STEPS TOWARD AUTOMATIC THEORY FORMATION

John Seely Brown
Information and Computer Science Department
University of California Irvine
Irvine, California

Abstract

This paper describes a theory formation system which can discover a partial axiomization of a data base represented *as* extensionally defined binary relations.- The system first discovers all possible intensional definitions of each binary relation in terms of the others. It then determines a minimal set of these relations from which the others can be defined. It then attempts to discover all the ways the relations of this minimal set can interact with each other, thus generating a set of inference rules. Although the system was originally designed to explore automatic techniques for theory construction for question-answering systems, it is currently being expanded to function as *a* symbiotic system to help social scientists explore certain kinds of data bases.

Introduction

For over a decade researchers in AI have been designing question-answering systems which are capable of deriving "implicit" facts from a sparse data base. Whether these systems use an axiomatic theorem proving approach or a procedurally oriented approach, they all must eventually face the problem of characterizing the generic knowledge or structural redundancies of their particular domain of discourse- Even the simplest of such domains contains countless subleties which somehow must be captured before a complete characterization can be achieved.

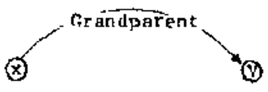
In this paper we discuss the problem of automatically constructing such a characterization. Our research has not been directed toward a theoretical investigation of this general problem as in Plotkin but rather towards constructing and experimenting with a prototype system which has been applied to large and structurally rich domains. We will provide numerous examples to show what is involved in forming conjectures about "apparent" structural redundancies in a given data base (model) and how such conjectures can be heuristically validated. The search for structure is inherently combinatorial. We will therefore discuss Bome ways to control the combinatorial explosion involved in forming and validating structural conjectures. Finally, we will discuss ways in which such a system might be used to help a social scientist discover structural theories about a set of observations over some "world". Indeed, our eventual aim is to expand this system into a tool for such uses.

Problem Definition

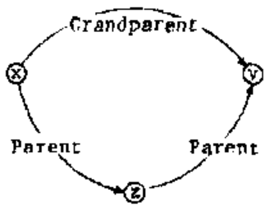
Before describing our system, a definition of the vague term "structural redundancies" is in order. Let us consider a data base consisting of the complete extensions of a collection of binary relations R - $\{R_1 \ldots\ldots R_n\}$. That is, for each R_i e R the data base contains all the triples (x R_i y) for each $(x,y) \in R_1$. The structural redundancies of this data base fall into three basic categories. The first category occurs where one of the relations is intensionally definable in terms of other relations in this data base. For example, suppose we have the three relations of Grandparent, Parent, and Spouse, each extensionally defined over some fixed universe P of people. Then, assuming the extensions of each relation are complete, we could discover the structural redundancy that for every $(x,y) \in$ Grandparent there exists a z such that:

$(x,y) \in$ Grandparent iff $(x,z) \in$ Parent and $(z,y) \in$ Parent

or Grandparent ≡ Parent/Parent where "/" denotes composition. If we represented the extensions of all the relations as a directed labelled graph, then for every arc of the form:



there would be a path of the form "Parent — Parent" bridging the triple (x Grandparent y) i.e.:

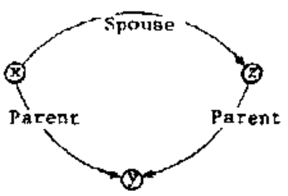


In other words, if "Grandparent ≡ Parent/Parent" holds, then the data base graph will always have a path of length two whose labels are "Parent" and "Parent" bridging every arc labelled "Grandparent".

An intensional definition of a relation can also be recursive as would be the case with the definition of Parent in terms of Spouse, e.g.:

Parent ≡ Spouse/Parent

In this case, for every arc of the form (x Parent y) there are two other arcs of the form (x Spouse z) and (z Parent y), i.e.:



The second kind of structural redundancy occurs where the existence of a path In the graph forms a sufficient but not a necessary condition for some other arc to exist. Tor example, suppose we augment our data base to Include the extensions of the Sibling and Cousin relations over the sane domain P. We would then discover that for every path of the form (x Sibling z) and (z Cousin y) there was always an arc of the form (x Cousin y). Such a pattern would lead to the rule:

Sibling/Cousin => Cousin

The third and simplest kind of structural redundancy occurs where the relation itself has certain logical properties such as:

Transitive
Symmetric
Reflexive
Irreflexive

The first two kinds of redundancies cover how a given relation interacts with other relations In R, whereas the last kind covers only how a relation Interacts with itself.

The structural patterns comprising all three categories form the basis of a logical theory (or axiomatic characterization) of the data base in the following way:

Initially, the data base consists of the complete extensional definitions of all the given relations. After our system discovers all the intensional definitions of the relations, then some subset of these relatione can be selected (to be called the atomic relations) from which the extension of all the remaining relations can be derived. The extension of a non-atomic relation can be computed by knowing only its intensional definition in terms of the selected atomic relations. Consequently, the extensional definitions of all non-atomic relations can be discarded with no loss of information. In addition, the extensions of the atomic relations can be pruned by using structural properties in the second and third categories. For example, if an atomic relation is symmetric, then half of its extension can be eliminated without loss of information. With respect to the two structural redundancies:

Parent = Spouse/Parent
Spouse symmetric

half of the parent and spouse extensions can be likewise discarded. What remains after all the redundant extensions have been eliminated is a kernel Bet of extensions.

The structural rules defined by the set of Intensional definitions, the logical properties of the atomic relations and the implicational statements in conjunction with a kernel set of extensions suffice to determine the complete extensions of all the relations. These rules plus the kernel set of data form a logical theory of the original extensional relations. Note that the complete extensional set of delations is necessarily a logical model of the resultant "theory". The fact that this model (or equlvalently — the extenaionally defined relations) is finite, as far as our system is concerned, presents the interesting situation that our system will often produce numerous rules that hold for this finite collection of data but which appear to be idiosyncratic. That is, if the collection of data is really a sample from some potentially infinite model, then of course numerous "rules" may be found that might prove to be false when the data base is enlarged.

Techniques to select the atomic sets are not discussed here but arc discussed at length in Brown3.

On Discovering Intensional Definitions

The basic idea underlying the discovery of intensional definitions is quite simple. Suppose we are trying to discover all the possible definitions of Ri e R. Since Ri exists in the data base, it has a set of 2-tuples which represent its extension. We commence our process by arbitrarily choosing one of these 2-tuples (*i»yi) and then searching for all possible simple directed paths that start at xi and end at yi We then delete the Intermediate nodes of each such path leaving only an ordered list of the labels of the path which we will call a Labelled Path Sequence (LPS). The end result of this process is a list of distinct LPS's, each stemming from at least one specific path from x± to yj. (Several xi -> yi paths can collapse onto one LPS,)

Depending on storage limitations etc., we repeat this process several times, choosing a different 2-tuple from Ri's extensions for each repetition. In this manner we construct a collection of lists L:

$$L = \left\{ L_{x_1y_1}, L_{x_2y_2} \ldots L_{x_ny_n} \right\}$$

where each $L_{x_iy_i}$ represents all the distinct LPS's bridging the 2-tuple (x_i, y_i).

From these lists we create some conjectures concerning R_i's intensional definition by forming the intersection of all the $L_{x_iy_i}$ lists, i.e.:

$$C = \bigwedge_{i=1}^{k} L_{x_iy_i}$$

(Computing this Intersection can be quite costly, so some care is required in choosing a good data structure for these lists). If this intersection Is non-empty, then the LPS's that lie in this intersection form the first set of conjectures and are therefore passed to the verifying procedures to determine if there are any counter-examples to these conjectures. If an LPS survives this check then it represents an intensional definition for Ri. For example, suppose R1 - R2 - R3 is an LPS which is in common to all the Lxy's. It is then conjectured that *R1* = R1/R2/R3. The intermediate nodes that were removed from the various paths underlying this LPS are possible bindings for the extensional variablesz1.*z2intheexpression*:

$$(x,y) \in R_1 \Rightarrow \exists z_1, z_2 \ni (x,z_1) \in R_1 \wedge (z_1,z_2) \in R_2 \wedge (z_2,y) \in R_3$$

On the other hand, the verifier determines whether for every x, y, and *z1* and z2 that satisfy the right-hand side of the above expression, the 2-tuple (x,y) is necessarily contained in Ri's extension- If this Is not true then the conjectured definition is considered to be "over-general".

A more interesting case arises when the intersection of all the Lxy's is empty. In such situations there is no simple compositional definition for Ri. Instead, its definition (If there is one) must involve a disjunct. An obvious strategy to pursue Is to keep intersecting the Lxy's until the Intersect becomes empty. At that point, undo the last intersection, leaving a non-empty intersect. Then, starting with this removed list, recommence forming the intersection with it and the next list and so on. For example, suppose we have L1,L2,L3,L4,L5,L6 as six lists of LPS's with the following properties:

$$
\begin{aligned}
L_{12} &= L_1 \wedge L_2 \\
L_{123} &= L_{12} \wedge L_3 = \emptyset \\
L_{34} &= L_3 \wedge L_4 \\
L_{345} &= L_{34} \wedge L_5 = \emptyset \\
L_{56} &= L_5 \wedge L_6
\end{aligned}
$$

We are left with three sets of non-empty intersections, namely L12,L34,L56 The conjectured definitions for Ri would then be of the form:

$$
\begin{aligned}
R_1 = \text{any LPS from } L_{12} &\vee \text{any LPS from } L_{34} \\
&\vee \text{any LPS from } L_{56}
\end{aligned}
$$

Thus an intensional definition of Ri would be a disjunct of relational compositions.

A moment's reflection on the above intersection process reveals that the outcome of these intersections can be critically dependent on the order in which the lists are intersected. For example, the lists L1,L2,L4 might very well form a non-void intersection and likewise the lists L3,L5,L6. If this were the case, we would then have unfolded a simpler set of conjectured definitions of the form:

$$R_1 = \text{any LPS from } L_{124} \vee \text{any LPS from } L_{356}$$

A dramatic example of this effect follows.

Given a moderately large data base of several hundred 2-tuples over thirty kinship relations, we were trying to discover some definitions of "Parent". The resultant intersections computed with the original ordering of the LPS lists (using a considerably more sophisticated algorithm for choosing what list to merge next) led to the disjunctive sets of conjectures found in Table Ia. However, a further search revealed a still better ordering yielding the conjectures found In Table Ib.

TABLE 1a

Conjectural Definitions for Parent Under Initial Ordering

				Grandparent/Nephew
				Grandparent/Niece
				Grandparent/Cousin/Nephew
				Grandparent/Cousin/Niece
		Sibling/Uncle		Grandparent/Son/Sibling
		Sibling/Sibling/Uncle		Grandparent/Nephew/Sibling
		Sibling/Offspring/Grandmother		Grandparent/Nephew/Brother
		Sibling/Son/Grandmother		Grandparent/Brother/Niece
Parent = Spouse/Sibling/Uncle	∨	Sibling/Uncle/Cousin	∨	Grandparent/Offspring/Sibling
		Sibling/Uncle/Sibling		Grandparent/Daughter/Sibling
		Sibling/Offspring/Grandparent		Grandparent/Niece/Sibling
		Sibling/Offspring/Grandfather		Grandparent/Niece/Brother
		Sibling/Son/Grandparent		Grandparent/Sibling/Niece
		Sibling/Son/Grandfather		Grandparent/Sibling/Nephew
				Grandparent/Sister/Nephew
				Spouse/Grandparent/Nephew
				Spouse/Grandparent/Niece

TABLE 1b

Conjectural Definitions for Parent Under "Optimal" Ordering

	Mother		Father
	Mother/Sibling		Father/Sibling
	Spouse/Father	∨	Spouse/Mother
Parent ≡	Wife/Father		Husband/Mother
	Mother/Daughter/Father		Father/Daughter/Mother
	Spouse/Father/Sibling		Spouse/Mother/Sibling
	Wife/Father/Sibling		Spouse/Mother/Sibling

Having great faith In Occam's Razor, we only attempt to verify conjectures that emerge from the minimal number of disjunctive intersections. Finding an optimal grouping of the L lists that lead to this minimal property iB combinatorlally equivalent to the classical covering problems , However, unlike many covering problems, a fairly simple heuristic turns out to be quite satisfactory (see Brown3 for more details).

A grouping for these lists which leads to the minimal number of non-null intersections induces a clustering of Ri's extension (i.e group together all the (x,y)'s that generated the LPS's of each disjunct). Often this clustering can indirectly induce a clustering or even a partition on either RI'S domain or range. This clustering should reflect some "internal" structure of the objects themselves. Considering the above Parent example reveals that the clustering of 2-tuples induced by the definitions of Table Ib shared the property that the first component of each of them were female. Likewise, the first component of the other 2-tuples were all male. But, as far as the system knew, no object in the domain had any distinguishing characteristics whatsoever. Thus, in some sense, the system had unfolded the necessary information to conjecture two unary predicates theretofore completely unknown to it. Justification of these predicates often requires supporting evidence which may be forthcoming if, in considering other relations, similar clusterings are induced.

Discovering Rules of Inference

We will concern ourselves only with discovering inference rules of a very restricted form. These rules will be either of the form:

1) $R_1/R_2/\ldots/R_n \Rightarrow R$

(where R itBelf could be also on the left)

or of the form:

2) $\Big|_{R_i} (R_1/R_2/\ldots/R_n) \Big|_{R_j} \Rightarrow \Big|_{R_i} R \Big|_{R_j}$

In the latter case, the relatione 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. In fact, these inference rules are directly usable bv the question-answering system detailed in Brown.

Discovering the first type of inference rule 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 realize that in creating an axiom system for a question-answerer we often want more than just a minimal independent set of axioms. We want axioms that enable us to answer "usual" questions without undue inferencing. Believing that typical questions often relate to subparts of the intensional definitions we Bought ways to use the structure of the relations themselves in isolating potentially "useful" rules.

One way to isolate these rules would be to use the intenaional definitions themselves. Specifically, any recursive definition is a particularly good candidate, such as:

Spouse/Parent => Parent

Another way t« generate potentially useful rules would be to discover definitions for a particular relation which turn out not to cover R's extension when their defining subspace is enlarged. An example of such a definition might be:

Cousin ≡ Sibling/Cousin
(Sibling/Cousin => Cousin)

which could arise from a data base in which every family had two children. But again, the key problem is finding ways that the structure of a relation can, itself, delimit "interesting" subsets of its own extensions so that we don't have to rely on chance for' providing such subspaces.

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, as mentioned above, splits the relation's extension into groups of 2-tuples covered by the particular disjunctive terms. For example, using the Parent example, we are lead to form the trivial rules:

Father => Parent
Mother => Parent
Husband/Mother => Parent
Wife/Father => Parent

However, the disjunctive definition underlying these rules has the property that the two disjuncts are disjoint. That is, if (x,y) is satisfied by one of the disjuncts, it necessarily is not contained in the other. By searching for disjunctive definitions whose disjuncts can overlap, and in which one of the disjuncts covers a maximal portion of the given relation's extension, we discover such rules as:

Parent/Sibling => Parent
Sibling/Offspring => Offspring
Offspring/Spouse => Offspring
Father/Cousin => Uncle

Another way to utilize the structure of a relation in isolating subspaces stems from the co-occurrence of compositional seouences in a given disjunct of its definition. For example, in considering a data base of over five hundred facts, we discovered the standard definition of Brother-in-Law:

```
                 Husband/Sis/Spouse      Husband/Sister
Brother/Spouse V Husband/Sib/Spouse   V  Husband/Sib
                 Spouse/Sister/Spouse    Spouse/Sister
```

Let us consider the last disjunct which asserts that:

1) Husband/Sister ≡ Husband/Sibling
2) Husband/Sister ≡ Spouse/Sister
3) Husband/Sibling ≡ Spouse/Sister

Line 1 suggests that in the "context" of "Husband" the relations "Sister" and "Sibling" function equivalently, or we could say that:

4) Sister => Sibling
5) Sibling => Sister,

when conditioned on the left with "Husband". Clearly, the notion of "context" is quite important for, although the rule "Sister => Sibling" could be independently discovered from the definition "Sibling ≡ Sister ∨ Brother", and hence would always be true, the rule "Sibling => Sister" is, in general, false. Using the expanded form of this rule i.e., (x, Sibling, y) => (x, Sister, 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 "Sister" and "Sibling". When restricted to this subdomain, these relations coincide (we again stress that the data base contains no such unary predicates as Male or Female).

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

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

The rule of inference which follows from this is:

$$\text{Son/Parent}\Big|_{\text{Niece}} \Rightarrow \text{Brother}$$

At first glance, the right-hand contextual relation (Niece) seems quite unnecessary, for a son of a parent would seem to be a brother. However, in exploring the "Son" relation with our system no such rule emerges. This is because the path seeking process underlying the construction of LPS's prevents the forming of paths with loops. Consequently, the possibility of the "son of a parent" being himself is necessarily prevented. However, in checking the over-generality of this definition, the verifier checks the entire extension of this conjecture and thus detects the need for an inequality predicate. 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" can be more subtle than the mere delimitation of the domains or ranges of relations. For example, when we were exploring a definition for Uncle, two disjuncts always co-occurred (a strong hint of the existence of structural cause). The definitions were:

Uncle ≡ Brother-in-Law/Parent
Uncle ≡ Husband/Sister-in-Law/Parent

At first glance, we might think that:

"Husband/Sister-in-Law => Brother-in-Law"

Considering Figure 1, we realize that although (y,z) ∈ Sister-in-Law, that (x,z) ∉ Brother-in-Law. However, the correct rule of inference requires a right context of |"Parent" (i.e., demanding that z must be a parent to someone):

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

Given the situation shown in Figure 2, we see that if z is a parent to q then in fact z must also have a spouse (say) w. Hence x would be a Brother-in-Law to w. In other words, the right hand context of "Parent" forces z to have a spouse w simply by asserting that z is the parent of q.

Figure 1

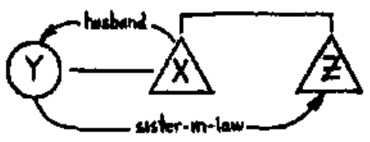
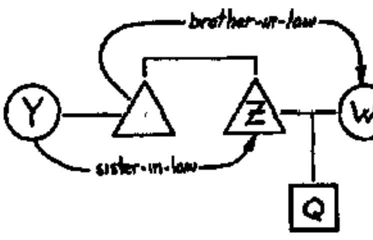


Figure 2



On Checking for Over-Generality

The result of intersecting the LPS lists is a collection of conjectural definitions for the particular relation R under study. A property of any such conjectural definition is:

$$P_1: \quad (x)(y)\ [(x,y) \in R \Rightarrow (x,y) \in \text{Conjectural Definition}]$$

As mentioned before, the problem of over-generality lies in verifying the converse of this property:

$$P_2: \quad (x)(y)\ [(x,y) \in \text{Conjectural Definition} \Rightarrow (x,y) \in R].$$

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[5] and, as such, will not be discussed here. Such verification is obviously 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 heuristics were applied, exhaustive techniques could be applied to the few remaining definitions so as to remove any doubt about their validity.

Our heuristic procedures are baaed 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 overlaps the range of y.

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

$$IR(x) = \{y \mid \exists z \ni (x,z) \in R \text{ and } (y,z) \in R\}$$

(note that inverse images do not form equivalence relations on the domain of the given relation). In a similar manner, we can define the inverse image of x with respect to a conjectural definition of R.

Our heuristic procedure picks an object x in the domain of the relation R being defined and computes both the sets IR(x) and I"Def"(x). It then checks to see if the set X"Def"(x) is equivalent to the set IR(x). If not, it rejects the "Def"; otherwise It chooses a new x from R's domain and repeats this procedure until the domain of R has been exhausted. A faster but less complete version of this heuristic doesn't bother to exhaust R'e domain but chooses only one element from each inverse image class. These checkB are only heuristic since they do not take into consideration the local connections of the relations within each inverse image class, but that is precisely why they are fast!

An example of the power and limitations of this heuristic may be seen from another date base (of over a hundred facta) analyzed by this system.

TABLE 2

Conjectured Definitions of "Niece" Prior to Hauristic 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

The definitions for the binary relation "Niece" were first explored. Table 2 lists the conjectural definitions resulting from intersecting the LPS lists, These definitions were then subjected to the above heuristic check which resulted in the definitions given in Table 3. Of the original fifteen intensional definitions eleven were rejected by our check and the remaining definitions were all correct.

TABLE 3

Conjectured Definitions of "Niece" Remaining After Heuristic Check

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

The need for developing heuristics for speeding up the verification of conjectured definitions is better appreciated whan we understand the complex interactions between the generation of conjectures and their subsequent verification. In fact, there is a constant switching of control between these two phases. For example, suppose the first set of definitions conjectured were subsequently disproved in the verification phase. Then all the paths in the data graph that lead to these conjectures would have to be temporarily "blocked" and a new set of conjectures would have to be invented. These new conjectures would then have to be verified, and so on, until all the conjectures passed the verification phase.

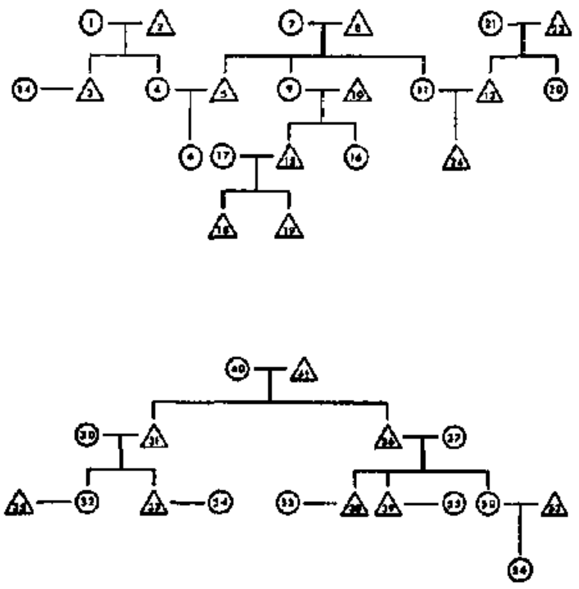
Our experiments have revealed that, for most of our test data, this process of conjecturing, verifying and blocking is often repeated a dozen or so times before a set of conjectures are formed which cannot be <u>disproved</u> within the data itself. Consequently, our system must typically verify many more conjectures than appear in the final output.

<u>Some Experiences</u>

We have applied this system (implemented in L on a 16-K PDP-9) to several types of data, the most common of which is kinship data. We primarily worked on kinship data because, although we know what to expect, the richness of its structure can present many challenges. Also, by choosing a domain that is intuitively understandable, it is easier to spot the more subtle relationships among the obvious. Although this domain is trivial in many respects, it was a great surprise to us to take a reasonably laree data base, generated — for test purposes — from the family tree shown in Figure 3 below, and perform a search for all the structural relationships that might have passes unnoticed from our biased viewpoint.

Figure 3

Family Tree Underlying an Extensional Data Base of Approximately 580 "Facts"



Two types of patterns could appear. On one hand, definitions might emerge that reflected idiosyncracies of this particular data which would "disappear" if we considered more data. On the other hand, completely valid definitions might appear which were logically equivalent to ones we already knew, but which might be "simpler" relative to some criterion. Both types of <u>unexpected patterns</u> may be seen in the first set of definitions discovered for the Uncle relation on this data base (see Table A). Clearly the first two of these definitions reflect the idiosyncracy that every uncle is married. The third definition which is complete yet involves no disjuncts and as such might be considered simpler than the standard two disjunct definition:

Uncle = Brother/Parent V Husband/Sibling/Parent

TABLE 4

First Definitions of Uncle

Uncle ≡	1)	Spouse/Aunt
	2)	Husband/Aunt
	*3)	Brother-in-Law/Parent
	4)	Spouse/Sister-in-Law/Parent
	5)	Husband/Sister-in-Law/Parent
	6)	Brother-in-Law/Spouse/Parent

It was not until these definitions were manually over-ridden and indeed a fair number more before our system came up with the "traditional" definition.

Appendices 1 and 2 give a feeling for the surprising multitude of structural definitions found on this data base. For example, Appendix 1 contains some of the three hundred definitions of "Uncle"which were discovered just on the above data base. The second appendix reveals the first set of definitions discovered for some of the other kinship relations. Note, for example, the seven universally valid definitions for "Cousin".

Not all our experimenting has been limited to kinship relations. Another domain that has been analyzed was made up of the extensional definitions of the accessibility relations between squares on a chess board for the various chess pieces. More precisely, the data base consists of a universe of 64 objects — representing the squares of a chess board — and a collection of extensionally defined relations. Some of these relations are geometric (e.g. East, Just-East, Positive-Diagonal) and the others are chess moves represented as binary relations. For example, (x,y)€ Knight means that a knight can move in one step from x to y. The task was to discover the definitions of the legal moves of the various chess pieces in terms of the other chess pieces and/ or in terms of the geometric relations. Although space precludes a thorough description of our findings, we indicate below some of the discovered definitions:

TABLE 5

Discovered Geometric Definitions

(Abbreviations: JW = Just-West, JE = Just-East, JN = Just-North, JS = Just-South)

```
                 JW/WEST
WEST ≡ JW   V    WEST/WEST
                 WEST/JW
```

Discovered Chess (Accessibility) Relations

```
Queen  =  Bishop    V Rook

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

*In this system, the user has the freedom to reject definitions even if they have been accepted by the verifier and thus search for additional definitions which involve more disjunctions than the current ones.

Weaknesses

There are several limitations to our system that deserve further attention. The first concerns the difficult area of characterizing when something cannot be true. All the definitions and axioms invented by our system express positive assertions. That is, our system never discovers the crucial fact that siblings can't marry and hence a father cannot be an uncle to his own child. Of course, we can hedge this problem by including just the right additional relations which cover precisely what can't hold over the original set but this quickly gets us into a combinatorial explosion. For example, we could include the relation "Non-Uncle" whose extension is the complement of Uncle and then perhaps we could uncover:

Father -> Non-Uncle.

Somehow such a solution seems unpleasing since once the positive assertions have been characterized one might hope that they could he combined with a few negative assertions to imply all the potential negative assertions.

The second major limitation is that nur system expects its sample data base to be complete, i.e. missing no data and containing no erroneous data. There are numerous wavs to circumvent this limitation, some of which have been implemented. For example, the verification phase not only checks a conjecture but Isolates, for any rejection, the counter-examples. The user can then decide whether or not to over-rule the verifier by deleting from the data hase these "counter-examples." However, a considerably more provocative approach to the problem should be possible. Relying on Occam's Razor, we might consider the entire set of discovered definitions and see how a particular set of changes on the data base affects the complexity of not only the definitions of the relation under Btudy, but also the collection of definitions for all the relations. Since altering a given relation's extension affects not only its own definition but also the definitions of all the other relations definable in terms of this relation, we would expect a dramatic global simplification for the "correct" changes to the data.

Further Plans

In addition to exploring the above issues we plan to investigate how to achieve more of a synergistic effect between the user and the computer in seeking the underlying structure of his data. The system is being re-implemented in LISP on a PDP-10 in hopes of having a friendlier environment for exploring the symbiotic uses of this kind of a theory formation system.

Acknowledgement

I am deeply Indebted to Dr-. Robert K, Lindsay for defining this research problem and for helping me understand some of its ramifications.

References

1. Plotkin, G. D. "A Further Note on Inductive Generalization," in Machine Intelligence, Vol. 6 New York, New York: American Elsevier Publishing, 1971.

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

3. Brown J. S. Unpublished Doctoral Dissertation, University of Michigan; or ICS TR #17, University of California at Irvine.

4. Brown, J. S. "Question Answering Through Parsing" ICS TR #24, University of California at Irvine.

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

Appendix 1

Definitions of Uncle Arranged in Order of Output

(1)

```
                Spouse/Aunt
                Husband/Aunt
                Bro-in-Law/Parent
Uncle         = Spouse/Sis-in-Law/Parent
                Husband/Sis-in-Law/Parent
                Bro-in-Law/Spouse/Parent
```

(2)

```
                                                      Sibling/Father
                                                      Brother/Father
                                                      Fath-in-Law/Husband/Cousin
                                                      Sibling/Father/Sibling
                Bro-in-Law/Father                     Brother/Father/Sibling
Uncle         = Bro-in-Law/Spouse/Mother        V     Sibling/Spouse/Mother
                Bro-in-Law/Husband/Parent             Sibling/Husband/Parent
                Bro-in-Law/Husband/Mother             Sibling/Husband/Mother
                                                      Brother/Spouse/Mother
                                                      Brother/Husband/Parent
                                                      Brother/Husband/Mother
                                                      Sibling/Fath-in-Law/Spouse
                                                      Brother/Fath-in-Law/Spouse
```

(3)

```
                                                      Spouse/Sister/Parent
Uncle         = Brother/Parent                  V     Husband/Sibling/Parent
                Brother/Spouse/Parent                 Husband/Sister/Parent
                                                      Bro-in-Law/Sister/Aunt
```

Appendix 2

Synopsis of Discovered Definitions for Data Base Underlying Figure 3

(1)

```
                Offspring/Uncle
                Offspring/Aunt
                Offspring/Spouse/Aunt
                Offspring/Husband/Aunt
Cousin        = Offspring/Spouse/Uncle(*)
                Offspring/Wife/Uncle(*)
                Offspring/Sibling/Parent
                Offspring/Bro-in-Law/Parent
                Offspring/Sis-in-Law/Parent
```

*Starred definitions represent definitions that are not universally valid but are valid for the given data base.

Appendix 2 (con't)

(2)

Fath-in-Law =
Father/Spouse
Spouse/Moth-in-Law
Husband/Moth-in-Law
Father/Offspring/Moth-in-Law
Father/Son/Moth-in-Law
Father/Sibling/Spouse(*)
Husband/Mother/Spouse
Husband/Parent/Spouse
Husband/Mother/Spouse

Moth-in-Law =
Mother/Spouse
Spouse/Fath-in-Law
Wife/Fath-in-Law
Mother/Sibling/Spouse
Wife/Father/Spouse
Wife/Parent/Spouse
Spouse/Father/Spouse
Mother/Father/Fath-in-Law
Mother/Offspring/Fath-in-Law

(3)

Grandchild =
Offspring/Offspring
Offspring/Spouse/Offspring
Offspring/Offspring/Spouse
Offspring/Sibling/Offspring(*)

(4)

Parent =
Father
Spouse/Mother
Husband/Mother

V

Mother
Spouse/Father
Wife/Father

(5)

Offspring =
Son
Son/Spouse

V

Daughter
Daughter/Spouse

(6)

Grandparent =
Parent/Parent
Parent/Spouse/Parent
Parent/Sibling/Parent
Spouse/Parent/Parent

(7)

Grandfather =
Fath-in-Law/Parent
Father/Parent
Spouse/Grandmother
Husband/Grandparent
Husband/Grandmother
Fath-in-Law/Spouse/Parent
Father/Spouse/Parent
Father/Offspring/Grandparent
Father/Offspring/Grandmother
Father/Son/Grandparent
Father/Son/Grandmother
Father/Sibling/Parent
Spouse/Mother/Parent
Husband/Parent/Parent
Husband/Mother/Parent
Spouse/Moth-in-Law/Parent
Husband/Moth-in-Law/Parent
Fath-in-Law/Daughter-in-Law/Grandparent
Fath-in-Law/Daughter-in-Law/Grandmother

(8)

Nephew =
Son/Sister
Son/Spouse/Sister
Son/Husband/Sibling
Son/Husband/Sister

V

Son/Sis-in-Law
Son/Spouse/Sis-in-Law
Son/Husband/Sis-in-Law