

A PARADIGM FOR REASONING BY ANALOGY

Robert E. Kling
Stanford Research Institute
Menlo Park, California
U.S.A.

ABSTRACT

A paradigm enabling heuristic problem solving programs to exploit an analogy between a current unsolved problem and a similar but previously solved problem to simplify its search for a solution is outlined. It is developed in detail for a first-order resolution logic theorem prover. Descriptions of the paradigm, implemented LISP programs, and preliminary experimental results are presented. This is believed to be the first system that develops analogical information and exploits it so that a problem-solving program can speed its search.

INTRODUCTION

An intelligent man thinks deeply and learns from his past experiences. Contemporary theorem-proving and problem-solving systems are continually designed to think ever more deeply and to ignore their past completely. A problem solver designed in any of the contemporary paradigms (such as resolution (1), GPS (2), and REF-ARF (3)) solves the same problem the same way each time it is presented. A fortiori, they are unable to exploit similarities between new and old problems to hasten the search for a solution to the new one. ZORBA, outlined in this paper, is a paradigm for handling some kinds of analogies.* This is the first instance of a system that derives the analogical relationship between two problems and outputs the kind of information that can be usefully employed by a problem-solving system to expedite its search. As such, ZORBA is valuable in three ways:

- (1) It shows how nontrivial analogical reasoning (AR) can be performed with the technical devices familiar to heuristic programmers, e.g., tree search, matching, and pruning.

* In Ref. (4), I show that there are several kinds of analogies from an information-processing point of view. We should hardly expect one paradigm to include them all. Restrictions on the varieties of analogy handled by ZORBA are described in the section entitled "Necessary Conditions for an Analogy."

- (2) It provides a concrete information-processing framework within which and against which one can pose and answer questions germane to AR.
- (3) Since it is implemented (in LISP), it is available as a research tool as well as a gedanken tool.

The last two contributions are by far the most important, although our attention will focus upon the first. In the 50's and 60's, many researchers felt that analogical reasoning would be an important addition to intelligent problem-solving programs. However, no substantial proposals were offered, and the idea of AR remained rather nebulous, merely a hope. ZORBA may raise more questions of the "what if?" variety than it answers. However, now, unlike 1968, we have an elementary framework for making these questions and their answers operational.

ZORBA PARADIGM

Although prior to ZORBA there were no concrete paradigms for AR, there was an unarticulated undeveloped paradigm within the artificial intelligence zeitgeist. Suppose a problem solver had solved some problem P and has its solution S . If a program is to solve a new, analogous P_A , it should do the following:

- (1) Examine S and construct some plan (schema) S' that could be used to generate S .
- (2) Derive some analogy $G: P \rightarrow P_A$.
- (3) Construct $G^{-1}(S') = S'_A$.
- (4) Execute S'_A to get S_A , the solution to P_A .

If P was solved by executing a plan, then S' would be available and step (1) could be omitted. Although nobody has explicated this idea in publications, from various conversations with workers in the field, I believe that the preceding description is close to the paradigm that many would have pursued. As such, it constitutes the (late-60's) conventional wisdom of artificial intelligence. Certainly this (planning) paradigm is attractively elegant! However, in 1969, when this research was begun, it was an inappropriate approach for two reasons:

- (1) There are no planning-oriented problem solvers that are fully implemented and operate in a domain with interesting nontrivial analogies.* This state of

* PLANNER at MIT and QA4 at SRI are two current planning-oriented problem solvers that are under development. The first is partially implemented and the second exists only on paper. It is not yet clear what problem-solving power PLANNER will have, and how effective it will be in domains with interesting analogies.

R. E. KLING

affairs probably will change in the next few years, but it now renders difficult any research that depends on the existence of such a system.

- (2) Given the plans generated by such a system, it is hard to know a priori at what level of generality the derived analogy will map into an executable analogous plan.* If S'_A fails, is G too strong, or wrong? Should G be modified and a variant S''_A computed, or should the system keep G , and just back up its planner and generate an alternative subplan using its own planning logic? At best this is a rather complex research issue which would involve a good planning-oriented problem solver as an easily accessible research tool. At worst, the preceding paradigm may be too simple and the development of a suitable G may be interactive with how much successful problem-solving has proceeded so far. (A complete G should not be attempted before some problem solving begins and is extended as needed in the course of solving P_A .)

Happily, there is an alternative approach that circumvents the preceding difficulties. Consider a system that has solved some problem P and is posed with a new (analogous) P_A to solve. Clearly, it must operate on some large data base sufficient to solve both P and P_A . (See Figure 1.) In addition to the subbase for solving P and P_A there

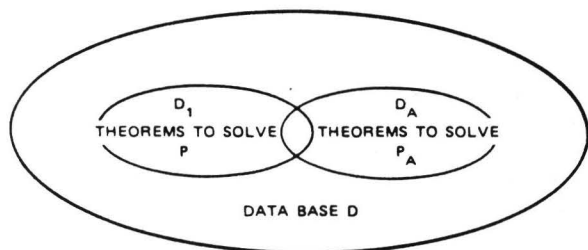


FIGURE 1 VENN DIAGRAM OF THEOREMS IN DATA BASE

are likely to be even more theorems in the set $D - (D_1 \cup D_A)$. Now, given P it is impossible to infer a minimal D_1 . In practice, a user may select some D_2 s.t. $D_1 \subseteq D_2 \subseteq D$ which the problem solver will access to solve P . If one studies the searches that problem solvers generate when

they work with nonoptimal data bases, it is obvious that many of the irrelevant inferences that are generated are derived from the data-base assertions (theorems, axioms, facts) in $D - D_1$ (or $D_2 - D_1$). In fact, as the number of theorems irrelevant to the solution P becomes large, the number of irrelevant inferences derived from this set begins to dominate the number of irrelevant inferences generated within D_1 and its descendants alone.* In fact, while a problem solver might solve P given an adequate and small D_1 , it may be swamped and run out of space before a solution given a D_2 that is much larger than needed.† Clearly, one effective use of analogical information would be to select a decent subset D_2 of D such that $\text{size}[D_1] \leq \text{size}[D_2] \ll \text{size}[D]$. For example, a typical theorem in algebra provable by QA3⁵--a resolution logic theorem proof--may require only 10 axioms (D_1) while the full algebraic data base has 250 axioms. If a system could select a D_2 such that $\text{size}[D_2] = 15$ axioms, a massive saving in search could be had. In fact, the theorem that would be unprovable on a D with $\text{size}[D] = 250$ would now be provable.

A second kind of information that would be useful to help solve P_A would be a set of lemmas (or subgoals) $L_1, \dots, L_j$ whose analogs $G(L_1), \dots, G(L_j)$ could be solved by the system before attempting P_A .

At this point I will not discuss how to recognize a lemma‡ and generate its analog;§ instead, I merely want to note that lemmas may be effectively used without using a planning language

* Even given an optimal data base, a problem solver will generate some irrelevant inferences.

† In general, automatic problem solvers and theorem provers run out of space rather than time when they fail to solve a problem. Ernst(2) emphasizes this point with regard to GPS, and I have had similar experiences with QA3(5), a resolution logic theorem prover.

* Recognizing lemmas depends upon the problem-solving system. For example, in resolution logic, some good criteria for lemmahood are:

- (1) A ground unit used more than twice (or k times) in a proof.
- (2) A unit that is a merge.
- (3) A clause that is the "least descendant" of more than 2 (or k) units.

§ Generating a lemma depends upon the system's ability to associate variables with variables and that may be tricky when skolem functions are introduced.

* See Ref. 4 for a discussion of this issue.

R. E. KLING

that forces backup in case of failure. Suppose we somehow get $G(L_1), \dots, G(L_j)$. A typical planner would order the $G(L_1)$, e.g., $G(L_1), G(L_2) \dots$ etc., attempt to solve them in sequence, and stop if any lemma fails to be solved. In contrast, we merely need to attempt each $G(L_i)$. If we get a solution, add $G(L_i)$ to the data base (like a theorem)* and continue with the next lemma. If we fail, continue anyway. At worst, we wasted some computation time. Each useful $G(L_i)$ decreases the number of steps in the solution of P_A and may decrease the depth of the solution tree. Thus, lemmas are helpful in getting a faster solution. Note, however, that a successful $G(L_j)$ need not be used in the solution of P_A . It is merely available. Thus, we are not bound by the fail-backup orientation of sequential planning logics.

In summary, if we use analogical information to modify the environment[†] in which a problem solver operates, we can effectively abbreviate the work a problem solver must perform. Of course, a well-chosen environment will always lead to a more efficient search. Usually, we have no idea how to tailor a subenvironment automatically to a particular problem. Here we do it by exploiting its analogy with a known solved problem. Now, the representations used, the analogy-generating programs, and the types of additional information output will depend upon the problem-solving system (and even the domain of application). Any further discussion needs to specify these two items.

APPLICATIONS TO RESOLUTION LOGIC

The preceding discussion referred to any problem solver and is just a proposal. Computer programs have been implemented to apply this paradigm to a resolution logic theorem prover, QA3.(5) For the class of analogies these programs handle, this is an accomplishment. When we begin to focus

upon a particular paradigm, two issues are more easily resolved:

- (1) What kinds of information are most useful to provide to the problem solver?
- (2) Which representations shall we use to describe the analogies and handle the necessary data?

Resolution logic is an inference rule whose statements are called clauses.* (1),(5) Thus, a resolution-oriented analogizer will deal with clauses and their descriptions. In contrast, GPS uses sets of objects to describe its states, and we would expect that an analogy system devoted to GPS would deal with (complex) objects and their attributes. Table 1 contrasts the kinds of information helpful to QA3 and GPS. An analogy facility developed for GPS would be oriented to its peculiar information structures instead of clauses and axioms indigenous to resolution.

Table 1

KINDS OF INFORMATION HELPFUL TO QA3 and GPS

QA3 (Resolution)	GPS
Relevant axioms	Relevant operators
Expected predicates	Abbreviated difference table
Lemmas	Subgoals
Admissible function nestings	Restrictions on operator applications

I want to digress briefly and describe the kinds of theorems that the implemented system, ZORBA-I, tackles. Briefly, they are theorem pairs in domains that can be axiomatized without constants (e.g., mathematics) and that have one-one maps between their predicates. The theorems are fairly hard for QA3 to solve. For example, ZORBA-I will be given proof of the theorem

- T1. The intersection of two abelian groups is an abelian group and is asked to generate an analogy with
- T2. The intersection of two commutative rings is a commutative ring.

* In fact, under some conditions, the axioms used to solve $G(L_i)$ may be deleted from D_2 so that size $[D_2]$ is decreased, and $G(L_i)$ is not attempted again inadvertently during the solution of P_A .

† Here environment is synonymous with data base. But it can also include permissible function orderings (in predicate calculus) and other kinds of restrictive information. Each rule restricting the "environment" could be translated into an equivalent new decision rule restricting the application of the inference procedures of the problem solver. However, I find it easier to think of ZORBA in terms of modified environments rather than (the equivalent) modified decision rules.

* A clause is an element in the conjunctive normal form of a skolemized wff in the predicate calculus. For example: $\neg \text{person } [x]$
 $\vee \text{father } [g(x); x]$ is the clause associated with:
 $\forall x \text{ person } [x] \rightarrow \exists y \text{ father } [y; x]$ (every person has a father).

R. E. KLING

Given

- T3. A factor group G/H is simple iff H is a maximal normal subgroup of G .

Generate an adequate analogy with

- T4. A quotient ring A/C is simple iff C is a maximal ideal in A .

None of these theorems are trivial for contemporary theorem provers. (See Table 2, in a later section, for a listing of additional theorem pairs.) T_1 has a 35-step proof and T_3 has a 50-step proof in a decent axiomatization. A good theorem prover (QA3) generates about 200 inferences in searching for either proof when its data base is minimized to the 13 axioms required for the proof of T_1 or to the 12 axioms required for the proof of T_3 . If the data base is increased to 20-30 reasonable axioms, the theorem prover may generate 600 clauses and run out of space before a proof is found. Note also that the predicates in the problem statement of these theorems contain only a few of the predicates used in any proof. Thus, T_1 can be stated using only {INTERSECTION; ABELIAN}, but a proof requires {GROUP; IN; TIMES; SUBSET; SUBGROUP; COMMUTATIVE} in addition. Thus, while the first set is known to map into {INTERSECTION, COMMUTATING}, the second set can map into anything.

Figure 2 shows a set P including all the predicates in the data base.

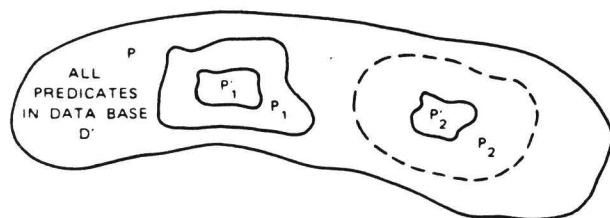


FIGURE 2 VENN DIAGRAMS OF RELATIONS IN STATEMENTS T , T_A , AND D'

We know P_1' and P_2' , the sets of predicates in the statements of the new and old theorems, T_A and T . In addition, we know the predicates P_1 in some proof of T (since we have a proof at hand). We need to find the set P_2 that contains the relations we expect in some proof of T_A , and we want a map G : $G(P_1) = P_2$.

Clearly, a wise method would be to find some G' , a restriction of G to P_1' such that $G'(P_1') = P_2'$. Then incrementally extend G' to G_1' , G_2' , ... each on larger domains until some $G'(P_1) = P_2$. ZORBA-I does this in such a way that each incremental extension picks up new clauses that could be used in a proof of T_A . In fact, if we get no new clauses from an extended G_j' , that may be reason to believe that G_j' is faulty. The next sections will describe the generation algorithm in a little more detail.

ZORBA'S REPRESENTATION OF AN ANALOGY

In the preceding sections I have implied that an analogy is some kind of mapping. The ZORBA paradigm--e.g., using an analogy to restrict the environment in which a theorem prover works--does not restrict this mapping very much. For different intuitively analogous theorem pairs, this mapping would need to be able to associate predicates (and axioms) in a one-one, one-many, or many-many fashion, possibly dependent upon context. For other theorem pairs, one-one mappings and context-free mappings are adequate. ZORBA-I is a particular set of algorithms that restricts its acceptable analogies to those which map predicates one-one with no context dependence. It allows one-many associations between axioms; e.g., one axiom of the proved theorem is associated with one or more axioms that will be used to prove the new, analogous theorem. More explicitly, a ZORBA-I analogy G is a relation $G^P \times G^C \times G^V$, where:

- (1) G^P is a one-one map between the predicates used in the proof of the proved theorem T and the predicates used in the proof of the unproved theorem T_A .
- (2) G^C is a one-many mapping between clauses. Each clause used in the proof of T is associated with one or more clauses from the data base D that ZORBA-I expects to use in proving T_A .
- (3) G^V is a many-many mapping between the variables that appear in the statement of T and those that appear in the statement of T_A .

Different sections of ZORBA-I use these various maps, e.g., G^V and/or G^P and/or G^C . Usually I will drop the superscript and simply refer to "the analogy G ." Thus "the analog of an axiom ax_k under analogy G " should be understood to mean $G^C[ax_k]$, and will often be mentioned simply as "the analog of ax_k ."

In the previous section I refer to a sequence of analogies $G_1, \dots, G_k$. ZORBA-I usually does not develop G^C in one step. Rather, it

R. E. KLING

incrementally extends some limited analogy into one that maps a few more variables, predicates, or clauses. This process is described in full detail in the next few sections. Here, I just want to define several terms that refer to this process. When I refer to "the analogy between T and T_A " I refer to a mapping that includes every variable in the statement of T , and every predicate and clause used in the proof of T . This "complete" mapping is obtained as the final step of a sequence of mappings that contain the associations of some predicates and some clauses. I refer to these incomplete mappings as "partial analogies." In addition, we are concerned with an important relationship between two (partial) analogies. A (partial or complete) analogy G_k is an extension of a partial analogy G_j if some of G_j , e.g., G_j^p , G_j^c , G_j^v , is a submap restriction of the corresponding submap G_k to a smaller domain. Intuitively, when we add a new predicate or clause association to G_j so as to create G_k , we say that G_j has been extended to G_k . We are now ready to survey ZORBA-I.

AN OVERVIEW OF THE ANALOGY-GENERATING ALGORITHM

I want to describe the ZORBA-I algorithm in two stages, first briefly in this section and then in greater detail in the following two sections. I will precede these descriptions by some background on the representations and information available to the system.

ZORBA-I is presented with the following:

- (1) A new theorem to prove, T_A .
- (2) An analogous theorem T (chosen by the user) that has already been proved.
- (3) Proof[T] that is an ordered set of clauses c_k s.t. $\forall k$ c_k is either
 - (a) A clause in $\neg T$
 - (b) An axiom
 - (c) Derived by resolution from two clauses

$$c_i \text{ and } c_j \quad j < k \text{ and } i < k.$$

These three items of information are problem-dependent. In addition, the user specifies a "semantic template" for each predicate in his language. This template associates a semantic category with each predicate and predicate-place and is used to help constrain the predicate mappings to be meaningful. For example, STRUCTURE[SET; OPERATOR] is associated with the predicate "group." Thus, ZORBA-I knows that "A" is a set and "*" is an operator when it sees group[A; *]. Currently, the predicate types (for algebra) are STRUCTURE, RELATION, MAP, and REL-STRUCTURE; the variable types are SET, OPERATOR, FUNCTION, and OBJECT.

In addition, ZORBA-I can make up a description descr[c] of any clause c according to the following rules regarding the predicates of c .

- (1) $\forall_p$ s.t. p and $\neg p$ appear in c , $\text{impcnd}[p] \in \text{descr}[c]$.
- (2) $\forall$ s.t. p appears in c , $\text{pos}[p] \in \text{descr}[c]$.
- (3) $\forall_p$ s.t. $\neg p$ appears in c , $\text{neg}[p] \in \text{descr}[c]$.

Thus, the axiom, every abelian group is a group,

$$\text{e.g., } \forall(x*) \text{ abelian } [x; *] = \text{group } [x; *],$$

is expressed by the clause

$$c_1: \neg \text{abelian } [x; *] \vee \text{group } [x; *],$$

which is described by

$$\text{neg } [\text{abelian}], \text{pos } [\text{group}]$$

Each element of a description, e.g., pos[group], is a "feature" of the description. Each feature corresponds to one predicate, so the number of features in a clause equals the number of predicates in the clause. The theorem, the homomorphic image of a group is a group, e.g.,

$$\begin{aligned} &\forall (x y * _1 * _2 \varphi) \\ &\text{hom } [\varphi; x; y] \wedge \text{group } [x; * _1] \\ &= \text{group } [y; * _2] \end{aligned}$$

is expressed by the clause

$$c_2: \neg \text{hom } [\varphi; x; y] \vee \neg \text{group } [x; * _1] \vee \text{group } [y; * _2]$$

and is described by

$$\text{neg}[\text{hom}], \text{impcnd}[\text{group}]$$

Two different clauses may have the same description. Let:

$$\begin{aligned} c_3: &\neg \text{intersection}[x; y; z] \vee \text{subset}[x; y] \\ c_4: &\neg \text{intersection}[x; y; z] \vee \text{subset}[x; z] \end{aligned}$$

Then:

$$\text{descr}[c_3] = \text{descr}[c_4] = \text{neg}[\text{intersection}], \text{pos}[\text{subset}]$$

Clause descriptions are used to characterize the axioms whose analogs we seek. ZORBA-I selects as analogs clauses that have descriptions that are close to the analogs of the descriptions* of axioms in the known axiom set. Although in a special context ZORBA-I actually uses an ordering relation on a set of descriptions to find a "best clause," it usually exploits a simpler approach. We will say that a clause c satisfies a description d iff $d \subseteq \text{descr}[c]$. Thus, several clauses may satisfy the same description.

*The "analog of a description" is defined later.

R. E. KLING

Let:

$$c_5: \neg \text{intersection}[x;y;z] \vee \neg \text{group}[y;*] \\ \vee \neg \text{group}[z;*] \vee \text{group}[x;*]$$

$$c_6: \neg \text{subgroup}[x;y;*] \vee \neg \text{subset}[x;y]$$

Then, the following statements are true:

- (1) $\{c_2, c_5\}$ satisfy $\text{impcnd}[\text{group}]$
- (2) $\{c_1, c_2, c_5\}$ satisfy $\text{pos}[\text{group}]$
- (3) c_1 satisfies $\text{neg}[\text{abelian}]$, $\text{pos}[\text{group}]$
- (4) $\{c_3, c_4, c_6\}$ satisfy $\text{pos}[\text{subset}]$
- (5) c_6 satisfies $\text{neg}[\text{subgroup}]$, $\text{pos}[\text{subset}]$
- (6) No clause of these six satisfies $\text{pos}[\text{intersection}]$

Clearly, if a description contains only a few features, then several clauses may satisfy it.

The semantic templates are used during both the INITIAL-MAP (when the predicates and variables in the theorem statements are mapped) as well as in the EXTENDER, which adds additional predicates needed for the proof of T_A and finds a set of axioms to use in proving T_A . The clause descriptions are used only by EXTENDER.

I intend the brief description that follows to provide an overview of ZORBA-I in preview to the next two sections of text, which describe it in considerable detail. In addition, this preview section may be a helpful "roadmap" for reference when the reader immerses himself in the details that follow later on.

ZORBA-I operates in two stages. INITIAL-MAP is applied to the statements of T and T_A to create an G_1^P , which is used by EXTENDER to start its sequence of G_j^P and G_j^V , which terminate in a complete G . INITIAL-MAP starts without a priori information about the analogy it is asked to help create. Both G^P and G^V are empty when it begins. It uses the system of the wffs that express T and T_A as well as the restrictions imposed by the semantic categories to generate G_1^P and G_1^V that include all the predicates and variables that appear in the two wffs. For example, the statements of $T_1 - T_2$ can contain three of the nine predicates used in $\text{proof}[T_1]$ and the statements of $T_3 - T_4$ can contain five of the 12 predicates used in $\text{proof}[T_3]$. In brief, it provides a starting point from which EXTENDER can develop a complete G .

The INITIAL-MAP uses a rule of inference called $\text{ATOMMATCH}[\text{atom}_1; \text{atom}_2; G]$, which extends analogy by adding the predicates and mapped variables of atom_1 and atom_2 to analogy G . Thus, ATOMMATCH now limits ZORBA-I to analogies where atoms* in the statements of T and T_A map one-one. INITIAL-MAP is a sophisticated search program

that sweeps ATOMMATCH over likely pairs of atoms, one of which is from the statement of T , the other from the statement of T_A . Alternative analogies are kept in parallel (no backup), and INITIAL-MAP terminates when it has found some analogy that includes all the predicates in the theorem statements. This one is output as G_1^P .

EXTENDER accepts a partial analogy generated by INITIAL-MAP and uses it as the first term in a sequence of successive analogies G_j . The axioms used in $\text{proof}[T]$ are few in comparison to the size of the large data base and comprise the "domain" for a complete G^C . For each axiom used in $\text{proof}[T]$, we want to find a clause from the data base that is analogous to it. The axioms used in $\text{proof}[T]$ are called AXSET and are used by EXTENDER in a special way. Each partial analogy G_j^P is used to partition AXSET into three disjoint subsets called $\text{ALL}[G_j]$, $\text{SOME}[G_j]$, and $\text{NONE}[G_j]$.

If all the predicates in an axiom $ax_k \in \text{AXSET}$ are in G_j^P , then ax_k is in $\text{ALL}[G_j]$; if some of its predicates are in G_j^P , then ax_k is in $\text{SOME}[G_j]$; and if none of its predicates are in G_j^P , then ax_k is in $\text{NONE}[G_j]$. For brevity, these sets will be called ALL , SOME , and NONE , and their dependence on G_j will be implicit. This partition is trivial to compute, and initially, none or a few ax_k are in ALL , and most ax_k belong to SOME and NONE . We want to develop a sequence of analogies G_j , $j = 1, \dots, n$, that contain an increasingly larger set of predicates and their analogs. If an axiom is contained in ALL , then by definition we know the analogs of each of its predicates. It cannot assist us in learning about new predicate associations. In contrast, we know nothing about the analogs of any of the predicates used in axioms contained in NONE . Analog clauses for these axioms are hard to deduce since we have no relevant information to start a search. Unlike these two extreme cases, the axioms in SOME are especially helpful and will become the focus of our attention. For each such axiom we know the analogs of some of its predicates from G_j . These provide sufficient information to begin a search for the clauses that are analogous to them. When we finally associate an axiom with its analog, we can match their respective descriptions and associate the predicates of each that do not appear on G_j^P . We can extend G_j to G_{j+1} , and thus the analogs of axioms on SOME provide a bridge between the known and the unknown, between the current G_j and a descendent G_{j+1} . When EXTENDER has satisfactorily terminated, $\text{ALL} = \text{AXSET}$, $\text{SOME} = \text{NONE} = \emptyset$. So the game becomes finding some way to systematically move axioms from NONE to SOME to ALL in such a way that for each ax_k moved, some analog $G_j[ax_k] = ax'_k$ is found that can be used in the proof of T_A . Moreover,

* Atoms, not predicates.

each new association of clauses should help us extend $G_j \rightarrow G_{j+1}$ by providing information about predicates not contained in G_j .

A DETAILED DESCRIPTION OF INITIAL-MAP

At heart, ZORBA-I is a heuristic program designed to generate analogies between theorem pairs stated in a subset of predicate calculus. It has been designed and implemented in a fairly modular manner to facilitate understanding and ease of generalization. Thus, much of the system can be described in algorithmic terms. In this section I hope to blend some appreciation of the heuristic foundations of the program while describing its operation with algorithmic clarity. ZORBA-I uses an interesting set of searching and matching routines, which have been empirically designed, generalized, and tested on a set of problem pairs ($T_1 - T_2$ and $T_3 - T_4$ are fair representatives of this set). The control structures of INITIAL-MAP and EXTENDER have been designed to pass fairly similar structures to the various match routines (described below). Thus, the following descriptions will cover cases where the structures to be mapped are fairly similar. For example, most of the routines that match sets of items assume that the sets are of equal cardinality and that they will map one-one. Such assumptions are valid for a large class of interesting analogies (such as the group-ring analogy in abstract algebra) and simplify the description of the various procedures. Analogies that require weaker assumptions and more complex procedures are described elsewhere. (6)

In the previous section I motivated the design of INITIAL-MAP and EXTENDER, which generate a restricted analogy and expand it to cover all the relations and axioms necessary for the new proof. ZORBA-I can be easily expressed in terms of these two functions as follows:

zorba₁[newwff;oldwff;AXSET]^{*} :=

- (1) Set analogies to the list of analogies generated by initial map[newwff;oldwff].
- (2) Apply extender[analogy; AXSET] to each analogy or analogies.
- (3) Return the resultant set of analogies.

The preceding description allows that there may be more than one analogy generated by either INITIAL-MAP or EXTENDER. In practice, however, each tends to generate but one (good) analogy. In the following paragraphs I will describe

INITIAL-MAP in some detail. EXTENDER will be discussed in the next section.

INITIAL-MAP is designed to take two first-order predicate calculus wffs and attempt to generate a mapping between the predicates and variables that appear in them. The variable mapping information is used to assist INITIAL-MAP in mapping predicates in cases of seeming ambiguity; INITIAL-MAP outputs a set of associated predicates that appear in the statements of T_A and T . This restricted mapping is used as a starting analogy by EXTENDER, which finds a complete mapping for all the predicates used in proof[T]. As a byproduct EXTENDER finds analogs for each of the axioms on AXSET. INITIAL-MAP (unlike EXTENDER) does not reference AXSET, the set of axioms used to prove T , and is symmetric with respect to caring which wff represents the proved or unproved theorem. INITIAL-MAP uses atommatch[atom₁;atom₂; G] as a rule of inference to add the predicate/variable information to analogy G. As its name hints, ATOMMATCH matches the predicates and variables of its atomic arguments and adds the resultant mapping to the developing analogy (G).

ATOMMATCH is used as an elementary operation by every matching routine in the INITIAL-MAP system (Figure 3). Thus, we will discuss it first

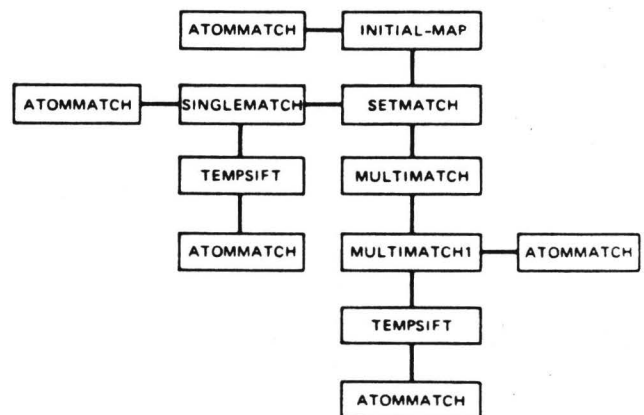


FIGURE 3 HIERARCHY OF MATCHING ROUTINES CALLED BY INITIAL-MAP

and then consider how INITIAL-MAP is organized to apply it intelligently. Consider how we might write an ATOMMATCH. Suppose, atom₁ and atom₂ are of the same order (same number of variables) and each variable place in each atom has the same semantic type. For example, let

atom₁ = intersection[x₁;x₂;x₃]
 atom₂ = intersection[y₁;y₂;y₃]

* AXSET is the set of axioms that appears in proof[T].

R. E. KLING

Clearly, we want

$$\text{intersection} \leftrightarrow \text{intersection}$$

and $x_i \leftrightarrow y_i^*$, $i = 1, 2, 3$.

So, if $\text{atom}_1 = p[x_1; \dots x_n]$

and $\text{atom}_2 = q[y_1; \dots y_m]$

and $p = q$ (thus, $n = m$)

we will set $p \leftrightarrow q$

and $x_i \leftrightarrow y_i$, $i = 1, 2, \dots, n$.

So far ATOMMATCH is quite trivial. Suppose, however, $p \neq q$ or $n \neq m$.

For example, let $\text{atom}_1 = \text{group}[x; *_1]$

and $\text{atom}_2 = \text{ring}[y; *_2; +_2]$.

Clearly we want to associate the set x with the set y and the operator $*_1$ with either or both of $*_2$ and $+_2$. ATOMMATCH can know which variables represent sets, etc., by checking the semantic templates associated with group and ring. Now, the template associated with group is structure [set; operator] while that associated with ring is structure [set; operator; operator]. We will map variables with each other so as to preserve predicate place ordering and semantic type. To handle the unequal number of variables, we will temporarily expand the atom group $[x; *_1]$ to include a dummy variable of type operator, "dummyop," and will rewrite it as $\text{group}[x; *_1; \text{dummyop}]$. The symbol "dummyop" is used to expand either (or both) atoms to be of the same order and a variable (possibly dummy) of the same semantic type in corresponding places in each atom. Then we can map the variables one-one in order of appearance. For example we can associate

$$x \leftrightarrow y$$

and

$$(*_1, \text{dummyop}) \leftrightarrow (*_2, +_2)$$

Then we can remove dummyop and rewrite

$$*_1 \leftrightarrow (*_2, +_2)$$

We can describe this process formally in two stages.

- (1) Make the two atoms type-compatible and of the same order by adding dummy variables whenever necessary.

Let $\text{atom}_1 = p[x_1; \dots x_n]$

$\text{atom}_2 = q[y_1; \dots y_m]$

$$\text{template } [\text{atom}_1] = \text{type}[p] [\text{type}[x_1] \dots \text{type}[x_n]]$$

$$\text{template } [\text{atom}_2] = \text{type}[q] [\text{type}[y_1] \dots \text{type}[y_m]]$$

Furthermore, suppose that the ordering of the types is the same in each template, even though the number of variables of each particular type need not be identical for corresponding "type blocks." Thus, in the preceding example, in both "group" and "ring" the type set precedes the type operator. Each template has one set variable, but a differing number of operator variables. Thus, we could partition the ordered set of variables in atom_1 and atom_2 by letting some x_i and x_{i+1} belong to the same partition if $\text{type}[x_i] = \text{type}[x_{i+1}]$. Now there are an equal number of partitions in both atom_1 and atom_2 . Returning to our example, we partition $\text{group}[x; *_1]$ into $[[x], [*_1]]$ and the $\text{ring}[y; *_2; +_2]$ into $[[y], [*_2; +_2]]$. (The brackets indicate that the order of elements is preserved.)

- (2) Map the partitioned subsets into each other, preserving their order within the partitions, and map elements into elements if the two subsets have an equal number of elements.

This completes our brief description of ATOMMATCH. From now on, we will consider ATOMMATCH as an elementary operation that will expand the developing analogy to include a (possibly) new predicate pair and (possibly) new pairs of variable associations. We need to know how to select pairs of atoms from the statements of T and T_A to be ATOMMATCHed.

We have two wffs representing T and T_A as arguments of INITIAL-MAP, and we want to find some way to slide ATOMMATCH over pairs of atoms selected from the wffs. First, note that the syntax of the wffs may be a helpful guide in selecting potential matches.

Suppose $T:A \Rightarrow p(x)$

$$T_A:B \Rightarrow q(y)$$

where A and B are any wffs.

We would presume that $p \leftrightarrow q$ (predicates)

$$x \leftrightarrow y$$
 (variables)

and $A \leftrightarrow B$ (sub-wffs),

where we expect that wffs A and B would be decomposed down to atoms for ATOMMATCH. If A and B had implication signs in them, we could decompose them similarly. There are many possibilities for the forms of T and T_A . We find that if T and T_A are closely analogous, then their syntactic forms are likely to be very similar. ZORBA, considers

* I will use a double-headed arrow " $\leftrightarrow$ " as in " $x \leftrightarrow y$ " to mean " x is associated with (analogous to) y ."

R. E. KLING

T and T_A to have the formats that can be represented by the generative grammar below

$$T \rightarrow A = A$$

$$A \rightarrow p[x_1 \dots x_n] \mid A[p[x_1 \dots x_n]]$$

INITIAL-MAP is designed to decompose the input wffs T and T_A into associated syntactic substructures until a subwff is either an atom $p[x_1 \dots x_n]$ or a conjunction of atoms

$$\bigwedge_{i=1}^k p_i[x_1 \dots x_n]$$

At this point it enters a hierarchy of selecting and matching routines (Figure 3) to decide which pairs of atoms shall be ATOMMATCHed. Naturally, if the subwffs are just atoms it calls ATOMMATCH directly. Otherwise, it enters a program hierarchy headed by a routine named SETMATCH, which selects appropriate atom pairs from the sets of conjuncted atoms in the subwffs.

In the following discussion, the number of atoms conjuncted in each set are assumed equal ($k = \ell$). SETMATCH can be described in terms of its subfunctions as follows:

Setmatch [set_1 ; set_2 ; ana]: *

- (1) Partition the atoms in set_1 and set_2 into subsets that have identical semantic templates (a "semantic partition"). Thus if set_1 is $group[x; *] \wedge abelian[y; *] \wedge intersection[z; x; y]$ the semantic partition will be $\{ \{ intersection[z; x; y] \} \{ group[x; *], abelian[y; *] \} \}$ since group and abelian are both of type struct[set; op].
- (2) Select the partitions of set_1 and set_2 that have but one element and call these $sing_1$ and $sing_2$, respectively.
- (3) The remaining partitions have more than one element; call them $mult_1$ and $mult_2$, respectively.
- (4) Match the atoms in $sing_1$ with those in $sing_2$ by executing $singlematch[sing_1; sing_2; ana]$.
- (5) Match the remaining atoms by executing $multimatch[mult_1; mult_2; ana]$.

SETMATCH, SINGLEMATCH, and MULTIMATCH are all heuristically designed one-pass matching strategies that make strong assumptions about the nature of the theorem statements T and T_A for an analogous theorem pair.

SETMATCH assumes that the atoms in set_1 and set_2 will map one-one and that the semantic partitions will map one-one. Suppose, we have a semantic partition thus:

$$\begin{aligned} \text{partition}_1 &= \{ \{atom_1 \ atom_2\} \ \{atom_3 \ atom_4\} \} \ \{atom_5\} \\ \text{partition}_2 &= \{ \{atom_6 \ atom_7\} \ \{atom_8 \ atom_9\} \} \ \{atom_{10}\} \end{aligned}$$

SETMATCH assumes that $\{atom_5\}$ and $\{atom_{10}\}$ will correspond, rather than $\{atom_5\}$ and, say $\{atom_6 \ atom_7\}$. It calls SINGLEMATCH to map the single-atom partitions onto the single-atom partitions.

In addition, it calls MULTIMATCH to map, in pairs, the partitions containing several atoms each.

MULTIMATCH assumes that the analogy will preserve semantic type sufficiently well so that atoms within a particular partition will correspond only to atoms in one other partition.

Thus, if $\{atom_1, atom_2\} \leftrightarrow \{atom_6, atom_7\}$

$$\begin{aligned} \text{then } atom_1 &\leftrightarrow atom_6 \text{ or } atom_7 \\ atom_2 &\leftrightarrow atom_6 \text{ or } atom_7 \end{aligned}$$

It forbids matches across partitions, such as

$$\begin{aligned} atom_1 &\leftrightarrow atom_6 \\ atom_2 &\leftrightarrow atom_8 \\ atom_3 &\leftrightarrow atom_7, \text{ etc.} \end{aligned}$$

SINGLEMATCH and MULTIMATCH also share a common default condition. If all but one of the elements of a set X are mapped with all but one of the elements of a set Y , then these two elements are associated by default without any further decision making. In SINGLEMATCH the sets X and Y are sets of atoms or partitions of atoms.

SINGLEMATCH [set_1 ; set_2 ; ana] may be easily described in terms of this default condition and a function called $tempsift[s_1; s_2; testfn; ana]$. TEMPSIFT applies $testfn[x; y]$ to the first element of s_1 and each successive element y of s_2 until it finds a $y' \in s_2$ such that $testfn[x; y'] = T$. It then executes

$$atommatch[x; y'; ana]$$

increments to the next element of x of s_1 , and seeks another $y'' \in s_2$, such that $testfn[x; y''] = T$, etc. Thus, for every $x \in s_1$, it finds the first $y \in s_2$ such that $testfn[x; y] = T$ and executes $atommatch[x; y] = T$. Typical $testfns$ check whether x and y have the same semantic template or are analogs of each other according to the developing analogy, ana. $Singlematch[set_1; set_2; ana]: =$

- (1) If set_1 and set_2 have but one element ("terminal default condition"), go to 8.

* When an analogy G is referenced within the description of an algorithm, it will be represented as a variable ana wherever that is more convenient.

R. E. KLING

- (2) Execute `tempsift[set1;set2;testfn1;ana]`, where `testfn1[x;y]` is true iff `x` and `y` have the same semantic template.
- (3) If `set1` and `set2` are empty, go to 9. If the terminal default condition is true, go to 8.
- (4) Execute `tempsift[set1;set2;testfn2;ana]`, where `testfn2[x;y]` is true iff the predicate letter in atom `y` is the analog of the predicate letter of that in atom `x` according to analogy `ana`.
- (5) If `set1` and `set2` are empty, go to 9. If terminal default conditions holds, go to 8.
- (6) Execute `tempsift[set1;set2;testfn3;ana]`, where `testfn3[x;y]` is true iff the type of the predicate appearing in atom `x` is the same as the semantic type of the predicate appearing in atom `y`.
- (7) If `set1` and `set2` are empty, go to 9. If the terminal default condition holds, go to 8. Otherwise print an error message and halt.
- (8) Apply `ATOMMATCH` to the remaining atoms of `set1` and `set2`.
- (9) STOP.

To illustrate the preceding algorithm with a simple example, let

```
set1 = {intersection[x;y;z], abeliangroup[x;*]}
set2 = {intersection[u;v;w],
        commutativering[u;*;+]}
```

Step 2 associates

```
intersection[x;y;z] ⇔ intersection[u;v;w]
```

and the terminal default condition associates

```
abeliangroup[x;*] ⇔ commutativering[u;*;+]
```

MULTIMATCH is a little more complex than SINGLEMATCH. First we need to decide which partitions are to be associated before associating atoms within partitions. Suppose we have two sets of partitions `set1` and `set2`. If both sets have but one partition each (a common case), then we expect these to be associated by default and declare them accordingly. Secondly, if in some partition of `set1` there is an atom with predicate `p` which is known to be analogous to predicate `q`, then the partition in `set2` that contains `q` should be associated with that which contains `p`. Remember that these partitions were constructed on the basis of semantic templates. Thus, while several atoms containing a predicate `p` may be in a particular partition, there will be only one partition that contains atoms with predicate `p`. Lastly, if in `set1` and `set2` there is but one partition that contains atoms whose

predicates have the same type, e.g., `STRUCTURE`, then we expect these partitions to be associated. Let `MULTIMATCH1` name the function that actually associates atoms within a partition according to analogy `ana`.

```
MULTIMATCH[set1;set2;ana]:=
```

- (1) If the terminal default condition for partitions holds, go to 7.
- (2) Let `pred[x]` = the predicate letter of atom `x`. For each partition `y`, sequence through each atom `x` ∈ `y`. If `pred[x]` is on analogy `ana` find the partition `z` ∈ `set2` such that the analog of `pred[x]` appears in `z`. Execute `MULTIMATCH1[y;z;ana]` for each such pair `y,z`.
- (3) If the terminal default condition holds, go to 7. If `set1` and `set2` are empty, go to 8.
- (4) For each partition `y` ∈ `set1`, select the first atom `x`. Find a partition `z` ∈ `set2` such that the type of predicates in `z` equals type `[x]`. If there is only one such `z` ∈ `set2`, execute `MULTIMATCH1[y;z;ana]`.
- (5) If the terminal default condition holds, go to 7. If `set1` and `set2` are empty, go to 8.
- (6) If `set1` or `set2` is still not exhausted, print an error message and halt.
- (7) Apply `MULTIMATCH1` to the remaining partitions in `set1` and `set2`.
- (8) STOP.

Each set of atoms in a partition has the same semantic template. This property defines a partition. Thus, at the level of abstraction provided by the templates, all of these atoms are alike and any differences need to be discriminated by other criteria. Let us consider an example to motivate the design of `MULTIMATCH1`. The theorem pair $T_3 - T_4$ can be written as:

$$\begin{aligned}
 T'_3 \quad & \forall (g,m,x,*_1) \text{ group}[g;*_1] \wedge \\
 & \text{propernormal}[m;g;*_1] \wedge \text{factorstructure}[x;g;m] \\
 & \wedge \text{simplegroup}[x;*_1] \Rightarrow \text{maximalgroup}[m;g;*_1] \\
 T'_4 \quad & \forall (r;n;y;*_2+*_2) \text{ ring}[r;*_2+*_2] \wedge \\
 & \text{properideal}[n;r;*_2+*_2] \wedge \text{factorstructure}[y;r;n] \\
 & \wedge \text{simpplering}[y;*_2+*_2] \Rightarrow \text{maximalring}[n;r;*_2+*_2]
 \end{aligned}$$

First ZORBA-I associates:

```
maximalgroup ** maximalring
m ** n
g ** r
*_1 ** (*_2, *_2)
```

R. E. KLING

when it decomposes $T'_3 - T'_4$ into subwffs distinguished by the syntax of the implication sign. Later an application of SINGLEMATCH adds:

```

propernormal ** properideal
factorstructure ** factorstructure
x ** y

```

MULTIMATCH is passed one partition from each wff. T'_3 contributes

```
{group[g;*_1], simplegroup[x;*_1]},
```

and T'_4 contributes

```
{ring[r;*_2;+_2], simplering[y;*_2;+_2]}.
```

If we apply the MULTIMATCH algorithm just described to each of these partitions, we find:

- Step 1. We do not satisfy the terminal default condition.
- Step 2. None of the predicates that appear in these partitions appear on the current analogy. We gather no new information here.
- Step 3. We still do not satisfy the terminal default condition.
- Step 4. We want to use MULTIMATCH1 to associate the atoms in these partitions.

Of these two partitions, the former pair have the template structure[set;operator] and the latter pair have structure[set;operator;operator]. Fortunately, our analogy has variable mapping information that is quite relevant here. We know that:

```

g ** r
x ** y

```

We can assume that if some variable appears in only one atom in a partition, the analogous atom is one that contains its analog variable, if it too appears in only one atom. For example, the variable "g" appears only in group[g;*_1], and its analog "r" appears only in ring[r;*_2;+_2]. So, we deduce:

```
group[g;*_1] ** ring[r;*_2;+_2]
```

A similar argument based upon

```
x ** y
```

leads us to deduce:

```
simplegroup[x;*_1] ** simplering[y;*_2;+_2]
```

although we could have also deduced this last association by our terminal default condition. Notice that " $*_1$ " is not a discriminating variable since it appears in both group[g;*_1] and simplegroup[x;*_1]. After each atom pair is associated, we apply ATOMMATCH to it to deduce more variable associations and update our analogy.

The preceding description of MULTIMATCH1 can be simplified and generalized by realizing that we are just using a specialized submap of the developing analogy to extend it further. This special submap is just that mapping of variables where each variable appears in only one atom of the partition. In the preceding example, the submap was just:

```

g ** r
x ** y

```

Multimatch1[partition₁;partition₂;ana]: =

- (1) Set ℓ_1 to a list of variables that appear in only one atom of partition₁.
- (2) Set ℓ_2 to similar list computed on partition₂.
- (3) Set $\text{anapr}_s = \{x' \leftrightarrow y' \mid x' \in \ell_1, y' \in \ell_2 \text{ and } y' \text{ is the analog of } x' \text{ by ana}\}$.
- (4) Execute $\text{tempsift}[\text{partition}_1; \text{partition}_2; \text{testfn}_4; \text{ana}]$, where $\text{testfn}_4[u;v]$ is true iff for some variable pair $x' \leftrightarrow y' \in \text{anapr}_s$ variable x' appears in atom u and variable y' appears in atom v .
- (5) STOP.

INITIAL-MAP has been completely described. At this point we have sufficient machinery to generate a mapping between the predicates and variables that appear in the statements of theorem pairs such as $T_1 - T_2$ and $T_3 - T_4$. Next we want to extend this mapping to include all the predicates that appeared in the proof of the proved theorem T and are likely to appear in the proof of the new theorem T_A . In addition, we would like to pick up a small set of axioms adequate for proving T_A . EXTENDER performs both functions.

A DETAILED DESCRIPTION OF EXTENDER

In the last section I described INITIAL-MAP in substantial detail. In comparison, EXTENDER is a far more complex and subtle system which I will explicate here less completely. I intend to accomplish several simple aims with this limited exposition:

- (1) Expose the reader to the motivation and rationale underlying the EXTENDER design.
- (2) Convey some appreciation for the flavor of some well-specified computational algorithms for creating an analogy.
- (3) Provide an intelligible, self-contained, introductory account of EXTENDER adequate for the general reader, and motivate the more sophisticated specialist to consult a more complete exposition. (6)

R. E. KLING

The rationale of EXTENDER depends upon a few simple related ideas. I will begin by explicating these, then develop MAPDESCR--the clause description mapping operation--and conclude with a discussion of two simple versions of EXTENDER.

In the last section I suggested that our complete analogy could be seen as the last map G_n in a series G_j of increasingly more complete analogies. Although we may be developing several such series in parallel, they all begin with the same G_1 --the analogy produced by INITIAL-MAP. Each G_j maps some subset of the predicates that appear in the proof of theorem T. Each distinct subset will, in general, lead to a different partition of AXSET into {ALL, SOME, NONE}. When we search for the analog of an axiom (clause), we will look for some clause that satisfies the analog of its description under the current analogy. Each clause has a unique description, $\text{descr}[c]$, which has been introduced in a previous section. We will denote the analog of $\text{descr}[c]$ by some analogy G_j as $G_j[\text{descr}[c]]$. $G_j[\text{descr}[c]]$ is equal to a copy of $\text{descr}[c]$ in which every predicate that appears in G_j is replaced by its analogous predicate. Predicates that are absent from G_j are left untouched. For example, suppose we have a trivial G_1 :

G_1 : abelian ∇ commutativering
 c_7 : $\neg \text{abelian}[x; *] \vee \text{group}[x; *]$
 d_7 : $\text{neg}[\text{abelian}], \text{pos}[\text{group}]. = \text{descr}[c_7]$
 $G_1[d_7] = \text{neg commutativering}, \text{pos}[\text{group}]$.

Suppose we are seeking to extend G_1 by finding the analog of c_7 . It is quite unlikely that we will find a clause that satisfies this description, ($G_1[d_7]$), since it would be derived from some (rare) theorem that relates a condition on commutative rings to a group structure. In any event, it would not be an analog of c_7 . If we sought all the clauses that satisfied $\text{neg}[\text{commutativering}]$, we would be sure to include c_8 and c_9 , which at least include c_8 , the clause we desire,

c_8 : $\neg \text{commutativering}[x; *, +] \vee \text{ring}[x; *, +]$
 c_9 : $\neg \text{commutativering}[x; *, +] \vee \text{commutative}[*, x]$

Thus, sometimes we want to search for clauses that satisfy descriptions with features, e.g., $\text{neg}[\text{commutativering}]$, that contain only predicates that appear on a particular analogy G_j . Now, what we are doing is a four-step process:

- (1) Make a description d for an axiom clause c , $\text{descr}[c]$.
- (2) Create an analog description $G_j[\text{descr}[c]]$ for the current analogy, G_j .

- (3) Delete from $G_j[\text{descr}[c]]$ any feature that contains a predicate that does not appear in G_j . Denote this restriction of $G_j[\text{descr}(c)]$ to G_j by $\overline{G_j[\text{descr}(c)]}$.
- (4) Search the data base for clauses that satisfy $\overline{G_j[\text{descr}(c)]}$.

In our example, $\overline{G_1[\text{descr}[c_7]]} = G_1[d_7] = \text{neg}[\text{commutativering}]$. $G_j[\text{descr}(c)]$ is a "restriction of the analog of the description of c to analogy G_j ." Since this phrase is quite cumbersome, we will simply call it a "restricted description" and implicitly understand its dependence on G_j .

At different times EXTENDER may seek clauses that satisfy a complete analogous description $G_j[\text{descr}]$ or just a restricted one $\overline{G_j[\text{descr}]}$. In summary, EXTENDER relies upon four key notions:

- (1) An ordered sequence of partial analogies G_j .
- (2) A partition of the axioms used in proof [T] (AXSET) into three disjoint sets: ALL, SOME, and NONE.
- (3) A search for clauses that satisfy the analogs of the description of the clauses in $\text{proof}[T]$.
- (4) A restriction of our descriptions relative to an analogy G_j by including only those features with predicates that appear in G_j .

INITIAL-MAP used an operation called ATOMMATCH in a rather clever way to extend its current analogy. Likewise, EXTENDER uses an operation called MAPDESCR for a similar purpose. Both operations use abstract descriptions in order to associate their data: ATOMMATCH uses the semantic template associated with a predicate, and MAPDESCR uses the description of the clauses it is associating. EXTENDER and INITIAL-MAP differ in that EXTENDER generates a new partial analogy each time it activates MAPDESCR (and the resultant mapping is new) while INITIAL-MAP uses ATOMMATCH to expand one growing analogy.

Each partial analogy G_j is derived from its antecedent G_{j-1} by adding

- (1) An association of one clause $ax_k \in \text{SOME}$ with one or more clauses from the data base.
- (2) An association of the predicates in those clauses.

A simple example will illustrate this amply. If G_1 is the initial analogy generated by INITIAL-MAP applied to the pair of theorems T_1 - T_2 , its predicate map is

R. E. KLING

abelian $\rightsquigarrow$ commutativering
 intersection $\rightsquigarrow$ intersection.

Suppose we know that $c_7 \rightsquigarrow c_8$. We would like to extend G_1 to G_2 by adding:

- (1) $c_7 \rightsquigarrow c_8$
- (2) abelian $\rightsquigarrow$ commutativering
 group $\rightsquigarrow$ ring.

To motivate the structure of MAPDESCR, let us design a version of it that would enable us to extend G_1 to G_2 in this example. MAPDESCR is charged with mapping neg[abelian], pos[group] (d_7) with neg[commutativering], pos[ring], when it knows that:

G_1 : abelian $\rightsquigarrow$ commutativering
 intersection $\rightsquigarrow$ intersection.

First, we can eliminate neg[abelian] from d_7 and neg[commutativering] from d_8 on the basis of G_1 , which associates "abelian" and "commutativering."

$G_1[\text{neg}[\text{abelian}]] = \text{neg}[\text{commutativering}]$.

Now we are simply left with associating pos[group] and pos[ring]. Since these are the only two elements left, have the same semantic type (STRUCTURE), and have the same feature (pos), we can map them by default and add

group $\rightsquigarrow$ ring

to G_2 .

Now, we can write a version of MAPDESCR which accepts as arguments two clause descriptions and an analogy G_j :

mapdescr[descr₁; descr₂; G_j] :=

- (1) $\forall x \ x \in \text{descr}_1$ s.t. $G_j[x] \in \text{descr}_2$,
 delete x from descr₁ and $G_j[x]$ from descr₂. Thus, we exclude all those features we know about from G_j .
- (2) $\forall x \ x \in \text{descr}_1$ and $x \in \text{descr}_2$, map the predicate that appears in x into itself and delete x from descr₁ and descr₂.
- (3) In the remnants of descr₁ and descr₂:
 - (a) If there are unique elements of descr₁ and descr₂ that have the same feature, e.g., pos, and semantically compatible predicates, associate those terms and delete them from the remnant descriptions. Here "semantic compatibility" means "same semantic type."

- (b) If more than one element of descr₁ and descr₂ have the same feature, e.g., pos, then discriminate within these elements on the basis of the semantic types of their predicates.
- (4) Return the resultant list of paired predicates.

Most often in my algebra data base a clause description consists of two, three, or four features. EXTENDER ensures that some of the predicates in any pair of clauses passed on to MAPDESCR are on G_j . Thus, by the time we reach step 3 of the MAPDESCR algorithm we often have descriptions of length one, which map trivially by default, or descriptions of length two with different features, e.g., pos and neg. Thus, step 3b, which requires disambiguation based upon predicate types, occurs rarely in this domain (abstract algebra).

When MAPDESCR returns a list of predicates pairs that result from mapping the description of a clause c_1 (descr₁, above) with the description of a clause c_2 (descr₂, above) according to analogy G_j , it creates a new analogy G_{j+1} . G_{j+1} is the same as G_j except that

- (1) Its predicate map is the union of the one returned by MAPDESCR and the one appearing on G_j .
- (2) Its clause mapping is the union of the one appearing on G_j and $c_1 \rightsquigarrow c_2$.

Thus, when EXTENDER is attempting to extend G_j , it creates a new analogy G_{j+1} , G_{j+2} , etc. for each clause pair it maps when those clauses were selected on the basis of information in G_j . Of course, there is a procedure to see whether the predicate associations of a new analogy have appeared in some previously generated analogy and thus prevent the creation of redundant analogies. In this case the two corresponding clauses are added to each existing analogy for which the predicate pairs returned by MAPDESCR are a subset of its clause map.

After I explicate one additional idea I can describe a simple version of EXTENDER. When EXTENDER is extending G_j it is searching the large data base for some clause that is the analog of an axiom $c_k \in \text{SOME}$. Now we could search for the set of clauses that satisfy $G_j[\text{descr}[c_k]]$, but we will run into the difficulty described earlier in this section. Thus, we search for clauses that satisfy $G_j[\text{descr}[c_k]]$. If G_j^p contains the correct analog for each predicate that appears on it, then the set of clauses C that satisfy $G_j[\text{descr}[c_k]]$ is guaranteed to contain the desired analog of c_k ("image" of c_k). We

R. E. KLING

will refer to C as the "candidate image set." Suppose that C has but one member, c' . Then we know that c is the analog (image) of c_k and should extend $G_j \rightarrow G_{j+1}$ by associating

$$c_k \mapsto c'$$

When the set of clauses that satisfies a restricted description contains only one, we are guaranteed that it is the image clause we seek if G_j^p does not contain any erroneous associations. Now, if C is empty, we have reason to suspect the correctness of G_j^p and we ought to stop developing this branch of the analogy search space. On the other hand, if C has more than one member, and G_j^p is correct, we know that our desired image is in C . If we have a clause c with description $\text{descr}[c]$ and some analogy G_j that contains only one of the predicates in c , then $G_j[\text{descr}[c]]$ will have but one feature and many clauses will satisfy it. If some later analogy G_k ($G_j^p \subset G_k^p$) includes another predicate from c in addition to the one on G_j , then $G_k[\text{descr}[c]]$ will have two features and will be satisfied by fewer clauses than $G_j[\text{descr}[c]]$. Thus, as sequence of analogies evolve, each clause will have decreasingly fewer candidate images that satisfy its restricted description.

To search for the clauses that satisfy the analog of a restricted (short) description, EXTENDER, invokes an operator $\text{shortdescr}[G_j]$. SHORTDESCR is dependent on G_j in three ways:

- (1) It searches for the analogs of clauses that appear on SOME (which is different for each G_j).
- (2) It generates descriptions that include only the predicates that appear explicitly in G_j .
- (3) It uses the predicate map G_j .

SHORTDESCR returns a (possibly empty) list of axioms (from SOME), each of which is paired with a set of clauses from the data base which satisfy the analog of its restricted description. Each axiom is guaranteed to have its analog under G_j in its associated "candidate image set." If we find no candidates at all, for any $ax_k \in \text{SOME}$, then we know that G_j contains some wrong predicate associations, and we ought to mark it as "infertile" and discontinue attempting to extend it. Of the images we find, we prefer those axiom-candidate associations with but one candidate image. If we apply MAPDESCR to each such pair, we can be sure that we have a consistent extension of G_j . Let us consider a primitive version of EXTENDER, EXTENDER1, which exploits these few ideas.

EXTEND1 [G_1 ; AXLIST] :=

- (1) Let analist = (G_1) , the set of active analogies.
- (2) If G_j is complete, STOP.
- (3) Partition AXLIST into {ALL, SOME, NONE} relative to G_j .
- (4) Set imlist to $\text{shortdescr}[G_j]$. If imlist = $\emptyset$, mark G_j as BARREN and go to 7.
- (5) Set unimages to the subset of imlist that has only one candidate analog for each axiom. If unimages = $\emptyset$, go to 7.
- (6) Apply MAPDESCR to each axiom and its analog that appears on unimages. If MAPDESCR adds a new analogy, add it to the end of analist.
- (7) If analist is empty, STOP. Otherwise, set G_j to the next element on analist. Go to 2.

The success of EXTEND1 is highly dependent upon the clauses in the data base. If there are few clauses then it is likely that some $ax_k \in \text{SOME}$ will have but one image under SHORTDESCR at each iteration and that EXTEND1 will be successful. As the data base increases in size with ever more clauses involving predicates that will appear in $\text{proof}[T_A]$, then it becomes more likely for SHORTDESCR to generate several images for every $ax_k \in \text{SOME}$ in some iteration. At this point it will fail to EXTEND G_j and miss the analogy altogether. To remedy this situation, we need a way for dealing with cases when SHORTDESCR returns several candidate images for each $ax_k \in \text{SOME}$. We need some way to select the clause from the candidate set that is most likely to be the analog we seek. When EXTENDER meets a situation of this sort, it orders all the images according to their likelihood of being analogous to the $ax_k \in \text{AXSET}$ with which they are paired. I will initiate the description of one such ordering relation by a simple example.

Consider, for example, the clause c_{10} and an analogy G_2 that includes

intersection $\leftrightarrow$ intersection

subgroup $\leftrightarrow$ subring

abeliangroup $\leftrightarrow$ commutativering

c_{10} : subgroup[x; y; *] $\vee$ $\neg$ group[x; *] $\vee$ $\neg$ group[y; *] $\vee$ $\neg$ subset[x; y]

d_{10} = neg[group], neg[subset], pos[subgroup]

$G_2[d_{10}]$ = pos[subring].

Suppose our data base contains two clauses c_{11} and c_{12} that satisfy $G_2[d_{10}]$:

c_{11} : subring[m; r; *; +] $\vee$ $\neg$ ideal [m; r; *; +]

d_{11} = neg[ideal], pos[subring]

c_{12} : subring[x; a; *; +] $\vee$ $\neg$ ring[a; *; +] $\vee$ $\neg$ ring[x; *; +] $\vee$ $\neg$ subset[x; a]

d_{12} = neg[ring], neg[subset], pos[subring].

R. E. KLING

We can compare c_{11} and c_{12} by comparing d_{11} and d_{12} with d_{10} (relative to G_2). We want a partial ordering of a set of descriptions relative to a target description and a particular analogy, e.g., a $\varphi_d[d_1; d_2; d; G_j]$, that orders description d_1 with respect to d_2 . A simple φ_d can be developed as follows:

$$\begin{aligned} \text{Let } d'_1 &= d_1 - \overline{G_j[d]} \\ d'_2 &= d_2 - \overline{G_j[d]} \\ d' &= d - \overline{G_j[d]} \end{aligned}$$

For d'_1 and d'_2 compute the number of features, e.g., pos, in common with d' . The description with the most features in common is closest to d .

In our example, we have

$$\begin{aligned} d'_{10} &= \text{neg}[\text{group}], \text{neg}[\text{subset}] \\ d'_{11} &= \text{neg}[\text{ideal}] \\ d'_{12} &= \text{neg}[\text{ring}], \text{neg}[\text{subset}]. \end{aligned}$$

Clearly d'_{12} is closer to d'_{10} than d'_{11} , so we select d'_{12} our closest description and c_{12} as the image of c_{10} under G_2 . After MAPDESCR maps $c_{10} \approx c_{12}$ it will add:

$$\begin{aligned} \text{group} &\approx \text{ring} \\ \text{subset} &\approx \text{subset} \end{aligned}$$

to G_2 to create an G_3 :

$$\begin{aligned} G_3: \quad \text{intersection} &\approx \text{intersection} \\ \text{subgroup} &\approx \text{subgroup} \\ \text{group} &\approx \text{ring} \\ \text{subset} &\approx \text{subset}. \end{aligned}$$

A more sophisticated φ_d can look at the semantic types of predicate that share common features if two descriptions are equivalent under the simple φ_d described above. EXTENDER uses an operator called MULTIMAP to select the best image (using φ_d) for a clause that has several candidates images with a restricted description under G_j . Exploiting this notion, we can write a more powerful EXTENDER called EXTEND2.

EXTEND2 [G_1 ; AXSET]: =

- (1) Let analyst = ($G_1 \dots G_j$), the list of active analogies. Start with analyst = (G_1).
- (2) If G_j is complete, STOP.
- (3) Partition AXSET into {ALL, SOME, NONE} relative to G_j .
- (4) Set imlist to shortdescr[G_j]. If imlist = $\emptyset$, mark G_j as "infertile" and go to 8.
- (5) Set unimages to the subset of imlist that has only one candidate analog for each axiom. If unimages = $\emptyset$, go to 7.

- (6) Apply MAPDESCR to each axiom and its analog that appears on unimages. If MAPDESCR adds a new analogy, add it to the end of analyst. Go to 8.
- (7) Apply MULTIMAP to imlist to select an optimal candidate image under φ_d for each axiom. Set unimages to this list of axioms paired with best candidates. Go to 6.
- (8) If analyst is empty, STOP. Otherwise, set G_j to the next element on analyst. Go to 2.

This version of EXTENDER is quite powerful and will handle a wide variety of theorem pairs. The reader who is interested in the behavior of EXTENDER in generating the sequency G_j is referred to a more detailed report (6) for case studies and further explication. The implemented versions of EXTENDER are far more complex than these simplified tutorial versions. They (1) allow backup, (2) have operations for combining a set of partial analogies into a "larger" analogy consistent with all of them, (3) have a sophisticated evaluation for deciding which particular axiom-candidate set to pass to MULTIMAP (in lieu of step 7 above), and (4) can often localize which predicate associations are contributing to an infertile analogy when one is generated. Table 2B contains a brief summary of ZORBA-1's behavior when it is applied to five T-TA pairs drawn from abstract algebra. The number of partial analogies generated includes G_1 generated by INITIAL-MAP.

Table 2A

THEOREMS REFERENCED IN TABLE 2B

- T1. The intersection of two abelian groups is an abelian group.
- T2. The intersection of two commutative rings is a commutative ring.
- T3. A factor group G/H is simple iff H is a maximal normal subgroup of G .
- T4. A quotient ring A/C is simple iff C is a maximal ideal in A .
- T5. The intersection of two normal groups is a normal group.
- T6. The intersections of two ideals is an ideal.
- T7. The homomorphic image of a subgroup is a subgroup.
- T8. The homomorphic image of a subring is a subring.
- T9. The homomorphic image of an abelian group is an abelian group.
- T10. The homomorphic image of a commutative ring is a commutative ring.

R. E. KLING

Table 2B

SUMMARY OF ZORBA-I PERFORMANCE

Theorem Pair	Number of Predicates in Theorem	Number of Predicates Mapped by I-MAP	Number of Axioms in Proof(T)	Number of Clauses Found by EXTENDER	Number of Partial Analogies Generated by ZORBA-I
T1 - T2	9	3	13	15	5
T3 - T4	12	5	13	17	7
T5 - T6	8	3	21	23	5
T7 - T8	5	4	6	7	2
T9 - T10	7	2	12	16	6

* Theorem statements appear in Table 2A on the preceding page.

NECESSARY CONDITIONS FOR AN ANALOGY

ZORBA-I has three necessary conditions for creating an analogy. The first, created by the form of ATOMMATCH, pertains to the form of the statements of T and T_A .

- (1) In the statements of T and T_A , atoms must map one-one from T to T_A .

Notice that we do not insist that predicates map one-one. Consider an INITIAL-MAP between

T1: The intersection of two abelian groups is an abelian group

and

T5: The intersection of an abelian group and a commutative ring is an abelian group.

$T1'$: $\text{abelian}[a;*_1] \wedge \text{abelian}[b;*_1] \wedge \text{intersection}[c;a;b] = \text{abelian}[c;*_1]$

$T5'$: $\text{abelian}[x;*_2] \wedge \text{cring}[y;*_2;+_2] \wedge \text{intersection}[z;x;y] = \text{abelian}[z;*_2]$

ATOMMATCH can map

$\text{abelian}[c;*_1] \rightsquigarrow \text{abelian}[z;*_2]$

and $\text{abelian}[b;*_1] \rightsquigarrow \text{cring}[y;*_2;+_2]$

at different times and handle many-one predicate maps. However, the EXTENDER would need to know (and it does not yet) how to handle this ambiguous information.

The second restriction is created by the extension of the analogy by finding image clauses that satisfy the incrementally improved analogy. To state this condition on the image clauses in a formal way, I need to introduce some simple terminology. Let us say that a clause c bridges a set of predicates P_1 to another set of predicates P_2 iff:

$$\begin{aligned} P_1 &\subset P_2 \\ P_1 \cup \text{preds}[C] &= P_2 \\ P_1 \cap \text{preds}[C] &\neq \emptyset \end{aligned}$$

and (redundantly)

$$\begin{aligned} P_2 \cap \text{preds}[C] &\neq \emptyset \\ P_2 &\neq P_2 \end{aligned}$$

Now consider two clauses, c_1 and c_2 . We will say that c_1 and c_2 bridge from P_1 to P_2 if $\exists P'$ and c_1 bridges from P_1 to P' and c_2 bridges from P' to P_2 . $P_1 \subset P' \subset P_2$. In general, we will say that an unordered set C of k clauses bridges from P_1 to P_2 iff $\exists P'_1, P'_2, \dots, P'_{k-1}$

such that:

- (1) $\exists c_1 \in C$ and c_1 bridges from P_1 to P'_1
- (2) $\forall_j x_j = 2 \dots k-1$ and $c_j \in C$
 c_j bridges from P'_j to P'_{j+1}
- (3) $\exists c_k \in C$ and c_k bridges from P'_{k-1} to P_2 .

Now let:

$\text{preds}[T]$ = predicates used in proof of T.

$\text{Pr}[T]$ = predicates used in statement of T.

G = analogy from T to T_A .

$\text{descr}[c]$ = description of clause c .

$G[\text{descr}[c]]$ = analog description of the description of c under G .

AXSET = axioms used in proof of T.

R. E. KLING

(2) A necessary condition for the EXTENDER to work is that:

- (a) $\exists c \subseteq \text{AXSET}$ and c bridges $\text{Pr}(T)$ to $\text{preds}[T]$.
- (b) and if $G[c] = \{c'', c''\}$ satisfies $G[\text{descr}[c']]$ for some $c' \in c$ $G[c]$ bridges from $\text{Pr}[T_A]$ to $\text{preds}[T_A]$.

More verbally, some subset of the axioms in the proof of T that bridge R the domain of INITIAL-MAP to $\text{preds}[T_A]$ has a set of image clauses under G that bridge the images of INITIAL-MAP to $\text{preds}[T_A]$. Thus, the proofs need not be isomorphic, merely that some subset of the axioms have a nearly isomorphic set of image axioms, similarly restricted to the bridging condition.

This bridging condition may seem rather non-intuitive from the vantage point of choosing a data base, but it should be clear that EXTENDER imposes this condition.

To develop analogies in domains that are described by predicate calculus with constants would require wholly different analysis algorithms. Consider a robot that is instructed to go from SRI to (1) an office on its floor, (2) Stanford University, (3) San Francisco, (4) New York City, (5) Chicago. These five problems could be stated to QA3 as

- $T_{10}. \quad \text{Es}_f \text{ at } [\text{robot}; \text{office}_5; s_f]$
- $T_{11}. \quad \text{Es}_f \text{ at } [\text{robot}; \text{Stanford}; s_f]$
- $T_{12}. \quad \text{Es}_f \text{ at } [\text{robot}; \text{San Francisco}; s_f]$
- $T_{13}. \quad \text{Es}_f \text{ at } [\text{robot}; \text{NYC}; s_f]$
- $T_{14}. \quad \text{Es}_f \text{ at } [\text{robot}; \text{Chicago}; s_f]$

By trivial syntactic matching we could associate office5 with Chicago, Stanford with San Francisco, etc. The robot's actions to get from SRI to Stanford or San Francisco, New York City, or Chicago are pairwise similar. But the INITIAL-MAP or extender would have to know the "semantics" of these (geographic) constants (with respect to SRI) and the robot's actions to assess which problems are adequately analogical and which action rules should be extrapolated to the unsolved problem.

RELATIONSHIP BETWEEN ZORBA-I AND QA3

In the preceding section, I have discussed the organization and use of ZORBA-I independently of QA3. In this section, I merely want to note how change in QA3 can affect the way in which the analogical information output by ZORBA-I can be used.

The present version of ZORBA-I outputs a set of clauses that it proposes as a restricted data base for proving T_A . If every clause in $\text{proof}[T]$ has at least one image clause, then simply modifying the QA3 data base is magnificently helpful. However, if the analogy is weak and we have only a partial set of images, what can we do? If every predicate used in the $\text{proof}[T]$ has an image, we could restrict our data base to just those clauses containing the image predicates. Could we do better? And what do we do with a partial analogy in which some clauses and some predicates have images, but not all of either? At this point we meet limitations imposed by the design of QA3. All contemporary theorem provers, including QA3, use a fairly homogeneous data base. QA3 does give preference to short clauses, since it is built around the unit-preference strategy. But it has no way of focusing primary attention upon a select subset of axioms A^* , and attending to the remaining axioms in $D - A^*$ only when the search is not progressing well. One can contrive various devices, such as making the clauses in A^* "pseudo-units" that would be attended to early. Or, with torch and sword, one could restructure QA3 around a "graded memory." (7) Basically we have to face the fact that our contemporary strategies for theorem proving are designed to be as optimal as possible in the absence of a priori problem-dependent information. And these optimal strategies are difficult to reform to wisely exploit a priori hints and guides that are problem dependent. This is not to say that various kinds of a priori information cannot be added. Rather, it is a separate and sizable research task to decide how to do it. I presume, but do not know, that these comments extrapolate to other problem-solving procedures, and a system that is organized around a priori hints, heretofore user supplied, may look very different than one which is designed to do its best on its own. QA3 was chosen because it was available and saved years of work developing a (new) suitable theorem prover. However, further research in AR may well benefit from relating to a more flexible theorem-proving system.

WHAT'S NEW?

What does ZORBA add to our understanding of AR? What does ZORBA leave unanswered? Pre-ZORBA, most researchers believed that analogies would relate to plans and (possibly to probably) include some sort of semantic information. ZORBA adds the following insights to our understanding of AR:

- (1) Some fairly interesting AR can be handled by modifying the environment in which a problem solver operates rather than forcing the use of a sequential planning language.

R. E. KLING

- (2) Each problem solver/theorem prover will use different a priori information and consequently will require different analogy-generation programs.
- (3) A good analogy generator will output some information helpful to speeding up a problem search as a byproduct of a successfully generated analogy.
- (4) Part of the problem of AR is to specify precisely how the derived analogical information is to be used by the problem solver.
- (5) An effective, nontrivial analogy generator can be adequately built that uses a simple theory and primitive semantic selection rules.
- (6) Although analogies are nonformal and are semantically oriented, nontrivial analogies can be handled by a special system wrapped around a highly formal theorem prover.

In contrast, ZORBA neglects:

- (1) Methods for handling those analogies that absolutely require a planning level generalization and sequential information.
- (2) Very weak analogies.
- (3) What to do with many rules of inference.
- (4) How to describe the "structure of an analogy."

ZORBA makes a substantial contribution to our pale understanding of AR, and in the process helps articulate additional questions that reveal our vast ignorance of analogical ways of knowing.

REFERENCES

1. N. J. Nilsson, Problem Solving Methods in Artificial Intelligence (McGraw-Hill, to be published 1971).
2. G. W. Ernst and A. Newell, "Some Issues of Representation in a General Problem Solver," AFIPS Conference Proceedings, Vol. 30 (1967), pp. 583-600.
3. R. E. Fikes, "REF-ARF: A System for Solving Problems Stated as Procedures," Artificial Intelligence, Vol. 1, pp. 27-120 (1970).
4. R. E. Kling, "An Information Processing Approach to Reasoning by Analogy," Artificial Intelligence Group TN10, Stanford Research Institute, Menlo Park, California (June 1969).

5. C. Green, "Theorem Proving by Resolution as a Basis for Question Answering Systems," in Machine Intelligence, Vol. 4, D. Michie and B. Meltzer, eds. (Edinburgh Univ. Press, Edinburgh, Scotland, 1969).
6. R. E. Kling, "Reasoning by Analogy with Applications to Heuristic Problem Solving: A Case Study," Stanford University Ph.D. Thesis forthcoming.
7. R. E. Kling, "Design Implications of Theorem Proving Strategies," AI Group Technical Note 44, Stanford Research Institute, Menlo Park, California (1970).