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.

(2) It provides a concrete information-processing framework within which and against which one can pose and answer questions germain to AR.

(3) Since it is implemented (in LISP), it is available as a research tool as well as a gedanken tool.

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."

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 , 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 a p p.
(3) Construct G (s') = s'$_A$.
(4) Execute S$_A$ to get S , 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.

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 6 be modified and a variant S. 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 α may be interactive with how much successful problem-solving has proceeded so far. (A complete α 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 system that has solved some problem P and is posed with a new (analogous) P to solve. Clearly, t must operate on some large data base sufficient o solve both P and P . (See Figure 1.) In addition to the subbase for solving P and P there

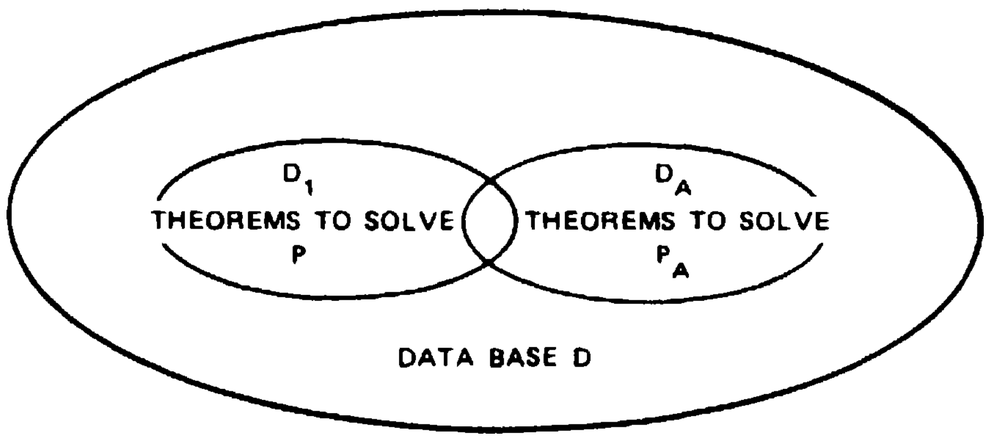

FIGURE 1 VENN DIAGRAM OF THEOREMS IN DATA BASE

re likely to be even more theorems in the set $D - (D_1 \cup D_A)$. Now, given P it is impossible to nfer a minimal D_1. In practice, a user may select some D_2 s.t. $D_1 \subseteq D_2 \subset D$ which the problem solver will access to solve P. If one studies he 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 and its descendants alone. In fact, while a problem solver might solve P given an adequate and small D , 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 size [D_2] ≤ size [D_1] << size [D]. For example, a typical theorem in algebra provable by QA3 —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 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 size[D] = 250 would now be provable.

A second kind of information that would be useful to help solve P would be a set of lemmas (or subgoals) $L_1, \ldots L_n$ whose analogs $G(L_1), \ldots G(L_n)$ 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.

that forces backup in case of failure. Suppose we somehow get $G(L_1)$, . . . $G(L_j)$ A typical planner would order the $G(L_1)$, e.g., $G(L_1)$, $G(L_2)$ ... 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 $G(L_1)$ 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 $G(L_1)$ 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 successG(L_j)eed 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 an 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 cummutative ring.

In fact, under some conditions, the axioms used to solve (L_i) may be deleted from D_2 so that size $[D_2]$ IS decreased, and (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

A clause is an element in the conjunctive normal form of a skolemized wff in the predicate calculus. For example: — person [x]
V father [g(x); x] is the clause associated with: Vx person [x] → 3y father [y;x] (every person has a father).

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 tor contemporary theorem provers. (See Table 2, in a later section, lor a listing oi additional theorem pairs.) T_1 has a 35-step prool and T_3 has a 50-step proof in a decent axiomatization. A good theorem prover (QA3) generates about 200 mlerences in searching for either prool when its data base is miaimized to the 13 axioms required lor 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 lew ol the predicates used in any prool. Thus, T can be stated using only flNTERSECTION; ABELIAN], but a proof requires {GROUP; IN; TIMES, SUBSET; SUBGROUP; COMMUTATIVE] in addition. Thus, while the first set is known to map into {INTERSECTION, COMMUTATIVERING} , the second set can map into anything.

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

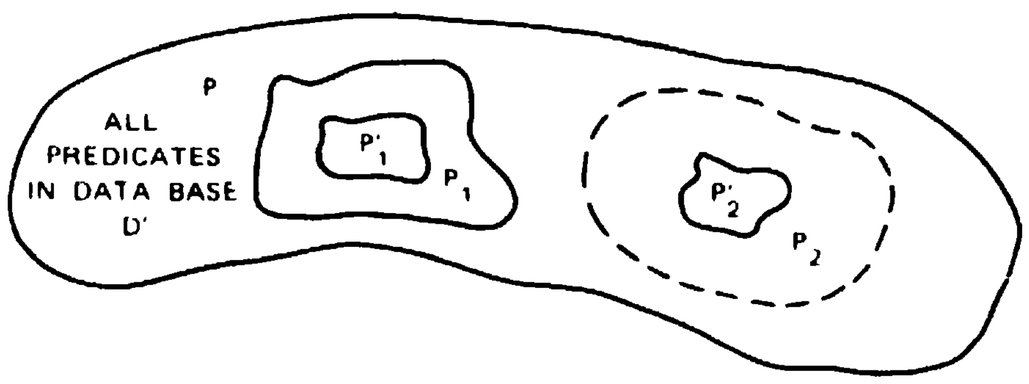

FIGURE 2 VENN DIAGRAMS OF RELATIONS IN STATEMENTS T, $T_{A'}$ AND D'

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) = P .

Clearly, a wise method would be to find some G , a restriction ol G to P_1 such that α' (P,) -

P2 Then incrementally extend G to $\alpha_{1'}$ α_2, each on larger domains until some G (P) - 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.. In tact, if we get no new clauses from an extended α'_j, that may be reason to believe that α_1' is faulty. The next sections will describe the generation algorithm in a little more detail.

ZORBA'S REPRESENTATION OE 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 sot 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 ol the proved theorem is associated with one or more axioms that will be used to prove the new, analogous theorem. More explicitly, a ZORBA-1 analogy G is a relation αP X A^c x α^v , where:

(1) α^p is a one-one map between the predicates used in the proof oi the proved theorem T and the predicates used in the proof of the unproved theorem T_A .

(2) $^{\alpha c}$ is a one-many mapping between clauses Each clause used in the prool oi T is associated with one or more clauses from the data base D that ZORBA-I expects to use in proving T..

(3) G 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., α^v and/or G^p and/or α^c. Usually I will drop the superscript and simply refer to "the analogy G." Thus "the analog of an axiom ax under analogy G" should be understood to mean $^{\alpha c}$fax 1, and will often be mentioned simply as "the analog oi ax .

In the previous section I refer to a sequence of analogies α_1 , ... αk. ZORBA-I usually does not develop α^c in one step. Rather, it

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" 1 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 αk. is an extension of a partial analogy αj if some of αj, e.g., α_j^p , α^c G $_f$ *is* a submap restriction of the corresponding submap u.^ to a smaller domain. Intuitively, when we add a new predicate or clause association to αj so as to create α_k, we say that α_j, has been extended to G. . 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[TI that is an ordered set of clauses c_K s.t. Vk c^ is either
  (a) A clause in ~i T
  (b) An axiom
  (c) Derived by resolution from two clauses
    c and c j < k and 1 < 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) V s.t. p and - p appear in c, impcond[p] c descr[c].
(2) V s.t. p appears in c, pos[p] C descr[c].
(3) V s.t. — p appears in c, neg[p] c descr[c].

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

e.g., V(x*) abelian [x; *1 =» group [x; *] ,

is expressed by the clause

c :—i abelian fx; *] V group Tx; *1 ,

which is described by

neg [abelian], pos [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.,

V ($_x$ y *$_1$ *$_2$ω)
hom [ω>;x;y] ^{A} group Ix; * 1
=> group Ty; *.J

is expressed by the clause

c :-i hom [cp;x;y| V -i group Ix; * 1 V group [y; *]

and is described by

negl horn], impcond[group]

Two different clauses may have the same description Let :

c : -i.ntersecuonfxjy;*] V subset fx;y]
c intersection[x;y;z] V subset[x;z]

Then :

descrlc] = descr[c] = neg[intersection], poslsubset|

Clause descriptions are used to characterize the axioms whose analogs we seek. ZORBA-1 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-1 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 c descrlc]. Thus, several clauses may satisfy the same description.

The "analog of a description" is defined later.

Let:

c intersectionfx; y; 7 J V - grouply; *J
V —i groupIz; *] V group[x; *]

c_6: -n subgroup[x; y; *] V ~i subsetfx;yI

Then, the following statements are true:

(1) {c_2,c_5 } satisfy impcondfgroupI
(2) {c_1,c_2,c_5 } satisiy posIgroup]
(3) satisiies neg[abelian], pos[group]
(4) {c_3,c_4,C_6 } satisiy posIsubset]
(5) C_6 satisfies negIsubgroupj, posIsubset
(6) No clause of these six satisfies pos [intersect ion]

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 lor the prooi oi T. 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 A_1^P which is used by EXTENDER to start its sc-

quence of α_1^p and A which terminate in a complete u. INITIAL-MAP starts without a priori information about the analogy it is asked to help create. Both α^p and α are empty when it begins. It uses the system of the wffs that express T and T. as well as the restrictions imposed by the semantic categories to generate OP and α_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 prooIlTj] and the statements of T - T. can contain five of the 12 predicates used in proof[T„]. In brief, it provides a starting point from which EXTENDER can develop a complete 0.

The INITIAL-MAP uses a rule of inference called ATOMMATCH[atomjatom ;0], which extends analogy by adding the predicates and mapped variables of atom^ and aton^ to analogy OL. Thus, ATOMMATCH now limits ZORBA-I to analogies where atoms in the statements of T and T. map one-one. INITIAL-MAP is a sophisticated search program

Atoms, not predicates.

that sweeps ATOMMATCH over likely pairs oi atoms, one of which is from the statement oI T, the other 1rom the statement oI T.. 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 O[?.

EXTENDER accepts a partial analogy generated by INITIAL-MAP and uses it as the first term in a sequence of successive analogies 0 The axioms used in pr oof \ T] a re f ew in c ompa r 1 s on to the sr/e oi the large data base and comprise the "domain" for a complete 0 . For each axiom used in proofIT1, we want to find a clause from the data base that is analogous to it. The axioms used in prooIIT] are called AXSET and are used by EXTENDER in a special way. Each partial analogy α^{op} is used to partition AXSET into three disjoint subsets called ALLIOJ, S0ME[0j, and NONE[0].

If all the predicates in an axiom ax, C AXSET
k
are in OP, then ax. is in ALL[α]; if some of its

predicates are in $\alpha_{1,}^{p}$ then ax_k is in S0ME[0]; and if none of its predicates are in α_J,^{p} then ax_k is in N0NE[α_1]. For brevity, these sets will be called ALL, SOME, and NONE, and their dependence on α_1 will be implicit. This partition is trivial to compute, and initially, none or a iew ax_k are in ALL, and most ax belong to SOME and NONE. We
n k
want to develop a sequence oI analogies α_1,] = 1, 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 oI some of its predicates from 0 . These provide sufficient inlormation 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 α_1^P. We can extend α_J to α and thus the analogs of axioms on SOME provide a bridge between the known and the unknown, between the current 0 and a descendent G ,j . When EXTENDER has satisfactorily terminated, ALL = AXSET, SOME = NONE = 0. 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 O,[ax_kI - ax_k is found that can be used in the proof of T . Moreover,
A

each new association of clauses should help us extend $C_j \rightarrow C_{j+1}$ by providing information about predicates not contained in C_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 tor a large class of interesting analogies (such as the group-ring analogy in abstract algebra) and simplify the description ot the various procedures. Analogies that require weaker assumptions and more complex procedures arc 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 now proof. ZORBA-I can be easily expressed in terms ot these two functions as follows:

zorba |newwff;oldwff;AXSET J: -

(1) Set analogies to the list of analogies generated by initial map[nowwlt;oldwffl.
(2) Apply extcnderTanalogy; 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

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

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 variablee 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^ 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[atorn]/, atom ; 6] as a rule of inference to add the predicate/variable information to analogy Ci. As its name hints, ATOMMATCH matches the predicates and variables of its atomic arguments and adds the resultant mapping to the developing analogy (k).

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

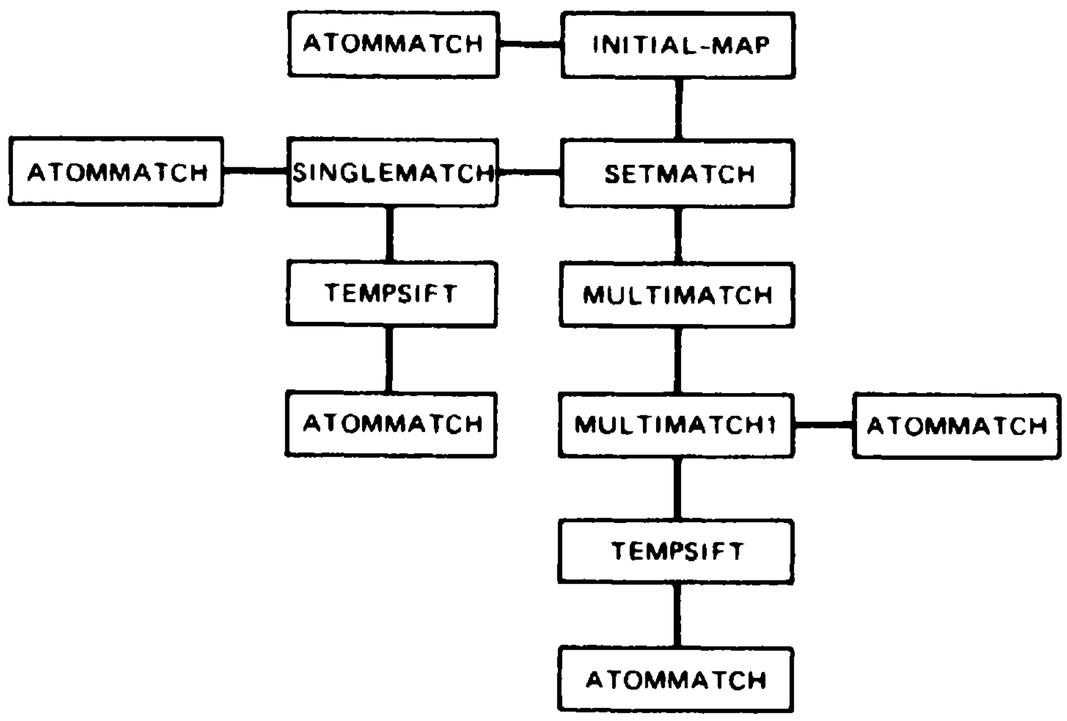

FIGURE 3 HIERARCHY OF MATCHING ROUTINES CALLED BY INITIAL-MAP

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

$atom_1$ = intersection[x_1; x_2, x_3]

$atom_2$ = intersection[y_1; y_2; y_3]

Clearly, we want

intersection ** intersection

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

So, if $atom_1 = p[x_1; \ldots x_n]$

and $atom_2 = q[y_1; \ldots y_m]$

and p = q (thus, n = m)

we will set p ** q

and x_i ** y_i, i = 1,2,...,n .

So far ATOMMATCH is quite trivial. Suppose, however, p ≠ q or n ≠ m.

For example, let $atom_1$ = group[x; $*_1$]

and $atom_2$ = 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 + . 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 group[x,$*_1$; 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 ** y

and

($*_1$,dummyop) **($*_2$, $+_2$) .

Then we can remove dummyop and rewrite

$*_1$ **($*_2$, $+_2$) .

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 $atom_1 = p[x_1; \ldots x_n]$

$atom_2 = q[y_1; \ldots y_m]$

template [$atom_1$] = type[p] [type[x_1] ... type [x_n]]

template [$atom_2$] = type[q] [type[y_1] ... 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 variablos. Thus, we could partition the ordered set of variables in $atom_1$ and $atorn_2$ by letting some x, and x ,, belong to the same partition if type[x = type[x .,]• Now there are an equal number of partitions in both atom, and atonU). Returning to our example, we partition group[x; *j] into ("Cx], [*]_]] and the ringfy; $*_2$; +>,] into I[y], r$*_2$;+$_2$]]. (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 to be ATOMMATCHed.

We have two wifs representing T and T 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 => p(x)

T_A:B =* q(y)

where A and D arc any wffs.

We would presume that p ** q (predicates)

x ** y (variables)

and A ^{m} B (sub-wffs) ?

where we expect that wits 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 "⇔" as in "x ⇔ y" to mean "x is associated with (analogous to) y."

T and T. to have the formats that can be ropre sented by the generative grammar below

T → A ⇒ A

A -> pTx ...x] A p[x ....x$_n$]

INITIAL-MAP is designed to decompose the in put wffs T and T_A into associated syntactic sub structures until a subwff is either an atom p[x_1 ... x_n] or a conjunction of atoms

$$\bigwedge_{i=1}^{k} p_i[x_1 \ldots x_n] \quad .$$

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 hier- archy 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 = *SL)*. SETMATCH can be described in terms of its subfunctions as follows:

Setmatch [set ; 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 grouptx; *] Δ abelian [y; *] Δ intersection! z; x; y] the se- mantic partition will be {{intersectionfz;x;y]j{grouplx; *], 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 sin$_1$ and sing$_2$, respectively.
(3) The remaining partitions have more than one element; call them mult$_1$ and mult,,, respectively.
(4) Match the atoms in sing, with those in sing$_2$ by executing singlematch[smg ; sing$_2$;ana].
(5) Match the remaining atoms by executing multimatch[mult ; mult ; ana|.

SETMATCH, SNGLEMATCH, and MULTIMATCH are all heunstically designed one-pass matching strate- gies that make strong assumptions about the na- ture ot the theorem statements T and T^ for an analogous theorem pair.

When an analogy α. is referenced within the de- scription of an algorithm, it will be represented as a variable ana wherever that is more convenient.

SETMATCH assumes that the atoms in set-, and set$_2$ will map one-one and that the semantic parti- tions will map one-one. Suppose, we have a se- mantic partition thus:

partition$_1$ = {{atom$_1$ atom$_2$} {atom$_3$ atom$_4$]} {atom$_5$}
partition$_2$- {{atom$_6$ atom$_7$} {atom$_8$ atom$_9$ J J {atom$_{10}$j

SETMATCH assumes that {atom$_c$} and {atora$_{ir}$J will correspond, rather than iatom.J and, say [atom$_{fi}$ atom$_7$}. It calls SNGLEMATCH 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 pre- serve semantic type sufficiently well so that atoms within a particular partition will corre- spond only to atoms in one other partition.

Thus, if {atom$_1$ atom$_2$] <-> [atom$_6$ atom$_7$}
then atom$_1$ ** atom$_6$ or atom$_7$
atom$_2$ ** atom$_6$ or atom$_7$

It forbids matches across partitions, such as

atom$_1$ ** atom$_6$
atom$_2$ ** atom$_8$
atom$_3$ ** atom$_7$, etc.

SNGLEMATCH and MULTIMATCH also share a common default condition. If all but one of the elements ot a set X are mapped with all but one of the ele- ments of a set Y, then those two elements are as- sociated by dciault without any further decision making. In SNGLEMATCH the sets X and Y arc sets ot atoms or partitions of atoms.

SNGLEMATCH [set$_1$; set$_2$; anal may be easily de- scribed in terms of this default condition and a function called tempsift[s,;s ;testfn;ana]. TEMPSIFT applies testfnjx;y] to the first element of S-, and each successive element y of s_2 until it finds ay € s_9 such that testfn|x;y] - T.

It then executes

atommatch[x; y ;ana] ,

increments to the next element of x of si, and seeks another y e s_2, such that testfrux ;y | - T, etc. Thus, for every x e s , it finds the first y € s_2 such that testfn[x;y] - T and exe- cutes 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 de- veloping analogy, ana. Singlematch[set,; sct$_2$; ana]: =

(1) If set} and set,, have but one eloment ("terminal default condition"), go to 8.

(2) Execute tempsift[set,; set_2; testfn..; ana], where testfn^txjyj is true iff x and y have the same semantic template.
(3) If set_1 and set_2, are empty, go to 9. If the terminal default condition is true, go to 8.
(4) Execute tempsift[set-p set_2; tcstfn$_2$; ana], where testf^txjy] 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 set_1 and set_2 are empty, go to 9. If terminal default conditions holds, go to 8.
(6) Execute tempsift[set ;set_2;testfn$_3$ ana], where testfn$_3$[x;yj 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 set_1 and set_2 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 set and set_2.
(9) STOP.

To illustrate the preceding algorithm with a simple example, let

set, = fintersection[x;y;z], abeliangroup[x;*1J

set_2 = {intersection[u;v;w], commutativeringlu; *; I] J

Step 2 associates

intersectionlx;y;zj <-> intersection[u;v;w]

and the terminal default condition associates

abeliangroup[x; *] *=* commutativeringlu; *; -/ J

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 set, and $set_{2>}$ 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 set_1 there is an atom with predicate p which is known to be analogous to predicate q, then the partition in set_2 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 set_1 and set_2 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(set ;set,;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 atomx e y. If pred[x] is on analogy ana 1 md the partition z c set_2 such that the analog of prcd[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 set. and set_2 are empty, go to 8.
(4) For each partition y e set-., select the first atom x. Find a partition z € set_0 such that the type of predicates in z equals type [x]. If there is only one such z C set_2, execute MULTIMATCH1[y;z; ana].
(5) If the terminal default condition holds, go to 7. If set, and set_9 are empty, go to 8.
(6) Ii set, or set is still not exhausted, print an error message and halt.
(7) Apply MULTIMATCH1 to the remaining partitions in set and set .
(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 T3 - T can be written as:

T_3' ∀ (g,m,x,$*_1$) group[g;*1] ∧
propernormal[m;g;$*_1$] ∧ factorstructure[x;g;m]
∧ simplegroup[x;$*_1$] ⇒ maximalgroup[m;g;$*_1$]

T_4' ∀(r;n;y;$*_2$,$+_2$) ring [r;$*_2$;$+_2$] ∧
properideal[n;r;$*_2$;$+_2$] ∧ factorstructure[y;r;n]
∧ simplering[y;$*_2$;$+_2$] ⇒ maximalring[n;r;$*_2$;$+_2$]

First ZORBA-I associates:

maximalgroup ↔ maximalring
m ↔ n
g ↔ r
$*_1$ ↔ ($*_2$,$+_2$)

when it decomposes T_3' - T_4' into subwffs distinguished by the syntax of the implication sign. Later an application of SINGLEMATCH adds:

propernormal ** proper ideal

iactorstructure ** factorstructure

x ↔ y

MULTIMATCH is passed one partition from each wff T_3 contributes

{groupLg;*], simplegroup[x; *]},

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 MULTIMATCH 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 structureTset;operator;opera tor]. Fortunately, our analogy has variable mapping information that is quite relevant here. We know that:

g ~ r

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! gj*], and its analog "r" appears only in nng[r; * ;$+_2$]. So, we deduce:

group!g; *] ~ ring[r; * ;-J]

A similar argument based upon

x ↔ y

leads us to deduce:

simplegroupfx; *] simplenng[y; * ';+ $_2$]

although we could have also deduced this last association by our terminal default condition. Notice that "* " is not a discriminating variable since it appears in both groupfg;$*_1$,] and simplegroupLx; *]. 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

MultimatchICpartition ;partition, ;ana]: -

(1) Set $\underline{\ell_1}$ to a list of variables that appear in only one atom of partition .

(2) Set $\underline{i_2}$ to similar list computed on $\underline{\text{partition}_2}$.

(3) Set anaprs = {x′ ↔ y′|x′ ∈ $\underline{\ell_1}$, y′ ∈ $\underline{\ell_2}$ and y′ is the analog of x′ by ana}.

(4) Execute tempsift[partition_1;partition_2; testfn_4; ana], where testfn_4[u;v] is true iff for some variable pair x′ ⇔ y′ ∈ anaprs 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 . 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.. In addition, we would like to pick up a small set of axioms adequate ior proving T . EXTENDER performs both functions

A DETAILED DESCRIPTION OF EXTENDER

In the last section 1 described INITIAL-MAP in substantial detail. In comparison, EXTENDER is a far more complex and subtle system which I will explicate here less completely. 1 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)

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 α_n in a series α_J of increasingly more complete analogies. Although we may be developing several such series in parallel, they all begin with the same α_1—the analogy produced by INITIAL-MAP. Each G 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, SCME, NONE}. When we search for the analog oi 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, descrTc], which has been introduced in a previous section. We will denote the analog of descrfc] by some analogy G_J as G_J[descr[c]]. $G_J$rdescrTc]] is equal to a copy of descr[c] in which every predicate that appears in α_1 is replaced by its analogous predicate. Predicates that are absent from G are left untouched. For example, suppose we have a trivial Gj:

G_1: abelian ⌜ commutativering

c : ~ʃi abeliunlx; *] V group [r, *]

d_7: negfabelianl,pos[group]. - descrlc$_7$]

G_1[c$_7$] -neg commutat lvering] ,pos[group] .

Suppose we are seeking to extend G^ by finding the analog of C7. It is quite unlikely that we will find a clause that satisfies this description, (G^fd_]), 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~. It we sought all the clauses that satisfied neg[commutativering], we would be sure to include c» and c_q, which at least include c„, the clause we desire,

c_o: -y commutativering[x; *; f] ᵛ ringlx; *; 1J

c : ~i commutat ivcringl x; *;( ] V commutativeT*; x]

Thus, somctimet» we want to search for clauses that satlsly descriptions with leatures, e.g., neglcommutativering], that contain only prcdicates that appear on a particular analogy Ci Now, what we arc doing is a four-step process:

(1) Make a description d for an axiom clause c, descrlc].
(2) Create an analog description C fdescr[c]l for the current analogy, G .
(3) Delete from G,[descr[c]] any feature that contains a predicate that does not appear in 6 . Denote this restriction of Qu[descr(c)] to Ci by 6 [descr(c)].
(4) Search the data base for clauses that satisfy CL [descr(c)].

In our example, 6,[descrfc]] = G,[d_7] = neg[commutetivenng]. Ci [descr(c)] is a "restriction of the analog of the description of c to analogy 6P." Since this phrase is quite cumbersome, we will simply call it a restricted description" and implicitly understand its dependence on j

At different times EXTENDER may seek clauses that satisfy a complete analogous description G [descr] or /just a restricted one G [descr]. In summary, EXTENDER relies upon four key notions:

(1) An ordered sequence of partial analogies ᵃ.i-
(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 ot the description of the clauses in proof[T].
(4) A restriction of our descriptions relative to an analogy Gj by including only those ieatures 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 oi 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 lrom its antecedent G ^ by adding

(1) An association of one clause ax € SOME with one or more clauses lrom the data base.
(2) An association oi the predicates in those clauses.

A simple example will illustrate this amply. 11 G. is the initial analogy generated by INITIAL-MAP applied to the pair of theorems Tj-T^, its predicate map is

abelian commutat Ivering

intersection<=>intersection.

Suppose we know that c_ •* Cy. We would like to extend G to G by adding:

(1) $c_?$ •• c_8
(2) abelian •• conunutat Ivering
group •• ring.

To motivate the structure of MAPDESCR, let us design a version of it that would enable us to extend 0. to G in this example. MAPDESCH is charged with mapping neg[abelianl, posjgroupl (d_) with negtcommutatlveringl, posfnngj, when it knows that:

G : abelian •• commuta t Ivering

intersect ion •• intersection.

First, we can eliminate negfabelian] from d and neg[commutativeringJ from d on the basis of G which associates "abelian" and "commutatlvering."

G|[neglabelian]] - negtcommutatlvering]1.

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

group ring

to G_2.

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

mapdescr[$descr_1$;$descr_2$;G_J]:=

(1) ∀x x ∈ $descr_1$ s.t. G_J[x] ∈ $descr_2$, delete x from $descr_1$ and G_J[x] from $descr_2$. Thus, we exclude all those features we know about from G_J.

(2) ∀x x ∈ $descr_1$ and x ∈ $descr_2$, map the predicate that appears in x into itself and delete x from $descr_1$ and $descr_2$.

(3) In the remnants of $descr_1$ and $descr_2$:

(a) If there are unique elements of $descr_1$ and $descr_2$ 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 descr and descr 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 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 oi 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 (descr , above) with the description of a clause c (descr , above) according to analogy G,, it creates a new analogy G G is the same as G except that

(1) Its predicate map is the union of the one returned by MAPDESCR and the one appearing on G .
(2) Its clause mapping is the union of the one appearing on Gj and c_1 <=> c_2.

Thus, when EXTENDER is attempting to extend G , it creates a new analogy G G , etc. for each clause pair it maps when those clauses were selected on the basis of information in G . 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. it is searching the large data base for some clause that is the analog of an axiom c, C SOME. Now we could search for the set of clauses that satisfy Gj[descrCc^]], but we will run into the difficulty described earlier in this section. Thus, we search for clauses that satisfy G\,[descr[c_k]]. If *G,* contains the correct analog for each predicate that appears on it, then the set of clauses C that satisfy Ga[descr[cjJ] is guaranteed to contain the desired analog of c ("image" of c). We

R E KLNC

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

When the set of clauses that satisfies a restricted description contains only one, we are guaranteed that it is the image clause we seek i1 OP does not contain any erroneous associations Now, if C is empty, we have reason to suspect the correctness of &P and we ought to s top developing this branch of the analogy search space. On the other hand, if C has more than one member, and GP is correct, we know that our desired image is in C. If we have a clause c with description descrlcl and some ana logy G that contains only one of the predicates in c, then Ci Tdescr[c]] wi 11 have but one f ea ture and many clauses will satisfy it. If some later analogy ti^ (CLP — *&Y)* includes another predicate from c in addition to the one on G then G.fdescrfc]] will have two fea t ures and will be satisfied by fewer clauses than G . [descr[c]1. Thus, as sequence ol analogies evolve, each clause wi11 have decreasingly fewer Candida t e images t ha t sa t isfy its rest ricted description.

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

(1) It searches 1 or the ana logs of clauses t hat appear on SOME (which is dit/erent for each G).
(2) It generates descript)ons that inc lude only the predicat es that appear explicitly in G_1.
(3) It uses the predicate map G_1^r.

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 in its associated "candidate image set." If we find no candidates at all, for any ax. € SOME, then we know that G contains some wrong predicate associations, and we ought to mark it as [1] infertile" and discont mue attempting to extend it. Of the images we find, we prefer those axiom-candidate associations with but one candidate linage. If we apply MAPDESCR to each such pair, we can be sure that we have a consistent extension of G,. Let us consider a primitive version of EXTENDER, EXTENDER|, which exploits these few ideas.

EXTENDI [G ;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 shortdescr[G_j]. If imlist = Ø, 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 - Ø, 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 EXTENDI is highly dependent upon the clauses in the data base. If there are few clauses then it is likely that some ax. € SOME will have but one image under SHORTDESCR at each iteration and t hat EXTENDI will be successlul As the data base increases in size with ever more clauses involving predicates that will appear in proof n\], then It becomes more likely for SHORTDESCR to generate several imaees for every ax, € SOME in some iteration. At this point it will fail to EXTEND & and miss the analogy alt ogether- To remedy this situation, we need a way tor dealing with cases when SHORTDESCR returns several candidate images for each ax. € 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. C AXSET with which they are pai red . I will mi 11ate the description of one such ordering relation by a simple example.

Consider, 1 or example, the clause c and an ana logy G_9 that includes

intersection ↔ intersection
subgroup ↔ subring
abeliangroup ↔ commutativering

c_{10}: subgroup[x; y; *] ∨ ¬ group[x; *] ∨ ¬ group[y; *] ∨ ¬ subset[x; y]

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

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

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

c_{11}: subring[m; r; *; +] ∨ ¬ ideal [m; r; *; +]

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

c_{12}: subring[x; a; *; +] ∨ ¬ ring[a; *; +] ∨ ¬ ring[x; *; +] ∨ ¬ subset[x; a]

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

We can compare c_{11} and c_{12} by comparing d_{11} and d_{12} with d_{10} (relative t G_2). We want a partial ordering of a set of descriptions relative to a target description and a particular analogy, e.g., a φ .[d,;d_9;d;6] that orders description d
o 1 +• J 1
with respect to d_2- A simple φ^ can be developed as follows:

Let $d_1' = d_1 - \overline{G_J[d]}$

$d_2' = d_2 - \overline{G_J[d]}$

$d' = d - \overline{G_J[d]}$

For d, and d compute the number of features, eg•> P°s, in common with d . The description with the most features in common is closest to d.

In our example, we have

d = neg[group], neg[subset]

d' - neglidealj

d = negTring], negfsubset].

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 C_2. After MAPDESCK maps c_{10} ** C_{12} it will add:

group ° ring
subset ° subset

to G_2 to create an G_3:

G_3: intersection ⇔ intersection
subgroup ⇔ subgroup
group ⇔ ring
subset ⇔ subset.

A more sophisticated '4^ can look at the semantic-types of predicate that share common leatures 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 . Exploiting this notion, we can write a more powerful EXTENDER called EXTEND2.

EXTEND2 [Q ;AXSET]: -

(1) $(G_1 \ldots G_j)$ active analogies. Start with analist

(2) If Q is complete, STOP.
(3) Partition AXSET into [ALL, SOME, NONE) relative to Ct
(4) Set Imlist to shortdescrTCi]. If Imlist - 0, mark CL 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 0, 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 anallst. Go to 8.
(7) Apply MULTIMAP to imlist to select an optimal candidate image under cp . for each axiom. Set unimages to this list of axioms paired with best candidates. Co to 6.
(8) If analist is empty, STOP. Otherwise, set d to the next element on analist. 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 6 is referred to a more detailed report (6) for case studies and further explication. The implemented versions oi 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 ZORHA-I's behavior when it is applied to live T-TA pairs drawn from abstract algebra. The number of partial analogies generated includes (J. generated by INITIAL-MAP.

Table 2A

THEOREMS REFERENCED IN TABLE 2B

TI. The intersection of two abellan groups is an abelian group.
T2. The intersection of two commutative rings is a cummutative ring.
T3. A lactor group G/H is simple iII H is a maximal normal subgroup ol G.
14. A quotient ring A/C is simple IfI C is a maximal ideal in A.
'It>. The intersection of two normal groups is a normal group.
T6. The intersections oi two ideals is an idea 1.
T7. The homomorphic image oi a subgroup is a subgroup.
T8. The homomorphic image oi 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 commulntive ring.

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-1 has three necessary conditions for creating an analogy. The first, created by the form of ATOMMATCH, pertains to the <u>form</u> 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': abelian *[AA;* $*_1$] Δ abelian [b; $*_1$] Δ intersection[c;a;b] = abelian[c; $*_1$]

T5': abelian[x; $*_2$] Δ cring[y;$*_2$;$+_2$] Δ intersection^; x;y] - abelian[z; ^]

ATOMMATCH can map

abelian[c; *] ** abelian[z; *]

and abelian[b; *] ** cring[y; * ; 4-]

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 <u>bridges</u> a set of predicates P_1 to another set of predicates P_2 iff:

$$P_1 \subset P_2$$

$$P_1 \cup preds[C] = P_2$$

$$P_1 \cap preds[C] \neq \emptyset$$

and (redundantly)

$$P_2 \cap preds[C] \neq \emptyset$$

$$P_2 \neq P_2 \quad .$$

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 ... 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 ... 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:

preds[T] - predicates used in proof of T.

Pr[T] = predicates used in statement of T

k - analogy from T to T_A.

descr[cI = description of clause c.

CL[descr[c]] - analog description of the description of c under Ci.

AXSET - axioms used in proof of T.

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

(a) $\exists$ c $\subseteq$ AXSET and c bridges Pr(T) to preds[T].

(b) and if G[c] = {c″, c″ satisfies G[descr[c′]] for some c′ $\in$ c G[c] bridges from $Pr[T_A]$ to $preds[T_A]$.

More verbally, some subset of the axioms in the proof of T that bridge R the domain of INITIAL-MAP to preds[T_A] has a set of image clauses under G that bridge the images of INITIAL-MAP to predsfT]. 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-mtuitivc 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}. $\exists s_f$ at [robot; office$_5$; s_f]

T_{11}. $\exists s_f$ at [robot; Stanford; s_f]

T_{12}. $\exists s_f$ at [robot; San Francisco; s_f]

T_{13}. $\exists s_f$ at [robot; NYC; s_f]

T_{14}. $\exists s_f$ at [robot; 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 pairwiee 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-1 can be used.

The present version of ZORBA-1 outputs a set of clauses that it proposes as a restricted data base for proving T.. If every clause in 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 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.

(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. KIIng, "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).