17

A SYMBIOTIC THEORY FORMATION SYSTEM

by

John Seely Brown

TECHNICAL REPORT #17 - May 1972

FORWARD

This report describes a computer system built by the author in 1968-69. During the summer of 1971 additional experiments were run of this system and the following document was then prepared. It constitutes the substantive portions of the author's dissertation submitted to the University of Michigan.

Although this research is described from the viewpoint of theory formation pertinent to data analysis, it may also be construed as a beginning attempt to understand some of the problems inherent in systems capable of forming a structural model of the world.

DEDICATION

This thesis is dedicated to Dr. Richard Hamming who first convinced me it was worth the trouble and then proceeded to so incessantly harrass me that I had no alternative but to finish its writing.

ACKNOWLEDGEMENTS

I would like to thank the members of my doctoral committee for their kind cooperation and guidance throughout the writing of this dissertation. In particular I am deeply grateful to Professor Robert K. Lindsay who suggested the central problem of this thesis and to Professor Walter Reitman who provided constant encouragement during the darker periods of this investigation.

I am also grateful to various individuals who have patiently listened to my assorted problems and have given me numerous insights. I would especially like to thank Professor William Ash, Professor Richard Karp, and Professor Raymond Reiter.

There are three individuals to whom I feel especially indebted for their valuable help. The first is Peter Headly whose unique philosophy on how to utilize mini computers enabled me to implement this system on a PDP-9. Secondly, to Frank Zdybel who provided all the editing and typing of this thesis. Lastly, to Professor Robert Bobrow whose unique grasp on the problems inherent in induction provided me constant help in formulating many of the ideas expressed herein.

Finally I would like to thank Loren C. French for his help in the preparation of many of the illustrations contained in this thesis.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF APPENDICES

CHAPTER ONE

Introduction

The interaction between the disciplines of data analysis and computer science has usually revolved around the use of computers to perform complex and lengthy numeric computations. As non-numeric uses of computers steadily develop, a more encompassing concept of data analysis emerges. This concept has much the character of a two-sided coin. On one side we find the traditional view in which data analysis primarily concerns techniques of paradigm or theory verification. On the obverse we find the view in which data analysis involves the more ambitious notion of theory formation.

Little research has been directed toward the use of the computer for automatic theory formation, at least partially because there is little agreement as to the meanings of the term "theory" and such related concepts as "model" and "structure". However, advances in such areas as question answering systems, theorem-proving systems and computational semantics are providing some new and interesting paradigms for theory formation and its uses as well as a more general understanding of the related abstract concepts. The notion of capturing man's knowledge in terms of discrete theories and models becomes more appealing as theory-based question answering systems become more powerful.

In this thesis we explore some of the problems surrounding automatic theory formation. In particular, we describe a symbiotic man-machine system (called "the Monkey's Uncle" (1)) which is capable of forming a limited class of theories and discovering the structure of certain collections of "facts".

There are at least two prior studies which illustrate the notions of "theory" and "structure" within the context of automatic theory formation. Amarel (2) has made an extensive study of automatic theory formation as applied to the problem of finding the structure of a special class of transformations defined on a lattice X. These transformations were mappings from X*X onto the given lattice X. The task for his theory formation system was then to take some examples of a particular transformation T (i.e. triples of the form <a,b,c> such that T(a,b) = c) and produce a program which would compute the transformation for the entire X*X set. The resulting program may be considered an intensional representation of the transformation, whereas the examples would be elements from its extensional representation. Amarel considered the movement from extensional to intensional representations as a pivotal challenge for theory formation. We shall see later that this notion captures some of the properties we wish to consider in a more general treatment of theory formation.

The other prime example of theory formation is Simon and Kotovsky's (3) model of sequence extrapolation. This classic study is concerned with discovering the pattern of an initial segment of some given infinite sequence so as to permit the generation of sequence elements successive to the sample. The data operated on by their system is the initially given segment and a set of successor functions; the "theory" which it unfolds is that rule which most "simply" generates the entire sequence. This rule specifies a periodicity for the sequence and reveals how neighboring elements are related in terms of the given successor functions. Discovering such a rule is essentially determining the structural properties of the initial segment.

In the above studies, and especially in Simon and Kotovsky's, the problems of abstraction, generalization and theory formation are closely interwoven; any system which attempts to construct a representation of the structure of a domain of data by having access to only part of the data domain must face the problems of induction and generalization. One of the reasons "the Monkey's Uncle" is a symbiotic system is precisely because we wish to leave open to the user the choice of deciding what "rules" are simplest and hence most general. The object for our system is to reveal as many rules as possible. The user then decides which rules reflect the structural properties he is most interested in.

We must formulate a consistent representation for our "data" or "observations" which will handle a wide variety of situations and yet be not so general as to prohibit the efficient application of combinatorial methods. A schema which appears to meet these criteria is to view a data item or fact as being a triple of the form (x, R, y) where x and y are objects from the domain under consideration and R is a binary relation connecting them. Examples of data which fit this framework are plentiful. R, for example, may be considered an attribute, x an object and y a value, as in: (car, Color, red); or, R might be directly interpreted as a binary relation such as (chair, Right(of), tables), (Max, Father(of), John), (Henry, Likes, Mary), (wheel, Part (of), truck) and so on.

Triples are also capable of representing complex "facts" which are composed of simpler entities. For example, a representation of: "John believes Mary likes Bob" is the complex triple: (John, Believes, (Mary, Likes, Bob)). Although permitting facts to be embedded inside other facts would enhance the generality of our system, we have purposely refrained from this practice. By so doing we have guaranteed that the arguments of a relation may be viewed as objects which have no finer structure. This permits a simple representation of the data as a directed graph whose nodes denote only the objects or arguments of a relation. The directed arcs connecting the nodes are tagged with the names of the relations pertaining between the respective nodes. Such a directed graph provides an efficient representation for combinatorial search procedures which discover all the possible paths between any two objects. Such searches are fundamental to our approach to theory formation.

Before launching into a discussion of our theory formation system, we wish to give an example which will show how and why such a system might be used.

Suppose we are presented with the collection of data shown in Table 1A which consists of thirty-two facts about the three relations R, L, and A defined over a domain of six objects. The task before us is to discover as much of the structure underlying these facts as possible. Ideally, we would like to find sufficient regularities in this data to allow an extremely compact characterization of the original domain. In the final analysis, this compact representation will constitute our theory.

TABLE 1A
Extensional Definitions of the Relations

(b, R, a)	(e, R, f)	(e, L, c)
(c, R, a)	(a, L, b)	(e, L, d)
(c, R, b)	(a, L, c)	(b, A, e)
(c, R, e)	(a, L, d)	(e, A, b)
(d, R, a)	(a, L, e)	(b, A, b)
(d, R, b)	(a, L, f)	(e, A, e)
(d, R, c)	(b, L, c)	(d, A, f)
(d, R, e)	(b, L, d)	(f, A, d)
(f, R, a)	(b, L, e)	(d, A, d)
(f, R, b)	(c, L, d)	(f, A, f)
(f, R, c)	(c, L, f)	(a, A, a)
(b, A, b)	(c, A, c)	(d, A, d)
(e, A, e)	(f, A, f)	

The triples contained in Table 1A can be used to construct the directed graph shown in Figure 1.1. Toward finding some regularities in this network we might first separate this graph into three sub-graphs, each pertaining to one of the given relations. This factoring is represented in Figure 1.2. Examining each graph by itself we can discover, by exhaustive and tedious inspection, that:

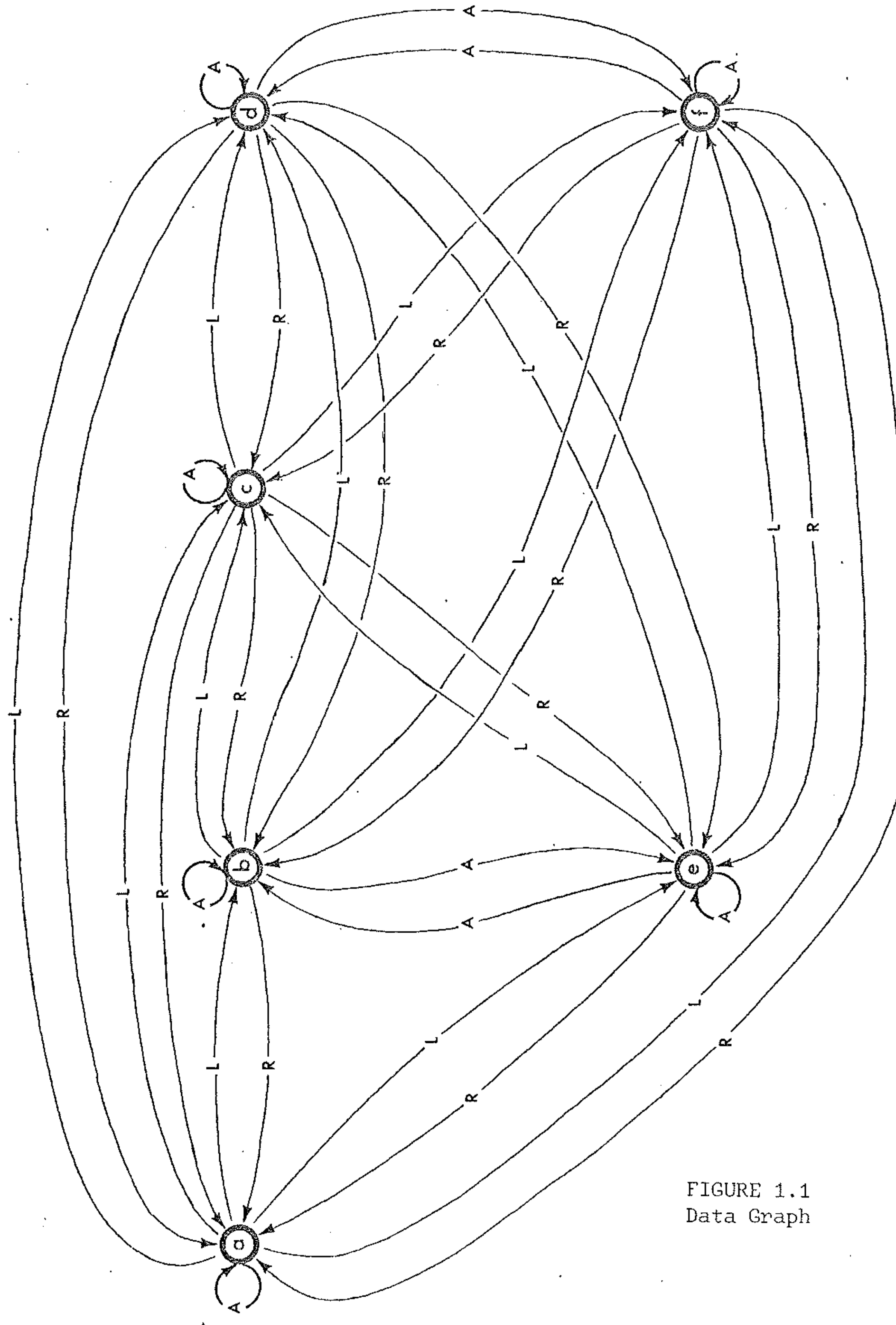
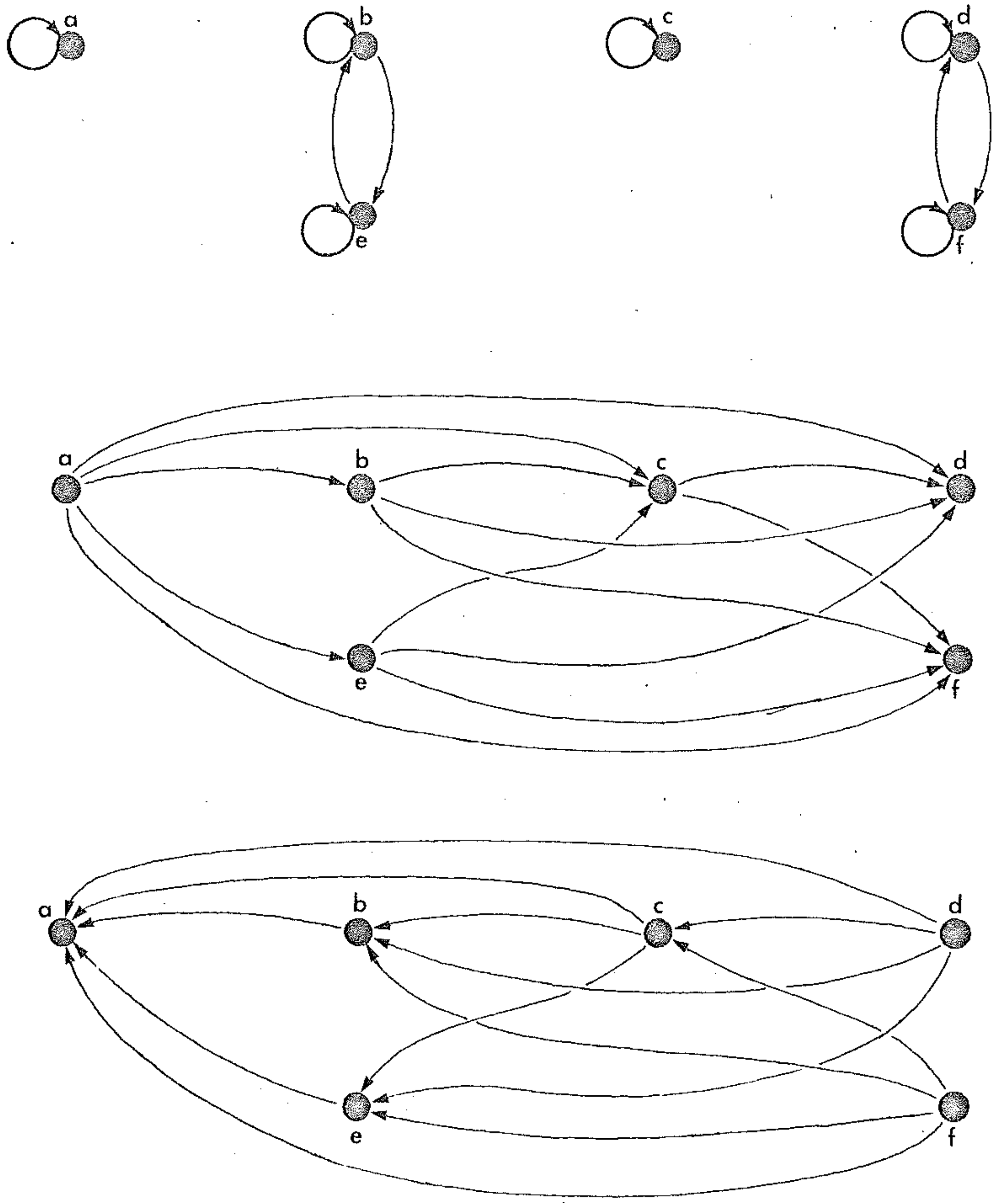


FIGURE 1.1
Data Graph

FIGURE 1.2
Decomposition of Data Graph



1) R is transitive, anti-symmetric and anti-reflexive
2) L is transitive, anti-symmetric and anti-relexive
3) A is transitive, symmetric and reflexive.

Knowing just the above we could drastically simplify the data by omitting triples which could be reconstructed given rules 1, 2 and 3 and some basic set of triples. There are still other types of regularities to be investigated, however. For example, we might wonder if any one of these relations could be completely defined in terms of the others. If so, then the triples constituting that relation could be eliminated and the final model need only specify how this relation could be reconstructed in terms of its defining relations.

We begin our search by looking for relationships between the factored sub-graphs. A comparison between the subgraphs for R and L reveals a striking similarity. Relation R is the converse of relation L. In other words, (x, R, y) is true if and only if (y, L, x) is true.

4) R = *L (where * means converse).

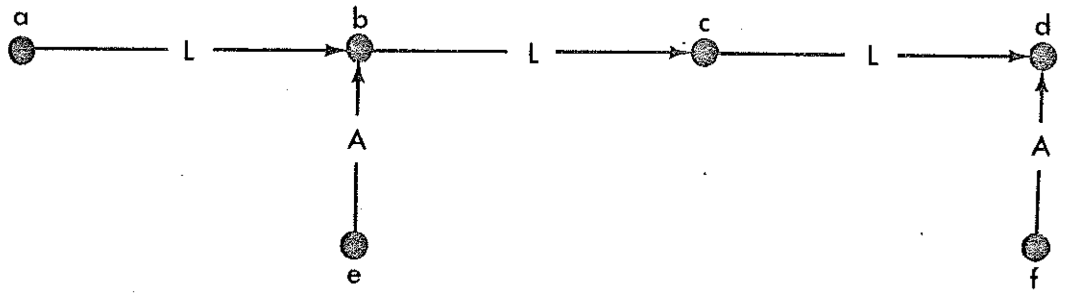
There still remain unexplored regularities in the interactions among these relations. These are the most difficult and tedious to detect because, unlike rule 4 above, these regularities need not occur over all the triples of a given relation and, unlike rules 1-3, the entire directed graph must be considered. It is doubtful that without mechanical assistance all of the following patterns or regularities would be detected:

For all x,y,z:

5) (x, R, y) and (y, A, z) => (x, R, z)
6) (x, A, y) and (y, R, z) => (x, R, z)
7) (x, L, y) and (y, A, z) => (x, L, y)
8) (x, A, y) and (y, L, z) => (x, L, y)

With the regularities expressed in rules 1-8 we can now produce the drastic simplification of this data represented by the directed graph of Figure 1.3.

FIGURE 1.3
Kernel of Data Graph



The five triples underlying this graph, in conjunction with the above eight rules, constitute our theory. Any fact contained in Table 1A not explicitly mentioned in our theory may be inferred by formulating these rules as axioms in a predicate logic and then utilizing a theorem-proving system.

We have thus simplified the original data sufficiently that the semantics of these three relations can probably be guessed. The relation R means "Right of", L means "Left of" and A means "At". These meanings might have been suspected from the arrangement of the nodes in Figures 1.1 and 1.2. If the nodes in these graphs had been randomly arranged, the resulting diagrams would probably have been confusing enough to obscure this interpretation. In Figure 1.3, however, the directed graph is so simple that any arrangement would still suggest that these relations denote some type of ordering.

Although this example is based on very simple data, it affords a feeling for the amount of searching, guessing and matching that is involved in unfolding just these simple patterns. In this thesis we want to examine how computers can be used to facilitate these tasks.

It might be worth noting that the notion of "theory" sketched out in this discussion has a strict logical counterpart. The above eight regularities may be considered as axioms involving individual variables, whereas the selected triples (representing the core of the compressed data) are axioms involving individual constants. The extensions of the relations as defined by Table 1 A then represent a logical or semantic model for the axiomatic theory.

When we commenced this project it was our idealized and naive goal to construct an automatic theory formation system which could examine a massive collection of data and automatically formulate a theory in the form of a set of axioms for a question answering system. As anyone who has constructed a question answering system can testify, such an axiom formation system would be a tremendous aid. Likewise, however, the same people will probably testify to the hopelessness of the task.

What follows, then, is an attempt to formulate a precise set of goals for a symbiotic theory formation system. These goals are discussed at length in Chapter Two; in Chapter Three we present the general framework for a system which can achieve them. Chapter Four deals with the details of our system's implementation; chapter Five contains various examples of it in operation. In Chapter Six we describe some combinatorial techniques which help to identify those relations of a data base which are most fundamental. The relations thus isolated represent a "core" of the initial data. It is our hope that the resulting representation will be so simple (as is Figure 1.3 above) that the user will be able to discover from it the underlying semantics of his data. In Chapter Seven, we exhibit a simplified question-answering system which is capable of utilizing the rules or patterns discovered by our main system and which, in addition, provides a means of checking the completeness of our derived theories. In the final chapter of this thesis we will discuss the possibility of using our symbiotic theory formation system to detect and correct errors in the data.

Although we present our research from the viewpoint of theory formation, much of this research is relevant to the far-reaching goal of a system which can invent (or select) its own inferencing procedures and thus capitalize on "special" properties of the given universe. In fact, we are greatly indebted to Robert K. Lindsay who, for reasons more closely tied to the above goal, conceived of the central problem discussed in this thesis. We believe that his insightful delineation of this class of problems was critical in our construction of an inductive system that can perform useful tasks. However, this is certainly just a first step toward the more ambitious goal mentioned above.

FOOTNOTES

(1) So called because the system spends much of its time climbing up and down trees; and because it produced its first noteworthy result during an exploration of the "Uncle" relation.

(2) S. Amarel, "On the Automatic Formation of a Computer Program which Represents a Theory," Self-Organizing Systems (Washington, D.C.: Spartan Books, 1962).

(3) H. A. Simon and K. Kotovsky, "Human Acquisition of Concepts for Sequential Patterns," Psychological Review, LXX (No. 6 1963), pp. 534-546.

CHAPTER TWO

Goals for a Theory Formation System

In this chapter we will delineate four quite different tasks and discuss the motivation behind each. These tasks will include the discovery of intensional definitions and the selection of the relations most fundamental to these; the discovery of rules of inference and the differentiation of objects in the data space according to properties revealed by the foregoing.

As mentioned earlier, the data for our system is presented in the form of binary relations. A binary relation R on a universe D of objects is extensionally defined by specifying a subset of 2-tuples from D*D. A particular "fact" or element of our data is a triple of the form (x, R, y) where the 2-tuple (x,y) is contained in the extension of R. The data file thus consists of a list of facts which represents a set of observations about some "world".

Discovery of Intensional Definitions

A relation can also be described intensionally without explicit reference to its extension. This can be done by stating its logical properties or by specifying how it can be defined in terms of some set of relations. In this manner, we move from extensional definitions of our relations to intensional ones. The "logical combinations" comprising these intensional definitions will rely on the logical operators: composition, disjunction and converse.

Def: The <u>Composition</u> of R1 and R2, denoted by "R1/R2" is defined:

For all x and y:
$[(x,R1/R2,y) \text{ iff } (\exists z)((x,R1,z) \wedge (z,R2,y))]$.

Def: The <u>Disjunction</u> of R1 and R2, denoted by "R1 V R2" is defined:

For all x and y:
$[(x,R1 \vee R2,y) \text{ iff } ((x,R1,y) \text{ or } (x,R2,y))]$.

Def: the <u>Converse</u> of R, denoted *R, is defined:

For all x and y:
$[(x,{*R},y) \text{ iff } (y,R,x)]$.

Def: A relation Ri is <u>intensionally</u> defined with respect to some given set of relations R1,R2,...,Rn by specifying how it can be constructed out of R1,R2,...,Rn using only the logical operators of composition, disjunction and converse.

In order to aid the appreciation of the concept of intensional definition we will mention two aspects of such definitions. First, intensional definitions are a certain type of specification of the interrelationships among relations. In fact, such definitions assert that these interrelationships are precisely characterizable by another relation. Secondly, an intensional definition can be a recursive or circular definition in that the given relation might interact with other relations so as to yield a logical definition of itself. We can, for example, define the "Parent" relation as:

Parent = Spouse/Parent

i.e., x is the parent of y if there exists a z such that x is the spouse of z and z is the parent of y.

The set of definitions for all the relations on a domain may indirectly form circular definitions even if no relation is defined in terms of itself. This is often the case with natural language dictionaries, where a chain of definitions may eventually lead one back to the original word in question (this is not to imply that such definitions do not convey information). Some examples involving binary relations might be:

```
Example 1:
    Def: Cousin = Offspring/Uncle
    Def: Uncle = Parent/Cousin

Example 2:
    Def: Uncle = Husband/Aunt
    Def: Aunt = Mother/Cousin
    Def: Cousin = Offspring/Uncle
```

We now define the first of our four tasks.

Task #1: Given the extensions of several binary relations over some universe D, discover all the intensional definitions of each relation in terms of the given collection of relations.

If we can discover <u>all</u> of the interrelationships among these relations which are expressible as intensional definitions, we are a long way toward representing the logical structure of the data base.

This point can be enhanced by realizing that a particular relation often has many intensional definitions, all of which must be equivalent in some sense since the extensions of each of these definitions are identical. The user, however, must decide if the definitions are logically equivalent. Of the many structural relationships presented to him by the system, some will represent important logical properties of the underlying universe from which the data was drawn while others will merely reflect idiosyncracies of the data sample. Suppose that our system has unfolded, for some given sample, two definitions of the term "Uncle". One of these asserts that an uncle is a parent of a cousin, while the other asserts that an uncle is either the brother of a parent or the husband of a sister of a parent. For the given sample, the extensions of these two definitions must concur. We see, however, that the first definition requires that all uncles be parents. It is up to the user to decide whether this reflects a valid structural property of the data universe or whether it merely stems from an idiosyncracy of the sample data. If the latter is the case, enlargement of the sample will probably disprove the definition.

Apart from revealing some of the interplay among the various relations, the discovery of intensional definitions for a given relation can allow a significant reduction in storage space requirements. This is because the extension of a relation will often involve large tables of information while its intensional definition will rarely require more than a few symbols. Of course, since intensional definitions do not contain references to objects, the original collection of facts about relations between objects cannot be entirely replaced by a collection of intensional definitions. This leads us to the problem of deciding which relations should be represented intensionally and which should be maintained extensionally. It seems reasonable that some of the relations are apt to be more basic than others. If the correct choice is made, a greater compression of the data can be achieved. In fact, many of the virtues of such systems as LEAP (1) and TRAMP (2) are derived from this kind of data compression.

Discovery of "Atomic" Relations

Our second task is to decide which relations are so basic they might be justifiably called atomic. Since our system doesn't know the meanings of the relations, atomicity cannot be based on real-world semantics and must be characterized solely in terms of the logical or syntactic properties of the particular binary relations. Still, any abstract definition of atomicity must fulfill our intuitive expectations. The following recursive definition adequately formalizes the notion:

Def: Relative to a specified set of intensional definitions, a set of relations R generates a particular relation Ri if either:

i) The set R contains all the defining relations in an intensional definition of Ri

or ii) The set R can generate all of Ri's defining relations.

Def: A subset R' of R is an atomic set of relations relative to a specified set of intensional definitions if R' can generate all relations in R and if, for any other set R" which does likewise, $|R'| \leq |R''|$.

In other words, R' constitutes an atomic set of relations if R' can generate R and if any other set which does likewise has a number of elements which is greater than or equal to the number of elements in R'.

Atomic sets are, therefore, the smallest sets from which all of the remaining elements can be defined. They are not unique, and, in fact, are critically dependent on the set of intensional definitions. Finding the smallest possible set of atomic relations requires first knowing all possible ways each relation can be defined in terms of the others.

We can now formally state our second task.

Task #2: Given a collection of relations and intensional definitons for each relation in the collection, determine the atomic relations.

Discovering the atomic relations for a data base allows us more than just the opportunity to compress our data by eliminating the extensions of all the non-atomic terms. Once we know which terms are atomic we can afford to delve more deeply into the interactions among just these terms. To make future combinatorial searches more efficient, we can also simplify the original data graph by eliminating all links pertaining to non-atomic terms.

The additional regularities we would like to discover are of two types. First, we hope to isolate patterns that might hold over only parts of the extensions of particular relations. These patterns often lead us to interesting conclusions about our relations. If a relation Ri is transitive, for example, then part of its extension is equal to the entire extension of the composite relation "Ri/Ri".

Secondly, we want to examine all possible pairwise, triplewise etc. interactions among these relations. There are several reasons for this. Suppose we discover that both R1 and R2 are transitive relations and suppose we know that R3 (a non-atomic relation) can be defined as "R3 = R1/R2". If we wish to know whether or not R3 is transitive we can, of course, directly inspect R3 for that property. Even if R3's extension were no longer available, however, we could still answer our question if we confirmed that the pairwise interaction: "R1/R2 = R2/R1" existed (3).

Discovery of Inference Rules

Another reason for studying all possible n-wise interactions concerns the discovery of rules of inference. Finding such rules is of extreme interest since, as the name of these rules suggests, they permit us to infer additional facts from those already known. For example, suppose we know the following two facts:

Fact 1: (Napoleon, Command, French Army)

Note that Ri is permitted to occur in the left hand side of the rule.

From a logical point of view, this restriction amounts to requiring that our rules of inference involve only universal quantifiers.

Discovery of Predicates

The above three tasks have all involved finding various types of interrelationships among the given relations. No mention has been made about the objects on which these relations are extensionally defined. It seems that any interesting theory formation system ought to be able to use the relational structure it has discovered to establish certain structural properties of the objects themselves.

This task is fundamentally different from the previous tasks because it requires the invention of descriptive terms for the objects. The prior tasks worked entirely within the framework of the given relations, finding relationships that could be expressed in terms of already defined concepts. Here, however, concepts in the form of predicates must be invented.

We are interested in the evolution of several types of descriptions for the objects. The simplest descriptions will be unary predicates which assert that an object either has a particular property or that it doesn't. The more complex descriptions will assert that some attribute of an object has one of a set of possible values. If that set has but two values then, of course, these two types of descriptions coincide. The critical idea in the generation of such predicates and attributes is the realization that they implicitly specify groupings of the objects. More precisely, both unary predicates and attributes define partitions on the object space. What we must do is to somehow use the structure discovered in the above three tasks to direct the formation of partition classes on the object space. This operation is, in fact, our last task:

Task #4: Determine partitions on the object space which reveal interesting properties about the objects.

In this chapter we have set forth four quite disparate problems which we expect our symbiotic theory formation system to solve. All of these tasks involve, in one way or another, the isolation or discovery of the structural properties of a given data set. The first task concerns unfolding those simple interrelationships which lead to intensional definitions. The second task is to select the most fundamental relationships and thereby provide a limited field upon which to focus more exhaustive inspection. The third task is to discover interrelationships among the relations which qualify as rules of inference. The last task is the creation of new relations - which often take the form of unary predicates - which reveal qualities of the objects themselves. The result of these four tasks should be a clearer understanding of the structure of our data.

FOOTNOTES

(1) Jerome A. Feldman and P. D. Rovner, "An Algol-Based Associative Language," Communications of the ACM, XIII (August, 1969), 439-449.

(2) William L. Ash and E. H. Sibley, "TRAMP: An Interactive Associative Processor with Deductive Capabilities," Proceedings ACM Nat'l. Conference (August, 1968).

(3) Theorem: If R1, R2 are transitive then R1/R2 is transitive if R1/R2 = R2/R1.

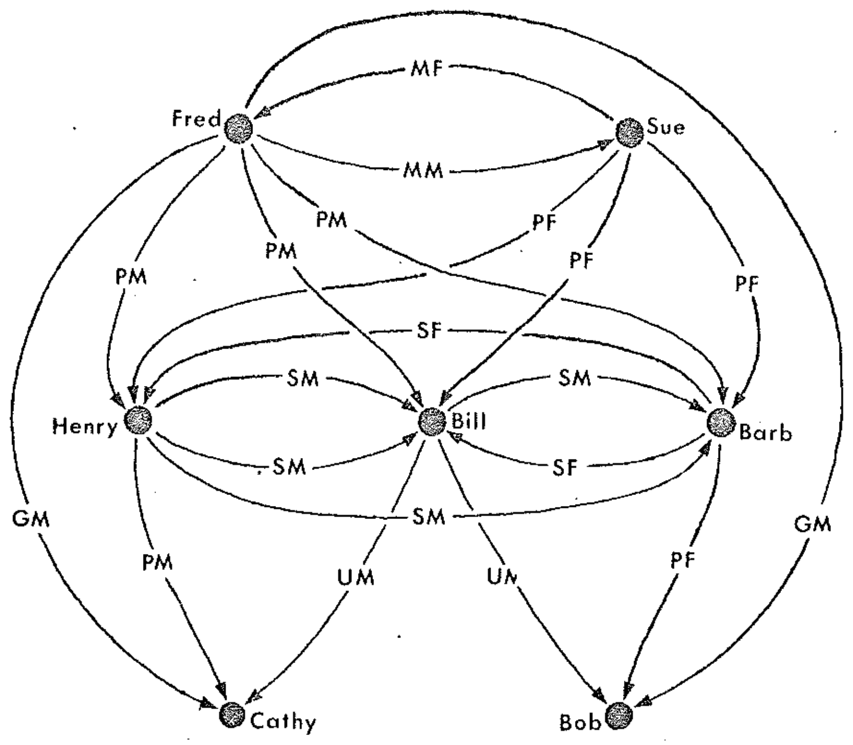
CHAPTER THREE

An Approach to Theory Formation

We have designed and implemented two quite different systems intended for the type of theory formation described in the previous chapters. We now describe some of the theoretical problems involved in their construction and the techniques we have used in their solutions. In this chapter we will concentrate on the theory formation system called "the Monkey's Uncle", relegating to Appendix 1 the description of a prior system which proceded from a purely algebraic formulation of the task. This prior system proved to be overly sensitive to errors in the data. In addition, the algebraic approach became less tenable when the amount of data was increased.

The primary purpose of "the Monkey's Uncle" is to discover in a given data set interrelationships or patterns which can be expressed as intensional definitions of its extensionally defined relations. To illustrate how a pattern could be translated into an intensional definition we consider the following simple case which consists of two subgraphs (see Figure 3.1) from a hypothetical data graph. The arcs of the subgraphs are labelled according to the mnemonics listed in Table 3A.

FIGURE 3.1
Data Graphs



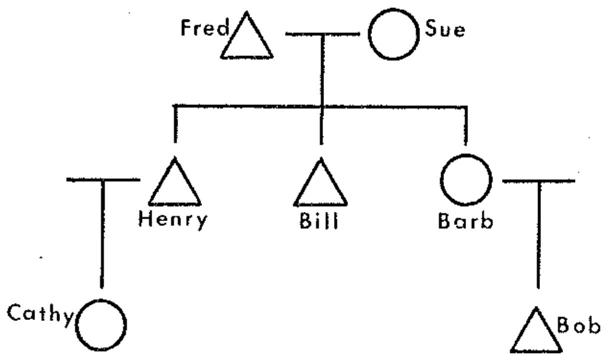
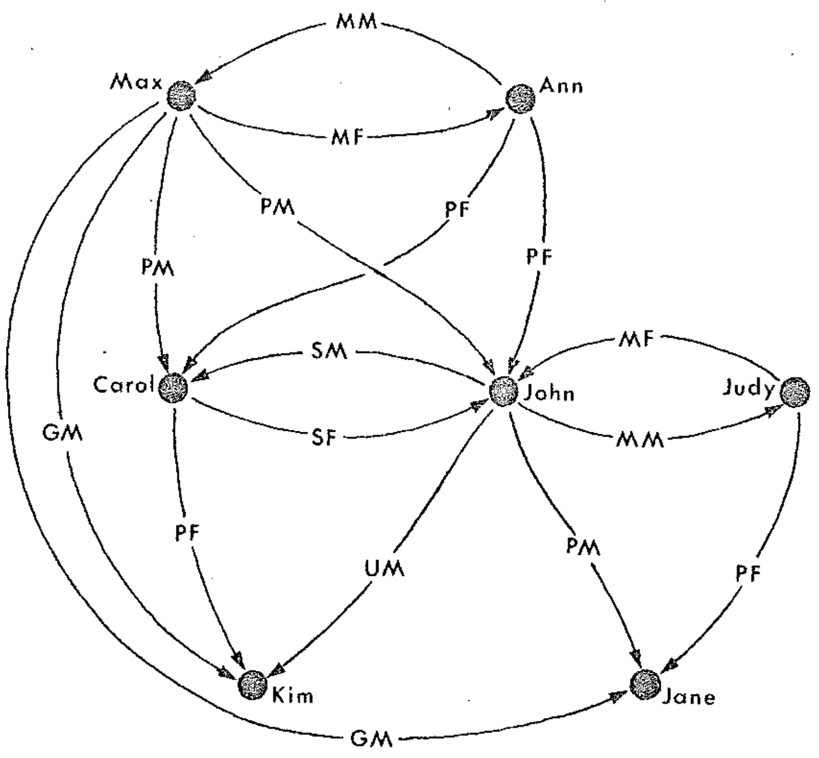


FIGURE 3.1 (cont.)



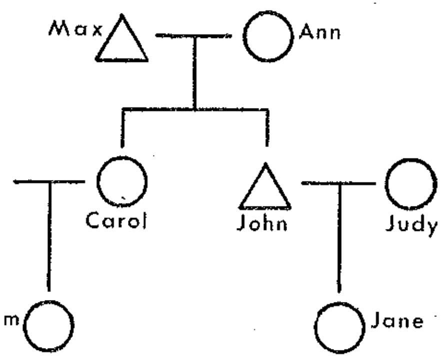


TABLE 3A
Mnemonics for Relations

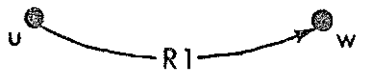
GM	=	Grandfather	GF	=	Grandmother
PM	=	Father	PF	=	Mother
MM	=	Husband	MF	=	Wife
SM	=	Brother	SF	=	Sister
UM	=	Uncle	PN	=	Parent (neuter)

TABLE 3B
Labelled Path Sequences

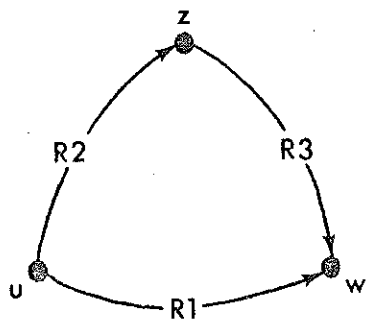
(FRED GM BOB)	(MAX GM JANE)	(FRED GM CATHY)
PM-PF	PM-PM	PM-PM
PM-UM	MM-PF-PM	PM-UM
MM-PF-PF	PM-MM-PF	MM-PF-PM
PM-SM-PF	PM-SF-PM	MM-PF-UM
MM-PF-UM	PM-SF-MM-PF	PM-SF-UM
PM-SF-UM	MM-PF-SF-PM	PM-SF-PM
PM-SM-UM	MM-PF-MM-PF	PM-SM-PM
MM-PF-SF-UM	MM-PF-SF-MM-PF	PM-SM-SF-UM
MM-PF-SM-PF		MM-PF-SF-UM
MM-PF-SM-UM		MM-PF-SF-PM
PM-SF-SM-UM	(MAX GM KIM)	MM-PF-SM-PM
PM-SM-SF-UM		MM-PF-SM-UM
PM-SM-SF-UM	PM-PF	PM-SF-SM-PM
PM-SM-SM-PF	PM-UM	PM-SF-SM-UM
MM-PF-SF-SM-UM	MM-PF-PF	PM-SM-SF-PM
MM-PF-SM-SF-UM	PM-SM-PF	PM-SM-SF-UM
	PM-SF-UM	MM-PF-SF-SM-PM
	MM-PF-UM	MM-PF-SF-SM-UM
	MM-PF-SM-PF	MM-PF-SM-SF-PM
	MM-PF-SF-UM	MM-PF-SM-SF-UM

* Note that the second letter of each mnemonic represents the sex (M = Male, F = Female, N = Neuter) and the first represents the generic category (G = Grandparent, M = Mate, S = Sibling, etc.). This notation was conceived by R. K. Lindsay (personal communication).

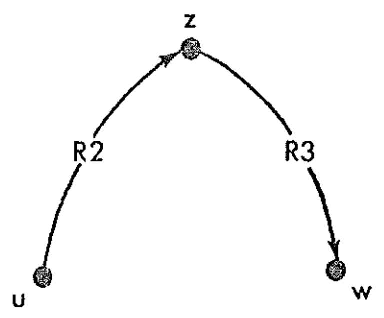
In order to understand the crucial idea behind this example, we must first recall the definition for the composition of two binary relations. The definition first asserts that if R1 = R2/R3 then for any (x,y) contained in the extension of R1 there exists an object z such that (x,z) is contained in R2 and (z,y) is contained in R3. Restated in terms of the directed graph representation of our data, this means that, if R1 = R2/R3, then for every arc:



there is a path which has the form:



<u>In addition,</u> the definition asserts that, if R1 = R2/R3, then for all z such that (u,z) is contained in R2 and (z,w) is contained in R3, (u,w) must be contained in R1. Reinterpreted, this means that for every path of the form:



there must also exist a simple arc from u to w labelled R1.

Returning to our example, we want to examine how the intensional definitions of "Grandfather" ("GM") might be discovered. Toward this goal, we select a fact, say (Fred, GM, Cathy) and search out all possible paths that bridge this fact but which do not involve loops. For each path discovered we record only the labels on its component arcs and not the nodes it passes through. For this particular 2-tuple these labelled path sequences (hence forth to be called LPS's) are listed in Table 3B under (Fred, GM, Cathy).

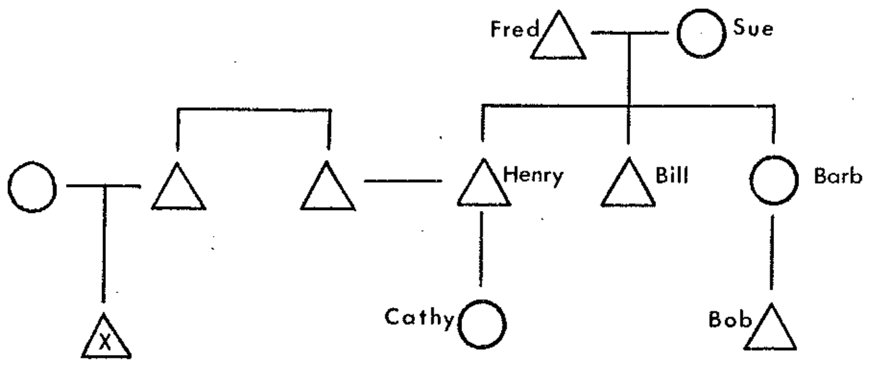
It is possible that some of these twenty LPS's represent valid compositional definitions for the relation "GM". In order to discover if a particular LPS does so, we must verify that every occurrence of this pattern in our data graph bridges a 2-tuple contained in the extension of the "Grandfather" relation. Since this verification is a costly operation (requiring a pattern search over the entire data space as contrasted with an anchored search from the 2-tuples of "GM"), it makes sense to gather supporting evidence from other "Grandfather" examples. Hence, we might consider all the paths bridging another example of Grandfatherhood - i.e. (Max, GM, Jane). Any LPS that reflects a valid compositional definition must be in both sets of path labels. We therefore form the intersect of these two sets and find:

```
                                                 PM/PM
(Fred,  GM, Cathy) and (Max, GM, Jane)  =>   PM/SF/PM
                                                 MM/PF/PM
                                                 MM/PF/SF/PM.
```

This new set is much smaller (four vs. twenty), due to the elimination of such LPS's as "PM-UM". A close examination of the associated conjecture: "GM = PM/UM" illustrates the two ways a conjectured definition might fail.

First, the validity of this definition rests on finding, for every example (x,y) of Grandfatherhood, a son (i.e. a "z") who is an uncle of the particular child ("y"). This could happen if every couple had at least two sons, a case which would probably reflect an idiosyncracy of the sample data. However, if the data base contained examples extracted from the following sample family structure (see Figure 3.2), then Fred would not only be the grandfather of

FIGURE 3.2
Hypothetical Family Tree



Bob but would also be the grandfather of x, which is clearly false. In other words, "PM/UM" is an over-general conjecture in that it generates examples outside of GM's extension. At the same time, it is under-general since there are examples of Grandfatherhood which cannot be made to fit PM-UM patterns.

Let us now delineate two strategies to help in discovering valid compositional definitions:

Strategy #1: Intersect all available lists of LPS's so as to maximize the chance of eliminating any under-general conjectures.

Strategy #2: Test the remaining LPS patterns to see if any of these patterns are over-general.

Strategy #1 has certain complications, as may be seen when a third list of LPS's (i.e. all paths bridging (Max, GM, Kim)) is intersected with the other two lists. This intersection is null, indicating that there is no LPS pattern in common with all three lists. Translated into the logic of relations, this means that the relation "GM" has no intensional definition consisting solely of the compositions of other relations defined on this data base (note the absence of the neuter relation "Parent" in the data graph). Examining this graph we see that a disjunctive definition is required because Max has both a daughter and son who function as mother and father for Kim and Jane respectively. Also, Strategy #1 must be revised to yield something besides "null" in these situations.

Revised Strategy #1:

> Intersect the list of LPS's until either all the lists are intersected or until a null intersection is formed. In the latter case, restart the process with the last list encountered and proceed from there (thus forming another disjunct of the definition).

A close look at the application of the revised strategy in this example shows not only that a disjunctive definition is required but that the order in which we intersect the various lists of LPS's affects the outcome. For example, if we first form the pairwise intersection of the LPS lists belonging to (Max, GM, Jane) and (Fred, GM, Cathy) and then form the pairwise intersection of those belonging to (Max, GM, Kim) and (Fred, GM, Bob) we arrive at the possible definitions shown in Example 1. The first disjunct comes from intersecting the first pair of lists and the second disjunct comes from intersecting the second pair of lists.

EXAMPLE 1

```
     ((MAX,JANE)Λ(FRED,CATHY))    ((MAX,KIM)Λ(FRED,BOB))

                                               PM/PF
              ?        PM/PM                   PM/UM
  GM          =        PM/SF/PM        V       PM/SM/PF
                       MM/PF/PM                PM/SF/UM
                       MM/PF/SF/PM             MM/PF/PF
                                               MM/PF/SM/PF
```

EXAMPLE 2

```
(((MAX,KIM)Λ(FRED,CATHY))Λ(FRED,BOB))    (MAX,JANE)

                                               PM/PM
                                               PM/MM/PF
              ?        PM/UM                   PM/SF/PM
  GM          =        PM/SF/UM        V       MM/PF/PM
                       MM/PF/UM                MM/PF/MM/PF
                                               MM/PF/SF/PM
                                               PM/SF/MM/PF
                                               MM/PF/SF/MF/PF
```

Suppose we had, instead, intersected the LPS lists belonging to (Max, GM, Kim) with those belonging to (Fred, GM, Cathy) and then intersected the resultant list with that of (Fred, GM, Bob). We then discover that this new, non-void list has no LPS patterns in common with those of (Max, GM, Jane). Under this ordering we generate the definitions shown in Example 2.

Although the definitions belonging to both examples are consistent with the given data, the definitions of the first example are preferable; at least some of them are universally consistent in that they can never generate a counter-example to Grandfatherhood, no matter how much we enlarge the data space. In the second example, none of the compositional definitions of the first disjunct are universally consistent.

To illustrate more fully how the end result of list intersection depends critically upon the order in which the pairwise intersections are performed, we have created a more realistic example. Using the simple family tree shown in Figure 3.3, we generated all the extensions of American kinship terms. This data was then input to our system and we created the LPS lists for all the examples of the "Parent" relation. For this data there are sixteen (x,y)

2-tuples in the Parent relation. There are, therefore, fifteen factorial orderings of their associated LPS lists. Of these possible orderings, we randomly generated twenty-three and proceeded to execute Strategy #1 with each. Figure 3.4 summarizes the frequency of occurrence of the resulting disjunctive definitions according to the number of disjuncts they contained. Table 3C lists the intensional definitions produced by: a) the first of our randomly selected orderings and then by: b) the "optimal" ordering of the lists. This "optimal" ordering was the only one of the twenty three that produced only two disjuncts. The first of the randomly selected orderings lead to a definition involving three disjuncts.

TABLE 3C (Part A)
Conjectural Definitions

						GN/NM
						GN/NF
						GN/CN/NM
						GN/CN/NF
				SN/UM		GN/OM/SN
				SN/SN/UM		GN/NM/SN
				SN/ON/GF		GN/NM/SM
	?			SN/OM/GF		GN/SM/NF
PN	=	MN/SN/UM	V	SN/UM/CN	V	GN/ON/SN
				SN/UM/SN		GN/OF/SN
				SN/ON/GN		GN/NF/SN
				SN/ON/GM		GN/NF/SM
				SN/OM/GN		GN/SN/NF
				SN/OM/GM		GN/SN/NM
						GN/SF/NM
						MN/GN/NM
						MN/GN/NF

TABLE 3C (Part B)

	PF		PM
PN $\overset{?}{=}$	PF/SN		PM/SN
	MN/PM		MN/PF
	MF/PM	V	MM/PF
	PF/OF/PM		PM/OF/PF
	MN/PM/SN		MN/PF/SN
	MF/PM/SN		MN/PF/SN

FIGURE 3.3
Hypothetical Family Tree

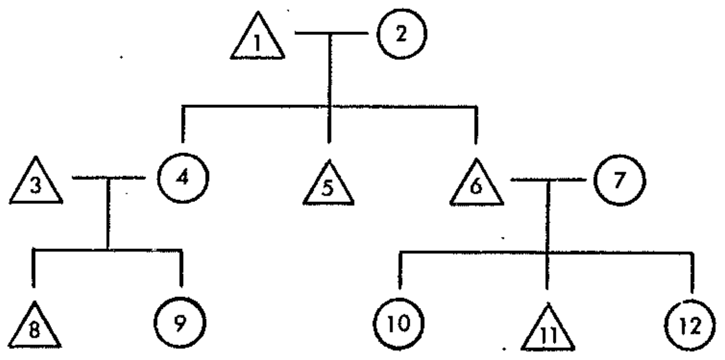
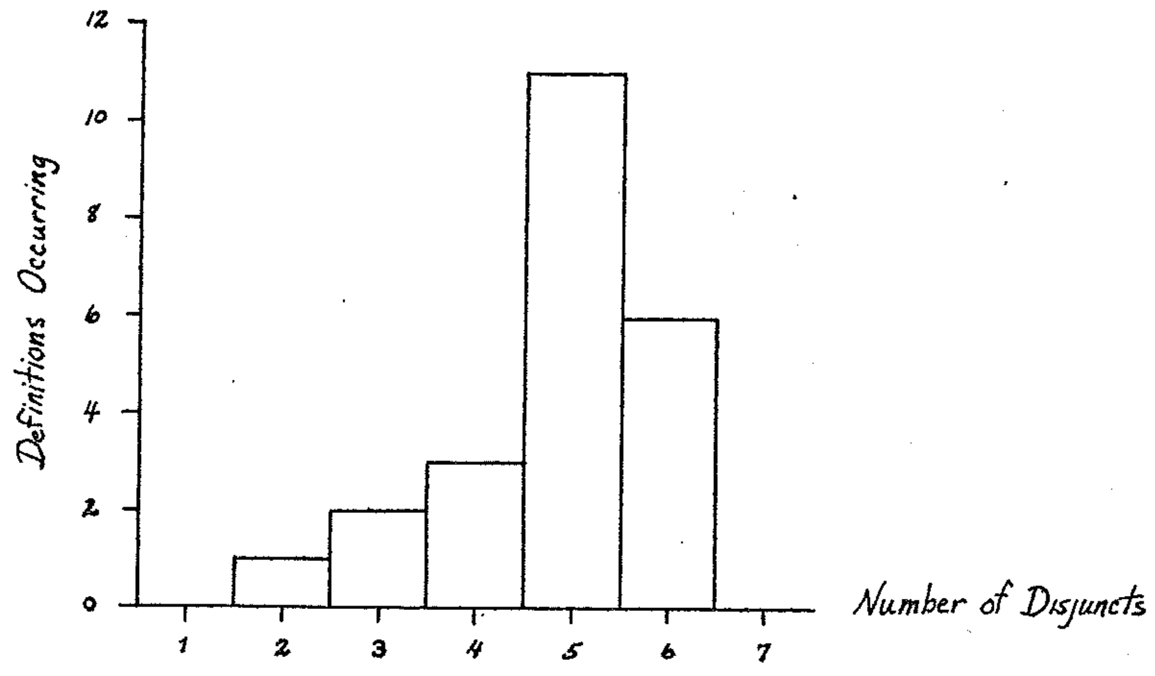


FIGURE 3.4
Frequency Distribution of Disjunctive Definitions



An examination of these two sets of conjectural definitions revealed that the ordering which lead to the fewest number of disjuncts consisted of grouping all the parents who were female together and then grouping all parents who were males together (remember that the system had no knowledge of unary predicates such as Male and Female). The grouping that produced the initial set of conjectural definitions was:

Group 1:

	(7 PN 10)	
(3 PN 8)	(7 PN 11)	Leading to MN/SN/UM
(3 PN 9)	(7 PN 12)	(see Figure 3.3)

Group 2:

	(6 PN 10)	
(4 PN 8)	(6 PN 11)	Leading to SN/UM, etc.
(4 PN 9)	(6 PN 12)	(see Figure 3.3)

Group 3:

(1 PN 4)	(2 PN 4)	
(1 PN 5)	(2 PN 5)	Leading to GN/NM, etc.
(1 PN 6)	(2 PN 6)	(see Figure 3.3)

Obviously, it is very difficult to abstract any apparent structure from this clustering.

This example suggests that whenever a relation is encountered that requires a disjunctive definition, we should probably search for the definition which involves the fewest number of disjuncts. Our experience has so strongly supported this belief that the current version of "the Monkey's Uncle" operates on a basic minimality principle:

> Principle:
>
> Always attempt to discover a disjunctive definition that involves the fewest number of disjuncts.

The application of this principle has produced a particularly interesting side effect in that the disjunctive definition selected under it reflects a particular clustering of the examples. We will illustrate in a later chapter how these clusterings can be suggestive of hitherto unknown relations which, when postulated, lead to a much simplified theory of the data.

On Checking for Over-Generality

The result of applying Strategy #1 to a data base is a collection of conjectural definitions for the particular relation R under study. A property of any such conjectural definition is:

P1: (x)(y) [(x,y) element of R =>
(x,y) element of Conjectural Definition].

The problem of over-generality lies in verifying the converse of this property:

P2: (x)(y) [(x,y) element of
Conjectural Definition => (x,y) element of R].

At first glance, the verification seems straight-forward: simply compute the extension of the conjectural definition and test for its inclusion in the extension of R. If it is included, then the conjectural definition has been verified. The problem of efficiently performing this computation is the concern of associative memory processors like TRAMP and, as such, will not be discussed here. Such verification is possible, but it is expensive for large data bases.

We were primarily interested in developing heuristic techniques that could be used to efficiently detect over-general conjectures. After these are eliminated, of course, the user has the option of applying exhaustive techniques to the few remaining definitions so as to remove any doubt about their validity.

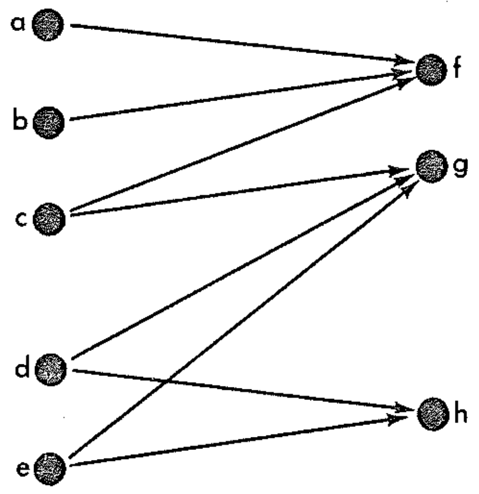
Our heuristic procedures are based on the concept of "the inverse image of x with respect to a binary relation R" informally defined as the set of all y's such that the range of x (i.e. R(x)) overlaps the range of y.

Def: The inverse image of x with respect to a binary relation R, denoted IR(x), is:

IR(x) = (y| there exists some z such that: (x,z) Element of R and (y,z) Element of R).

Inverse images do not form equivalence relations on the domain of the given relation. If R is represented by the following directed graph, then the collection of inverse images is exhibited in Table 3D. From these we see that inverse image sets form a cover but not a partition.

TABLE 3D
Hypothetical Relation



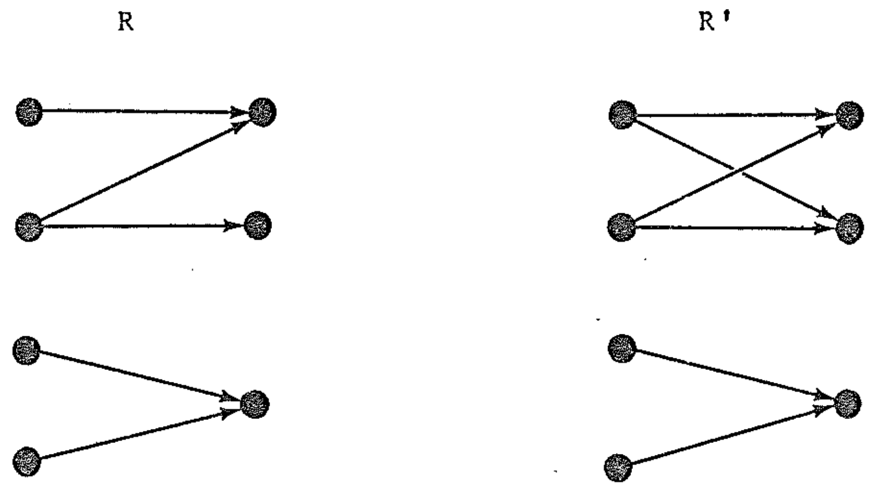
IR(x)

IR(a) = a,b,c
IR(b) = a,b,c
IR(c) = a,b,c,d,e
IR(d) = c,d,e
IR(e) = c,d,e

In a similar manner, we can define the inverse image of x with respect to a conjectural definition of R since, clearly, the extension of a conjectural definition can be computed if the extensions of the relations occurring in that definition are known.

Our heuristic procedure picks an object x in the domain of the relation R being defined and computes both IR(x) and I"Def"(x). It then checks to see if I"Def"(x) is contained in IR(x). If not, it rejects the "Def"; otherwise it chooses a new x and repeats this procedure until the domain of R has been exhausted. A faster but less complete version of this heuristic is to choose only x's which have not occurred in any of the previously computed inverse images.

These checks are only heuristic since they do not take into consideration the local connections within each inverse image class. This difficulty is illustrated below as we note that the relations R and R' would be claimed equivalent according to this check. Nevertheless, the type of global structure captured appears to be significant. For example, the inverse image of a brother is the set of his other brothers, as contrasted with the set of his siblings. This property is precisely what is needed to eliminate such over-general conjectures as "SM = SN".



A more realistic understanding of the power and limitations of this heuristic may be seen from a data base analyzed by this system. The data reflect the family tree shown in Figure 3.5 below.

The definitions for the binary relation "Niece" were first explored. Table 3E lists the conjectural definitions resulting from the execution of Strategy #1. These definitions were then subjected to the above heuristic check which resulted in the definitions given in Table 3F of the original thirty-two intensional definitions twenty-three were rejected by our check and the remaining definitions were all correct. The definitions for "Uncle" were then explored with the results similarly detailed in Tables 3G and 3H. This time, the heuristic failed to detect that 8 of the remaining definitions were over-general (Table 3I contains the list of conjectural definitions which survived the exhaustive check). For example, the definition "Uncle = Husband/Aunt/Cousin" (i.e., MM/UF/CN) passes the test since no notice is taken of the fact that this would imply that one's father is also one's uncle.

The need for developing an heuristic for speeding up the verification of conjectured definitions is better appreciated when we understand the complex interactions between the generation of conjectures and their subsequent verification. In fact, there is a constant cycling of information between these two phases. Suppose the first set of definitions conjectured were subsequently disproved. Then all the paths in the data graph that lead to these

FIGURE 3.5
Family Tree Underlying Data Base

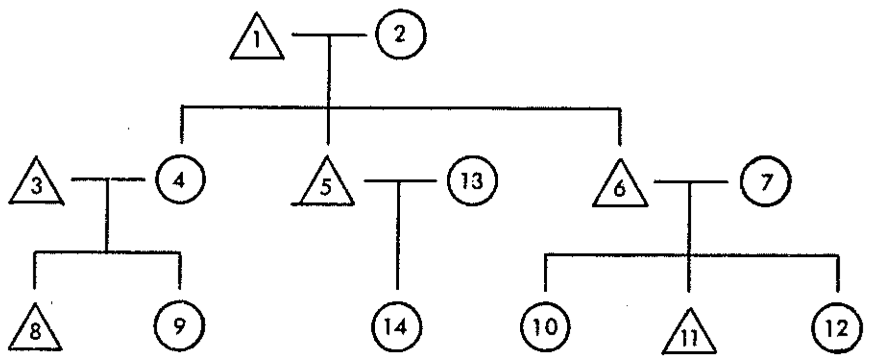


TABLE 3E

Conjectured Definitions of "Niece"
Prior to Heuristic Check

Cousin/Daughter
Cousin/Offspring
Cousin/Daughter/Spouse
Cousin/Cousin/Daughter
Cousin/Cousin/Offspring
Offspring/Uncle/Offspring
Offspring/Uncle/Daughter
Daughter/Uncle/Offspring
Daughter/Uncle/Daughter
Offspring/Aunt/Offspring
Offspring/Aunt/Daughter
Daughter/Aunt/Offspring
Daughter/Aunt/Daughter
Cousin/Cousin/Nephew
Cousin/Offspring/Spouse

TABLE 3F

Conjectured Definitions of "Niece"
Remaining After Heuristic Check

Daughter/Aunt/Daughter
Daughter/Aunt/Offspring
Daughter/Uncle/Daughter
Daughter/Uncle/Offspring

TABLE 3G

Conjectured Definitions of "Uncle"
Prior to Heuristic Check

Spouse/Aunt
Husband/Aunt
Brother-in-Law/Parent
Parent/Cousin
Father/Cousin
Spouse/Aunt/Cousin
Husband/Aunt/Cousin
Spouse/Sister-in-Law/Mother
Husband/Sister-in-Law/Mother
Spouse/Sister-in-Law/Aunt
Husband/Sister-in-Law/Aunt
Spouse/Sister-in-Law/Parent
Husband/Sister-in-Law/Parent
Spouse/Parent/Cousin
Spouse/Mother/Cousin
Husband/Parent/Cousin
Husband/Mother/Cousin
Brother-in-Law/Aunt/Cousin
Brother-in-Law/Sister-in-Law/Aunt
Brother-in-Law/Parent/Cousin
Brother-in-Law/Spouse/Parent
Brother-in-Law/Brother-in-Law/Aunt
Parent/Cousin/Cousin
Father/Cousin/Cousin
Parent/Offspring/Aunt
Parent/Daughter/Aunt
Father/Offspring/Aunt
Father/Daughter/Aunt
Parent/Niece/Parent
Father/Niece/Parent

TABLE 3H

Conjectured Definitions of "Uncle"
Remaining After Heuristic Check

Spouse/Aunt
Husband/Aunt
Brother-in-Law/Parent
Father/Cousin
Spouse/Aunt/Cousin
Husband/Aunt/Cousin
Spouse/Sister-in-Law/Mother
Husband/Sister-in-Law/Mother
Spouse/Sister-in-Law/Aunt
Husband/Sister-in-Law/Aunt
Spouse/Sister-in-Law/Parent
Husband/Sister-in-Law/Parent
Spouse/Mother/Cousin
Husband/Parent/Cousin
Husband/Mother/Cousin
Brother-in-Law/Aunt/Cousin
Brother-in-Law/Sister-in-Law/Aunt
Brother-in-Law/Parent/Cousin
Brother-in-Law/Spouse/Parent
Brother-in-Law/Brother-in-Law/Aunt
Father/Cousin/Cousin
Father/Offspring/Aunt
Father/Daughter/Aunt

TABLE 3I

Conjectured Definitions of "Uncle"
Remaining After Exhaustive Check

Spouse/Aunt
Husband/Aunt
Brother-in-Law/Parent
Father/Cousin
Spouse/Sister-in-Law/Mother
Husband/Sister-in-Law/Mother
Spouse/Sister-in-Law/Parent
Husband/Sister-in-Law/Parent
Spouse/Mother/Cousin
Husband/Parent/Cousin
Husband/Mother/Cousin
Brother-in-Law/Spouse/Parent
Father/Offspring/Aunt
Father/Daughter/Aunt
Father/Niece/Parent

conjectures would have to be temporarily "removed" and a new set of conjectures would have to be invented. These would then have to be checked, and so on.

Our experiments have revealed that, for most of our test data, this process is often repeated a dozen times before a set of conjectures are formed which can not be disproved within the data itself. Consequently, our system must typically verify many more conjectures than appear in the final output.

CHAPTER FOUR

A Description of "the Monkey's Uncle"

In this chapter we will discuss some of the implementation details of "the Monkey's Uncle". Justification for some of the design philosophy will also be presented in the light of certain experimental results that have been obtained.

A unifying characteristic of the tasks described in the previous chapter is their reliance on lenghty non-numeric computations. Because we were interested in a practical system, one on which we could afford to run a great many experiments, we decided to explore the use of small computers which could do symbol manipulation reasonably cheaply. A 16-k, two DEC-Tape PDP-9 was our choice of the available machinery. Since we anticipated large amounts of data and heavy reliance on list processing, we sought a language which was slanted toward list processing but which would enable us to efficiently build arbitrary data structures and easily pack data into them. Such a language, called L6, existed on the IBM 7090. Using it as a prototype, an L6/9 interpreter (1) was built which provided several extensions to circumvent some of the speed problems associated with interpreters (2).

We wanted an interactive system which would enable the user to monitor the progress of the computation and, most importantly, would allow him to modify his data base quickly and conveniently. One of our goals was a system which would facilitate tentative modifications of the data and then reveal the consequences of these modifications - both locally (as pertaining to the particular relation modified) and globally (as pertaining to the entire set of definitions for all the relations). We hoped that, thus equipped, the user would be able to "play" with his data in a way which would enable him to get a feeling for existent interdependencies.

The current version of "the Monkey's Uncle" occupies about six thousand words of core. On initialization, it first requests the user to load a file containing the names of the relations to be considered in the data base. It then requests the data file to be loaded (in which each item is of the form: "Relation (X) = X1,X2,...,Xn"). Each item is processed by adjoining each of its n 2-tuples to a labelled directed graph. When all the input data are processed, the extensions of each relation are computed, packed into tables and stored on DEC tape. The system then requests the name of the particular relation to be explored and inquires as to whether or not:

1) Recursive definitions are to be allowed,

2) The extension of the relation is to be printed,

3) Any check for overgenerality should be made, and, if so, whether an exhaustive check should be made following the heuristic check,

4) Any intensional definitions should be discarded automatically (which might be the case if the user were interested in disjunctive definitions and already knew an all-encompassing compositional definition),

5) Only disjoint disjunctive definitions are to be generated (a disjunctive definition for R is disjoint if every 2-tuple of R is contained in one and only one of the disjuncts of the definition).

Lastly, it requests the number of data samples of the particular relation that should be considered thus permitting the user to examine only a portion of his data. In addition to these requests, there are numerous console switches that can be activated to permit closer examination of the system's activities.

Conjecture Generation

After responses are made to the above requests, the system enters the conjecture generation phase. A 2-tuple is selected from the extension of the specified relation R and the labelled path sequences (LPS's) of all the directed paths bridging this 2-tuple are collected onto a special list - henceforth to be called a P-list. Before another 2-tuple is considered, each path on the current P-list is merged into a structure called the INT-list. For any given LPS on the current P-list, either it does or doesn't already occur on the INT structure. In the latter case, it is added to the INT structure along with information specifying which 2-tuple generated it.

In order to fully understand what happens in the former case we must examine the INT structure itself. Each element of this structure has two fields. The first field consists of a binary n-tuple, or template, the ith bit of which denotes the ith 2-tuple of R's extension. The second field is a storage cell where an LPS can be saved. In the case where we have an LPS on the P-list that already exists on the INT structure, we locate it on the INT structure and then set to one the template bit corresponding to the current 2-tuple.

After the desired subset of R's extension has been processed, the system starts to construct its conjectures by attempting to locate the LPS's which are in the intersection of all the P-lists. Such LPS's can be immediately determined by simply scanning the INT structure for those elements whose n-tuples are all ones. If there exist no such n-tuples, then, for that relation, there are no discoverable intensional definitions which involve only compositions. In other words, if a definition is to be found it will necessarily involve several disjunctive terms.

The problem of finding a disjunctive definition is approachable through the INT structure without having to directly intersect P-lists. Recalling how the bits of each template are determined, we see that, in fact, each occurrence of a particular template specifies an LPS that is contained in all the P-lists which correspond to bits set to one.

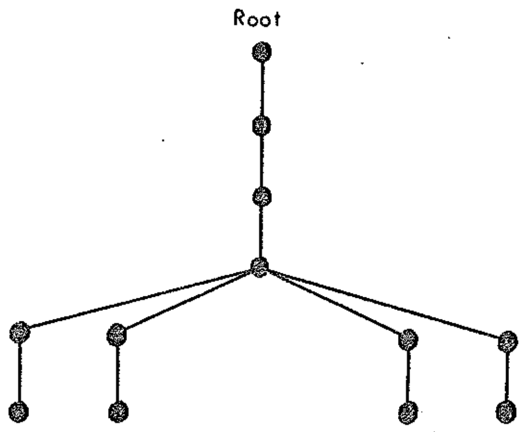
A disjunctive definition is constructed by simply choosing a set of templates whose union covers the n-tuple of all ones - i.e., all of the 2-tuples of R under consideration. Each disjunct of the definitions corresponds to one of the selected templates. To convert a particular template into all the LPS's which are associated with it, we scan the INT structure for elements whose first field equals that template. For any such element, its second field contains an associated LPS. Each LPS is linked to one template. This template, however, need not be uniquely associated with this LPS. Consequently, the INT structure must be exhaustively scanned for all elements whose template field equals the given template. In this manner, all LPS's associated with the template are found. The problem of finding the simplest disjunctive definition now resembles a classical covering problem in which we must select the smallest number of n-tuples whose union covers the n-tuple of all ones - i.e., all the 2-tuples of R under consideration. Viewing each n-tuple as a template, the above may be restated in terms of finding a minimal number of templates whose union covers the identity template.

Thus, the central problem of finding the simplest disjunctive definition is reduced to finding a minimal set of templates whose union is the identity template and then constructing the LPS's from them. In general, this problem has no good algorithmic solution for it requires an exhuastive search of a tree of all possible combinations of templates. What follows is a description of a heuristic technique for restricting this search, which has yielded excellent results on our experimental data.

Initially we define a "Goal" template and set all of its bits to "one". We then scan the INT structure to see if any of its templates covers the Goal template. If we find any, we are finished. If there is no such template, we search for the template with the maximum number of ones.

When this is determined, all templates which have that number of ones are located and pushed onto a stack. For each template on the stack, a subgoal template is created which represents the bits that remain to be covered if that particular template were to be added to the set of templates constituting the solution. A look ahead procedure is then called that takes each subgoal template and computes the number of ones that could be covered on the next iteration if this subgoal template were to be made a goal template. A maximum number is then computed over the set of all the subgoals (at this level). All subgoal templates which, under one step look-ahead, did not achieve this maximum are eliminated from further consideration. Control is then passed to a supervisor which could permit a variety of strategies to be invoked. Our current strategy simply determines whether the next iteration of this process will find a total cover (by testing to see whether the computed maximum equals the remaining number of bits to be covered). If it will, then, in turn, each subgoal is raised to goal status and the whole process is repeated. If the cover will not be completed on the next pass, all the subgoals are erased except the last subgoal which is raised to goal status. The process is then repeated. In summary, this strategy prunes all branches off the subgoal tree except one until it discovers that the next depth in the tree will yield a solution (i.e. that the union of all the templates along the chain from the initial node to the terminal node will form a cover). In this latter case, all the branches stemming from the next to last node are preserved.

FIGURE 4.1
Subgoal Tree



In Figure 4.1 each node denotes a particular template and the union of all templates along any path from the root to the terminal node forms the identity template. When the desired tree is completed, special routines are executed which derive from the tree a list of disjunctive definitions by converting each template into the LPS's underlying it. The set of LPS's associated with a selected template constitute all of the compositional alternatives for that disjunct.

Advantages and Disadvantages of the "INT" Structure

The INT data structure enables us to completely avoid having to worry about the order in which we perform pairwise intersections of P-lists. This is, in part, due to each template specifying precisely what P-lists each LPS is contained in. However, this solution has inherent in it a space time trade-off in that it requires the storage of every LPS encountered. In simply intersecting the P-lists, only two P-lists need be explicit at any one time. These are then intersected leaving only the resultant list to be kept. If the data-graph is highly interconnected, each P-list will be large since it represents all possible paths from x to y. If each p-list has few LPS's in common with any other, then the INT structure will grow approximately as n times the size of the average P-list. If the data graph is to represent a large amount of data, the INT structure could become unreasonably large. On the other hand, it seems intuitively plausible that interesting data (or, at least, any data to which one might want to apply this system) is at least locally structured in the sense that there are apt to be clusters of 2-tuples which have many of the same kinds of interconnections. In this case, the chance of not finding a particular LPS already on the INT structure should decrease as more and more LPS's are added to that structure. If so, the growth rate of the INT structure should approach some asymptotic value.

To explore this critical hypothesis, we constructed a data file of approximately six hundred facts from a hypothetical family tree. The growth behavior of the INT structure was recorded over seven different relations in the data base. Figure 4.2 illustrates how its growth levels off as more 2-tuples are considered.

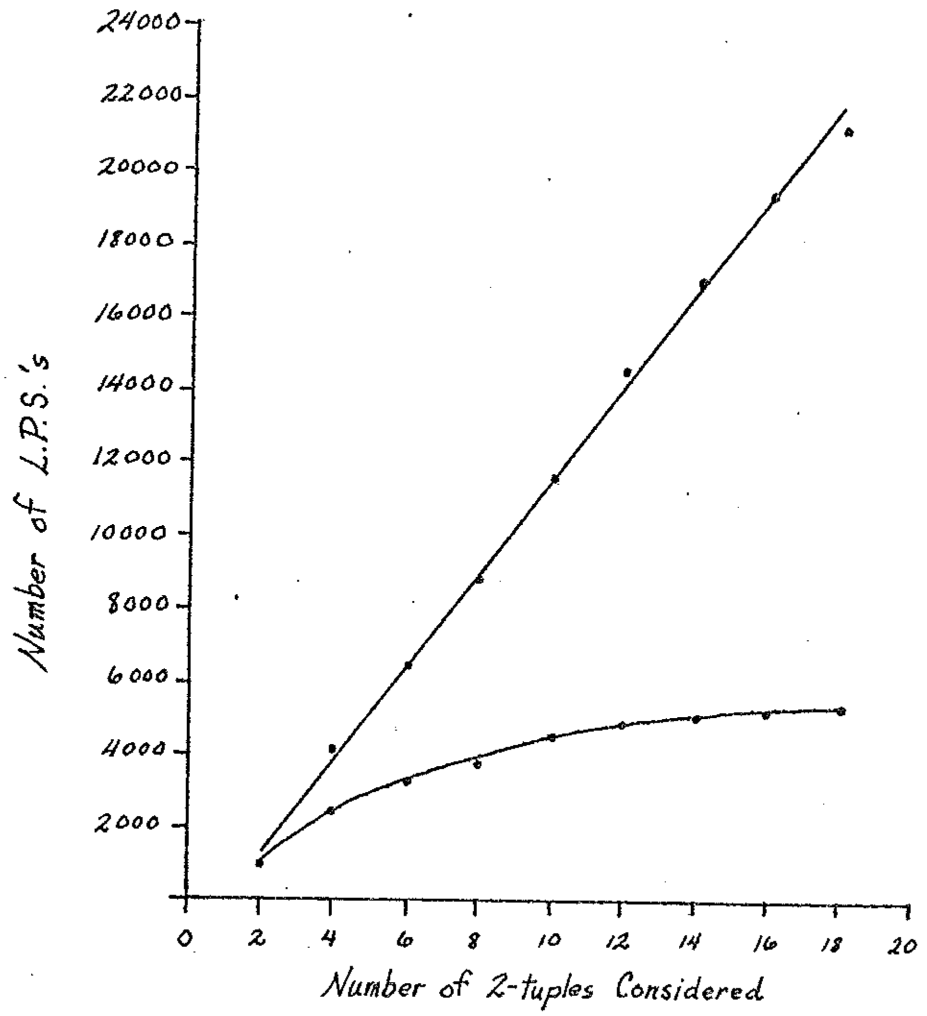
The template approach achieves yet another advantage over the direct intersection of P-lists. The search for disjoint disjunctive definitions proceeds by first selecting the best template (as defined above). Then, when the next level goal template is determined (by complementing the union of the preceding selected templates), only those LPS's are considered whose associated templates are subsets of the current goal template. To check the proper containment of one binary pattern in another is extremely efficient in most machines.

Conjecture Verification

On finishing the conjecture generation phase, the system enters the verification phase. As mentioned earlier, conjecture verification can occur with various degrees of completeness, ranging from no verification at all to exhaustive checks. The computations involved are all of the sort:

> Given the set S of individuals, compute the resulting set R(S) for some given relation R or its converse.

FIGURE 4.2
Growth Rates



For example, to compute the inverse image of x with respect to the compositional definition R1/R2/R3, we must compute the following sets:

```
S1 = R1(x)
S2 = R2(S1)
S3 = R3(S2)
S4 = *R3(S3)
S5 = *R2(S4)
S6 = *R1(S5)
```

where S6 is the desired set.

The techniques used involve linear searches of tables which contain all of the 2-tuples of the given relation. Since the system has always been extremely core-bound, more sophisticated techniques involving any form of redundant coding were considered infeasible. Our explicit concerns in conjecture verification have focused primarily on useful heuristics rather than on clever data structures.

Interactive phase

When a set of conjectures is finally discovered which satisfies the criteria established by the user, the system enters the "Direct" (Interactive) mode. Among other things, this enables the user to examine either rejected conjectures (a capability which is especially important if no satisfactory definitions have been obtained) or the final answers. In either case, interactive examination of these structures enables the user to print out only that information relevant to his current interests or hypotheses.

After the user has satisfied his curiosity about the output, he can enter a second interactive phase. This phase allows him to:

1) Block any definitions from further consideration
2) Delete any triples from the data graph
and 3) Insert new triples in the data graph.

In addition, the user can invoke a display procedure which requests a 2-tuple and a compositional definition. It then reconstitutes the exact path underlying the definition and prints out the intermediate nodes of this path. In this way the user can discover the persons (or objects) which functioned as the intermediate nodes for any particular LPS. The use of this procedure may best be understood by means of an example. Suppose, for example, that one discovered the following definition for "GM" (Grandfather):

GM = PM/PN V PM/MN/PN

(Father/Parent or Father/Spouse/Parent)

and that only one 2-tuple (perhaps, <a,b>) required the second disjunct i.e., was not covered by the first disjunct. This definition could then be collapsed into one disjunct by finding the "z" such that (a, PM, z) and (z, MN/PN, b) and by then adding the fact (z, PN, b) to the data base.

Modification of the data base is, of course, potentially disasterous and should not be contemplated unless there is supporting evidence for such a change. However, if we believe in Occam's razor, convincing evidence is often forthcoming since (as will be discussed later) a few small changes can drastically simplify the intensional definitions for the entire collection of relations.

In this chapter we have tried to convey some of the considerations underlying the design of "the Monkey's Uncle". We have viewed the system as if it were broken into three distinct phases - namely conjecture generation, conjecture verification, and interactive inspection and modification. In reality, the first two phases are quite intertwined; each calling on the other when necessary. The actual coding of this system has proceeded from a modular point of view in that a large collection of fairly independent combinatorial routines were constructed. Using these routines as building blocks, a wide variety of approaches to this problem have been experimented with, culminating in the current version of "the Monkey's Uncle".

FOOTNOTES

(1) An interpreter was decided upon because of the absolute premium on core memory. An estimated factor of ten in core efficiency was achieved over compiled code. The interpreter occupies 1680 words including I/O handlers.

(2) Mr. Peter Headly implemented this language on the PDP-9.

CHAPTER FIVE

Exploring the Behavior of "the Monkey's Uncle"

This chapter is to illustrate how "the Monkey's Uncle" can be used to probe all the interrelationships of a given relation; to generate intensional definitions for a set of relations; to find "Rules of Inference," and finally to discover clusters of objects on which the relations are defined.

Since the output of this system is lengthy and complex, we shall detail the actual output generated in the exploration of only a single relation. For the remaining examples we will use a condensed representation of the output and relegate the actual output to appendices.

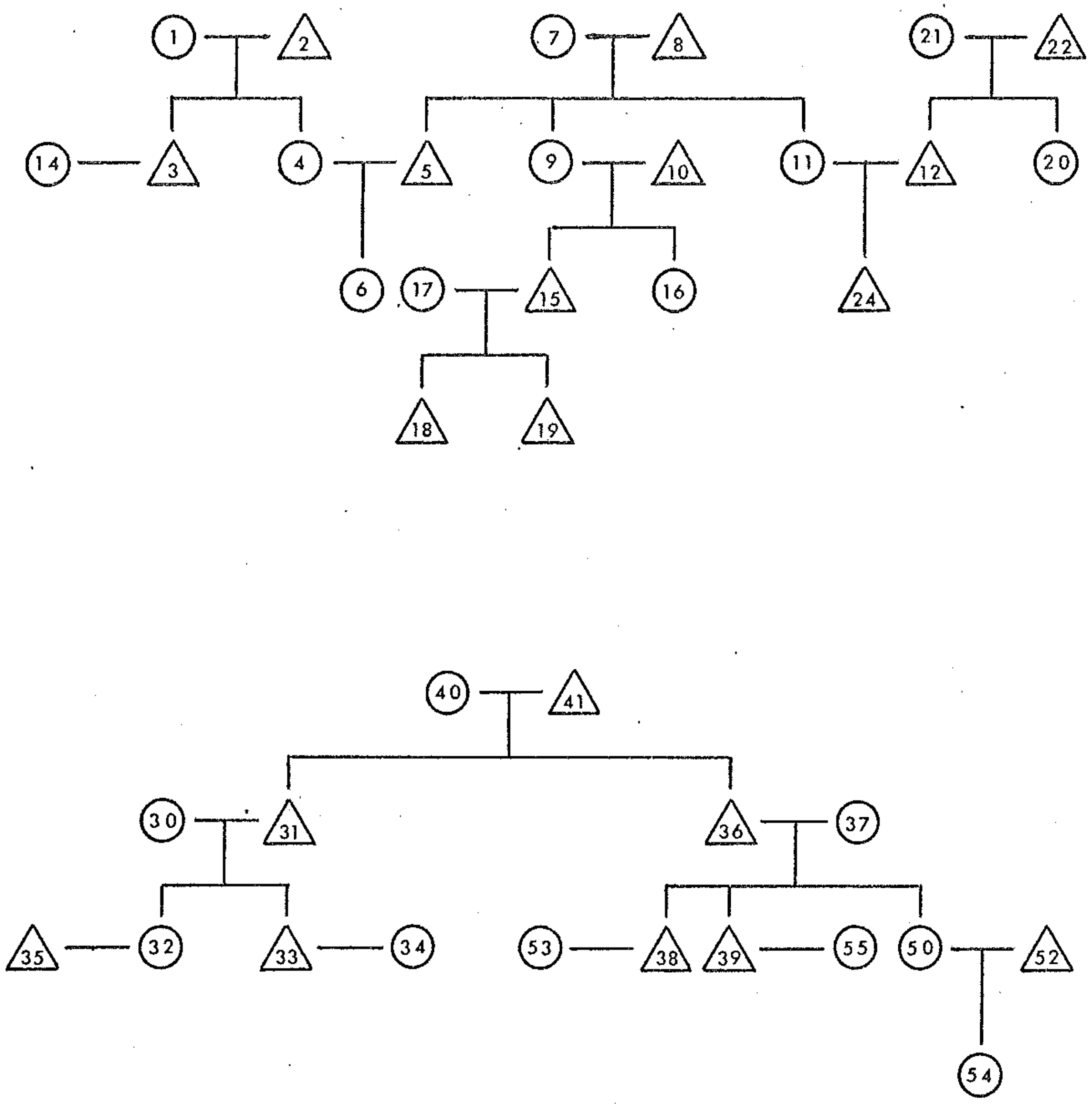
All of the examples considered in this chapter are based on data selected from our kinship system (with the single exception of the absence of divorce). Our hope in so doing is that the reader will more easily understand processes within the system in terms of the results which it produces. This will be especially true in understanding the significance of various situations underlying the discovery of the rules of inference in several of our examples. Although this data domain is well understood and quite structured, the application of our system to this data will produce some distinguishing results. As far as our system "knows", this data might have been generated from a totally foreign culture. The analysis it provides is actually quite different from that produced by an anthropologist. An anthropologist would endeavor to analyze this foreign culture's kinship terminology relative to his set of atomic terms (i.e., in terms of biological parents and spouses).

Our system makes no presupposition about anyone else's primitive relations. It attempts to unfold a theory entirely within the framework of that culture with absolutely no biases concerning the importance of the underlying biological family tree. We emphasize that "the Monkey's Uncle" is dedicated to finding recursive abstract theories, all of the terms of which are either 1) primitives of the data, 2) defined with respect to other terms or 3) defined with respect to themselves.

An Illustration of "the Monkey's Uncle"

The first set of examples was obtained by analyzing the data file listed in Appendix 4. This file contains 580 (x,R,y) facts involving thirty-two kinship relations defined on a domain of forty-five individuals. To help the reader in understanding this maze of facts, we include in Figure 5.1 a family tree representation of this data (which, of

FIGURE 5.1
A Complicated Family Forest



course, would be unknown to the system until it had discovered a theory based on the relations coincidental with Parent and Spouse). Each individual in the tree is assigned a number which serves as his name in the extensional definitions of the given relations. Each relation in this file is designated by a two-letter mnemonic of the sort previously described. A complete list of these mnemonics may be found in Appendix 3.

The following two pages are an annotated version of the first part of the output from the analysis of the "Uncle" ("UM") relation. The underlined parts are typed by the user. The first two messages request that input data be loaded into the high speed paper tape reader. The next line indicates that this data has been digested (i.e. that a labelled directed graph has been constructed) and requests the name of the relation to be explored. This is followed by a series of self-explanatory questions which ends with a request for the number of samples to be considered in this pass. Since we had specified that the extensions be printed, the next few lines are the sixteen 2-tuples constituting the subset of the "UM" extension that is investigated (1). The order of this printed set of 2-tuples is used to generate a template in which there is a one to one correspondence between each bit of the template and each

FIGURE 5.2
Sample Output

```
                    ****WELCOME TO THE MONKEY'S UNCLE****


PLEASE LOAD THE RELATION NAMES
NOW LOAD YOUR DATA FILE


TYPE IN RELATION TO BE EXPLORED
UM
RECURSIVE ? NO
PRINT 2-TUPLES? YES                                       first 16 2-tuples
WHAT DEGREE OF VERIFICATION IS DESIRED? COMPLETE          of the UM relation
ANY DEFINITIONS TO BE BLOCKED? NO
DISJOINT ?YES
 # OF SAMPLES?  16
<039,054> <038,054> <036,032> <036,033> <031,038> <031,039> <031,050>
<012,006> <012,015> <012,016> <010,006> <010,024> <005,015> <005,016>
<005,024> <003,006>

X
<CRITERIA SATISIFIED....EXAMINE SOLUTION> <-- Entering interactive phase

----<-- Command to print           Goal template
LIST T  top level goal

027153:471173 |377776  000000| 037777 <-- Special flags

031173:440000 |377776  000000| 025103 <-- Pointer to list of definitions
----
25103T <--                     Solution template

025103: 540000  LM/MN/PN  MM/LF/PN  MN/LF/PN    LM/PN    MM/UF

025111:     MN/UF                      Command to print the list starting at
----                                   location 25070 and to interpret core
* <-- Exit from this mode              according to a prespecified translation
LEAVING EXAMINE MODE                                                  table

NOW ANY DEFINTIONS TO BE BLOCKED? YES
ERASE CURRENT LIST?NO
NEW DEFINITIONS NOW
: MN/UF
: MM/UF
: LM/PN
: MN/LF/PN    <-------- definitions to be blocked
: MM/LF/PN
: LM/MN/PN
: *  <------ done
```

FIGURE 5.2 (cont.)

```
YXX
<CRITERIA SATISIFIED....EXAMINE SOLUTION>
----
LIST T                              goal template

026177:467627  377776  000000   000000

027627:440000  376676  000000   025133 <-- pointer to first disjunct list
----            pointer to
25133T          second template     solution template

025133: 525157  MM/LF/PM  MN/LF/PM  } <-- first disjunct
----
LIST 25157

025157:465123  001100  000000  000000

025123:440000  001100  000000  025163
----
25163T

025163: 540000  LM/SF/PM  LM/SN/PM  MM/SF/PM  MM/SN/PM  MN/SF/PM  }  second
                                                                  }  disjunct
025171: EM/XM/PF  EM/PM/PM  EM/PN/PM  EM/XF/PF  EM/PF/PM          }
----
*
LEAVING EXAMINE MODE

NOW ANY DEFINTIONS TO BE BLOCKED? NO
DISPLAY ?NO
DELETIONS ? NO
ADDITIONS? NO

TYPE IN RELATION TO BE EXPLORED
```

2-tuple (i.e. the first 2-tuple : <39,54>, corresponds to the high order bit of the template, the 2-tuple to the second highest order bit, and so on). Any definitions for "UM" must at least account for each "one" bit of this template.

After the criteria defined by the user's responses to the first five questions are satisfied, the system enters the "Direct" mode. Typing "LIST T" causes the goal template to be typed out, followed by the templates which covered the maximum number of 2-tuples (the general form of the data structure pointed to by "T" is shown in Figure 5.3). We discover that this template equals the goal template, which signifies that a set of definitions has been generated which require only compositions (that is, definitions consisting of single disjuncts). The list of definitions associated with the given template is pointed to by the last entry on line "031560" which, when interpreted, reveals the list:

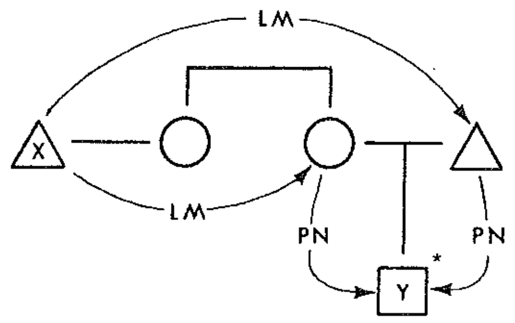
	LM/MN/PN		Brother-in-Law/Spouse/Parent
	MM/LF/PN		Husband/Sister-in-Law/Parent
	MN/LF/PN		Spouse/Sister-in-Law/Parent
UM =	LM/PN	i.e. Uncle =	Brother-in-Law/Parent
	MM/UF		Husband/Aunt
	MN/UF		Spouse/Aunt

Note that each of these terms constitutes a valid definition of Uncle and that none of them are disjunctive. This may be contrasted with the standard definition of Uncle: "Brother/Parent V Husband/Sister/Parent" - which involves two disjuncts.

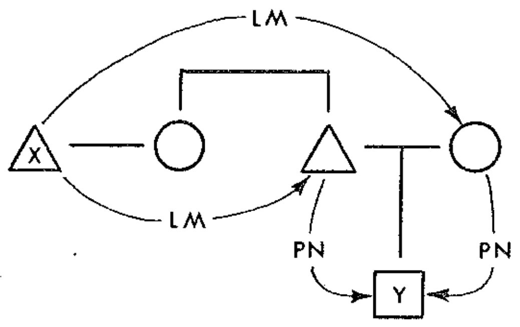
Reflections on the First Set of Discovered Definitions

We will consider the meaning and implications of these definitions so as to illustrate several facets of this type of system. Of particular interest is its discovery of the "Brother-in-Law/Parent" definition which is, in some sense, simpler than the standard definition. The validity of the definition can be understood by examining the four cases in which Brother-in-Law-hood arises:

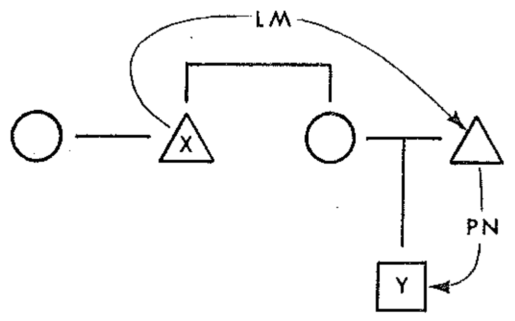
Case 1:



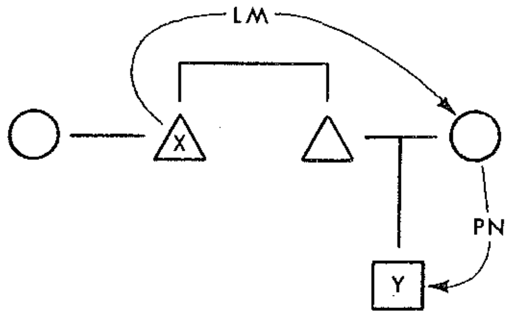
Case 2:



Case 3:



Case 4:



* Squares denote indivduals of unspecified sex.

In all four cases, x is the Uncle of y. We note that Brother-in-Law/Father (as distinct from Brother-in-Law/Parent) was not generated as a definition. This definition is consistent with cases 1, 2 and 3 but fails in case 4 (and for <31,38> in the data set - see Figure 5.1). In other words, "LM/PN" is a universally valid definition of Uncle whereas "LM/PM" is not.

Next, we consider the definition "Uncle = Husband/Aunt". This definition is surely not universally valid since there exist uncles who are bachelors. However, scanning our data we discover that in fact every uncle in our data sample is married. Consequently, the generation of this definition reveals this additional piece of structural information about this particular data set. Our system has no way of knowing if, in fact, this finding reflects the true state of affairs or whether it is merely an idiosyncracy of the data sample. In either case, the identification of this kind of structural property in the data is important. Closely associated with the "Husband/Aunt" definition is the definition "Uncle = Spouse/Aunt". Anyone familiar with the marital conventions of our culture understands that "Spouse/Aunt" is logically equivalent to "Husband/Aunt". We will later see how the co-occurrence of definitions which have terms in common will enable us to create rules of inference which can be used to establish logical equivalences.

A similar co-occurrence is noted in two of the three remaining definitions, i.e., "MM/LF/PN" and "MN/LF/PN". At first glance one might expect that "MM/LF/PN" is logically equivalent to "LM/PN"; however, the co-occurrence of these two definitions is again a reflection of the fact that for this particular data sample every Uncle is married.

Digging Beneath the First Set of Discovered Definitions

Since, in our example, the system discovers a set of compositional definitions covering all the designated 2-tuples, it has no reason to search for disjunctive definitions whose disjuncts necessarily cover only some of the 2-tuples. However, if the user wishes to explore all the possible logical interrelationships of the data, he must consider patterns in the data that do not uniformly hold over the entire extension of a given relation. To force this kind of exploration, the user causes the system to automatically reject all compositional definitions.

For this and other reasons, whenever the user exits from "Direct" mode (by typing an "*"), the system will ask the user whether or not he wishes to block any definitions. By listing all the compositional definitions, the user thus causes the system to recycle until definitions satisfying this restriction have been found. We performed the above operation (see Figure 5.2) and discovered that at least one definition had been discovered whose first disjunct covered only the 2-tuples corresponding to "376676" (i.e. Whose first disjunct covers all 2-tuples except the 8th and 11th). The definitions found at this juncture are summarized below. We have associated with each disjunct the template covered by it.

Uncle =

Husband/Sister-in-Law/Father
"376676" Spouse/Sister-in-Law/Father

V

Spouse/Sister/Father
Spouse/Sibling/Father
Husband/Sister/Parent
Husband/Brother/Parent
Brother-in-Law/Sister/Father
Brother-in-Law/Brother/Father
Son-in-Law/Father-in-Law/Mother
Son-in-Law/Father/Father
Son-in-Law/Parent/Father
Son-in-Law/Mother-in-Law/Mother
"004400" Son-in-Law/Mother/Father

On Digging Still Deeper

Since we had required all disjunctive definitions to be disjoint and since the first disjunct covered fourteen 2-tuples, the second disjunct must cover exactly two (the template underlying the first set of disjuncts is "376676" while the template for the second is "004400"). A more "balanced" definition might be obtained by blocking the first two definitions, since they accounted for so much of the data. An additional reason we wanted to block these definitions was, as the reader probably realizes, our desire to ascertain if and when we would "discover" the structure we knew was there - i.e., the standard definition of "Uncle". On doing this we find that there exist two alternative sets of disjunctive definitions and that the first disjuncts of both of these sets covers the same number of 2-tuples (see Figure 5.4 below).

This example manifests a somewhat interesting situation. We note that the template associated with the second disjunct in alternative two is a proper subset of the template associated with the the first disjunct of alternative one. We note also that the reverse is true with respect to the terms contained in these two disjuncts - i.e., the set of compositional terms forming the first

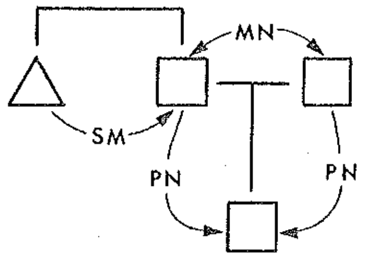
FIGURE 5.4
Discovered Definitions

```
                        ALTERNATIVE 1:

              Uncle =

                 Brother/Parent
 "376036"         Brother/Spouse/Parent

                              V

                          Husband/Sister/Parent
                           Husband/Sibling/Parent
          "00174"           Spouse/Sister/Parent
                             Brother-in-Law/Sister/Aunt


                        ALTERNATIVE 2:

              Uncle =

                 Brother-in-Law/Husband/Mother
                  Brother-in-Law/Husband/Parent
                   Brother-in-Law/Spouse/Mother
 "301776"           Brother-in-Law/Father

                              V

                          Sibling/Father
                           Brother/Father
                            Sibling/Father/Sibling
                             Brother/Father/Sibling
                              Sibling/Spouse/Mother
                               Sibling/Spouse/Parent
                                Brother/Husband/Mother
                                 Brother/Spouse/Mother
                                  Brother/Husband/Parent
                                   Brother/Husband/Mother
                                    Sibling/Father-in-Law/Spouse
                     "07600"         Brother/Father-in-Law/Spouse
```

disjunct in alternative one is a subset of the set of terms under the second disjunct in alternative two.

At first, this phenomenon might seem somewhat paradoxical, since one might expect that the greater the number of examples taken into consideration (i.e., the larger the template), the greater the variety of the conjectured definitions. However, let us consider this specific situation

from a slightly different point of view: the 076000 template generates (2) a subgraph of the graph generated by the 376036 template. This subgraph possesses fewer structural constraints and hence is apt to have more compositional definitions since this subgraph possesses fewer "counter-examples" for the conjectured definitions. In other words, there is a trade-off between more LPS's and a greater chance of encountering a counter-example to a particular LPS. For example, those definitions not contained in disjuncts underlying the "376036" template reflect structural idiosyncracies of that subgraph which are invalidated when the subgraph is expanded. In fact, the only definitions which are invariant under such expansion are "SM/PN" and " SM/MN/PN". Both of these are universally valid and represent the following situation:



We will argue later that even definitions which can be invalidated by enlargement of the data space may be used to help generate "rules of inference".

The process of discovering definitions for a given relation, then blocking these definitions and attempting to discover additional definitions can go on until all definitions have been found. Discovery of all the definitions for "Uncle" on this particular data base required six iterations of this process, starting from one-disjunct definitions and ending up with three disjuncts in the definitions. Table 5A summarizes all the definitions unfolded. The complete output for this analysis is reproduced in Appendix 5. The set of definitions most nearly resembling the "expected" definitions for "Uncle" are those in alternative one above, with the single exception of "Brother-in-Law/Sister/Aunt". Despite the complexity of this data base, there are no examples of an uncle who is a

TABLE 5A

Definitions of Uncle
Arranged in Order of Output

(1)	MN/UF		
	MM/UF		
	LM/PN		
UM =	MN/LF/PN		
	MM/LF/PN		
	LM/MN/PN		
(2)			LM/SF/PM
			LM/SN/PM
			MM/SN/PM
UM =	MN/LF/PM	V	MN/SN/PM
	MM/LF/PM		MN/SF/PM
			EM/XM/PF
			EM/PM/PM
			EM/PN/PM
			EM/XF/PF
			EM/PF/PM
(3)			SM/PM
			XM/MM/CN
			XM/LM/CN
	LM/PM		SM/PM/SN
UM =	LM/MN/PF	V	SM/MN/PF
	LM/MM/PN		SM/MM/PN
	LM/MM/PF		SM/MM/PF
			SN/XM/MN
			SM/XM/MN
(4)			
			MN/SF/PN
UM =	SM/PN	V	MM/SN/PN
	SM/MN/PN		MM/SF/PN
			LM/SF/UF
(5)			
			MM/SF/PF
UM =	MM/LF/PF	V	MM/SN/PF
	MN/LF/PF		MN/SF/PF

TABLE 5A (cont.)

(6)			SM/PF
	LM/PF		OM/XM/PM
	PM/OM/UF		OM/PM/PF
	PM/NM/PM		OM/XF/PM
UM =	PM/NM/PN	V	OM/PN/PF
	PM/NM/PF		OM/PF/PF
	LM/MN/PM		SM/MN/PM
	LM/MF/PN		SM/MF/PN
	LM/MF/PM		SM/MF/PM

(7)			SN/PM		
			SM/PM		
	SM/PF		XM/LM/CM		
	OM/XM/PM		SN/PM/SN		
	OM/PM/PF		SM/PM/SN		MN/SF/PN
	OM/XF/PM		SN/MN/PF		MM/SN/PN
UM =	OM/PN/PF	V	SN/MM/PN	V	MM/SF/PN
	OM/PF/PF		SM/MN/PF		LM/SF/UF
	SM/MN/PM		SM/MM/PN		
	SM/MF/PN		SM/MM/PF		
	SM/MF/PM		SN/XM/MN		
			SM/XM/MN		

father and whose wife has two sisters. If there had been such an example, the definition "Brother-in-Law/Sister/Aunt" would have been rejected, since said person would have been an "uncle" to his own child.

The above analysis of the "Uncle" relation helps to illustrate that a collection of data often contains a surprising number of structural relationships which are apt to go undetected. If these relationships are universally valid then they should be understood before axiomatization of the data (such as for a question-answering system) is attempted.

A Shallow but Broad Application

Using the same data file as in the above example, we shall summarize the results of applying this system to each relation in turn. We shall, however, limit ourselves to only one cycle for each relation. That is, rather than seek out all possible intensional definitions for a particular relation, we will settle for the first set of definitions discovered. What follows in Table 5B is a summary of these discovered definitions. Those definitions which we feel reflect idiosyncracies of the data sample have been starred. Again, we wish to stress that what constitutes an idiosyncracy in the initial data base is often determinable by examining a larger data sample. This is one of the reasons why we have chosen a set of relations whose structure and meaning are well understood. In this case one can easily detect the idiosyncratic structures that emerge and can then infer what the artificial constraints in the initial data sample were that lead to the formation of these particular definitions. Being made aware of these constraints is often valuable in determining how to select your next data sample.

TABLE 5B
Summary of Discovered Definitions

```
                                  MM/SF/MN                        MM/SF
LM  =     SM/MN     V             MM/SN/MN            V           MM/SN
                                  MN/SF/MN                        MN/SF

                                  MF/SM/MN                        MF/SM
LF  =     SF/MN     V             MF/SN/MN            V           MF/SN
                                  MN/SM/MN                        MN/SM

              OM/SF
              OM/MN/SF                        OM/LF
     NM  =    OM/MM/SN           V            OM/MN/LF
              OM/MM/SF                        OM/MM/LF

                                              OF/SM
              OF/LM                           OF/MN/SM
     NF  =    OF/MN/LM           V            OF/MF/SN
              OF/MF/LM                        OF/MF/SM

                                   ON/UM
                                   ON/UF
                                   ON/MN/UF
                                   ON/MM/UF
                         CN  =     ON/MN/UM (*)
                                   ON/MF/UM (*)
                                   ON/SN/PN
                                   ON/LM/PN
                                   ON/LF/PN

            PM/MN                                  PF/MN
            MN/XF                                  MN/XM
            MM/XF                                  MF/XM
            PM/ON/XF                               PF/SN/MN
XM  =       PM/OM/XF                   XF  =       MF/PM/MN
            PM/SN/MN (*)                           MF/PN/MN
            MM/PF/MN                               MN/PM/MN
            MM/PN/MN                               PF/PM/XM
            MM/PF/MN                               PF/ON/XM
```

TABLE 5B (cont.)

	ON/ON			OF/ON
DN =	ON/MN/ON		DF =	OF/MN/ON
	ON/ON/MN			OF/ON/MN
	ON/SN/ON (*)			OF/SN/ON (*)
	LM/PN			
	MM/UF			
	MN/UF			
UM =	MN/LF/PN		UF =	LF/PN
	MM/LF/PN			LF/MN/PN
	LM/MN/PN			
	PM			PF
PN =	MN/PF	V		MN/PM
	MM/PF			MF/PM
ON =	OM	V		OF
	OM/MN			OF/MN
	XM/PN			
	PM/PN			
	MN/GF			PF/PN
	MM/GN			MN/GM
	MM/GF		GF =	MF/GN
	XM/MN/PN			MF/GM
	PM/MN/PN			XF/PN
	PM/ON/GN			XF/EF/GM
	PM/ON/GF			
	PM/OM/GN			
	PM/OM/GF			
GM =	PM/SN/PN			
	MN/PF/PN			
	MM/PN/PN			
	MM/PF/PN			PN/PN
	MN/XF/PN		GN =	PN/MN/PN
	MM/XF/PN			PN/SN/PN
	XM/EF/GN			MN/PN/PN
	XM/EF/GF			

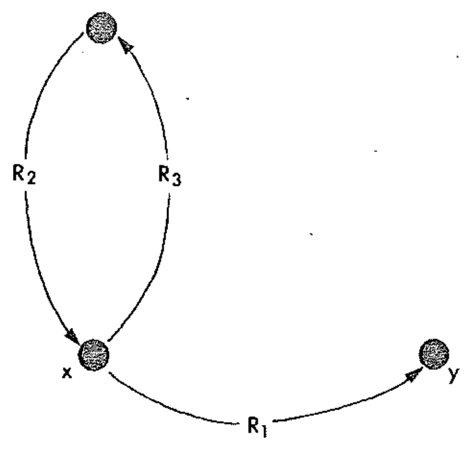
TABLE 5B (cont.)

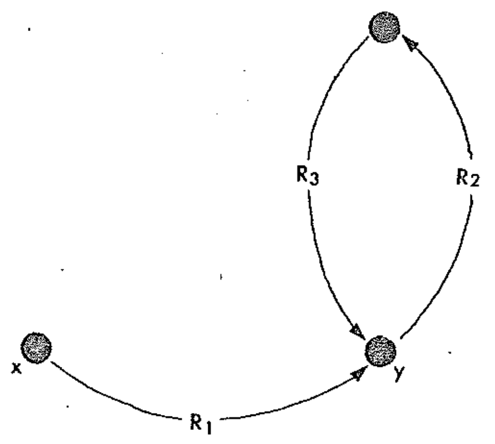
EM =	MN/OF	EF =	MN/OM
	MM/ON		MF/ON
	MM/OF		MF/OM
	MN/OF/MN		MN/OM/MN
	MM/ON/MN		MF/ON/MN
	MM/OF/MN		MF/OM/MN
	MN/SF/ON		MN/SM/ON
	MN/SF/OM		MF/SN/ON
	MM/SN/ON		MF/SM/ON
	MM/SN/OM		
	MM/SF/ON		
	MM/SF/OM		
	PM/ON/OF		

Discovering Recursive Definitions

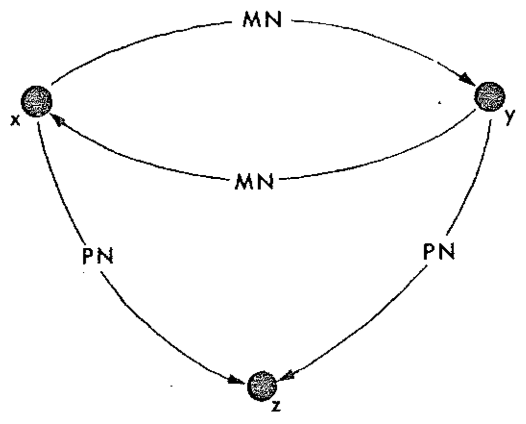
The above set of definitions has been generated under the constraint that the relation being defined is not allowed to appear in its own definition. This restriction is established by responding "NO" to the query: "RECURSIVE?". There are, however, reasons to permit non-trivial (i.e., R = R) recursive definitions. These can be appreciated by remembering that definitions are conjectured on the basis of finding paths that bridge groups of 2-tuples. Since only paths containing no loops are permitted, any path underlying a recursive definition necessarily involves a different aspect (2-tuple) of the given relation than the one currently being bridged.

Otherwise, the recursive definition would involve a loop on either the first or second node of the 2-tuple (see below).





However, both of these cases are blocked. A situation in which non-looping recursive definitions exist is exemplified by the Parent relation:



In this case we have "PN = MN/PN". For (x, PN, z) we use the path through (y, PN, z) and vice versa. This kind of recursive definition is of interest since it involves a special kind of redundancy. One portion of the given relation, used in conjunction with other relations, overlaps another portion of the given relation. As such, these definitions - if known - could be used in a different setting to fill out missing parts of the extension of the given relation (3), and hence are intimately related to inference rules.

A situation quite similar to the above arises with the discovery of a transitive relation in that a relation R is transitive if R/R is a subset of R. Not every 2-tuple in R is required to be bridged by R/R, but any R/R must bridge some 2-tuple of R. If we allow recursive, non-disjoint, disjunctive definitions, then one disjunct can be R/R. Some examples of recursive definitions obtained on the same data base are listed below. We will see later how these particular definitions function as inference rules.

1) PN = MN/PN
2) PN = PN/SN V PM V PF
3) ON = ON/MN
4) ON = SN/ON V ON/MM V ON/MF

Discovering Rules of Inference

Using the examples of this and preceding chapters we shall now consider techniques for discovering rules of inference. We will concern ourselves only with inferential rules of a very restricted form. These rules will be either of the form:

1) R1/R2/.../Rn => R
(where R itself could be one of these Ri's)

or of the form:

2) $$|_{Ri} \; (R1/R2/\ldots/Rn) \; |_{Rj} \;\Rightarrow\; |_{Ri} \; R \; |_{Rj}$$

In the latter case, the relations under the vertical bar denote a required context before this rule can be applied. The similarity of these rules to context free and context sensitive grammars is not accidental. Indeed, in Chapter 7 we discuss how we can utilize these kinds of rules, in connection with a context free (or context sensitive) parser, to construct a question-answering system.

Considering rules of form 1 we understand them to mean:

For any x and y [(x,y) element of R1/R2/.../Rn =>
(x,y) element of R].

The discovery of such rules would appear to be straight-forward. We need merely locate a sequence of relations whose compositional extension is contained within the extension of R. The problem, however, lies not in finding such sequences but in finding "useful" sequences. Although we have no way of making this distinction precise, we realized that there were several ways to use the structure of the relations themselves in isolating potentially "useful" rules.

One of the most apparent ways to generate such rules of inference would be to discover definitions for a particular relation which turn out to be under-general when their defining subspace is enlarged (see Figure 5.4 - template 07600 - 376036). Such definitions are rules of inference because any conjectured definition is always checked for inconsistency (i.e., over-generality) over the entire data file and not over just the current subspace. In other words, any definition constructed over a subspace of the data is always checked to see if it contains any 2-tuples that are not in the extension of the given relation as defined over the entire space. Discovering under-general definitions is easily achieved by isolating a subset of the data and then discovering intensional definitions on this subset. We then note which of these definitions become invalidated as the subset is enlarged. The key problem is finding ways that the structure of a relation can, itself, delimit "interesting" subsets of its own extensions.

Determining Subspaces for the Discovery of Inference Rules

One way in which the structure of a relation can induce a natural partition on its extension is by possessing a disjunctive definition. This, of course, splits the relation's extension into groups of 2-tuples covered by the particular disjunctive terms. For example, the definition for Parent, mentioned earlier in this chapter, was:

```
PN =    PM              V       PF
        MM/PF                   MF/PM
```

The set of 2-tuples covered by the first disjunct are all the examples of Parents who are also husbands - or who are also males (note that in this context a male and a husband function equivalently). This division defines a subspace in which PM coincides with PN and thus leads to the rules:

```
PM => PN                        Father => Parent
MM/PF => PN             Husband/Mother => Parent
```

Similarly, with the other disjunct, we have:

PF => PN	Mother => Parent
MF/PM => PN	Wife/Father => Parent

Another way to utilize the structure of the relations in isolating subspaces stems from the co-occurrence of compositional sequences in a disjunct. Reconsidering the definition of Brother-in-Law:

```
                       MM/SF/MN             MM/SF
LM =   SM/MN   V       MM/SN/MN     V       MM/SN
                       MN/SF/MN             MM/SF
```

we examine the last disjunct which asserts that:

1) MM/SF = MM/SN
2) MM/SF = MN/SF
3) MM/SN = MN/SF

Line 1 suggests that in the "context" of "MM" the relations "SF" and "SN" function equivalently, or we could say that:

4) SF => SN
5) SN => SF,

when conditioned on the left with "MM". Clearly, the notion of "context" is quite important for, although the rule "SF => SN" could be independently discovered from the definition "SN = SF V SM", and hence would always be true, the rule "SN => SF" is, in general, false. Using the expanded form of this rule (i.e. (x, SN, y) => (x, SF, y)) we see that the rule is true whenever x is a female. One way to guarantee x's femininity is to require that x be in the range of the "Husband" relation. In other words, the Husband relation delimits a subset of the domain of "SF" and "SN". When restricted to this subdomain, these relations coincide.

A considerably more subtle example of the powers of contextual constraints is manifested in two definitions for "Nephew" that emerged from some data:

a) Nephew = Brother/Niece
b) Nephew = Son/Parent/Niece

The rule of inference which follows from this is:

```
Son/Parent|       => Brother
          Niece
```

At first glance, the right-hand contextual relation (Niece) seems quite unnecessary, for certainly a son of a parent is a brother. However, in exploring the "Son" relation with "the Monkey's Uncle" no such rule emerges. Since the path seeking process underlying the construction of LPS's forbids loops, the possibility of the "Son of a Parent" being himself is necessarily prevented. However, in checking the over-generality of this definition, the entire extension of "Son/Parent", which would include such cases of self-looping, is computed. This explains why the above rule did not survive when "Son" was explored. However, in the context of "Niece", it does survive because the reflexive (self-looping) cases of "Son/Parent" are necessarily male and hence fall outside this context!

The use of "context" is more general than mere delimitation of the domains or ranges of relations. This can best be appreciated from an algebraic veiwpoint. From this vantage we see that two compositional definitions are equivalent if the boolean multiplication of their adjacency matrices are equal. Equation 1 above asserts that A(MM) x A(SF) is equal to A(MM) x A(SN) (where A(R) means the adjacency matrix of R). Since all possible compositions of a set of relations forms a semigroup which need not have the cancellability property, we should not expect that A(SF) = A(SN) would necessarily follow from A(MN) x A(SF) = A(MN) x A(SN). However, two distinct matrices can be made equivalent by either pre-multiplying them by a given matrix - its left "context"- or by post-multiplying them by a matrix - its right "context". Rules of inference which utilize "context" in this way can be quite counter-intuitive. The following example illustrates this.

In the Uncle definitions, (previously mentioned in this chapter) we found that "Uncle = Brother-in-Law/Parent" and "Uncle = Husband/Sister-in-Law/Parent". At first glance we might think that: "Husband/Sister-in-Law => Brother-in-Law". Considering Figure 5.5, we realize that although (y, LF, z) is true, (x, LM, z) is not! However, the correct rule of inference requires:

$$\text{Husband/Sister-in-Law}|_{\text{Parent}} \Rightarrow \text{Brother-in-Law}$$

a right context of "Parent". Given the situation shown below in Figure 5.6, we see that if z is a parent to q then in fact z must be a spouse to w. Hence x would be a brother-in-law to w. In other words the right hand context of "Parent" forces the existence of w simply by asserting the existence of q.

We have illustrated various ways that both context free and context sensitive rules of inference can be found. We will return to this subject in Chapter 7 where we describe a system which can utilize either type of rule, although, as of this writing, no systematic exploration of the more complicated context sensitive rules has transpired. Instead, we have attended mainly to the context free rules which result from recursive definitions.

FIGURE 5.5
One Possibility

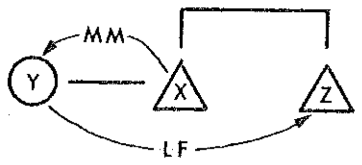
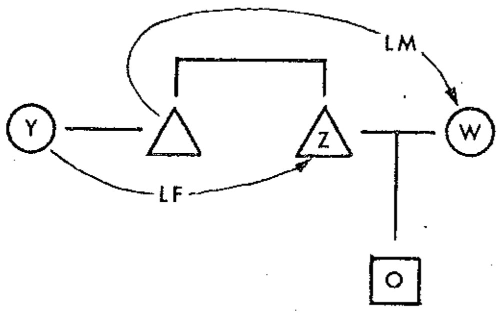


FIGURE 5.6
Another Possibility



Discovering Predicates and Attributes

Lastly, we shall consider examples of how partitions on the domain of objects can be induced by our system. The principle technique relates to the occurrence of minimal disjunctive definitions. We can easily get partitions on D*D from these. Unfortunately, we wish partitions on D. Let us consider, for example, four such definitions that have emerged from this data base. They are:

1)	Offspring	=	Son	V	Daughter
2)	Sibling	=	Brother	V	Sister
3)	Spouse	=	Husband	V	Wife
4)	Parent	=	Father	V	Mother

Each definition, in turn, induces a partition on a subset of D*D. These partitions have the interesting property of allowing the partition on the extension to be trivially collapsed onto a partition on their domains. More precisely, suppose T is a partition on D*D. Each of the above patterns has the property that a new partition T' on D can be defined by:

$$x \; T' \; x' \text{ iff there exists a } z \text{ s.t. } (x,z) \; T \; (x',z)$$

Although each of these partitions can be trivially reduced to a partition on their respective domains, we might wonder why they are of interest. A further examination of these four partitions (T'1, T'2, T'3, T'4) reveals that they can be combined into one partition covering nearly all of D. The partition that emerges on "most of" D is the one underlying the property of gender. If such a property were invented for this data, it could lead to a significant simplification of many of the context-sensitive rules of inference.

Not only disjoint disjunctive definitions lead to interesting partitions. An example of a non-disjoint definition that we have already considered is:

Uncle = Husband/Aunt or Brother-in-Law/Father

By considering the 2-tuples that hold for the second disjunct but not for the first, we can construct a partition for the property of being a bachelor. However, unless this partition was consistent with several other, we would have no particular reason for singleing it out.

In this chapter we have illustrated the functioning of "the Monkey's Uncle" over a fairly large collection of data. We have tried to show how various facets of the system can lead to many unexpected results.

Since all of these examples have stemmed from kinship systems, we include in Appendix 2 a series of examples which bear little similarity to this kind of data base. For example, we illustrate how "the Monkey's Uncle" characterized legal chess moves when given examples of such moves. We also explore how the system can be applied to state transition systems thus discovering how one transition, action or "verb" might be defined in terms of other actions. In fact this system is quite general; however, to appreciate its "discovery power" we chose domains whose semantics are especially familiar to the reader.

FOOTNOTES

(1) "The Monkey's Uncle" restricts the number of 2-tuples under investigation for any one relation to thirty-five. Of course, the relation can have a much larger extension, but at any one time the system will use, at most, thirty-five 2-tuples in the construction of its conjectures.

(2) The subgraph generated by a template is determined as follows. Consider all the 2-tuples corresponding to the "ones" of the template. For each such 2-tuple, form all the directed paths from its first component to its second. The desired subgraph is the union of these directed paths for all the specified 2-tuples. In Appendix 1 we discuss the algebraic significance to the subgraphs.

(3) This observation leads to certain complications: the discovery of such recursive definitions depends on the entire extension being present. Hence, this concept of redundancy is useful when extending these definitions to cover an expanded set of data or when data is purposely deleted from the original sample after the definition is discovered.

CHAPTER SIX

Identifying the Atomic Relations

Until now we have been primarily concerned with the discovery of various types of interrelationships. In this chapter, we discuss how those interrelationships which take the form of intensional definitions can be used to identify the relations in the data base which are so fundamental that they might be called atomic. Our definition of atomicity revolves around the concept of "generators". This concept is in turn based on the set of intensional definitions for the given collection of relations.

As discussed previously, the selection of atomic terms is predicated on finding a minimal set of relations which generates the remaining relations. Before presenting an algorithm for finding minimal generating sets, we want to look at a particular example. In this example we will see the critical role that circular definitions play in determining atomic sets. We will then develop a representation which accents circularity and establish some new theorems which concern the use of this representation. We will then discuss some heuristic procedures based on this representation and describe an interactive system which enables the user to selectively use these heuristics in determining atomic sets.

Suppose we consider the following set of six relations along with their intensional definitions. Given these definitions, we can determine a set of generators for each relation by noting the relations used in its definitions:

RELATION		INTENSIONAL DEFINITION	GENERATOR SET
R1	=	R2/R3 V R2/R6	[R2,R3,R6]
R2	=	R4/R5/R4	[R4,R5]
R3	=	R1/R2/R6 V R2/R1	[R1,R2,R6]
R4	=	R2/R3	[R2,R3]
R5	=	R4/R1/R4	[R1,R4]
R6	=	R1/R2/R4	[R1,R2,R4]

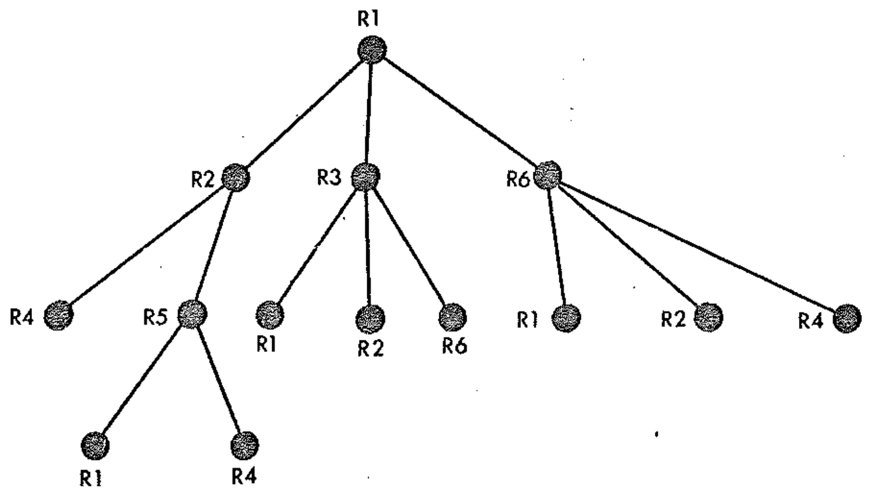
Any relation which can be intensionally defined has, a priori, a set of generators which consists of its defining relations.

The concept of a generating set can be naturally extended. For example, if we know that the set: [R2,R3,R6] "generates" R1 we can then ask whether the enlarged set consisting of R1 and R2, R3, R6 can generate any additional relations. In this particular case we note that the set [R2,R3,R6] not only generates R1 but R4 as well. We thus consider the set [R1,R2,R3,R4,R6] which in turn generates R5. In other words, by knowing just R2, R3, and R6 we can construct R1 and R4 and, by knowing these relations, we can then construct R5. Another way to view the above is to consider the set [R2,R3,R6] as directly generating R1 and R4 and indirectly generating R5. In speaking of what a set generates, we will mean both what the set directly and indirectly generates.

We might inquire if R1 has a smaller generating set than [R2,R3,R6]. To answer this question we must determine what relations are needed to generate R2, R3 and R6. Examining the intensional definitions for these three relations we find:

[R4,R5]	GENERATES	R2
[R1,R2,R6]	GENERATES	R3
[R1,R2,R4]	GENERATES	R6

The various ways that R1 can be generated is represented by the tree:



If we replace the relation R3 by its generators we discover another set of generators for R1, namely [R1,R2,R6]. Note that R1 now appears in its own generating set. This reappearance is due to the circularity of the intensional definitions. What this means is that if R1 is to be generated then R3 must be explicitly present since generation of R3 requires explicit knowledge of R1. In other words, if some subset of [R1,R2,R3,R4,R5,R6] is going to generate all the remaining relations then either R1 or R3 must be present in that generating set.

Once the concept of a generating set is understood the notion of atomicity follows closely. As described in Chapter 3, given a set of relations R along with an intensional definition for each, then any R', a subset of R, is a set of atomic relations if R' generates R and if R' has the least number of elements of any generating set. It follows immediately that if R' is an atomic set then no proper subset of R' can generate R. R', however, need not be unique since set inclusion defines a partial ordering on the set of all generating sets and the atomic sets are simply the minimal elements under this ordering. In yet a more profound way, atomic sets need not be unique. Note that atomic sets are necessarily based on the intensional definitions of the relations. If a particular relation has several intensional definitions, then the atomic set may vary accordingly. For example, if we define "Uncle" in terms of "Cousin" and "Cousin" in terms of "Uncle", then either "Uncle" or "Cousin" must appear among the atomic relations.

There is a straight-forward way to determine the atomic sets of relations: given the set R of relations, form all possible subsets of R and see which of these generate all the other relations in R. We then choose those generating subsets with the fewest numbers of relations in them.

There are 2^n possible subsets of a set with n elements. In order to determine whether or not a given subset generates all of R we might have to "expand" (see example below) that set n times. Therefore, the upper bound on this computation grows as $nx2^n$ for a set of n relations. Of course, one would test all the subsets of k elements before considering subsets of k+1 elements and hence, if there always existed a small generating set, this approach might be feasible, even if inelegant.

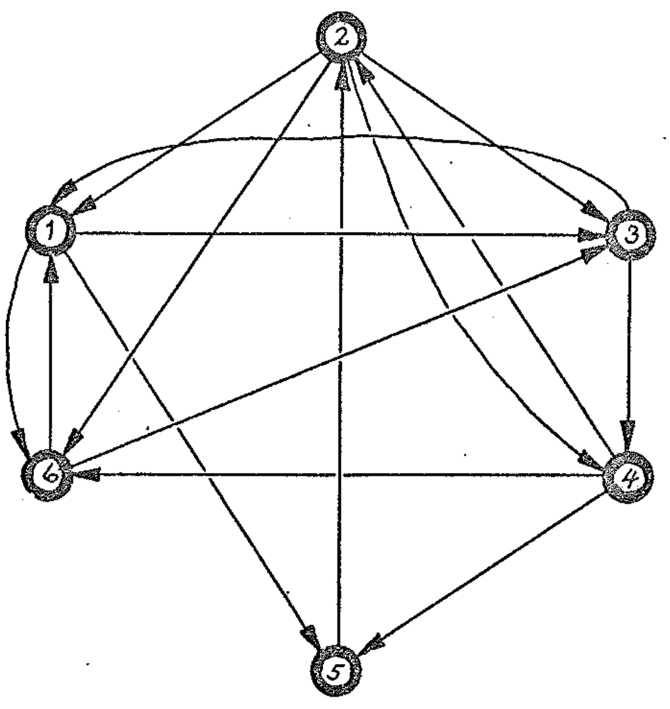
Let us consider this process as applied to the relations in Table A. We note that no singleton set could possibly generate R and, of all the fifteen two element subsets of R, only three could possibly generate R. They are: [R4,R5], [R2,R3] and [R1,R4]. We must first compute what each of these subsets can generate:

```
[R4,R5]  =>  [R2,R4,R5]  =>  [R2,R4,R5]
[R2,R3]  =>  [R2,R3,R4]  =>  [R2,R3,R4]
[R1,R4]  =>  [R1,R4,R5]  =>  [R1,R2,R4,R5]  =>
             [R1,R2,R4,R5,R6]  =>
             [R1,R2,R3,R4,R5,R6]  = R
```

The set [R1,R4] generates the entire set R and is the only subset of two elements to do so. R1 and R4 therefore comprise a unique atomic set for R.

Because of the combinatorial properties of this problem a representation was sought which might lead to a good heuristic solution. Clearly, we would like to factor the problem in such a way as to permit the determination of the importance of each relation, by itself, in generating R. Since the phenomenon of circularity plays a large role in determining which relations must be in generating sets, we sought a representation which would make these circularities explicit.

A directed graph can be constructed which captures the required information. We define this graph as follows: let each relation be a node and define its predecessors to be those relations (nodes) that directly generate that relation (node). For the six relations mentioned earlier we thus derive the following directed graph:



Intuitively, we may think of these directed graphs in the following manner. A given node is "triggered" either by being explicitly named as a member of a proposed generating set or by having all of its immediate predecessors "triggered". To compute what a set of relations (nodes) generates, simply trigger the initial nodes and compute what other nodes are consequently triggered. In other words, let R' be the initial set of nodes. Examine the relative complement of R' and see which nodes in this complement have all their predecessors in R'. Expand R' to include these nodes and then iterate this process until R' ceases to grow. If the resulting set includes all the nodes, that is, if every node has been triggered, then the original R' generates all of R.

A good approach to the problem of finding a minimal generating set for R would be to decompose this problem into that of finding subsets of R such that:

1) These subsets are pairwise disjoint,

2) Their union equals the set R

and 3a) Either the union of <u>all</u> of their minimal generating sets is also a minimal generating set for R

or 3b) The union of <u>some</u> of their minimal generating sets is a minimal generating set for R (and the sets comprising this union can be easily selected).

Let us now examine why the directed graph representation aids us in finding these subsets. Towards this end we introduce some graph-theoretic concepts and establish some new theorems.

Def: A strong component of a directed graph consists of a set of nodes, SC, such that for any two distinct nodes x and y which are elements of SC there exists a directed path from x to y and also from y to x.

Def: Given a directed graph G, a relation T can be defined on the vertices of G such that:

x T y <=> x and y are in the same strong component.

A well known theorem in graph theory asserts that T is an equivalence relation and hence induces a partition on the vertices of the given directed graph. In addition, this relation enables us to define a new graph:

Def: A condensation graph G' is formed from a directed graph G as follows:

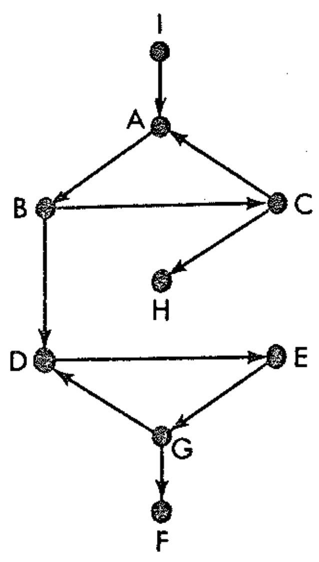
Let G = (V,E) and G' = (V',E'). Then V' = the partition classes of V with respect to T and (x',y') is an element of E' if there exists x and y such that x T x', y T y' and (x,y) is an element of E.

In other words, the condensation graph of G is formed by collapsing each strong component of G onto a representative node from that component. The successors of this node are the nodes (strong components) whose immediate predecessors are contained in the given strong component.

A condensation graph is necessarily acyclic since, if it contained a cycle, the distinct nodes constituting that cycle would form a strong component and hence would have been collapsed onto one node by T.

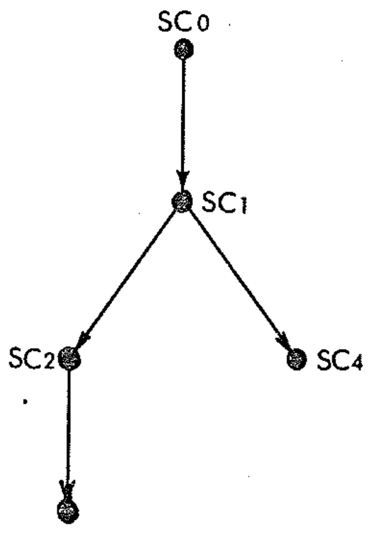
An example will help clarify the above concepts:

G:



Strong Components
SC0 = [I]
SC1 = [a b c]
SC2 = [d e g]
SC3 = [f]
SC4 = [h]

G':



Since G' is acyclic it represents a partial order whose minimal elements are the nodes of G' with in-degree zero.

Theorem: A set of nodes capable of generating G must have at least one node in each of the minimal strong components of G.

Informal Proof: Since each minimal strong component has no predecessors, no other nodes could cause it to be triggered.

Since any node of in-degree one is necessarily triggered whenever its predecessor node is triggered, we can collapse each of these nodes onto its predecessor. If this reduction is performed on G the only effect it has on G' is to collapse all nodes representing strong components of singleton elements in G onto their respective predecessors (except for minimal elements that have no predecessors, which we define as remaining unaffected). Let us call this reduced version of G': G*.

We develop the above terminology because we will show that the elements of G* are precisely the partition classes of R whose minimal generating sets have the desired properties. Before we formalize the above statement we need one key theorem which reveals the critical connection between circular definitions and strong components.

Theorem: Given a directed graph G = (N, E), which has at least two nodes and has exactly one strong component, then: a set of nodes $N_0 \subset N$ generates G if and only if every elementary cycle in G has at least one node in N .

Proof:

(If)

Let N_0 contain at least one node from every elementary cycle in G and let N^1 be the set nodes generated from N_0. We assume $N^1 \subsetneq N$ and then derive a contradiction: Since N^1 is properly contained in N there exists an $n_w \in N - N^1$. Since n_w lies in N which by assumption is a strong component of at least two nodes, then n_w must have at least one predecessor--call it n_{w-1} which must also be in $N - N^1$ for otherwise n_w would have been generated. Since the graph, by assumption, consists of only one strong component, the identical argument can be repeated indefinitely yielding a sequence n_w, n_{2-1}, n_{w-2},... all of which are contained in $N - N^1$. However, since the set $N - N^1$ is necessarily finite, this infinite sequence must involve a cycle, but this contradicts the hypothesis that every cycle must have at least one node in N_0.

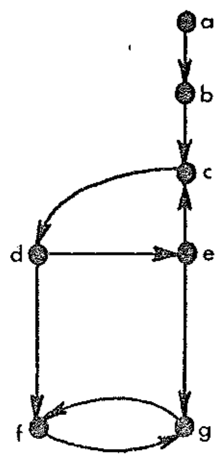
(Only if)

Given any set N_0 we can partition the set of elements generated by N_0 as follows: Consider the sequence of disjoint sets $N_0, N_1, N_2, \ldots, N_k$ such that the set N_{i+1} is directly generated by $\bigcup_{j=0}^{i} N_j$ -- i.e. for any node in N_{i+1} all of its predecessors are contained in $\bigcup_{j=0}^{i} N_j$ and for no smaller i is this true. This sequence can be extended until some N_k is reached which is empty. We assume that N_0 generates N and that there exists some cycle C in G which is disjoint from N_0. Since N_0 generates G we can construct a generating sequence $N_0, N_1, \ldots, N_k$ such that $\bigcup_{j=0}^{k} N_j = N$. This means that there exists a smallest number $i \geq 1$ such that $N_i \wedge C \neq \emptyset$ but that $\bigcup_{j=0}^{i-1} N_j \wedge C = \emptyset$. Let $n \in C \wedge N_i$. Since $n \in N_i$ all n's predecessors must be contained in $\bigcup_{j=0}^{i-1} N_j$, but at least one of n's predecessors, say n*, is contained in C. This means that there exists N_k, $k < i$, such that $n^* \in N_k$ and hence $N_k \wedge C \neq \emptyset$ which contradicts the hypothesis that N_i was the first set to intersect C.

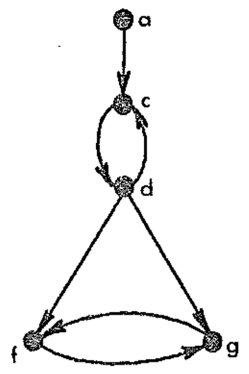
From the above theorem we see that a minimal generating set for a given strong component will be the smallest set of nodes which covers all the cycles of that component.

We have thus described a technique for finding a minimal set of generators for a strong component. Forming the union of these minimal sets for all the strong components provides us a set of generators for the entire graph. However, this resulting set is not necessarily minimal, for we have not taken into consideration possible interactions between these strong components. More precisely, we wonder if generating all the nodes in one strong component could effect any of its successor components. An example will illustrate how, in fact, "triggering" one component can lead to another component being triggered:

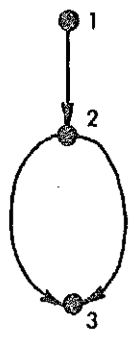
G



Reduced G



G*



From G^* we proceed to find minimal generating sets. Since there is one strong component - [a] - with in-degree of zero, we first choose it, and, since it consists of only one node from G, we necessarily choose that node. We note that, on triggering the chosen node, no other nodes in the reduced G are fired. We therefore examine the successor component of [a] which is [c,d]. Either c or d can generate this entire component. Suppose we choose c. We observe that c, of course, triggers d and that d can, in fact, trigger both f and g. In other words, generating [c,d] indirectly generates [f,g]. This example shows that there can be a positive interaction between strong components. To take advantage of this interaction, simply start with those strong components with in-degree zero and work downward via each component's successor components.

In other words, the original problem has now been decomposed into finding minimal generating elements for the strong components of the reduced G. This factoring is significant since:

$$2^N \gg \sum 2^{N_i} \quad \text{Where} \quad \sum N_i = N$$

Nevertheless, the problem of finding the minimal set of nodes that covers all the elementary cycles in a strong component of k nodes still grows as 2^k. This would have been discouraging except for the fact that we can invoke a heuristic for finding a minimal cover which, although simplistic for a general minimal covering problem, is quite reasonable for the original problem. In other words, we feel we have found a representation for the original problem which transforms it into a new space where a relatively poor heuristic on this new space becomes relatively powerful on the original space.

Our heuristic procedure is as follows. Consider all elementary cycles of length k. Determine a covering set for these cycles by first choosing the node in common with the greatest number of them, then discard the cycles in common with the chosen node and choose the most common node again until all cycles of length k have been covered. Add the selected nodes to the generating set and then remove them from the graph. Test to see if the resulting graph is acyclic. If it is not, then increase k by 1 and repeat until the graph becomes acyclic. If, at any time, there is more than one node in common with the greatest number of cycles, carry all choices along until the tie can be resolved.

Example: Consider the directed graph underlying the original example in this chapter. It is a strong component which cannot be reduced (i.e., it has no nodes of in-degree

1). There are no cycles of length one but there are three cycles of length two. We next determine which node has the greatest number of cycles passing through it - i.e., node 1. We eliminate all cycles involving this node and choose the next most common node. Nodes 2 and 4 both have one cycle in common - a tie. We therefore have two potential generating sets:

[1,2] and [1,4].

We therefore remove nodes 1 and 2 and find that the graph is still cyclic. This means that the set [1,2] would necessarily require at least one more node. On the other hand, removing nodes 1 and 4 removes all cycles and hence [1,4] is conjectured as a minimal generating set. This example is given detailed consideration in Appendix 8.

There are, of course, numerous variations to the above heuristic. One of these would be to count all the elementary cycles (as contrasted with only elementary cycles of length k) passing through a given node and then remove the node with the largest number. We would then repeat this process until that component becomes acyclic.

Instead of constructing a non-interactive program to ascertain minimal (or nearly minimal) generating sets we decided to extend our symbiotic system so as to enable the user to invoke additional combinatorial routines to help him isolate these sets. Such a system should allow the user to experiment with different heuristic approaches to the particular problem at hand and to select the techniques best fitted to the structure of his problem space. For example, he might first try counting all the elementary cycles passing through each node. If one node had vastly more elementary cycles than other nodes, he might simply remove that node. If, however, all the nodes had about the same number of elementary cycles passing through them, then he might revert to counting elementary cycles of particular lengths.

Since there was no need for intimate interaction between the system for generating intensional definitions and the one for ascertaining the minimal generating sets (and since the PDP-9 was no longer available) the following package of functions was coded in Stanford LISP 1.6 for the PDP-10. These routines were designed to utilize the interactive nature of this version of LISP and to enable the user to quickly synthesize a particular strategy. Appendix 7 contains a typical session with this package as applied to the above example. What follows here is a brief description of this system.

ECYC consists of a collection 14 LISP functions which can be invoked as direct or indirect procedures. Most of these procedures utilize the array features of LISP 1.6 to help gain speed in "marking" and "unmarking" nodes in the graph. The package consists of five groups of routines.

The first group aids the user in building an array structure to represent the directed graph. These routines expect the graph to be presented as S-expressions: (R1 (R2 R3 ...) R2 (...) ... Rn (...)) where the sublist following each Ri is either a list of predecessors or successors. These routines, then, transform this S-expression into a LISP array - the form used by most of the other functions. The second group of functions displays local information (in-degrees, out-degrees, etc.) about the graph so that the user can have some independent verification of the inputted graph structure. The third group allows the user to modify the graph on a temporary basis. He can "block" any node or list of nodes of a graph and thus implicitly define a new subgraph. This modification is meant to be temporary and the user can efficiently restore the graph to its original form or any prior form. In addition, the user can define and name a new permanent subgraph. The fourth class of functions enables the user to execute several tests on both implicitly and explicitly defined subgraphs. Of special interest are two predicates which test if a subgraph is cyclic or if two nodes are directly connected.

The last group of functions performs most of the combinatorial work. Here, we have functions which 1) compute all the strong components of a graph, 2) determine the number of elementary cycles passing through a given node and 3) display all the elementary cycles. Additional functions compute only elementary cycles of a fixed length and determine how many of these pass through a given node.

Combining these functions in various ways enables one to quickly compose either complex direct commands or new programs. Again, our philosophy has been to construct a set of tools to enable the user to more effectively explore his data or, in this case, his collection of intensional definitions.

CHAPTER SEVEN

A Simple Question-Answerer

The previous six chapters have concerned techniques for adducing from a network of interconnected facts certain structural properties theretofore unknown about those facts. In this chapter we will again concentrate on this kind of network but with a different end in mind. If we can form abstractions by examining paths in a network and thus formulate a theory about the data represented by the network, then by utilizing that theory we should be able to make certain predictions. Such predictions might, for instance, suggest missing or deleted links in the network on which the theory is based.

For example, suppose "the Monkey's Uncle" has discovered (as part of its theory about some given data) the rule: "a husband of a parent is a father". If a new set of facts is then encountered such as:

```
(Jack, Husband, Judy)
(Judy, Parent, Jim)
```

then a simple prediction could be that Jack is the father of Jim. This kind of prediction is of special interest to us since it parallels the tasks performed by deductive question-answering systems.

The primary goals of this chapter are to explore more fully the nature of these predictions and to show how the type of theory created by "the Monkey's Uncle" can be utilized by a question-answering system. To satisfy these goals we have constructed a special purpose question-answering system.

The approach we have taken in the design of this system is quite unorthodox. It differs significantly from both the highly popular general purpose question-answering systems which utilize predicate calculus representations of their theories and from the relational compiler approach underlying the TRAMP system. Following the inner workings of this system will not only provide substantial insight into the kind of structure "the Monkey's Uncle" is good at capturing but also elucidate some of the fundamental problems of question-answering systems.

Foundations

In the following sections we will concentrate on how our question-answering system performs deductions. We will outline a series of examples, each of which becomes progressively more complicated, and show how our system handles each case. Through these examples the reader will be able to understand the subtle differences and similarities between intensional definitions and rules of inference (both of which are produced by "the Monkey's Uncle"). In addition, the problems surrounding "possibility" questions (i.e. Can x be y's R?) will be explored.

The basic theory behind our system is relatively straightforward. Suppose that we are given a network of facts and a particular relation, such as grandparent (GN), whose semantic meaning is expressed by the intensional definition:

(x, GN, y) iff (x, PN/PN, y)

(i.e., x is the grandfather of y if and only if x is the parent of a parent of y). In terms of the framework established in the first five chapters (and supposing that the data base is complete in the sense that all possible facts are explicit), there must be for any particular fact (x, GN, y), by definition, a labeled path sequence (LPS) of the form PN-PN which bridges that particular triple. Likewise, as noted earlier, every LPS bridging the fact (x, GN, y) need not necessarily coincide with the meaning of "Grandparent". For example, some LPS's might imply concepts more general than "Grandparent" (as would the path

Parent-Uncle). Still others might be undergeneral and fail to capture all the possible variations inherent in the meaning of "Grandparent". We recall that one of the principle tasks of "the Monkey's Uncle" is to sift out all such LPS's, leaving behind only those whose meaning coincides with that of the given relation.

By applying "the Monkey's Uncle" to each relation contained in a given network, definitions and/or rules of inference are discovered which preserve the meaning of the particular relation under study. For example, in studying "Parent", the recursive rule that the spouse of a parent is a parent was discovered (i.e. MN/PN => PN).

Since this rule preserves the meaning of "Parent", applying it to the simple LPS "PN-PN" generates the new LPS "PN-MN-PN" which must also mean "Grandparent". Applying all the applicable recursive rules to a definition of "Grandparent" generates a potentially infinite collection of LPS's, each of which implies the concept of "Grandparent" (see Table 7A for further expansion).

To determine if a particular person is the grandparent of another person, we could enumerate all the LPS's bridging these two people and determine if any one of these labeled path sequences were contained in the potentially infinite

TABLE 7A
Sentences Equivalent to "Grandparent"

Partial List of Sentences Meaning "Grandparent"	Inference Rules Which, When Inverted, Can Be Used to Generate the Language
GN	
PN - PN	
PN - MN - PN	1) PN/PN => GN
MN - PN - PN	2) PN/MN => PN
MN - PN - MN - PN	3) MN/PN => PN
PN - SN - PN	4) PN/SN => PN
PN - SN - SN - PN	5) SN/SN => SN
PN - ON - PN - PN	6) ON/PN => SN
.	
.	
.	

set of LPS's generated from the definition of "Grandparent".

This potentially infinite set of LPS's might be viewed as a language in which each sentence (LPS) has the same meaning: that of the starting state (i.e. relation) from which all of these sentences were generated. The rules of inference may likewise be viewed as a grammar characterizing that language. Determining whether or not a particular sentence (LPS) is contained in this language is precisely the kind of task which is performed by a recognition parser. In other words, suppose we have a collection of context free grammar rules which can be applied to the intensional definition of a given relation and thus generate all the sentences which have the same "meaning" as that relation. These rules can be inverted and used to recognize whether or not a particular given sentence has the meaning of that relation.

The barest sketch of a question-answering strategy thus emerges: to determine if a particular (x, y) is contained in a given relation R, first isolate an LPS which starts at x and terminates at y. Then parse this sentence (LPS) to see if it can be reduced to the relation R.

Admittedly, the above exposition has glossed over numerous logical problems, but before delving into these we think it is important to provide an intuitive explanation of the connections between this approach to question-answering and language recognition. In fact, we have often been struck by the numerous similarities between the problems inherent in achieving natural language deep structures and those which surround deductive question-answering systems. Even at the most superficial level, a transformational parser attempts to establish the equivalence of two disparate surface strings by mapping both of these strings onto the same deep structure and thus showing that these two surface strings have, in essence, the same meaning. We will soon see that the behavior of our system mimics, to some extent, that of a transformational parser. Our rules are, of course, only context free (or in some cases context sensitive) but nevertheless the application of these rules may be understood as transforming part of the data base network.

We now proceed to discuss the details of our question-answering system.

The Data Base

The data base for this system is similar to the data base used by "the Monkey's Uncle" (i.e. (x, R, y) triples) except that now, of course, the data base will be sparse or missing numerous "facts" (if the base contained all possible facts about the given "world" then no deductive capability whatever would be required of our question-answering system). The data base can contain both atomic and non-atomic relations.

Crucial to our system is that for any (x, R, y) triple stored in the data base, its converse must also be explicitly stored. This requirement is satisfied by our STORE routine which determines the converse of each relation and builds the appropriately labeled backwards link. This reciprocity assures us that whenever we wish to prove that a particular (x, R, y) fact is implicitly contained in the base we need only search along directed paths from x to y in order to obtain all "relevant" information.

The result of such a search is an LPS, or a sentence, connecting x to y. As with "the Monkey's Uncle", the search algorithm returns only paths which do not involve loops. The importance of not allowing loops is crucial. It provides us with the basic tool for handling axioms or rules that would otherwise require an explicit statement of inequality within the rule. For example, suppose we had the small set of facts:

1) (Jack, Brother, John)
2) (John, Brother, Jim)

and we asked the question: "Is Jim Jack's brother?". Calling our search routine would produce the sentence: "Brother-Brother" which linked Jack to Jim and our task would be to determine if this sentence implied "Brother". At first glance, this sentence would trivially seem to imply "Brother" since Brotherhood is a transitive relation (i.e. Brother/Brother => Brother). However, this rule is not precisely true since Brotherhood is only "almost" transitive and what we really mean is that:

$$(x)(y)\,\exists z\,[(x, \text{Brother}, z) \;\&\; (z, \text{Brother}, y) \;\&\; x \# y) \Rightarrow (x, \text{Brother}, y).$$

In other words, we must explicitly check to see that x and y are not equal before we perform the reduction of "Brother-Brother => Brother". Because our search algorithm does not permit loops, the check is implicit in the generation of a sentence and therefore alleviates us from having to check for this property (we will later see that there are some additional problems with "equality" that are not so easily disposed of). Note that "the Monkey's Uncle" discovers rules of inference consistent only with this interpretation, i.e., when it unfolds the rule "SN/SN => SN" the inequality of the domain and range elements is assumed.

<u>Rules of Inference</u>

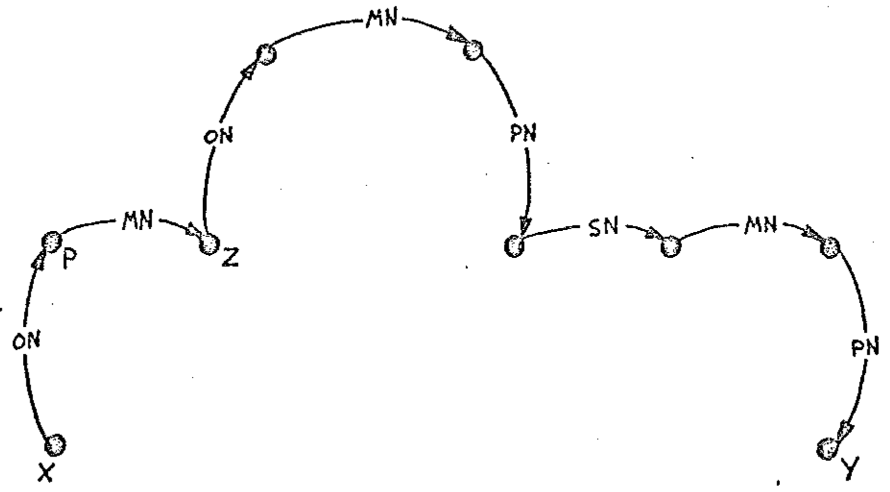
The rules of inference for this system are of the form generated by "the Monkey's Uncle" -- that is to say, any rule may be viewed as a context free (another more experimental version of this system also allows context sensitive) phrase structure rewrite rule. Of noteworthy importance is that our system needs only inference rules interrelating the "atomic" relations and for the remaining relations it needs only their intensional definitions (multiple definitions are easily handled and, in some cases, can increase the system's efficiency). Table 7B illustrates some of these rules.

TABLE 7B

Partial List of Kinship Axioms

PM => PN	OM/PN => SM
PF => PN	SM/SN => SM
OM => ON	OM/MN => OM
OF => ON	SM/ON => OM
SM => SN	PM/ON => MM
SF => SN	MM/PN => PM
MM => MN	PM/SN => PM
MF => MN	OF/PN => SF
ON/PN => SN	SF/SN => SF
SN/SN => SN	OF/MN => OF
ON/MN => ON	SF/ON => OF
SN/ON => ON	PF/ON => MF
PN/ON => MN	MF/PN => PF
MN/PN => PN	PF/SN => PF
PN/SN => PN	

Let us consider an application of these rules. Suppose we wish to determine if x is the cousin of y. Our first step is to search the data base for a directed path from x to y. In so doing, suppose we discover the following path:

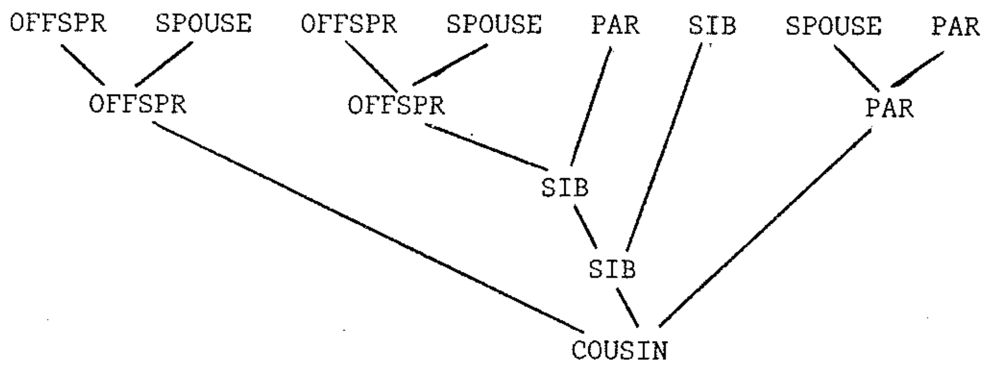


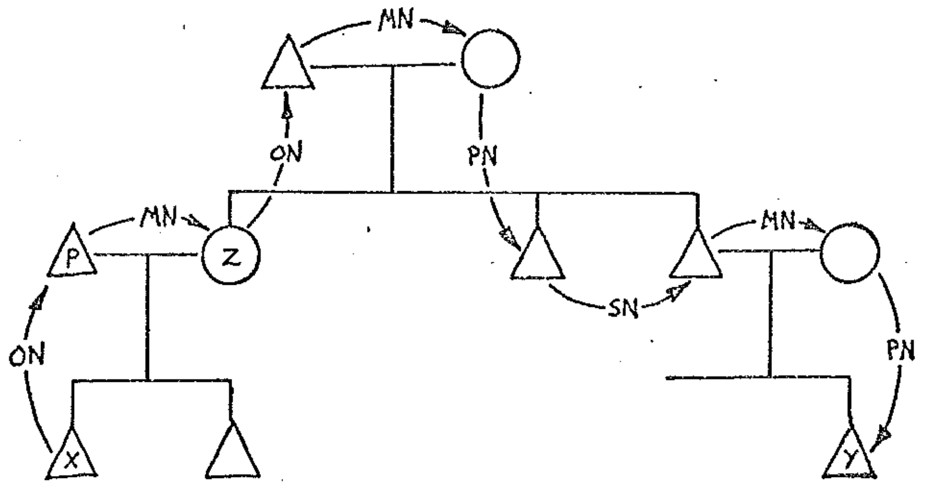
which translates into the sentence:

ON MN ON MN PN SN MN PN

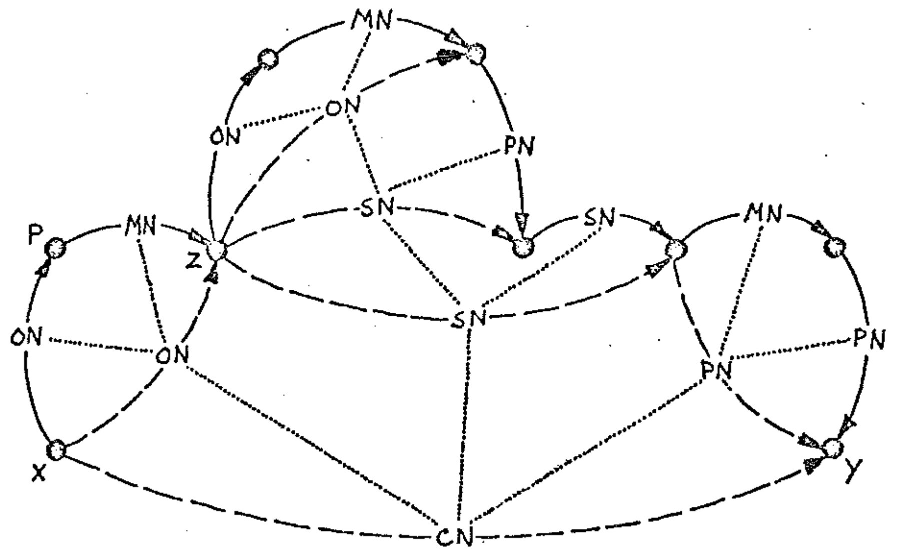
(Offspring Spouse Offspring Spouse Par Sib Spouse Par)

We must attempt to reduce this sentence to determine if it implies the cousin relation. Retrieving the intensional definition of cousin (ON SN PN = CN) we adjoin it to the axiom list (see Table 7B) as (ON SN PN => CN). We can now parse this sentence as shown below. Since the sentence can be reduced to "CN" it implies that x and y are indeed cousins. To help in understanding the semantics of this example we provide a family tree consistent with this path.





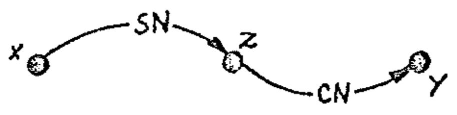
Examining just the application of the first rule, we see that rewriting "ON MN" as "ON" is tantamount to specifying an implicit new link between x and z, i.e., if x is the offspring of p and p is the spouse of z, then x must also be the offspring of z. This fact is true regardless of whether or not (x, Offspring, z) is explicitly contained in the data. Pictorially we therefore have:



In the diagram above, the dashed lines are implicit (added) links and the dotted lines from these implicit links define which links caused their generation (1). Note that a "complete" data base would have all these implicit links represented as actual facts.

This example seems more complicated than it really is. Note that every arc underlying the initial sentence is an atomic relation. This means that the sentence is already tailored to coincide with the rules of inference. Surely it is possible for a path to be generated which involves a non-atomic relation. What will happen in such a case? If the rules of inference do not include a reduction of the given non-atomic relation, then that part of the sentence must be irreducible.

Again, suppose we are trying to determine if x is the cousin of y and, in searching our data base, we discover:



(x, Sibling, z)
(z, Cousin, y)

Since a sibling of a cousin is a cousin, one might expect that if our rules of inference are complete, they should contain the rule:

SN CN => CN.

However, we have agreed earlier that knowing the definitions of non-atomic relations in terms of the atomic relations (which are characterized by the rules of inference) should free us from having to spell out all the rules of inference governing all possible interactions among the relations.

In the previous example we used the definition:

CN = ON/SN/PN

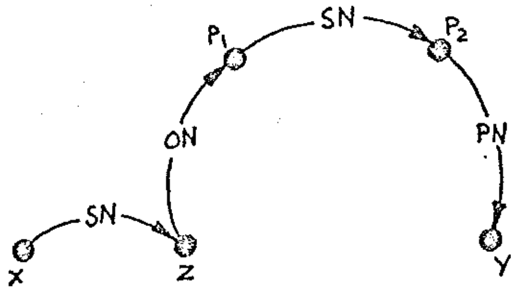
to arrive at the rule:

ON SN PN => CN.

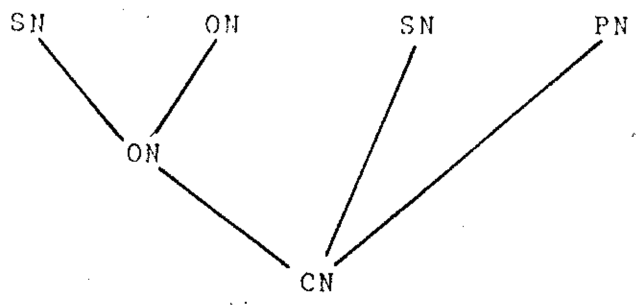
Since defintions actually represent two-way implications, we know that if x is the cousin of y there must exist at least two other people (i.e., p_1 and p_2, which are not necessarily referenced in the data base) forming a hypothetical path:

ON-(P_1)-SN (P_2)-PN.

In other words, given the path "SN-CN" this path can be mythically expanded into: "SN-ON-SN-PN", which reflects the following situation:



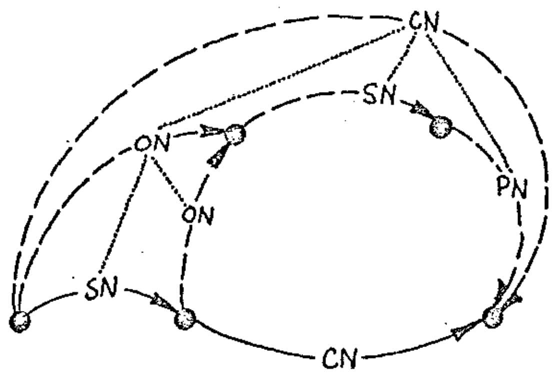
Since this expanded path now involves, by definition, only atomic terms the rules of inference should be directly applicable. In fact, the following parse exists:



thus showing that a sibling of a cousin is in fact a cousin and moreover <u>without</u> ever requiring an explicit rule which states:

SN CN => CN.

In terms of the above network, this parse represents the following implicit links (dashed lines):



This method of analysis circumvents some of the limitations of question-answering systems which are forced to bind all of their existential variables in a given rule of inference to explicit data. For example, in the TRAMP system, even with the inference rule "SN/ON => ON", the above reduction could not be achieved unless the data base explicitly mentioned p_i. Unfortunately, the problems inherent in not knowing the identity of the person through which these hypothetical paths pass are not so easily solved. The following set of examples sheds considerable light upon the nature of these problems and provides us with an introduction to questions which refer to possible states of the world.

The Handling of "Can" Questions

A characteristic of all the above examples is that each question has a unique answer, i.e., there is no uncertainty as to whether the sibling of a cousin is a cousin. If, however, we pose the question:

"Is a cousin of a cousin a cousin?"

a slightly different situation is encountered. A cousin of a cousin might be a cousin, but then again might not be. Additional information about the particular case is clearly required. In other words, the rule:

CN/CN => CN

leads sometimes to incorrect conclusions and therefore does not qualify as an inference rule. Expanding each occurrence of cousin by its definition indicates where the uncertainty resides. The expanded path of CN-CN is:

ON-SN-PN-ON-SN-PN

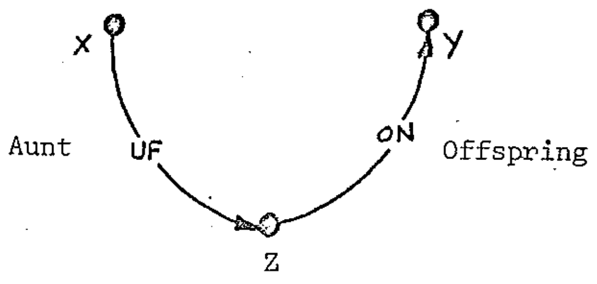
which cannot be parsed any further than:

$$\begin{tikzcd}
ON & SN & PN \arrow[dr, dash] & & ON \arrow[dl, dash] & SN & PN \\
 & & & MN & & &
\end{tikzcd}$$

Suppose we make explicit the existentially quantified persons that occur in the definition of cousin, i.e.:

$$\begin{tikzcd}
x\ ON\text{-}(p_1)\text{-}SN\text{-}(p_2)\text{-} & PN \arrow[dr, dash] & \text{-}(p_3)\text{-} & ON \arrow[dl, dash] & \text{-}(p_4)\text{-}SN\text{-}(p_5)\text{-}PN\ y \\
 & & (p_2\ MN\ p_4) & &
\end{tikzcd}$$

This reduction makes the assumption that persons p_2 and p_4 are distinct and therefore the inference rule asserts that they are married. However, we have no guarantee that this assertion is correct since a parent of an offspring would be a spouse only if no loops had been permitted in the generation of this sentence. In this particular case, the expanded sentence was not generated by an explicit traversal of the data graph. Thus, although the original sentence was generated under this restriction, the mechanism which grants such a guarantee has not been invoked on the expanded sentence. A moment's reflection reveals that when a cousin of a cousin is still a cousin, a loop is involved; p_2 equals p_4 and PN-ON reduces to the identity. An example taken from Lindsay (2) better illustrates this problem. Suppose we encounter the path:



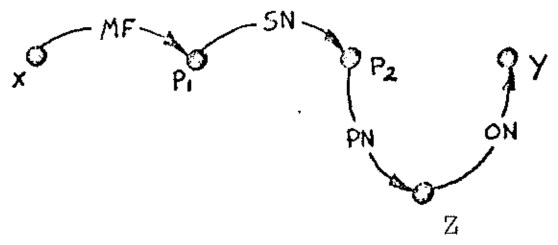
Can x be the sister of z?

The underlying structure of these two examples can best be understood by considering "possible states of the world". Given the two facts:

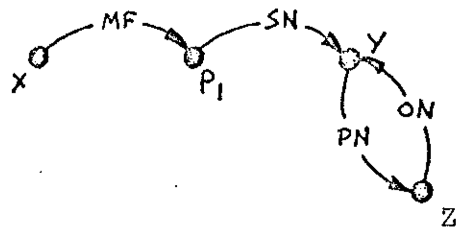
(x, Aunt, y)
(y, Offspring, z)

there are four possibilities consistent with these facts as shown below:

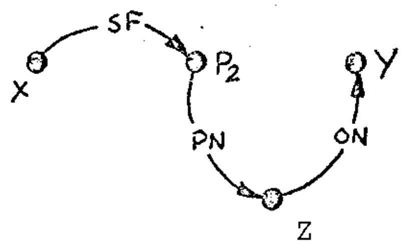
STATES



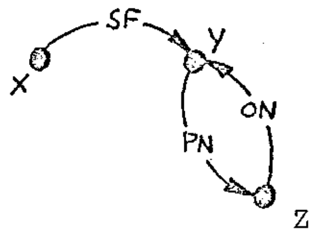
1) x # p_1
   y # p_2



2) x # p_1
   $y = p_2$



3) $x = p_1$
   $y \neq p_2$



4) $x = p_1$
   $y = p_2$

Given no additional information, we have no way of deciding which state (1,...,4) reflects the actual situation. Expressed differently, our data base certainly contains explicit references to x and y, but might contain no information about the existentially quantified persons p_1 and p_2. Lacking this information, it is impossible to determine if y equals p_2 or if x equals p_1.

Nevertheless, there clearly exists a state in which the aunt of an offspring is a sister (i.e. state 4); a similar analysis holds for the a cousin of a cousin. What we would like is some natural way to handle these possible states in case no further information is forthcoming. We emphasize that this problem occurs only upon encountering a path which involves a non-atomic relation for which there is no available additional information.

One possible approach to this problem might be to construct a purposely ambiguous grammar (a set of inference rules) which would give multiple parsings of a sentence with each parse reflecting a possible state of the world. This has been achieved by augmenting the grammar to incorporate the possibility of loops. Whenever a path is expanded a special flag "F" is placed around the inserted definition. For example, the expansion of CN-CN is:

```
ON SN PN F ON SN PN
```

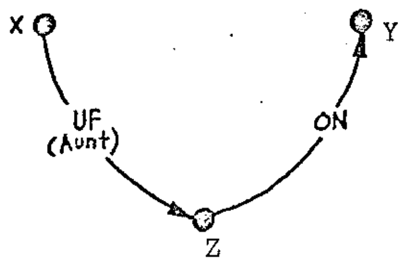
The augmented grammar contains rules of the form:

```
MN F MN => I Married (I is the Identity Relation)
PN F ON => I
PM F ON => I Male
PF F ON => I Female
   F    => Off
     .
     .
     .
```

(This last rule permits the flag to be turned off so that the initial set of inference rules may be applied)

The application of any of these rules introduces an implicit loop in the derived data graph and thereby allows us to implicitly cover all the possible consequences of a particular sentence.

An example will help to clarify how these augmented rules introduce implicit loops. Let us reconsider the problem of determing if an aunt of an offspring can be a sister. The initial data path was:



or UF-ON.

Expanding this path we encounter the following two sequences:

Sentence 1: MF SN PN F ON
Sentence 2: SF PN F ON

Restricting our attention to the latter sentence, we find two possible reductions. These are listed on the next page along with the data paths consistent with them.

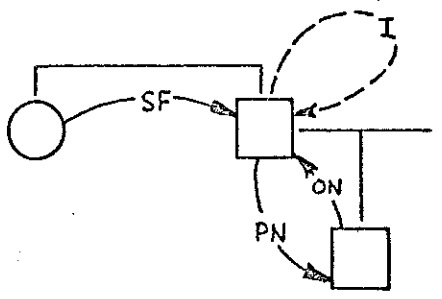
The only basic question that remains unexplored is how to derive conclusions which are certain but which involve sentences containing non-primitives. For example, suppose

=> State 4

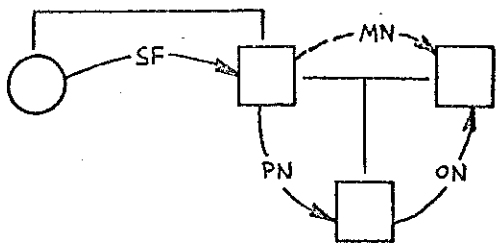
(I functions as both left and right identity)

$$\begin{tikzcd}
 & & \mathrm{SF} \arrow[dll, dash] \arrow[d, dash] & & \\
 & & \mathrm{I} \arrow[dl, dash] \arrow[dr, dash] \arrow[drr, dash] & & \\
\mathrm{SF} & \mathrm{PN} \arrow[ddr, dash] & & \mathrm{F} \arrow[d, dash] & \mathrm{ON} \arrow[ddll, dash] \\
 & & & \mathrm{Off} & \\
 & & \mathrm{MN} & &
\end{tikzcd}$$

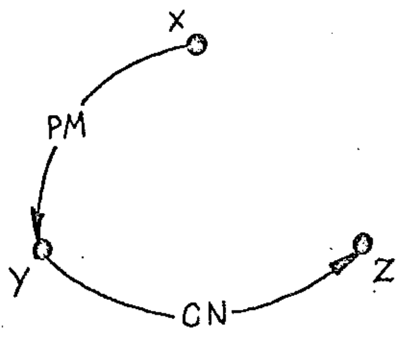
=> State 3



VS.



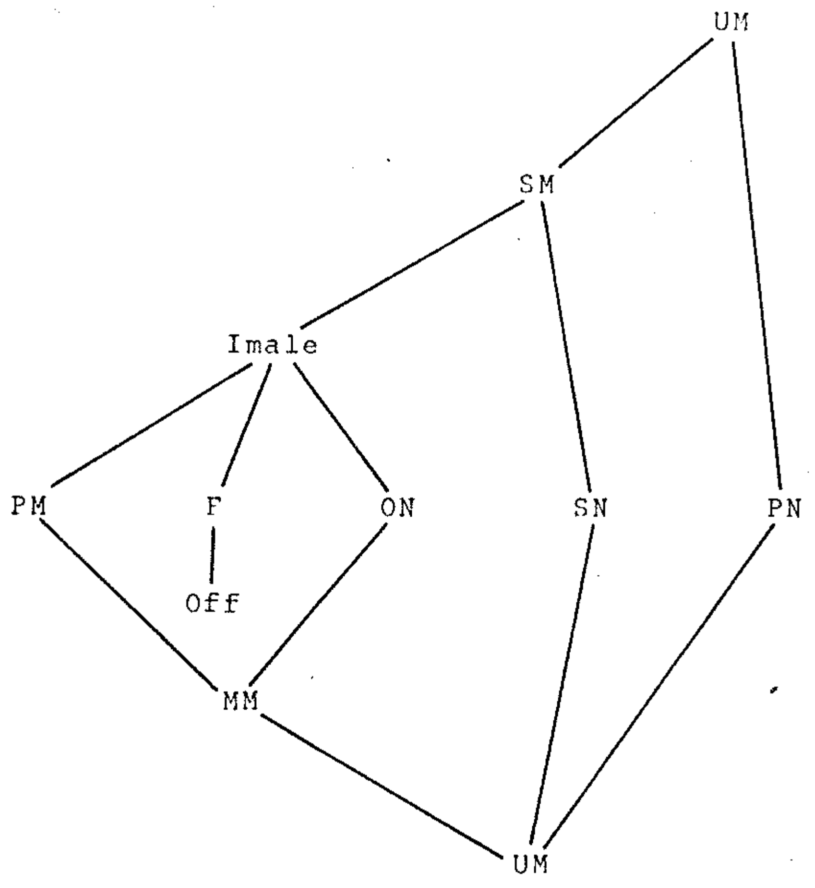
we encounter a data path:



Is x the uncle of z? The expansion of this path is:

PM F ON SN PN.

Because of the flag F we now find that there are two possible parses. One of these reduces the sentence to "SM PN" and then to "Uncle"; the other reduces first to "MM SN PN" and then to "Uncle".



Since both parses reduce to Uncle, and since one involves using the flag and the other does not, we are assured that in either case this sentence implies the "Uncle" relation.

The Question-Answerer

The above sections have highlighted some of the more interesting aspects of this approach to question-answering. We now provide a brief description of our system which consists of five basic blocks of LISP procedures. The first of these parses an input statement, determining whether it is a fact or a question. If the statement is a fact, it stores the statement and its converse in the dynamically grown data graph. If the statement is a question, the first block formats it as an (x, R, y) triple and passes it on to the second block. This block is responsible for searching the data graph and thus constructing all possible LPS's that bridge the (x, y) 2-tuple mentioned in the statement.

After each LPS is generated, a check is made to see if it contains any non-atomic relations. If it does, it is pushed on a hold list and the search is continued for another LPS. Whenever an LPS is encoutered which consists solely of atomic relations, that LPS is immediately passed to the theorem-prover (parser). If the theorem-prover fails to reduce this sentence to the desired relation, control is passed back to the search routine which picks up where it left off. If a successful reduction is performed, control is immediately returned to the top level with the appropriate response.

If the searching block has discovered all possible LPS's and no successful reduction has occurred for LPS's involving only atomic terms, the third block is entered which begins processing the above-mentioned hold stack. Each of these LPS's is, in turn, popped off this stack, expanded (note that if the non-atomic relation has a disjunctive defintion, then a sentence is created for each disjunct) and passed to the theorem-prover.

Theorem-Proving

Several different parsers have been experimented with, such as Wood's context-sensitive parser and a straight substitution parser. There is an obvious trade-off between parsers which are designed to stop after the first successful parse and those designed to find all parses. The latter is ideal if a particular sentence has no appropriate final reduction. We hope to soon be able to experiment with Earley's parser

Summary

We have outlined above a question-answering system which complements "the Monkey's Uncle" -- one which is capable of using the theories produced by the latter. We have discussed in some detail the philosophy behind our question-answerer so as to expose some of the more subtle aspects of the kind of theory "the Monkey's Uncle" can generate. Although we do not pretend that the approach to deduction unfolded above is suitable for truly general purpose question-answering systems, we believe that new avenues for achieving special types of deduction should be explored. We hope that this treatment of question-answering will at least lead to a greater appreciation of the similarity between grammars and inference rules on the one hand, and parsers and theorem-provers on the other.

FOOTNOTES

(1) Notation originally suggested by Martin Kay (personal communication).

(2) Robert K. Lindsay, "Inferential Memory as the Basis of Machines Which Understand Natural Language," in Computers and Thought, edited by Edward Fiegenbaum and Julian Feldman (New York: McGraw-Hill,1963), 217-33.

CHAPTER EIGHT

Conclusion

In this thesis we have explored various techniques for abstracting structural information from a network of facts. We have concentrated our discussion on a symbiotic system ("the Monkey's Uncle") which formulates structural rules about the data in the form of intensional definitions of the binary relations constituting the data or inference rules which express interactions among these relations. We have explored a facet of induction. Of course the movement from particulars to generalities is immensely more complex and varied than is captured by "the Monkey's Uncle". Nevertheless we hope this system is a first step in investigating techniques which can discover representations of the structure underlying a domain of data.

Chapter Seven deviated significantly from its predecessors. In Chapter Seven we discussed a problem closely related to the converse of our initial problem. There we moved from induction to deduction in that we were primarily concerned with utilizing rules produced by "the Monkey's Uncle" to deduce additional information. One of our tasks was to make explicit the implicit data. But what is meant by implicit data? Considering implicit data presupposes that we have in mind some structural properties of the data or "world" from which the data was sampled. The discovery of these properties was, of course, the central task of our thesis.

Contrasting the problems of induction and deduction naturally leads to an intriguing issue. "The Monkey's Uncle" functions under the assumption that all possible data is contained in the given data network. It searches for redundancies in this data that can be expressed as specific rules. To find these redundancies it expects all the data to be present. What happens if there is some missing data? For example, suppose we are examining various cases of grandparent in which we expect to find the path "PN-PN" bridging each example. However because of (possibly) an oversight, we encounter one example in which the only path is "PN-MN-PN" (i.e. parent-spouse-parent). [Note that all the previous examples which had "PN-PN" would also have had "PN-MN-PN" which would eventually lead to the inference rule "MN/PN => PN".] Because of the absence of the "PN-PN" path "the Monkey's Uncle", unless instructed to the contrary, would ascribe significance to its absence and thus conjecture a definition of grandparent to be: "GN = "PN/PN V PN/MN/PN" and, in addition, would reject the above inference rule. If there were just a few missing segments of data one might expect that statistical techniques could be employed to isolate such cases (and thus another aspect of induction appears). We, however, have been skeptical of such an approach and have purposely designed our system so that the user decides whether or not particular omissions are significant.

Further Research:

1) Throughout this thesis we have concentrated on structural rather than statisitcal aspects of our data. In coping with missing data we are interested in techniques which can confirm or disconfirm any alterations the user imposes on the data base. Towards solving this problem we conjecture the following phenomena will occur (having great faith on Occam's razor). Suppose, instead of trying to verify a particular alteration in terms of statistical properties of that particular relation, we consider the global consequences of the alternative in terms of the entire collection of intensional definitions. If the extension of an atomic relation is incorrect then we conjecture those definitions utilizing the particular relation will be inordinately complex. Therefore, any changes in the atomic relations which result in a drastic simplification in all the definitions involving those

relations are probably correct. For example, in the above "grandparent" example, the addition of one fact to the extension of "PN" would significantly simplify the intensional defintion for "grandparent". We would like to explore this possibility by taking a complete data base and randomly deleting 2-tuples from the extension of relations that are known, ahead of time, to be atomic.

2) Our research has pointed out the inadequacies of an algebraic approach to theory formation. We would like to explore merging the two approaches in a completely interactive system. Towards this end we are contemplating rewriting "the Monkey's Uncle" in LISP for the PDP-10. Once in LISP, experimenting with new hypothesis formation and verification strategies would be greatly facilitated. In addition we would be in position to merge the algebraic approach (which requires a reasonably large core) with the list processing approach underlying "the Monkey's Uncle".

Our experience with using small computers for this kind of research has been quite disappointing. Although L6/9 proved to be an excellent language for small computers, the lack of appropriate peripherals for quickly editing and recompiling programs became a significant drawback to extensive experimentation with new strategies.

APPENDIX 1

Towards An Algebraic Approach to Theory Formation

Introduction

Given a collection of binary relations $R = \{R_1, \ldots, R_n\}$ defined on some universe U, the search for intensional definitions of one relation in terms of other relations in R can be characterized algebraically. Recalling that an intensional definition which is defined with respect to composition (for this discussion we will only consider this class of intensional definitions) is defined as:

$$R_i = R_{i_1}/R_{i_2}/ \ldots /R_{i_n} \text{ if and only if } \forall x,y$$
$$\exists z_1, \ldots, z_{n-1} \left[(x\ R_i\ y) \Longleftrightarrow \left((x\ R_{i_1}\ z_1) \wedge (z_1\ R_{i_2}\ z_2) \wedge \ldots \wedge (z_{n-1}\ R_{i_n}\ y) \right) \right]$$

It is clear that the composition operator "/" is equivalent to boolean multiplication of the respective adjacency matrices of the given relations. More precisely, if $R_1 = R_2/R_3$, then letting M_1, M_2 and M_3 denote the adjacency matrices of R_1, R_2 and R_3 respectively, we note that:

$$M_1 = M_2 \odot M_3$$

where '$\odot$' is defined over n x n boolean matrices as:

$$A = B \odot C \Longleftrightarrow A_{ij} = \bigvee_{k=1}^{n} B_{ik} \wedge C_{kj}.$$

Examining the composition operator on binary relations, one can easily verify that the composition of two binary relations is still a binary relation and that the operator is associative. More generally, given the collection R of binary relations, one can define a semigroup S_R generated by R as the closure (under composition) of R. In other words, by forming all possible compositional sequences made from relations in R, the resulting structure is a semigroup. Likewise, one can define the semigroup S_M whose elements are the boolean adjacency matrices corresponding to binary relations of S_R. The semigroups S_R and S_M are trivially isomorphic with respect to the mapping $M_i \iff R_i$ and therefore we are free to discuss either, with our observations pertaining to both structures. However, as we will see later, S_M is a particularly easy structure to generate with a binary computer.

Intensional definitions of the relations in R are directly derivable from the semigroup S_R. Consider a representation of S_R by its Cayley multiplication table which specifies for any two elements R_u, $R_v \in S_R$ a third element R_ω (not necessarily distinct) such that:

$$R_\omega = R_u * R_v .$$

Suppose R_ω is one of the relations in R but R_u, R_v are not contained in R. In such cases we locate two terms whose product is R_u and similarly for R_v. Since S_R is generated by R, eventually every relation in S_R can be traced back to the sequence of relations in R that generated it. In other words, given $R_w = R_u * R_v$, we find the sequence of relations that generated R_u and then those that generated R_v. Since $R_w = R_u * R_v$, composing these two sequences together gives us a defining sequence for R_w which is specified entirely in terms of the generating relations, i.e.,

$$\begin{array}{lll} \text{if} & R_u = R_{i_1} * \ldots * R_{i_k} & R_{i's} \in R \\ & R_v = R_{j_1} * \ldots * R_{j_n} & R_{j's} \in R \\ \text{then} & R_w = R_{i_1} * \ldots * R_{i_k} * R_{j_1} * \ldots * R_{j_n}. & \end{array}$$

Since R_w by hypothesis was in R, we have unfolded an intensional definition of one relation in R in terms of the other relations in R. Of course, R_w might occur on the right-hand side of the equals sign, in which case the definition is recursive.

Growing the semigroup leads to other advantages. For example, we have discussed in Chapter 6 the search for "atomic" relations which translate under the above representation into a search for a minimal set of generators for S_R. What is advantageous about this representation is that although we might discover some subset R' of R that can generate S , we might also discover a still smaller set X (i.e., $|X| < |R'|$) that can also generate S_R where X is not a subset of R. In this case a "new" relation or set of relations (derivable from R) will have been discovered that more "economically" explains the data. The discovery of such new relations which are external to the original relations constituting the data base would be a powerful extension to our system.

Before discussing in more detail the algebraic formulation of this aspect of theory formation, we will explore some of its computational advantages and disadvantages.

Computational Advantages

Central to the formation of S_R is the generation of the closure of the collection of binary relations R. This task is ideally suited for digital computers since the fundamental operations in forming closure are:

1) Performing a boolean multiplication of two adjacency matrices;

2) Determining if the resulting product matrix has already occurred.

The first operation is facilitated by the fact that in forming the closure of $R = \{R_1, R_2, \ldots, R_n\}$ all multiplications can be of the form:

$$\begin{array}{l} R_w \ast R_i \\ \overbrace{R_{i_1} \ast \ldots \ast R_{i_n}}^{R_w} \ast R_i \end{array}$$

where $R_{i_k} \in R$ but not necessarily R_w.

We can represent each row of R_w as a bit-packed binary word (or words) and for each relation R_i in R we can represent its <u>columns</u> as a bit-packed binary word (or words). With this representation, each i,j entry in the product matrix is computed by "AND"ing the ith word of R_w with the jth word of R_i - an extremely fast operation. In addition, since every required multiplication only has R_1, ..., R_n as a possible right-hand element, only these need be converted and stored in their column representation. In other words, the resulting product matrices need never be converted to column form.

The identification of whether or not the resulting product matrix is unique requires, in principle, checking this new matrix against all the ones that have already been generated. This check, however, can be performed by a hash coding technique which generates an index into a hash table directly from the n x n matrix stored as a sequence of bit-packed words. If the product is unique, then it gets stored in the hash table along with two pointers identifying what its immediate generators were. These pointers facilitate the recursive reconstruction of the sequence of multiplications which were used in generating this element of the semigroup in terms of the given generators.

Disadvantages

The primary disadvantage of this approach is the combinatorial explosion underlying the generation of the semigroup. For example, if the generating relations for the semigroup are represented as n x n matrices, then the number of matrices making up the semigroup can be as many as 2^{2^n} (this is, of course, the worst case). What is clearly needed is a way to select a sub-universe of the given universe; restrict the generating relation to this sub-universe (i.e. reduce the n x n matrices to m x m, where m << n); and somehow utilize the structure of this semigroup to discover the sought-after definitions, etc. The following sections will describe how this can be achieved.

Selecting a Sub-Universe

In order to manifest the problems inherent in selecting a sub-universe on which to generate the semigroup, we consider the following example. Let us suppose that our original universe $U = \{a,b,c,d,e,f,g\}$ and that we have three relations defined on U represented by the following matrices:

$$M_1 = \begin{Bmatrix} M_{1_{ab}} = 1 \\ M_{1_{cb}} = 1 \\ M_{1_{eg}} = 1 \end{Bmatrix} \qquad M_2 = \begin{Bmatrix} M_{2_{ac}} = 1 \\ M_{2_{ed}} = 1 \\ M_{2_{ef}} = 1 \end{Bmatrix} \qquad M_3 = \begin{Bmatrix} M_{3_{cb}} = 1 \\ M_{3_{db}} = 1 \\ M_{3_{fg}} = 1 \end{Bmatrix}$$

Among the various relators in a presentation of the semigroup S_M generated by these three relations is the relator: $M_1 = M_2 \circ M_3$. Because S_R and S_M are isomorphic, this relator may be translated directly into an intensional definition of R_1 as $R_1 = R_2/R_3$. Let us now define a sub-universe U' of U as being the set $\{a,b,d,e,f,c\}$ and restrict the three matrices on U to U'. Relative to these new matrices, M_1' , M_2' , M_3' we can again generate a new semigroup, say $S_{M'}$. However we can directly verify that on the semigroup $S_{M'}$, the relator $M_1 = M_2 \circ M_3$ is no longer satisfied, thus destroying our chance of discovering the intensional definition of R_1.

Since we desire to use S_R to discover the intensional definitions of the generating relations of $S_{R'}$, we would like every relator on S_R to hold (under the natural mapping) on $S_{R'}$. In other words, we want $S_{R'}$ to be a homomorph of S_R. That is to say if

$$R_i = R_{i_1} * \ldots * R_{i_n} \qquad R_{i_j} \in S_R \, .$$

and if $h: S_R \rightarrow S_{R'}$ *then*

$$\begin{tikzcd}[column sep=small]
h(R_i) \arrow[d] & = h(R_{i_1} * \ldots * R_{i_n}) & = & h(R_{i_1}) \arrow[d] & * \ldots * & h(R_{i_n}) \arrow[d] \\
R_i' & = & = & R_{i_1}' & * \ldots * & R_{i_n}'
\end{tikzcd}$$

In the above example, restricting R to that particular U' and using these restricted relations to generate the semigroup $S_{R'}$ caused at least one relator of S_R to be unsatisfied on $S_{R'}$. Clearly the mapping h: $R_i \Rightarrow R_i|u'$ could not therefore be a homomorphism from S_R to $S_{R'}$.

We are interested in determining an algorithm for selecting a U' such that the mapping h: $R_i \Rightarrow R_i|u'$ is extendable to a homomorphism from S_R to $S_{R'}$. Towards this end we consider the collection R to be represented as a labelled directed graph G constructed so that if $(x,y) \in R_i$ then there is an arc from x to y labelled R_i. Given an edge (x,y) in G we can define a subgraph G' generated by this edge as the subgraph consisting of all directed paths (including non-simple paths) from x to y. Relative to G'_{xy}

we can further define a relation T_{xy} as:

Def: $T_{xy} = \{(x',y') \mid x'$ precedes y' on some path in $C'_{xy}\}$

Since C'_{xy} consists of both simple and non-simple paths, T_{xy} is not a partial ordering because the anti-symmetric property can be violated. T_{xy} specifies a subset of the cartesian product U x U. We can define the restriction of R_i to T_{xy} as:

Def: $$R_i|_{T_{xy}} = \{ (x',y') \mid (x',y') \in R_i \;\&\; (x',y') \in T_{xy} \}$$

In terms of these definitions we may now formulate the following theorem:

Theorem: Given a collection of binary relations R and a T_{xy} as defined above, then

$$R_i|_{T_{xy}} * R_j|_{T_{xy}} = (R_i * R_j)|_{T_{xy}} \text{ where } R_i \in R$$

Proof:

Suppose $(u,v) \in R_i|_{T_{xy}} * R_j|_{T_{xy}}$. Then there exists z such that $(u,z) \in R_i|_{T_{xy}}$ and $(z,v) \in R_j|_{T_{xy}}$. But this implies that $(u,v) \in T_{xy}$ (by definition of T_{xy}) and is therefore in $(R_i * R_j)|_{T_{xy}}$. Now suppose that $(u,v) \in (R_i * R_j)|_{T_{xy}}$ which implies that 1) $(u,v) \in T_{xy}$ and 2) there exists a z such that $(u,z) \in R_i$ and $(z,v) \in R_j$. To show that $(u,v) \in R_i|_{T_{xy}} * R_j|_{T_{xy}}$ we need only show that (u,z) and (z,v) are contained in T_{xy}. Since $(u,v) \in T_{xy}$ u must precede v on some directed path from x to v. Since G' consists of all possible directed paths from x to v and since u precedes v on some directed path for x,v then G' must also consist of all directed paths from u to v. The path (u,z)-(z,v) is one such path and therefore these 2-tuples are in T_{xy}.∎

Because T_{xy} is defined with respect to paths of arbitrary length in G', this proof can be immediately generalized to establish the following corollary:

Corollary: Given a collection of binary relations R and T_{xy}, then

$$R_{i_1}|_{T_{xy}} * R_{i_2}|_{T_{xy}} * \ldots * R_{i_n}|_{T_{xy}} = (R_{i_1} * \ldots * R_{i_n})|_{T_{xy}}$$

Since any element of S can be written (by definition) as some sequence--

$$u = R_{i_1} * R_{i_2} * \ldots * R_{i_n} \qquad u \in S_R$$

the above corollary provides the link between homomorphisms and the T_{xy}'s as follows.

Theorem: Given a semigroup S_R generated from a collection of binary relations R under composition and a mapping:

$$h_{xy} : R_i \rightarrow R_i|_{T_{xy}}$$

then h_{xy} is a homomorphism from $S_R \rightarrow S_{R'}$ where $S_{R'}$ is the semigroup generated by restricting each relation in R to T_{xy}.

Discussion

One of the virtues of defining T_{xy} in the above fashion is that once G has been constructed (i.e. represented by an appropriate data structure) the generation of particular T_{xy}'s is extremely fast. It is of no surprise that certain choices of x,y lead to an h_{xy} which so drastically collapses S_R that nearly "everything" holds on its homomorph. For such cases the homomorph is of little value. We have investigated certain heuristics which enable us to reject some T_{xy} before computing its allied homomorph but we delay reporting on them without further experimental data.

APPENDIX 2

Generalized Applications of "the Monkey's Uncle"

In this appendix we will explore the applicability of "the Monkey's Uncle" to some different types of data and examine the range of theories obtainable from it. The first experiment deals with a state transition system which models a collection of actions defined on some ideal "world". An action may be thought of as a "verb" or an operator which, when applied, causes a change in the current state of the "world". The theory we are interested in discovering concerns how a particular action (verb) can be defined in terms of other actions (verbs) which are already known to the system. Viewing these actions as operators, our theory should also indicate such properties as whether or not these actions commute. Finally, by discovering all the ways actions can be defined, we can find some minimal set of actions from which all the others can be derived. Although these actions and states of the world are abstract, this paradigm is currently being used in psychological experiments where the states of the world are represented as graphic configurations and the actions are "levers" that cause the (computer generated) graphic configuration to transform its current state. This psychological study concerns how subjects obtain and represent the structure of this "ideal" world as an example of a complex concept formation task.

The remaining experiments (i.e. 2,3,4) concern the discovery of the rules of chess moves. These experiments start with simple boards and progress up to the official eight-by-eight board. Some of these experiments presented severe memory strains to the existing system which was designed to run in a 16 K machine, including the interpreter. Consequently, these experiments not only show some of the generality of this kind of system but also indicate some ways to collapse or sample data without destroying the chance of finding the sought-after structure.

Experiment #1

Task: Given a state space and a collection of unary operators or "actions" which map a given state onto a new state, determine the structure of these actions. In particular, discover how any particular action can be synthesized out of other actions.

Discussion: For this experiment we consider a domain of six states - i.e., S1, S2, ... , S6 and a collection of five actions - i.e., A1, A2, ... , A5. Each action is represented extensionally by specifying explicitly how it maps each state onto the other states. The table below presents this information.

	A			B			C	
S1	=>	S2	S1	=>	S4	S1	=>	S3
S2	=>	S3	S2	=>	S5	S2	=>	S1
S3	=>	S1	S3	=>	S6	S3	=>	S2
S4	=>	S6	S4	=>	S1	S4	=>	S5
S5	=>	S4	S5	=>	S2	S5	=>	S6
S6	=>	S5	S6	=>	S3	S6	=>	S4

	D			E	
S1	=>	S6	S1	=>	S5
S2	=>	S4	S2	=>	S6
S3	=>	S5	S3	=>	S4
S4	=>	S2	S4	=>	S3
S5	=>	S2	S5	=>	S1
S6	=>	S1	S6	=>	S2

"The Monkey's Uncle" can be directly applied to this data and with its help we can investigate how any given action can be synthesized out of the compositions of other actions. We summarize below the results of the first pass of "the Monkey's Uncle" over the data:

A	=	E*B	=	D*E	=	C*C	=	B*D
B	=	A*D	=	E*A	=	D*C	=	C*E
C	=	A*A	=	E*D	=	D*B	=	B*E
D	=	A*E	=	E*C	=	C*B	=	B*A
E	=	A*B	=	D*A	=	C*D	=	B*C

A brief glance at this table reveals several interesting results. First, we note that every action has four possible definitions consisting of the compositions of two other actions. In fact, since each definition for a given action differs from each other definition in terms of the first relation and the second relation, we can verify that:

> For any two distinct actions Ai and Aj there necessarily exists a third action such that:
>
> Ai = Aj*Ak and Ai*Ak' = Aj.

With respect to the definitions selected below, we can draw a "generator" graph for this system:

A = C*C
B = C*E
C = A*A
D = E*C
E = C*D

$$\begin{tikzcd}
 & & C \arrow[dll, bend right] \arrow[drr, bend left] \arrow[ddl] & & \\
A \arrow[urr, bend right] & & & & D \arrow[dl, bend left] \\
 & B & & E \arrow[ll, bend left] \arrow[ur, bend left] &
\end{tikzcd}$$

and determine that, with respect to these definitions, two actions (C and F) form a basis from which the remaining three actions can be derived. It is clear from the total set of definitions that these actions do not commute, i.e., that the order of their application matters.

Experiment #2

Task: Given an N x N chess board, define extensionally the following set of local and global topological relations:

Local Relations

JE	=>	Just East
JW	=>	Just West
JS	=>	Just South
JN	=>	Just North

Global Relations

EE	=>	East
WW	=>	West
SS	=>	South
NN	=>	North
PU	=>	Positive Diagonal Running "Up"
PD	=>	Positive Diagonal Running "Down"
NU	=>	Negative Diagonal Running "Up"
ND	=>	Negative Diagonal Running "Down".

Our task is to discover how the global relations can be intensionally defined in terms of the local relations and themselves - (i.e., recursive definitions are to be allowed). Also we would like to see what clues can be found concerning the existence of such unary predicates as "interior" and "exterior" points on this space.

The table below summarizes the results of this exploration on a four-by-four board.

<u>Summary of Results</u>

```
WW = JS/WW/JN   V   JN/WW/JS   (Non-disjoint)

                    JW/WW
WW =    JW   V      WW/WW      (Disjoint)
                    WW/JW

EE = JS/EE/JN   V   JN/EE/JS   (Non-disjoint)
```

```
                    JE/EE
EE =    JE   V      EE/EE          (Disjoint)
                    EE/JE

SS = JE/SS/JW   V   JW/SS/JE   (Non-disjoint)

                    JS/SS
SS =    JS   V      SS/SS          (Disjoint)
                    SS/JS

NN = JW/NN/JE   V   JE/NN/JW   (Non-disjoint)

                    JN/NN
NN =    JN   V      NN/NN          (Disjoint)
                    NN/JN

                                PD/PD
                                JE/PD/JN
         JE/JN                  JE/JN/PD
PD =     JN/JE        V         PD/JE/JN
                                PD/JN/JE
                                JN/JE/PD
                                JN/PD/JE

                                PU/PU
                                JW/PU/JS
         JW/JS                  JW/JS/PU
PU =     JS/JW        V         PU/JW/JS
                                PU/JS/JW
                                JS/JW/PU
                                JS/PU/JW

                                NU/NU
                                JE/NU/JS
         JE/JS                  JE/JS/NU
NU =     JS/JE        V         NU/JE/JS
                                NU/JS/JE
                                JS/JE/NU
                                JS/NU/JE

                                ND/ND
                                JW/ND/JN
         JW/JN                  JW/JN/ND
ND =     JN/JW        V         ND/JW/JN
                                ND/JN/JW
                                JN/JW/ND
                                JN/ND/JW
```

In terms of the East, West, North and South relations, we sought both disjoint and non-disjoint definitions. As can be seen in the following, each kind of definition generates important information for abstracting the above-mentioned unary predicates. For example, by considering the following two definitions for WW (and by knowing what these relations mean, which is of course unknown to our system) we can see that:

Non-disjoint Def.: WW = JS/WW/JN V JN/WW/JS

For any (x,y) which is contained in the first disjunct and <u>not</u> in the second, it must represent two points along the "southern" border. Likewise, those 2-tuples valid in the second disjunct and not in the first lie along the "northern" border. Clearly those 2-tuples valid in both disjuncts are interior points relative to these two borders. The disjoint definition for WW is:

$$\text{Disjoint Def.:}\quad WW = JW \;\; V \begin{array}{c} JW/WW \\ WW/WW \\ WW/JW \end{array}$$

which communicates the additional information that JW commutes with WW. It further specifies that WW is transitive and that the 2-tuples in WW that don't have a "bridge" i.e. the 2-tuple (x,y) has a bridge if

$(x,y) \in WW \Rightarrow \exists z \mid (x,z) \in WW \ \& \ (z,y) \in WW$

are exactly those of JW.

Looking at the definition for diagonals, we likewise note that they are all transitive. In addition, these definitions reveal the commutative nature of certain subsets of the local relations. For example:

```
JE/JN = JN/JE
JW/JS = JS/JW
JE/JS = JS/JE
JW/JN = JN/JW
```

The actual form of the data 2-tuples is derived from the following numeration of the four-by-four chess board:

```
14   9   8  15
 3   2  12   5
 6  11  10   4
16   7   1  13
```

To illustrate how the user can benefit from additional information generated by the system, we include a copy of the first page of output and show how he can note additional regularities. By taking the two templates (see line X):

```
037763|770 => JN/WW/JS
343617|174 => JS/WW/JN
```

and forming their logical difference, etc., we can detect an invariance over those 2-tuples in the intersection and

```
                ****WELCOME TO THE MONKEY'S UNCLE****

PLEASE LOAD THE RELATION NAMES
NOW LOAD YOUR DATA FILE

TYPE IN RELATION TO BE EXPLORED
WW
RECURSIVE ? YES
PRINT 2-TUPLES? YES
WHAT DEGREE OF VERIFICATION IS DESIRED? C
ANY DEFINITIONS TO BE BLOCKED? N                      Extension Table
DISJOINT ?N
 # OF SAMPLES?  35
TOTAL # OF SAMPLES ONLY = 024
<016,013> <016,001> <016,007> <014,015> <014,008> <014,009> <012,005>
<011,004> <011,010> <010,004> <009,015> <009,008> <008,015> <007,013>
<007,001> <006,004> <006,010> <006,011> <003,005> <003,012> <003,002>
<002,005> <002,012> <001,013>

XX
<CRITERIA SATISIFIED....EXAMINE SOLUTION>    The ith bit of the template
                                             corresponds to the ith 2-tuple
----                                         in the extension table.
LIST T

035035:474645  377777  774000  000000

034645:474635  037763  770000  035045  <- Line X

034635:440000  343617  774000  034001
----
34001T

034001: 474005  JS/WW/JN
----
35045T

035045: 475051  JN/WW/JS
----
*
LEAVING EXAMINE MODE

NOW ANY DEFINTIONS TO BE BLOCKED? YES
ERASE CURRENT LIST?NO
NEW DEFINITIONS NOW
: JS/WW/JN
: JN/WW/JS
```

those in the symmetric difference. The following significant pattern emerges:

Symmetric Difference	Intersection
All 2-tuples whose first component is 16,14,6,9,2,1	All 2-tuples whose first component is 12,11,10,8,7,3

Since the first component seems to determine which disjunct the 2-tuple will fall into, we might focus our attention on what domain elements are missing from the first component of the WW 2-tuple:

Missing 2-tuples

Any 2-tuple whose first component is
4,5,13 or 15.

Evidently, these points constitute the "western" border.

Experiment #3

Task: Given a five-by-five chess board which is defined by specifying the complete extensions of JE,JW,JS and JN and on which is specified all the moves of a particular chess piece, determine the rules for the moves of the given piece. These rules are to be expressed either in terms of the above-mentioned local relations or may be recursive rules defining a legal move in terms of itself and the local relations.

Discussion: The following rules were discovered:

```
   1)  Knight =    JW/JN/JN  V  JE/JN/JN  V
                   JE/JE/JN  V  JW/JW/JN  V
                   JW/JW/JS  V  JW/JS/JS  V
                   JE/JS/JS  V  JE/JE/JS
```

We see that the system discovered an eight term disjunctive definition of a Knight's move. For any particular term it also found all of its logical equivalents. For example, for the first term above, it unfolded:

JW/JN/JN = JN/JW/JN = JN/JN/JW

In each case, we simply chose the first term.

2) Bishop = JS/BI/JN V JN/BI/JS V X

This definition is the first example of a recursive definition. It states that a Bishop's move is either of the form:

a) Move just south, then make a Bishop's move and then move just north

or b) Move just north, then make a Bishop's move and then move just south.

Any move following these rules will be a legal Bishop's move, whereas a recursive rule referencing a diagonal generates counter-examples - e.g., JE/JN/BI covers some moves but would permit others that are illegal. The need for two disjunctive terms emerges because of the southern and northern boundaries.

There is, however, one glitch to the above, which is represented by the trailing "X" in the definition. When we first ran this case, the system returned saying that there was no satisfactory recursive definition since four cases (2-tuples) remained "unexplained". The system identified these cases and then proceeded to "explain" the remaining cases. These four cases were the two extreme points on each diagonal. In such situations a Bishop may move just south but not just north, or else vice versa.

```
3)  Rook =       JS/RK/JN   V   JE/RK/JW   V
                 JN/RK/JS   V   JW/RK/JE
```

This recursive rule amounts to saying that a Rook's move is either of the form:

a) Move just south, then make a Rook's move and then move just north

or b) Move just east, then make a Rook's move and then move just west

or c) Move just north, then make a Rook's move and then move just south

or d) Move just west, then make a Rook's move and then move just east

```
4)  King =       JS   V   JN   V   JE   V   JW   V
```

JS/JE V JS/JW V JN/JE V JN/JW

As with the Knight's moves, the system generated an eight term disjoint disjunctive definition.

5) Queen and Pawn.

The Pawn coincides with two of the local relations and any experiments involving it were postponed until the more difficult eight-by-eight case was considered. The Queen, on the other hand, produced an immediate problem. Simply storing the explicit possible moves for a Queen surpassed storage allocation for the relevant segment of the program. Consequently, we deferred exploration of this piece until the full eight-by-eight case, at which time sampling techniques were invoked for all the pieces. Of course, the process of discovering recursive definitions of a chess move is especially sensitive to incomplete extensional representations of the pieces. By studying the smaller board configurations it was possible to store the complete extensions of the chess pieces (except for the above mentioned case).

Experiment #4

Task: Determine a way to select a subset of the extension of each piece's move so that all pieces can be represented in core simultaneously. Then search for definitions of the moves which are represented in terms of the local and global relations and the moves of the other pieces.

Discussion: The need to develop such sampling techniques is apparent from the fact that representing all of the moves of all the pieces on even a four-by-four board used so much of the available memory that the system often ran out of space during the hypothesis evaluation phase. On an eight-by-eight board, the situation was, of course, completely hopeless.

As mentioned in the prior experiment, the search for recursive definitions must, in general, be limited to complete representations of the data. However, the situation for non-recursive rules is quite different. In seeking such rules there are two basic problems to consider.

Problem #1: Hypothesis Generation: It is clear that the generation of LPS's that involved the relation itself would often be foiled by an incomplete representation of its extension. For non-recursive LPS's we need only those 2-tuples of the other relation that bridge the existing 2-tuples of the given relation. This property is achievable for the chess data if we construct our data base according to the following algorithm:

1) Include the complete extensions of the local topological relations;

2) Include the complete extension of all the chess moves and global relations but where the <u>domains</u> of the pieces and global relations are restricted to some small representative subset of the board.

In other words, the end position of any move is free to fall within any square on the entire board so long as the origin of the move lies within this subset.

<u>Problem #2: Hypothesis Verification:</u> Once some potential rules are discovered, then hypothesis verification determines if these rules generate any "non-moves". Since only a partial representation of the moves is present, this check could be disasterous. Since a King's move might be "move just east and then move just north", verification would proceed by examining every such move to see if it were contained in the list of King's moves, and of course, in the above scheme not all samples chosen from the extension would be. One circumvention of this difficulty might be simply to turn off the verification phase (e.g., ask for "No" verification). We might think we could get away with this if our data base were sparse enough. However, our data, being generated by the above scheme, is very "dense"

wherever it exists. In any event, eliminating verification leads, for example, to such statements as "a Bishop's move is identical to a Queen's move".

There does exist a partial verification scheme which we can employ. Suppose R = A V B where A,B are compositional sequences. By restricting the verification phase on A and B to those elements in R's domain, the over-general definitions on this data base are weeded out without disturbing the correct ones. With this technique (selectable by means of an internal switch in the system) we obtain the following results:

Summary of Pesults (1)

Bishop	=	NU V PU V ND V PD (i.e., up and down positive and negative diagonals)
Pook	=	EE V WW V SS V NN (easterly or westerly or southerly or northerly)
King	=	JS V JN V JE V JW V JS/JE V JS/JW V JN/JE V JN/JW
Knight	=	JW/JN/JN V JE/JN/JN V JE/JE/JN V JW/JW/JN V JW/JW/JS V JW/JS/JS V JE/JS/JS V JE/JE/JS
Queen	=	Bishop or Rook
Pawn	=	JS V JN

Footnotes

(1) These results were obtained by using the complete extensions of JE, JW, JN and JS and restricting all the relations (i.e., NU, PU, ND, PD, WW, EE, SS, WW) and the chess moves to a sub-domain consisting of four squares on the board. Even with a sub-domain this small the available memory was so limited that adding eight Pawn moves caused the system to exceed memory capacity. Consequently, the first four chess pieces were run without the Pawn moves being stored.

APPENDIX 3

Mnemonics of American Kinship Terms

PM	Father
PF	Mother
PN	Parent
OM	Son
OF	Daughter
ON	Offspring
SM	Brother
SF	Sister
SN	Sibling
MM	Husband
MF	Wife
MN	Spouse
GM	Grandfather
GF	Grandmother
GN	Grandparent
DM	Grandson
DF	Granddaughter
DN	Grandchild
UM	Uncle
UF	Aunt
NM	Nephew
NF	Niece
CM	Cousin
CF	Cousin
LM	Brother-in-Law
LF	Sister-in-Law
EM	Son-in-Law
EF	Daughter-in-Law
XM	Father-in-Law
XF	Mother-in-Law

Traditional Definitions of American Kinship Terms

Parent	=	Father V Mother
Offspring	=	Son V Daughter
Sibling	=	Brother V Sister
Spouse	=	Husband V Wife
Grandfather	=	Father/Parent
Grandmother	=	Mother/Parent
Grandparent	=	Parent/Parent
Grandson	=	Son/Offspring
Granddaughter	=	Daughter/Offspring
Grandchild	=	Offspring/Offspring
Uncle	=	Brother/Parent
Aunt	=	Sister/Parent
Nephew	=	Son/Sibling
Niece	=	Daughter/Sibling
Cousin	=	Offspring/Sibling/Parent
Brother-in-Law	=	Brother/(Husband V Wife) V Husband/Sister V Husband/Sister/Wife V Husband/Sister/Husband
Sister-in-Law	=	Sister/(Husband or Wife) V Husband/Sister V Husband/Sister/Wife V Husband/Sister/Husband
Son-in-Law	=	Husband/Daughter
Daughter-in-Law	=	Wife/Son

Father-in-Law	=	Father/Spouse
Mother-in-Law	=	Mother/Spouse

APPENDIX 4

Extensional Definitions

CN (33)	=	38;39;50
CN (24)	=	6;15;16
CN (50)	=	32;33
CN (38)	=	32;33
CN (15)	=	6;24
CN (39)	=	32;33
CN (32)	=	38;39;50
CN (16)	=	6;24
CN (6)	=	15;16;24
XM (41)	=	37
XM (8)	=	4;10;12
XM (31)	=	35;34
XM (10)	=	17
XM (22)	=	11
XM (41)	=	30
XM (36)	=	53;55;52
XM (2)	=	14;5
XF (21)	=	11
XF (9)	=	17
XF (7)	=	4;10;12
XF (30)	=	35;34
XF (37)	=	53;55;52
XF (1)	=	14;5
XF (40)	=	30;37
PN (10)	=	15;16
PN (11)	=	24
PN (9)	=	15;16
PN (30)	=	32;33
PN (12)	=	24
PN (2)	=	3;4
PN (31)	=	32;33
PN (37)	=	38;39;50
PN (50)	=	54
PN (5)	=	6
PN (22)	=	12;20
PN (17)	=	18;19
PN (4)	=	6
PN (7)	=	5;9;11
PN (52)	=	54
PN (21)	=	12;20
PN (15)	=	18;19
PN (41)	=	31;36
PN (8)	=	5;9;11
PN (1)	=	3;4
PN (40)	=	31;36
PN (36)	=	38;39;50
PM (52)	=	54
PM (36)	=	38;39;50
PM (31)	=	32;33
PM (41)	=	31;36
PM (22)	=	12;20
PM (12)	=	24
PM (15)	=	18;19
PM (10)	=	15;16
PM (8)	=	5;9;11
PM (5)	=	6
PM (2)	=	3;4
PF (1)	=	3;4
PF (40)	=	31;36
PF (17)	=	18;19
PF (4)	=	6
PF (30)	=	32;33
PF (7)	=	5;9;11
PF (21)	=	12;20
PF (37)	=	38;39;50
PF (11)	=	24
PF (50)	=	54
PF (9)	=	15;16
GN (10)	=	18;19
GN (37)	=	54
GN (8)	=	6;15;16;24
GN (2)	=	6
GN (21)	=	24
GN (9)	=	18;19
GN (1)	=	6
GN (40)	=	32;33;38;39;50
GN (41)	=	32;33;38;39;50
GN (22)	=	24
GN (36)	=	54
GN (7)	=	6;15;16;24
GM (2)	=	6
GM (22)	=	24
GM (10)	=	18;19
GM (8)	=	6;15;16;24
GM (36)	=	54
GM (41)	=	32;33;38;39;50
GF (9)	=	18;19
GF (40)	=	32;33;38;39;50
GF (1)	=	6
GF (37)	=	54
GF (7)	=	6;15;16;24
GF (21)	=	24
ON (20)	=	21;22
ON (15)	=	9;10
ON (24)	=	11;12
ON (19)	=	15;17

ON (5)	= 7;8
ON (50)	= 36;37
ON (4)	= 1;2
ON (39)	= 36;37
ON (33)	= 30;31
ON (36)	= 40;41
ON (12)	= 21;22
ON (11)	= 7;8
ON (16)	= 9;10
ON (18)	= 15;17
ON (9)	= 7;8
ON (3)	= 1;2
ON (54)	= 50;52
ON (6)	= 4;5
ON (38)	= 36;37
ON (32)	= 30;31
ON (31)	= 40;41
OM (33)	= 30;31
OM (39)	= 36;37
OM (12)	= 21;22
OM (15)	= 9;10
OM (18)	= 15;17
OM (5)	= 7;8
OM (3)	= 1;2
OM (31)	= 40;41
OM (38)	= 36;37
OM (19)	= 15;17
OM (24)	= 11;12
OM (36)	= 40;41
OF (9)	= 7;8
OF (50)	= 36;37
OF (32)	= 30;31
OF (6)	= 4;5
OF (54)	= 50;52
OF (16)	= 9;10
OF (20)	= 21;22
OF (11)	= 7;8
OF (4)	= 1;2
UM (39)	= 54
UM (12)	= 6;15;16
UM (31)	= 38;39;50
UM (5)	= 15;16;24
UM (3)	= 6
UM (38)	= 54
UM (10)	= 6;24
UM (36)	= 32;33
UF (4)	= 15;16;24
UF (16)	= 18;19
UF (14)	= 6
UF (30)	= 38;39;50
UF (11)	= 6;15;16
UF (55)	= 54
UF (53)	= 54
UF (37)	= 32;33
UF (9)	= 6;24
UF (20)	= 24
NM (24)	= 5;9;20;10;4
NM (38)	= 31;30
NM (39)	= 31;30
NM (33)	= 36;37
NM (18)	= 16
NM (15)	= 5;11;4;12
NM (19)	= 16
NF (32)	= 36;37
NF (6)	= 3;9;11;14;10;12
NF (16)	= 5;11;4;12
NF (54)	= 38;39;53;55
NF (50)	= 31;30
EM (12)	= 7;8
EM (52)	= 36;37
EM (10)	= 7;8
EM (5)	= 1;2
EF (55)	= 36;37
EF (14)	= 1;2
EF (11)	= 21;22
EF (17)	= 9;10
EF (4)	= 7;8
EF (53)	= 36;37
EF (30)	= 40;41
EF (37)	= 40;41
EF (34)	= 30;31
MN (1)	= 2
MN (35)	= 32
MN (39)	= 55
MN (30)	= 31
MN (12)	= 11
MN (5)	= 4
MN (10)	= 9
MN (22)	= 21
MN (2)	= 1
MN (8)	= 7
MN (33)	= 34
MN (40)	= 41
MN (4)	= 5
MN (55)	= 39
MN (11)	= 12

MN (41)	=	40
MN (7)	=	8
MN (9)	=	10
MN (17)	=	15
MN (14)	=	3
MN (38)	=	53
MN (52)	=	50
MN (32)	=	35
MN (15)	=	17
MN (3)	=	14
MN (21)	=	22
MN (53)	=	38
MN (50)	=	52
MN (34)	=	33
MN (37)	=	36
MN (36)	=	37
MN (31)	=	30
MM (5)	=	4
MM (35)	=	32
MM (52)	=	50
MM (31)	=	30
MM (41)	=	40
MM (22)	=	21
MM (39)	=	55
MM (15)	=	17
MM (8)	=	7
MM (10)	=	9
MM (3)	=	14
MM (33)	=	34
MM (2)	=	1
MM (38)	=	53
MM (36)	=	37
MM (12)	=	11
MF (32)	=	35
MF (55)	=	39
MF (9)	=	10
MF (50)	=	52
MF (14)	=	3
MF (34)	=	33
MF (21)	=	22
MF (4)	=	5
MF (30)	=	31
MF (53)	=	38
MF (17)	=	15
MF (40)	=	41
MF (1)	=	2
MF (11)	=	12
MF (37)	=	36
MF (7)	=	8
SN (5)	=	9;11
SN (38)	=	39;50
SN (12)	=	20
SN (4)	=	3
SN (32)	=	33
SN (18)	=	19
SN (31)	=	36
SN (16)	=	15
SN (11)	=	5;9
SN (50)	=	38;39
SN (9)	=	5;11
SN (20)	=	12
SN (39)	=	38;50
SN (3)	=	4
SN (33)	=	32
SN (19)	=	18
SN (36)	=	31
SN (15)	=	16
SM (12)	=	20
SM (38)	=	39;50
SM (18)	=	19
SM (5)	=	9;11
SM (36)	=	31
SM (15)	=	16
SM (33)	=	32
SM (39)	=	38;50
SM (31)	=	36
SM (3)	=	4
SM (19)	=	18
SF (50)	=	38;39
SF (20)	=	12
SF (4)	=	3
SF (32)	=	33
SF (11)	=	5;9
SF (16)	=	15
SF (9)	=	5;11
LM (5)	=	10;12;3;14
LM (36)	=	30
LM (10)	=	5;11;12;4
LM (12)	=	5;9;10;4
LM (33)	=	35
LM (3)	=	5
LM (38)	=	55;52
LM (39)	=	53;52
LM (35)	=	34;33
LM (31)	=	37
LM (52)	=	38;39;53;55

```
LF (53)  = 39;50;55;52
LF (17)  = 16
LF (55)  = 38;50;53;52
LF (50)  = 53;55
LF (11)  = 4;10;20
LF (16)  = 17
LF (30)  = 36;37
LF (4)   = 14;9;11;10;12
LF (14)  = 4;5
LF (20)  = 11
LF (34)  = 32;35
LF (9)   = 4;12
LF (37)  = 30;31
LF (32)  = 34
DN (15)  = 7;8
DN (50)  = 40;41
DN (54)  = 36;37
DN (19)  = 9;10
DN (39)  = 40;41
DN (16)  = 7;8
DN (6)   = 1;2;7;8
DN (32)  = 40;41
DN (24)  = 7;8;21;22
DN (18)  = 9;10
DN (33)  = 40;41
DN (38)  = 40;41
DF (16)  = 7;8
DF (54)  = 36;37
DF (50)  = 40;41
DF (6)   = 1;2;7;8
DF (32)  = 40;41
```

APPENDIX 6

An Example of Some of the Functions in ECYC
As Applied to the Graph in Chapter Six

```
                                  Load graph from disk file "NETWORK" and
                                  construct array representation - to be called
*(BUILDPRED NET NETWRK)           "NET"

NODES WITH ZERO PREDECESSORS ARE: NIL
NODES WITH ZERO SUCCESSORS ARE: NIL

NIL
                                  Print the "NET" structure in terms of
*(PRINTPRED NET)                  the predecessor of each node
PREDECESSOR GRAPH IS:

R1 :    (R2 R3 R6)
R2 :    (R4 R5)
R4 :    (R2 R3)
R6 :    (R1 R2 R4)
R3 :    (R1 R2 R6)
R5 :    (R1 R4)
NIL
*(PRINTSUC NET)                   Print the "NET" structure in terms of
SUCCESSOR GRAPH IS:               the successor of each node

R5 :    (R2)
R3 :    (R4 R1)
R6 :    (R3 R1)
R4 :    (R5 R6 R2)
R2 :    (R3 R6 R4 R1)
R1 :    (R5 R3 R6)                Compute the number of elementary cycles
NIL                               passing through each node. (then place
*(ORDER (ECYCLE NET))             them in ascending order).

((R5 15) (R6 17) (R1 21) (R4 21) (R3 22) (R2 23))
*(MAPC (FUNCTION PRINT) (LECYCLE NET))

(R5 R2 R3 R4)                     List the elementary
(R5 R2 R3 R4 R6 R1)               cycles in "NET"
(R5 R2 R3 R1)
(R5 R2 R6 R3 R4)
(R5 R2 R6 R3 R1)
(R5 R2 R6 R1)
(R5 R2 R6 R1 R3 R4)
(R5 R2 R4)
(R5 R2 R4 R6 R3 R1)
(R5 R2 R4 R6 R1)
(R5 R2 R1)
(R5 R2 R1 R3 R4)
(R5 R2 R1 R6 R3 R4)
(R3 R4 R6)
(R3 R4 R6 R1)
(R3 R4 R2)
(R3 R4 R2 R6)
```

```
 R3 R4 R2 R6 R1)
(R3 R4 R2 R1)
(R3 R4 R2 R1 R6)
(R3 R1)
(R3 R1 R6)
(R6 R1)
(R4 R2)                  list all elementary cycles of
NIL                      length 2.
*(SETQ C2 (LECYCLEL NET 2))

((R3 R1) (RG R1) (R4 R2))        Find the number of elementary cycles
*(INVERT C2) <------------------ each node has passing through it
                                 with respect to a given list of cycles.
((R2 1) (R4 1) (R6 1) (R1 2) (R3 1))
*(BLOCK @(R1) NET)
                  \______ Temporaily block node
NIL                       R1 in "NET".
*(CYCLIX\X\C↑U
(CYCLIC NET) <---- Test to see if the blocked
                   structure is cyclic
T <- "True"
*(INVERT (DELETE (R1) C2)) <---- Delete all cycles from C2 list
                                 that involve the node R1
((R2 1) (R4 1))
*(BLOCK @(R2) NET)
      At this point, both R1 &
NIL <- R2 are blocked.
*(CYCLIC NET)
                 Unblock node R2.
T
*(UNBLOCK @(R2) NET)
                   Now block node R4
NIL
*(BLOCK @(R4) NET)
                      Is the graph resulting
NIL <---------------- from blocking nodes
*(CYCLIC NET)         R1 & R4 cyclic?

NIL
*(PRINTSUC NET) <---- Print the successor graph of the
SUCCESSOR GRAPH IS:   graph resulting from these
                      blocks.

R5 :   (R2)
R3 :   (R4 R1)
R6 :   (R3 R1)
R4 :   NIL <-- blocked
R2 :   (R3 R6 R4 R1)
R1 :   NIL <-- blocked
NIL
*
```

Since blocking nodes R1 & R4 caused "NET" to become acyclic, these nodes must cover all the elementary cycles.

BIBLIOGRAPHY

Amarel, S., "On the Automatic Formation of a Computer Program Which Represents a Theory," in Self-Organizing Systems. Washington, D.C.: Spartan Books, 1962.

Ash, William "A Language and Data Structure for Fact Retrieval." unpublished Ph.D dissertation, University of Michigan, 1972.

Ash, W. L., and Sibley, E. H. "TRAMP: and Interactive Associative Processor with Deductive Capabilities." Proceedings ACM National Conference. Las Vegas, August, 1968.

Elliot, R. W. "A Model for a Fact Retrieval System." University of Texas TNN- 42, Austin, Texas, May, 1965.

Feldman, J. A., and Rovner, P. D. "An Algol-Based Associative Language." Communications of the ACM, XIII (August, 1969), pp. 439-449.

Gold, E. M. "Language Identification in the Limits," Information and Control, X (1967) pp. 447-474.

Hamburger, Henry, and Wexler, Kenneth "Identifiability of Transformation Grammars." Paper presented at the Workshop on the Grammar and Semantics of Natural Language, Palo Alto, Calif., Sept. 19, 1970.

Knowlton, Kenneth C. "A Programmer's Description of L^6." Communications of the ACM, VIII (August, 1966), 103-9.

Lawler, E. L. "Covering Problems: Duality Relations and a New Method of Solution." J. SIAM Applied Math, XIV (Sept. 1966), pp. 1115-1132.

Levien, R. E., and Maron, M. E. "Relational Data File: a Tool for Mechanized Inference Execution and Data Retrieval." The Rand Corporation, RM-4793-PR, December, 1965.

Lindsay, R. K., "Inferential Memory as the Basis of Machines which Understand Natural Langauge," in Computers and Thought, Edited by Edward Fiegenbaum and Julian Feldman. New York: McGraw Hill, 1963.

Raphael, B., "SIR: A Computer Program for Semantic Information Retrieval," in Semantic Information Processing, Edited by Marvin Minsky. Cambridge: MIT Press, 1968.

Schwarcz, R. M.; Burger, J. F.; and Simmons, R. E. "A Deductive Logic for Answering English Questions." _Communications of the ACM_, XIII (March, 1970), pp. 167-183.

Siklossy, Laurent "A Language-Learning Heuristic Program." _Cognitive Psychology_, II (October, 1971), pp. 479-495.

Simon, H. A., and Kotovsky, K. "Human Acquisition of Concepts for Sequential Patterns." _Psychological Review_, LXX (No. 6 1963), pp. 534-546.