APPENDIX 5

SOME THOUGHTS ON AUTOMATIC THEOREM PROVING IN DATA BASE DESIGN AND USE

by

W. W. Bledsoe

The paper sketches some of the ways in which research in Automatic Theorem Proving (ATP) can support the interdisciplinary project on Data Base Methodology being conducted at The University of Texas.

DATA BASES

Here we treat a data base as a list of facts and information (which might be distributed over several geographic locations), along with a set of rules of inference for using these facts. (Figure 1)

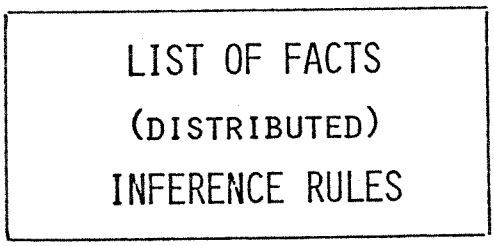


Figure 1
A Data Base

Querries to this data base are processed by

a) Direct lookup

b) By Inference

Also the data base must be tested somehow for internal consistency.

For example, if we have the statements

1) John is older than Mary

2) Mary is 15 years old

in the base, we want to answer querries such as

a) Is Mary 15 years old?

b) Is Mary older than 25?

c) Is John older than 12?

The last two, of course, would require simple inference. Much more complicated cases are desirable and, to some extent, possible.

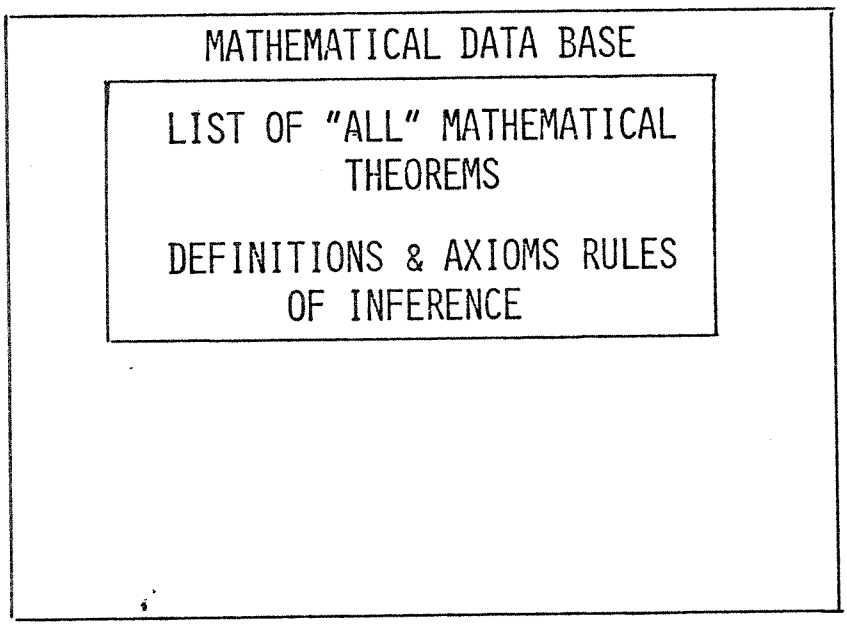
If we add the entry

3) Mary is younger than 4 years,

what does the whole thing mean? What, if anything, can it be use for?

EXAMPLES

For example we might have a mathematical Data base (Figure 2)

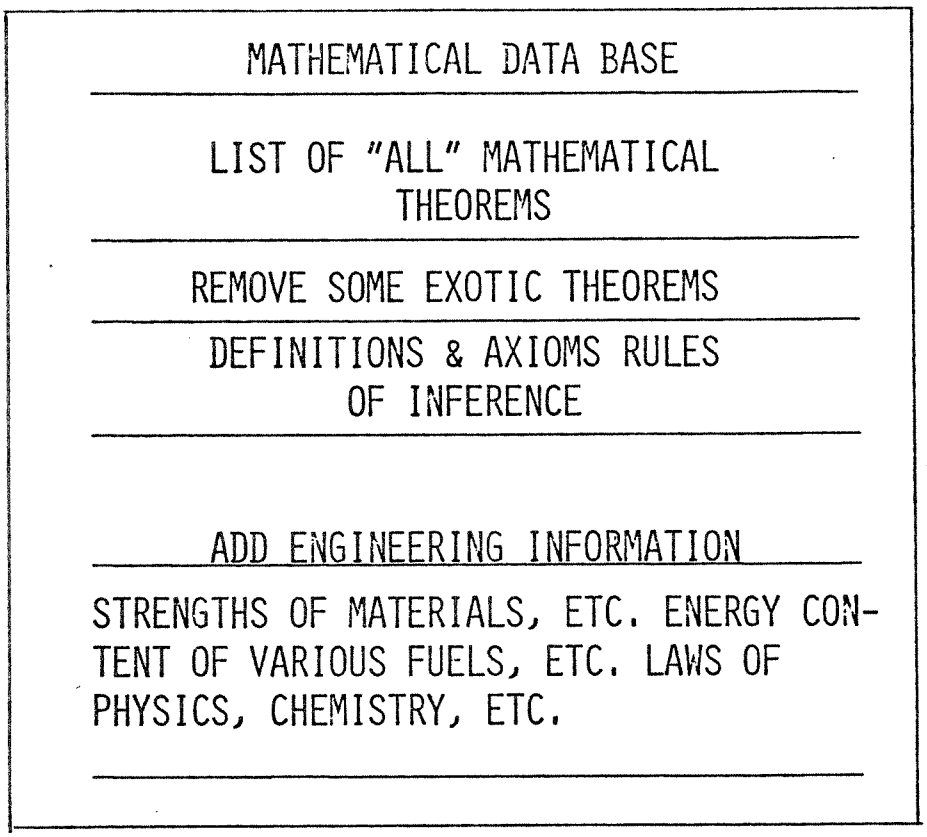


A. PROVE A THEOREM BY FINDING IT ALREADY IN THE DATA BASE.

B. PROVE A THEOREM BY INFERRING IT FROM THEOREMS IN THE DATA BASE.

Figure 2

We can extend this to an engineering data base (Figure 3).



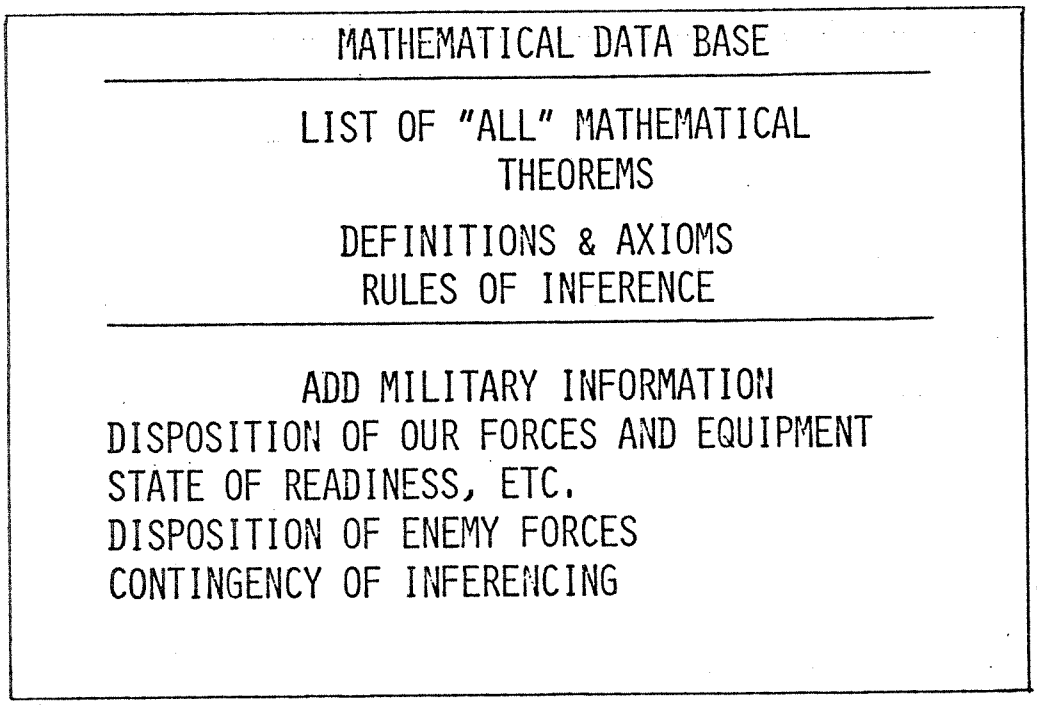
A. Prove a theorem by finding it already in the Data Base.

B. Prove a theorem by Inferring it from Theorems in the Data Base.

C. Answer question about the design of a bridge or the feasibility of a space probe.

Figure 3

Or extend it to a military data base (Figure 4).



A. PROVE A THEOREM BY FINDING IT ALREADY IN THE DATA BASE.

B. PROVE A THEOREM BY INFERRING IT FROM THEOREMS IN THE DATA BASE.

C. ANSWER A QUESTION ON OUR ABILITY TO REPULSE A CONJECTURED ATTACK.

Figure 4

Many other examples easily come to mind.

SOME OBSERVATIONS

Several points can be made

- AUTOMATIC INFERENCING IS CLEARLY DESIRABLE IN ALL SUCH EXAMPLES.
- IF WE HAD TRULY POWERFUL AUTOMATIC THEOREM PROVERS, IT WOULD CHANGE OUR CURRENT PROPOSAL FOR DATA BASE DESIGN.
- FOR THE NEXT 10-30 YEARS WE MUST SETTLE FOR A "MODERATE" ABILITY OF ATPS, BUT EVENTUALLY ATP WILL DOMINATE.
- THE PRESENT PROJECT SHOULD USE THIS MODERATE CAPACITY, USING AUTOMATIC INFERENCING BUT NOT EXPECTING TOO MUCH.

ATP AT THE UNIVERSITY OF TEXAS

The University of Texas has been one of the leading centers for ATP since 1968, and is the most successful in actually carrying out proofs of moderately difficult theorems on the computer. Our provers have been LISP programs for the CDC 6600 computer and the DEC 10. In addition to a number of theorems proved in set theory[1], calculus[2], analysis[3,7,9], and topology[3,9], we have seen our program and ideas successfully used in program verification systems [5,6], and in incremental design of programs with documentation and verification [10]. Also we have been early proponents of new directions [9] now finding their way in ATP research.

Others at UT (Skilossy, Simmons, Chester, and other students) have been or are now engaged in some form of ATP research. The research on inference in semantic net [11] seems especially pertinent here.

ROLE IN DATA BASE DESIGN

We would expect to use the concepts from ATP, not the actual programs in data base design. A team effort would insure that ATP ideas would be integrated into the project in an effective way.

The research would blend nicely with a larger effort here, funded by NSF, on general ATP.

At this time we feel that inferencing (in Data bases) can best be done at two levels

1) At the hardware level (simple inferences)

2) At the software level.

In 2) certain "pertinent" information is retrieved from the data base ("semantic paguig") to be used in core memory for deeper inferencing.

We believe that only a moderate capability in ATP can be depended upon during the next 10-30 years. However in the long run ATP will be the dominant factor in Data Base design. It is crucial that ATP research, geared to that application, (here and elsewhere) be supported in the interim.

Two important factors in data base design are

a) conflicting data

b) changing data

These seem to point more toward

1) automatic inferencing, and

2) man-machine cooperation.

LONG-TERM RESEARCH INTEREST

Our group here has a long-term interest in deep inference in data bases, where a sizable ATP capacity is required. We will be pursuing this interest independent of this project.

Included in our concerns are

- uncertain and conflicting knowledge
- predicting with probabilities
- (limited) natural language input and output
- man-machine interaction

EXAMPLE

DATA BASE

ALL THE NEWS PAPER STORIES ON THE MIDDLE EAST FILED BY THE MAJOR NEWS AGENCIES DURING THE LAST 10 YEARS. (WITH OLDER STORIES CAREFULLY CULLED)

\+ RULES FOR INFERENCING

QUERRY:

WHAT IS THE LIKELIHOOD OF SYRIA ATTACKING ISRAEL WITHIN THE NEXT TWO DAYS?

TASK:

DETERMINE THE SOURCE OF THE MAJOR INCONSISTENCIES IN THE DATA BASE.

We could add to this

INTELLIGENCE INFORMATION ON

- TROOP STRENGTHS AND DEPLOYMENT
- RECENT MOVEMENT

ETC.

Man-Machine Interaction

1. The user would add, subtract, or change, data and inferencing rules.

2. The user could help with the inferencing on difficult problems (e.g., suggesting relevant facts).

Man-Machine Interaction

1. The user would add, subtract, or change, data and inferencing rules.

2. The user could help with the inferencing on difficult problems (e.g., suggesting relevant facts).

REFERENCES

1. W. W. Bledsoe, Splitting and reduction heuristics in automatic theorem proving. A. I. Jour. 2(1971), 55-71.

2. W. W. Bledsoe, Robert S. Boyer and William H. Henneman, Computer Proofs of Limits Theorems. A. I. Jour. 3(1972), 27-60.

3. W. W. Bledsoe and P. Bruell, A man-machine theorem-proving system. A. I. Jour. 5(1974), 51-72.

4. W. W. Bledsoe and Mabry Tyson, The UT Interactive Theorem Prover. The University of Texas at Austin, Math. Department Memo ATP-17, May 1975.

5. D. I. Good, R. L. London and W. W. Bledsoe, A Complete Method for Higher Order Logic. Ph.D. thesis, Case Western Reserve Univ. Jennings Computer Center Report 1117.

6. W. W. Bledsoe and Mabry Tyson, Typing and Proof by Cases in Program Verification. Machine Intelligence 8, Donald Michie and E. W. Elcock (Eds.), Ellis Horwood Limited, Chichester, pp. 30-51.

7. A. Michael Ballantyne and W. W. Bledsoe, Automatic Proofs of Theorems in Analysis Using non-standard Techniques. J. ACM, Vol. 24(1977) pp. 353-374.

8. W. W. Bledsoe, Non-resolution Theorem Proving. University of Texas Math. Department Memo ATP-29, Sept. 1975. To appear in the A. I. Jour.

9. W. W. Bledsoe, A Maximal Method for Set Variables in Automatic Theorem Proving. University of Texas Math. Department Memo ATP-33A, July 1977. To be presented at IJCAI-77, MIT, Aug. 1977.

10. Mark Moriconi, An Interactive System for Incremental Program Design and Verification

11. D. Chester and R. F. Simmons, Influences in Quantified Semantic Networks.

A COMPUTER ARCHITECTURE FOR A FDSS

Jack Lipovski

Computer architecture aims to make recent advances in hardware technology (especially LSI) useful to the new and demanding software envisioned for a very large distributed and intelligent data base management system. Some preliminary architectural features of a planned system are herein sketched and some problems for research and development are delineated.

Three major computing systems are to be accomodated. Firstly, users interface with the data base system through a network of intelligent terminals. Secondly, intelligent discs are located at various nodes in this network and are powerful enough to search the data where it is stored to avoid shipping large quantities of data through the network. Thirdly, an array computer will use parallelism to extend the analytical capacity of artificially intelligent software. We submit that these three major systems have to be accomodated because none of them alone, nor any pair of them, are adequate to support the envisioned software. However, each system can be effectively and economically built with LSI modules, so the total system will take advantage of LSI economics. We aim to design each system so that it will interface well with the other systems and, conversely, we are relieved of the need to perform each function in any one system alone. The object of studying the three systems together is to develop each one of them to fit together later on in an integral system. While we do not propose to build all of them in this project, but only the intelligent disc, we will design the disc to support distributed queries in the network and to support deep theorem proving in the array computer. Other proposals have been or will be submitted to study the other systems. If the other proposals are funded, they will be used in research conducted in this proposal. Otherwise, they will be simulated in this proposed research effort.

Moreover, each system will be designed to work as I/O devices with existing computers to provide considerable improvement in their performance, even though the greatest improvement in performance can only be expected from using all three computing systems together in a total system.

In the following paragraphs, we outline the three systems. The intelligent disc, which we plan to build, will be discussed in more detail. The other two will be sketched for completeness.

1. Other Systems Architectures

1.1 The Network

The network will consist of small microcomputers in intelligent terminals and intelligent secondary memories and communication will be accomplished by packet switching in the network. Although the terminals deserve some study, we need not specify them at this stage except to say that they have to be able to maintain the user's schema, a compiler for the data base language, and means to direct packets, embodying the query, through the network to the intelligent discs. Upon sending out a query from an intelligent terminal, the object of a packet will generally be a file on an intelligent disc. The file will be explained in the next section. A group of files will be at one physical node of the network of different cylinders of the disc, or even in tertiary memory in that node. There may be several physical nodes distributed through the network. In processing a complex query, references from one file to another will require that packets be sent from files to files as well. In retrieving the answer to a query, packets will be sent out from files to intelligent terminals.

This architecture requires that the intelligent disc node be able to examine an incoming packet to determine which file is to be operated on. A queue of incoming packets will be buffered and scheduled by a

conventional microcomputer associated with the disc at the node. Records will be checked for locking to prevent interference among queues. Once a file is in position to be searched by the logic in the intelligent disc, and all required records have been locked to the user, the file will be entirely searched in each disc revolution, as discussed in the next section.

Of significance, this network architecture combines the problem of accessing file data from intelligent terminals with the problem of solving complex queries where one file has to be linked up with other files. It offers hope in simplifying problems of protection, lockout and deadlock by locking records within the intelligent disc.

1.2 The Array Processor

The array processor will be used to support artificially intelligent software by means of parallelism. Two forms of parallelism are useful. In vector parallelism, a very wide word width processor is created by work on vector operands. This is commonly referred to as single instruction stream multiple data stream (SIMD) processing. In concurrent parallelism, small independent processors simultaneously but independently operate on separate pieces of data. This is referred to as multiple instruction stream multiple data stream (MIMD) processing.

In artificial intelligence programs using LISP, lists can be "vectorized" by writing them as paranthesized strings. Operations like EQUAL can be executed on two strings as though they were vectors. Operations like CDR or CAR can be done in a parallel machine as simply as in a conventional machine, but no better. However COND and MAP do not take much advantage from vector parallelism.

In a concurrent machine, each independent processor can evaluate different lists using standard LISP techniques. Potentially, all

LISP primitives can be executed faster through parallelism. In order not to have to store the entire LISP interpreter in each memory, a set of common memories store fragments of the interpreter. Each processor can fetch instructions from one of the common memories, but all processors accessing any one common memory must be accessing exactly the same word in it. With a fixed program like a LISP interpreter, we believe it will be possible to carefully schedule fragments into common memories to use this technique. Then very small, cheap processors with a small amount of local memory should be able to efficiently execute concurrent LISP programs.

The key to both vector and concurrent parallelism is the design of a powerful but inexpensive computer switching array. We have submitted a proposal to NSF to build a prototype computer using such a switch. This computer can be used to experiment with concurrent and vector parallelism in executing artificial intelligence programs.

2. Intelligent Disc Architecture

From our earlier work on the CASSM system at the University of Florida and from related work on the RAP system at the University of Toronto, we have established techniques which will efficiently store relational data bases and semantic networks on a disc. The logic associated with the disc makes it sufficiently intelligent to resolve almost all typical relational queries and sufficiently intelligent to greatly assist extracting useful data from a semantic network for artificial intelligence programs.

2.1 Physical Description of Disc Hardware

The disc architecture will consist of multiple moving head discs, in which all heads are on a common frame, and there is one head on each disc surface. (We are looking at IBM 3330 or equivalent discs

that store about 109 bits per removable disc pack). By moving the frame, the heads are located over a given "cylinder". One or more such discs will be operated together so that their "cylinders" form a larger cylinder. The data on this larger cylinder is called here a file. For instance, if three IBM 3330 discs are operated together, a larger cylinder would have 60 heads on 60 surfaces. Two hundred such files are stored on the 200 cylinders of an IBM 3330 disc. More files may be stored in tertiary memory, and paged into cylinders of the disc. Exactly one file will be under the moving disc heads at any time.

In one revolution of the discs, an "instruction" is executed on the entire file. A typical query consists of in the order of ten instructions which will be executed on the same file. Upon receipt of a query at an intelligent disc by the microprocessor that controls the disc, the file requested by the query is positioned under the heads, either by moving the heads or by moving the file in from tertiary memory. The heads remain positioned over the file as the disc revolves, to execute the query, for about ten revolutions. The heads are then positioned over the file needed by the next query. Each head will have a "microprocessor" similar in complexity to the popular microprocessors but having quite different organization and instruction set. It will be attractive to put each "microprocessor" in an LSI chip. A disc track and "microprocessor" are called here a cell. The logic looks like a chain of identical cells. See figure 1.

2.2 Storage of Data

The file consists of records of a variable number of words, and the words are fixed length and are divided into fields. Each

word has a mark bit on the disc for content search operations. Records correspond to tuples in the relational data base system and to nodes in semantic networks. The first word of each record stores a bit stack for context search operations. Other words appear to store domain names and items in the tuple, or arcs incident from the node in the network. See figure 2 for storage of relational tables and figure 3 for storage of (semantic)networks. Figure 2a shows two relations while figure 2b shows their storage on the disc. Figure 3a shows a network like a semantic network, while figure 3b shows storage of the network on the disc.

Note, in figure 2b that a file can contain many relations (tables) and that each tuple (row) is stored as a record. In this figure, two relations (officer and parts) happen to be stored on the same file. Note that the tuples from different relations can be intermixed, but that a field in the first word in each tuple or record identifies the relation that the tuple is in by a code word. For each domain (column), a word containing a pair of code words in fields, domain name and domain value, are stored.

Note, in figure 3b that each node is stored as a record and records are numbered according to their position from top to bottom in the file. For instance, node "Tom" is stored in the 21st record from the top of the file. For each node, its corresponding record contains in its first word a field containing the code word of the node name and in succeeding words a pair of fields associated with each are in the network that is incident out of the node. The fields are best explained by example. For instance, in record 20 corresponding to the node "John", the arc (John, father, Tom) is represented by

the word (father, 21) where "father" is a code word and 21 is a number, since 21 is the record number for the node "Tom".

Though not fully shown in the above examples, the key problem is efficient storage of data. That is why code words are used rather than character strings. A mechanism to convert between code words and character strings by means of hardware has been worked out. Moreover, the left field of each word can be generated by means of a counter in the "microprocessor" associated with the disc track rather than stored on it if the code words are consecutively numbered. These fields that are generated in the "microprocessor" are called imaginary fields. The user need not concern himself about whether data is stored in real or imaginary fields, for instructions will treat the files as shown in figures 2b or 3b, whether or not some left fields are actually generated by hardware.

2.3 Content Searching

Each word is provided with a bit (mark bit) which can be modified by a content search instruction. The bit is set if the content of the word matches the argument of the instruction, and is cleared otherwise. For instance, in figure 2b, if the operand of a content search were P#, 30, then the mark bit of the eighth word would be set and all other mark bits would be cleared. Content searching is normally used to single out individual words to be rewritten, output, or deleted.

2.4 Set Oriented Context Searching for Relational Data Bases

Each record is provided with a bit stack located in the first word of the record. If the argument of a context search "push", instruction is in a record, a 1 bit is pushed on the bit stack for

that record, else a 0 bit is pushed on the bit stack. A context search instruction could "AND" the result of the search with the top bit on each bit stack, or "OR" or "AND the COMPLEMENT", etc.

Consider a query to locate Captain Smith in figure 2. The query is translated into the following program:

```
1)  PUSH            "IS-AN"       "OFFICER"
2)  AND             "RANK"        "CAPTAIN"
3)  AND             "NAME"        "SMITH"
4)  MARK(OUTPUT)    "LOCATION",   ————————
```

Instruction 1 pushes a 1 onto the bit stacks for the first and third records, and a 0 onto that of the second record. Instruction 2 AND's a 1 bit onto the bit stack of the first record, but AND's a 0 onto the other bit stacks. Instruction 3 does the same (in this simple example). Instruction 4 marks, by content searching within records that have a 1 bit on top of their bit stacks, the words whose left fields are "LOCATION". The marked words are output as the response to this query. This query is effectively answered in four disc revolutions. Typical queries should be answered in ten disc revolutions, independent of the size or complexity of the file.

In hardware, the results of the search are stored temporarily in a one bit wide random access memory, which has 1 bit per record, and are processed by pipelining to appear to move the results to the bit stack so that one context search can be executed each disc revolution. Complex Boolean queries can be analyzed over all tuples in a file in a number of revolutions proportional to the number of terms in the query expression and independent of the size of the file. (No conventional data base management system can approach

this ideal.) Moreover, there is no need for directories to locate relations within a file and tuples from a relation can be scattered throughout the file because the entire file is searched each disc revolution.

2.5 Other Set-Oriented Functions

It is possible to find the intersection of two sets in two disc revolutions. The one bit wide, RAM (mentioned in section 2.4) is initially cleared. In the first revolution, elements (code words) of the first set provide addresses to set bits of the RAM. In the second revolution, elements (code words) of the second set provide addresses to read bits from the RAM. If a 1 is read, the word of the second set is marked. Only if an element is in the intersection will that bit be both set and read, and the word marked. (Other researchers have also shown that duplicates can be deleted by a similar procedure.)

It is possible to execute an inner product "threshold search" as shown in figure 4. Each word on the disc has an associated weight, as word A in record 23 has weight 3. The argument of the instruction also has a weight. The argument, its weight, and a storage buffer are in registers in each head. If the word matches the argument, the two weights are multiplied and saved in the buffer. The bottom word of each record contains an accumulator, the number in the buffer is added to it. Thus, an inner product "threshold" search can be conducted simultaneously over all records in a file.

The buffer can also be used for simpler functions. The maximum, minimum, sum or count of marked words can be conducted in each record or the entire file. In particular, after an inner product

threshold search, the set with the maximum accumulator value can be marked. Equally important, the number of marked words can be counted before they are output, to determine whether there are too many to be of interest.

2.6 Network Oriented Context Search Instructions

Pointers from one record to another are stored by putting the record number of the second record in the right field of a word in the first record. See figure 3 again, where the "father" pointer from record 20 (for "John") points to record 21 (for "Tom"). The RAM discussed earlier is used to transfer tokens. The RAM is initially cleared. If the argument of a token transfer search is found in the left field of a word in a record having a 1 bit on the top of its stack, the right field is used as an address to set a bit in the RAM. In the following revolution, a counter that counts records as they pass over the head is used to address the RAM to push the values stored there onto the bit stacks of the records. Pipelining allows the second revolution to be "hidden" so that tokens can be effectively transferred in one disc revolution.

Consider a query to find the grandsons of "John" in figure 3. The instructions are:

```
1)  PUSH           "IS-A",     "John"
2)  PUSH           "FATHER",   TOKEN
3)  PUSH           "FATHER",   TOKEN
4)  MARK(OUTPUT)   "IS-A",     ------
```

After the first revolution, a 1 is effectively pushed onto the bit stack of record 20, and a 0 is pushed onto all other stacks as discussed in section 2.4. After the second revolution, a 1 bit is

pushed onto the bit stacks of records 21 and 23 simultaneously, and after the third, a 1 bit is pushed onto that of record 25. After the next revolution, the work "IS-A", "BILL" is output. Such a query is effectively executed in four revolutions.

One of the most useful applications of pointers and token transfers across pointers is semantic paging for deep theorem proving programs. See figure 5. Context addressing, threshold searching and so on can be used to select one or more nodes of a network. Then tokens can be transferred without regard to pointer names from these nodes in n layers, one layer per revolution, to mark a subgraph containing the selected nodes and all nodes up to n arcs distant from the node. The records so marked can then be paged into a parallel computer for analysis by a deep theorem proving program. Semantic paging should effectively filter the data to a small size subgraph that is manageable in a parallel computer, so that it can thoroughly analyze the subgraph at high speed.

2.7 Other Hardware Functions

The disc "microprocessor" will also collect garbage words by a hardware mechanism that operates concurrently with instructions that are evaluating a query. Also, data can be input and marked words can be output while instructions are processing a query. (Interlocks will be provided so that inputs or outputs from one query are not mixed with those from another.) Character string to code word translation is carried out automatically upon input and code word to character string translation is automatically carried out on output. Finally, disc processor instructions are to be stored on and fetched from the disc itself to manage "demons".

The disc is capable of storing a large number of "demons" by storing data words and instruction words in records. Data words are searched by context or by token transfer to activate instruction words in the records satisfying a query. The activated instructions are executed on the disc one at a time as they are deactivated. These concurrent hardware functions increase the performance of the intelligent disc, and make possible some new and possibly revolutionary software techniques.

3. Parallel Computation in Automatic Theorem Proving

There are several ways in which parallel computation might speed up an automatic prover.

A. Evaluating an And-node.

Whenever the prover is asked to prove a subgoal like

$$\forall x(P(x) \wedge Q(x))$$

one processor can be asked to prove $P(x)$ (ie., to find a value or values, for x that will satisfy this formula), and another processor can be asked to prove $Q(x)$. The answers from these two would then be reconciled (if possible) to obtain a common value (or values) or x satisfying by $A(x)$ and $B(x)$.

B. Evaluating an Or-node.

Whenever the prover is proving an or-node of the form $A \vee B$, or is trying a list of possible strategies to obtain the proof of a given subgoal, a separate processor can be assigned to work each of A and B, or each of the strategies. The subgoal would be satisfied when one of these processors succeeded.

C. Simplification and Reduction.

Much of modern theorem proving involves rewriting a formula into a canonical form. For example, the formula (1 + y - 5 + x) might be rewritten as (x + y - 4), or the formula ($x \varepsilon A_{\cap} B$) might be rewritten as (($x \varepsilon A$)$_{\wedge}$ ($x \varepsilon A$)). Parallel processors could greatly speed up this kind of process.

Many of these examples of parallel computation can be handled by an extension of LISP which would allow a parallel COND. That is for the command

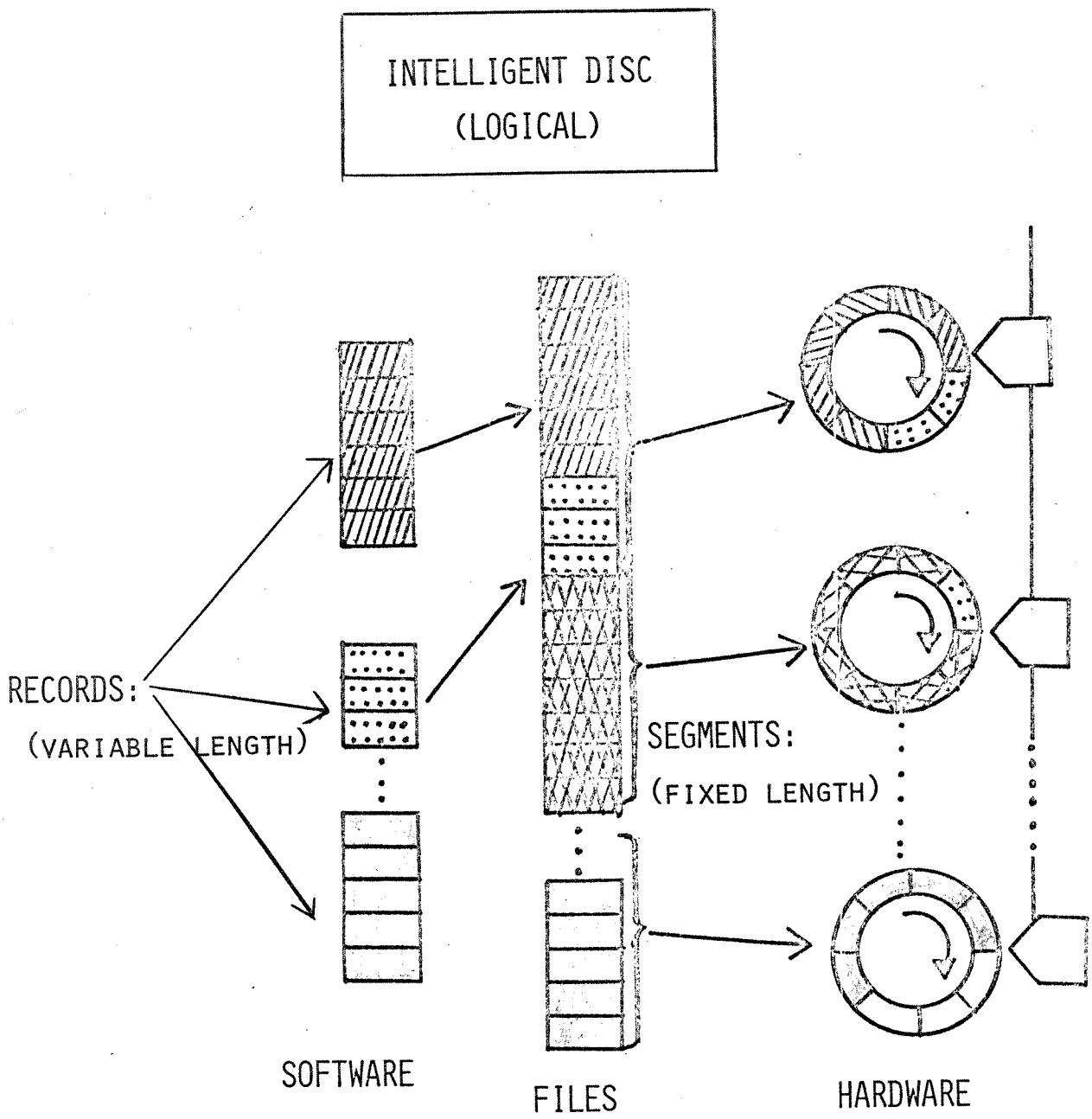
```
(COND
    ( P  A )
    ( Q  B )
    ( R  C ) ),
```

it would simultaneously calculate P, Q, and R, determine which was true, and return accordingly A, B, or C (or some function of them if more than one of P, Q, and R was true).

FIGURES

The figures are selected from the enclosed view graphs. Numbers are shown on the bottom left of each view graph.

Figure	Number of View Graph
1	7
2a	13b
2b	13a
3a	21a
3b	21b
4	17
5	22


INTELLIGENT DISC
(LOGICAL)
RECORDS:
(VARIABLE LENGTH)
SEGMENTS:
(FIXED LENGTH)
SOFTWARE
FILES
HARDWARE

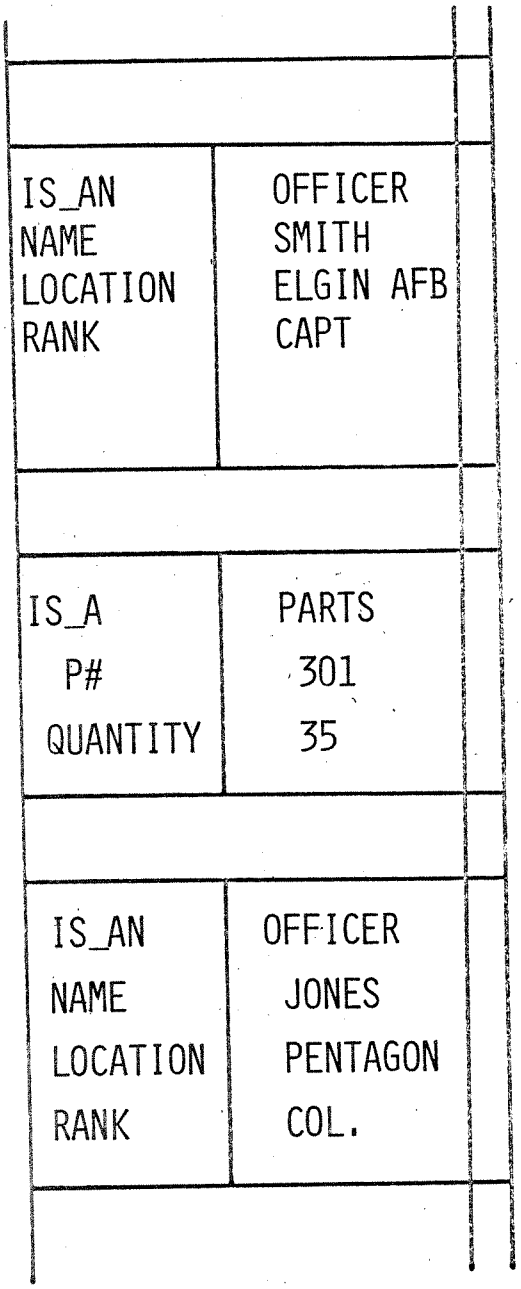
OFFICER

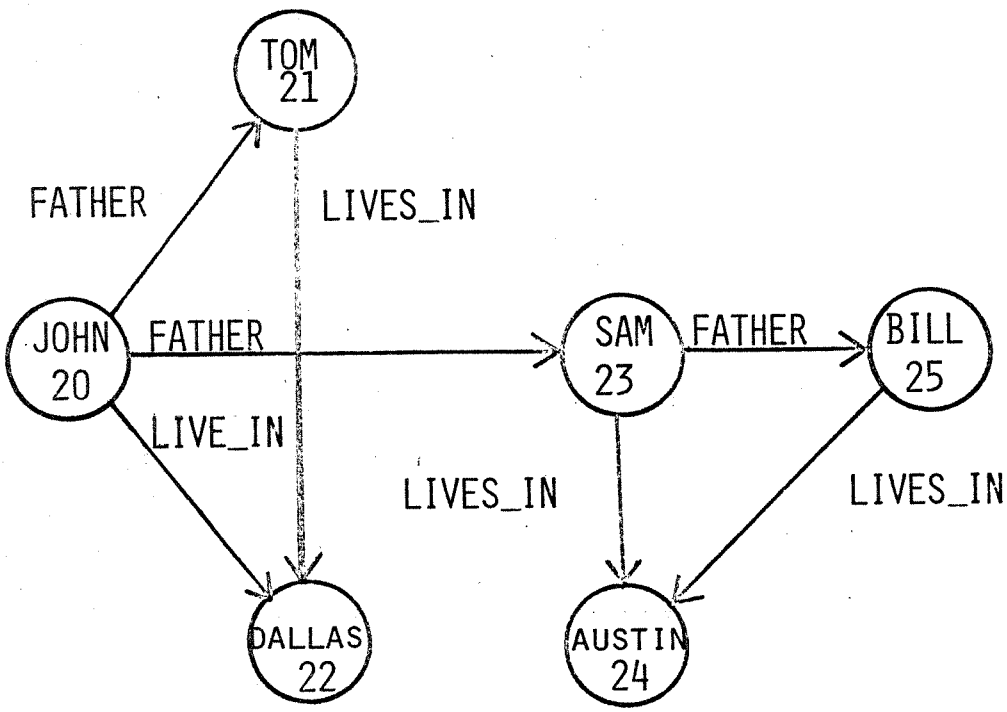
NAME	LOCATION	RANK
SMITH	ELGIN AFB	CAPT.
JONES	PENTAGON	COL.

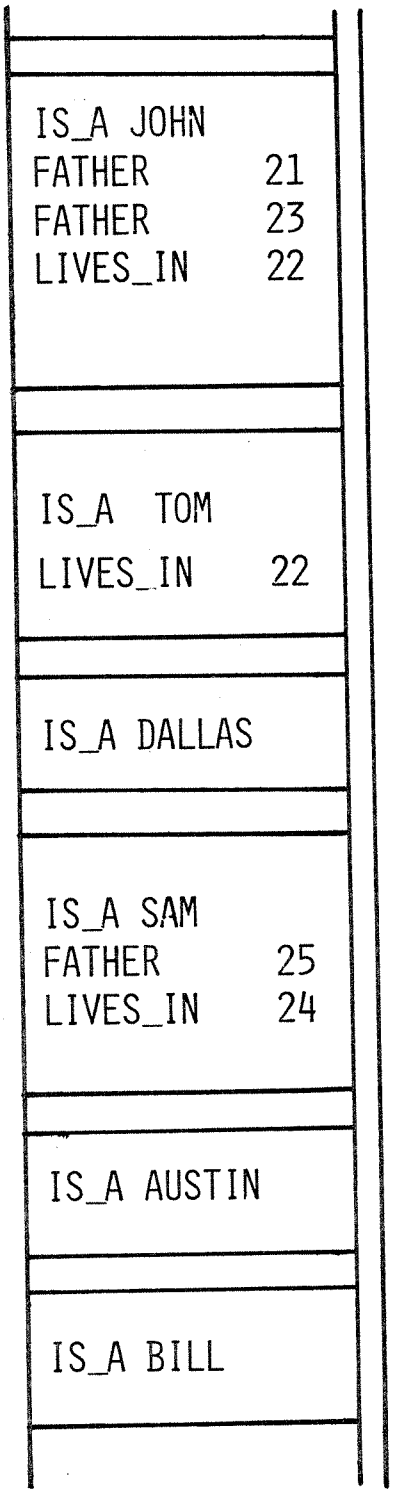
PARTS

P#	QUANTITY
301	35

TUPLES ARE STORED AS RECORDS


IS_AN
NAME
LOCATION
RANK
OFFICER
SMITH
ELGIN AFB
CAPT
IS_A
P#
QUANTITY
PARTS
301
35
IS_AN
NAME
LOCATION
RANK
OFFICER
JONES
PENTAGON
COL.

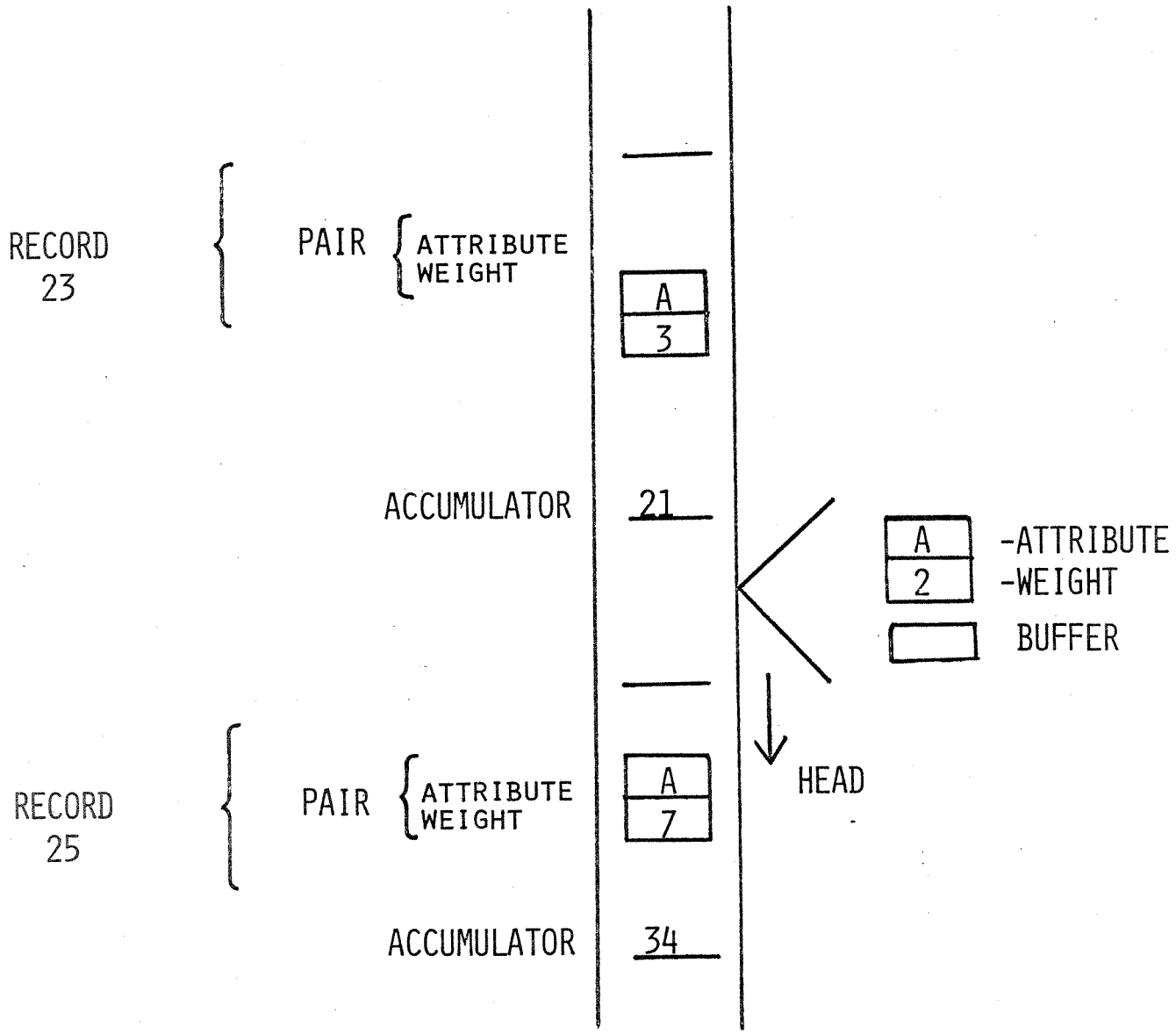

TOM
21
FATHER
LIVES_IN
JOHN
20
FATHER
SAM
23
FATHER
BILL
25
LIVE_IN
LIVES_IN
LIVES_IN
DALLAS
22
AUSTIN
24



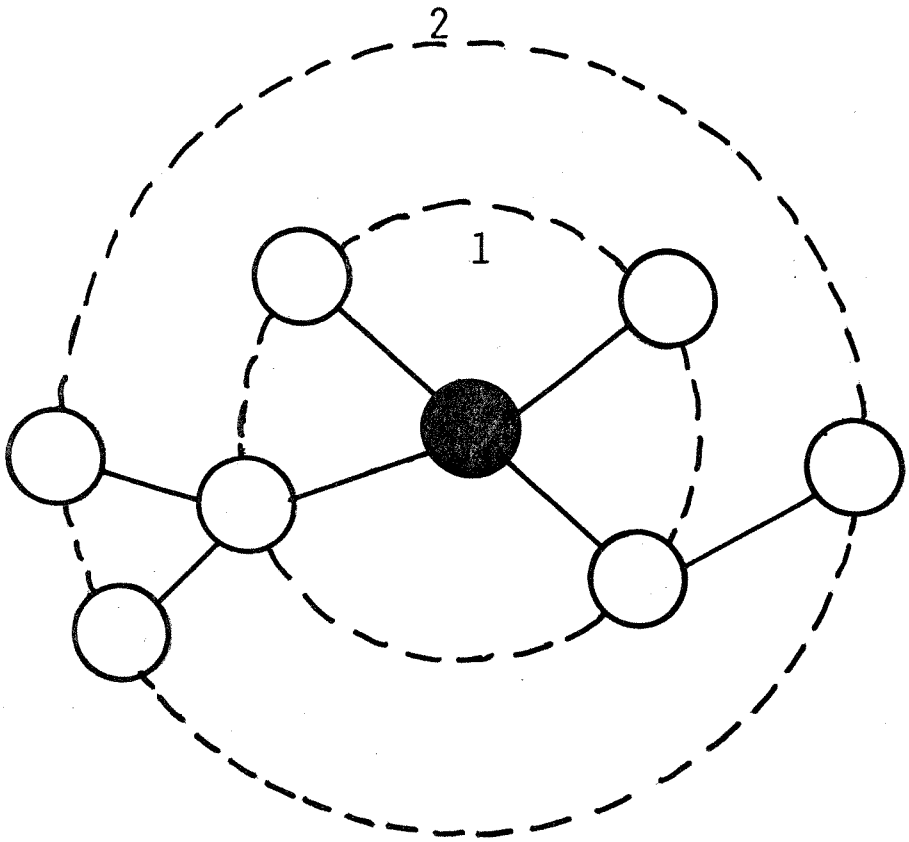
NODES ARE STORED AT RECORDS.
ARCS ARE STORED AS RECORD NUMBERS.

THRESHOLD FUNCTION

SEARCH



SEMANTIC PAGING



1) SELECT NODE(S) BY CONTENT OR CONTEXT.

2) TRANSFER TOKENS OUT THROUGH ARCS N TIMES.

3) OUTPUT ALL NODES WHICH RECEIVED TOKENS.