scrolling back over the history of all interactions.

The editor is called either from the helper (Section 9) or from the RENDEZVOUS supervisor (see Section 10). The call from the helper may come at any stage of query processing, providing the helper has itself been invoked either on the user's initiative or on the system's. The call from the supervisor comes only after the system's version of the query has been presented to the user. This is the point at which the user is given the option of editing either his original query or the system's paraphrase.

The editor accomplishes its task through edit cycles. During each edit cycle the user is given the opportunity to:

1) insert a word or phrase before a word in the query;
2) insert a word or phrase after a word in the query;
3) delete a word or phrase from the query;
4) replace a word or phrase in the query.

In principle, option 4) alone is sufficient, but we included the other three because casual users often do not realize how to use 4) to achieve 1), 2), and 3).

Consider the following example in which the user decides not to accept the system's paraphrase. At this point the system goes into the editing mode:

```
        YOUR PRESENT QUERY IS:
        LOS ANGELES SUPPLIERS OF PIPES SUPPLYING LOS ANGELES
        PROJECTS

        THE SYSTEM'S INTERPRETATION IS:
        PRINT THE NAME OF EVERY SUPPLIER LOCATED IN LOS ANGELES
        WHO SENT A SHIPMENT OF A PART NAMED PIPE TO A PROJECT
        LOCATED IN LOS ANGELES
-------------------------------------------------------------
S 009   TO RECOVER FROM THE INCORRECT INTERPRETATION OF YOUR
        QUERY, PLEASE INDICATE WHICH OF THE FOLLOWING ACTIONS
        YOU WOULD LIKE TO TAKE:
              1  MAKE CHANGE(S) IN YOUR PRESENT QUERY
              2  MAKE CHANGE(S) IN SYSTEM'S INCORRECT VERSION
              3  TYPE IN A NEW VERSION OF QUERY
              4  ABANDON THE QUERY ALTOGETHER
        SELECT ONE ITEM BY NUMBER
U 009   2


S 010   DO YOU WISH TO
          1  INSERT A WORD OR PHRASE BEFORE A WORD IN THE QUERY
          2  INSERT A WORD OR PHRASE AFTER A WORD IN THE QUERY
          3  DELETE A WORD OR PHRASE FROM THE QUERY
          4  REPLACE A WORD OR PHRASE IN THE QUERY
        SELECT EXACTLY ONE ITEM BY NUMBER
U 010   4


S 011   ENTER WORD OR PHRASE TO BE REPLACED; (OR JUST HIT
        'SEND' TO SKIP THIS REQUEST AND CONTINUE)
U 011   LOS ANGELES


S 012   THIS PHRASE OCCURS 2 TIMES.  WHICH OCCURRENCE(S) DO YOU
        WISH TO CHANGE
              1  FIRST
              2  SECOND
        SELECT ONE OR MORE ITEMS BY NUMBER
U 012   2


S 013   ENTER WORD OR PHRASE TO REPLACE 'LOS ANGELES'; (OR JUST
        HIT 'SEND' TO SKIP THIS REQUEST AND CONTINUE)
U 013   SAN FRANCISCO


        YOUR PRESENT QUERY IS:
        PRINT THE NAME OF EVERY SUPPLIER LOCATED IN LOS ANGELES
        WHO SENT A SHIPMENT OF A PART NAMED PIPE TO A PROJECT
        LOCATED IN SAN FRANCISCO
-------------------------------------------------------------
S 014   ANY OTHER CHANGE?
U 014   NO
```

When the user selects one of the editing options (see response U010 above), the system asks the user to provide the context of the insertion or the fragment of the query to be deleted or replaced (see S011 and U011 above). If the word or phrase given by the user occurs more than once, the editor asks the user which of the occurrences he wants changed (see S012 above). Next, in the case of an insertion or replacement, the new word or phrase is requested (see S013 above) and the appropriate action is taken. This completes an edit cycle and the user can initiate another one if he so desires (see S014).

On each of the interactions of an edit cycle the user is given the option to terminate the cycle. During the testing of the system we have found that it is not unusual for a user to change his mind in the middle of an edit cycle. He can then exercise this option to cancel or re-do the present edit cycle. Alternatively, through the use of the helper, he can select other options, including restarting the editing from scratch. When the user indicates that he has no more changes to make (S014, U014), possibly after executing several edit cycles, the edited query is passed to the analyzer for a new analysis.

7 DATA BASE RETRIEVER

The retriever receives a formal DEDUCE query which is output from the analyzer and the menu driver in combination. It applies this query to the relational data base and returns the answer to the user.

The sample data base contains relations which are small enough in extension to be stored and manipulated in virtual memory. In Version 2 we plan to interface with System R [14] for data retrieval from much larger data bases.

For easy manipulation in APL, each relation is stored as a two dimensional numeric array. Accordingly, all string values (e.g., MONTEREY) are stored in encoded numeric form. This in turn implies that 1) all string values appearing in each query must be encoded into their numeric representation and 2) all values in the answer which can be decoded into strings must be so decoded. The time spent for these transformations, however, is more than offset by the retrieval time saved through use of the vector and matrix manipulation features offered by APL. With small volumes of string output we have found that a net saving between 30% and 60% is achieved by this conversion. With larger volumes of string output the net time saving will be less, because more internal numeric representation has to be converted back into the string form.

Dates receive a special encoding. They are stored as 8-digit integers in which the first four digits denote the year, the next two the month, and the last two the day. For example, AUGUST 13, 1977 is stored as 19770813. This allows easy comparison using ordinary numeric comparators.

DEDUCE queries may include dates in which either the day or the day and month are omitted. In such cases the encoding for data base retrieval depends on the comparator as well as the date itself. Thus, DATE=•/10/1976 is encoded as DATE>19761000 & DATE≤19761031, because it means DURING OCTOBER 1976. On the other hand, DATE>•/10/1976 is encoded as DATE>19761031, because it means AFTER OCTOBER 1976.

Interpretation of a DEDUCE query is done in three phases. The first phase compiles the query into two tables and a target list. The second phase executes these tables and creates a relation that contains all attributes in the target list. Finally, the third phase assembles the answer for the user.

7.1 Compilation of DEDUCE

The first table of a compiled DEDUCE query holds the simple restrictions or qualifications. For example, for the query

```
FIND: SUPPLIER(*SNO=x1,*SNAME,*SLOC=DETROIT,*RATE=EXCELLENT)
      & SHIP(SNO=x1,PNO=x2)
      & PART(PNO=P37,PNO=x2)
```

the restriction table is:

RELATION	ATTRIBUTE	COMPARATOR	VALUE
SUPPLIER	SLOC	=	DETROIT
SUPPLIER	RATE	=	EXCELLENT
PART	PNO	=	P37

The second table holds the join terms of the query. For the above query this table is:

RELATION.1	ATTRIBUTE.1	ATTRIBUTE.2	RELATION.2
SUPPLIER	SNO	SNO	SHIP
PART	PNO	PNO	SHIP

where ATTRIBUTE.1 and ATTRIBUTE.2 show the attributes over which RELATION.1 and RELATION.2 are to be equi-joined.

Finally, the target list is a list of all the attributes to be printed. In the cited example, the list consists of SNO, SNAME, SLOC, RATE.

7.2 Query execution

Execution starts with the simple restrictions of the query (if any). If more than one restriction is on the same relation, the second restriction is applied on the result of the first, the third on the result of the second, etc. Each of the restrictions is performed by a single APL operator.

The execution of joins (if any) in a query is split into two passes. First, an index is generated between all pairs of relations that are to be joined starting with the pair having the smallest total number of tuples. This eliminates most of the tuples that would not eventually appear in the final join. Note again that, as in the case for restrictions, if a relation is to be joined more than once, the second index is applied to the result of the first, the third on the result of the second, etc. The second pass employs the indexes to guide the production of the final joins. The intermediate results of a query with multiple joins are usually very small due to the first pass and the fact that Version 1 applies joins on keys only.

The final step in the execution is a projection on the attributes specified in the target list; this produces a (possibly empty) relation containing the answer to the query.

7.3 Answer generation

The result relation of phase 2 is passed to an answer routine which, depending on the query type, returns appropriately phrased answers for the user. For a FIND (i.e, PRINT) type of query the response is one of the following:

1) THE ANSWER TO YOUR QUERY IS:

followed by a table which is the result of his query provided that the table contains no more than 20 lines (see example A);

2) THE ANSWER TO YOUR QUERY CONSISTS OF n LINES.
DO YOU WANT THEM LISTED?

(where n is greater than 20) followed by a table if the user replies yes;

3) NO OBJECTS IN THE DATABASE QUALIFY
AS AN ANSWER TO YOUR QUERY

possibly followed by some explanations for the emptiness of the result relation (see example B).

For a query of the EXIST type the possible answers are:

1) YES. THERE ARE n <focus> IN THE DATABASE
SATISFYING YOUR QUERY.
DO YOU WANT THEM (IT) LISTED?

where <focus> is the name of the relation representing the objects whose existence is being questioned (i.e., SUPPLIER, PART, PROJECT, or SHIPMENT); if the user replies affirmatively, there follows a listing for each of the n objects of the values of those attributes of the focus relation cited in the query (see example D);

2) NO. THERE IS NO <focus> IN THE DATABASE
SATISFYING YOUR QUERY.

possibly followed by some explanation (see example F).

For queries of the COUNT type the responses are the same as for queries of the EXIST type, except that the YES/NO part is eliminated.

Under certain conditions, when the relation containing the answer is empty, the answer routine gives the user one or more reasons for this emptiness. The purpose is, of course, to help the user in formulating his later queries. There are many reasons (combinatorial on the number of conditions specified in the query) that may cause the result relation

to be empty. Version 1 supports two kinds of reasons, chosen for the relatively small amount of processing needed to discover whether and how they may apply in any specific instance.

The first kind of reason explains in English which simple qualification conditions in the query would by themselves have yielded an empty result relation. The second kind of reason explains which arguments of joins would by themselves have yielded an empty result relation, even though none of their simple qualifications did so. For example, consider the following query Q and the retriever's response A:

Q: IS THERE ANY SUPPLIER NAMED JONES AND RATED EXCELLENT WHO, DURING MARCH 1976, SENT A SHIPMENT OF A PART NAMED CAM TO A PROJECT NAMED MOJAVE LOCATED IN MONTEREY?

A: THERE IS NO SUPPLIER IN THE DATABASE SATISFYING YOUR QUERY.
NOTE THAT IN THIS DATABASE
1. THERE IS NO PART NAMED CAM
2. THERE IS NO PROJECT NAMED MOJAVE LOCATED IN MONTEREY
3. THERE IS NO SUPPLIER NAMED JONES RATED EXCELLENT

Explanation 1 is a reason of the first kind (emptiness of a simple qualification). Explanations 2 and 3 are reasons of the second kind (emptiness of join arguments). Although there exists a project named MOJAVE, and MONTEREY is the location of at least one project, these two conditions are not satisfied simultaneously. The same is true for suppliers named JONES and suppliers rated EXCELLENT.

The retriever notes any cases of emptiness resulting from simple qualifications or join arguments during execution of the query. After query execution these notes are translated into simple English using a portion of the generator described in Section 5.

We believe that the explanations offered by the answer routine are much more helpful to the casual user than a flat no answer. With the answers we provide, the user should be able to avoid many unnecessary queries and proceed to formulate his later queries with a better knowledge of the data base.

8 DISPLAY SUBSYSTEM

8.1 Design Approach

Users interact with RENDEZVOUS via the display subsystem. The goal of this subsystem is to give casual users a familiar, easy-to-use interface for communicating with RENDEZVOUS. The special interface needs of computer scientists and professional programmers have accordingly been sacrificed where necessary.

To gain insight into users' interface preferences, we performed informal experiments with a variety of users:

1) to shake down the RENDEZVOUS system itself;

2) to provide a source of test queries created by actual users;

3) to expose new interface problems we had not considered and make the design of the display subsystem more user-oriented.

The informal experiments involved giving subjects a sequence of simple problems to solve. Each subject mentally translated each problem into one or more queries expressed in English of his choosing. The subject then entered these queries into RENDEZVOUS. An experimenter was present observing the subject, who was encouraged to comment on any system aspect. The experimenter would also occasionally solicit comments on specific items of interest. Over 30 such experiments were performed.

8.2 Components of the display subsystem

The display subsystem has four main parts:

1) formatted screen display;
2) adapted keyboard;
3) self-documentation;
4) history keeper.

We shall now discuss these parts separately. However, the reader should realize that the user sees them as a combined whole.

8.2.1 Formatted Screen Display

Fig. 6 and Fig. 7 show samples of the formatted screen display. The display screen is 24 lines by 80 columns and is divided into four viewports (or fields): the query, history, input, and message viewports.

The query viewport at the top of the screen contains a copy

of the current state of the user's query. The current query state reflects the changes (e.g., editing, spelling correction, rephrasing) the user or the system has made either to the user's original query or to an incomplete system rephrasing. The query viewport may also contain indications of the system's present understanding of the user's query (for example, see the query viewport output in Fig. 6 and 7.)

Below the query viewport is the history viewport, which is a window to an online record (history) of the session up to and including the current question. The history may be scrolled vertically by special keys which are described later.

Scrolling allows users to answer the current question based on earlier questions and responses. This memory aid is useful, e.g., for users who have forgotten what happened in past interactions and who need to review what has been specified before answering the current question. A drawback of scrolling, however, is that it is an unfamiliar operation to present users (future users may be entirely different in this respect). In the experiments scrolling was not used extensively. When it was used, subjects tended to scroll back at most one screen (i.e., one history viewport full of information).

Together, the history and query viewports always occupy a fixed number of lines--currently 19 (the screen has 24 in all). However, within this limitation, the query viewport expands and contracts dynamically to accommodate (without wasted lines) the information displayed therein. To prevent the history viewport from becoming too small to be usable, the query viewport is restricted to a maximum size of 11 lines. This still allows the largest system question so far encountered (8 lines long) to be completely viewed in the history viewport without scrolling.

Below the history viewport is the input viewport, which receives user input from the keyboard. Its size is fixed at 3 lines. This size appears to be sufficient to accommodate single-sentence -- and some double-sentence -- queries. With a 2-line input viewport, one subject's query overflowed the input viewport.

Finally, below the input viewport is the message viewport, in which the system places messages to the user about special input-output conditions. For example, the message SCROLL FORWARD FOR MORE tells the user that more output may be seen by scrolling forward. These messages are displayed with heightened intensity to catch the attention of the user.

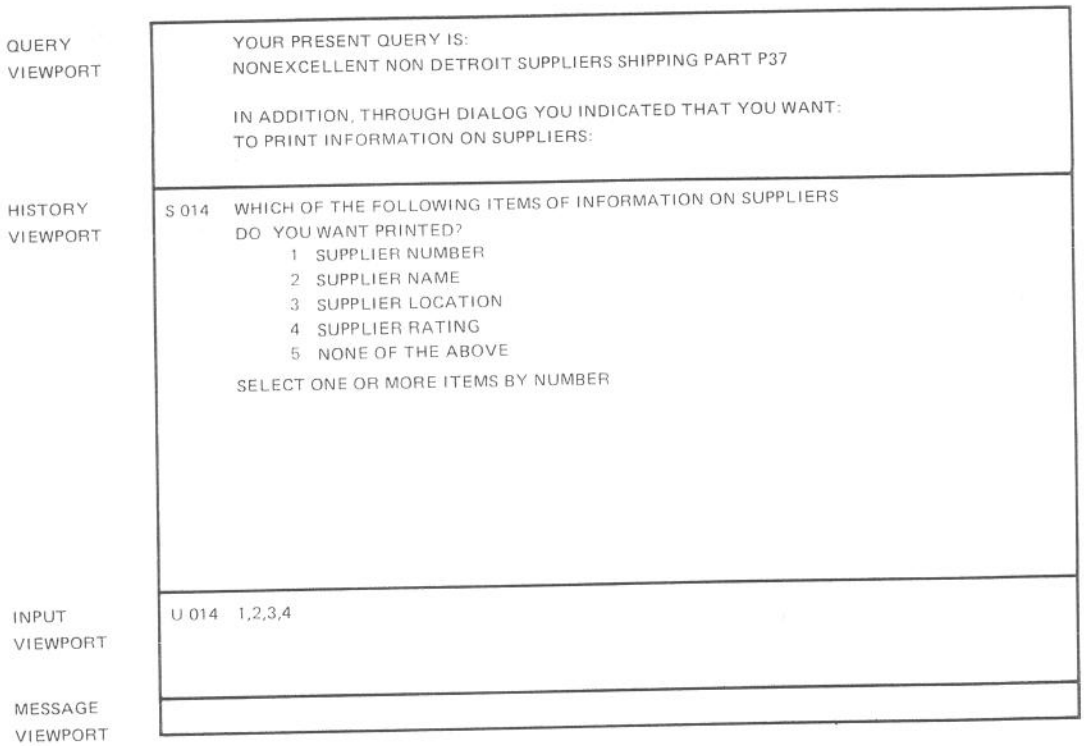

Figure 6. Display with Feedback

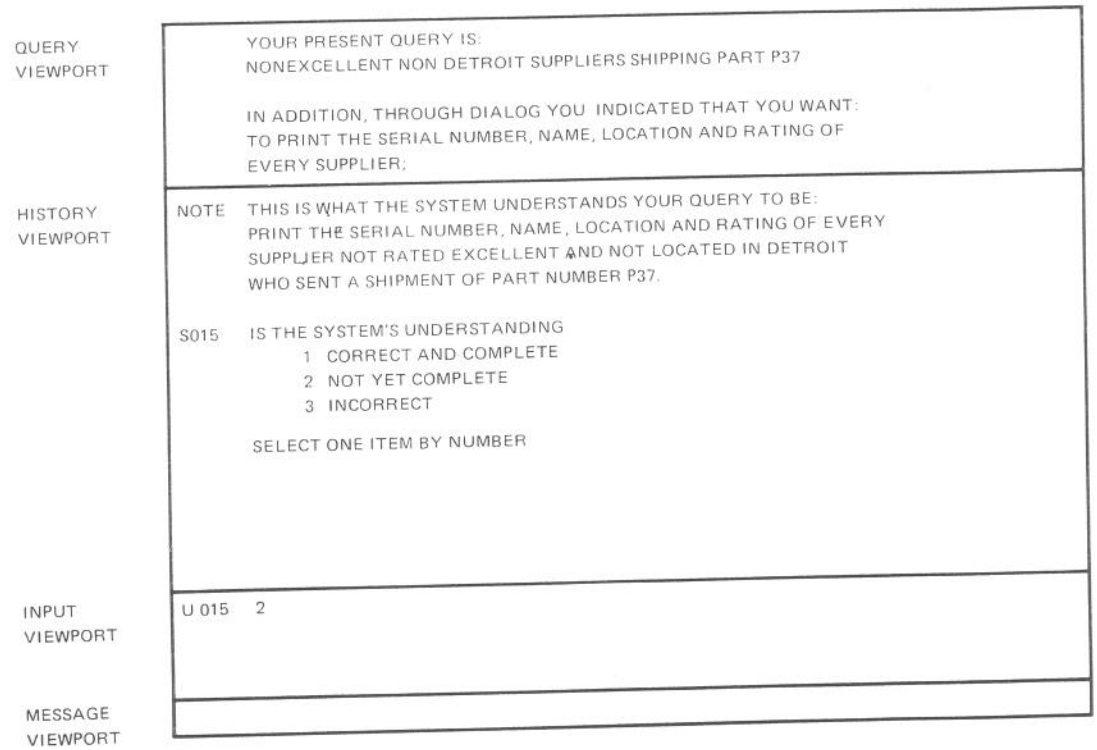

Figure 7. Display with more Feedback

8.2.2 Adapted Keyboard

The keyboard of the IBM 3277 display terminal provides more options than are needed for RENDEZVOUS input. To present a simpler, but still adequate, set of options to casual users, certain parts of the keyboard were made inaccessible through the use of cardboard covers. The adapted keyboard (Fig. 8) consists of data entry keys, data editing keys, and special function keys. The special keys shown in Fig. 8 are summarized in Fig. 9.

The 42 data entry keys are the SEND, numeral, letter, punctuation, and space keys. The SEND key is used to send the current contents of the input viewport to RENDEZVOUS. The keyboard is then effectively locked until the next output is displayed for the user. The SHIFT key is one of those rendered inaccessible to the user. The principal benefit of this action is avoidance of errors in use of the SHIFT key, including inadvertent entry of upper case APL characters (many of which are Greek letters). A minor disadvantage of making the SHIFT key inaccessible is that all letters are in upper case. Hence, proper names, e.g., supplier or project names, cannot be distinguished from other words by capitalization of the first letter.

The three data editing keys are CARRIAGE RETURN/LINE FEED, CURSOR RIGHT, and CURSOR LEFT. The cursor is a visible screen position marker which indicates where input will next be entered on the screen. Casual users who had not used a display terminal before needed some instruction on how to use the cursor.

The six special function keys are CONTINUE, HALF-FORWARD, FORWARD, HALF-BACK, BACK, and HELP. The first five keys control scrolling. To scroll the interaction history one screen backward in time, one depression of the BACK key suffices. Similarly, to scroll forward one screen, the FORWARD key should be depressed once. The HALF-FORWARD and HALF-BACK keys allow the user to proceed half a screen at a time. The CONTINUE key advances the screen in one step to the most recent interaction in the history. The HELP key may be pressed by the user whenever the system requests input and the user desires assistance from the system.

USING THE COMPUTER TERMINAL AND KEYBOARD

TO ENTER INPUT TO THE SYSTEM, JUST TYPE IN THE INPUT AND PRESS THE 'SEND' KEY ON THE BOTTOM RIGHT OF THE MAIN KEYBOARD. PLEASE NOTE: THE '?' CHARACTER IS NOT AVAILABLE ON THIS KEYBOARD.

WHENEVER THE MESSAGE 'SCROLL FORWARD FOR MORE...' APPEARS AT THE BOTTOM OF THE SCREEN AND YOU ARE READY FOR MORE SYSTEM OUTPUT, PRESS EITHER OF THE '½ SCREEN FORWARD' OR '1 SCREEN FORWARD' KEYS ON THE FAR RIGHT SIDE KEYBOARD.

TO SEE PREVIOUS SYSTEM OUTPUT, PRESS EITHER OF THE '½ SCREEN BACK' OR '1 SCREEN BACK' KEYS ON THE FAR RIGHT SIDE KEYBOARD.

PRESS THE 'HELP' KEY ON THE FAR RIGHT SIDE KEYBOARD WHENEVER YOU DESIRE HELP FROM THE SYSTEM.

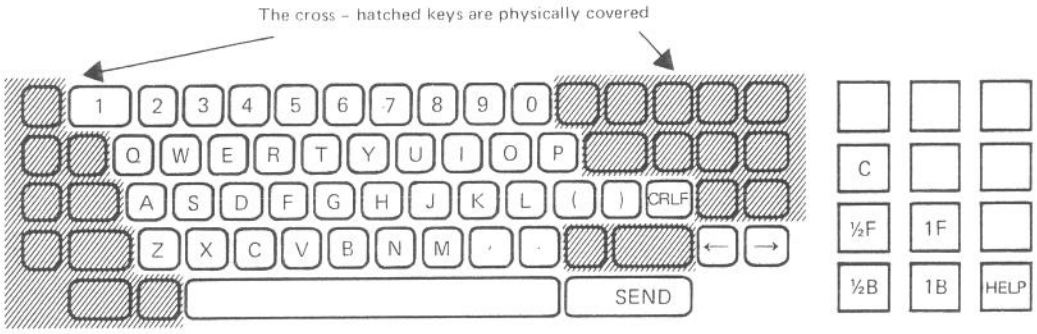

Figure 8. Adapted IBM 3270 Keyboard with Instructions

Abbreviation (from Fig.8)	Actual Keyboard Label	Function
CRLF	CARRIAGE RETURN/ LINE FEED	Move cursor to start of next input line
SEND	SEND	Send input to system
→	→	Move cursor right
←	←	Move cursor left
C	CONTINUE	Bring question into view for which input is currently requested
½F	½ SCREEN FORWARD	Display history ½ screen forward
1F	1 SCREEN FORWARD	Display history 1 screen forward
½B	½ SCREEN BACK	Display history ½ screen back
1B	1 SCREEN BACK	Display history 1 screen back
HELP	HELP	Request help from system system

Figure 9. Guide to Special Function Keys

8.2.3 Self-Documentation

The third part of the display subsystem is the self-documentation. An abbreviated form of this documentation appears in hard-copy form attached to the keyboard immediately above the keys. A more comprehensive form is stored in RENDEZVOUS as part of the information accessible to the user during the session whenever he requests HELP.

8.2.4 History keeper

To enable the user to review past interactions between himself and the system during the current session, all information entered from the keyboard and all RENDEZVOUS output are accumulated in a history variable in memory as they are generated. For space reasons, the history variable is dumped periodically from memory into a scratch history file. Through appropriate control settings various internal versions of the user's query can be included in the recorded history for debugging purposes.

8.3 Summary of display subsystem

The display subsystem is designed to give inexperienced computer users a familiar, easy-to-use interface for communicating with RENDEZVOUS. We aimed at ease-of-use by adopting clearly separated viewports for system feedback, interaction history, and keyboard entry; also by adopting a simple multiple-choice question and answer display, a simplified keyboard with very few special function keys, and easily accessible documentation on how to use the system. Of course, more experimentation with casual users is needed to make further improvements in the display subsystem.

9 HELPER

The helper component provides assistance to the user during his interaction with the RENDEZVOUS system. By pressing the HELP key, the user can call the helper at any time the system is requesting input. When the user calls for help, the system switches from the query processing mode to the help mode.

In help mode the system offers a menu of help options to the user (see example B). The menu offers:

1) information on the data base;
2) help with the keyboard;
3) a facility for invoking the query editor;
4) a facility for dropping the current query and entering a fresh query;
5) a facility for resuming the processing of the user's current query;
6) a facility for ending the user's session.

Option 1 offers the user information about the data base. This information is stored in canned messages which are organized around the data base relations, supplier, project, part and shipment. The user can request to see information on any or all of the above relations. For each relation, the messages basically describe its attributes and give samples of their values (see example B).

Option 2 displays to the user simple instructions on how to use the keyboard. These instructions are also stored in canned messages. After the information for either option 1 or 2 has been displayed to the user, the system stays in the help mode and the user can therefore ask for more information. The system switches back to the query processing mode when the user selects option 3, 4, or 5. The effects of these options and option 6 are self-explanatory.

10 SUPERVISOR

The principal role of the supervisor is to invoke the other RENDEZVOUS components (but not the display support) as necessary. Its first action in a session is to ask the user to type in his name. It then asks him whether he wants to get acquainted with the RENDEZVOUS system. If he enters YES, the supervisor presents him with a brief introduction about the system's behavior, the data base, the keyboard, and the help facility. If the user enters NO, this information is simply not displayed.

The supervisor proceeds to ask the user to type in his query, after which it invokes first the analyzer, then the menu-driver, and then the generator. However, this sequence of invocation is interrupted, and may even be abandoned, if the user depresses the HELP button (see Section 9).

When the generator has completed its translation of the DEDUCE query into an unambiguous English paraphrase, the supervisor asks the user whether he agrees with the system's interpretation. If he does, the supervisor invokes the data retriever to execute the DEDUCE query and thereby get the answer from the data base.

If the user indicates that the system's interpretation is incomplete, the supervisor invokes the menu driver to allow him to augment the system's version by introducing additional specifications or items to be printed. After that, the supervisor again invokes the generator to produce a new interpretation for the user's approval.

If the user indicates that the system's interpretation is incorrect, the supervisor asks him to choose between editing his own query, editing the system's version, and entering a new query altogether. If he chooses to edit, the supervisor invokes the menu-driven editor. When editing is finished, the supervisor once again invokes the analyzer, menu driver, and generator.

After the retriever has delivered the answer from the data base to the user, the supervisor asks the user whether his next query is a simple modification of the one just executed. This question is intended to encourage the user to take advantage of the context of his immediately preceding query in formulating his new query, whenever possible. If the user responds negatively, the supervisor requests the user to enter his new query, and the whole process is repeated.

11 CONCLUSION

11.1 Experience with Version 1

As of August 1977 some 30 subjects have tried the system. Each subject was presented with a sequence of simple problems typed on cards. As an example, one problem was as follows:

> A colleague claims that pumps were shipped to only one city during February 1976. To which projects and which cities were pumps actually sent in this month?

The subject was advised to translate each problem into one or more data base queries, using his own words and phrasing. Prior to their sessions with the system, most of the subjects were not familiar with the data base. However, we found that a superficial, one-screen tutorial on its contents was enough to get them started on the problems. The system would bring additional information to light as the user interacted with the system to solve each problem, and whenever the user made an explicit request for help. Most subjects, but not all, seemed to prefer this on-the-job approach to acquiring knowledge about the data base -- rather than lengthy computer-assisted instruction immediately after logging on at the terminal.

All of the problems whose solution entailed the execution of at least one join were of such a character that the join(s) could be <u>implied</u> by appropriate use of English. The subjects invariably chose to imply, rather than specify, these joins; and, of course, Version 1 was well equipped to handle these implied joins.

While many of the subjects were computer scientists (due to their ready availability in this laboratory), other specialties were also represented: in particular, there were secretaries, librarians, administrative people (including one purchasing agent), and medical staff -- people with little or no experience in using computers. Incidentally, about half the computer scientists were visiting from Europe, and for them English was a second language. Accordingly, we believe that, within the logical class of queries tested by the problem set (and this class was deliberately planned to be within the logical capabilities of the system), the system was subjected to some rather severe stresses in the form of mis-communication, misunderstanding, and departures (sometimes quite radical) from normal English grammar. Of course, these experiments can only be regarded as preliminary, but they do suggest that, for this domain of discourse, we have achieved a reasonably high degree of robustness, and this is largely due to the heavy emphasis on feedback to the user.

11.2 Future Directions

In Version 1 we placed a major emphasis on conversation concerning the data base extension, and we believe that we have demonstrated a substantial part of the conversational behavior essential to successful communication between a casual user and a formatted data base. However, there are many obvious improvements which can and should be incorporated into Version 2.

The speed of the Version 1 analyzer is unsatisfactory: a time-shared 370 model 168 is required to bring the average analysis time down to less than one minute. A major factor in this slowness is the use of an interpretive implementation language (APL) for which a compiler was not available.

Another obvious requirement for Version 2 is to support queries that are significantly more complicated from the logical standpoint. This will necessitate the adoption of an internal query language which defines a much richer logical class of queries, permitting negation of arbitrary scope, numeric quantifiers, two or more variables with a common range, the subquery concept of DEDUCE [10], the GROUP BY concept of SEQUEL [14], and much freer use of library functions.

Version 2 must also be able to answer queries about 1) the system's capabilities and 2) various intensional aspects of the data base, including the immediate operating environment of the micro-world supported by the data base extension. To accomplish this, we shall need a knowledge base of broader scope and greater sophistication than that of Version 1. In this regard we expect to draw upon the recent work of Schank and Abelson [12] and Zadeh [13].

An additional goal for Version 2 will be the unification of the knowledge base used by the analyzer, menu-driver, generator, and helper. This is expected to be a very difficult goal to attain fully, because some of these components appear to need quite different forms of knowledge about the data base.

To provide efficient retrieval from large relational data bases, Version 2 will interface with System R [14].

12 ACKNOWLEDGMENT

We wish to thank: Professor L. A. Zadeh of U.C. Berkeley for participating in the testing of Version 1 and providing a number of constructive suggestions, particularly in regard to the helper and menu-driver; G. Hogsett, D. Mokski, and P. Sih for helpful system advice and part of the code for the display subsystem; and Professor D. V. Ribbens of Universite de Liege, Belgium for implementing an interface package which translates the analyzer output into a form acceptable to the menu-driver.

13 REFERENCES

1 C. H. Kellogg, J. Burger, T. Diller, K. Fogt, "The CONVERSE Natural Language Data Management System: Current Status and Plans", Proc. ACM Symposium on Information Storage and Retrieval, Univ. of Maryland, College Park 1971 pp 33-46

2 F. P. Thompson, P. C. Lockemann, B. H. Dostert, R. Deverill, "REL: A Rapidly Extensible Language System", Proc. 24th ACM National Conference, New York, ACM 1969 pp 399-417

3 W. A. Woods, "Progress in Natural Language Understanding: An Application to Lunar Geology", AFIPS Conference Proceedings, 1973, vol 42, pp 441-450

4 W. J. Plath, "REQUEST: A Natural Language Question-Answering System", IBM J Res and Dev, vol 20, no 4, pp 326-335, 1976

5 V. M. Briabrin, D. A. Pospelov, "DILOS -- Dialog System for Information Retrieval, Computation, and Logical Inference", in D. Michie (ed.), "Machine Intelligence", vol 19, American Elsevier, New York 1977

6 D. L. Waltz, "An English Language Question Answering System for a Large Relational Data Base", to be published in Comm. ACM 1978

7 G. G. Hendrix, "The LIFER Manual", Technical Note 138, Stanford Research Institute, Menlo Park, Calif., 1977

8 J. Mylopoulos, A. Borgida, P. Cohen, N. Roussopoulos, J. Tsotsos, H. Wong, "TORUS: A Step towards Bridging the Gap between Data Bases and the Casual User", Information Systems, vol 2, no 2, pp 71-77, 1976

9 E. F. Codd, "Seven Steps to Rendezvous with the Casual User", Proc. IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, April 1-5, 1974, in J. W. Klimbie and K. I. Koffeman (eds.), "Data Base Management", North-Holland 1974

10 C. L. Chang, "DEDUCE -- A Deductive Query Language for Relational Data Bases", in C. H. Chen (ed.), "Pattern Recognition and Artificial Intelligence", Academic Press, New York 1976

11 C. L. Chang, "Finding Missing Joins for Incomplete Queries on Relational Data Bases", IBM Research Report RJ2145, San Jose, California, January 1978

12 R. C. Schank, R. P. Abelson, "Scripts, Plans, Goals, and Understanding", Halstead Press 1977

13 L. A. Zadeh, "PRUF: A Meaning Representation Language for Natural Languages", to be published in Information & Control

14 M. M. Astrahan et al, "System R: Relational Approach to Database Management", ACM Trans. on Database Systems, vol 1, no 2, June 1976, pp 97-137

15 A. K. Chandra, "Implementation of Conjunctive Queries in Relational Data Bases", IBM Technical Disclosure Bulletin, vol 18, no 10, March 1976, pp 3520-3526.

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

17 J. Slocum, "Speech Generation from Semantic Nets", Proc. 13th Annual Meeting of the Association for Computational Linguistics, Boston, Massachusetts, October 30 - November 1, 1975.

18 N. Goldman, "Computer Generation of Natural Language from a Deep Conceptual Base", A.I.Memo 247, Artificial Intelligence Laboratory, Stanford University, January 1974.

19 A. Herskovits, "The Generation of French from a Semantic Representation", A.I.Memo 212, Artificial Intelligence Laboratory, Stanford University, August 1973.

20 G. Heidorn, "Natural Language Inputs to a Simulation Programming System", Technical Report NPS-55HD72101A, Naval Postgraduate School, Monterey, California, October 1972

21 T. W. Finin, "Casting the RENDEZVOUS Analyzer Rules in Augmented Transition Network Form", IBM Research Report RJ2146, San Jose, California, January 1978

APPENDIX A DEDUCE SYNTAX FOR RENDEZVOUS VERSION 1

The complete DEDUCE sublanguage is described in [10]. However, for RENDEZVOUS Version 1, we restrict ourselves to a subclass of DEDUCE queries. In this subclass a query cannot contain subqueries, numerical quantifiers, or functions except COUNT and EXIST, and the same relation cannot appear in more than one clause. The syntax of DEDUCE for this subclass is given as follows:

1. <char> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N
| O|P|Q|R|S|T|U|V|W|X|Y|Z|-|/
2. <digit> ::= 0|1|2|3|4|5|617|8|9
3. <integer> ::= <digit>
| <integer><digit>
4. <string> ::= <char>
| <digit>
| <string><string>
5. <value> ::= <string>
6. <variable> ::= x<integer>
7. <relation> ::= <char><string>
8. <attribute> ::= <char><string>
9. <comparator> ::= <
| ≤
| >
| ≥
| =
| ≠
10. <tuple> ::= <attribute>=<variable>
| <attribute><comparator><value>
| <tuple>,<tuple>
11. <*tuple> ::= *<attribute>
| *<attribute>=<variable>
| *<attribute><comparator><value>
| <*tuple>,<*tuple>
| <*tuple>;<tuple>
| <tuple>,<*tuple>
12. <clause> ::= <relation>(<tuple>)
13. <*clause> ::= <relation>(<*tuple>)
14. <conjunction> ::= <clause>
| <conjunction>&<conjunction>
15. <*conjunction> ::= <*clause>
| <*conjunction>&<*conjunction>
| <*conjunction>&<conjunction>
| <conjunction>&<*conjunction>
16. <*clause-conjun> ::= <*clause>
| <*clause>&<conjunction>
| <conjunction>&<*clause>
| <conjunction>&<*clause>&<conjunction>
17. <find-type-query> ::= FIND <*conjunction>
18. <exist-type-query> ::= EXIST <*clause-conjun>
19. <count-type-query> ::= COUNT <*clause-conjun>

APPENDIX B Rule Syntax

Definitions of <char>, <integer>, and <comparator> are given in Appendix A.

```
 1.  <cstring>    ::=  <char>
                    |  <cstring><char>
 2.  <word>       ::=  <cstring>
                    |  (
                    |  )
                    |  <comparator>
 3.  <function>   ::=  <cstring>
 4.  <lexclass>   ::=  <integer>
 5.  <argument>   ::=  <integer>
 6.  <termno>     ::=  <integer>
 7.  <atom>       ::=  <word>
                    |  <lexclass>
 8.  <sequence>   ::=  <cstring>
                    |  <lexclass>
                    |  <sequence>,<sequence>
 9.  <lterm>      ::=  <atom>
                    |  ¬<atom>
                    |  *
                    |  *<char>
                    |  <lexclass>&<lexclass>
                    |  (<sequence>)
                    |  (<sequence>,•)
                    |  <lexclass><function><argument>
10.  <option>     ::=  ?<cstring>?<cstring>
                    |  <option>?<cstring>
11.  <rterm>      ::=  <word>
                    |  <termno>
                    |  <termno>?<termno>?<termno>
                    |  <option>
                    |  <function><argument>
                    |  <termno>_<function>_<cstring>
                    |  <cstring>_X<termno>
12.  <left>       ::=  <lterm>
                    |  <left> <lterm>
13.  <cursor>     ::=  +<integer>
                    |  -<integer>
                    |  %
14.  <right>      ::=  <rterm>
                    |  <right> <rterm>
                    |  <right> <cursor>
15.  <rule>       ::=  <left> -> <right>
```

APPENDIX C Knowledge Base Used by Generator

C.1 Relational Knowledge, Table GRAR:

Relation	Attribute	Domain	Key	Homebase
SUPPLIER	SNO	SNO	1	0
SUPPLIER	SNAME	SNAME	0	0
SUPPLIER	SLOC	LOCATION	0	0
SUPPLIER	RATE	SRATING	0	0
PART	PNO	PNO	1	0
PART	PNAME	PNAME	0	0
PART	QOH	QUANTITY	0	0
PART	QOO	QUANTITY	0	0
PART	PTYPE	PTYPE	0	0
PROJECT	JNO	JNO	1	0
PROJECT	JNAME	JNAME	0	0
PROJECT	JLOC	LOCATION	0	0
SHIP	SNO	SNO	1	SUPPLIER
SHIP	PNO	PNO	1	PART
SHIP	JNO	JNO	1	PROJECT
SHIP	DATE	DATE	1	0
SHIP	QSHIP	QUANTITY	0	0

C.2 Semantic Knowledge, Table GDS:

Domain	Order
SNO	0
PNO	0
JNO	0
SNAME	0
PNAME	0
JNAME	0
PTYPE	0
LOCATION	0
SRATING	1
QUANTITY	1
DATE	1

C.3 Semantic Knowledge, Table GRS:

Relation	Linking	Entity
SUPPLIER	0	1
PART	0	1
PROJECT	0	1
SHIP	1	1

C.4 Linguistic Knowledge, Table GRL:

Relation	English name
SUPPLIER	SUPPLIER
PART	PART
PROJECT	PROJECT
SHIP	SHIPMENT

C.5 Linguistic Knowledge, Table GRAL

Relation	Attribute	Attribute-Designator	Referential-Preposition
SUPPLIER	SNO	SERIAL-NUMBER	OF
SUPPLIER	SNAME	NAME	OF
SUPPLIER	SLOC	LOCATION	OF
SUPPLIER	RATE	RATING	OF
PART	PNO	SERIAL-NUMBER	OF
PART	PNAME	NAME	OF
PART	QOH	QUANTITY-ON-HAND	OF
PART	QOO	QUANTITY-ON-ORDER	OF
PART	PTYPE	TYPE	OF
PROJECT	JNO	SERIAL-NUMBER	OF
PROJECT	JNAME	NAME	OF
PROJECT	JLOC	LOCATION	OF
SHIP	SNO	SUPPLIER-SERIAL-NUMBER	IN
SHIP	PNO	PART-SERIAL-NUMBER	IN
SHIP	JNO	PROJECT-SERIAL-NUMBER	IN
SHIP	DATE	DATE	OF
SHIP	QSHIP	QUANTITY-SHIPPED	IN

C.6 Linguistic Knowledge, Table GPRINTCOMP:

=	OF-EXACTLY	EXACTLY	•
>	GREATER-THAN	MORE-THAN	BETTER-THAN
<	LESS-THAN	LESS-THAN	LESS-THAN
≥	OF-*-OR-MORE	*-OR-MORE	*-OR-BETTER
≤	OF-*-OR-LESS	*-OR-LESS	*-OR-LESS-THAN-*
≠	OTHER-THAN	A-QUANTITY-OTHER-THAN	•

C. 7 Linguistic Knowledge

Table GDATECOMP consists of the following three tables:

C.7.1 Table for Day

=	ON
>	AFTER
<	BEFORE
≥	ON-*-OR-AFTER
≤	ON-*-OR-BEFORE
≠	ON-A-DATE-OTHER-THAN

C.7.2 Table For Month

=	DURING
>	AFTER
<	BEFORE
≥	DURING-*-OR-AFTER
≤	DURING-*-OR-BEFORE
≠	DURING-A-MONTH-OTHER-THAN

C.7.3 Table For Year

=	DURING
>	AFTER
<	BEFORE
≥	DURING-*-OR-AFTER
≤	DURING-*-OR-BEFORE
≠	DURING-A-YEAR-CTHER-THAN

C.8 Key-Patterns, Table GKP:

Relation	Key-Designator
SUPPLIER	SUPPLIER-NUMBER
PART	PART-NUMBER
PROJECT	PROJECT-NUMBER

C.9 Pure Entity Patterns, Table GNP:

Relation	Pattern-Number	Attribute	English-Phrase	Pre-negation	GPRINTCOMP-Column
SUPPLIER	1	*	*	0	•
SUPPLIER	1	SNAME	NAMED	1	•
SUPPLIER	1	RATING	RATED	1	4
SUPPLIER	1	SLOC	LOCATED-IN	1	•
PROJECT	1	*	*	0	•
PROJECT	1	JNAME	NAMED	1	•
PROJECT	1	JLOC	LOCATED-IN	1	•
PART	1	*	*	0	•
PART	1	PNAME	NAMED	1	•
PART	1	PTYPE	OF-TYPE	1	•
PART	1	QOH	WITH-A-QUANTITY-ON-HAND	0	2
PART	1	QOO	WITH-A-QUANTITY-ON-ORDER	0	2

C.10 Linking Patterns For The SHIP Relation, Table GLP:

Pattern	Entry	Attribute	English-Phrase	Dependency	GPRINTCOMP-Column
1	PNO	*	SHIPPED	0	•
1	PNO	QSHIP	IN-A-QUANTITY	0	2
1	PNO	DATE	•	0	•
1	PNO	SNO	FROM	0	•
1	PNO	JNO	TO	0	•
2	SNO	*	WHO	0	•
2	SNO	DATE	,*,	0	•
2	SNO	*	SENT-A-SHIPMENT	0	•
2	SNO	QSHIP	CONTAINING	0	3
2	SNO	PNO	OF	QSHIP	•
2	SNO	JNO	TO	0	•
3	JNO	*	TO-WHICH-A-SHIPMENT	0	•
3	JNO	QSHIP	CONTAINING	0	3
3	JNO	PNO	OF	QSHIP	•
3	JNO	*	WAS-SENT	0	•
3	JNO	DATE	•	0	•
3	JNO	SNO	BY	0	•
4	•	*	*	0	•
4	•	QSHIP	CONTAINING	0	3
4	•	PNO	OF	QSHIP	•
4	•	DATE	SENT	0	•
4	•	SNO	FROM	DATE	•
4	•	SNO	BY	0	•
4	•	JNO	TO	0	•