A TEXT KNOWLEDGE BASE FOR THE AI HANDBOOK

Robert F. Simmons

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

AI-TR-84-05 December 1983
CS-TR-83-24

A Text Knowledge Base for the AI Handbook

Progress Report

Robt F. Simmons

The University of Texas, Austin

Abstract

This research (supported by NSF Grant IST 8200976) aims at defining a consistent set of text representation conventions for organizing fifty pages of the AI Handbook as an inferential knowledge base founded on a procedural logic system of general inference schemas for answering questions from it. After a year of research on the AI Handbook project, we have developed a prototype, natural-language, text knowledge system that includes a data base manager to compile the text knowledge and to make it available to navigational commands. The text is represented as logical propositions which form a set of text axioms to model its content. English questions and commands are translated to corresponding logical formulae and treated as theorems to be proved with respect to the text model. The logical form is that of semantic relations (SR) -- logical predicates with varying numbers and ordering of arguments. To compute effectively with such a free form, a relaxed unification procedure was defined as the basis of the SR theorem prover. The use of procedural logic augmented with fast, compiled Lisp functions has shown that questions can be answered in times ranging from a few tenths of a second to minutes of CPU time on a DEC2060 system. The navigational capabilities of the data base manager make available larger contexts surrounding the text and offer the user complete freedom to explore the text and to extract any desired information from it.

1. Text Knowledge System

The purpose of a text knowledge system (TKS pronounced Teks) is to provide an online, inferential, text database that in addition to performing customary data management functions, also provides natural language query and command capabilities and responds in natural language generated or extracted from the relevant text. The primary utility of a TKS is essentially the same as that of a scientific or engineering handbook; to make immediately available in English the accumulated knowledge from a field of study. Transformation of a handbook into a knowledge base offers the usual computational advantages of speed of access, instant reorganization of data according to a point of view, access from a distance, automatic production of teaching and lecture materials, reference finding and extraction, and potentials for use as a teaching and consulting system.

Because the TKS is a data management system that includes a theorem prover, natural language grammars, lexicon, and other semantic materials, the system can also be expected to be useful as a text laboratory for studying methods of parsing, semantic and discourse analysis, and generally for studying the syntax, contexts, content, style, and structure of text. Since its ELISP environment provides a range of from 8 to more than 16 million words of storage (depending on hardware address-length), it forms an ideal environment for analysis and application of such large text-bases as machine readable dictionaries and the various medical history and diagnostic collections so important in today's computational research projects.

The user of a TKS can work at several levels. First is the natural language stratum in which simple

English commands or queries are given and the system responds with selected or generated text. At the next level, the user may query the system in the formal language of semantic relations (SRs) and augment its inferential capabilities by adding taxonomic relations and rules of paraphrase. Linguistically oriented users may add grammar and lexical materials to expand the English subset -- after brief training in the forms of lexical and grammar rules. At this level the user is concerned with the text structure and the theorem prover. At the third level the user may be familiar with LISP and HCPRVR (Horn Clause Prover) and be able to experiment with improved procedures for semantic and discourse analysis, and alternate approaches to answering questions. Such a skilled programmer can also modify the system to provide capabilities for stylistic or content analysis, or may use it as a text laboratory to support experiments in applying text knowledge to teaching or consulting systems.

A schematic diagram for the present version of TKS is provided in Figure 1. This diagram is derived from the programs and because of its highly recursive, inter-related modules, it is to be read as a schematic rather than as a flowchart. When the user types a command, the evaluation system determines whether it is an English string or a call to some function. If an English string is given, the grammar and lexical system analyze it and translate it into a formal SR query. The dotted line connecting the grammar to the Netdb process displays the future possibility of automatically analyzing the text into database format. If the command is not English it must be a call to some function such as the following:

- *Axioms* -- given a list of one or more axioms such as grammar rules, lexical entries, paraphrase rules, or general HCPRVR (Horn Clause Prover) procedures, axioms for use by the theorem prover are formed.
- *Ask* -- Three questioning answering procedures: ASKB, ASKD, and ASKL, are provided to give respectively, breadth-first, depth-first, and procedural logic flows of control for answering an SR. (See section 5 for details.)
- *Netdb* -- given as input a text in the form of a rooted graph of semantic relations (i.e. an SR tree), Netdb compiles a text database with every SR assigned both as the value of a LISP atom and asserted as an axiom accessible by the head term of the SR. (See section 3.)
- *Navigation* -- The two LISP functions Options and Open provide navigational access to the text graph. Options takes an English word or phrase and returns the SRs that it accesses. Open expands a node of the graph by showing its immediate ancestor and its descendents. Both of these provide readable English outputs.
- *Lisp or HCPRVR* -- Full access to LISP and the HCPRVR contained in it is available as a normal use of the system.

When a query or command is received -- either from the grammar or from the input console -- the selected QA system accesses the database axioms using general rules of inference as well as particular rules for paraphrasing text or questions. As it progresses toward completing all subquestions it constructs partial answers which may eventually prove useful for cooperative response (Kaplan [1980]). If it succeeds in answering all subquestions it then constructs an English concatenation of text. With the command X, for expand, it presents an extracted text and its environment. At this point the user may further explore the text starting from that position to discover additional relevant material. Retrieved material is recorded so that new files may be constructed for any user purpose. A range of LISP functions (not shown) has been used to extract subsets of lexical, grammatical, and other data for various purposes.

The TKS system described in this progress report has so far been tested with more than twenty pages of text and about one hundred questions. Its database capabilities appear to be satisfactory within the limits studied; its English grammar and semantics are easily augmented to encompass increasingly large subsets of the language; and its question answering abilities, while weak by human standards, are sufficient to

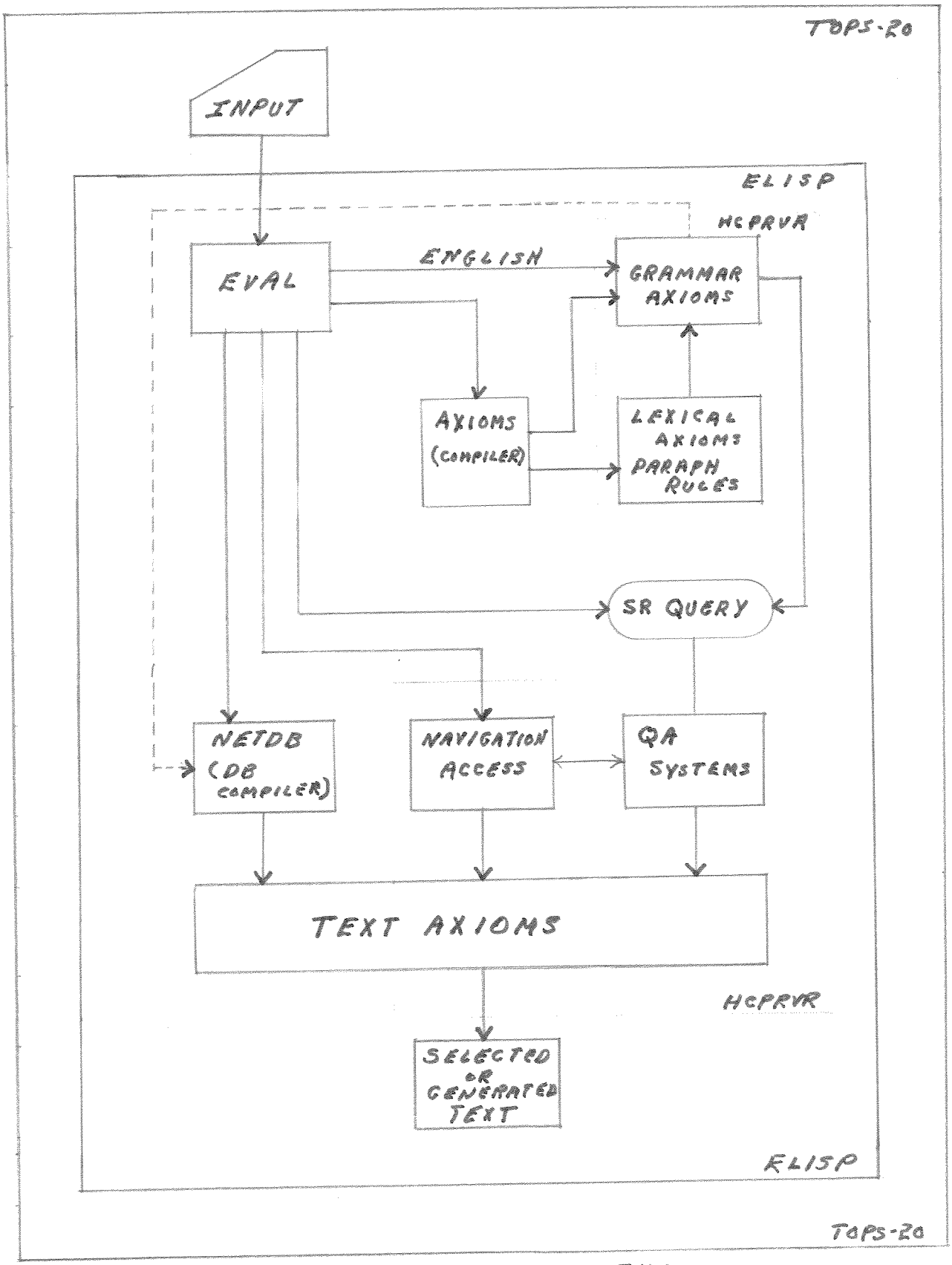

FIGURE 1 SCHEMATIC OF TKS

answer questions that correspond fairly closely to the text. By using paraphrase rules it is possible to program the system to answer any question for which the text provides an answer. Section 5, however, shows that such a strategy is not fully satisfactory.

2. Representation of Text

The text proposed for this study was fifty pages of *The Handbook of Artificial Intelligence* by A. Baar and E. Feigenbaum. The authors were kind enough to make available a machine-readable version of volume 1 from which we extracted the sections A and B from Chapter II and A, B, and part of C from Chapter III, all totaling 55 pages. Our first study concerned five pages of sections B1 and B2 which we hand-analyzed into a network of semantic relations (SRs). A sample of the resulting structure is shown in Appendix 2.1.

This representation discards much of the common meta-expressions of English. For example, consider the sentence "A state-space representation of a problem employs two kinds of entities: states, which are data structures giving snapshots of the problem at various stages of its solution, and operators, which are means for transforming the problem from one state to another." In this example, the terms "two kinds of entities", the two uses of "which are", and the indirect expression "means for" disappear in the resulting representation shown below:

```
(EMPLOY TNS PRES
  INSTR (REPRESENTATION *OF STATE-SPACE *OF PROBLEM NBR SING DET A)
        AE (STATE NBR PL
              EQUIV (STRUCTURE *OF DATA NBR PL
                        INSTR* (GIVE AE (SNAPSHOT NBR PL)
                                  *OF (PROBLEM DET THE NBR SING)
                                  *AT (STAGE MOD VARIOUS NBR PL
                                         *OF (SOLUTION *OF IT NBR SING))))

             *AND (OPERATOR NBR PL
                     INSTR* (TRANSFORM AE (PROBLEM DET THE NBR SING)
                                  *FROM (STATE MOD ONE NBR SING)
                                  *TO ANOTHER))))
```

The simplification is equivalent to: "The state-space representation of a problem employs states -- data structures giving snapshots of the problem at various stages of its solution, and operators transforming the problem from one state to another."

In a later experiment with the same text we used surface semantic relations, SSRs which have the property that every signal given in the text is retained in the formalism. In the SSR representation all the meta-language was retained. *Answering questions was more easily and accurately accomplished with the deeper structure of SRs than with the SSRs.* But the same questions could be answered in either structure provided the shallower one was supported by the use of paraphrase rules (which accomplished the same effect of deleting the meta-terms) and the deeper by paraphrase rules which deleted meta-terms from the question. Since the questions can generally be expected to be in more informal English than the text, the study supported the notion that the deeper structure was the more economical one to use.

We also studied representation in SRs using triples in contrast to the "long" form shown above. In the triple representation the example sentence appears as follows:

```
(EMPLOY1 INSTR REPRESENTATION1) (EMPLOY1 AE STATE1)
(REPRESENTATION1 *OF STATE-SPACE1)
(REPRESENTATION1 *OF PROBLEM1)   (PROBLEM1 NBR SING)
(PROBLEM1 DET A)                 (STATE1 NBR PL )
(STATE1 EQUIV STRUCTURE1)        (STRUCTURE1 *OF DATA1)
(DATA1 NBR PL)                   (DATA1 INSTR* GIVE1)
(GIVE1 AE SNAPSHOT1)             (SNAPSHOT1 NBR PL)
```

```
(SNAPSHOT1 *OF PROBLEM1)
(PROBLEM1 NBR SING)
(STAGE1 MOD VARIOUS)
(STAGE1 *OF SOLUTION1)
(SOLUTION1 NBR SING)
(OPERATOR1 NBR PL)
(TRANSFORM1 AE PROBLEM1)
(PROBLEM1 NBR SING)
(STATE2 MOD ONE)
(TRANSFORM1 *TO ANOTHER)
(PROBLEM1 DET THE)
(GIVE1 *AT STAGE1)
(STAGE1 NBR PL)
(SOLUTION1 *OF IT)
(STATE1 *AND OPERATOR1)
(OPERATOR1 INSTR* TRANSFORM1)
(PROBLEM1 DET THE)
(TRANSFORM1 *FROM STATE2)
(STATE2 NBR SING)
```

The comparison of question-answering times for the two representations did not reveal a clear advantage of one over the other. The long form of SRs required considerably less storage than the triples, but the logic of the QA procedure was much more transparent when defined over the triples (see Section 5). A more decisive study remains to be accomplished.

It can be noted from the examples above that the sentence analysis does not resolve anaphoric references. In the "deep" representation these references were resolved by hand resulting in a highly interconnected network. In the shallow representation they were not resolved at all, but the SSRs were embedded in a higher level text structure to result in connections via the text tree (i.e. rooted text graph).

The text tree is a labeled outline structure imposed on the text. Its form is that of an SR with nodes and labeled arcs whose values are other SRs. The topmost SR is "AIH", which dominates (abbreviated) chapter titles with the arc TOPIC, provides the full title as the value of the TXT arc, and may provide an SR representation for that text as the value of an SR arc. Each dominated node is treated in a similar fashion. Figure 2 displays a portion of a text tree for Chapter B.

```
================================================================
(AIH
 TXT
 (AI HANDBOOK)
 SR
 (HANDBOOK DET THE *OF (INTELLIGENCE TYPE ARTIFICIAL))
 TOPIC
 (REPRESENTATION
  TXT
  (B PROBLEM REPRESENTATION)
  SR
  (REPRESENTATION *OF PROBLEM CHAPT B)
  SUBTOPIC
  (REPRESENTATION
   TXT
   (B1/. STATE-SPACE REPRESENTATION)
   SR
   (REPRESENTATION *OF STATE-SPACE)
   DESCR
   (USE TXT
        (A STATE-SPACE REPRESENTATION OF A PROBLEM EMPLOYS TWO KINDS OF
           ENTITIES:)
        SR
        (EMPLOY TNS
                PRES
                INSTR
                (REPRESENTATION *OF STATE-SPACE)
                AE
                (KIND NBR PL QTY TWO *OF (ENTITY NBR PL PREP OF)))
        ELEMENT
        (STATE TXT
               (STATES,)
               SR
```

```
            (STATE NBR PL)
            DESCR
            (STRUCTURE TXT
               (WHICH ARE DATA STRUCTURES GIVING /"SNAPSHOTS/" OF THE
                      CONDITION OF THE PROBLEM AT EACH STAGE OF ITS
                      SOLUTION,)
                SR
                (BE TNS
                    PRES
                    AE
                    (STATE NBR PL)
                    EQUIV
                    (STRUCTURE TYPE
                               DATA
                               INSTR*
                               (GIVE TNS
                                     PRPRT
                                     AE
                                     (SNAPSHOT NBR
                                               PL
                                               *OF
                                       (CONDITION DET
                                                  THE
                                                  *OF
                                                  (PROBLEM DET
                                                           THE
                                                           )))
                                     RANGE
                                     (STAGE NBR
                                            SING
                                            PREP
                                            AT
                                            QFY
                                            EACH
                                            *OF
                                            (SOLUTION *OF ITS)))))))
      ELEMENT
      (OPERATOR TXT
                (AND OPERATORS,)
                SR
                (OPERATOR NBR PL)
                DESCR
                (MEANS TXT
               (WHICH ARE MEANS FOR TRANSFORMING THE PROBLEM FROM ONE
                          STATE TO ANOTHER)
                       SR
                       (BE TNS
                           PRES
                           AE
                           (OPERATOR NBR PL)
                           EQUIV
                           (MEANS INSTR*
                                  (TRANSFORM TNS
                                             PRPRT
                                             AE
                                             (PROBLEM DET THE)
                                             *FROM
                                             (STATE QTY ONE NBR SING)
                                             *TO
                                             (ANOTHER)))))))
EXAMPLE
(PUZZLE TXT
(A STRAIGHTFORWARD EXAMPLE OF STATE-SPACE REPRESENTATION IS THE
```

```
   SIMPLE, WELL-KNOWN PUZZLE)
SR
(BE AE
    (EXAMPLE MOD
             STRAIGHTFORWARD
             DET
             A
             NBR
             SING
             *OF
             (REPRESENTATION *OF STATE-SPACE))
    EQUIV
    (PUZZLE MOD SIMPLE TYPE WELL-KNOWN DET THE NBR SING))
NAME
(EIGHT-PUZZLE TXT
              (CALLED THE 8. -PUZZLE/.)
      SR
      (CALL TNS PAST AE (EIGHT-PUZZLE DET THE NBR SING))
              DESCR
              (TRAY TXT
                    (AN 8. -PUZZLE IS A SQUARE TRAY)
                    SR
                    (BE TNS
                        PRES
                        AE
                        (EIGHT-PUZZLE NBR SING DET AN)
                        SUP
                        (TRAY MOD SQUARE DET A NBR SING))
                    ELEMENT
                    (TILE TXT
                   (CONTAINING EIGHT SQUARE TILES OF EQUAL
                               SIZE,)
                          SR
                          (CONTAIN INSTR
                                   (TRAY)
                                   AE
                                   (TILE MOD
                                         SQUARE
                                         QTY
                                         EIGHT
                                         NBR
                                         PL
                                         *OF
                                         (SIZE MOD EQUAL)))
                          SPEC
                          (NUMBER TXT
                                  (NUMBERED 1. TO 8.)
                                  SR
                                  (NUMBER TNS
                                          PAST
                                          AE
                                          (TILE)
                                          *FROM
                                          (ONE)
                                          *TO
                                          (EIGHT))))
                    ELEMENT
                    (SPACE TXT
                           (THE SPACE
                                FOR
                                THE
                                NINTH
                                TILE
```

```
        IS
        VACANT
        (SEE FIG/. B1-1))
SR
(BE AE
    (SPACE *FOR
    (TILE MOD NINTH NBR SING)
           DET
           THE
           NBR
           SING)
    ST
    (VACANT))))]
```

Figure 2. An Extract from a Text Tree

Arcs used in the text tree include abbreviations for the classes: description, example, definition, element, representation, procedure, model, reason, exception, and goal. These are an empirical expansion of common discourse categories given in rhetoric teaching books. Each of the classes is designed to answer a simple question such as

- What is a description of _____?
- What is an example of _____?
- What is a reason associated with _____?

In the navigational approach to exploring the text, the classes show the user what questions a given segment can answer.

We expect that when the fifty pages of text have been analyzed an inventory of the classes and their usage will allow us to form an improved classification system with fairly definite rules for the usage of the labels in analyzing additional text of this type. The text tree has the most desirable property of organizing clauses, sentences, or larger units of text into a hierarchic structure that forms a summarizing extract at any level of abstraction by deleting nodes below that level and presenting the value of the remaining text arcs. Study of the hand-constructed text trees is expected to show the discourse patterns used in this class of expository writing. We believe that such patterns can form the base for constructing expository schemas that will be helpful in computing text trees resolving referential terms in the process. Moderately successful experiences using this technique for narrative texts are described in Simmons [1983].

3. Text Database Compiling

The database compiler, called Netdb, accepts a text tree as input. Each list contained in the text tree is an SR where the first element is the head term and the remaining elements form a list of pairs. The SR structure is true of the tree and of all subtrees included. Netdb first constructs a Lisp atom by concatenating the head term to a unique number and then sets that atom's value to the list. For example the phrase, "a square tray containing eight tiles" is represented as the following SR:

```
(CONTAIN TNS PRPRT INSTR (TRAY MOD SQUARE DET A NBR SING)
         AE (TILE QTY EIGHT NBR PL))
```

The head, CONTAIN is concatenated with a generated number, say 1035, to form CONTAIN^1035, whose value is set to the full SR. Similarly, TRAY^1036 and TILE^1037 are formed to evaluate to the two embedded SRs.

Each SR is then modified to include a backlinking arc, BK*, to the SR in which it is embedded. The root of the text tree is the only SR without a backlink. Supposing the example phrase above was part of the SR,

```
(DEFINE AUX (BE TNS PRES) TNS PAST AE (EIGHT-PUZZLE DET A NBR SING)
        REF (CONTAIN TNS PRPRT INSTR (TRAY...) AE (TILE...)))
```

then the example phrase would be modified with backlinks as follows:

```
(CONTAIN BK* DEFINE^1034 ... INSTR (TRAY BK* CONTAIN^1035 ...)
                               AE (TILE BK* CONTAIN^1035 ...))
```

At the completion of these operations each SR is further rewritten to substitute the atom names such as TRAY^1036 and TILE^1037 for the embedded SRs, to include its own node-name as the value of the arc NODE, and to assert the SR as a procedural logic axiom using the procedure ASSERT. The result of the assertion is to add, for example, the CONTAIN SR as a value of the property AXIOMS to the atom CONTAIN. Similarly any nested SRs such as (TRAY...) and (TILE...) are asserted as axioms of their head terms.

The result is a text database accessible by any headword of an SR or from any of the list of specialized atoms such as CONTAIN^1035, TRAY^1036, etc. The following section on navigating the database will make its capabilities clear.

4. Database Navigation

The navigation system uses three procedures, Eval, Option, and Open. Option of an English word or phrase employs a procedural logic function to ask it as a question and present the first matching axiom followed by the query to the user "Another?" By answering Y or Yes the user can successively view all the axioms associated with a word or all the matching axioms associated with a phrase.

When such an axiom is presented, the user may use the Lisp function EVAL to examine the SR associated with any token such as CONTAIN^1035. This is accomplished merely by typing the literal token. If light-pen or mouse control is available, simply "touching" the token will be sufficient to display its SR content.

If additional context is desired, the procedure Open with a token as argument, e.g. (OPEN CONTAIN^1035), will print the immediate ancestor SR of the argument and the SRs of the argument's descendents. For example, the ancester of CONTAIN^1035 provides the SR of DEFINE^1034 which includes the arc REF CONTAIN^1035 and its own node-name DEFINE^1034 as well. The descendents of CONTAIN^1035 include the arc-value pairs, INSTR TRAY^1036 and TILE^1037, but what is presented is their expansion into the original SRs. Figure 3 shows a navigational access starting with Options for the word "puzzle" followed by a series of Open calls to explore the tree in various directions.

```
================================================================
[PHOTO:  Recording initiated  Fri 16-Sep-83 11:22AM]

*(YAK) "Yak puts the system in theorem proving mode and
        signals this mode with the "?" prompt."
?(OPTIONS PUZZLE)

REPRESENTATION^1006
(PUZZLE TXT (A STRAIGHTFORWARD EXAMPLE OF STATE-SPACE
 REPRESENTATION IS THE SIMPLE, WELL-KNOWN PUZZLE)
SR BE^1036 NAME EIGHT-PUZZLE^1040 ELEMENT STATE^1079
ELEMENT OPERATOR^1089 NODE PUZZLE^1035)
((OPTIONS PUZZLE))
```

```
ANOTHER? ?(OPEN EIGHT-PUZZLE^1040)
(EIGHT-PUZZLE BK*
              (PUZZLE TXT
                 (A STRAIGHTFORWARD EXAMPLE OF STATE-SPACE
                REPRESENTATION IS THE SIMPLE, WELL-KNOWN PUZZLE)
                      SR
                      BE^1036
                      NAME
                      EIGHT-PUZZLE^1040
                      ELEMENT
                      STATE^1079
                      ELEMENT
                      OPERATOR^1089
                      NODE
                      PUZZLE^1035)
              TXT
              (CALLED THE 8 -PUZZLE/.)
              SR
              (CALL TNS PAST AE EIGHT-PUZZLE^1042 NODE CALL^1041)
              DESCR
              (TRAY TXT
                    (AN 8 -PUZZLE IS A SQUARE TRAY)
                    SR
                    BE^1044
                    ELEMENT
                    TILE^1047
                    ELEMENT
                    SPACE^1057
                    NODE
                    TRAY^1043)
              REPR
              (FIGURE TXT
                      ((|----|----|----|)
                       (| 2  | 1  | 6  |)
                       (|----|----|----|)
                       (| 4  |    |  8 |)
                       (|----|----|----|)
                       (| 7  | 5  |  3 |)
                       (|----|----|----|)
                       (FIGURE B1-1/. AN 8 -PUZZLE/.))
                      NODE
                      FIGURE^1062)
              PROCEDURE
              (MOVE TXT
                    (A TILE MAY BE MOVED BY SLIDING IT VERTICALLY OR
                       HORIZONTALLY INTO THE EMPTY SQUARE/.)
                    SR
                    MOVE^1064
                    NODE
                    MOVE^1063)
              GOAL
              (PROBLEM TXT
                       (THE PROBLEM IS TO TRANSFORM SOME PARTICULAR TILE
                            CONFIGURATION, SAY, THAT OF FIGURE B1-1, INTO
                            ANOTHER GIVEN TILE CONFIGURATION, SAY, THAT OF
                            FIGURE B1-2/.)
                       SR
                       BE^1072
                       SOLUTION
                       FIGURE^1078
                       NODE
                       PROBLEM^1071))
NIL
```

```
ANOTHER? ?Y

BE^1036
(PUZZLE MOD SIMPLE TYPE WELL-KNOWN DET THE NBR SING NODE PUZZLE^1039)
((OPTIONS PUZZLE))

ANOTHER? ?Y

REPRESENTATION^1228
(PUZZLE NAME TOWER-OF-HANOI^1321 NODE PUZZLE^1320)
((OPTIONS PUZZLE))

ANOTHER? ?(OPEN TOWER-OF-HANOI^1321)
(TOWER-OF-HANOI
 BK*
 (PUZZLE NAME TOWER-OF-HANOI^1321 NODE PUZZLE^1320)
 TXT
 (AN EXAMPLE THAT LENDS ITSELF NICELY TO PROBLEM-REDUCTION
   REPRESENTATION IS THE FAMOUS TOWER OF HANOI PUZZLE/.)
 SR
 (BE AE EXAMPLE^1323 NODE BE^1322)
 ELEMENT
 (DISK TXT
       (IN ONE COMMON VERSION THERE ARE THREE DISKS, A, B, AND
        C, OF GRADUATED SIZES/.)
       SR
       BE^1328
       NODE
       DISK^1327)
 ELEMENT
 (PEG TXT
      (THERE ARE ALSO THREE PEGS, 1 /, 2 /, AND 3)
      SR
      BE^1334
      NODE
      PEG^1333)
 STATE
 (INITIAL-STATE
  TXT
  (INITIALLY THE DISKS ARE STACKED ON PEG 1 /, WITH A,
        THE SMALLEST, ON TOP AND C, THE LARGEST, AT THE BOTTOM/.)
  SR
  STACK^1338
  NODE
  INITIAL-STATE^1337)
 PROBLEM
 (TRANSFER TXT
           (THE PROBLEM IS TO TRANSFER THE STACK TO PEG 3 /,
                AS IN FIGURE B2-1, GIVEN THAT (A) ONLY ONE
                DISK CAN BE MOVED AT A TIME AND (B) NO DISK
                MAY BE PLACED ON TOP OF A SMALLER DISK/.)
           SR
           BE^1348
           FIGURE
           STATE^1363
           NODE
           TRANSFER^1347)
 PROCED
 (SOLUTION TXT
  (ONLY ONE OPERATOR NEED BE USED IN THE SOLUTION: GIVEN DISTINCT PEGS
   I, J, AND K, THE PROBLEM OF MOVING A STACK OF SIZE N > 1 FROM
   PEG I TO PEG K CAN BE REPLACED BY THE THREE PROBLEMS: 1 MOVING
   A STACK OF SIZE N - 1 FROM I TO J, 2 MOVING A STACK OF SIZE 1
```

```
   FROM I TO K, 3 MOVING A STACK OF SIZE N - 1 FROM J TO K/.)
           ELEMENT
           PRIMITIVE-PROBLEM^1365
           NODE
           SOLUTION^1364)
 REPR
 (PROBLEM-DESCRIPTION
  TXT
  (EACH PROBLEM DESCRIPTION CAN NOW BE GIVEN BY SPECIFYING
   THE SIZE N OF THE STACK TO BE MOVED, THE NUMBER OF THE
   SENDING PEG, AND THE NUMBER OF THE RECEIVING PEG/.)
  ELEMENT
  INITIAL-PROBLEM^1367
  ELEMENT
  TRANSFORMATION^1368
  NODE
  PROBLEM-DESCRIPTION^1366))
NIL
ANOTHER? ?Y

EXAMPLE^1323
(PUZZLE *OF TOWER-OF-HANOI MOD FAMOUS DET THE NBR SING
  NODE PUZZLE^1326)
((OPTIONS PUZZLE))

ANOTHER? ?Y

REPRESENTATION^1006
(TXT (A STRAIGHTFORWARD EXAMPLE OF STATE-SPACE REPRESENTATION
      IS THE SIMPLE, WELL-KNOWN PUZZLE)
 SR BE^1036 NAME EIGHT-PUZZLE^1040 ELEMENT STATE^1079
 ELEMENT OPERATOR^1089 NODE PUZZLE^1035)
((OPTIONS PUZZLE))

ANOTHER? ?Y

BE^1036
(MOD SIMPLE TYPE WELL-KNOWN DET THE NBR SING NODE PUZZLE^1039)
((OPTIONS PUZZLE))

ANOTHER? ?N

2@POP

[PHOTO:  Recording terminated  Fri 16-Sep-83 11:25AM]
```

Figure 3. Navigating Through the Database

At the present stage of research the displays have not yet been engineered to maximize human convenience and esthetic properties, but they do present the data in a complete enough fashion so that the entire text base is accessible by these commands. The display engineering phase is postponed for several months until the delivery of new Lisp machine hardware that provides bit-mapped windowed displays in which attractive presentations can be programmed.

Several functions have been programmed in Lisp to save and file any portions of the text base that are examined and to extract any data that has been axiomatized. Further experience is expected to show us which of these functions should be provided as standard database operations.

5. Query Systems

During the first year of the research project, eight procedures were developed for questioning the representation. The last three of these were breadth-first Lisp, depth-first Lisp, and depth-first procedural logic versions which are currently integrated into the system. Each QA system accepts an SR representation for a query or command, uses taxonomic inheritance in seeking an answer, and presents its result in terms of SRs, expanded SRs (using Open), and extracted English text. Each is an SR theorem prover that uses a form of the unification algorithm called *relaxed unification* to prove or disprove a question-SR, taken as a theorem with respect to the text axioms. Each uses taxonomic and other inference relations given in the English lexicon to relate questions that are paraphrases of the text axioms. Each presents a first answer if found, and upon successive requests every answer it can derive.

The theory of question-answering embodied in these systems takes an SR query as a theorem to be proved from the text axioms and supporting lexical information. Central to a theorem proving procedure, the unification algorithm provides a fast, effective technique for binding variables and matching query clauses with data axioms (including inference rules). But the unification algorithm assumes that atomic clauses are n-tuples of the form,

```
<Predicate-name Arg1, ... Argn>
```

a predicate name followed by a fixed length list of arguments. SRs in contrast are of the form,

```
<Predicate-name arc-val1, ... arc-valn>
```

where the number and order of arc-vals may vary in accordance with their expression or absence in the natural language sentence. Thus the unification algorithm needed to be generalized for this application.

Relaxed Unification

In the unification algorithm, two n-tuples, n1 and n2, unify if Arity(n1) = Arity(n2) and if every element in n1 matches an element in n2. Two elements e1 and e2 match if e1 or e2 is a variable, or if e1 = e2, or in the case that e1 and e2 are lists of the same length, each of the elements of e1 matches a corresponding element of e2.

Since semantic relations (SRs) are unordered lists of binary relations that vary in length and since a question representation (SRq) can be answered by a sentence candidate (SRc) that includes more information than the question specified, the Arity is revised to Arity(SRq) Less/Equal Arity(SRc).

The primitive elements of SRs include words, arcnames, variables and constants. Arcnames and words are organized taxonomically, and words are further organized by the discourse structures in which they occur. One or more element of taxonomic or discourse structure may imply others. *Words in general can be viewed as restricted variables whose values can be any other word on an acceptable inference path that joins them.* The matching constraints of unification can thus be relaxed by allowing two terms to match if one implies the other in a taxonomic closure.

The matching procedure is further adapted to read SRs effectively as unordered lists of triples and to seek for each triple in SRq a corresponding one in SRc. The two SRs below match because Head matches Head, Arc1 matches Arc1, Val1 matches Val1, etc. even though they are not given in the same order.

SRq (Head Arc1 Val1, Arc2 Val2, ..., Arcn Valn)
SRc (Head Arc2 Val2, Arc1 Val1, ..., Arcn Valn)

The SR may be represented (actually or virtually) as a list of triples as follows:

SRq ((Head Arc1 Val1)(Head Arc2 Val2) ..., (Head Arcn Valn))

Two triples match in Relaxed Unification according (at least) to the conditions shown in Figure 4. The query triple, A R B may match the candidate giving + + + to signify that all three elements unified. If the first two elements match, the third may be matched using the procedures CLOSAB or CLOSCP to relate the non-matching C with the question term B by discovering that B is either in the abstractive closure or the complex product closure of C. The *abstractive closure* of an element is the set of all triples that can be reached by following separately the SUP and EQUIV arcs and the INST and EQUIV* arcs. The *complex product closure* is the set of triples that can be reached by following a set of generally transitive arcs (not including the abstractive ones). The arc of the question may have a synonym or a converse and so develop alternative questions, and additional questions may be derived by asking such terms as C R B that include the question term A in their abstractive closure. Both closure procedures should be limited to n-step paths where n is a value between 3 and 6.

Computational Cost

In the above recursive definition the cost is not immediately obvious. If it is mapped onto a graphic representation in semantic network form, it is possible to see some of its implications. Essentially the procedure first seeks a direct match between a question term and a candidate answer; if the match fails, the abstractive closure arcs, SUP, INST, EQUIV, and EQUIV* may lead to a new candidate that does match. If these fail, then complex product arcs, *OF, HAS, LOC, AND, and OR may lead to a matching value. The graph below outlines the essence of the procedure.

```
A---R---B---SUP---Q
        |---INST---Q
        |---EQUIV---Q
        |---EQUIV*---Q
        |---AND---Q
        |---OR----Q
        |---LOC---Q
        |---*OF---Q
        |---HAS---Q
```

This graph shows nine possible complex product paths to follow in seeking a match between B and Q. If we allow each path to extend N steps such that each step has the same number of possible paths, then the worst case computation, assuming each candidate SR has all the arcs, is of the order, 9 raised to the Nth. If the A term of the question also has these possibilities, and the R term has a synonym, then there appear to be 2*2*9**Nth possible candidates for answers. The first factor of 2 reflects the converse relation by assigning the A term 9**N paths. Assuming only one synonym, each of two R terms might lead to a B via any of 9 paths, giving the second factor of 2. If the query arc is also transitive, then the power factor 9 is increased by one.

In fact, SRs representing ordinary text appear to have less than an average of 3 possible CP paths and few arcs have synonyms, so something like 2*3**Nth seems to be the average cost. So if N is limited to 3 there are about 2*81=162 candidates to be examined for each subquestion. These are merely rough estimates, but if the question is composed of 5 subquestions, we might expect to examine something on the order of a thousand candidates in a complete search for the answer. Fortunately, this is accomplished in less than seconds of computation time.

The length of transitive path is also of importance for two other reasons. First, most of the CP arcs

```
============================================================
Query Triple:  A R  B
Match Candid.  + +  +  + means a match by unification.
               + +  C  (CLOSAB C B)
               + +  C  (CLOSCP R C B)
               + R1 +  (SYNONYM R R1)
               B R1 A  (CONVERSE R R1)
               C +  +  (CLOSAB C A)

where CLOSAB stands for Abstractive Closure and is defined in
procedural logic (where the symbol < is shorthand for the reversed
implication sign <--, i.e. P < Q S is equivalent to Q ^ S --> P):

        (CLOSAB N1 N2) < (OR (INST N1 N2)(SUP N1 N2))
        (INST N1 N2) < (OR (N1 INST N2)(N1 EQUIV* N2))
        (INST N1 N2) < (INST N1 X)(INST X N2)
        (SUP N1 N2) < (OR (N1 EQUIV N2)(N1 SUP N2))
        (SUP N1 N2) < (SUP N1 X)(SUP X N2)

CLOSCP stands for Complex Product Closure and is defined as

        (CLOSCP R N1 N2) < (TRANSITIVE R)(N1 R N2)
                           "N1 R N2 is the new A R B"
        (CLOSCP R N1 N2) < (N1 *OF N2)**
        (CLOSCP R N1 N2) < (N1 LOC N2)**
        (CLOSCP R N1 N2) < (N1 *AND N2)
        (CLOSCP R N1 N2) < (N1 *OR N2)
--------
** These two relations turn out not to be universally true complex
products; they only give answers that are possibly true, so they
have been dropped for most question answering applications.
--------
============================================================
```

Figure 4.

Conditions for Matching Question and Candidate Triples

lead only to probable inference. Even superset and instance are really only highly probable indicators of equivalence, while LOC, HAS, and *OF are even less certain. Thus if the probability of truth of match is less than one for each step, the number of steps that can reasonably be taken must be sharply limited. Second, it is the case empirically that the great majority of answers to questions are found with short paths of inference. In one (early) all-answers version of the QA-system, we found a puzzling phenomenon in that all of the answers were typically found in the first fifteen seconds of computation although the exploration continued for up to 50 seconds. Our current hypothesis is that *the likelihood of discovering an answer falls off rapidly as the length of the inference path increases.*

Explicit Rules of Inference

This generalization of the unification algorithm is designed to allow rules of inference to be stated in terms of SRs rather than as fixed n-tuples and to minimize the costs of searching for matching arc-value pairs. So in addition to the rules built into the algorithms there exist also rules for drawing conclusions by combining text axioms. For example, we know that a commander of troops is responsible for the outcome of their battles. So if we know that Cornwallis commanded an army and the army lost a battle, then we can conclude correctly that Cornwallis lost the battle. An SR inference rule to this effect is shown below:

```
((LOSE AGT X AE Y) < (SUP X COMMANDER)(SUP Y BATTLE)
     (COMMAND AGT X AE W) (SUP W MILITARY-GROUP)
     (LOSE AGT W AE Y))

Text axioms:
     (COMMAND AGT CORNWALLIS (ARMY MOD BRITISH))
     (LOSE AGT (ARMY MOD BRITISH) AE (BATTLE *OF YORKTOWN-HEIGHTS))
Theorem:
     (LOSE AGT CORNWALLIS AE (BATTLE *OF YORKTOWN-HEIGHTS))
```

Inference procedures such as these are obviously necessary to combine related facts given in the text, but they are also useful to show how one or more text axioms can be paraphrased in a question or assertion. For example, consider the question, "Is problem-reduction representation distinguished from state-space representation?" A candidate answer is "Often distinguished from state-space representation is a technique called problem-reduction representation." We can use a paraphrase rule that states that "a technique called X --> X" to enable the match and accept the candidate as an answer.

The use of explicit inference rules of this type is a slow computational process in comparison to the use of those built into the relaxed unification procedure. The time for answering a question using explicit SR inference rules is measured in seconds and minutes; that for relaxed unification in tenths of seconds. Using explicit inference rules, the power of the QA system can theoretically be increased to any desired degree -- that is, for any question for which a human judges that a portion of the text provides an answer, an inference rule can be prepared that rewrites the question in those terms and so finds that answer. But the greater the number of inference rules used, the slower the system runs. A satisfactory solution to this dilemma requires that the inference rules must somehow be applied at the time of representing the text and queries so that only relaxed unification need be applied at the time of answering a question. For a range of English paraphrase rules this procedure can easily be built into the grammar that translates English to text axioms; it is not clear that such a solution is feasible for making explicit the many implications that a series of text axioms entail. *A thorough study of the tradeoff relations between representation conventions and inference rules remains a high priority for continued research on text knowledge systems.*

6. English Interface for TKS

The Text Knowledge System accepts English questions and commands, translates them to SRs, and when an answer is found, presents the English statement from which the answering SRs were derived. When the the answer is presented, control is returned to the user with the message, "Type ? for help" The help message explains the commands: N, for next answer, P to show the SR forms of the answer, X to expand the answer to show its context, T to find the time cost, and A to terminate the search for additional answers. At any time after the answer is printed, the user may also use the navigational facilities or even ask a new question. With no great pride in the present capabilities of the system, a protocol of a question-answering session is included with comments as Appendix 1. This protocol shows the capability of the system to answer quite simple questions and to expand the answers using the navigational facilities.

Given an English sentence or question in the form of a string terminated by a period or question mark, the string is passed to a procedural logic grammar for parsing and translation to text SR forms. The technique of using such a grammar was described in Simmons and Chester [1982] and Simmons [1983]. The grammar used is a revised form of the one published in those references that has been rewritten to increase efficiency and to reduce the possibility of redundant translations. It has been tested with approximately one hundred questions and is still expanding to account for additional questions in our sample. Rules for lexical knowledge acquisition have been included so that additions to the dictionary and semantic event forms are more easily accomplished.

When a string is presented to this grammar, a lexical search procedure is first called to determine if all words in the string are known to the system. For unknown words, the system requests a lexical assertion in the standard form, <Wordform Wordclass Canonical-form Semantic-class>; the system then tries again and continues scanning the string until all words are found in the lexicon. The semantic testing is then disabled and the system attempts to translate the string into an SR. Disabling the semantic testing phase has the effect of allowing any constituent that is well-formed syntactically to be translated using the relevant semantic event form (SEF) if it exists, or printing an incomplete one with variables as an informative error message to the user that it needs that SEF with constants in place of the variables. Such messages along with the resulting (relaxed) translation provide the linguist constructing the grammar with sufficient information to add the needed semantic information, then to try the parse again until it succeeds with no semantic error messages.

When the string has been translated to a surface semantic relation (SSR) a further transformational procedure is applied to reduce the SSR to a more basic SR form that discards much of the surface information and may delete meta-language terms such as "a kind of", "a technique called", etc. At this stage it is also possible to apply paraphrase transformations to translate the SSR either to a canonical form (if such can be defined) or into several variations to increase the probability of matching questions.

For automatic text analysis this is the stage where discourse structuring can be accomplished to organize the clauses of the text into an appropriate hierarchic structure and to resolve anaphoric references. Some techniques for these computations are known, but so far no system has been developed for accomplishing them on large samples of text. Our question answering research in this project will result in a hand-produced discourse structure for the fifty pages of text that have been studied. The question grammar largely accounts for the syntactic and semantic usages of the vocabulary of this text, so we are in an excellent position to write rules for combining SRs into larger discourse structures of the type we analyzed by hand. Research that we have earlier accomplished (Alterman [1982], Simmons [1983]) has shown some effective methods for structuring narrative texts, establishing references for pronouns, definite noun phrases, ellipses, and other anaphora in the process. *It will be a high priority for continued research to*

further develop and apply these methods to the handbook text.

In this handbook research we have not so far generated English answers; instead we believe that the author's text must be respected and that answers to questions should be extracted from the text rather than generated. It is our intention, however, to provide the generation capabilities described previously (Simmons [1983]) to give short answers and to construct summaries. An important finding in the work cited is that the symmetry of procedural logic results in a set of grammar rules that are symmetric with respect to analysis and generation; as a consequence, given an SR and the grammar that produced it, the same grammar constructs an English string to represent the SR. Generating short answers to questions and defining summarizing procedures is thus an easily programmed task in this system.

7. Continuing Development

After a year of research on the AI Handbook project, we have developed a natural-language, text knowledge system that includes a data base manager to compile the text knowledge and to make it available to navigational commands. The text is represented as logical propositions which form a set of text axioms to model the content of the text. English questions and commands are translated to corresponding logical formulae and treated as theorems to be proved with respect to the text model. The logical form is that of semantic relations -- logical predicates with varying numbers and ordering of arguments. To compute effectively with such a free form, a relaxed unification procedure was defined as the basis of the SR theorem prover. The use of procedural logic augmented with fast, compiled Lisp functions has shown that questions can be answered in times ranging from a few tenths of a second to minutes of CPU time on a DEC2060 system.

Tests of up to twenty pages of text predict that the response time will not degrade seriously for fifty. The use of ELISP with its several million words of addressable memory has shown that it is not necessary to use any specially designed data management system for handling book-length texts. By transferring the system to a LISP machine such as the Symbolics 3600, it will be possible to deal with more than 16 million words of addressable memory. The apparent conclusion is that extended Lisps supported by paged operating systems can provide sufficient data management capability for large text files. (However, we have not yet reached a point at which we can say how many text-pages can be stored per million words, since this involves a better understanding of how the dictionary grows as a function of increasing size of text.)

Experience with the present system's question answering capabilities suggests that it is quite weak in terms of human competence, but nevertheless effective for answering brief questions or accepting brief commands. As the question increases in length, the probability that a similar combination of constituents exists in the text falls off dramatically. There is nothing intrinsically difficult about long questions; every sentence SR as literally given by the text can be answered. But the richness of English paraphrase capabilities is so great that there is little probability that a long question will result in a set of constituents corresponding to a set in the text. This weakness can be corrected by three methods:

- Increasing the size of the system dictionary, particularly by including superset and instance terms for each word,
- Augmenting the dictionary with many paraphrase rules to translate from one SR to another,
- Designing the representation to be closer to some still unknown canonical form, e.g. kernel SRs, such that most meaning preserving paraphrases of that kernel will be translated into the kernel.

The third of these methods is the one on which research needs most to be concentrated. Related work on canonical forms for kernel sentences by Harris [1982] and others inspired by his approach suggests

directions for this research, and some benefits may be possible from something like Schank's [1973] primitives.

The most important area in text knowledge research continues to be that of discourse representation. This study has explored three alternative (hand-produced) representations -- SRs, Triples, and SSRs -- by examining their effectiveness for answering a sample of English questions. Two principles emerged:

- Inference rules that show a paraphrase relation between a question and an answering text can always be written, but to avoid excessive computation times in answering the question, they should be applied when translating the text or question to logical form.
- There is an inverse relation between the "depth" of representation of text and the number of explicit inference rules that must be applied to attain a given level of power in answering questions. One representation is "deeper" than another if it has a canonical form that includes more paraphrases than the other.

Research must continue on computing representations of text, both to discover optimal "canonical" forms and to establish procedures for computing discourse structures, resolving anaphora along the way. Additional study of the relation between canonical representations and question answering can be expected to increase the power and speed of the text knowledge system.

Development of a system for automatically (or even semi-automatically) translating book-length text into discourse trees should receive a very high priority. This work need not wait upon a final solution for the representation problem; it depends mainly upon the development of a very large lexicon and fairly complete grammars for English sentences. The experience of Slocum [1982] in German/English translation resulting in a German dictionary in computational form of more than ten thousand entries and a grammar that has succeeded in parsing over one thousand pages of German text is cited as evidence that large dictionaries and grammars are computationally manageable. At present it appears practical to prepare a lexicon and grammar for the fifty pages of handbook text to translate its sentences into semantic relations. Automatic resolution of anaphora and computation of discourse structures remain very active research areas and only partial solutions are available, but for the relatively small handbook text it appears possible to prepare rules that can combine English clauses into discourse structures patterned after those constructed by hand. It is probable that such a line of research will soon result in useful aids for constructing approximations to discourse structures that can be edited rapidly into final form.

References

Alterman, R., A System of Seven Coherence Relations for Organizing Event Concepts, PhD Dissertation, Dept. Comp. Sci., Univ. of Texas, Austin, 1982.

Harris, Z., *A Grammar of English on Mathematical Principles,* John Wiley and Sons, New York, 1982.

Kaplan, S.J., *Cooperative Responses from a Portable Natural Language Database Query System,* Stanford Heuristic Programming Project, HPP-79-19, Comp. Sci. Dept., Stanford, Palo Alto, Calif., 1979.

Schank, R.C. (1973). Identification of Conceptualizations Underlying Natural Language. In R.C. Schank, K. Colby (eds.), ^&Computer Models of Thought and Language\&, W.H. Freeman and Co., San Francisco, pp.187-247.

Simmons, R. F., *Computations from the English,* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1983.

Simmons, R. F., and Chester, D., Relating Sentences and Semantic Networks with Clausal Logic, mss. Univ. Texas., Dept. Comp. Sci., Austin, 1980, *Comm. ACM* Aug. 1982.

Slocum, Jonathan, An Experiment in Machine Translation, Proc. Assn. Comp. Ling., Phila., 1980; (personal communication 1983).

Appendix 1. Recorded Q/A Session

This appendix shows a recording of a question-answering session with the current version of TKS. No claim is yet made that the natural language system is very strong; indeed far more questions can be failed than answered, and many that we believe should be answered still require more work. The recording was made using the shallow SSR structure and almost every question requires the use of general inference rules. Navigational facilities are also illustrated using the LISP functions OPEN and OPTIONS. Comments on the recording are delimited by double angle brackets, << >>.

```
[PHOTO:  Recording initiated  Wed 26-Oct-83 6:19PM]

[Link from CS.SIMMONS, TTY36]

 Tops-20 Command processor 5.1(121354)


@ELISP
[Keeping Elisp]
Elisp, 9 9 83    <<Elisp is loaded, now we load the TKS>>

*(DSKIN "TKS1.HCP")
Files-Loaded

*(NETDB AIHDB)   <<NETDB creates a database from the text
NIL                tree in AIHDB>>

*(pp examples)   <<The following are a set of carefully
                     debugged English questions that the
                     TKS grammar and QA logic can handle>>

(DV EXAMPLES
((WHAT IS THE PROBLEM OF THE EIGHT-PUZZLE)
 (WHAT TWO THINGS DOES A STATE-SPACE REPRESENTATION USE)
 (WHAT DOES A STATE-SPACE REPRESENTATION USE)
 (WHAT GIVES SNAPSHOTS OF A PROBLEM)
 (WHAT TRANSFORMS A PROBLEM)
 (HOW MANY TILES ARE THERE IN AN EIGHT-PUZZLE)
 (HOW MANY TILES ARE IN AN EIGHT-PUZZLE)
 (HOW IS A TILE MOVED IN THE EIGHT-PUZZLE)
 (WHAT ARE THE COMPONENTS OF THE SPECIFICATION OF A STATE-SPACE REPRESENTATION)
 (WHAT IS THE COMPLETE SPECIFICATION OF A PROBLEM IN A STATE-SPACE
       REPRESENTATION)
 (WHAT IS THE SIZE OF THE STATE SPACE IN AN EIGHT-PUZZLE)
 (WHAT ENTITIES ARE USED BY THE STATE SPACE REPRESENTATION)
 (HOW DOES A DIRECTED GRAPH RELATE TO A STATE SPACE)
 (WHAT IS THE SOLUTION OF A PROBLEM IN A STATE-SPACE REPRESENTATION)
 (WHAT IS A STATE-SPACE)
 (WHAT IS A PARTIAL FUNCTION)))

*(DQA)  <<We call DQA to allow direct typing of English>>

 *** Welcome to the Text Knowledge System !! ***
 To use system just type in query, ending with a ? or .
 followed by a carriage return
 Type Q. to exit from the system

>What is a state-space?

CONTINUE? >(setq limit 2000)  <<This sets the interrupt at 2000
                                inferences before the next continue
                                is asked>>
```

```
2000
CONTINUE? >y

(QWD VBE NP)                    <<The syntactic pattern of the question>>

(STATE-SPACE DEF V)             <<The SR form of the translated question>>

The set of all attainable states of a problem is often called its state
space.

Type ? for help >?

S - Print Sr answer             T - Elapsed time
Y - Next candidate              A - Stop answering, return to TKS
E - Elaborate                   X - Expand
L - To look at query            R - Use RULE BASED QA
Anything else is evaluated in Lisp

Type ? for help >X    <<X to show a larger context for the answer>>

B1. State-Space Representation
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
An 8 -puzzle is a square tray containing eight square tiles of equal size,
The set of all attainable states of a problem is often called its state
space.
The 8-puzzle, for example, has a state space of size 9!2
The four operators defined for the 8 -puzzle form a set of partial functions
on the state space:

Type ? for help >t  <<To get a rough timing figure which includes
                      parsing and QA operations>>

770 ms

Type ? for help >l  <<repeats the question relation>>

(STATE-SPACE DEF V)

Type ? for help >e  <<Repeats the answer>>

The set of all attainable states of a problem is often called its state
space.

Type ? for help >y   <<To get another answer>>

a state space can be treated as a directed graph

Type ? for help >x

B1. State-Space Representation
The complete specification of a state-space problem has three components.
One is a set O of operators or operator schemata.
In addition, one must define a set s of one or more initial states.
and find a predicate defining a set G of goal states.
A state-space problem is then the triple S, O, G
.
A solution to the problem is a finite sequence of applications of operators
that changes an initial state into a goal state.
a state space can be treated as a directed graph

Type ? for help >y
```

a state space can be treated as a directed graph
whose nodes are states

Type ? for help >x

B1. State-Space Representation
The complete specification of a state-space problem has three components.
One is a set O of operators or operator schemata.
In addition, one must define a set s of one or more initial states.
and find a predicate defining a set G of goal states.
A state-space problem is then the triple S, O, G

.
A solution to the problem is a finite sequence of applications of operators
that changes an initial state into a goal state.
a state space can be treated as a directed graph

whose nodes are states
and whose arcs are operators transforming one state to another.
for example, if state 1 is a state to which any of three operators can be
applied, transforming it to state 2 , 3 , or 4 , then the corresponding graph
would be as in figure b1-3. nodes 2 , 3 , and 4 are called the successors of
node 1 node 1 | \ | \ | \ node node node 2 3 4 figure b1-3. directed
arcs.

Type ? for help >r
 * <<The * shows that r that called the rule-based QA found the same
      text as given above; a - would show that exactly the same
      answer had been found>>

CONTINUE? >y

CONTINUE? >y

CONTINUE? >n

.. Top TKS level..

{;;**}

>What is a partial function?

(QWD VBE NP)
(FUNCTION MOD (PARTIAL))
The four operators defined for the 8 -puzzle form a set of partial functions
on the state space:

Type ? for help >x <<Let's expand and see if there is a more
                      informative answer>>

A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
The four operators defined for the 8 -puzzle form a set of partial functions
on the state space:
Each operator, if it applies to a given state at all, returns exactly one new
state as its result.
in more complex problems, however, the operators often contain variables.

Type ? for help >t

601 ms

```
Type ? for help >a

... Returning to TKS...

{;;*****************************************************************}

>How does a directed graph relate to a state-space?

 Back to top level TKS

>How does a directed graph relate to a state space?

(RELATE AE (GRAPH AE* (DIRECT)) *TO (SPACE *OF (STATE)) INSTR Y)
 Number of answers : 0       Elapsed time : 36 msec

 Question : (RELATE AE (GRAPH AE* (DIRECT)) *TO (SPACE *OF (STATE)) INSTR Y)

 Do you want to use RULE BASED QA (Y/N) >y

a state space can be treated as a directed graph

Type ? for help >x

The complete specification of a state-space problem has three components.
a state space can be treated as a directed graph

Type ? for help >s   <<The detailed Database structure of the answer>>

(TREAT REF (GRAPH BK* ^1224-TREAT AE* ^1229-DIRECT PREP AS NODE ^1228-GRAPH)
AE (SPACE BK* ^1224-TREAT DET A *OF ^1227-STATE NBR SING NODE ^1226-SPACE)
NODE ^1224-TREAT)

Type ? for help >(open ^1228-graph)  <<Note that OPEN gives a somewhat
                                       larger context than X>>

B Problem Representation
B1. State-Space Representation
A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
The complete specification of a state-space problem has three components.
One is a set O of operators or operator schemata.
In addition, one must define a set s of one or more initial states.
and find a predicate defining a set G of goal states.
A state-space problem is then the triple S, O, G
.
A solution to the problem is a finite sequence of applications of operators
that changes an initial state into a goal state.
a state space can be treated as a directed graph

NIL
Type ? for help >t

1480 ms

Type ? for help >a

... Returning to TKS...

>(pp allans)  <<ALLANS contains the set of answers obtained for a
```

```
                question>>

(DV ALLANS
((TREAT REF
        (GRAPH BK* ^1224-TREAT AE* ^1229-DIRECT PREP AS NODE ^1228-GRAPH)
        AE
        (SPACE BK* ^1224-TREAT DET A *OF ^1227-STATE NBR SING NODE ^1226-SPACE)
        NODE
        ^1224-TREAT)))

{;;*****************************************************************}

>What entities are used by the state-space representation?

(QWD NP VP)
(USE AE (ENTITY) INSTR (REPRESENTATION *OF (STATE-SPACE)) AE Y)
 Number of answers : 0       Elapsed time : 79 msec

 Question : (USE AE (ENTITY) INSTR (REPRESENTATION *OF (STATE-SPACE)) AE Y)

 Do you want to use RULE BASED QA (Y/N) >y

A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another

Type ? for help >x

B Problem Representation
B1. State-Space Representation
A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
The complete specification of a state-space problem has three components.

Type ? for help >t

2047 ms

Type ? for help >a

... Returning to TKS...


{;;*****************************************************************}

>What is the size of the state space in the eight-puzzle?

CONTINUE? >y

CONTINUE? >y

(QWD VBE NP)
(SPACE *OF (STATE) PARTOF (EIGHT-PUZZLE) SIZE X)
 Number of answers : 0       Elapsed time : 161 msec

 Question : (SPACE *OF (STATE) PARTOF (EIGHT-PUZZLE) SIZE X)

 Do you want to use RULE BASED QA (Y/N) >y
```

```
CONTINUE? >y

The 8-puzzle, for example, has a state space of size 9!2

Type ? for help >t

7999 ms

Type ? for help >x

A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
The set of all attainable states of a problem is often called its state
space.
The 8-puzzle, for example, has a state space of size 9!2
--since there are 9 ! configurations of the tiles but only half this number
can be reached from any given starting configuration.

Type ? for help >y
 -                          <<an identical answer via a different
                              inference path>>
CONTINUE? >y
 -  -                       <<more identicals>>
CONTINUE? >y

 Back to top level TKS

{;;*************************************************************************}

>What is the complete specification of a problem in the state-space
> representation?

CONTINUE? >y

CONTINUE? >y

(QWD VBE NP)
(SPECIFICATION MOD (COMPLETE)
               *OF (PROBLEM RANGE (REPRESENTATION *OF (STATE-SPACE))))
The complete specification of a state-space problem has three components.
B1. State-Space Representation

Type ? for help >x

Ai Handbook
B Problem Representation
B1. State-Space Representation
A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
The complete specification of a state-space problem has three components.
One is a set O of operators or operator schemata.
In addition, one must define a set s of one or more initial states.
and find a predicate defining a set G of goal states.
A state-space problem is then the triple S, O, G

A solution to the problem is a finite sequence of applications of operators
that changes an initial state into a goal state.
a state space can be treated as a directed graph
```

```
b2. problem-reduction representation

Type ? for help >t

928 ms

Type ? for help >y
 *
 Number of answers : 1       Elapsed time : 1296 msec

 Question : (SPECIFICATION MOD (COMPLETE) *OF (PROBLEM RANGE (REPRESENTATION
*OF (STATE-SPACE))))

 Do you want to use RULE BASED QA (Y/N) >n

 .. Back to top level TKS ...


{;;**********************************************************************}

>What is an example of state-space representation?

(QWD VBE NP)
(REPRESENTATION *OF (STATE-SPACE) EXAMPLE X)
B1. State-Space Representation
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle

Type ? for help >x

Ai Handbook
B Problem Representation
B1. State-Space Representation
A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
The complete specification of a state-space problem has three components.
b2. problem-reduction representation

Type ? for help >?

S - Print Sr answer          T - Elapsed time
Y - Next candidate           A - Stop answering, return to TKS
E - Elaborate                X - Expand
L - To look at query         R - Use RULE BASED QA
Anything else is evaluated in Lisp

Type ? for help >l

(REPRESENTATION *OF (STATE-SPACE) EXAMPLE X)

Type ? for help >r

CONTINUE? >y

CONTINUE? >n


.. Top TKS level..
```

```
{;;*****************************************************************}

>What is an example of a problem-reduction representation?

(QWD VBE NP)
CONTINUE? >y

(REPRESENTATION *OF (PROBLEM-REDUCTION) EXAMPLE X)
b2. problem-reduction representation

Type ? for help >x

Ai Handbook
B Problem Representation
B1. State-Space Representation
b2. problem-reduction representation
often distinguished from the state-space representation of problems is a
technique called problem-reduction representation.
in the problem-reduction approach, the principal data structures are problem
descriptions or goals.
the transformations permitted are defined as operators.
an example that lends itself nicely to problem-reduction representation is
the famous tower of hanoi puzzle.
NIL
Type ? for help >t

859 ms

Type ? for help >a

... Returning to TKS...


{;;*****************************************************************}
>
>How many tiles are there in an eight-puzzle?

(ADJ NP VP)
(TILE QTY Y *OF (EIGHT-PUZZLE))
An 8 -puzzle is a square tray containing eight square tiles of equal size,

Type ? for help >x

B1. State-Space Representation
A straightforward example of state-space representation is the simple,
well-known puzzle called the eight-puzzle
An 8 -puzzle is a square tray containing eight square tiles of equal size,
The space for the ninth tile is vacant see fig. b1-1

|----|----|----|
| 2. | 1. | 6. |
|----|----|----|
| 4. |    | 8. |
|----|----|----|
| 7. | 5. | 3. |
|----|----|----|
figure b1-1. an 8 -puzzle.

A tile may be moved by sliding it vertically or horizontally into the empty
square.
The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.
The set of all attainable states of a problem is often called its state
```

space.
The four operators defined for the 8 -puzzle form a set of partial functions on the state space:

Type ? for help >a

... Returning to TKS...

{;;**********}

>(? (options tile))

^1041-TRAY
(TILE SHAPE ^1045-SQUARE QTY ^1046-EIGHT NBR PL *OF ^1047-SIZE AE*
^1049-NUMBER NODE ^1044-TILE)
((OPTIONS TILE))

ANOTHER? >(open ^1044-tile]

B Problem Representation
B1. State-Space Representation
A state-space representation of a problem employs two kinds of entities: states, which are data structures giving "snapshots" of the condition of the problem at each stage of its solution, and operators which are means for transforming the problem from one state to another
A straightforward example of state-space representation is the simple, well-known puzzle called the eight-puzzle
An 8 -puzzle is a square tray containing eight square tiles of equal size, The space for the ninth tile is vacant see fig. b1-1

2.	1.	6.
4.		8.
7.	5.	3.

figure b1-1. an 8 -puzzle.

A tile may be moved by sliding it vertically or horizontally into the empty square.
The problem is to transform some particular tile configuration, say, that of figure b1-1, into another given tile configuration, say, that of figure b1-2.
The set of all attainable states of a problem is often called its state space.
The four operators defined for the 8 -puzzle form a set of partial functions on the state space:
The complete specification of a state-space problem has three components.

NIL
ANOTHER? >n

NIL
>What does a state-space representation use?

(USE INSTR (REPRESENTATION *OF (STATE-SPACE)) AE Y)
Number of answers : 0 Elapsed time : 102 msec

Question : (USE INSTR (REPRESENTATION *OF (STATE-SPACE)) AE Y)

Do you want to use RULE BASED QA (Y/N) >y

A state-space representation of a problem employs two kinds of entities:

```
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another

Type ? for help >t

965 ms

Type ? for help >a

... Returning to TKS...

{;;*****************************************************************}

>What gives snapshots of a problem?

(GIVE INSTR Z AE (SNAPSHOT *OF (PROBLEM)) U Y)
 Number of answers : 0       Elapsed time : 58 msec

 Question : (GIVE INSTR Z AE (SNAPSHOT *OF (PROBLEM)) U Y)

 Do you want to use RULE BASED QA (Y/N) >y

A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another

Type ? for help >t

3271 ms

Type ? for help >a

... Returning to TKS...


{;;*****************************************************************}
>
>What is the problem of the eight-puzzle?

(QWD VBE NP)
(PROBLEM *OF (EIGHT-PUZZLE))
The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.
An 8 -puzzle is a square tray containing eight square tiles of equal size,

Type ? for help >y

 Number of answers : 1       Elapsed time : 1729 msec

 Question : (PROBLEM *OF (EIGHT-PUZZLE))

 Do you want to use RULE BASED QA (Y/N) >y

An 8 -puzzle is a square tray containing eight square tiles of equal size,
The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.

Type ? for help >n

** Illegal command **
```

```
Type ? for help >a

... Returning to TKS...

{;;*****************************************************************}

>What transforms a problem?

(TRANSFORM INSTR Z AE (PROBLEM) U Y)
 Number of answers : 0       Elapsed time : 144 msec

 Question : (TRANSFORM INSTR Z AE (PROBLEM) U Y)

 Do you want to use RULE BASED QA (Y/N) >y

A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another

Type ? for help >x

B1. State-Space Representation
A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another

Type ? for help >a

... Returning to TKS...

{;;*****************************************************************}

>What is an operator?

(QWD VBE NP)
(OPERATOR DEF V)
A state-space representation of a problem employs two kinds of entities:
states, which are data structures giving "snapshots" of the condition of the
problem at each stage of its solution, and operators which are means for
transforming the problem from one state to another

Type ? for help >y  <<We've already seen that answer, let's find more>>
 *
The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.

Type ? for help >y

The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.
The operators, corresponding to possible moves, might be defined with
separate operators for each of tiles 1 through 8

Type ? for help >y

The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.
However, a more concise definition is made possible by viewing the empty
square as the object to be moved and stating the operators in terms of the
movements of this square
```

```
Type ? for help >y

In this formulation, only four operators are used:

Type ? for help >y

In this formulation, only four operators are used:
Up move the blank up one square

Type ? for help >x

The problem is to transform some particular tile configuration, say, that of
figure b1-1, into another given tile configuration, say, that of figure b1-2.
However, a more concise definition is made possible by viewing the empty
square as the object to be moved and stating the operators in terms of the
movements of this square
In this formulation, only four operators are used:
Up move the blank up one square
Down move the blank down one square
Left move the blank left one square
Right move the blank right one square
an operator may be inapplicable in certain states, as when it would move the
blank outside the tray of tiles.

Type ? for help >t

1833 ms

Type ? for help >y

In this formulation, only four operators are used:
Down move the blank down one square

Type ? for help >y

In this formulation, only four operators are used:
Left move the blank left one square

Type ? for help >y

In this formulation, only four operators are used:
Right move the blank right one square

Type ? for help >y

an operator may be inapplicable in certain states, as when it would move the
blank outside the tray of tiles.

Type ? for help >y

Each operator, if it applies to a given state at all, returns exactly one new
state as its result.

Type ? for help >y

if, for a particular state and operator, the variables can be instantiated in
more than one way,then each instantiation yields one new state, and the
operators of the problem, if they are to be considered as defining functions,
are more accurately termed operator schemata.

Type ? for help >x

The four operators defined for the 8 -puzzle form a set of partial functions
```

```
on the state space:
in more complex problems, however, the operators often contain variables.
if, for a particular state and operator, the variables can be instantiated in
more than one way,then each instantiation yields one new state, and the
operators of the problem, if they are to be considered as defining functions,
are more accurately termed operator schemata.

Type ? for help >y

an operator may change a single problem into several subproblems;

Type ? for help >y

an operator may change a single problem into several subproblems;
in addition, several different operators may be applicable to a single
problem, or the same operator may be applicable in several different ways.

Type ? for help >x

b2. problem-reduction representation
the transformations permitted are defined as operators.
an operator may change a single problem into several subproblems;
to solve the former, all the subproblems must be solved.
in addition, several different operators may be applicable to a single
problem, or the same operator may be applicable in several different ways.
in this case, it suffices to solve the subproblems produced by any one of the
operator applications.
a problem whose solution is immediate is called a primitive problem.
thus, a problem representation using problem reduction is defined by a triple
consisting of--
reasoning proceeds backward from the initial goal.

Type ? for help >a

... Returning to TKS...

>
>q.

..Exiting TKS .....
NIL

*^C
@pop

[PHOTO:  Recording terminated  Wed 26-Oct-83 6:52PM]
```

Appendix 2

```
((N100 (REPRESENTATION *OF PROBLEM SNT C98 LABEL (BB0) INST N101 INST N102))
 (N101 (REPRESENTATION *OF N80 SNT C99 LABEL (BB1) *FROM* N103 INST N137))
 (N79 (PROBLEM NODE N79))
 (N80 (STATE-SPACE *OF N79 EQUIV* N79 SNT C100))
 (N102 (REPRESENTATION *OF N50 EQUIV* N50 SNT C201 HAS N105 EQUIV* N130 AE* N103
       ))
 (N50 (REDUCTION *OF N104 EQUIV* N104 SNT C201))
 (N104 (PROBLEM EQUIV N106 EQUIV N102 SNT C98))
 (N106 (GOAL NBR PL EQUIV* N104 SNT C201))
 (N103 (DISTINGUISH TNS PAST AE N102 *FROM N101 FREQ OFTEN SNT C200))
 (N105 (STRUCTURE NBR PL *OF DATA EQUIV N107 SNT C201))
 (N107 (DESCRIPTION *OF N104 EQUIV* N106 AE* N109 SNT C201))
 (N108 (GIVE TNS PSTPRT MD POSSBLE AE N51 RESULT N110 SNT C202))
 (N51 (DESCRIPTION SUP N107 NBR SING ORDER INITIAL))
 (N110 (SOLVE TNS PAST AE N107 INSTR N111 SNT C202))
 (N109 (CHANGE TNS PRES TIME ULTIMATELY AE N107 SNT C202 INSTR N111 *TO N112))
 (N111 (TRANSFORMATION NBR PL ORDER SEQUENCE INST N114 SNT C202))
 (N112 (SUBPROBLEM *OF N106 SUP N106 NBR PL HAS N89 EQUIV N113 SNT C207))
 (N89 (SOLUTION NBR SING TYPE IMMEDIATE SNT C202))
 (N113 (PROBLEM NBR SING TYPE PRIMITIVE SNT C207))
 (N114 (TRANSFORMATION NBR PL AE* N52 EQUIV N115 SUP N111 SNT C203))
 (N52 (PERMIT TNS PAST))
 (N115 (OPERATOR NBR PL INSTR* N116 INST N90 INST N91 SNT C203))
 (N116 (CHANGE TNS PRES MD MAY AE N117 *TO N118 SNT C204))
 (N117 (PROBLEM SUP N104 QTY 1. SNT C204))
 (N118 (SUBPROBLEM NBR PL QTY SEVERAL INST N119 *OF N117 SUP N117 SNT C204))
 (N119 (SUBPROBLEM NBR PL SUP N118 QFY ALL AE* N120 SNT C204))
 (N120 (SOLVE TNS PAST MD POSSBLE RESULT N121 AE N119 SNT C204))
 (N121 (SOLVE TNS PAST MD POSSBLE RESULTOF N120 AE N117 SNT C204))
 (N122 (APPLY MD POSSBLE *TO N53 AE N90 LST* N124 SNT C205))
 (N53 (PROBLEM SUP N104))
 (N90 (OPERATOR SUP N115 QTY SEVERAL IDENT DIFFERENT AE* N122 SNT C205))
 (N123 (APPLY *TO N53 AE N91 LST* N124 MANNER N54 SNT C205))
 (N54 (WAY NBR PL QTY SEVERAL IDENT DIFFERENT NODE N54 SNT C205))
 (N91 (OPERATOR SUP N115 IDENT SAME QTY ONE AE* N123 SNT C205))
 (N124 (OR [LST N122 N123] RESULT N125 RESULT N127 INSTR* N128 SNT C206))
 (N125 (SUBPROBLEM NBR PL SUP N118 SNT C206))
 (N126 (SUBPROBLEM NBR PL SUP N125 SNT C206))
 (N127 (APPLICATION QTY ONE QFY ANY *OF N90 *OF N91 RESULT N126 SNT C206))
 (N128 (SOLVE AE N55 INSTR N124 RESULT N129 SNT C206))
 (N55 (SUBPROBLEM SUP N126 QFY ALL))
 (N129 (SOLVE MD POSSBLE AE N53 SNT C206))
 (N130 (TRIPLE EQUIV N102 HAS N133 HAS N131 HAS N134 SNT C208))
 (N131 (DESCRIPTION NBR SING SUP N107 AE* N92 *FROM* N135 AE* N136 SNT C208))
 (N132 (SUBPROBLEM SUP N118 NBR PL SNT C208))
 (N133 (OPERATOR NBR PL SUP N115 QTY SET INSTR* N92 SNT C208))
 (N92 (TRANSFORM TNS INF AE N131 *TO N132 INSTR N133 SNT C208))
 (N134 (DESCRIPTION NBR PL EQUIV* N113 SUP N107 *OF N113 QTY SET SNT C208))
 (N135 (REASON TNS PRPART *FROM N131 *TO N134 AE* N136 SNT C209))
 (N136 (PROCEED TNS PRES DIRECTION BACKWARDS AE N135 SNT C209))
 (N137 (REPRESENTATION SUP N101 HAS N138 EG N139 SNT C100))
 (N138 (ENTITY NBR PL QTY 2. INST N140 INST N141 SNT C100))
 (N140 (STATE NBR PL EQUIV N142 C100))
 (N142 (STRUCTURE *OF DATA INSTR* N143 SNT C100))
 (N143 (GIVE TNS PRES INSTR N142 AE N144 *AT N145 SNT C100))
 (N144 (SNAPSHOT NBR PL *OF N146 SNT C100))
 (N146 (PROBLEM NBR SING HAS N145 SUP N79 SNT C100))
 (N145 (STAGE NBR PL IDENT VARIOUS *OF N147 SNT C100))
 (N147 (SOLUTION NBR SING SNT C100))
 (N141 (OPERATOR NBR PL EQUIV* N148 SNT C100))
 (N148 (PROCEDURE NBR PL INSTR* N149 SNT C100))
```