STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-147

COMPUTER SCIENCE DEPARTMENT
REPORT NO. CS-216

REASONING BY ANALOGY WITH APPLICATIONS TO HEURISTIC PROBLEM SOLVING: A CASE STUDY

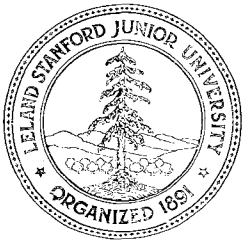
BY

ROBERT ELLIOT KLING

AUGUST 1971

COMPUTER SCIENCE DEPARTMENT

STANFORD UNIVERSITY



213

This procedure (ZORBA) is studied in detail for a resolution theorem proving system. A set of algorithms (ZORBA-I) which automatically generates an analogy between a new unproved theorem, T_A, and a previously proved theorem, T, is described in detail. ZORBA-I is implemented in LISP on a PDP-10.

A large set of axioms, D, that is sufficient to prove a variety of non-trival theorems is provided. The user supplies (1) T_A; (2) T; and (3) a proof of T, proof [T]. ZORBA-I outputs a set of axioms D' ($D' \subset D$) which it proposes for proving T_A. The axioms in D-D' are deleted and a proof of T_A is attempted.

ZORBA-I creates an analogy, α, which consists of two submaps:

(1) a one-one map between the predicates that appear in proof[T] and D';

(2) a one-many map between the axioms that are used in proof [T], called AXSET, and those in D'.

A complete analogy α includes all the predicates and axioms that appear in proof[T]. A partial analogy contains only some of these. One partial analogy α_k is an extension of another partial analogy α_j, if one of the submaps of α_j is a restriction of the corresponding submap of α_k. ZORBA-I operates by developing a sequence of partial-analogies that terminate in a complete α.

A program called INITAL-MAP creates the first partial analogy, α_1, by associating the predicates that appear in the statements of T and T_A. A second program (EXTENDER) uses a small set of operators which transform a partial-analogy into an extended partial-analogy. It uses syntactic description of the clauses in AXSET to instigate searches through D to find analogs for each clause. Each new clause association may create a new partial-analogy. The sequence of partial analogies finally terminates in a complete analogy which includes D' as a submap.

ZORBA-I is examined in terms of its empirical performance on paris of analogous theorems drawn from abstact algebra. A D is chosen with 250 clauses and D' is found for each of several theorems that requre only 5-20 axioms to prove them. Analytical studies are included which show that ZORBA-I can be useful to aid automatic theorem proving in many pragmatic cases while it may be a detriment in certain specially contrived cases.

The limitation of ZORBA-I's representation of an analogy are discussed along with proposals for future research.

BIBLIOGRAPHIC DATA SHEET	1. Report No. STAN-71-CS-216	2. AIM-147	3. Recipient's Accession No.
4. Title and Subtitle REASONING BY ANALOGY WITH APPLICATIONS TO HEURISTIC PROBLEM SOLVING: A CASE STUDY			5. Report Date August, 1971 6.
7. Author(s) Robert Elliot Kling			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Computer Science Department Stanford University Stanford, California 94305			10. Project/Task/Work Unit No. 11. Contract/Grant No. SD-183
12. Sponsoring Organization Name and Address Advanced Research Projects Agency			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts An information-processing approach to reasoning by analogy is developed that promises to increase the efficiency of heuristic deductive problem-solving systems. When a deductive problem-solving system accesses a large set of axioms more than sufficient to solve a particular problem, it will often create many irrelevent deductions that are derived from the unnecessary axioms. These irrelevent deductions may be quite numerous and saturate the memory of the problem solver before it solves the problem. At the current state of the art, the most complex problems solved by automatic procedures require less than two dozen axioms to solve. A data base twice this size is sufficient to render any but the simplest problem unsolvable. In general, there is no decision procedure which can be used ro restrict a data base to a set of necessary axioms. Here, any analogy with some previously solved problem and a new unsolved problem is used to restrict the data base to a small set of appropriate axioms.

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited	UNCLASSIFIED	191
	20. Security Class (This Page) UNCLASSIFIED	22. Price

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-147

August 1971

COMPUTER SCIENCE DEPARTMENT REPORT
NO. CS 216

REASONING BY ANALOGY WITH APPLICATIONS TO HEURISTIC PROBLEM SOLVING: A CASE STUDY

by

Robert Elliot Kling

ABSTRACT: An information-processing approach to reasoning by analogy is developed that promises to increase the efficiency of heuristic deductive problem-solving systems. When a deductive problem-solving system accesses a large set of axioms more than sufficient to solve a particular problem, it will often create many irrelevent deductions that are derived from the unnecessary axioms. These irrelevent deductions may be quite numerous and saturate the memory of the problem solver before it solves the problem. At the current state of the art, the most complex problems solved by automatic procedures require less than two dozen axioms to

*This research was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense under Contract SD-183.

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Reproduced in the USA. Available from the Clearing house for Federal Scientific and Technical Information, Springfield, Virginia 22151. Price: Full size copy $3.00; microfiche copy $ .95.

solve. A data base twice this size is sufficient to render any but the simplest problem unsolvable. In general, there is no decision procedure which can be used to restrict a data base to a set of necessary axioms. Here, an analogy with some previously solved problem and a new unsolved problem is used to restrict the data base to a small set of appropriate axioms.

This procedure (ZORBA) is studied in detail for a resolution theorem proving system. A set of algorithms (ZORBA-I) which automatically generates an analogy between a new unproved theorem, T_A, and a previously proved theorem, T, is described in detail. ZORBA-I is implemented in LISP on a PDP-10.

A large set of axioms, D, that is sufficient to prove a variety of non-trivial theorems is provided. The user supplies (1) T_A; (2) T; and (3) a proof of T, proof [T]. ZORBA-I outputs a set of axioms D' (D' $\subset$ D) which it proposes for proving T_A. The axioms in D-D' are deleted and a proof of T_A is attempted.

ZORBA-I creates an analogy, α, which consists of two submaps:

(1) a one-one map between the predicates that appear in proof[T] and D';

(2) a one-many map between the axioms that are used in proof [T], called AXSET, and those in D'.

A complete analogy α includes all the predicates and axioms that appear in proof[T]. A partial analogy contains only some of these. One partial analogy α_k is an extension of another partial analogy α_j, if one of the submaps of α_j is a restriction of the corresponding submap of α_k. ZORBA-I

operates by developing a sequence of partial-analogies that terminate in a complete α.

A program called INITIAL-MAP creates the first partial analogy, α_1, by associating the predicates that appear in the statements of T and T_A. A second program (EXTENDER) uses a small set of operators which transform a partial-analogy into an extended partial-analogy. It uses syntactic description of the clauses in AXSET to instigate searches through D to find analogs for each clause. Each new clause association may create a new partial-analogy. The sequence of partial analogies finally terminates in a complete analogy which includes D' as a submap.

ZORBA-I is examined in terms of its empirical performance on paris of analogous theorems drawn from abstract algebra. A D is chosen with 250 clauses and D' is found for each of several theorems that require only 5-20 axioms to prove them. Analytical studies are included which show that ZORBA-I can be useful to aid automatic theorem proving in many pragmatic cases while it may be a detriment in certain specially contrived cases.

The limitation of ZORBA-I's representation of an analogy are discussed along with proposals for future research.

FOREWORD

This thesis is the first comprehensive report of a five-year project that studied the use of analogies to aid deductive problem solving. Some of the ideas presented here will appear in the professional literature. Portions of Chapter III are to be presented[1] at the 1971 IFIP Conference held at Ljubljana, Yugoslavia. Chapters III, IV, and V are to be presented[2] at the International Joint Conference on Artificial Intelligence, to be held in London, September 1971.

This research has been supported by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS 12-2221, and Rome Air Development Center under Contract AF 30(602)-4147.

ACKNOWLEDGMENTS

In 1966, Professor Edward Feigenbaum suggested a study of reasoning by analogy as a fruitful area for a quarter research project. His fertile suggestion gave rise to an elementary paradigm for exploiting analogies to aid deductive problem solving. Two years and two paradigms later, I developed the more elegant and successful approach which is presented in this thesis. During the five years I needed to complete this work, I was supported by the Artificial Intelligence Laboratory of Stanford Research Institute. I appreciate the encouragement offered me by Dr. Charles Rosen who supported this work through periods of promise, through times of doubt and through episodes of hopelessness. I also thank Dr. Bertram Raphael who maintained my support after he succeeded Dr. Rosen as head of the Artificial Intelligence Laboratory. At various times I have had valuable conversations with Professor Thomas Cover of Stanford and Drs. Robert Yates and Peter Hart of S.R.I. regarding the use of analogies in deductive reasoning.

Professor Feigenbaum significantly aided my effort to find representations and a language to express the fruits of this research.

CONTENTS

LIST OF TABLES

is quite new, for I have developed an operationally specific model for a kind of reasoning by analogy that has barely been studied in the past.

Some writers have demonstrated the usefulness of analogies to aid concept acquisition as a helpful adjunct to problem solving. Wertheimer's[3*] studies with school children (concept-acquisition) and Polya's[4,5] extensive examples of heuristic aids to problem solving (concept-formation) are two cases in point. In addition, a few experimental studies[6] verify the usefulness of relevant experience. However, none of these workers specify in any detail the cognitive processes that are invoked to create, appreciate, and exploit analogical information. Some artificial intelligence (AI) researchers have created problem-solving[7] and theorem-proving[8] and game-playing programs[9] that generate fewer irrelevant inferences or play a better game (of bridge) based on the experiences they have had in the past. However, each of the programs is designed to slowly improve from problem to problem and "learns" to perform well only after exposure to a large number of problems or games. While they develop a kind of sophistication that is general for a particular domain of discourse — e.g., geometry, logic, bridge — they are unable to extrapolate the quite powerful problem-dependent information that we associate with learning by analogy.

Thus we face a difficult situation: We want to study an important cognitive process at an operational level of detail which (a) has no adequate model in the problem-solving literature, and (b) is unprecedented in existing computer-based problem-solving systems.[†]

---

*References are listed at the end of this thesis.

[†]The one AI program that exhibits a variety of analogical reasoning in solving problems that appear on the Miller Analogies Test cannot use the analogies it generates to assist some deductive problem-solving system and contributes little to our discussion. It is described in more detail in the next section of this chapter.

The approach that I will explicate in the following chapters is devoted to designing and implementing a new artificial system that is sufficiently complete to generate and exploit analogies between pairs of fairly complex problems. It is able to substantially improve the performance of a deductive problem-solving program that is associated with it. In addition, its qualitative behavior resembles many features of human problem solving[10,11] including set, productive confusion, developing relevant abstractions, evaluation of promising leads, and the creation of partial solutions. The mechanisms that it includes may well be imcorporated in some later simulation of human analogical reasoning. This approach parallels a previous important linkage between AI research and the simulation of cognitive processes. In 1964 Danield Bobrow[12] reported his development of a program that solved algebra word problems of the sort studied by high school students. He had created a program that was sufficiently complex to solve many problems of this class. After he reported his work, Simon and Paige[13] analyzed the problem-solving protocols of high school and college students asked to solve similar problems. They found many of the mechanisms that were used by people to be represented in Bobrow's program. When we are researching a new area, a research strategy that precedes a validated simulation model with a model that is merely sufficient to perform the appropriate behavior seems necessary. In order to create a sufficiency model, we first need to find the set of operations necessary to produce our desired behavior. While referring to computer simulation of perceptual processes, Gyr[14] emphasizes this order and writes:

> "It should also be pointed out that the above problems require, first of all, research with the computer itself in order to establish, for example, what internal organization is required for the generation of a precept or capacity by the computer. Following this, the computer behavior must be compared with the behavior of living organisms."

The paradigm I will describe for reasoning by analogy can be appreciated both as a novel advancement of contemporary AI technology and as a fertile addition to the sparse psychological literature concerned with reasoning by analogy as a cognitive process. For the latter, it suggests a set of necessary operations that can be included in a simulation model.

In the next chapter, I will begin to limit the kinds of problem solving and analogies that we will study. Several varieties of analogy will be distinguished from the point of view of the kind of information processing that is necessary to recognize and exploit them. One class of analogies is selected for further study. A paradigm (ZORBA*) for utilizing this class of analogies is developed. In addition, a particular instance of this paradigm (ZORBA-I) is studied in detail for a particular kind of problem-solving system, a resolution-logic theorem prover, which in turn is applied to a particular domain of discourse — abstract algebra. Thus, at each stage our study will become increasingly specific. Consequently, many recurrent terms will need to be redefined periodically. For example, here I am content to allow the reader to use my preliminary definition of an analogy as a sort of similarity. In the next chapter, several varieties of analogy will be distinguished. Later still, within the context of ZORBA-I, an analogy will be represented as a particular set of one-one, one-many, and many-many mappings.

The next two sections place ZORBA in the context of contemporary AI and problem-solving research. This chapter concludes with a brief outline of the dissertation.

B. ZORBA in the Context of AI Research

Although ZORBA is an unprecedented extension of AI into the mechanical generation and exploitation of analogies to aid heuristic

---

*ZORBA is an acronym for (ZO) Reasoning By Analogy. Zorba was a passionate, intuitive Greek, and many of our contemporaries consider analogy an intuitive process outside the realm of reason.

problem solving, it draws many ideas from a long research tradition. These include:

(1) ZORBA-I is associated with a particular heuristic problem solver (resolution theorem prover) and necessarily relates to many of its particular features — e.g., axiomatic data base, single rule of inference, etc.

(2) ZORBA can easily be described by using many concepts that are recurrent and basic in the heuristic problem-solving literature. These include ideas such as a search space, legal-move generator, candidate-move ordering function, and matching routines. ZORBA operates with a search space in which each node is a special kind of mapping ("partial analogy") and has routines for generating successor nodes ("descendant analogies") that contain more information than their ancestry. When a node has several descendants, an ordering function is invoked to select the descendant most likely to be a valid extension of it. Matching routines are invoked to create the set of possible extensions, in selecting the most plausible member of this set, and in generating the actual extension. These processes will be described in substantial detail in Chapter IV.

(3) Specific AI programs have dealt with elementary forms of reasoning by analogy and learning applied to heuristic problem solvers. For completeness I want to describe this work here.

One program in the AI tradition stands out for its potential relevance. In 1964, Tom Evans[15] reported developing a system that successfully solved problems from the Miller Analogies Test, a widely used intelligence test. A testee is asked to select one of five given figures that satisfy a given analogical relationship. For example, which of (a)...(e) is to Diagram C as

Diagram A is to Diagram B in Figure 1 below? Evans' program was highly successful in solving many problems in this class and was one of the most complex programs of its day. However, much of its complexity was devoted to the pattern-recognition aspects of its activity -- e.g., separating  into  and  rather than  and  . The algorithms he developed for actually generating and testing his analogies are not described. In fact, he admits attempting "all possible combinations" of associations until he finds an appropriate analogy. Fortunately, he is dealing with only two or three (geometric) objects and the relations between them, and he has to consider, at most, 10 to 20 possible mappings. In contrast, some of the analogies we will treat in this thesis allow over 10^{11} possible mappings (Chapter VII) from which we must select one analogy! When Evans finally generates his analogy by his unspecified process, he stops. He doesn't exploit his analogy to aid some other problem-solving process. (Later in this introduction, we will discuss some experiments by Dreistatdt who showed that people can use some simple visual analogies to aid deductive problem solving.) Despite these limitations, Evans does contribute two key ideas which are exploited in ZORBA: (1) An analogy is viewed as a special kind of mapping, and (2) an analogy between a picture P_1 and a picture P_2 can be derived by matching a description P_1 with a description of P_2 and associating the corresponding objects and relations.

ZORBA is concerned with the axioms and rules of inference used to solving a deductive problem rather than the relations that describe a two-dimensional diagram. Thus, the description (of axioms) it uses* are quite different from Evans' picture description language. In addition, the number of possible mappings it could generate if it tried "all possible combinations" of the relation it wants to associate are prohibitively large (Chapter VII). Thus, we need explicit heuristics for restricting the set of mappings to those that are most plausible, and for selecting the best among these. Unfortunately, he leaves us in the dark, and we must invent our processes anew.

*See Chapter III.

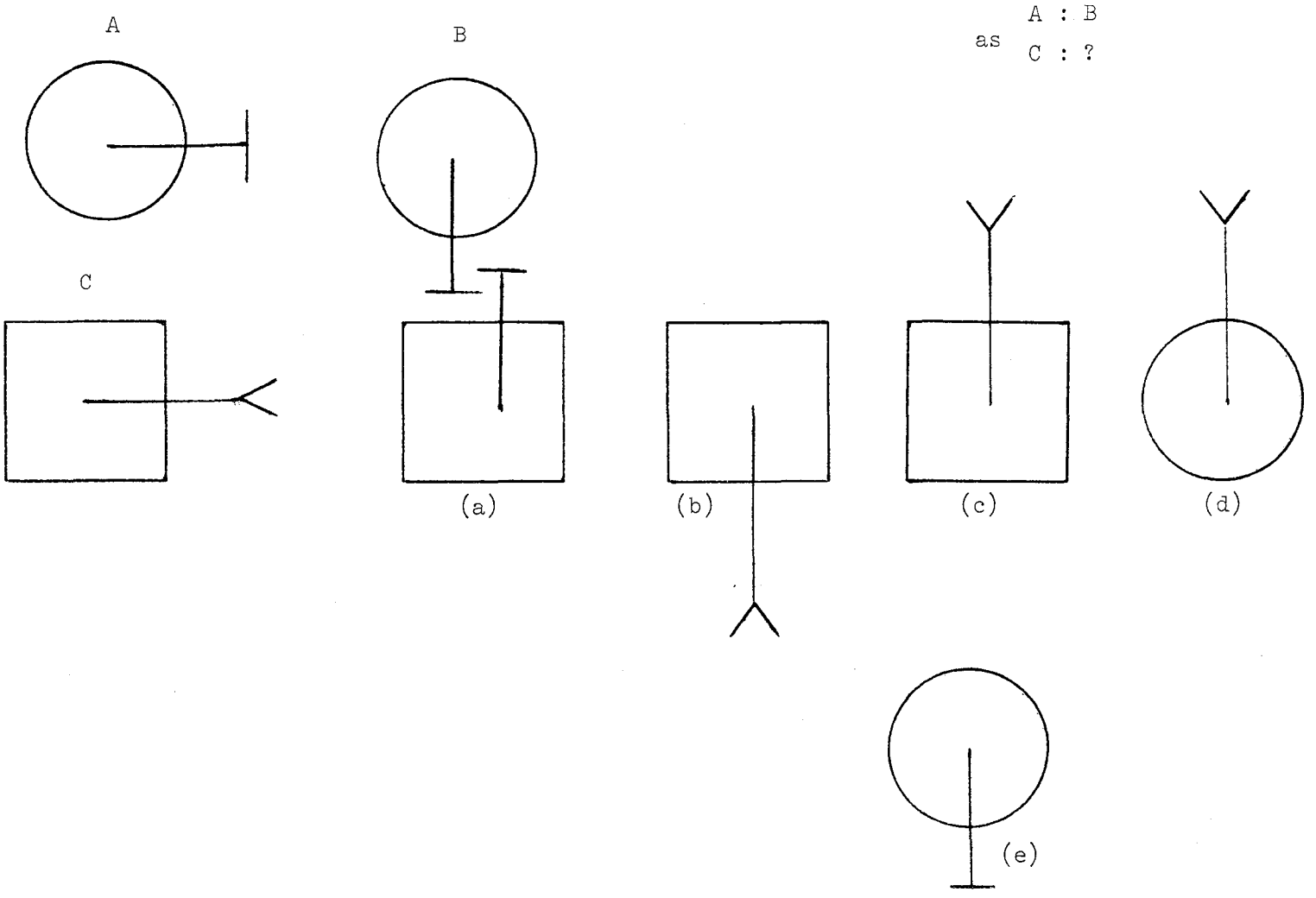


FIGURE 1. SAMPLE ITEM FROM MILLER ANALOGIES TEST.

Evans' analogy is the only implemented AI program that treats treats some of the problems explored here. In 1969, Joseph Becker[16] sketched a model of induction and analogy for a semantic memory system that added two ideas to the literature:

(1) An analogy is defined as a specific kind of one-one mapping between kernels of a semantic net.[16]

(2) A means of creating an analogy between two situations that were "essentially analogous," differing in unessential ways, is developed in terms of a weighting scheme.

Becker is interested in the processes that underly understanding natural language and concept formation. His universe of discourse deals with simple situations such as:

S_1: Peter the monkey ate two bananas at 3:00 a.m. on Tuesday. Peter is at the zoo.

S_2: Harlow the monkey ate a banana for breakfast on Saturday at home.

S_3: Susan fed Harlow a banana in the park.

He is concerned with generalizing over sets of situations such as S_1 - S_3 to "induce": "monkeys eat bananas." He gives analogy formation a crucial developmental role in the process by generating an analogy between S_1 and S_2, and between S_2 and S_3, to induce his generalization. He includes a means of scoring the relative importance of elementary facts that compose a situation. For example, here, we want to neglect Susan's feeding Harlow (S_3), and Harlow's eating on Saturday (S_2), to generate an analogy between S_1 and S_2. I am purposely vague about the details of Becker's treatment since he uses special representations and terminology that would demand too much description to develop adequately in this introduction. This work was a valuable gedanken piece since Becker explicated his analogy generation process in some detail, in contrast to Evans who neglected to explicate this process at all. Unfortunately, Becker

never implemented his model or reported extensions or variations of his paradigm.

The AI literature dealing with analogical reasoning contains only the two paradigms cited here. Any work relating analogy to problem solving must start from scratch.

(In Chapter II, we survey the kinds of information processing required by various kinds of analogies.) Fortunately, many of the processes required for non-trivial analogical reasoning (Chapters III-VI) can be carried out with the techniques that are well known to artificial intelligence: tree-search,[17] matching, etc. Since we are using analogies to expedite the search for the solution of a new problem, we need to relate our analogizing system to some existing problem-solving system. The AI tradition provides several candidates (Chapter III), and a resolution-logic system has been chosen as an experimental vehicle for the approach that is developed here. This particular problem-solving (theorem-proving) system has been developed as part of a well defined research tradition that goes back to the very first deductive problem solving system that was implemented: the Logic Theorist of Newell, Shaw, and Simon[18]. From the vantage point of heuristic theorem proving, the use of analogical information that is developed here (Chapter III) is one kind of heuristic for decreasing the search space that includes the desired proof. In fact, ZORBA-I uses an analogy to select a small set of axioms that are likely to be necessary for a problem solution from a date base that includes considerably more (irrelevant) axioms. Methods for selecting relevant axioms prior to solving a problem have been an outstanding unresolved issue in the heuristic problem-solving and theorem-proving fields. Here, we are able to provide a novel approach to this important matter.

In summary, ZORBA-I provides a link between the heretofore separate areas of reasoning by analogy, and heuristic problem solving. Little work has been reported in the former area, and the research reported in this dissertation breaks new ground in our understanding

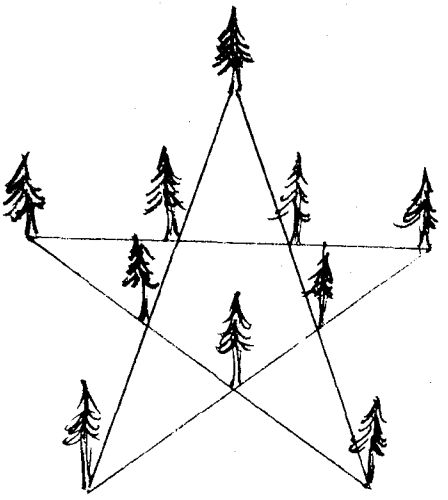
of the process of analogy generation. Heuristic problem solving is one of the classical areas of AI research, with a relatively rich tradition of paradigms and important research issues. Our work falls directly within this tradition by tackling an important unsolved issue in heuristic problem solving (data-base reduction) by applying analogical information within the context of a currently popular problem-solving paradigm (resolution logic).

C. ZORBA in the Context of Contemporary Problem Solving Research in Psychology

The use of analogies to aid problem-solving ability falls into two classical areas of cognitive psychology: concept-formation and problem-solving (or directed thinking). Unfortunately, very little of the research literature is even peripherally relevant to the work that is reported here. In 1969, Driestadt[19] reported a clever experiment that studied the use of (visual) analogies to aid the problem (puzzle) solving task. He asked his subjects to solve several problems that required a particular geometric configuration as a solution. For example, a "tree planting problem" in which ten trees must be arranged in five rows of four trees per row was presented. The problem statement and its solution are shown in Figure 2. Some of his subjects were shown a set of pictures that embodied a five-pointed star pattern required for the solution. Different pictures contained a playing card joker, a rocket zooming to the stars (Figure 2), and an aquarium with starfish. These pictures were withheld from control subjects who required significantly more time to solve these problems than subjects who were presented with the pictorial aids. Dreistadt concluded that visual analogies were a useful aid to this problem-solving task. Dreistadt's work is progressive insofar as it is the only reported research that directly relates the usefulness of analogies to problem-solving speed. Unfortunately, he doesn't study the way his subjects create the analogy and represent it to themselves.

Given data: 10 Trees

Solution:



5 rows
4 trees/row

Associated Pictures:

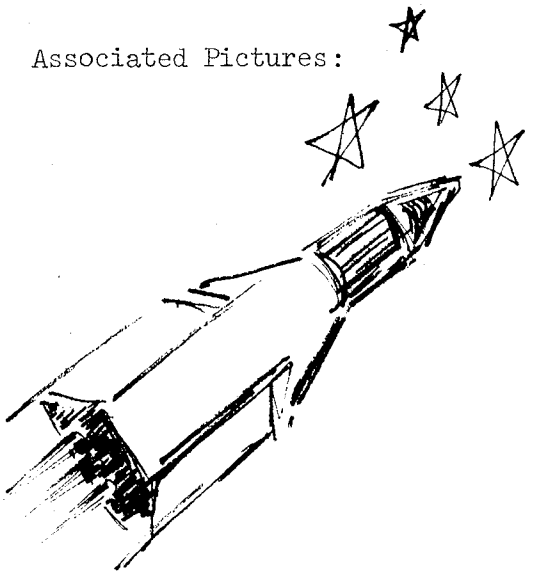


FIGURE 2. TREE PLANTING PROBLEM

Two styles of problem-solving research that potentially could aid our understanding fail to be relevant for similar reasons:

(1) Some researchers pose problems to subjects that require novel uses for familiar objects[20]. For example, a piece of paper may have to be rolled into a tube to transfer steel balls from one container to another. Various studies have been conducted to learn how problem-solving ability varies with a demonstration of the "appropriate functions," irrelevant but superficially similar training problems, etc.

(2) Learning theorists[21] have long been concerned with transfer of the solution rules of one problem set (training set) to other related problems. In a typical experiment a subject will be given a set of problems to solve. In the course of solving them he will learn some rule that applies to each problem in the set. He is then presented with a second problem set which requires a solution rule which may be similar, more general, or quite different from the rule learned in solving the first set. The subject's ease in solving the second set of problems is studied as a function of the relationship between the solution rules used in the first and second problem sets.

Both these research styles openly develop their results at a different level of generality than we need here. Most contemporary psychologists are concerned with <u>behaviors</u> rather than consciousness. The latter was exorcised from academic psychology near the turn of the century in lieu of the former, which is more amenable to experimental observation. Most experimental results are stated in behavioral terms — e.g., the use of visual analogies can aid some kinds of problem solving. The description of underlying mechanisms (other than S-R patterns) and representations necessary to develop an information-processing model of the sort we want here require infer-

ences about the contents of a person's consciousness that are unsettling to most contemporary experimental psychologists. A protocol analysis in the spirit of Newell[22] could conceivably be carried out for many of the experiments reported in the literature, if the researchers were more interested in the details of the ongoing inner processes of their subjects. Unfortunately, we have no such reports relating to the role of analogical reasoning in problem solving to rely upon.

D. Preview of the Following Chapters

The remainder of this dissertation follows a simple pattern. Chapter II is devoted to exploring the kinds of information that can be transferred between analogous problems. It was originally written as a solitary document in 1969, after ZORBA-I was conceived, and before it was implemented. Its conceptual framework is a little different from that which appears in other chapters. All except Chapter II were written after the bulk of experimental work was complete, and provide a post-ZORBA view of reasoning by analogy. Chapter II is included in its original version, since it provides an important study that serves as a pre-ZORBA introduction. ZORBA is introduced in Chapter III, and expounded in Chapters IV and V. The experimental results appear in a table at the end of Chapter V and are interspersed throughout Chapter VI. In contrast to the experimental results, a set of interesting formal properties of the algorithms is developed in Chapter VII. In particular, conditions under which the use of analogies aids problem-solving efficiency are discussed. Chapter VIII includes comments on extensions to ZORBA-I to include a wider variety of analogies. We conclude with a retrospective glance and suggestions for future research in Chapter IX. A brief note which describes ZORBA-I as an implemented operating system appears as Appendix C.

II AN INFORMATION-PROCESSING APPROACH TO REASONING BY ANALOGY

A. Introduction

Reasoning by analogy (RBA) has been discussed in artificial intelligence circles because of its extraordinary value in human problem solving and its elusiveness to mechanization. Without an ability to analogize, we would be unable to generalize, induce, or theorize. Moreover, thinking would be rather tedious, as we would have to solve each distinct problem afresh, without referring to previous experience. Fortunately the spectrum of similarities we are able to exploit is rather wide, encompassing many types, each with its own subvarieties. Unfortunately, we call much of this diverse behavior "reasoning by analogy." Hopefully, in the near future, we can develop some useful refinements for RBA. For the present, I'll simply describe some of the activities that are considered RBA.

With respect to any particular kind of analogy, RBA includes the following activities:

(1) Given a particular problem, theorem, or situation (PTS) find a previously known PTS that is analogous.

(2) Given a PTS, produce a special kind of analogous PTS. This would include producing the mechanical analog of an electrical circuit, the n-dimensional analog of a 2-dimensional geometric theorem, the continuous analog of a discrete function, the interpersonal analog of an international conflict, the French analog of a Greek idiom, etc.

(3) Given an explanation of the functioning of some PTS, provide an explanation for the functioning of some analogous PTS.

each part.

(4) Given two PTS's that are allegedly analogous, find at least one coherent analogy between them.

(5) Given two analogous PTS's and a set of consequences of observations drawn from one, infer an analogous set of consequences, inferences, or observations about the other.

manageable proportions this discussion covers problems that can be solved by deduction from some initial set of axioms, or derived by the application of a set of operations to a set of initial states, or that can easily be transformed into this form. Although it is possible to fit a wide variety of problems[24], including geometric constructions, puzzles, and robot manipulation tasks, into this framework, the majority of problems considered here are theorems in the usual sense*. More structured than "real-world problems," this class offers a decent starting point for any mechanized analogical problem solving that hopes to be successful.

1. Change of Parameters

Two PTS's are recognizable as identical up to a change of parameters — e.g.,

$$I_1 = \int_{-\infty}^{\infty} (1 + x^2)^{-3}\, dx \quad , \quad n > 0$$

$$I_2 = \int_{-\infty}^{\infty} (n + x^2)^{-3}\, dx \quad , \quad n > 0 \ .$$

Computing I_1 and I_2 are "parameter-variant" problems.

2. Generalization

In each pair of PTS, one is a generalization (or simplification of the other:

(1) Let the pair of PTS be the 3-ring and 5-ring Tower of Hanoi puzzles. The 5-ring puzzle is more general than the 3-ring puzzle.

(2) T1: Given a triangle ABC, prove that the three vertex-angle bisectors meet in a unique point.

T1′: The premises of T1 imply that this point is the center of the inscribed circle.

T2: Given a tetrahedron WXYZ, prove that the bisectors of any three dihedral angles that do not meet in a common vertex intersect in a unique point.

T2′: The premises of T2 imply that this point is the center of the inscribed sphere.

3. Similar Relational Structures

The pairs of theorems T3/T4 and T5/T6 are "relationally isomorphic" when represented as graphs with nodes and links of different types to represent relations and objects of different classes. (The partitions of nodes and branches is, in effect, a categorical semantics for the graph language.) In viewing the proofs of these theorem pairs, one finds that they are identical up to a set of substitutions (e.g., abelian group/commutative ring, angle bisector/perpendicular bisector, etc.) that results from the mapping associated with the analogy.

T3: The bisectors of the three vertex angles of a triangle intersect in a unique point that is the center of the inscribed circle.

T4: The perpendicular bisectors of the three sides of a triangle intersect in a unique point that is the center of the circumscribed circle.

T5: The intersection of two abelian groups is an abelian group.

T6: The intersection of two commutative rings is a commutative ring.

This class is an extension of the parameter-variant class, and with some provision for mapping sets (clusters of nodes) into sets of different cardinality, they may also include many generalization-type analogies. Note that the relational isomorphism is "local."

The preceding analogies were selected for their transparency, but even isomorphisms can be complex. For example, consider:

T7: Let ABC and abc be two triangles in the same plane defined within a k-dimensional finite geometry over the Galois field $GF[p^n]$. Let these triangles be perspective for a point O, such that O, A, and a are collinear, O, B, b are collinear, and O, C, c are collinear. Let γ be the point of intersection of AB and ab, β that of AC and ac, and α that of BC and bc. Then the points α, β, and γ are collinear.

T8: Let X, Y, and Z be three subgroups of a geometric set of subgroups of G_k such that no one of them in the group is generated by the other two. We select other subgroups of the geometric set as follows: each of them is in the group {X,Y,Z}; O is any such subgroup not contained in any one of the subgroups {X,Y}, {Y,Z}, {Z,X}; x,y,z are such subgroups different from O,x,y,z and contained respectively in {O,X}, {O,Y}, and {O,Z}. Let μ, ξ, and ν be the subgroups of the geometric set of subgroups common to the respective pairs of groups {X,Y},{x,y}, {Y,Z},{y,z}, and {Z,X},{z,x}. Then each of the two subgroups μ, ξ, and ν, is in the subgroup generated by the other two.

Every k-dimensional projective geometry over a Galois field $GF[p^n]$ is capable of a concrete representation by an abelian group of order $p^{(k+1)n}$ and type (1,1,1,...,1) if we consider each subgroup of order p^n as a point in the geometric space.[25] This association renders T7 logically equivalent and relationally isomorphic to T8, although this correspondence is hardly obvious.

4. Plans are Identical

Plans for the solution of each PTS are identical (at some level of abstraction) — e.g.,

(1) T9: If a given affine transformation commutes with every other affine transformation, then that transformation is the identity.

T10: If a given affine transformation commutes with all the translations, then that transformation is also a translation (see Fig. 3 for proof plan).

(2) T11: If F(w) is the Fourier transformation of f(t), prove that $e^{-jwt}F(w)$ is the Fourier transformation of f(t-T).

T12: If F(w) is the Fourier transformation of f(b), prove that $\frac{1}{(a)} F\left(\frac{w}{a}\right)$ is the Fourier transformation of f(at) (see Fig. 4 for proof plan).

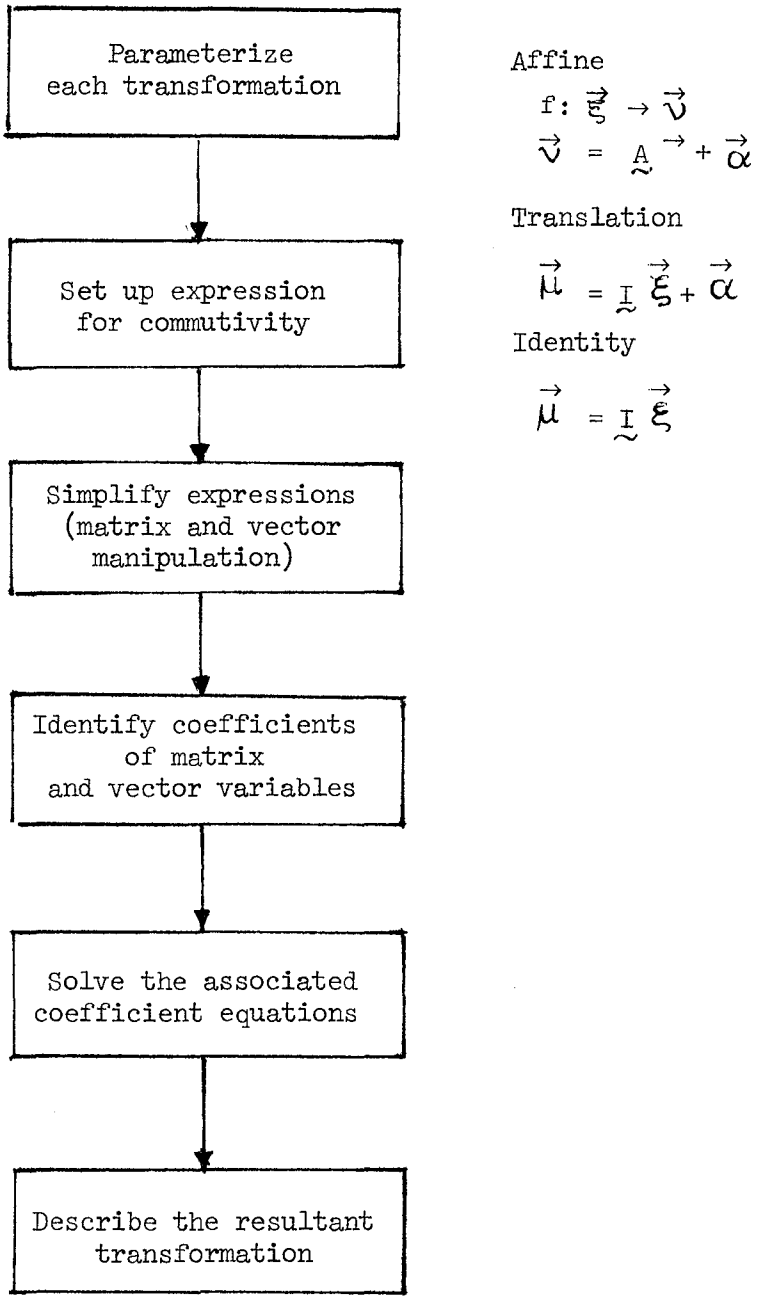


FIGURE 3. PLAN FOR PROVING THEOREM 9 OR THEOREM 10.

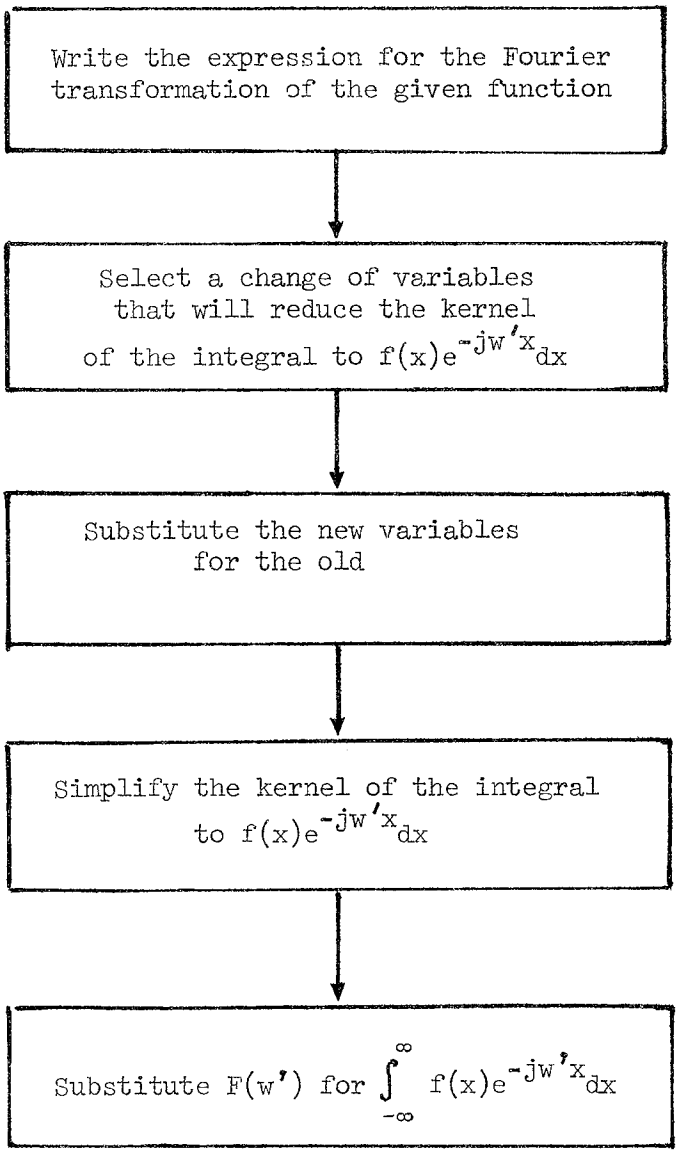


FIGURE 4. PLAN FOR PROVING THEOREM 11 OR THEOREM 12

(3) T13: Let an arbitrary line ℓ_1 intersect each of three parallel lines s_1, s_2, s_3 at points p_1, p_2, and p_3, respectively. Let another arbitrary line ℓ_2 intersect s_1, s_2, and s_3 at points q_1, q_2, and q_3, respectively. Then

$$\frac{\overline{p_1p_2}}{\overline{p_2p_3}} = \frac{\overline{q_1q_2}}{\overline{q_2q_3}}$$

T14: Let an arbitrary line ℓ_1 intersect each of three parallel planes s_1, s_2, and s_3 at points p_1, p_2, and p_3, respectively. Let another arbitrary line ℓ_2 intersect s_1, s_2, and s_3 at points q_1, q_2, and q_3, respectively. Then

$$\frac{\overline{p_1p_2}}{p_2p_3} = \frac{\overline{q_1q_2}}{\overline{q_2q_3}}$$

(See Fig. 5 for proof plan.)

Although a coherent planning language for this diverse set of problems has not yet been written, it is clear that they are "identical" at some level of abstraction easily accessible to people.

5. Change of Representation

The solutions of both PTS's involve a common change of representation and style of argument.

(1) P1: Consider the classical truncated checkerboard domino-covering problem.

P2: Consider a 3 x 3 cubical apple with a worm on its surface. The worm travels from cube to adjacent cube, boring a hole without ever

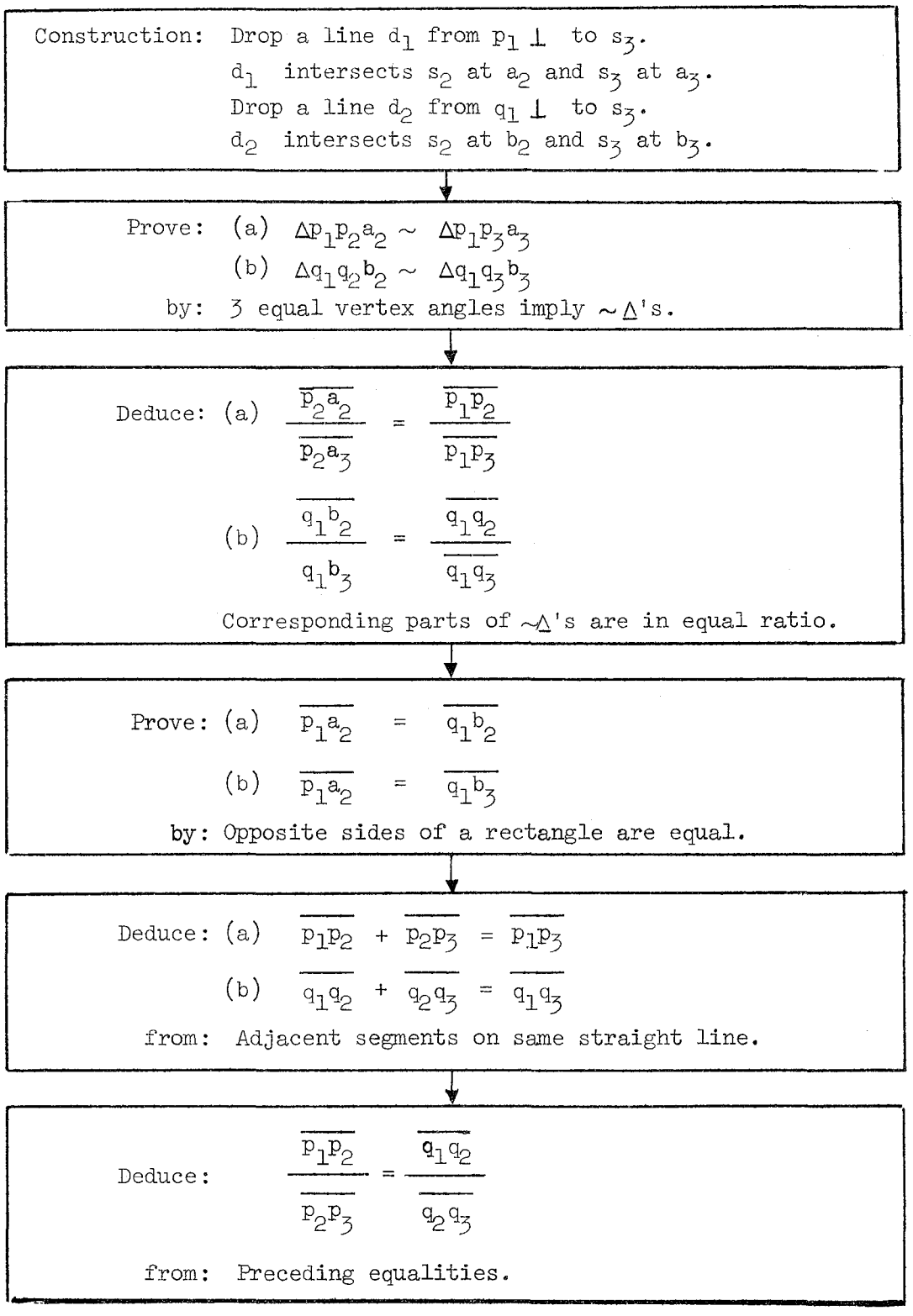


FIGURE 5. PLAN FOR PROVING THEOREM 13 OR THEOREM 14

returning to a previously drilled subcube. Prove that it is impossible for the worm to terminate his path in the centermost cube.

(2) If A is a matrix with transpose A^T,

T15: $(A^T)^T = A$

T16: $(A\ B)^T = B^T\ A^T$.

(3) If A is a matrix with inverse A^{-1},

T17: $(A^{-1})^{-1} = A$

T18: $(A\ B)^{-1} = B^{-1}\ A^{-1}$.

Each of the preceding problem pairs entails similar representations. The truncated checkerboard and the cubical apple problems are both solved by coloring adjacent cells black and white and then using a parity argument. If we restrict ourselves to a simple operator-free matrix algebra, T15 and T16 are most easily proved by representing each matrix by its ij^{th} element and manipulating the ij^{th} terms according to the specified computations. On the other hand, T17 and T18 are both easily proved by simple algebraic operations. What a person extrapolates from T15 to T16 or from T17 to T18 is a specific representation in which problem-solving ability is enhanced. If a person were faced with the problem $(A^T)^{-1} = (A^{-1})^T$, he might be unsure of which representation to choose, and would try either one. (It turns out that either representation affords straightforward proofs.)

6. Common Subproblem

Both problems involve a common subproblem.

(1) Let $\underset{\sim}{A}$ be a matrix with elements a_{ij} and an inverse $\underset{\sim}{B}$. Then

$$b_{ij} = \frac{c_{ij}}{\det[A]}$$

and

$$\det[A] = \sum_{j=1}^{n} (-1)^{1+j} a_{ij} c_{ij}$$

where c_{ij} is the i-j^{th} cofactor of $\underset{\sim}{A}$. Thus, the computation of $\underset{\sim}{A}^{-1}$ and $\det[\underset{\sim}{A}]$ share the common subproblem of computing some cofactor of $\underset{\sim}{A}$.

(2) Consider a robot in a room full of scattered metal furniture.

P1: The robot is asked to paint the floor of the room.

P2: The robot is asked to replace each piece of furniture with a similar wooden piece from the next room.

Each of these problems can be solved by first clearing all the metal furniture out of the given room, and in that sense they are analogous.

Problems that involve only one common subproblem can be really rather different and still allow useful problem-solving extrapolations. Probably these extrapolations are best regarded by treating the subproblems as substantial problems unto themselves. For example, every time we encounter a trigonometric integral in solving some problem, we become better integrators and increase our facility for rapidly guessing appropriate substitutions. Thus, the extrapolation of integration techniques from one problem to another is due to recognizing the need for our developed skill as an integrator, rather than noting some gross aspects of problem structure.

D. Information Transfer Between Problem Solutions

A bold step toward RBA will be taken when an automatic algorithm for creating an analogy between two problem statements is developed. Presumably, such an algorithm need only know the two theorem statements and have access to the data base of axioms. Even if one has a detailed analogy that is limited to the relations and objects explicitly mentioned in the theorem statements, one still must know how to use this information to accelerate the search for a solution. While some useful information may be gleaned from this "restricted analogy," much of the information in a proof of the new theorem often involves additional relations, facts, and patterns of inference that are absent from the problem statement. Any interesting analogy-generating algorithm will need to operate upon theorem proofs as well as theorem statements. The search for analogous "additional information" helps pin down the viability and level of abstraction that can be expected from a given analogy. If we don't obtain much side information, we may believe that our analogy is too specific. If we obtain too much, our analogy may be too general.

In any but the most simple problems, the solution is derived in terms of relations that do not appear in the problem statements.

Suppose we had a magical system that could offer information helpful to proving an unknown theorem if it were given an analogous proved theorem. What kind of interesting advice could we expect from this program? At one extreme it might be clairvoyant and offer a complete solution to the baffling problem. Short of such omniscience, what kind of partial information would be helpful? Textbook writers often append hints of two types to the problems they provide:

(1) Problem difficulty (easy, hard)

(2) "Hints" that include:

  (a) Suggested representation

  (b) Appropriate methods

(c) Relevant principles or theorems

(d) Valuable subproblems.

In the second case, there seem to be three different levels of "heuristic detail," each with possible attendant information.

1. Representations

Representations are mentioned in Section C-5 of this chapter.

A style of argument may be added — e.g., induction, parity, etc. Additional details such as which parameter to induce on may be included.

2. Plan

Consider a problem solution as a sequence of states S_j and state transition operators P_j, as in Fig. 6 below:

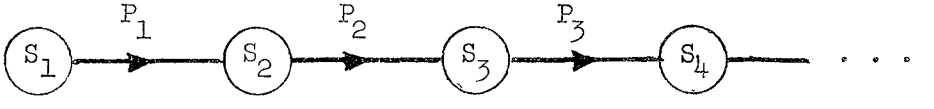


FIGURE 6. A PLAN DEPICTED AS A SEQUENCE OF STATES

Although the depicted operators have unary inputs, several inputs may be possible, as in inference from several intermediate results (states)[26]. Likewise, there may be several outputs — e.g., a problem may be split into several intermediate subproblems. A plan is any sequence of state descriptions and/or operator descriptions that parallels an alleged problem solution. These descriptions are usually abstracted versions (patterns) that may have several candidates in the object

language. In this sense they are weakly specified. Several varieties of information may be offered as a plan:

(1) A sequence of operations or methods may be specified (Fig. 4).

(2) A sequence of patterns that describe the expected state sequence (a state "monitor").

(3) A sequence of subgoals described in the object language (Fig. 5).

A functional planning language needs at least four features:

(1) Its own logic

(2) A well-defined nexus between the planning states/methods and the object-language states/methods

(3) An ability to reference future results

(4) Some facility to manipulate data representations so that the flow between different processes is smooth.

These features are integral to an autonomous planning system. When we focus on the kinds of information that can be extrapolated from problem to problem at a planning level we find:

(1) Plan, as described above

(2) Estimates on the difficulty of various subproblems

(3) Conditions describing when to terminate a process: "simply the kernel of an integral to $f(x)e^{-jw'x}$ (Fig. 4)

(4) Operator inputs: "deduce S_j from Theorem T_k and S_{j-1}."

This wealth of side information (nonsequential) associated with a plan can range in abstraction from being detailed

in the object language (a particular theorem) to some more abstract description (e.g., a theorem relating groups and homomorphisms, a sufficient condition for a set to be a group, etc.)

3. Object Language Level

(1) Explicit subproblems and lemmas (Fig. 5)

(2) Relevant theorems that will be used in the proof (Fig. 5)

(3) The set of relations to be used in the problem solution

(4) Problem difficulty.

The set of relevant theorems need not be structured with their relevant subgoals as in Fig. 5, but may be an unordered set of which the PSS is conspicuously conscious.

It is now clear that the range of helpful advice is rather broad, both in level of detail and degree of structure. Although a restricted analogy could be generated first and the "helpful information" later, it would be nice if some of it were a byproduct of the analogy-generating program.

E. Automated Use of Analogical Information

In this section, I will combine several themes that have run through this chapter and apply them to the automation of analogical problem solving. First, I will summarize some of the key points that I have mentioned in the preceding sections.

(1) The idea of analogy is ill defined. There are at least several kinds of related analogies.

(2) Each of these analogical varieties would best be recognized by somewhat different means.

(3) The kinds of information that are extrapolable between analogous problems of each variety are quite different. Thus, the algorithms and designs for using these diverse types of information are likely to be quite different.

(4) One of the key issues in extending analogical information is knowing in advance the level of generalization that will hold for each analogized parameter, method, operation, theorem, or fact.

(5) A set of strategy/planning languages that would allow an appropriate degree of generality would be quite complex. These facts imply:

   (a) No analogy-oriented PSS* (APSS) should be expected to process all varieties of analogy, since each involves a somewhat different style of information processing.

   (b) An APSS that attempts to extrapolate general sequential plan-like information or patterns of inference, and attempts to actively direct a problem solver that incrementally infers and tests inferences against its supervisory schema would be quite complex.

Many of the example problems presented in Section C of this chapter push the limit of contemporary PSS and will probably be non-trivial for any of the planning-oriented systems that will emerge in the next few years. Thus, we end up wanting to use analogical information <u>without</u> creating plans or other forms of skeletal solution structures.

The means of doing this are actually very simple if we review our situation again. A typical APSS will have a large data base and be presented with a pair of problems: One is unsolved and the other

---

* A PSS is a "problem-solving system."

has already been solved and its solution is available to the APSS. I want to underscore a critical way in which this situation differs from the typical PSS. Most PSS's work on a "minimal" data base for which the user has selected an adequately small set of axioms*. When the data base of a typical PSS is expanded to include some irrelevant axioms, they begin to generate a substantial set of irrelevant inferences (due to the interaction of relevant and irrelevant axioms and their descendent inferences). Consequently, they begin to flounder in their search and may fail to solve problems that are easily solvable with a minimal axiom set. To be concrete, consider a PSS that uses 8 axioms to prove some theorem T with a search that generates 100 inferences for, say, a 20-step proof. Adding 10 more axioms to the data base may force it to generate 500 inferences before finding its 20-step proof. In a sense, these figures are doctored, since a set of 10 axioms can be chosen that will have no appreciable effect on the search-space size, while another set of 10 may be added that can explode the search space almost arbitrarily.†

Since an APSS will be proving somewhat diverse theorems with a (usually) common data base, it is in principle bound to seek proofs in a context abundant in excessive and irrelevant data. One key method for exploiting analogical information is to select a subset of axioms appropriate for proving the new theorem. Then, we are constricting the context in which theorem proving takes place by narrowing the set of accessible axioms. The usual strategies of the particular PSS can be used unmodified; the analogical information

---

*All known resolution systems[17] and GPS[27] operate this way. Gelernter's Geometry theorem-prover[28] is the only system that accessed a superset of the necessary axioms. He used a special model to cut his search space to include only relevant inferences.

†In the first case, add axioms that use many distinct predicate letters and many distinct function symbols. In the latter case, use axioms with only one or two predicate letters, and choose axioms that will resolve with most of the others, preferably recursively.

is merely used to narrow the context sufficiently to reduce the search space to a more manageable size. Selecting an appropriate axiom set is one of several kinds of information that may be added independently of PSS strategy. A more complete and suggestive listing includes:

(1) Restricting the set of admissible relations

(2) Restricting the set of admissible operator symbols

(3) Restricting the set of admissible axioms

(4) Restricting the order of operator nesting

(5) Generating analogous subproblems, solving them, and adding them as axioms.

This list can be extended, depending on the kinds of information used by a particular PSS. Thus, if a PSS has a look-ahead estimator (like REF-ARF)[29], then that too may be analogized without modifying the PSS structure. The key idea is that an effective means of exploiting analogical information is to modify the context in which a particular theorem prover operates, rather than subjecting it to a planning-like scheme that supervises the sequence of its inference making.

Now, the actual means for generating analogies and extrapolating analogous axioms depends upon the representations and PSS used. These details have been developed and implemented for a resolution-based theorem prover and are described in the following chapters.

III AN INTRODUCTION TO ZORBA

A. Introduction

ZORBA, outlined in this chapter, 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 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 the situation in 1968, we have an elementary framework for making these questions and their answers operational.

B. 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 α: $P_A \longrightarrow P$.

(3) Construct $\alpha^{-1}(S') = S'_A$.

(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 have come to 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 the research was begun, it was an inappropriate approach for two reasons:

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

---

*PLANNER[30] at MIT and QA4[31] at SRI are two current planning-oriented problem solvers 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.

analogy will map into an executable analogous plan. If S'_A fails, is α too strong, or wrong? Should α be modified and a variant S''_A computed, or should the system keep α, 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 that would involve a good planning-oriented problem solver as an easily accessible research tool. At worst, the preceding paradigm may be too simple and a suitable α may be developed interactively 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 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 Fig. 7). In addition to the subbase for solving P and P_A

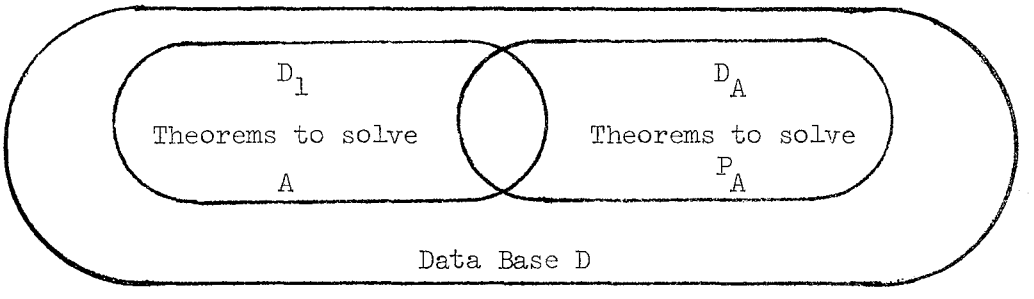


FIGURE 7. VENN DIAGRAM OF THEOREMS IN DATA BASE

there 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. $D_1 \subseteq D_2 \subset D$ 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 assertion (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 $size[D_1] \leq size[D_2] \ll size[D]$. For example, a typical theorem in algebra provable by QA3[32] — 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 obtained. 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_A would be a set of lemmas (or subgoals) $L_1, \ldots, L_j$ whose analogs $\alpha(L_1), \ldots \alpha(L_j)$ could be solved by the system before attempting P_A.

---

*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[33] emphasizes this point with regard to GPS, and I have had similar experiences with QA3[32], a resolution logic theorem prover.

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 that forces backup in case of failure. Suppose we somehow get $\alpha(L_1)$, ... $\alpha(L_j)$. A typical planner would order the $\alpha(L_i)$, e.g., $\alpha(L_1)$, $\alpha(L_2)$ ... etc., attempt to solve them in sequence, and stop if any lemma fails to be solved. In contrast, we merely need to attempt each $\alpha(L_i)$. If we get a solution, we add $\alpha(L_i)$ to the data base (like a theorem)‡ and continue with the next lemma. If we fail, we continue anyway. At worst, we wasted some computation time. Each useful $\alpha(L_i)$ 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 $\alpha(L_j)$ need <u>not</u> 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 <u>environment</u>‡ in which a problem solver operates, we can effectively

---

*Recognizing lemmas depends on 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 on the system's ability to associate variables with variables, and the association may be tricky when skolem functions are introduced.

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

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 on the problem-solving system (and even the domain of application). Any further discussion needs to specify these two items.

C. 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[32]. For the class of analogies these programs handle, this is an accomplishment. When we begin to focus on a particular paradigm, two issues are more easily resolved:

(1) What kinds of information are most useful to provide the problem solver?

(2) Which representations shall we use to describe the analogies and handle the necessary data?

Actually, these two issues interact. For example, if we want to study planning-level analogies, then we need a problem-solver that can create and execute plans for the problems it attempts. In turn, we expect to be passing it information that refers to its sequence of plans, criteria for subgoal completion, etc. Many important details of this research are affected by our choice of problem-solving system. In addition, the classes of analogy we can study are affected by the kinds of problem-solver that we choose. Earlier I noted that a planning-problem-solving system was not available when this research was begun. In the beginning stage of this research it was unclear how wide a variety of problem domains we

would like to consider. Both abstract algebra and plane geometry are rich in analagous problems, and we wanted to be able to consider at least both. Thus, rather than turning to a specialized problem-solving system like Gelernter's GEOMETRY[28] machine or Norton's Group-Theorist[29], we decided on a general-purpose system. Our choice between a resolution-based system and a GPS-like system was strongly influenced by the recent development of a relatively flexible resolution-theorem prover in the laboratory at Stanford Research Institute where this research was carried out. In fact, this resolution system, QA3, was implemented in LISP on the SDS-940, the same language and same machine on which ZORBA was to be implemented*. No other, equally powerful problem-solving system was available in either LISP or on the SDS-940 at that time. At least a year's work was saved by opting to use QA3 as an experimental vehicle.

Resolution is attractive on its own merits, as well:

(1) It is a highly popular inference system that is currently receiving a vast amount of attention. The results of ZORBA-I can be relatively easily related to other developments in this "hot" area of study.

(2) Resolution uses first-order predicate calculus, which has substantial expressive power. Any problem whose solution may be deduced from a set of first-order axioms in some natural inference system can be transformed into a resolution theorem†. QA3[24] has been used to solve the monkey

---

*ZORBA programs were later converted to PDP-10 LISP when the SRI Artificial Intelligence Laboratory changed machines.

†Since our discussion is shifting to resolution, our terminology will shift from the language of problem solving to the language of theorem-proving, with the following equivalences:

problem ~ theorem

solution ~ proof

and bananas problem and the tower of Hanoi puzzle, handle questions pertaining to drug interactions, make diagnostic inferences in a simple medical application, and prove theorems in geometry, algebra, and number theory.

(3) Practical resolution systems are more powerful than competing systems like GPS. The resolution system allows more natural representations for some applications, particularly mathematics.

Resolution logic is an inference rule whose statements are called clauses* [17]. 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.

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

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

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.

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 is 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 to 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, T1 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, COMMUTATIVERING}, the second set can map into anything.

Figure 8 shows a set P including all the predicates in the data base.

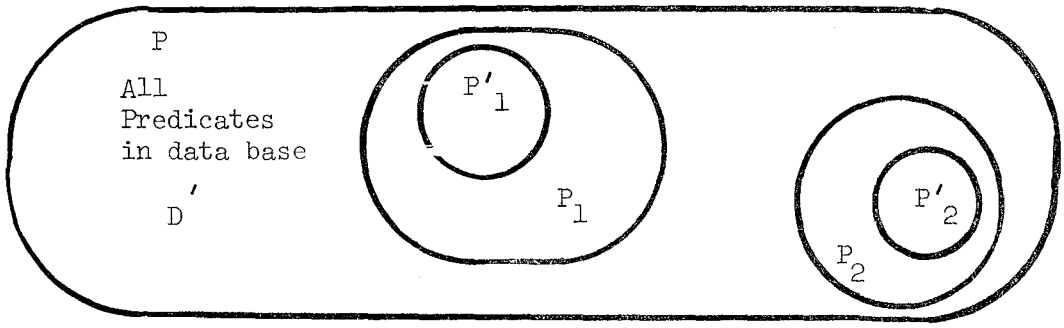


FIGURE 8. VENN DIAGRAMS OF RELATIONS IN STATEMENTS T, T_A, AND D'

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 α: $\alpha(P_1) = P_2$.

Clearly, a wise method would be to find some α', a restriction of α to P'_1 such that $\alpha'(P'_1) = P'_2$. Then incrementally extend α' to α'_1, α'_2, ..., each on larger domains until some $\alpha'(P_1) = P_2$. 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 α'_j is faulty. The next section will describe the generation algorithm in a little more detail.

D. 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 on 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 that 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 explicity, a ZORBA-I analogy α is a relation $\alpha^P \times \alpha^c \times \alpha^v$, where:

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

(2) α^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) α^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., α^v and/or α^p, and/or α^c. Usually I will drop the superscript and simply refer to "the analogy α." Thus, "the analog of an axiom ax_k under analogy α" should be understood to mean $\alpha^c[ax_k]$, and will often be mentioned simply as "the analog of ax_k."

In the previous section I refer to a sequence of analogies $\alpha_1, \ldots \alpha_k$. 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" 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 α_k is an extension of a partial analogy α_j if some of α_j — e.g., α^p_j, α^c_j, α^v_j — is a submap restriction of the corresponding submap of α_k to a smaller domain. Intuitively, when we add a new predicate or clause association to α_j so as to create α_k, we say that α_j has been extended to α_k. We are now ready to survey ZORBA-I.

E. 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 chapters. 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], which is an ordered set of clauses, $\{c_k\}$ s.t. $\forall k$ c_k, is defined by:

  (a) A clause in $\neg$ T, or

  (b) An axiom, or

  (c) Derived by resolution from two clauses c_i and c_j $\quad j < k$ 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 RELSTRUCTURE; 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:

(1) $\forall_p$ s.t. p and $\neg$ p appear in c, impcond[p] $\in$ descr[c].

(2) $\forall_p$ s.t. p appears in c, pos[p] $\in$ descr[c].

(3) $\forall_p$ s.t. $\neg$ p appears in c, neg[p] $\in$ descr[c].

Thus, the axiom, every abelian group is a group* — e.g.,

$\forall$(x *) abelian[x;*] $\rightarrow$ group[x;*] — is expressed by the clause

c_1: $\neg$ abelian[x;*] $\vee$ group[x;*], 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.,

$\forall$ (x y $*_1$ $*_2$ φ)
hom[φ;x;y] $\wedge$ group[x;$*_1$]
$\rightarrow$ group[y;$*_2$] — is expressed by the clause

c_2: $\neg$ hom[φ;x;y] $\vee$ $\neg$ group[x;$*_1$] $\vee$ group[y;$*_2$]

and is described by

neg[hom], impcond[group].

Two different clauses may have the same description.

Let:

c_3: $\neg$ intersection[x;y;z] $\vee$ subset[x;y]

c_4: $\neg$ intersection[x;y;z] $\vee$ subset[x;z].

---

*See Appendix A for the definitions and semantic templates of the predicate letters.

Then:

descr[c_3] = descr[c_4] = neg[intersection], pos[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 axion 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 $d \subseteq$ descr[c]. Thus, several clauses may satisfy the same description.

Let:

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

c_6: $\neg$ subgroup[x;y;*] $\vee\neg$ subset[x;y].

Then, the following statements are true:

(1) {c_2,c_5} satisfy impcond[group]

(2) {c_1,c_2,c_5} satisfy pos[group]

(3) c_1 satisfies neg[abelian], pos[group]

(4) {c_3,c_4,c_6} satisfy pos[subset]

(5) c_6 satisfies neg[subgroup], pos[subset]

(6) No clause of these six satisfies pos[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 predi-

---

*The "analog of a description" is defined in Chapter V.

cates 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 chapters 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 α_1^p, which is used by EXTENDER to start its sequence of α_j^p and α_j^v, which terminate in a complete α. INITIAL-MAP starts without a priori information about the analogy it is asked to help create. Both α^p and α^v are empty when it begins. INITIAL-MAP uses the pair of wffs that express T and T_A as well as the restrictions imposed by the semantic categories to generate α_1^p 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 proof[T_1], and the statements of T_3 - T_4 can contain five of the 12 predicates used in proof[T_3]. In brief, INITIAL-MAP provides a starting point from which EXTENDER can develop a complete α.

The INITIAL-MAP uses an operator called atommatch[$atom_1$;$atom_2$;α], which extends analogy by adding the predicates and mapped variables of $atom_1$ and $atom_2$ to analogy α. 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 α_1^p.

---

*Atoms, not predicates.

EXTENDER accepts a partial analogy generated by INITIAL-MAP and uses it as the first term in a sequence of successive analogies a_j. The axioms used in proof[T] are few in comparison to the size of the large data base, and comprise the "domain" for a complete a^c. For each axiom used in proof[T], we want to find a clause from the data base that is analogous to it. The axioms used in proof[T] are called AXSET and are used by EXTENDER in a special way. Each partial analogy a_j^p is used to partition AXSET into three disjoint subsets called all[a_j], some[a_j], and none[a_j].

If all the predicates in an axiom $ax_k \in$ AXSET are in a_j^p, then ax_k is in all[a_j]; if some of its predicates are in a_j^p, then ax_k is in some[a_j], and if none of its predicates are in a_j^p, then ax_k is in none[a_j]. For brevity, these sets will be called ALL, SOME, and NONE, and their dependence on a_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 a_j, $j = 1, \ldots n$, 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 can not 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 a_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 a_j^p. We can extend a_j to a_{j+1}, and thus the analogs of axioms on SOME provide a bridge between the known and the unknown, between the current a_j and a descendent a_{j+1}, and thus the analogs of axioms

on SOME provide a bridge between the known and the unknown, between the current α_j and a descendent α_{j+1}. When EXTENDER has satisfactorily terminated, ALL = AXSET, SOME = NONE = $\emptyset$. So the game becomes finding some way to systematically move axioms from NONE to SOME to ALL in a way that for each ax_k moved, some analog $\alpha_j(ax_k) = ax'_k$ is found that can be used in the proof of T_A. Moreover, each new association of clauses should help us extend $\alpha_j \rightarrow \alpha_{j+1}$ by providing information about predicates not contained in α_j.

The following chapters are devoted to a detailed explication of ZORBA-I. INITIAL-MAP is a comparatively simple system and will be covered in Chapter IV. EXTENDER is far more novel in its conception and complex in its details. It will be introduced in Chapter IV and examined in greater detail in Chapter V. I recommend that the serious reader skim these two chapters to acquaint himself with most of the concepts and a few examples. Such a prelude will illuminate the following discussion like the bright sun burning off a morning fog.

IV A DESCRIPTION OF INITIAL-MAP

A. Introduction

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 chapter I hope to evoke 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 in which 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 in Chapter VIII.

In the previous chapter I provided a rationale for 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:

$Zorba_1$[newwff;oldwff;AXSET*]:=

(1) Set analogies to the list of analogies generated by initialmap[newwff;oldwff].

(2) Apply extender[analogy; AXSET] to each analogy or <u>analogies</u>.

(5) 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 chapter.

ZORBA-I is designed to find the analog of each axiom in AXSET and thus create α^c. The brief description of EXTENDER in the previous chapter suggests that if we know the analog of at least one predicate in the domain of α^p, then we can partition AXSET into ALL, SOME, and NONE to start EXTENDER creating a series of partial analogies α_2, α_3, ... that terminates in some complete α_n. The algorithm for INITIAL-MAP that is described here is designed to find an association between each predicate in the statement of T_A with each predicate that appears in the statement of T. For most interesting theorems, the theorem statements are usually expressed with more than one predicate. Consequently, INITIAL-MAP will typically provide an α_1^p that will have more than one predicate association and that is more than sufficient to initiate EXTENDER. In Chapter VIII, a simpler version of INITIAL-MAP that often works will be described. Once the system gains some experience (creates some analogies) in a particular domain, it could dispense with INITIAL-MAP and use the analogs of those predicates that it found in the past and appear in proof[T] as α_1^p. However, here we will adopt a quite conservative approach and show how a good α_1^p can be developed in the absence of any a priori predicate associations whatsoever.

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

---

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

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 by-product, 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[$atom_1$; $atom_2$; α] as an operation to add the predicate/variable information to analogy α. As its name hints, ATOMMATCH matches the predicates and variables of its atomic arguments and adds the resultant mapping to the developing analogy (α).

B. The Design of ATOMMATCH

ATOMMATCH is used as an elementary operation by every matching routine in the INITIAL-MAP system (Figure 9). Thus, we will discuss it first, and then consider how INITIAL-MAP is organized to apply it intelligently.

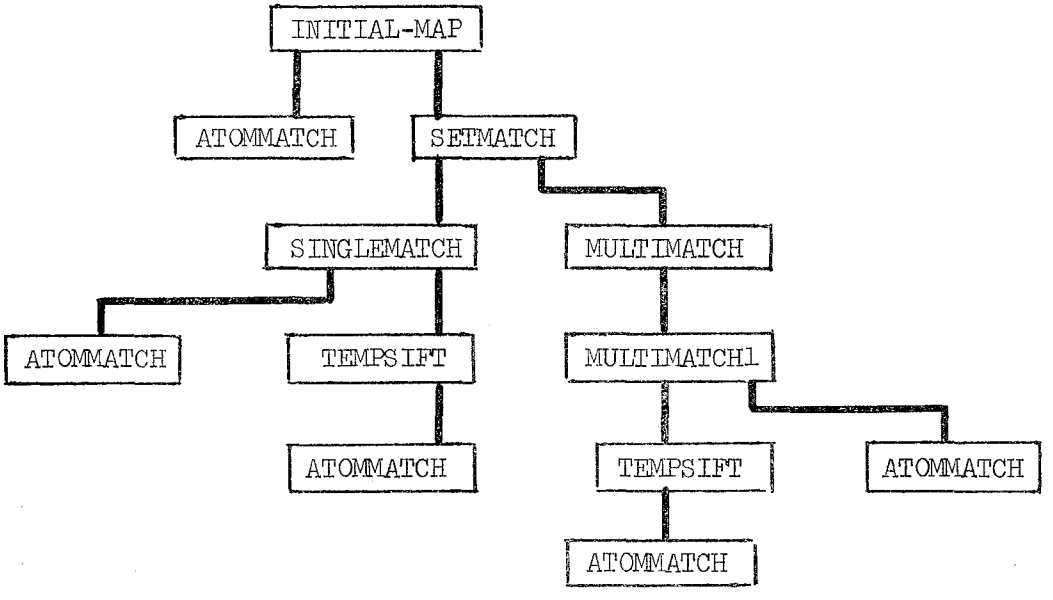


FIGURE 9. HIERARCHY OF MATCHING ROUTINES CALLED BY INITIAL-MAP

Consider how we might write an ATOMMATCH. Suppose $atom_1$ and $atom_2$ are of the same order (same number of variables) and each variable place in each atom has the same semantic type. For example, let

$$atom_1 = \text{intersection}[x_1;x_2;x_3]$$

$$atom_2 = \text{intersection}[y_1;y_2;y_3]$$

Clearly, we want

$$\text{intersection} \sim \text{intersection}^*$$

and

$$x_1 \sim y_1 \ , \ 1 = 1, 2, 3 \ .$$

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

and $atom_2 = q[y_1; \ldots y_m]$ and $p = q$ (thus, $n = m$) and we will set

$$p \sim q$$

and

$$x_i \sim y_i \ , \ i = 1, 2, \ldots, n.$$

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

$$atom_1 = \text{group}[x; *_1]$$

and

$$atom_2 = \text{ring}[y; *_2; +_2].$$

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 ring is structure[set;operator;operator]. We will map

---

*I will use the symbol "$\sim$" as in "$x \sim y$" to mean "x is associated with (analogous to) y."

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 ensure that a variable (possibly dummy) of the same semantic type is in corresponding places in each atom. Then we can map the variables one-one in order of appearance. For example, we can associate

$$x \sim y$$

and

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

Then, we can remove dymmyop and rewrite

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

We can describe this process formall in two stages:

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

Let

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

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

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

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

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_1 and x_{1+1} belong to the same partition if $\text{type}[x_1] = \text{type}[x_{1+1}]$. Now there are an equal number of partitions in both

$atom_1$ and $atom_2$. Returning to our example, we partition group[x;$*_1$] into [[x], [$*_1$]], and the 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.

C. The INITIAL-MAP Control Structure

We have two wffs representing T and T_A as arguments of INITIAL-MAP, and we want to find some way to slide ATOMMAICH 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 and p and q are unary predicates.

We would presume that $p \sim q$ (predicates)
$x \sim y$ (variables)
and $A \sim B$ (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 T and T_A to have the formats that can be represented by the generative grammar:

$$T \rightarrow (A \rightarrow A)$$

$$A \rightarrow p[x_1 \ldots x_n]A \mid p[x_1 \ldots x_n] \ .$$

INITIAL-MAP is designed to decompose the input wffs T and T_A into associated syntactic substructures until a subwff is either an atom $p[x_1 \ldots x_n]$ or a conjunction of atoms $\bigwedge_{i=i}^{k} p_i[x_1 \ldots x_n]$. At this point INITIAL-MAP enters a hierarchy of selecting and matching routines (Figure 9) 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;*] ∧ abelian[y;*] ∧ intersection[z;x;y] the semantic partition will be
{{intersection[z;x;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.

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

(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:

$$partition_1 = \{\{atom_1\ atom_2\}\{atom_3\ atom_4\}\}\{atom_5\}$$

$$partition_2 = \{\{atom_6\ atom_7\}\{atom_8\ atom_9\}\}\{atom_{10}\}.$$

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 partions 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 $\{atom_1\ atom_2\} \sim \{atom_6\ atom_7\}$
then $atom_1 \sim atom_6$ or $atom_7$
$atom_2 \sim atom_6$ or $atom_7$.

It forbids matches across partitions, such as

$atom_1 \sim atom_6$

$atom_2 \sim atom_8$

$atom_3 \sim atom_7$, etc.

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;ana]. Typical testfn's 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.

(2) Execute tempsift[set_1;set_2;$testfn_1$;ana], where $testfn_1$[x;y] 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_1;set_2;$testfn_2$;ana], where $testfn_2$[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 set_1 and set_2 are empty, go to 9.
If terminal default condition holds, go to 8.

(6) Execute tempsift[set_1;set_2;$testfn_3$;ana], where $testfn_3$[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 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_1 and set_2.

(9) STOP.

To illustrate the preceding algorithm with a simple example, let

set_1 = {intersection[x;y;z], abeliangroup[x;*]}

set_2 = {intersection[u;v;w], commutativering[u;*,+]}.

At Step 2 we associate:

intersection[x;y;z] ~ intersection[u;v;w].

Then, since we satisfy the terminal default condition, we associate:

abelian[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, set_1 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 that 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_1;set_2ana]: =

(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 $x \in y$.
If pred[x] is on analogy ana find the partition $z \in set_2$ 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 set_1 and set_2 are empty, go to 8.

(4) For each partition $y \in set_1$, select the first atom x. Find a partition $z \in set_2$ such that the type of predicates in z equals type[x]. If there is only one such $z \in set_2$, execute MULTIMATCH1[y;z;ana].

(5) If the terminal default condition holds, go to 7.
if set_1 and set_2 are empty, go to 8.

(6) If set_1 or set_2 is still not exhausted, print an error message and halt.

(7) Apply MULTIMATCH1 to the remaining partitions in set_1 and set_2.

(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:

T_3' $\forall$ $(g,m,x,*_1)$ group$[g;*_1]$ $\wedge$

propernormal$[m;g;*_1]$ $\wedge$ factorstructure$[x;g;m]$

$\wedge$ simplegroup$[x;*_1]$ $\rightarrow$ maximalgroup$[m;g;*_1]$

T_4' $\forall(r;n;y;*_2;+_2)$ ring$[r;*_2;+_2]$ $\wedge$

properideal$[n;r;*_2;+_2]$ $\wedge$ factorstructure$[y;r;n]$

$\wedge$ simplering$[y;*_2;+_2]$ $\rightarrow$ maximalring$[n;r;*_2;+_2]$.

First, ZORBA-I associates:

maximalgroup $\sim$ maximalring

$m \sim n$

$g \sim r$

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

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

propernormal $\sim$ properideal

factorstructure $\sim$ factorstructure

$x \sim y$.

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:

$$g \sim r$$

$$x \sim y .$$

We can assume that if some variable appears in only one atom in 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:

$$\text{group}[g;*_1] \sim \text{ring}[r;*_2;+_2] .$$

A similar argument based upon

$$x \sim y$$

leads us to deduce:

$$\text{simplegroup}[x;*_1] \sim \text{simplering}[y;*_2;{}_2]$$

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:

$g \sim r$

$x \sim y$.

Multimatch[partition$_1$;partition$_2$;ana]: =

(1) Set $\underline{\ell_1}$ to a list of variables that appear in only one atom of $\underline{\text{partition}_1}$.

(2) Set $\underline{\ell_2}$ to similar list computed on $\underline{\text{partition}_2}$.

(3) Set anaprs = $\{x' \sim y' \mid x' \in \ell_1,\ y' \in \ell_2$ 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 $x' \sim y'$ 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_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 and is described in the next two chapters.

V AN ELEMENTARY DESCRIPTION OF EXTENDER

A. Introduction

In the last chapter 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 first 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 continue into the next chapter for a more complete exposition.

The rationale of EXTENDER depends on 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 α_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.

B. The Analogs of Clause Descriptions

Each clause has a unique description, descr[c], which has been introduced in Chapter III. We will denote the analog of descr[c] by some analogy α_j as α_j[descr[c]]. α_j[descr[c]] is equal to a copy of descr[c] in which every predicate that appears in α_j is replaced by its analogous predicate. Predicates that are absent from α_j are left untouched. For example, suppose we have a trivial α_1:

α_1: abelian ~ commutativering

c_7: ¬abelian[x;*] ∨ group[x;*].

d_7: neg[abelian], pos[group]. = descr[c_7]

α_1[d_7] = neg[commutativering], pos[group].

Suppose we are seeking to extend α_1 by finding the analog of c_7. It is quite unlikely that we will find a clause that satisfies this description, (α_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 neg[commutativering], we would be sure to include c_8 and c_9, which at least include c_8, the clause we desire:

c_8: ¬commutativering[x;*;+] ∨ ring[x;*;+]

c_9: ¬commutativering[x;*;+] ∨ commutative[*;x] .

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

(1) Make a description d for an axiom clause c, descr[c].

(2) Create an analog description α_j[descr[c]] for the current analogy, α_j.

(3) Delete from α_j[descr[c]] any feature that contains a predicate that does not appear in α_j. Denote this restriction of α_j[descr[c]] to α_j by $\overline{\alpha_j[\text{descr}[c]]}$.

(4) Search the data base for clauses that satisfy $\overline{\alpha_j[\text{descr}[c]]}$.

In our example, $\alpha[\text{descr}[c_7]] = \alpha_1[d_7]]$ = neg[commutativering] α_j[descr[c]] is a "restriction of the analog of the description of c to analogy α_j^p." Since this phrase is quite cumbersome, we will simply call it a "restricted description" and implicity understand its dependence on α_j^p.

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

(1) An ordered sequence of partial analogies α_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 proof[T].

(4) A restriction of our descriptions relative to an analogy α_j, by including only those features with predicates that appear in α_j.

C. Mapping Descriptions

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 a_j is derived from its antecedent a_{j-1} by adding:

(1) An association of one clause $ax_k \in$ 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 a_1 is the initial analogy generated by INITIAL-MAP applied to the pair of theorems T_1 - T_2, its predicate map is

abelian ~ commutativering
intersection ~ intersection.

Suppose we know that $c_7 \sim c_8$. We would like to extend a_1 to a_2 by adding:

(1) $c_7 \sim c_8$

(2) abelian ~ commutativering
group ~ ring.

To motivate the structure of MAPDESCR, let us design a version of it that would enable us to extend a_1 to a_2 in this example. MAPDESCR is charged with mapping neg[abelian], pos[group] (d_7) with neg[commutativering], pos[ring] when it knows that:

a_1: abelian ~ commutativering
intersection ~ intersection.

First, we can eliminate neg[abelian] from d_7 and neg[commutativering] from d_8 on the basis of a_1, which associates "abelian" and "commutativering."

a_1[neg[abelian]] = neg[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 ~ ring

to a_2.

Now we can write a version of MAPDESCR that accepts as arguments two clause descriptions and an analogy a_j:

mapdescr[$\underline{\text{descr}}_1$;$\underline{\text{descr}}_2$;$a_j$]:=

(1) $\forall x$ $x \in \underline{\text{descr}}_1$ s.t. $a_j[x] \in \underline{\text{descr}}_2$, delete x from $\underline{\text{descr}}_1$ and $a_j[x]$ from $\underline{\text{descr}}_2$. Thus, we exclude all those features we know about from a_j.

(2) $\forall x$ $x \in \underline{\text{descr}}_1$ and $x \in \underline{\text{descr}}_2$, map the predicate that appears in x into itself and delete x from $\underline{\text{descr}}_1$ and $\underline{\text{descr}}_2$.

(3) In the remnants of $\underline{\text{descr}}_1$ and $\underline{\text{descr}}_2$:

(a) If there are unique elements of $\underline{\text{descr}}_1$ and $\underline{\text{descr}}_2$ 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 $\underline{\text{descr}}_1$ and $\underline{\text{descr}}_2$ 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 a_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 on predicate types, occurs rarely in this domain (abstract algebra).

When MAPDESCR returns a list of predicate pairs that result from mapping the description of a clause c_1($\underline{descr}_1$, above) with the description of a clause c_2($\underline{descr}_2$, above) according to analogy a_j, it creates a new analogy a_{j+1}. a_{j+1} is the same as a_j except that:

(1) Its predicate map is the union of the one returned by MAPDESCR and the one appearing on a_j.

(2) Its clause mapping is the union of the one appearing on a_j and $c_1 \sim c_2$.

Thus, when EXTENDER is attempting to extend a_j, it creates a new analogy a_{j+1}, a_{j+2}, etc. for <u>each</u> clause pair it maps when those clauses were selected on the basis of information in a_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 predicate map.

D. <u>The Candidate Image Set</u>

After I explicate one additional idea I can describe a simple version of EXTENDER. When EXTENDER is extending a_j it is searching the large data base for some clause that is the analog of

an axiom $c_k \in$ SOME. Now we could search for the set of clauses that satisfy $\alpha_j[\text{descr}[c_k]]$, but we will run into the difficulty described earlier in this section. Thus we search for clauses that satisfy $\alpha_j[\text{descr}[c_k]]$. If α_j^p contains the correct analog for each predicate that appears on it, then the set of clauses C that satisfy $\alpha_j[\text{descr}[c_k]]$ is guaranteed to contain the desired analog of c_k ("image" of c_k). We 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 $\alpha_j \rightarrow \alpha_{j+1}$ by associating

$$c_k \sim c' .$$

When the set of clauses that satisfies a restricted description contains one member, we are guaranteed that it is the image clause we seek if α_j^p does not contain any erroneous associations. Now, if C is empty, we have reason to suspect the correctness of α_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 α_j^p is correct, we know that our desired image is in C. If we have a clause c with description descr[c] and some analogy α_j that contains only one of the predicates in c , then $\alpha_j[\text{descr}[c]]$ will have but one feature and many clauses will satisfy it. If some later analogy α_k $(\alpha_j^p \subseteq \alpha_k^p)$ includes another predicate from c in addition to the one on α_j , then $\alpha_k[\text{descr}[c]]$ will have two features and will be satisfied by fewer clauses than $\alpha_j[\text{descr}[c]]$. Thus, as sequences 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 shortdescr[α_j] . SHORTDESCR is a dependent on α_j in three ways:

(1) It searches for the analogs of clauses that appear on SOME (which is different for each α_j).

(2) It generates descriptions that include only the predicates that appear explicitly in a_j.

(3) It uses the predicate map a_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 that satisfy the analog of its restricted description. Each axiom is guaranteed to have its analog under a_j in its associated "candidate image set." If we find no candidates at all, for any $ax_k \in$ SOME, then we know that a_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 a_j.

E. Simple Versions of EXTENDER

Let us consider a primitive version of EXTENDER, EXTEND1, which exploits these few ideas.

Extend1[a_1;AXSET]:=

(1) Let analist = (a_1), the set of active analogies.

(2) If a_j is complete, STOP.

(3) Partition AXSET into {ALL, SOME, NONE} relative to a_j.

(4) Set imlist to shortdescr[a_j].
If imlist = ϕ, mark a_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 = ϕ, 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 α_j to the next element on analist. Go to 2.

The success of EXTEND1 is highly dependent on the clauses in the data base. If there are few clauses, then it is likely that some $ax_k \in$ 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 proof$[T_A]$, it becomes more likely that SHORTDESCR will generate several images for every $ax_k \in$ SOME in some iteration. At this point it will fail to extend α_j and miss the analogy altogether. To remedy this situation, we need a way of dealing with cases in which SHORTDESCR returns several candidate images for each $ax_k \in$ 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 liklihood of being analogous to the $ax_k \in$ 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 α_2 that includes

intersection $\sim$ intersection

subgroup $\sim$ subring

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]

$\overline{\alpha_2[d_{10}]}$ = pos[subring].

Suppose our data base contains two clauses c_{11} and c_{12} that satisfy $\overline{\alpha_2[d_{10}]}$:

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] .

We can compare c_{11} and c_{12} by comparing d_{11} and d_{12} with d_{10} (relative to α_2). We want a partial ordering of a set of descriptions relative to a target description and a particular analogy — e.g., a $\varphi_d[d_1;d_2;d\ \alpha_j]$ — that orders description d_1 with respect to d_2. A simple φ_d can be developed as follows:

Let:

$$d_1' = d_1 - \overline{\alpha_j[d]}$$
$$d_2' = d_2 - \overline{\alpha_j[d]}$$
$$d' = d - \overline{\alpha_j[d]} .$$

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

d'_{10} = neg[group], neg[subset]

d'_{11} = neg[ideal]

d'_{12} = neg[ring],neg[subset] .

Clearly, d_{12}' is closer to d_{10}' than d_{11}', so we select d_{12}', our closes description, and c_{12} as the image of c_{10} under α_2. After MAPDESCR maps $c_{10} \sim c_{12}$ it will add:

group $\sim$ ring

subset $\sim$ subset

to a_2 to create an a_3:

a_3: intersection $\sim$ intersection

subgroup $\sim$ subgroup

group $\sim$ ring

subset $\sim$ subset .

A more sophisticated φ_d can look at the semantic types of predicates 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 candidate images with a restricted description under a_j. Exploiting this notion, we can write a more powerful EXTENDER called EXTEND2.

Extend2[a_1; AXSET]: =

(1) Let analist = (a_1 ... a_j), the list of active analogies. Start with analist = (a_1).

(2) If a_j is complete, STOP.

(3) Partition AXSET into {ALL, SOME, NONE} relative to a_j.

(4) Set imlist to shortdescr[a_j].
If imlist = $\emptyset$, mark a_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 analist. 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 analist is empty, STOP.
Otherwise, set α_j 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 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 2 contains a brief summary of ZORBA-I's behavior when it is applied to five $T - T_A$ pairs drawn from abstract algebra. The number of partial analogies generated includes α_1 generated by INITIAL-MAP.

Table 2

SUMMARY OF ZORBA-I PERFORMANCE

Theorem Pairs	Number of Predicates in Theorem Proof	Number of Predicates Mapped by I-MAP	Number of Axioms in Proof [T]	Number of Analog Axioms Found by EXTENDER	Number of Partial Analogies by ZORBA-I
T_1 - T_2	9	3	13	15	5
T_3 - T_4	12	5	13	17	7
T_5 - T_6	8	3	21	23	5
T_7 - T_8	5	4	6	7	2
T_9 - T_{10}	7	2	12	16	6

T_1 : The intersection to two abelian groups is an abelian subgroup of the parent group.

T_2 : The intersection of two commutative rings is a commutative subring of the parent rings.

T_3 : A factor group G/H is simple if H is a maximal normal subgroup of G.

T_4 : A quotient ring A/C is simple if C is a maximal ideal in A.

T_5 : The intersection of two normal groups is a normal group.

T_6 : The intersection of two ideals is an ideal.

T_7 : The homomorphic image of a subgroup is a subgroup.

T_8 : The homomorphic image of a subring is a subring.

T_9 : The homomorphic image of an abelian group is an abelian group.

T_{10}: The homomorphic image of a commutative ring is a commutative ring.

VI EXPERIMENTS WITH ZORBA-I

A. Introduction

Our previous discussions have been rather abstract and have drawn upon various examples in a piecemeal fashion. Now we are ready to explore the behavior of ZORBA-I when it is applied to a full-scale problem. In this exposition, descriptions of the algorithms have preceded any experimental results. This ordering is pedagogically motivated, to allow briefer explanations to accompany the experiments that are reported here. Also, this order parallels the history of ZORBA-I's development. These algorithms were first conceived during the Winter of 1969 and briefly reported at the Machine Intelligence Workshop held at Stanford University in February of that year. They were favorably received, but required implementation and experimental validation to test their value. At that time several key ideas were visionary leaps. Attempting to reduce the size of a data base used by a theorem prover by exploiting an analogy was well conceived (on paper) at that time. A simple form of EXTENDER involving clause descriptions and a sequence of partial analogies were integral to the conception. All of these ideas were developed in a testable form. But there were no guarantees to their validity or value. For example, in the earliest conception, clause descriptions were static through EXTENDER's search. There were no guarantees that different descriptions might not be needed at different stages of search. It turned out that both approaches were needed. A static description, descr[c], is computed for a clause. At each stage EXTENDER uses a select subset of this description, based on a_j^p to compute a restricted description, to search for analogous clauses. The notion of a restricted description, as well as several refinements of EXTEND2 that are developed in this chapter, were conceived after a crude version of ZORBA-I was implemented. EXTENDER was developed in an interactive time-sharing environment (using PDP-10 LISP). It is unlikely that the program

would have progressed very far with a paper and pencil approach only. A data base of 239 clauses dealing with abstract algebra, called ALGBASE (Appendix B) was created to provide a sizeable set of axioms. No existing theorem prover could even attempt to prove any of the theorems used in these experiments without trimming the data base substantially. On one hand, the experience gained and the resultant successes with this large data base were invaluable to developing ZORBA-I. On the other hand, the massive size of the data base made hand simulations infeasible. Even to simply decide which clauses satisfy α_j[descr[c]] for a particular analogy α_j and clause c, it is helpful to have a computer to quickly search the data base.

At this stage of discussion, we experience a certain creative tension. We have a set of fruitful, but untested ideas. Will they work? I labored with ZORBA-I under this tension for over a year and found the successes that I am presenting here.

ZORBA-I was developed by structuring it to run on two problem pairs, T_1-T_2 and T_3-T_4 (Table 2). Later, it was run on the remaining problem pair (Table 2) and successfully created the appropriate analogies without difficulty. In the course of its development, EXTENDER underwent several changes. Each change was accompanied by a new insight into the process of analogy generation. These insights will be presented in this chapter along with the algorithms that embody them. Prior to examining ZORBA-I's behavior in greater detail, I want to introduce a representation that will simplify our understanding of ZORBA-I's operation.

B. Analogy Space

At the highest level, we can look at ZORBA-I's behavior in terms of the partial analogies that it generates. Figure 10 portrays a simple space containing seven (partial) analogies.

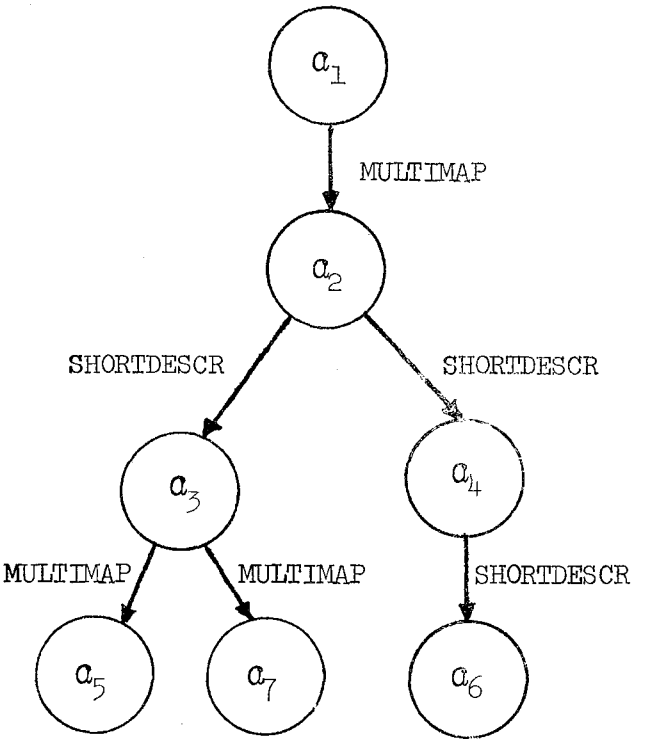


FIGURE 10. A SAMPLE ANALOGY SPACE

The arrows between the nodes that represent partial analogies are labeled with SHORTDESCR and MULTIMAP, which were described in the last chapter. Each of these is a search procedure for finding a clause from the data base (here ALGBASE) that is likely to be analogous to a clause in AXSET. The association of the clauses is used to extend a partial analogy a_j to a_{j+1}. SHORTDESCR and MULTIMAP can also be viewed as operators that extend (transform) one partial analogy into a more complete partial analogy. A great deal of computation is hidden below this level of description, but is determined by it. For example, in Figure 10, a_6 is extended from a_4 by SHORTDESCR. We know that a_4 induces a unique partition of AXSET into all$[a_4^p]$, some$[a_4^p]$, and none$[a_4^p]$ (Chapter III). Since a_6 is the only extension of a_4, we presume that SHORTDESCR found only one $ax_k \in$ SOME with but one image, c_j. We know that descr$[ax_k]$ is matched with descr$[c_j]$ to create the new a_6^p.

A great deal of ZORBA-I's behavior can be concisely represented by the analogy space representation of Figure 10. It presents ZORBA's decision procedures explicitly by showing which partial analogies are directly related, and, implicitly which operators failed. For example, we know that shortdescr[α_j] failed if MULTI-MAP is used to extend α_j. Here each α_j is the abstract set of maps defined in Chapter III. As our discussion unfolds, a partial analogy will become more concrete as it is elaborated through various examples.

C. ZORBA-I in Action

We are just about ready to watch ZORBA-I generate an analogy. Let's consider the theorem pair T_1 - T_2.

T_1: The intersection of two abelian groups is an abelian subgroup of the parent groups.

T_2: The intersection of two commutative rings is a commutative subring of the parent rings.

Suppose a theorem-prover (QA3) has proved T_1 and wants to prove T_2. Furthermore, suppose it knows that T_1 and T_2 are analogous.

ZORBA-I is given the following information:

(1)* T_1': $\forall$(a b c *) abelian*[a;x] $\vee$ abelian[b;*] $\vee$ intersection[c;a;b] $\rightarrow$ absubgroup[c;a;*]

(2) T_2': $\forall$(x y z * +) commring[x;*;+] $\vee$ commring[y;*;+] $\vee$ intersection[z;x;y] $\rightarrow$ commsubring[z;x;*;+].

---

*See Appendix A for the definitions of these predicate symbols

(3) The (resolution) proof tree of theorem T_1 (Table 3) from which it extracts AXSET, the set of axioms used in the proof (Table 4).

These three items are problem-dependent. In addition, ZORBA-I can refer to the semantic template (Appendix A) of any predicate, and it can access a large data base. In these experiments ALGBASE (Appendix B) is the data base used.

When ZORBA-I starts on the problem just presented, it first executes initial-map[T_1'; T_2'] to find analogs for the predicates intersection, abelian, and absubgroup with members of the set {intersection, commring, commsubring}. This process was described in some detail in Chapter III. INITIAL-MAP outputs a single analogy α_1:

α_1^p: intersection ~ intersection
abelian ~ commring
absubgroup ~ commsubring .

Next, EXTENDER is applied to α_1 and it attempts to find an analog for each axiom in AXSET. In the process, it generates a sequence of several analogies (Table 5) which terminates in a complete analogy (Table 6).

Table 3

RESOLUTION PROOF OF THEOREM T_1'

1	*abelian[a1, star4]	negation of theorem
2	group[x, star]-abelian[x, star]	axiom
3	group[a1,star 4]	from 1,2
4	abelian[b1,star4]	negation of theorem

continued

*See Appendix A for definition of predicate symbols and Appendix B for description of clause format.

Table 3
Continued

5	group[b1,star4]	negation of theorem
6	intersection[x1,a1,b1]	from 4,2
7	group[k,star] - intersection[k,g,h] -group[g,star] -group[h,star]	axiom
8	group[x1,star] -group[a1,star] -group[b1,star]	from 6,7
9	group[x1,star4] -group[a1,star4]	from 5,8
10	group[x1,star4]	from 3,9
11	subset[x,y] -intersection[x,y,z]	axiom
12	subset[x1,a1]	from 6,11
13	subgroup[h,g,star] -subset[h,g] -group[g,star] -group[h,star]	axiom
14	subgroup[x1,a1,star] -group[a1,star] -group[x1,star]	from 12,13
15	subgroup[x1,a1,star4] -group[x1,star4]	from 3,14
16	subgroup[x1,a1,star4]	from 10,15
17	-absubgroup[x1,a1,star4]	negation of theorem
18	absubgroup[x,y,star] -abelian[x,star] -subgroup[x,y,star]	axiom
19	-abelian[x1,star4] -subgroup[x1,a1,star4]	from 17,18
20	-abelian[x1,star4]	from 16,19
21	abelian[g,star] -group[g,star] -commutative[star,g]	axiom
22	abelian[x1,star4] -commutative[star4,x1]	from 10,21
23	-commutative[star4,x1]	from 20,22
24	commutative[star,s] in[sk4[star,s1,s]	axiom
25	in[sk4[star4,x1]	from 23,24
26	in[a,z] -in[a,x] -intersection[x,y,z]	axiom
27	in[a,b1] -in[a,x1]	from 6,26
28	in[sk4[star4,x1]b1]	from 25,27
29	commutative[star,g] -abelian[g,star]	axiom
30	commutative[star4,b1]	from 4,29

Table 3
Continued

31	commutative[star,s] in[sk3[star,s]s]	axiom
32	in[sk3[star4,x1],x1]	from 23,31
33	in[sk3[star4,x1],b1]	from 32,27
34	commutative[star,s] -times[star,sk4[star,s1,sk3[star,s],c]	axiom
35	-times[star4,sk4[star4,x1],sk3[star4,x1],c]	from 23,34
36	commutative[star,s] times[star,sk3[star,s],sk4[star,s],sk5[star,s]]	axiom
37	times[star4,sk3[star4,x1],sk4[star4,x1],sk5[star4,x1]]	from 23
38	times[star,b,a,c] -in[a,s] -in[b,s] -times[star,a,b,c] -commutative[star,s]	axiom
39	times[star4,sk4[star4,x1],sk3[star4,x1],c] -in[sk3[star4,x1]s] -in[sk4[star4,x1],s] -commutative[star4,s]	from 37,38
40	-in[sk3[star4,x1],s] -in[sk4[star4,x1],s] -commutative[star4,s]	from 35,39
41	-in[sk4[star4,x1],b1]-commutative[star4,b1]	from 33,40
42	-in[sk4[star4,x1],b1]	from 30,41
43	contradiction	from 28,42

Table 4

AXSET FOR ABSGPT (THEOREM T_1)

ABSGPT-1 group[x;star] $\vee \neg$abelian x;star]
neg[abelian] pos[group]

ABSGPT-2 group[k;star] $\vee \neg$intersection[k;g;h] $\vee \neg$group[g;star] $\vee \neg$group[h;star]
neg[intersection] impcond[group]

ABSGPT-3 subset[x;y] $\vee \neg$intersection[x;y;z]
neg[intersection] pos[subset]

ABSGPT-4 subgroup[h;g;star] $\vee \neg$subset[h;g] $\vee \neg$group[g;star] $\vee \neg$group[h;star]
neg[group] neg[subset] pos[subgroup]

ABSGPT-5 absubgroup[x;y;star] $\vee \neg$abelian[x;star] $\vee \neg$subgroup[x;y;star]
neg[subgroup] neg[abelian] pos[absubgroup]

ABSGPT-6 abelian[g;star] $\vee \neg$group[g;star] $\vee \neg$commutative[star;g]
neg[commutative] neg[group pos[abelian]

ABSGPT-7 commutative[star;s] $\vee$ in[sk4(star;s),s]
pos[in] pos[commutative]

ABSGPT-8 in[a;z] $\vee \neg$in[a;x] $\vee \neg$intersection[x;y;z]
neg[intersection] impcond[in]

ABSGPT-9 commutative[star;g] $\vee \neg$abelian[g;star]
neg[abelian] pos[commutative]

ABSGPT-10 commutative[star;s] $\vee$ in[sk3(star;s),s]
pos[in] pos[commutative]

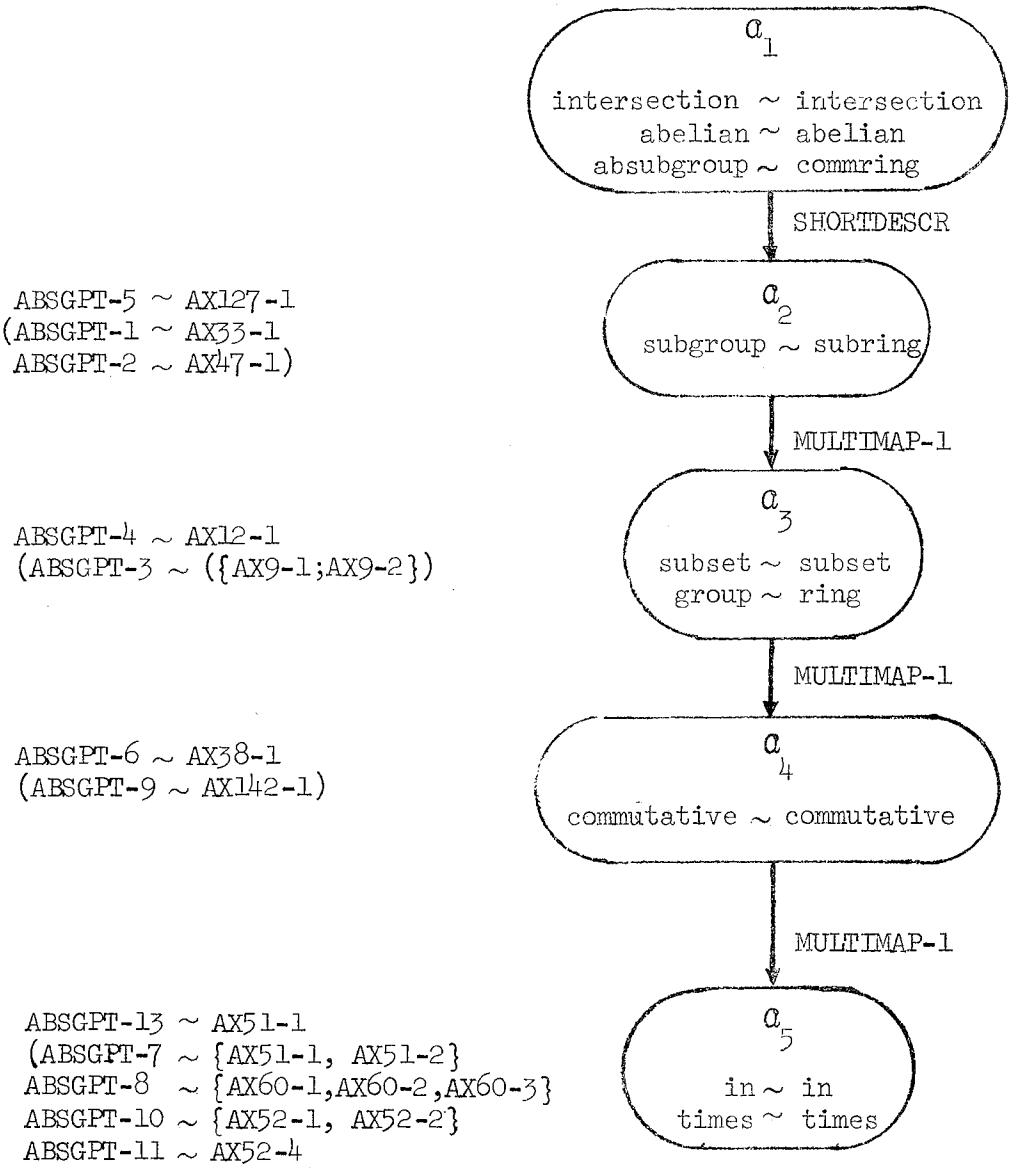
ABSGPT-11 commutative[star;s] $\vee \neg$times[star;sk4(star,s), sk3(star;s),c]
pos[in] pos[commutative]

ABSGPT-12 commutative[star;s] $\vee$ times[star;sk3(star,s),sk4(star,s), sk5(star;s)]
pos[times] pos[commutative]

ABSGPT-13 times[star;b;a;c] $\vee \neg$in[a;s] $\vee \neg$in[b;s] $\vee \neg$times[star;a;b;c] $\vee \neg$commutative[star;s]
neg[commutative] neg[in] impcond[times]

Table 5

ANALOGY SEARCH SPACE FOR
$T_1' - T_2'$



ABSGPT-5 ~ AX127-1
(ABSGPT-1 ~ AX33-1
ABSGPT-2 ~ AX47-1)

ABSGPT-4 ~ AX12-1
(ABSGPT-3 ~ ({AX9-1;AX9-2})

ABSGPT-6 ~ AX38-1
(ABSGPT-9 ~ AX142-1)

ABSGPT-13 ~ AX51-1
(ABSGPT-7 ~ {AX51-1, AX51-2}
ABSGPT-8 ~ {AX60-1,AX60-2,AX60-3}
ABSGPT-10 ~ {AX52-1, AX52-2}
ABSGPT-11 ~ AX52-4
ABSGPT-12 ~ AX52-3)

Table 6

COMPLETE ANALOGY (α_5) FOR ABSGPT

$\alpha^p = \alpha_5^p$:

intersection ~ intersection
abelian ~ commutative
absubgroup ~ commsubring
subgroup ~ subring
subset ~ subset
group ~ ring
commutative ~ commutative
in ~ in
times ~ times

$\alpha^c = \alpha_5^c$:

ABSGPT-1* ~ AX33-1
ABSGPT-2 ~ AX47-1
ABSGPT-3 ~ {AX9-1; AX9-2}
ABSGPT-4 ~ AX12-1
ABSGPT-5 ~ AX127-1
ABSGPT-6 ~ AX38-1
ABSGPT-7 ~ {AX51-1; AX51-2}
ABSGPT-8 ~ {AX60-1; AX60-2; AX60-3}
ABSGPT-9 ~ AX142-1
ABSGPT-10 ~ {AX52-1; AX52-2}
ABSGPT-11 ~ AX52-4
ABSGPT-12 ~ AX52-3
ABSGPT-13 ~ AX51-1

*The ABSGPT-j axioms from AXSET appear in Table 4 and the AXn-k axioms from ALGBASE appear in Table 7.

Table 7

CLAUSES FROM ALGBASE (Appendix B)

ANALOGOUS TO AXSET FOR ABSGP (Table 4)

AX9-1 subset[x,y] ∨ ¬intersection[x,y,z]
neg[intersection], pos[subset]

AX9-2 subset[x,z] ∨ ¬intersection[x,y,z]
neg[intersection], pos[subset]

AX12-1 subring[x,a,star,plus] ∨ ¬ring[a,star,plus]
¬subset[x,a] ∨ ¬ring[x,star,plus]
neg[subset], neg[ring], pos[subring]

AX33-1 ring[r,star,plus] ∨ ¬commring[r,star, plus]
neg[commring], pos[ring]

AX38-1 commring,star,plus] ∨ ¬ringer,star,plus]
¬commutative[star;r]
neg[commutative] neg[ring] pos[commring]

AX47-1 ring[x,star,plus] ∨ ¬ring[a,star,plus]
¬ring[b,star,plus] ∨ ¬intersection[x,a,b]
neg[intersection] impcond[ring]

AX51-1 times[star,b,a,sk73[b,a,star,s]] ∨ ¬in[a,s] ∨
¬in[b,s] ∨ ¬times[star,a,b,sk73[b,a,star,s]] ∨
¬commutative[star,s]
neg[commutative] neg[in] impcond[times]

AX52-1 comm[star,s] ∨ in[sk75[star,s],s]
pos[in] pos[commutative]

AX52-2 commutative[star,s] ∨ in[sk76[star,s],s]
pos[in] pos[commutative]

AX52-3 commutative[star,s]∨times[star,sk75[star,s],sk76[star,s]∨
[star,s]]
pos[times] pos[commutative]

AX52-4 commutative[star,s]∨¬times[star,sk76[star,s],sk75[star,s],c]
neg[times] pos[commutative]

AX60-1 in[x,c]∨ ¬in[x,a] ∨
¬in[x,b]∨¬intersection[c,a,b]
neg[intersection] impcond[in]

AX60-2 in[x,b] ∨¬in[x,c] ∨
¬intersection[c,a,b]
neg[intersection] impcond[in]

AX60-3 in[x,a]∨¬in[x,c] ∨
¬intersection[c,a,b]
neg[intersection] impcond[in]

AX127-1 commsubring[x,y,star,plus] ∨¬commring[x,star,plus]
¬subring[x,y,star,plus]
neg[subring] neg[commring] pos[commsubring]

AX142-1 commutative[star,r]∨¬commring[r,star,plus]
neg[commring] pos[commutative]

AX142-2 ring[r,star,plus]∨¬commring[r,star,plus]
neg[commring] pos[ring]

AX143-1 commring[r,star,plus] ∨¬commutative[star,r]
¬ring[r,star,plus]
neg[ring] neg[commutative] pos[commring]

Table 5 presents an elaborated version of the analogy space search that was introduced in Figure 10. Each partial analogy, a_j, is depicted with the incremental information it adds to a_{j-1}. For example, a_3 is shown associating the predicates commutative ~ commutative and the clauses ABSGPT-6 ~ AX38-1. This means that

$$a_3^p = a_2^p \cup \text{commutative} \sim \text{commutative}.$$

a_3 was created by associating the axiom ABSGPT-6 from AXSET (Table 4) with the clause AX38-1 from ALGBASE (Table 7).

The clause associations that appear in parentheses next to the node representing a_3 contain an association between ABSGPT-3 (from AXSET) and {AX9-1, AX9-2} (from ALGBASE). Both of these clauses from ALGBASE satisfy a_3[descr[ABSGPT-3]], and they are associated after a_3 is created:*

ABSGPT-3: ¬intersection[x;y;z] ∨ subset[x;y].

a_3^p contains the analogs of both predicates that appear in ABSGPT-3. Consequently, we can immediately seek its analog after creating a_3. Certainly, no descendant partial analogy a_j ($j > 3$) will add any new information to aid us in finding the analog of this axiom. In addition, we can see that a_3 is created from a_2 by the application of an operator MULTIMAP1, which is a close relative of MULTIMAP and will be described below. ZORBA-I generates a complete analogy a_5 as the fifth term in a sequence of fertile partial analogies.

Now, this description of ZORBA-I is quite informative to a person intimate with the algorithms employed. Substantial computation that is integral to ZORBA-I is unrepresented in Table 5 and needs to

---

*For simplicity we will refer to {AX9-1, AX9-2} as the image of ABSGPT-3, since these two clauses are description-equivalent. We will speak of (candidate) images of an axiom when these are two or more sets of clauses that are not description-equivalent.

be elaborated for an uninitiated reader. For example, SHORTDESCR searched for the images of many clauses (SOME) when extending α_1 and found that one clause (ABSGPT-5) had but one image. The results of the other searches are omitted in the "analogy space protocol" represented in Table 5. MULTIMAP1 is used to select a likely image clause for an axiom that has more than candidate image based on heuristic ordering function φ_d (Chapter V).

Table 8 fills in a sample of this detail in portraying a little of the information flow through SHORTDESCR. We can see that shortdescr[α_1] finds at least one candidate image for each clause in SOME. Only one axiom (ABSGPT-5 from Table 4) has only one candidate image. It is passed onto MAPDESCR to create α_2^p from α_1^p and the predicate associations that arise from mapping the description of ABSGPT-5 with the description of AX127-1.

A new analogy α_2 is created and is checked to see whether $\alpha_2^p \subseteq \alpha_j^p$ for any α_j. Here, the only analogy generated so far is $\alpha_2^p \not\subseteq \alpha_2^p$. (In fact, $\alpha_1^p \subset \alpha_2^p$.) Referring to Table 7 we see how α_2 is created from α_1 by SHORTDESCR associating ABSGPT-5 with AX127-1. We can also see that this association adds

subgroup ~ subgroup

to α_1^p and creates a larger α_2^p :

α_2^p : intersection ~ intersection
abelian ~ commring
abeligansubgp ~ commring
subgroup ~ subring .

Finally, α_2 is added to the list of active analogies, and since it is the only unextended analogy, it is extended next.

Table 8

SEGMENT OF PROTOCOL FROM EXTENDER SEARCH

(a)

EXTEND a_1 (generated by INITIAL-MAP)

PARTITION AXSET

ALL = ϕ

SOME = {ABSGPT-1; ABSGPT-2; ABSGPT-3; ABSGPT-5; ABSGPT-8; ABSGPT-9}

NONE = {ABSGPT-4; ABSGPT-7; ABSGPT-10; ABSGPT-11; ABSGPT-12; ABSGPT-13}

APPLY SHORTDESCR TO SOME

ABSGPT-1 has 7 candidate images under a_1^p

ABSGPT-2 has 9 candidate images under a_1^p

ABSGPT-3 has 9 candidate images under a_1^p

ABSGPT-5 has 1 candidate image under a_1^p

ABSGPT-6 has 6 candidate images under a_1^p

ABSGPT-8 has 9 candidate images under a_1^p

ABSGPT-9 has 7 candidate images under a_1^p

Select axioms from AXSET with 1 candidate image

APPLY MAPSESCR to ABSGPT-5 and AX127-1

Create a_2

a_2 is a new partial analogy

Select the next partial analogy to extend: a_2

Table 8
(Concluded)

(b)

EXTEND a_2

PARTITION AXSET

ALL = {ABSGPT-5}

SOME = {ABSGPT-1, ABSGPT-2, ABSGPT-3, ABSGPT-4, ABSGPT-6, ABSGPT-8, ABSGPT-9}

NONE = {ABSGPT-7, ABSGPT-10, ABSGPT-11, ABSGPT-12, ABSGPT-13}

APPLY SHORTDESCR to SOME

SELECT BUGSET SOME

BUTSET = {ABSGPT-4}

ABSGPT-4 has 3 candidate images under a_2

APPLY MULTIMAP1 to ABSGPT-4 and its candidate images:

{AX12-1, AX126-2, AX128-1}

ORDER the candidate image set by φ_d: AX12-1 is the best candidate

APPLY MAPDESCR to ABSGPT-4 and AX12-1

CREATE a_3.

a_3^p is a new partial analogy

SELECT the next partial analogy to extend: a_3

EXTEND a_3

PARTITION AXSET

ALL = {ABSGPT-1, ABSGPT-2, ABSGPT-3, ABSGPT-4, ABSGPT-5}

SOME = {ABSGPT-6, ABSGPT-8, ABSGPT-9}

NONE = {ABSGPT-7, ABSGPT-10, ABSGPT-11, ABSGPT-12, ABSGPT-13}

AXSET is partitioned with respect to a_2^p (Table 8a), and we are ready to execute shortdescr[a_2] in a little detail (Table 8a):

SOME[a_2]: = {ABSGPT-1, ABSGPT-2, ABSGPT-3, ABSGPT-4, ABSGPT-6, ABSGPT-8, ABSGPT-9}

ABSGPT-4: $\neg$group[h,*]] $\vee \neg$group[g;*] $\vee \neg$subset[h;g] $\vee$ subgroup[h;g;*]

ABSGPT-6: $\neg$commutative[*;g] $\vee \neg$group[g;*] $\vee$ abelian[g;*]

---

a_1^p[descr[ABSGPT-6] = a_2^p[descr[ABSGPT-6] = pos[abelian].

Since a_2^p does not add any information to a_1 about additional predicates in ABSGPT-6, the search for any clauses that satisfy its restricted description will be identical in shortdescr[a_2]. In contrast, a_2^p did add the analog of the predicate SUBGROUP which appears in ABSGPT-4:

---

a_1^p[descr[ABSGPT-4]] $\subset a_2^p$[descr[ABSGPT-4] = pos[subgroup].

We expect that a_2 will enhance our ability to search for the analog of ABSGPT-4, but will add nothing to our search for the analog of ABSGPT-6. Shortdescr[a_2] should seek the analog of ABSGPT-4 (and any clauses that it similarly effects) and skip over those clauses that do not contain predicates that a_2^p added to a_1^p. In this case, ABSGPT-4 is the only clause that a_2^p informs us about. (The only other clause in AXSET that references the predicate SUBGROUP was used to create a_2). Often, but not in this problem, the "budding set" contains several members.

Definition: An axiom $ax_k \in$ some[a_j] is a member of budset [a_j] iff ax_k includes some predicate p that is contained (with analog) in $a_j^p - a^p_{j-1}$, where a_{j-1} is the immediate parent of a_j.

By limiting our searches in SHORTDESCR to BUDSET, we eliminate much excess computation. For example, here some[a_2] contains seven axioms, but we will only search for an extension to a_2 with one of them. When the data base D is large, these searches are rather costly, and the restriction of SHORTDESCR to BUDSET is important for computational efficiency. In addition to this pragmatic issue, we have an important theoretical observation. Suppose a clause ax_k in budset[a_j] fails to have an image under a_j. If we can assume that the appropriate analog of ax_k is in D and satisfies a[descr[ax_k]] for the correct analogy a, then we know that a_j^p contains at least one faulty predicate association. Clearly, this improper association is in the set $a_j^p - a^p_{j-1}$, since previously (at a_{j-1}) we had either (1) no search $ax_k \in$ none[a_{j-1}]). The use of BUDSET enables us to localize the error in a faulty a_j when one arises.

Let's return to our discussion of EXTEND-3's behavior with $T_1 - T_2$. Shortdescr[a_2] is invoked to search for the image of budset[a_2] and finds that one axiom (ABSGPT-4) has an image set. It then uses MULTIMAP-1 to order the image set by φ_d (Chapter V) and associates ABSGPT-4 with the most likely candidate clause (AX12-1) from AIGBASE. MAPDESCR is invoked again, and a new partial analogy, a_3. is created. EXTENDER iterates again and continues its process until it finds the complete analogy (Table 5).

By now the reader should have a good grasp of ZORBA-I's inner processes. In order to follow its behavior through the remaining experiments that it performed, I need to describe two operators that extend a partial analogy. First, we need to explore MULTIMAP, which was introduced in the last chapter with EXTENDER2. Then, we can consider a new kind of operation (called CHUNK) which can accrete ("chunk") one "superanalogy" from merging two or more partial analogies.

MULTIMAP was described (Chapter V) as an extension process that allows MAPDESCR to be applied to an axiom $ax_k \in$ SOME, and the most likely of several candidates that are ordered by a likelihood function φ_d. A simple φ_d has already been outlined, and MULTIMAP was described as applying MAPDESCR to each axiom with more than one candidate image. In our preceding discussion we considered a simpler version of MULTIMAP, called MULTIMAP1. This operation attempts to extend a_j only if SHORTDESCR failed and budset$[a_j]$ had but one member with more than one candidate image. These candidates $\{c_j\}$ are ordered by φ_d and MAPDESCR, applied to ax_k and the best clause, c_ℓ, under this ordering. This procedure is adequate for generating an analogy for $T_1 - T_1$ (Table 5). This particular problem is the only one of the difficult four (Table 2) that can be solved with only SHORTDESCR and MULTIMAP1. We need to discuss how ZORBA-I behaves when SHORTDESCR fails and BUDSET contains several members, each with several candidate images, as well as the behavior of ZORBA-I when SHORTDESCR succeeds with a BUDSET with several members. We will discuss the first item next.

D. An Example of MULTIMAP

In our preceding example, each partial analogy could be extended by the application of SHORTDESCR on MULTIMAP1. If shortdescr$[a_j]$ failed, then only one clause (ax_k) in budset had more than one candidate image. a_j could be extended by ordering the candidate image set by φ_d and associating the best candidate with ax_k. This serendipitous arrangement occurs rarely if shortdescr$[a_j]$ fails. Typically, if shortdescr$[a_j]$ fails, several clauses will have more than one candidate image. We are then faced with two decisions:

(1) Which ax_k in BUDSET shall we decide to map with their candidate images?

(2) Which candidate image shall be selected for each clause?

We have already decided the answer to Question 2 by using φ_d to order the candidates. Our answer to Question 1 is really at issue here. We have several choices:

(1) Extend α_j to several distinct partial analogies by pairing <u>each</u> ax_k BUDSET with its best candidate image under φ_d.

(2) Extend α_j to <u>one</u> partial analogy by pairing only one $ax_k \in$ BUDSET with its best candidate image under φ_d.

ZORBA-I chooses the last of these three alternatives for two reasons:

(1) Two extensions of the same analogy often result in redundant searches. (This issue is discussed in Section E of this chapter.)

(2) More than one extension of an analogy at each stage of iteration will enlarge the search space exponentially. Now, we know (Chapter V) that if $\overline{\alpha_j[\text{descr}(ax_k)]}$ yields a candidate image set with more than one member, we can expect that our desired image is in this set if α_j^p is valid. If we extend α_j by mapping only one $ax_k \in$ BUDSET, we will pick up the remaining elements of BUDSET at some other level ($k>j$) of iteration.

We need some $\varphi_c[c_1;c_2]$ that will order the clauses in BUDSET based on their (heuristic) desirability. ZORBA-I uses two criteria in ordering BUDSET (relative to α_j):

(1) If a clause c_1, has more features available (based on $\overline{\alpha_j[\text{descr}[c]]}$ than a clause c_2, we should prefer c_1 to c_2. Each feature available in the restricted description helps us narrow the search for its desired candidate image.

(2) If two clauses c_1 and c_2 are equivalent by the preceding criterion, then we might prefer c_1 to c_2 by a second criterion. φ_d is a heuristic ordering function; it might give us the wrong image for a particular clause. If we add fewer new predicates to α_j^p, we take less risk in α_j than if we add more predicates (and err).

ZORBA-I uses both criteria in their obvious order to create a clause-ordering function φ_c. Since we have a φ_c that is oriented toward providing the clause that will give us additional information at the least risk, MULTIMAP runs φ_c over SOME. Occasionally, φ_c will prefer clauses that do not appear in BUDSET since their restricted description has more features than that of any clause in BUDSET. (The following example will illustrate this point more fully.) Using φ_c, we can write MULTIMAP as follows:

Multimap[α_j] :=

(1) Order budset[α_j] by φ_c described above.

(2) Select the best element ax_k of SOME.

(3) Find the set $\{c_j\}$ of clauses from the data base that satisfy $\overline{\alpha_j[\text{descr}(ax_k)]}$.

(4) Order $\{c_j\}$ by φ_d.

(5) Apply MAPDESCR to ax_k and c_j, the best clause in $\{c_j\}$.

(6) Check to see if mapdescr[ax_k;c_j] $\cup$ α_j^p creates a new α_{j+1}^p . If so, create α_{j+1} and exit.

(7) Set ax_k to the next best element of budset[α_j].

If all have been tried, execute an error. Otherwise, go to (3).

We have just defined MULTIMAP as it is used by ZORBA-I. Both φ_c and φ_d are treated as functional parameters, and we can have a family of MULTIMAP operators with each one having a particular pair of heuristic ordering functions φ_c and φ_d .

We are now ready to consider a particular example, T_5 - T_6, from Table 2. The analogy-space search for this pair of theorems is shown in Table 9. SHORTDESCR and MULTIMAP are invoked alternately to generate a final complete analogy a_5. Since T_5 concerns a property of intersecting normal groups, this problem pair is referred to as INTNOR. The axioms (AXSET) are listed in Table 10 and the reader is referred to the listing of ALGBASE in Appendix B.

To check which axioms are paired with them in Table 9, let's look at an iteration of EXTENDER that invokes MULTIMAP. EXTENDER finds that shortdescr[a_3] fails. It then orders some[a_3] as in Table 11. These clauses fall into three φ_c-equivalent groups. The first group, INTNOR-13...INTNOR-19, could add one predicate to a_3^p and have two features available for a search. A second set composed of {INTNOR-1, INTNOR-2, INTNOR-3, INTNOR-5} also could add one predicate to a_3^p, but have a restricted description with only one feature. The last set is composed of but one axiom, INTNOR-21, which could add three predicates to $a_3{}^p$ if its analog were found. $a_3{}^p$ add subset ~ subset to $a_2{}^p$ and budset[a_3^p] equals {INTNOR-5}.

However, we don't want to search for the image of INTNOR-5 in preference to any of the clauses preferred by φ_c. The clauses of the first set in Table 11 have fewer candidate images than the clauses in the two lower-ranking sets. The last column lists the number of clauses that satisfy $\overline{a_3[\text{descr}[ax_k]]}$ for each ax_k in some[a_3]. If we choose a clause that has only three candidate images in preference to one with nine, we can assume that φ_d will have an easier time in ordering the set. This is not necessarily true, and is purely a heuristic decision. We decide to choose a clause with the fewest candidate images to extend a_3. Now, we don't want to search for the candidate images of every axiom in some[a_3], since these searches are expensive. Thus, we use the φ_c above to order clauses by their <u>likelihood</u> of having a small candidate image set. The criterion is simple: as the number of features in a restricted description of a clause increases, its candidate

Table 9

ANALOGY SEARCH SPACE FOR

T_5 - T_6 ANALOGY

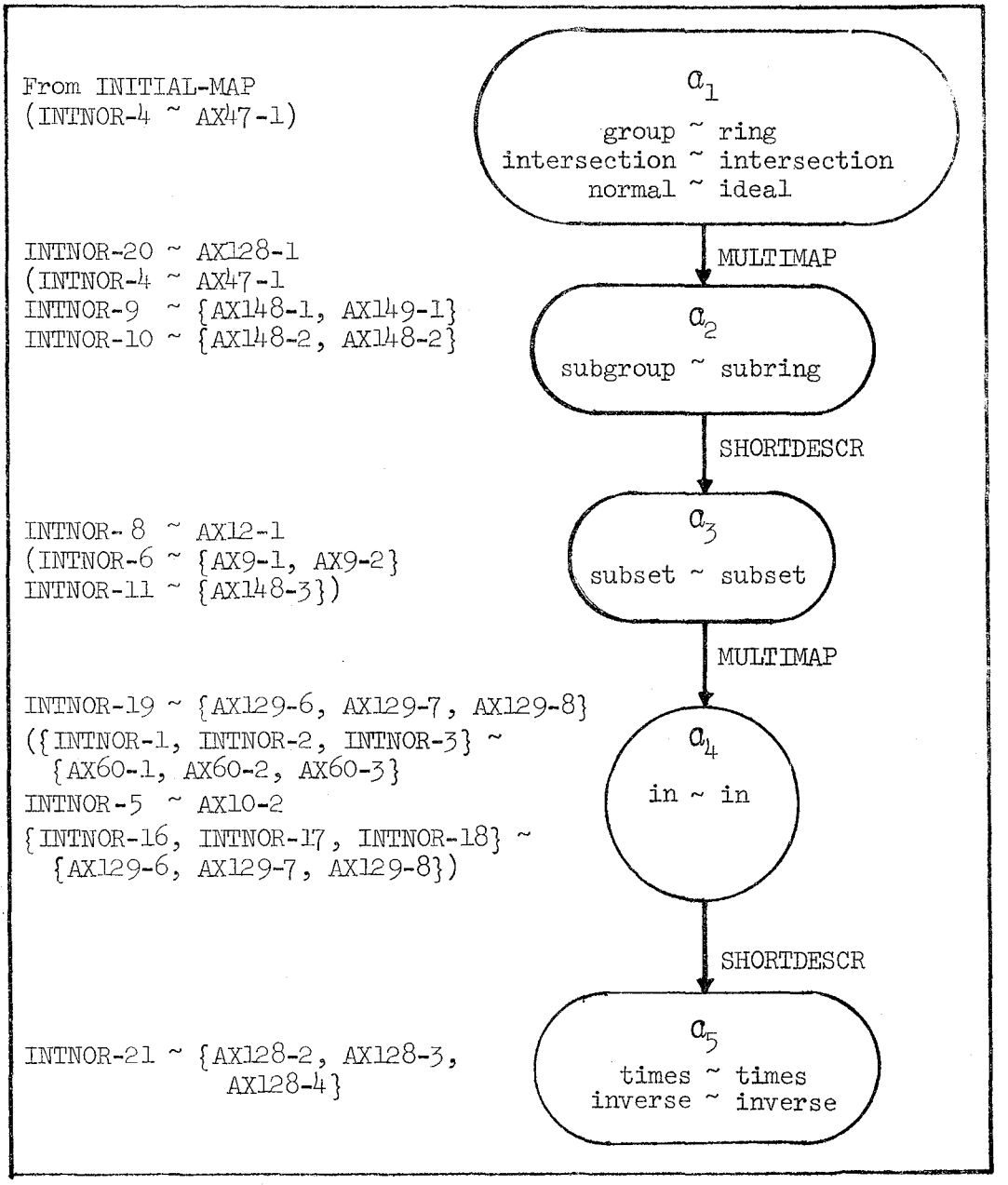


Table 10

AXSET FOR THEOREM T_5

T_5 : The intersection of two normal groups is a normal group

T_5': group[g;*] ∧ normal[a;g;*] ∧ normal[b;g;*] ∧ intersection[c;a;b] → normal[c;g;*]

INTNOR-1 in[x;c] ∨ ¬in[x;a] ∨ ¬in[x;b] ∨ ¬intersection[c;a;b]
neg[intersection], impcond[in]

INTNOR-2 in[x;b] ∨ ¬in[x;c] ∨ ¬ intersection[c;a;b]
neg[intersection], impcond[in]

INTNOR-3 in[x;a] ∨ ¬in[x;c] ∨ ¬intersection[c;a;b]
neg[intersection], impcond[in]

INTNOR-4 group[k;star] ∨ ¬intersection[k;g;h] ∨ ¬group[g;star] ∨ ¬group[h;star]
neg[intersection], impcond[group]

INTNOR-5 in[a,y] ∨ ¬in[a;x] ∨ ¬ sybset[x;y]
neg[subset], impcond[in]

INTNOR-6 subset[x,y] ∨ ¬intersection[x;y;z]
neg[intersection], pos[subset]

INTNOR-7 subset[x;z] ∨ ¬intersection[x;y;z]
neg[intersection], pos[subset]

INTNOR-8 subgroup[h;g;star] ∨ ¬group[h;star] ∨ ¬group[g;star] ∨ ¬ subset[h;g]
neg[subset], neg[group], pos[subgroup]

INTNOR-9 group[h;star] ∨ ¬subgroup[h;g;star]
neg[subgroup], pos[group]

INTNOR-10 group[g;star] ∨ ¬subgroup[h;g;star]
neg[subgroup], pos[group]

INTNOR-11 subset[h;g] ∨ ¬subgroup[h;g;star]
neg[subgroup], pos[subset]

Table 10
(Concluded)

INTNOR-12 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ ¬in[sk_{52}(star;g;h);h]
neg[in] neg[subgroup] pos[normal]

INTNOR-13 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ ¬inverse[star;sk51(star;g;h); sk49[star;g;h)]
pos[inverse neg[subgroup] pos[normal]

INTNOR-14 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ times[star;sk50(star;g;h]; sk51[star;g;h]; sk52[star;g;h)]
pos[times] neg[subgroup] pos[normal]

INTNOR-15 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ times[star; sk49(star;g;h); sk48(star;g;h); sk50(star;g;h)]
pos[times] neg[subgroup] pos[normal]

INTNOR-16 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ in[sk51(star;g;h);g]
pos[in] neg[subgroup] pos[normal]

INTNOR-17 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ in[sk50(star;g;h); G]
pos[in] neg[subgroup] pos[normal]

INTNOR-18 normal[h;g;star ∨ ¬subgroup[h;g;star] ∨ in[sk49(star;g;h); g]
pos[in] neg[subgroup] pos[normal]

INTNOR-19 normal[h;g;star] ∨ ¬subgroup[h;g;star] ∨ in[sk48(star;g;h); H]
pos[in] neg[subgroup] pos[normal]

INTNOR-20 subgroup[h;g;star] ∨ ¬normal[h;g;star]
neg[normal] pos[subgroup]

INTNOR-21 in[u;h] ∨ ¬in[hh;h] ∨ ¬in[gg;g] ∨ ¬in[y;g] ∨ ¬in[m;g] ∨ ¬times[star;gg;hh;y] ∨ ¬times[star;y;m;u] ∨ ¬inverse[star;m;gg] ∨ ¬normal[h;g;star]
neg[normal] neg[inverse] neg[times] impcond[in]

Table 11

SOME[α_3] ORDERED BY φ_c FOR MULTIMAP

Axioms*	The Predicate Axiom Can Add to α_3^p	$\overline{\alpha_3[\text{descr}(c)]}$	Number of Clauses from ALGBASE that Satisfy the Restricted Description
	Set 1		
INTNOR-19	in	neg[subring], pos[ideal]	3
INTNOR-18	in	neg[subring], pos[ideal]	3
INTNOR-17	in	neg[subring], pos[ideal]	3
INTNOR-16	in	neg[subring], pos[ideal]	3
INTNOR-15	times	neg[subring], pos[ideal]	3
INTNOR-14	times	neg[subring], pos[ideal]	3
INTNOR-13	inverse	neg[subring], pos[ideal]	3
INTNOR-12	in	neg[subring], pos[ideal]	3
	Set 2		
INTNOR-5	in	neg[subset]	9
INTNOR-3	in	neg[intersection]	6
INTNOR-2	in	neg[intersection]	6
INTNOR-2	in	neg[intersection]	6
	Set 3		
INTNOR-21	{ in, times, inverse }	neg[ideal]	7

*See Table 10 for axioms corresponding to the names listed here.

image set decreases in size. Computing φ_c is cheap and allows us to choose a low-risk clause in advance of searching for its candidate image set.

Now, all the clauses within the (three) sets in Table 11 are φ_c-equivalent. MULTIMAP (defined above) just chooses the first clause, which is INTNOR-19 here. It uses this clause to extend a_3 to a_4. EXTENDER then iterates to consider extensions of a_4. Shortdescr[a_4] finds a single candidate image for an axiom in some[a_4] and creates a final partial analogy, a_5 (Table 9).

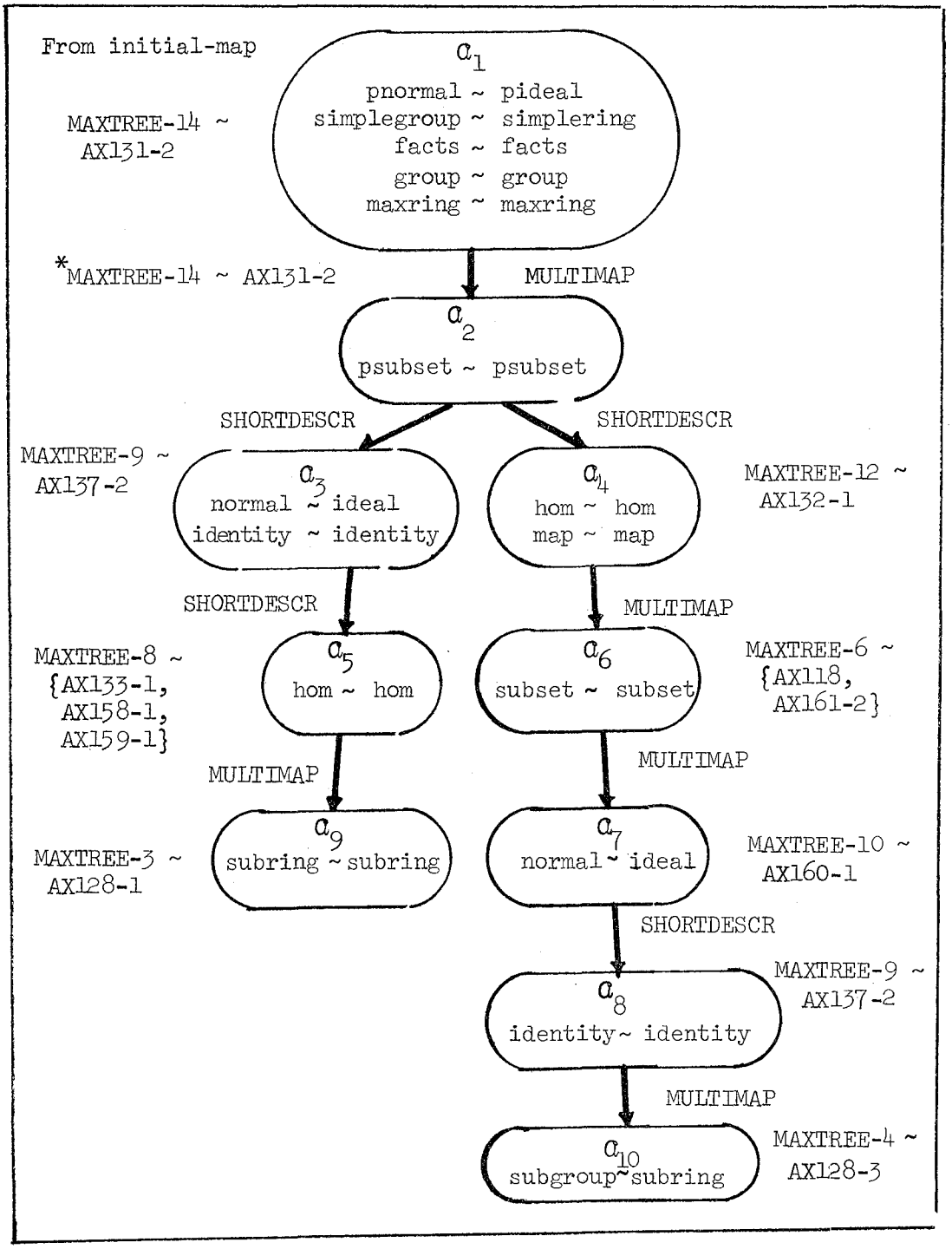
E. The Chunking Process

In our preceding discussion of SHORTDESCR, BUDSET contained only one axiom at each stage of iteration. Extending a_j by SHORTDESCR led to a unique extension a_{j+1} when it was successful. In general, BUDSET has more than one member, and SHORTDESCR can be successful in these cases too. In each such event, we can naturally have several partial analogies, and each is a legitimate extension of its immediate predecessor. A simple example of this phenomenon is presented in Table 12 for an attempt to generate an analogy for problem T_3 - T_4 (Table 2). Partial-analogy a_2 has two distinct descendants by SHORTDESCR. Both add the associations of different predicates to $a_2{}^p$, and they do not conflict. (See Tables 12 and 13.)

EXTENDER creates two distinct descendants, a_3 and a_4 (Table 12), and continues its search. When a_3 is extended to a_5 it adds HOM ~ HOM which is contained in $a_4{}^p$. a_4 is extended in rapid sequence to a_6, a_7, and a_8. EXTENDER then extends a_5 to to a_9, and extends a_8 to a_{10} and finds that it has developed a complete a^p for this problem. Unfortunately, it developed a redundant line of search: $a_9^p \subset a_{10}^p$. A substantial amount of work was spent in developing a_5 and a_9 that may have been avoided. Suppose we created a larger partial-analogy $a_{11}^p = a_3^p \cup a_4^p$. Since a_3 and a_4

Table 12

ANALOGY-SPACE SEARCH FOR T_3 - T_4 WITHOUT CHUNK



*See Table 13 for definitions of the MAXTREE axioms.

Table 13

AXSET FOR THEOREM T_3

T_3 : If a factor group G/M is simple, M is a maximal normal subgroup of G.

T_3' : Group[g;*] ∧ pnormal[m;g;*] ∧ facts[x;g;m] ∧ simplegroup[x;*] → maximal[m;g;*]

MAXTREE-1 ¬ident[star;x;g] ∨¬pnormal[x;g;star]
neg[pnormal] neg[ident]

MAXTREE-2 normal[x;g;star] ∨ ¬pnormal[x;g;star]
neg[pnormal] pos[normal]

MAXTREE-3 subgroup[x;g;star] ∨ ¬normal[x;g;star]
neg[normal] pos[subgroup]

MAXTREE-4 subset[h;g] ∨ ¬subgroup[h;g;star]
neg[subgroup] pos[subset]

MAXTREE-5 hom[hommap[star;n;g;x]; g:x] ∨ ¬group[g;star]
∨ ¬normal[n;g;star]
neg[facts] neg[normal] ∨ neg[group] ∨ pos[hom]

MAXTREE-6 map[phi;x;map[x;b;a;phi] ∨ ¬hom[phi;a;b]
∨ ¬subset[x;a]
neg[subset] neg[hom] pos[map]

MAXTREE-7 group[g;star] ∨¬simplegroup[g;star]
neg[simplegroup] pos[group]

MAXTREE-8 ¬identity[star2;y;b] ∨ ¬hom[f;a;b]
∨ ¬group[a;star1] ∨ ¬group[b;star2]
∨ ¬map[f;x;y] ∨ identity[star1; x:a]
neg[map] neg[group] neg[hom] impcond[ident]

MAXTREE-9 ¬normal[y;b;star2] ∨ ¬identity[star;x;g]
∨ ¬psubset[x;g] ∨ ¬simplegroup[g;star]
neg[simplegroup] neg[psubset] pos[ident]
neg[normal]

Table 13
(Concluded)

MAXTREE-10	normal[y;b;star2] ∨ ¬group[a;star1] ∨ ¬group[b;star2] ∨ ¬group[b;star2] ∨ ¬hom[phi;a;b] ∨ ¬map[phi;x;y] ∨ ¬normal[x;a;star1] neg[map] neg[hom] neg[group] impcond[normal]
MAXTREE-11	psubset[x;g] ∨ ¬pnormal[x;g;star]
MAXTREE-12	psubset[y;b] ∨ ¬hom[phi;a;b] ∨ ¬group[a;star1] ∨ ¬group[b;star2] neg[map] neg[group] neg[hom] impcond[psubset]
MAXTREE-13	maximal[m;g;star] ∨ ¬group[g;star] ∨ ¬pnormal[m;g;star] ∨ pnormal[otherset(star;g;m); g;star] impcond[pnormal] neg[group] pos[maximal]
MAXTREE-14	maximal[m;g;star] ∨ ¬group[g;star] ∨ ¬pnormal[m;g;star] psubset[m;otherset star;g;m] pos[subset] neg[pnormal] neg[group] ∨ pos[maximal]

are descended from the same partial-analogy α_2 by SHORTDESCR and they do not have conflicting associations, we expect that our super-analogy will expedite our search. We need to define a new operator called CHUNK, to create this new "large" partial analogy.

Definition: A partial-analogy α_ℓ is chunked from a set of partial analogies $\{\alpha_k\}$ if:

(1) Each of the α_k are descendent from the same partial-analogy α_j by SHORTDESCR.

(2) None of the α_k have conflicting associations in α_k^p — e.g., α_k^p has $p \sim q$ and α_{k+2}^p has $p \sim r$.

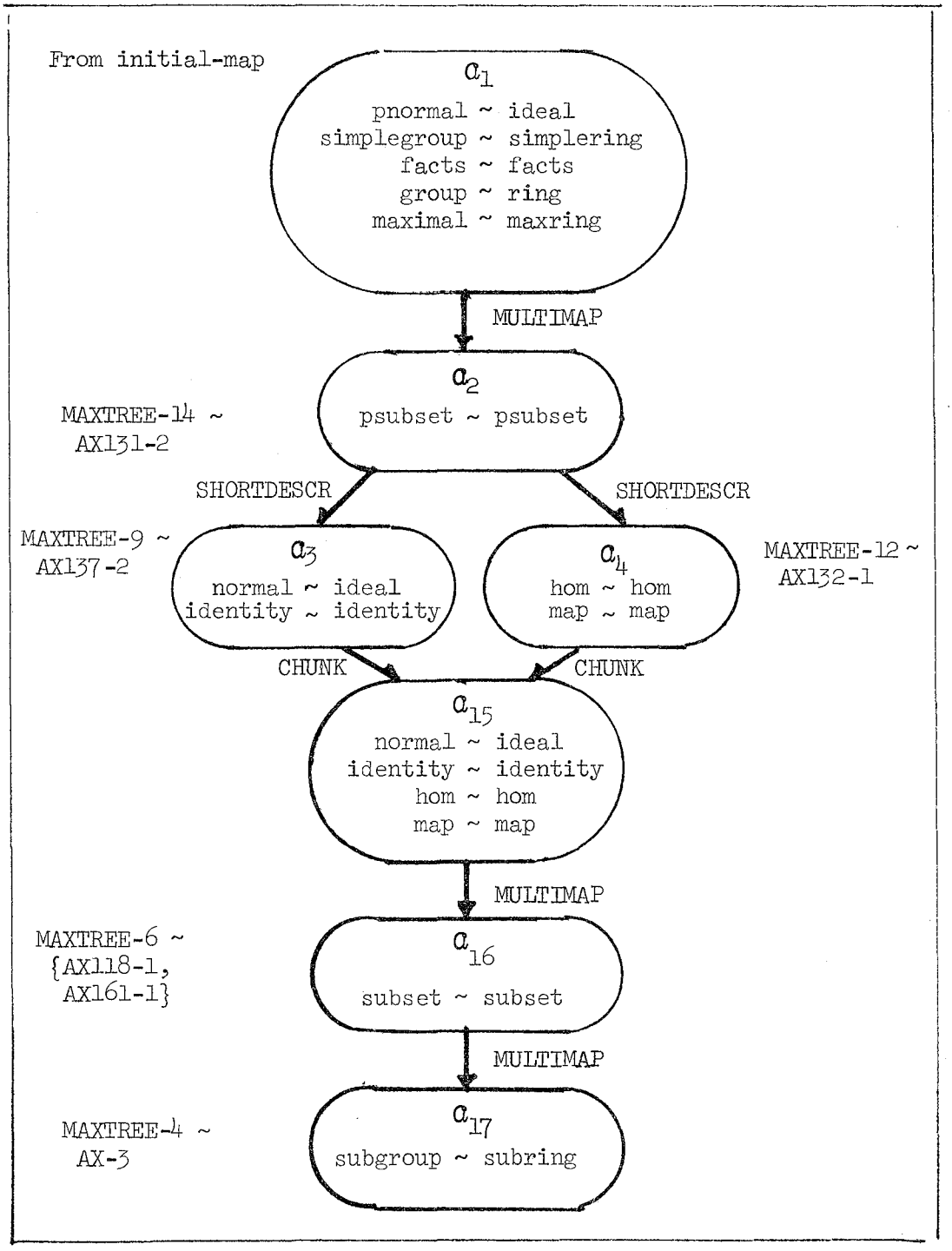
(3) $a_\ell^p = \cup_k a_k^p$.

A modified search is depicted in Table 14. A careful comparison between the searches with and without CHUNK is quite instructive. Both develop identically up to a_4. a_{15}^p (Table 14) = $a_3^p \cup a_4^p$ (Table 12).

Now, MAXTREE-8 is in all[a_{15}] and the node corresponding to a_5 (Table 12) is not developed. In our CHUNK-free search, a_7^p adds normal ~ ideal to a_6^p. In the search with CHUNK, that association is carried directly from a_3 to a_{15} by CHUNK. Hence, it need not be developed again. Chunk[a_3;a_5] is a_{15} and quite easy to compute, in contrast to every other partial-analogy which requires at least one search through the data base to find a clause that satisfies a given restricted description. If we disregard a_{15} (the CHUNKED partial-analogy) as a very low-cost item, then we see that the search with CHUNK created only six partial-analogies in contrast to the ten partial-analogies created in the search without CHUNK.

Table 14

ANALOGY-SPACE SEARCH FOR T_3 - T_4 WITH CHUNK



VII ANALYTICAL APPROACHES TO ZORBA-1

A. Introduction

Previously, we have treated ZORBA-I as a pragmatically motivated and empirically developed system. We have informally studied various experiments by the case analysis of particular examples. In contrast, we will now study four properties of ZORBA-I from a formal and analytic viewpoint:

(1) How the use of semantic types decreases the size of the search space of admissible α^p.

(2) How the use of an INITIAL-MAP prior to EXTENDER further reduces this search space.

(3) Conditions for which an analogy will aid or hinder a resolution theorem prover.

(4) Necessary conditions that AXSET and T_A must satisfy for EXTENDER to operate successfully.

ZORBA-I initiates its development of an analogy α without any a priori information regarding associations between particular predicate pairs. However, it does demand that associated predicates be of the same semantic type. This restriction considerably limits the number of possible mappings that could qualify for α^p, and will be briefly discussed first.

Suppose M_T axioms are used to prove T and they reference P_T predicates. Furthermore, suppose the data base D contains M_D axioms that include P_D predicates. If we assume that our predicates and axioms map are one-one, we can have:

$$\frac{M_D!}{(M_D - M_T)!} \quad \text{axiom mappings}$$

and

$$\frac{P_D!}{(P_D - P_T)!}$$ predicate mappings.

Each of these will associate one axiom (or predicate) used in proof[T] with one axiom (or predicate) from the data base D. If $M_D = 250$, $M_T = 10$, $P_D = 40$, and $P_T = 10$, we have about 10^{24} axiom mappings $(\frac{250!}{240!})$ and about 10^{15} $(\frac{40!}{30!})$ predicate mappings possible. Now let's look at the introduction of types. In ZORBA-I types are utilized to maintain a meaningful analogy. For example, in geometry, we prefer to associate a triangle with some other object such as a tetrahedron, or regular polygon, rather than a relationship such as bisect or parallel. A ZORBA-I user must specify a type for each predicate. He can use these types to insist upon having certain predicates map into predicates within the same equivalence class. Of course, a user can default by declaring every predicate to be of one type only, and thus allow a wider variety of mappings. Here, we want to see how the number of possible predicate mappings is reduced by the exploitation of types. For our purposes, here, a "type" will simply be the label of an equivalence class of predicates.

Suppose we have t types, and let $\frac{P_D}{t} = K_D$. There are K_D predicates of each type in the data base D. Furthermore, let there be K_T predicates of each type among the M_T predicates that appear in proof[T] $(t \times K_T = M_T)$. Then, there are $\left[\frac{K_D!}{(K_D - K_T)!}\right]^t$ possible predicate mappings.

Let $P_D = 36$, and $P_T = 12$; then the number of possible predicate mappings is indicated in the table below:

t(number of types)	t = 1	t = 2	t = 3	t = 4
$\frac{K_D!}{(K_D - K_T)!}$	6×10^{17}	1.69×10^{14}	1.64×10^{12}	6.5×10^{10}

$K_D = P_T/t$; $K_T = P_T/t$ $P_D = 36$ $P_T = 12$

In this artificial example, the inclusion of a new type reduces the number of possible maps by a factor of 10^2. Still, the number of potential maps is large (about 10^{10}). In the case of theorem T_1 - T_2 and ALGBASE, the number of potential maps is 4.3×10^5 with any types at all. Of course, we are seeking just one good α^p in this large search space.

In the preceding analysis, we assume that each predicate can be associated with any other predicate. In fact, ZORBA-I makes a more restrictive assumption. INITIAL-MAP insists that a predicate that appears in the statement of T be associated with a predicate that appears in the statement of T_A. Again, artificially,* suppose that K predicates of each type appear in the statement of T. Then,

$$K! + \left[\frac{(K_D - K)!}{(K_D - K_T + K)!}\right]^t$$ predicate associations are possible.

If K = 1, t = 4, $K_D = 9$, and $K_T = 3$ (since $P_D = 36$ and $P_T = 12$), then only 2049 mappings are permissible. Thus, breaking the creation of α^p into two portions, creates an "additive" rather than multiplicative effect on the combinatorial possibilities. Again, we substantially diminish the size of our search space. These reductions are quite striking for the three real problems for which the

*Our artificiality is to suppose that there are an equal number of predicates of each type. For example, ALGBASE has the following distribution of predicate types: PROP-1; MAP-3; STRUCTURE-8; RELATION-19; RELSTRUCTURE-11.

number of possible mappings have been completed in Table 15. We see that the effect of INITIAL-MAP and EXTENDER is to reduce the possible number of mappings by several orders of magnitude. It is also clear that EXTENDER allows the majority of possible mappin s.

B. Time-Space Analysis of the ZORBA-I Algorithm

In these few pages I want to outline an argument pertaining to the efficiency of a theorem-proving system containing ZORBA-I and QA3 (e.g., generates an analogy to restrict the data base) to one without ZORBA-I (e.g., does not restrict its data base).

Both the INITIAL-MAP and EXTENDER procedures of ZORBA-I are predicate mapping procedures.

INITIAL-MAP associates each predicate that appears in the statement of the unproved theorem T_A with a predicate that appears in the statement of the proved theorem T.

Since T and T_A are both given, we know the set of possible maps. Suppose T_A (and T) each include P_s predicates in their statements; then there are at most $P_s!$ maps (assume one-one maps). The addition of semantic types restricts the set of admissible predicates that may be associated with a given predicate. For example, if there are t types with P_s predicates per type in the theorem statements, then the number of possible maps is

$$\left[\left(\frac{P_s}{t}\right)\right]^t \quad (t < P_T) \quad (= \text{for } k = 1).$$

Note:
$$P_T! > \left[\left(\frac{P_t}{t}\right)\right]^t \qquad \left[\begin{matrix}\text{easily}\\ \text{proved}\end{matrix}\right] \quad t > 1.$$

($P_T = 12$, and $t = 3 \rightarrow 12! > [4!]^3$, etc.).

Table 15

NUMBER OF PREDICATE MAPS CONSISTENT WITH TYPE RESTRICTIONS

Theorem-Pair	Number of Predicates		Number of Allowable Maps		
	Total	Theorem Statements	No Decomposition	Initial-Map	Extender
$T_1 - T_2$	9	3	8.6×10^9	1	4.3×10^5
$T_3 - T_4$	12	5	2.5×10^{11}	16	6.12×10^3
$T_5 - T_6$	8	3	1.2×10^9	1	5.4×10^4

T_1: The intersection to two abelian groups is an abelian subgroup of the parent group.

T_2: The intersection of two commutative rings is a commutative subring of the parent rings.

T_3: A factor group G/H is simple if H is a maximal normal subgroup of G.

T_4: A quotient ring A/C is simple if C is a maximal ideal in A.

T_5: The intersection of two normal groups is a normal group.

T_6: The intersection of two ideals is an ideal.

$$\frac{P_D}{(P_D - P_T + P_S)!} \quad \text{choices}$$

for the map generated by EXTENDER (momentarily excluding type restrictions). In practice, $P_D \gg P_T \gg P_S^*$, so that the number of possible maps generated by INITIAL-MAP is <u>much</u> smaller than the number of maps that could be generated by EXTENDER.

Thus the worst behavior of ZORBA-I is most likely to be induced by the worst behavior of EXTENDER.

In my algebra data base:

$P_D = 40$ - constant for all theorems

$P_T \sim 10\text{-}12$

$P_S \sim 2\text{-}4$

} Theorem Dependent

C. <u>Background on EXTENDER</u>

EXTENDER accepts a one-one association of predicates output by INITIAL-MAP. Thus it knows the analogs of P_S predicates and must find those of $(P_T - P_S)$ more. Clearly, it does the most work if $P_S = 1$. (P_S must ≥ 1.)

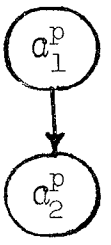
Superficially EXTENDER works as follows:

(1) Take the current predicate map α_j^p and uses it to associate some selected axiom ax_k from proof[T] with some (hopefully) analogous clause c_ℓ from the data base.

(2) Use the current predicate map α_j^p to associate the predicates of ax_k with those in c_ℓ. ax_k has been chosen so that some (but not all) of its predicates appear in α_j^p. In this association of ax_k, c_ℓ is used to learn the associations of new predicates and create a new map α_{j+1}^p which includes the union of those on α_j^p.

(3) Go to 1 (iterate until all predicates are on some a_j^p).

Observation: In the best case, one axiom would include all P_T-P_S predicates and EXTENDER would iterate only once, as illustrated below:



Provided by INITIAL-MAP

Some ax_k has all P_T-P_S predicates

In addition, we assume that we select the correct c_ℓ to associate with it immediately, and that the clause-description-matching routine outputs only one predicate association.

Observation: If only one predicate is added at each iteration of EXTENDER, we require $P_T - P_S$ iterations to complete a^p. Thus, the maximum depth of the analogy search space is $P_T - P_S$, minimally 1 (as above), and (of course) is usually in between 1 and $(P_T - P_S)$.

D. Worst-Case Analysis of EXTENDER

In this section I want to describe how EXTENDER may be inefficient when compared with a resolution system. What we want to study here are axiom systems that:

(1) Force EXTENDER to generate a "maximum" number of partial analogies in its search for a^c between T and T_A.

(2) Allow a resolution program (with an unsophisticated strategy) to generate a "minimum" number of resolvents in its search for a proof of T_A.

So, we are comparing the number of partial analogies generated by EXTENDER with the number of clauses in a resolution search. If we really want to be exact we should compare:

n_1 . cost[partial analogy] + n_2 . cost[resolvent]

with

n_3 . cost[resolvent]

where

n_1 = number of partial analogies generated by EXTENDER

n_2 = number of nodes in resolution search with the EXTENDER-derived data base

n_3 = number of nodes in resolution search without the smaller data base provided by EXTENDER.

For now, we will simply compare n_1 with n_3 under various conditions. In a later section I will comment on the relative costs of generating a partial analogy in EXTENDER's search and generating a resolvent in a resolution search.

I will explicate the following results:

(1) If a data base is explicit designed to befoul EXTENDER and aid resolution, we can find conditions where $n_1 \gg n_3$ ($n_2 = n_3$). Analogy seeking in this case decreases the efficiency of the overall ZORBA-I-QA3 system.

(2) A few simple conditions on the axioms in a data base can create searches where $n_3 \geq n_1$, and $n_2 \geq n_3$. Now, the addition of analogy seeking may do no worse, and may substantially aid the performance of the resolution system. These axiom systems are highly contrived (particularly for $n_1 = n_2 = n_3$) and are of limited usefulness.

(3) Finally, we deduce from 2, above and the nature of pragmatically interesting axiom systems, that $n_1 \ll n_3$, $n_2 \ll n_3$, and $n_1 + n_2 \ll n_3$.

In uncontrived cases, the addition of analogies can be a substantial aid to the system performance.

I'll begin by motivating the design of an axiom system to create Condition 2 above and then restrict it further to create Condition 1.

Let's consider how we may generate a worst-case for ZORBA. First, we would like an analogy search tree as deep as possible. From the preceding discussion, we want only to add one predicate association to the analogy at each iteration.

A little thought will show you (as it showed me) that this criterion forces the axioms used in proof[T] into a particular form. Each clause must contain at most two predicates, each appearing in a literal of opposite sign. In this case, a mapped clause can add information about (at most) one predicate to the current a_j^p. Thus, two predicate clauses force EXTENDER to generate a new a_j^p for each predicate-pair added. Note that a clause may contain only one predicate, or more than two literals. In the following example, c_1 and c_2 are admissible, while c_3 (three predicates) is not:

c_1: $\neg p_1[x;y] \vee p_1[y;x]$

c_2: $\neg p_1[x;y] \vee \neg p_1[x;z]\ p_2[x;y;z]$

c_3: $\neg p_1[x;y] \vee \neg p_2[s;y;z]\ p_3[z;x;y]$

Let's consider a particular "worst" axiom system, and see how to generate an associated "worst" axiom set, called WORSTBASE (Table 16).

Table 16

WORSTBASE AXIOM SET

c_4: $p_1[a,b]$

c_5: $\neg p_1[x;y] \vee p_2[y;x;g[x;y]]$

c_6: $\neg p_2[x;y;z] \vee p_3[z;y;x]$

c_7: $\neg p_3[w;x;y] \vee p_4[w;f[x;y]]$

c_8: $\neg p_4[g[u;v];z] \vee p_4[z;g[u;v]]$

c_9: $\neg p_4[f[x;y];z] \vee p_5[z;f[x;y]]$

c_{10}: $\neg p_4[f[x;y];z] \vee p_5[f[x;y]]$

c_{11}: $\neg p_5[x;y] \vee p_5[y;x] \vee p_6[x;x;y;y]$

c_{12}: $\neg p_{15}[x;y;y] \neg p_{15}[y;y;x]\ p_{16}[x;x;y;y]$

c_{13}: $\neg p_{150}[x;y;h(y)] \neg p_{150}[h(y);y;x]\ p_{16}[x;x;y;y]$

c_{14}: $\neg p_{250}[x;r[x]] \vee p_{16}[x;y;y;x]$

c_{15}: $\neg p_{14}[\quad] \vee p_{150}[\quad]$

c_{16}: $\neg p_{140}[\quad] \vee p_{15}[\quad]$

c_{17}: $\neg p_{14}[\quad] \vee p_{15}[\quad]$

c_{18}: $\neg p_{140}[\quad] \vee p_{150}[\quad]$

T: $p_6[a;a;b;b]$

Suppose T_A is $p_{16}[c;c;d;d]$.
INITIAL-MAP will associate

$$\alpha_1^p: \; p_6 \sim p_{16}$$

and associated with α_1^p, SOME = $\{c_{11}\}$.
We want to have a data base in which c_{11} will potentially map into k-different clauses.

Consider two clauses, c_{12} and c_{13}:

$$c_{12}: \; \neg p_{15}[x;y;y] \vee \neg p_{15}[y;y;x] \vee p_{16}[x;x;y;y]$$

$$c_{13}: \; \neg p_{150}[x;y;g(y)] \vee \neg p_{150}[h(y);y;x] \vee p_{16}[x;x;y;y].$$

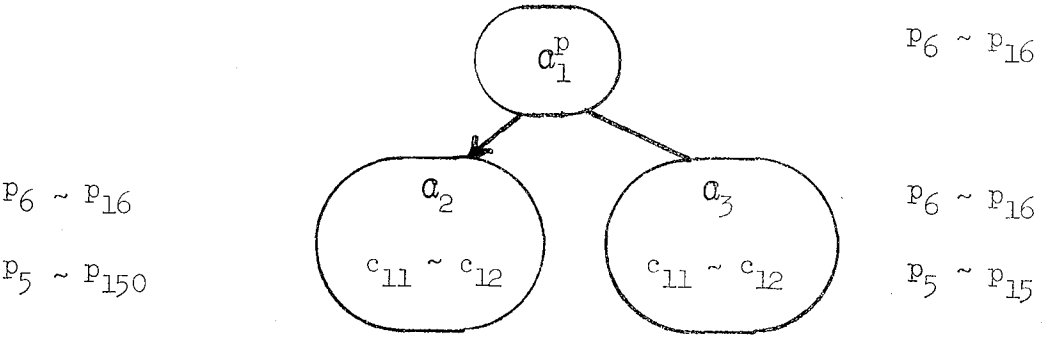
The description of c_{11} restricted to α_1^p — e.g., $pos[p_{16}]$ — will cull out both c_{12} and c_{13}.

Also, the following equivalence shows that c_{12} and c_{13} are description equivalent relative to α_1^p:

$$descr[c_{12}] = neg[p_{15}], \; pos[p_{16}]$$

$$descr[c_{13}] = neg[p_{150}], \; pos[p_{16}].$$

Thus, EXTENDER will generate two descendant analogies:



If we had a clause c_{14} in the data base, then we would have still a third descendent analogy, which adds $p_5 \sim p_{150}$.

$$c_{14}: \; \neg p_{250}[x;r[x;r[x]] \vee p_{16}[x;y;y;x]$$

For the moment, let's create a data base that gives us just two alternatives at each stage:

a_2^p and a_3^p both create a SOME = $\{c_9;c_{10}\}$

and we want clauses that will give us two alternatives for each analogy.

Consider:

c_{15}: $\neg p_{14}[\quad] \vee p_{150}[\quad]$

c_{16}: $\neg p_{140}[\quad] \vee p_{15}[\quad]$

c_{17}: $\neg p_{14}[\quad] \vee p_{150}[\quad]$.

(Predicate arguments are irrelevent because we select candidates based on predicate sign features, since only these appear in the clause descriptions.)

a_2^p can associate c_{10} with either c_{15} or c_{18}, and a_3^p can associate it with either c_{15} or c_{16}, as illustrated in Figure 11.

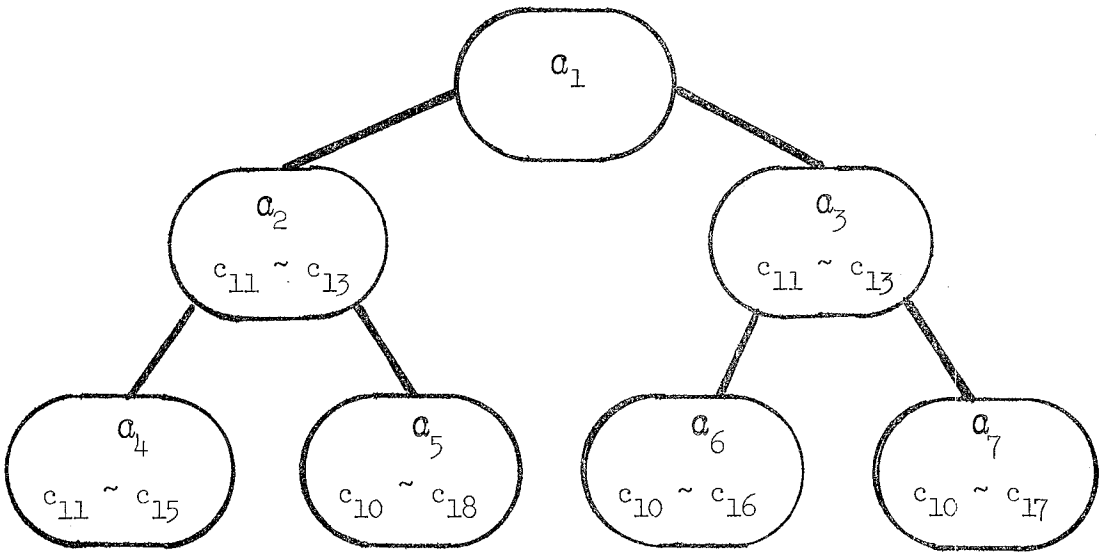


FIGURE 11. FRAGMENT OF ANALOGY SPACE FOR WORSTBASE.

In general, if we want to generate a maximally ambiguous data base that will force k-ary branching in the analogy tree, we need k! description-equivalent axioms in it for each axiom used in proof[T].

$$1 + \frac{k^{(P_T - P_S - 1)} - 1}{k - 1} \text{ nodes}$$

In this case, there may actually be k-analogies. Suppose that actually only one clause of the form $c \vee p_{20}[\ \ ]$ appears, so that at the last level of search, one analogy emerges that includes all $K_T - K_S$ predicates.

Let's compare this situation with the behavior of the resolution program. Suppose we just attempt to prove T_A with the set of axioms described above.

$\neg T_A$ will (in general) resolve with each of c_{13} and c_{12}, creating two resolvents, R_1 and R_2 (corresponding to a_2 and a_3 in Figure 11).

R_1 will resolve with c_{16} to produce an R_3 (corresponding to a_6), and with c_{17} to produce an R_4 (corresponding to a_7).

Likewise, R_2 will resolve with c_{15} and c_{18} to produce two resolvents (at least).

Each new analogy corresponds to the addition of a new predicate association to its ancestor analogy.

In "resolution-space," we need to resolve two clauses to introduce a new predicate into the search space. So, each partial analogy corresponds to some resolution. However, if the axioms have more than two literals per clause, we will need to have additional resolutions (which don't introduce new predicates) to clash with these "extra" literals. Thus, in general, our axiom sets are not restricted to clauses of length two.

From the preceding discussion we can deduce:

(1) If our data base is limited to two clauses, each of which contains two predicates (as specified above, and no function symbols for reasons to be specified below), then a resolution search for the proof of T_A may generate a number of resolutions equal to the number of analogies generated by EXTENDER in its search for the analogy between T and T_A.

(2) If the data base contains an axiom set in which each clause contains at most two predicates, and the number of literals in some clauses exceeds two, then EXTENDER will generate fewer partial analogies in its search than a resolution program proving T_A will in its search.

The preceding statements non-formally state that the number of partial analogies generated by EXTENDER will be less than or (at worst) equal to the number of resolvents generated by a resolution search program ($n_1 \leq n_3$).

Now let's look at a restriction of this axiom system that creates a situation in which $n_1 \gg n_3$ (analogy seeking is detrimental), and $n_2 \geq n_3$). Consider a variant of c_{13}, c'_{13}, which appears below with c_{12} and T_A.

c_{12}: $\neg p_{15}[x;y;y] \vee \neg p_{15}[y;y;x] \vee p_{16}[x;x;y;y]$

c'_{13}: $\neg p_{150}[x;y;h[y]] \vee \neg p_{150}[h(y);y;x] \vee p_{16}[x;y;y;y]$

T_A : $p_{16}[c;c;d;d]$ (from our previous discussion).

$\neg T_A$ will resolve with c_{12} but not with c'_{13}, since $\neg p_{16}[c;c;d;d]$ will unify with $p_{16}[x;x;y;y]$ but not with $p_{16}[x;y;y;y]$. Here we have a case in which two clauses (c_{12} and c'_{13}) are identical in terms of their descriptions and are indistignuishable in analogy-space, yet will not resolve with the same clause (here $\neg T_A$) and hence do not generate equivalent resolvents in resolution space.

I'll paraphrase his situation. To get $n_1 >> n_3$, we want to generate many more partial analogies than resolvents. Now, the axiom system we developed in our previous discussion was designed so that n_1 could equal n_3 (2-clauses). Each partial-analogy represents one potential resolution. Now, suppose our axiom system is such that clauses that are equivalent at the description level and are expected to resolve in analogous ways do not in fact resolve.

For example, see Figure 12.

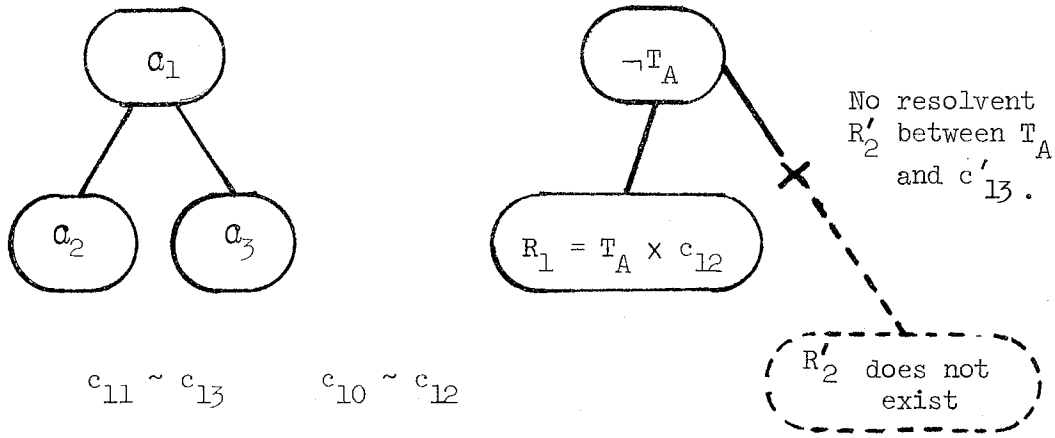


FIGURE 12. COMPARISON OF THE ANALOGY SPACE AND ITS ASSOCIATED RESOLUTION SPACE

Now, in resolution space, R'_2 * may be attempted, but never created. In contrast, a_2 is attempted and is added as a new analogy. Thus, in analogy space, a_2 is indistinguishable from a_3, and will lead to descendent analogies (Figure 11) that will never have any equivalent resolvents in the resolution search. In fact, if there are k description-equivalent variants of each clause in proof[T] on the data base, and each axiom in proof[T] is a 2-clause, then

---

*Notice that we name R'_2 before we say that it doesn't exist. This is much like needing to describe a purple cow in order to point out that none of those exist either.

$$n_1 = m\text{-}1 \quad \text{and} \quad n_3 = \frac{k^{n-2}-1}{k-1} + 1.$$

If k 2 and n = 10, $n_1 = 9$ and $n_3 = 256$; if k = 2, n = 12, $n_1 = 11$, and $n_3 = 1024$.

Let us review these two cases and see what is at issue. In the case of the first "bad" axiom system we simply let the axioms be description-equivalent in such a way that EXTENDER could create a distinct partial analogy for each resolvent that could be created by the resolution search procedure. In the second case, the "poor axiom" system, we created axioms that did not resolve because appropriate literals would not unify, while (superficially) at the level of clause descriptions, k-clauses at each level appear equivalent. In passing, I want to note that the proofs for which resolution-without-EXTENDER searches are more efficient than resolution-with-EXTENDER searches are in linear-format with no axiom applied more than once. These are a subset of "input proofs"[30] and it is known that only certain theorems may have a proof in this form. It is a very restricted proof format.

We have just considered two extreme cases. In the best case, we develop just one partial analogy C_2 since some clause in AXSET contains all the predicates we need. In the worst case, we may generate many more analogies than resolvents since EXTENDER's descriptions are insensitive to some features of resolution — e.g., when two clauses unify. Nevertheless, we had to construe a special data base to confuse EXTENDER. In the next few paragraphs I shall describe how a "real" data base differs from our construed one.

In our creation of WORSTBASE (Table 16), we made three assumptions, none of which is true, in general. (Of course, <u>all</u> are true for our contrived WORSTBASE but none is true in our more typical ALGBASE.) The assumptions are as follows:

(1) Each clause contains either one or two predicate letters. This is a highly artificial assumption that interacts with other assumptions. In contrast, ALGBASE contains 239 clauses, of which 98 have 3 predicate letters, 33 have 4 predicate letters, and 3 have 5 predicate letters (in addition to 98 that contain 2 predicate letters, and 7 that contain one predicate letter.) Over 50% of the clauses in ALGBASE contain more than two predicate letters.

(2) Each resolvent will resolve with some small number of clauses -- approximately k. Now, if each clause contains but 2 predicate letters and the data base contains relatively many (say 10 to 50) predicates, then we expect each clause could conceivably resolve with only a few other clauses. Again, this assumption is highly artificial. We developed our artificial data base in such a way that the possible resolvents would be as few as possible. However, most axiom systems are quite "rich" and allow many resolvents (inferences). For example, in WORSTBASE, the negation of the theorem resolves with only k_r clauses. In contrast, in ALGBASE, $\neg T_1$ (Appendix B) will create 29 resolvents at one level of inference, and $\neg T_3$ can create 55 resolvents at one level of inference. These resolvents can easily create hundreds of resolvents at the next and deeper levels.

(3) At each level of the EXTENDER search, there are k description-equivalent candidate images. We created a situation in which k nearly isomorphic axiom systems are embedded in the same data bases. In ALGBASE there are 3 (k = 3); group-ring; group-group, ring-ring. The first is the only "genuine" analogy, while the latter two are identities. Now α_1^p is rich enough to map some ring-related predicates into some group-related predicates at the outset -- e.g., group~ring or normal~ideal -- and consequently

rule out the spurious maps (identities) that provide 3-way (k-way) branching. In a geometry data base, which included properties of triangles, tetrahedrons, and regular polygons, there could be two legitimate analogies: triangle~tetrahedron and triangle~regular polygon. Again, we expect the INITIAL-MAP of the problem statements to provide a α_1^p that will select out the proper analogy.

All of these assumptions interact to create a data base WORSTBASE that gives EXTENDER a comparatively hard time compared to unaided resolution, while suggesting that more pragmatic axiom systems will give EXTENDER a much easier search than unaided resolution.

E. 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. 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:

T_{11}: The intersection of two abelian groups is an abelian group,

and

T_{12}: The intersection of an abelian group and a commutative ring is an abelian group.

T_{11}': $\text{abelian}[a;*_1] \vee \text{abelian}[b;*_1] \vee \text{intersection}[c;a;b] \rightarrow \text{abelian}[c_1;*_1]$

T_{12}': $\text{abelian}[x;*_2] \vee \text{commring}[y;*_2+_2] \vee \text{intersection}[z;x;y] \rightarrow \text{abelian}[z;*_2]$.

ATOMMATCH can associate

$$\text{abelian}[c;*_1] \sim \text{abelian}[x;*_2]$$

and

$$\text{abelian}[b;*_1] \sim \text{commring}[y;*_2;+_2]$$

at different times and handle many-one predicate maps — e.g., abelian ~{abelian, commring}. 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:

$$P_1 \subset P_2$$

$$P_1 \cup \text{preds}[c] = P_2$$

$$P_1 \cap \text{preds}[c] \neq \emptyset$$

and (redundantly)

$$P_2 \cap \text{preds}[c] \neq \emptyset$$

$$P_1 \neq P_2 \ .$$

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 bridge from P_1 to P', and c_2 bridges from P' to P_2. Hence, $P_1 \subseteq P' \subseteq P_2$. In general, we will say that an unordered set of C of k clauses C bridges from P_1 to P_2 iff $\exists$ P'_1, P'_2 ... P_k, and

(1) $\exists$ $c_1 \in C_1$ and c_1 bridge from P_1 to P'_1, P'_1.

(2) $\forall_j$, $j = 2, \ldots, k-1$ and $c \in C$, and
c_j bridges from P'_j to P'_{j+1}.

(3) $\exists$ $c_k \in C$ and c_k bridge from P'_{k-1} to P_2.

Let us define:

Pr[T] = Predicates used in proof of T.

preds[T] = Predicates used in statement of T.

α = Analogy from T to T_A.

descr[c] = Description of clause c.

α[descr[c]] = Analog description of the description of c under α.

ax[T] = Axioms used in proof of T.

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

$\exists$ c = ax[T] and c bridges Pr[T] to preds[T]

s.t. <u>for</u> $\alpha(c) = \{c'', c' \text{ satisfies[descr[c']]}\}\ \forall\ c' \in c$
$\alpha(c)$ bridges from $Pr[T_A]$ to preds[T].

More verbally, <u>some subset</u> of the axioms in the proof of T that bridge from the domain of INITIAL-MAP to $preds[T_A]$ have a set of image clauses under α that bridge the images of INITIAL-MAP to $preds[T_A]$. Thus, the proofs need not be isomorphic, but some restricted subset have a nearly isomorphic image similarly restricted to the bridging condition.

VIII VARIATIONS OF ZORBA-I

A. Introduction

We have surveyed ZORBA-I by analyzing its structure (Chapter VII) and examining its behavior on pairs of theorems drawn from abstract algebra (Chapter VI). By design, it is limited to generating analogies for a subclass of the "relationally similar" analogies described in Chapter II. In particular, ZORBA-I is restricted by the following assumptions:

(1) α^p is a one-one map.

(2) α^p associates predicates of the same type.

(3) The axioms in AXSET are free of constants.

(4) The statements of T and T_A are free of function symbols.

(5) The atoms in the statements of T and T_A can be associated one-one.

These specifications are stated abstractly, and they limit the domains to which ZORBA-I can be applied. Theorem pairs in abstract algebra that exploit the group-ring analogy seem to satisfy Assumption 1, while many interesting analogies in plane-geometry do not. Eliminating constants from our axioms (Assumption 3) limits us to mathematics and some puzzles. Almost every analogy I have seen preserves semantic types (Assumption 2) for some suitable set of types.

If the strategies that are aceptable by the theorem-proving system insist that an axiom is either in the data base or not considered at all, then α^c must be complete for ZORBA-I to give it useful information. On the other hand, if the theorem prover can prefer some axioms to others, then it could use an incomplete α^c as a guide for which axioms to prefer. In this chapter, we will consider variations of ZORBA-I that relax some of these restrictions.

B. Variations of EXTENDER for One-Many Predicate Maps

In EXTENDER, α^p is limited to one-one predicate associations by MAPDESCR (Chapter V), the algorithm that associates clause descriptions. The version presented (and implemented) will halt if it can not find a one-one mapping. Suppose we generalized MAPDESCR to create one-many predicate mappings. A new algorithm of this sort would be the product of some fresh research, but we will assume, for the moment, that we have one. Then we need to redefine the analog of a clause description $\alpha_j^p[\text{descr}[c]]$, to include our one-many associations. For example, if:

$$\text{descr}[c] = \text{neg}[p_1]\ \text{pos}[p_2]$$

and

$$\alpha_j^p: \quad p_1 \sim \{q_1, q_2\}$$
$$p_2 \sim q_3$$

then $\alpha_j[\text{descr}[c]]$ may be the set $\{\text{neg}[q_1], \text{pos}[q_3]; \text{neg}[q_2], \text{pos}[q_3]; \text{neg}[q_1], \text{neg}[q_2], \text{pos}[q_3]\}$.

We could consider a candidate image of c to be a clause that satisfies any of these three descriptions.

Some of this discussion can be clarified by studying a simple example. Consider the following pair of theorems:

T_{14}: A point on the bisector of an angle is equidistant from its sides.

T_{13}: A point on the perpendicular bisector of a line segment is equidistant from its end points.

Table 17 contains statements of these theorems and illustrative figures. Table 18 includes some of the axioms necessary to prove them, and Table 19 contains definitions of the predicate symbols used in the axioms. The two theorems can be proved by following a similar plan (proving two right triangles congruent as a subgoal)

Table 17

STATEMENTS, FIGURES, AND AXSET FOR BISECTION THEOREMS

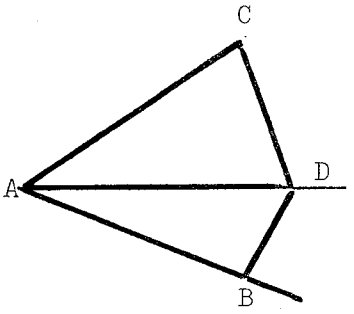


Figure for Theorem T_{14}

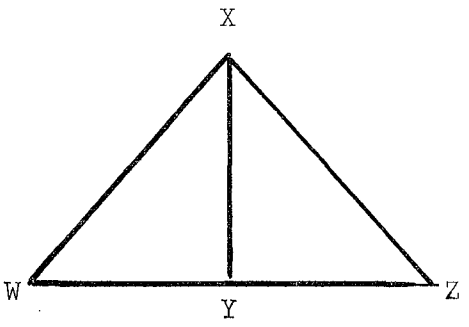


Figure for Theorem T_{13}

T_{14}': perpendicular[line[D;B]; line[A;]
∨ perpendicular[line[D;C]line[C;A]
∨ abisect[line[D;A]; angle[A;ep[B;C]]
→ eqlen[line[B;D]; line[D;C]].

T_{13}': perpendicular[line[X;Y]; line[W;Z]]
∨ lbisect[line[X;Y]; line[W;Z]]
→ eqlen[line[X;W]; line[X;Z].

Table 18

AXIOMS NEEDED TO PROVE T_{13}' AND T_{14}'

AX1: eqlen[line[x;y]; line[x;y]]

AX2: ¬triangle[x;y;z] ∨ ¬ritang[y;ep[x;z]]
∨ ritriangle[y;x;z]

AX3: ¬ritriangle[x;y;z] ∨ ¬ritriangle[u;v;w]
∨ ¬eqlen[line[y;z]; line[v;w]]
∨ ¬eqang[y;ep[x;z]; v; ep[u;w]]
∨ tcongruent[x;y;z;u;v;w]

AX4: ¬tcongruent[x;y;z;u;v;w] ∨ eqlen[line[ ]; line[ ]]

AX5: ¬abisect[line[y;u]; angle[x;ep[y;z]]]
∨ egang[y;ep[y;z]; y; ep[x;u]]

AX6: ¬abisect[line[x;y]; line[u;v]]
∨ ¬intersect[r;line[x;y]; line[u;v]]
∨ eqlen[line[u;r]; line[r;v]]

AX7: ¬ritriangle[x;y;z] ∨ ¬ritriangle[u;v;w]
¬eqlen[line[x;y]; line[x; z]]

AX8: ¬perpendicular[line[x;y]; line[u;v]]
∨ ¬intersection[line[x;y]; line[u;v] ∨ ritang[z;ep[u;y]]

Table 19

DEFINITIONS OF PREDICATE AND FUNCTION SYMBOLS FOR GEOMETRY

abisect[line[x;y]; angle[u;ep[u;w]]	Line $\overline{xy}$ bisects uvw.
eqang[y;ep[x;z];v;ep[u;w]]	$\measuredangle$ xyz = $\measuredangle$ uvw.
eqlen[line[x;y]; line[u;v]]	$\overline{xy} = \overline{uv}$
intersect[z;line[x;y]; line[u;v]	Lines $\overline{xy}$ and $\overline{uv}$ intersect at point z.
lbisect[line[x;y]; line[u;v]]	Line $\overline{xy}$ bisects line $\overline{uv}$.
perpendicular[line[x;y]line[u;v]]	$\overline{xy} \perp \overline{uv}$.
ritang[y;ep[x;z]]	$\measuredangle$ xyz is a right-angle vertex y and end points [ep] x and z.
ritriangle[x;y;z]	Points x, y and z form a right triangle with right-angle $\measuredangle$ xyz.
tcongruent[x;y;z;u;v;w]	$\triangle$ xyz $\cong \triangle$ uvw.
triangle[x;y;z]	Points x, y and z form a right triangle.

or by using analogous sets of axioms to prove each. Now, suppose that we have proved T'_{13} using axioms {AX1, AX4, AX6, AX7, AX8}, and want to find an analogous set of axioms to use in the proof of T'_{14}. First, ZORBA-I would use INITIAL-MAP to associate the atoms in the statements of T'_{13} and T'_{14} to produce α_1^p. The version of INITIAL-MAP described in Chapter III will not work here since it takes account neither of function symbols (ATOMMATCH) nor one-many atom associations (SETMATCH). Suppose that INITIAL-MAP was appropriately generalized and able to produce a partial analogy α_1^p.

α_1^p: perpendicular ~ perpendicular

lbisect ~ abisect

eqlen ~ eqlen .

Now, ZORBA-I calls EXTENDER with α_1 as its starting analogy. It partitions AXSET and computes

$$\text{some}[\alpha_1] = \{\text{AX4, AX6, AX7, AX8}\}$$

$$\overline{\alpha_1[\text{descr[AX6]}]} = \text{neg[abisect]}, \text{pos[eqlen]}.$$

AX6 is the only clause in some[α_1] whose restricted description (with respect to α_1) has more than one feature. AX7, for example, has neg[eqlen] as a description and we expect many clauses in a geometry data base to satisfy this single feature. We want AX6, the definition of line bisection, to be associated with AX5, the definition of angle bisection. Now AX5 satisfies α_1[descr[AX6]], but has predicate (intersect) which is not associated with any predicate in AX6. We are assuming that our new MAPSECR (Chapter V) has been generalized to handle associations of this sort. Suppose we associate AX6 with AX5 and generate α_2:

$$\alpha_2^p: \quad \text{eqlen} \sim \{\text{eqang, eqlen}\}$$
$$\text{perpendicular} \sim \text{perpendicular}$$
$$\text{lbisect} \sim \text{abisect} \ .$$

We extend α_2 and compute:

$$\text{some}[\alpha_2] = \{\text{AX7, AX8}\} \ .$$

AX7 is the axiom used to prove the congruence of the two right triangles for T'_{13}. Now, we want to get the analogous axiom (AX3) to use in the proof of T'_{14}. First, compute

$$\text{descr[AX7]} = \text{neg[ritriangle], neg[eqlen], pos [tcongruent]}.$$

We now have to create an $\overline{\alpha_2[\text{descr[AX7]}]}$ that can be satisfied by AX3. Either neg[eqlen] or neg[eqlen], neg[eqang] will suffice, though φ_d (Chapter V) would probably prefer the latter. Again, MAPDESCR applied to AX7 and AX3 would need to handle the one-many map.

$$\text{eqlen} \sim \{\text{eqlen, eqang}\}$$

This example has been chosen for its (relative) simplicity. It exemplifies some of the vagaries of analogies with one-many predicate associations.

(1) Most predicates are associated one-one. Only a few predicates associate one-two or one-three.

(2) If a predicate p is associated with predicates q and r: $p \sim \{q,r\}$; then p may be associated with q in order to find the analog of one axiom, with r to find the analog of a second axiom, and with both q and r to find the analog of a third axiom. In this example, the associations

AX1 ~ AX2
AX6 ~ AX5
AX7 ~ AX3

satisfy this pattern.

In the last paragraph I have outlined the kind of changes that EXTENDER would require if analogies with one-many predicate maps were to be allowed. Notice that we are still dealing with a class of analogies that fit the ZORBA paradigm: a set of axioms analogous to those of the proved theorem can prove our new theorem. In this case, the correspondence between the predicates in which the axioms are expressed is more complex than in the one-one case we have examined in detail, while the axiom associations are still of the same sort.

C. Variations of INITIAL-MAP

We have discussed the limitations imposed upon ZORBA-I by EXTENDER. In addition, INITIAL-MAP imposes Restrictions 5 and 6 mentioned at the beginning of this chapter. The axioms used to prove a theorem may include function symbols since EXTENDER, which considers the axioms, ignores function symbols in its clause descriptions. In fact, all of the algebra proofs described in Chapter V and VI use function symbols in the set of necessary axioms.

In the geometric example (T'_{13} and T'_{14}) we have just considered, theorem statements and axioms that rely heavily upon function symbols provide a natural and elegant representation. It is possible to rewrite these axioms without function symbols. For example, lbisect[line[x;y]; line[u;v]] becomes line[z;x;y] $\vee$ line[w;u;v] $\vee$ lbisect[z;w]. However, a function-free axiomation requires longer clauses and more (symmetry) axioms. ATOMMATCH (Chapter IV), the operation that associates variables in clauses, would need to be generalized to include function symbols and an ability to generate more than one mapping when the functions allow syntactic symmetries. In addition, the section of SETMATCH that associates atoms into sets based on distinct analogous variables (Chapter IV) might use function symbols to aid discriminations.

The version of INITIAL-MAP that is described in Chapter III is a rather complex matching program that exploits the syntactic structure of the wffs to decide which atoms are to be associated. INITIAL-MAP performs a simple, crucial role in ZORBA-I — it creates α_1^p. EXTENDER needs α_1^p for a starting point and cares neither how α_1^p is generated nor whether it contains all the predicates used in the theorem statements. α_1^p need contain only one predicate to activate EXTENDER. If α_1^p contains more predicates, then it provides a more fertile beginning for EXTENDER. In particular, some $[\alpha_1]$ will increase with the size of α_1^p. We saw (Chapter VII) that increasing the size of α_1^p can dramatically decrease the size of the total search space for α^p. The point of these observations is quite simple: we can often generate an α_1^p with a much simpler version of INITIAL-MAP than was described in Chapter III. Even when we generalize from one-one to one-many predicate maps, we have kept our mapping type-invariant (Chapters III and VII). Predicates that are associated must be of the same semantic type.

Again, we will presume that if a predicate appears in the statement of a theorem T, then its analog will appear in the statement of T_A. Now, consider the following version of INITIAL-MAP, called INITIAL-MAP1:

Initial-map1[newwff;oldwff]: =

(1) Partition the predicates that appear in newwff by their semantic type.
Do the same for the predicates that appear in oldwff.

(2) For each predicate partition of newwff that contains only one element, pair it with the predicate partition of oldwff that has the same type, if it has but one member.

(3) If this set of paired-predicates is non-empty, set it equal to α_1^p.
If it is empty, set α_1^p to initial-map[oldwff;newwff] using the algorithm of Chapter IV.

(4) Stop and Whistle.

For example, let T_{13}' and T_{14}' mentioned earlier in this chapter be oldwff and newwff respectively. Furthermore, let RELATION be the semantic type of {perpendicular, lbisect, abisect}, and EQREL be the semantic type of the predicate eqlen (Table 19). INITIAL-MAP1 will create $\alpha_1^p = \{\text{eqlen} \sim \text{eqlen}\}$. We still might want to run INITIAL-MAP on the two wffs to expand α_1^p to include the remaining predicates they contain. But we are not compelled to do this. This simplified version of INITIAL-MAP can be used either as a preprocessor or as a substitute for it. It also provides some means for handling wff pairs with non-isomorphic syntax when the semantic types are fine enough to unambiguously associate predicates based on predicate types only.

D. Treating Constants

In addition to restricting its predicate association to one-one maps, ZORBA-I does not allow axiom systems that include constants. In contrast to the one-many maps treated in the preceding paragraphs, creating analogies in axiom systems that include constants will probably require analysis algorithms different in spirit from INITIAL-MAP and EXTENDER.

Consider a robot that is instructed to go from SRI to (a) an office on its floor; (b) Stanford University; (c) San Francisco; (d) New York City; (e) 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 office 5 with Chicago, Stanford with San Francisco, etc. The robot's actions to get from SRI to Stanford, 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.

E. 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 a 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 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 set have images? 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 on 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 rig various devices, such as making the clauses in A* "pseudo-units" that would be attended to early. Or, with torch and sword, one could restruc-

ture QA3 around a "graded memory" which orders clauses according to a user specified ordering function. 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. These optimal strategies are difficult to reform to wisely exploit a priori hints and guides that are problem-dependent. Various kinds of a priori information can be added. It is a separate and sizeable research task to decide how to do it. I presume, but do not know, that these comments extrapolate to other problem-solving procedures. A system that is organized around a priori hints, heretofore user-supplied, may look very different than one that is designed to do its best on its own. QA3 was chosen because it was available and saved years of work in developing a new theorem prover that would be more suitable. However, further research in AR may well benefit from focusing attention on a more flexible theorem-proving system that can accept a wide range of "advice" from the analogy generator.

IX ZORBA IN RETROSPECT

I should like to encapsulate some of the key concepts that are implicit in ZORBA and embodied in ZORBA-I:

(1) Some fairly interesting analogical reasoning 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 utilize different a priori information and consequently will require different analogy-generation programs tailored to its representations. In Chapter III, I suggested how an analogy system oriented toward GPS would differ from one oriented toward resolution logic.

(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 reasoning by analogy is to specify precisely how the derived analogical information is to be used by the problem solver. For the class of analogies handled by ZORBA we tacitly assume that restricting our data base is the means to exploit the analogy. For other kinds of analogies (Chapter III) a wider variety of uses may be suggested for the information to be derived from the analogy. We would like a system to automatically decide that one analogy can be used only to provide a particular subgoal for the problem solver while another analogy can be used to provide a complete plan and still another analogy can be used only to suggest a particular set of axioms without specifying the sequences they should be used in.

(5) An effective, nontrivial analogy generator can be adequately built that uses a simple type theory and primitive semantic selection rules.

(6) Although analogies are nonformal and are semantically oriented, nontrivial analogies can be handled by a semi-formal system wrapped around a highly formal theorem-prover.

The first and last remarks suggest a fertile research strategy. Many good analogies are suggestive; the relational structures between the solved and unsolved problems are similar, but not homomorphic. In fact, their relationships are less well behaved than any of the mappings in our mathematicized language. For example, the geometric analogy described in Chapter VIII needs to allow a few predicates to map one-many while most are mapped one-one. The restrictions on the analogy α (Chapter III) were largely defined after most of ZORBA-I was implemented. Much of the formalism employed in explaining the algorithms is also post-hoc. Creating a formal description of ZORBA-I has helped clarify and articulate such concepts as the restrictions on α^p and the analog description of a clause when α^p is one-many. ZORBA-I was pragmatically designed within the framework of clause descriptions and sequences of partial analogies; any procedure that worked for abstract algebra was acceptable. The freedom from formalism in the early stages of this research focused attention on a rich class of theorems regardless of their formal properties. In contrast, a research strategy that attempted to formally define an analogy at the outset and proceed with much rigor most likely would have yielded a complete procedure for a less interesting class of problems. The choice of a problem domain is tricky. We want a wide variety of analogies. At the same time they must entail sufficiently simple problems to be solved by our simple-minded problem-solving systems. In mathematics both abstract algebra and Euclidean geometry are "dense" in analogies between pairs of theorems that are not very complex. In contrast, the analogies that can be

exploited in number theory are between theorems that require theorem provers much more powerful than we now have. We all use analogies to aid solving problems and proving theorems, regardless of the area we are considering. However, most domains are sparse in good nontrivial analogies between simple problems. We would be aided by a small catalog of the kinds of analogies we can find at various levels of problem difficulty in different areas. Such a study could refine the approach of Chapter II to include the role of semantic types and restrictions on the submaps of α — e.g., α^p.

Even within the ZORBA paradigm we need at least two styles of generating an analogy. ZORBA-I is an instance of one, and the comments about treating constants in the context of robot manipulation problems (Chapter VIII) calls for another, still undeveloped approach.

ZORBA-I passes only a modified data base to its associated theorem prover. Much more information is latent — e.g., how to use a particular axiom. In resolution, for example, a ground-unit clause may be needed only once in a proof, but generate a vast number of irrelevent resolvents in the search for that proof. It may unify with literals in many different clauses and be given a great deal of attention in a unit-preference strategy. We need to learn how to specify when such a unit should be used. More generally, we need to learn how to specify and represent such information for a problem-solving system.

PLANNER is a problem-solving system that has recently been developed at M.I.T. It allows a user to specify whether a particular theorem is to be used for forward inference or backward chaining. It incorporates a flexible pattern matching language and appropriate features to allow a user to select the theorems which may be used in inference chains. From the point of view of problem-solving research it makes little difference whether such advice is given by knowledgable persons or an analogy-generating program. From our point of

view, since PLANNER[30] is designed to accept advice, it may be a superb vehicle for handling a wider variety of analogical information if its problem solving power is adequate. It is not yet clear whether PLANNER can prove any of the theorems used in the experiments reported here. If PLANNER were to be used as a target problem solving vehicle for a new analogy system (call it ZORBA-II), then ZORBA-II would place somewhat different constraints on its mappings than does ZORBA-I. Table 20 depicts the English, wff, resolutions and PLANNER representations of two axioms that were found to be analogous by ZORBA-I (Chapter VI). Resolution represents these wffs by one clause each while PLANNER distinguishes two possible theorems which differ in their use. A THANTE theorem is used to make a forward inference. For example, (THANTE (X) (P X)(THASSERT (Q_yX))) could assert (Q A) when the data base includes (P A). In contrast, a THCONSE theorem is used for backward chaining. (THCONSE (X) (Q X) (THGOAL (P X))) will be triggered to set up the goal (THGOAL (P A)) if it is ever attempting to prove (Q A). The two uses correspond to the same wff:

$$\forall(x)\ p[x] \rightarrow q[x]$$

ZORBA-I has but one axiom map, α^c, which associates clauses one-many. We would expect that ZORBA-II would have an axiom submap that associates THANTEs with THANTEs and a separate submap to associate THCONSEs with THCONSEs. More verbally, if an axiom ax_k was used to prove a theorem T by backward chaining (THCONSE), we would expect the analog of ax_k would prove the analogy theorem T_A by backward chaining also. By using PLANNER theorem types, we can map local proof structure (under our analogy) by preserving theorem types under the analog α.

I have purposely omitted two important issues:

(1) Given a theorem T how can we recognize a good analogous theorem T from among the set of theorems we have proved?

(2) How do the representations we use affect our ability to perceive and exploit analogies?

Table 20

English:

Every abelian group is a group

predicate calculus wff: $\forall(x\ *)$ abelian[x;*] $\rightarrow$ group[x;*]

resolution clause: $\neg$ abelian[x;*] $\vee$ group[x;*]

PLANNER consequent theorem:

```
(THCONSE (X *) (ABELIAN X *) (THGOAL (GROUP X *)))
```

PLANNER antecedant theorem:

```
(THANTE (X *) (GROUP X *) (THASSERT (ABELIAN X *)))
```

English:

Every commutative ring is a ring

predicate calculus wff: $\forall(r\ *\ +)$ commring[r;*;+] $\rightarrow$ ring[r;*;+]

resolution clause: $\neg$commring[r;*;+] $\vee$ ring[r;*;+]

PLANNER consequent theorem:

```
(THCONSE (R * +) (COMMRING R * +) (THGOAL (RING R * +)))
```

PLANNER antecedent theorem:

```
(THANTE (R * +) (Ring R * +) (THASSERT (COMMRING R * +)))
```

Resolution and PLANNER Representations

The answers to these two interact. Within the set of algebra theorems: $\{T_1, T_2 \ldots T_{10}\}$ that were used as examples in Chapters IV-VII, the following procedure will satisfy problem (1):

(1) Convert T_A from a wff to a clause and create its description. (Chapter III)

(2) Replace each predicate in the resultant description with its semantic-type. Thus, neg[group] will become neg[structure].

(3) Search through memory to find the theorems that satisfy the type-description of T_A.

Now, the forms of the theorem statements we have used are so nearly isomorphic that this simple search will give us a small set of good candidate analogs. A variant of ZORBA-I could be used to test which of these few candidates create a complete $\mathcal{C}$, and we have solved our problem.

Now, suppose that our theorem statements are not so similar. Consider the following theorems:

T_1 : $\forall(x\ y\ *)$
abelian[x;*] $\wedge$ abelian[y;*] $\wedge$ intersection[z;x;y]
$\rightarrow$ absubgroup[z;x;*]

T_2 : $\forall(r;s;w;x;+)$
commring[r;*;+] $\wedge$ commring[s;*;+] $\wedge$ intersection[w;x;+]
$\rightarrow$ commsubring[w;r;*;+]

T_2' : $\forall(r;s;w;*;+)$
commring[r;*;+] $\wedge$ commring[s;*;+] $\wedge$ intersection[w;*;+]
$\rightarrow$ commutative[*;w] $\wedge$ ring[w;*;+] $\wedge$ subring[w;r;*;+]

Now, the procedures described in Chapters IV and VIII will easily create the proper analogy between T_1 and T_2. T_2' is logically

equivalent to T_2. The predicate "commsubring" has been replaced with its definition.

None of the procedures we have described will find the analogy $T_1 - T_2'$.

(1) α_1^p is no longer one-one. The predicate ABSUBGROUP now has no analog that appears in the proof or statement of T_2'.

(2) The predicates in the statements of T_1 and T_2' do not correspond (INITIAL-MAP is foiled).

Unfortunately, a slight shift of form sabotages all of our algorithms. This unhappy observation should be a starting point for future research.

APPENDIX A

DEFINITIONS OF PREDICATES AND THEIR SEMANTIC TEMPLATES

Abelian[x;*] structure[set;operator]

x is an abelian group under the operation "*".

Absubgp(x;y;*] relstructure[set;set;operator]

x is an abelian subgroup of y under the operation "*".

Assoc[*;S] relation[operator;set]

The set s is associative under the operation "*".

Closed[*;S] relation[operator;set]

The set s is closed under the operation "*".

Commutative[*,S] relation[operator;set]

The set s is commutative under the operation "*".

Commring[R;*;+] structure[set;operator;operator]

The set r is a commutative ring under the operations "*" and "+".

Commutativering[r;*;+] same as commring[r;*;+]

Commsubring[x;y;*;+] relstructure[set;set;operator;operator]

The set x is a commutative subring of the set y under the operations " " and "+".

Clring[x;*;+] structure[set;operator;operator]

The set x is a commutative ring with a multiplicative unit under the operations "*" and "+".

Dist[*;+;S] relation[operator;operator;set]

The operation * is distributive over the operation "+" on the set "S".

Equalset[x;y] relation[set;set]

Set x equals set y.

Factorstructure[x;y;z] same as facts[x;y;z]

Facts[x;y;z] relation[set;set;set]

x = y/z.

Group[x;*] structure[set;operator]

The set x is a group under the operation "*" and the additive operation "+".

Ident[*;a;x] relation[operator;object;set]

A is the identy element of the set x under the operation "*".

In[x;S] relation[object;set]

$x \leftarrow S$

Intersection[x;y;z] relation[set;set;set]

$x = y \cap z$.

Inv[*;a;S] relation[operator;object;set]

The inverse of a under operation "*" — e.g., a^{-1}.

Lassoc[*;S] relation[operator;set]

The set S is left-associative under the operation "*".

Ldist[*;+;S] relative[operator;operator;set]

The operation "*" is left-distributive over the operation "+"

Map[f;x;y] map[F_N;set,set]

f map the set x into the set y. f:x — y.

Map[f;a;b] map[FN,object,object]

f[a] = b.

Maximal[x;y;*] relstructure[set,set,operator,operator]

x is a maximal subgroup of y under the operation "*"

Maximalgroup[x;y;*] same as maximal[x;y;*]

Maximalring[x;y;*;+] same as maxring[x;y;*;+]

Maxring[x;y;*;+] relstructure[set,set,operator,operator]

x is a maximal subring of y under the operation * and +.

Nonempty[S] prop[set]

$S \neq \emptyset$.

Normal[x;y;*] relstructure[set,set,operator]

x is a normal subgroup of y under the operation "*".

Pideal[x;y;*;+] relstructure[set,set,operator,operator]

x is a proper ideal of ring y under the operations "*" and "+".

Pnormal[x;y;*] relstructure[set,operator]

x is a proper-normal subgroup of y under the operation "*".

proper[same as pideal[x;y;*;+]

propernormal[x;y;x] same as pnormal[x;y;*]

Pnsubgroup[x;y;*] relstructure[set,set,operator]
set x is a proper normal subgroup of y under the set "*].

Psubset[s;y] relstructure[set,set]
x ⊂ y . . .

Rassoc[*;S] relstructure[operator,operator,set]
The set S is right-associative under the operation "*".

Rdist[*;+S] relstructure[operator,operator,set]
The operation "*" is right-distributive over operation "+"
in the set S.

Ring[R;*;+] structure[set,operator,operator]
The set R is a ring under the multiplicative operation "*]
and the additive operation "+".

Simplegroup[x;*] structure[set;operator]
The set x is a simplegroup under the operation "*".

Simplering[x;*;+ structure[set,operator,operator]
The set x is a simplering under the operation * and +

Subgroup[x;y;*] relstructure[set;set;operator]
The set x is a subgroup of y under "*".

Subring[x;y;*;+] relstructure[set,set,operator,operator]
The set x is a subring of the set y under "*" and "+".

Subset[x;y] relation[set,set]
x y.

Times[*;x;y;z] relation[operation,object,object,object]

z = x * y.

Unitring[S,*,+] structure[set,operator,operator]

The set S is a ring with a unit under "*" and "+".

Welldef[*;S] relation[operator,set]

The operation "*" is well defined on the set S.

APPENDIX B

LISTING OF ALGBASE

The following listing of the 239 clauses in ALGBASE uses the predicate symbols defined in Appendix A. They are indexed by a "clause name" to facilitate reference when considering the experiments reported in Chapter VI. These names are indexed by two parameters, n and k, in the format AXn-k. For example, AX3-2 and AX3-3 are the names of two clauses in ALGBASE. The k-parameter following the dash numbers a clause with respect to its parent wff, AXn. The n-parameters are generated sequentially (by QA3) whenever a new wff enters its data base. Thus, the first wff entered is named AX1 and its clauses are named AX1-1, AX1-2, etc. If AX25 is

$$\forall(x)\ p[x] \rightarrow q[x] \wedge r[x],$$

then

$$\text{AX25-1:}\quad \neg p[x] \vee r[x]$$

and

$$\text{AX25-2:}\quad \neg p[x] \vee q[x]$$

are its two derivative clauses. In principle, all clauses are independent, while in practice the same skolem function may appear in several clauses that are descendent from the same parent wff. For example, see the clauses associated with AX52 that define the predicate "commutative." Skolems functions that were automatically generated by a prenex algorithm (in QA3) are denoted by SKj — e.g., SK5.

The listings in this appendix were printed on a PDP-10 line printer and conform to its type set. The or sign ($\vee$) is omitted between literals, and the negation sign ($\neg$) appears as a dash (-). Finally, the description of each clause (Chapter III) is printed directly below it.

```
1, AX1-1 SUBGROUP[H,G,STAR] -NORMAL[H,G,STAR]

NEG[NORMAL] POS[SUBGROUP]


2, AX1-2 IN[U,H] -IN[HH,H] -IN[GG,G] -IN[Y,G] -IN[M,G]
-TIMES[STAR,GG,HH,Y] -TIMES[STAR,Y,M,U] -INVERSE[STAR,M,GG]
-NORMAL[H,G,STAR]

NEG[NORMAL] NEG[INVERSE] NEG[TIMES] IMPCOND[IN]


3, AX2-1 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR]
-IN[SK5[STAR,G,H],H]

NEG[IN] NEG[SUBGROUP] POS[NORMAL]


4, AX2-2 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR]
INVERSE[STAR,SK4[STAR,G,H],SK2[STAR,G,H]]

POS[INVERSE] NEG[SUBGROUP] POS[NORMAL]


5, AX2-3 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR]
TIMES[STAR,SK3[STAR,G,H],SK4[STAR,G,H],SK5[STAR,G,H]]

POS[TIMES] NEG[SUBGROUP] POS[NORMAL]


6, AX2-4 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR]
TIMES[STAR,SK2[STAR,G,H],SK1[STAR,G,H],SK3[STAR,G,H]]

POS[TIMES] NEG[SUBGROUP] POS[NORMAL]


7, AX2-5 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR]
IN[SK4[STAR,G,H],G]

POS[IN] NEG[SUBGROUP] POS[NORMAL]


8, AX2-6 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR]
IN[SK3[STAR,G,H],G]

POS[IN] NEG[SUBGROUP] POS[NORMAL]
```

9, AX2-7 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR] IN[SK2[STAR,G,H],G]

POS[IN] NEG[SUBGROUP] POS[NORMAL]

10, AX2-8 NORMAL[H,G,STAR] -SUBGROUP[H,G,STAR] IN[SK1[STAR,G,H],H]

POS[IN] NEG[SUBGROUP] POS[NORMAL]

11, AX3-1 GROUP[H,STAR] -SUBGROUP[H,G,STAR]

NEG[SUBGROUP] POS[GROUP]

12, AX3-2 GROUP[G,STAR] -SUBGROUP[H,G,STAR]

NEG[SUBGROUP] POS[GROUP]

13, AX3-3 SUBSET[H,G] -SUBGROUP[H,G,STAR]

NEG[SUBGROUP] POS[SUBSET]

14, AX4-1 SUBGROUP[H,G,STAR] -GROUP[H,STAR] -GROUP[G,STAR] -SUBSET[H,G]

NEG[SUBSET] NEG[GROUP] POS[SUBGROUP]

15, AX5-1 NORMAL[X,Y,STAR] -ABELIAN[Y,STAR] -SUBGROUP[X,Y,STAR]

NEG[SUBGROUP] NEG[ABELIAN] POS[NORMAL]

16, AX6-1 SUBGROUP[X,Y,STAR] -RING[Y,STAR,PLUS] -SUBRING[X,Y,STAR,PLUS]

NEG[SUBRING] NEG[RING] POS[SUBGROUP]

33, AX12-1 ABELIAN[G,STAR] -GROUP[G,STAR]
-COMMUTATIVE[STAR,G]

17. AX7-1 ABELIAN[X,STAR] -ABELIAN[Y,STAR] -SUBGROUP[X,Y,STAR]

NEG[SUBGROUP] IMPCOND[ABELIAN]

18. AX8-1 SUBGROUP[BB,AA,STAR] -HOM[F,A,B] -GROUP[A,STAR] -SUBGROUP[AA,A,STAR] -MAP_1[F,AA,BB]

NEG[MAP_1] NEG[GROUP] NEG[HOM] IMPCOND[SUBGROUP]

34. AX21-1 GROUP[G,STAR] -WELLDEF[STAR,G] -ASSOC[STAR,G] -IDENTITY[STAR,E,G] -IN[SK12[STAR,G],G]

NEG[IN] NEG[IDENTITY] NEG[ASSOC] NEG[WELLDEF] POS[GROUP]

35. AX21-2 GROUP[G,STAR] -WELLDEF[STAR,G] -ASSOC[STAR,G] -IDENTITY[STAR,E,G] INVERSE[STAR,SK12[STAR,G],SK11[STAR,G]]

POS[INVERSE] NEG[IDENTITY] NEG[ASSOC] NEG[WELLDEF] POS[GROUP]?]

36. AX21-3 GROUP[G,STAR] -WELLDEF[STAR,G] -ASSOC[STAR,G] -IDENTITY[STAR,E,G] IN[SK11[STAR,G],G]

POS[IN] NEG[IDENTITY] NEG[ASSOC] NEG[WELLDEF] POS[GROUP]

37. AX23-1 NORMAL[N,G,STAR] -GROUP[G,STAR] -FACTS[X,G,N]

NEG[FACTS] NEG[GROUP] POS[NORMAL]

38. AX24-1 FACTS[SK13[STAR,B,A],A,B] -GROUP[A,STAR] -NORMAL[B,A,STAR]

NEG[NORMAL] NEG[GROUP] POS[FACTS]

39. AX26-1 HOM[SK14[STAR,Z,Y,X],Y,X] -FACTS[X,Y,Z] -GROUP[Y,STAR]

NEG[GROUP] NEG[FACTS] POS[HOM]

40. AX27-1 IDENTITY[STAR,B,X] -GROUP[A,STAR] -FACTS[X,A,B]

NEG[FACTS] NEG[GROUP] POS[IDENTITY]

```
42,      AX29-1        NORMAL[X,G,STAR]        -GROUP[G,STAR]
-NORMAL[A,G,STAR] -NORMAL[B,G,STAR] -INTERSECTION[X,A,B]

NEG[INTERSECTION] NEG[GROUP] IMPCOND[NORMAL]


43,      AX30-1        NORMAL[BB,AA,STAR]      -GROUP[A,STAR]
-NORMAL[B,A,STAR] -HOM[F,A,AA] -MAP1[F,B,BB]

NEG[MAP1] NEG[HOM] NEG[GROUP] IMPCOND[NORMAL]


44,      AX31-1       INVERSE[STAR2,EE,H]      -GROUP[G,STAR1]
-GROUP[H,STAR2]       -HOM[PHI,G,H]       -INVERSE[STAR1,E,G]
-MAP1[PHI,E,EE]

NEG[MAP1] NEG[HOM] NEG[GROUP] IMPCOND[INVERSE]


45,      AX32-1       TIMES[STAR2,B1,B2,B3]      -MAP1[F,A1,B1]
-MAP1[F,A2,B2]       -MAP1[F,A3,B3]     -TIMES[STAR1,A1,A2,A3]
GROUP[G,STAR1]

POS[GROUP] NEG[MAP1] IMPCOND[TIMES]


46,      AX32-2       TIMES[STAR2,B1,B2,B3]      -MAP1[F,A1,B1]
-MAP1[F,A2,B2]       -MAP1[F,A3,B3]     -TIMES[STAR1,A1,A2,A3]
GROUP[H,STAR2]

POS[GROUP] NEG[MAP1] IMPCOND[TIMES]


47,      AX32-3       TIMES[STAR2,B1,B2,B3]      -MAP1[F,A1,B1]
-MAP1[F,A2,B2]       -MAP1[F,A3,B3]     -TIMES[STAR1,A1,A2,A3]
MAP[F,G,H]

POS[MAP] NEG[MAP1] IMPCOND[TIMES]


48,      AX32-4       TIMES[STAR2,B1,B2,B3]      -MAP1[F,A1,B1]
-MAP1[F,A2,B2]       -MAP1[F,A3,B3]     -TIMES[STAR1,A1,A2,A3]
-HOM[F,G,H]

NEG[HOM] NEG[MAP1] IMPCOND[TIMES]
```

49. AX33-1 RING[R,STAR,PLUS] -COMMRING[R,STAR,PLUS]

NEG[COMMRING] POS[RING]

50. AX33-2 IN[A,R] -COMMRING[R,STAR,PLUS]

NEG[COMMRING] POS[IN]

51. AX33-3 TIMES[STAR,B,A,C] -TIMES[STAR,A,B,C] -COMMRING[R,STAR,PLUS]

NEG[COMMRING] IMPCOND[TIMES]

52. AX34-1 COMMRING[R,STAR,PLUS] -RING[R,STAR,PLUS] -IN[SK15[PLUS,STAR,R],R] -IN[SK16[PLUS,STAR,R],R] -TIMES[STAR,SK16[PLUS,STAR,R],SK15[PLUS,STAR,R],SK17[PLUS,STAR,R?]]]

NEG[TIMES] NEG[IN] NEG[RING] POS[COMMRING]

53. AX34-2 COMMRING[R,STAR,PLUS] -RING[R,STAR,PLUS] -IN[SK15[PLUS,STAR,R],R] -IN[SK16[PLUS,STAR,R],R] TIMES[STAR,SK15[PLUS,STAR,R],SK16[PLUS,STAR,R],SK17[PLUS,STAR,R]?]]

POS[TIMES] NEG[IN] NEG[RING] POS[COMMRING]

54. AX35-1 RING[R,STAR,PLUS] -UNITRING[R,STAR,PLUS]

NEG[UNITRING] POS[RING]

55. AX35-2 IDENTITY[STAR,SK18[PLUS,STAR,R],R] -UNITRING[R,STAR,PLUS?]]

NEG[UNITRING] POS[IDENTITY]

56. AX36-1 UNITRING[R,STAR,PLUS] -RING[R,STAR,PLUS] -IDENTITY[STAR,E1,R]

NEG[IDENTITY] NEG[RING] POS[UNITRING]

57. AX38-1 COMMRING[R,STAR,PLUS] -RING[R,STAR,PLUS]
-COMMUTATIVE[STAR,R]

NEG[COMMUTATIVE] NEG[RING] POS[COMMRING]

58. AX39-1 COMMUTATIVE[PLUS,R] -RING[R,STAR,PLUS]

NEG[RING] POS[COMMUTATIVE]

59. AX40-1 RDIST[STAR,PLUS,R] -RING[R,STAR,PLUS]

NEG[RING] POS[RDIST]

60. AX41-1 LDIST[STAR,PLUS,R] -RING[R,STAR,PLUS]

NEG[RING] POS[LDIST]

61. AX42-1 NONEMPTY[R] -RING[R,STAR,PLUS]

NEG[RING] POS[NONEMPTY]

62. AX43-1 HOM[SK19[PLUS,STAR,Z,Y,X],Y,X] -FACTS[X,Y,Z]
-RING[Y,STAR,PLUS]

NEG[RING] NEG[FACTS] POS[HOM]

63. AX44-1 IDEAL[B,A,STAR,PLUS] -RING[A,STAR,PLUS]
-FACTS[X,A,B]

NEG[FACTS] NEG[RING] POS[IDEAL]

64. AX45-1 FACTS[SK20[PLUS,STAR,B,A],A,B]
-RING[A,STAR,PLUS] -IDEAL[B,A,STAR,PLUS]

NEG[IDEAL] NEG[RING] POS[FACTS]

65. AX46-1 RING[B,STAR,PLUS] -HOM[F,A,B] -RING[A,STAR,PLUS]

NEG[HOM] IMPCOND[RING]

66. AX47-1 RING[X,STAR,PLUS] -RING[A,STAR,PLUS]
-RING[B,STAR,PLUS] -INTERSECTION[X,A,B]

NEG[INTERSECTION] IMPCOND[RING]

67. AX49-1 TIMES[STAR,E,X,X] -IN[X,S] -IDENTITY[STAR,E,S]

NEG[IDENTITY] NEG[IN] POS[TIMES]

68. AX49-2 TIMES[STAR,X,E,X] -IN[X,S] -IDENTITY[STAR,E,S]

NEG[IDENTITY] NEG[IN] POS[TIMES]

69. AX50-1 IDENTITY[STAR,E,S] IN[SK21[S,E,STAR],S]

POS[IN] POS[IDENTITY]

70. AX50-2 IDENTITY[STAR,E,S]
-TIMES[STAR,SK21[S,E,STAR],E,SK21[S,E?] ,STAR]]
-TIMES[STAR,E,SK21[S,E,STAR],SK21[S,E,STAR]]

NEG[TIMES] POS[IDENTITY]

71. AX51-1 TIMES[STAR,B,A,SK73[B,A,STAR,S]] -IN[A,S]
-IN[B,S] -TIMES[STAR,A,B,SK73[B,A,STAR,S]]
-COMMUTATIVE[STAR,S]

NEG[COMMUTATIVE] NEG[IN] IMPCOND[TIMES]

72. AX52-1 COMMUTATIVE[STAR,S] IN[SK75[STAR,S],S]

POS[IN] POS[COMMUTATIVE]

73. AX52-2 COMMUTATIVE[STAR,S] IN[SK76[STAR,S],S]

POS[IN] POS[COMMUTATIVE]

74. AX52-3 COMMUTATIVE[STAR,S]
TIMES[STAR,SK75[STAR,S],SK76[STAR,S],SK77[STAR,S]] 7[STAR,S]]

POS[TIMES] POS[COMMUTATIVE]

75. AX52-4 COMMUTATIVE[STAR,S]
-TIMES[STAR,SK76[STAR,S],SK75[STAR,S],C]

NEG[TIMES] POS[COMMUTATIVE]

76. AX53-1 IN[SK25[X],X] -NONEMPTY[X]

NEG[NONEMPTY] POS[IN]

77. AX54-1 TIMES[STAR,A,W,V] -IN[A,S] -IN[B,S] -IN[C,S]
-IN[SK26[C,B,A,STAR,S],S] -IN[SK27[C,B,A,STAR,S],S]
-IN[SK28[C,B,A,STAR,S],S]
-TIMES[STAR,A,B,SK26[C,B,A,STAR,S]]
-TIMES[STAR,SK26[C,B,A,STAR,S],C,SK27[C,B,A,STAR,S]]
-TIMES[STAR,B,C,SK28[C,B,A,STAR,S]] -ASSOC[STAR,S]

NEG[ASSOC] NEG[IN] IMPCOND[TIMES]

78. AX54-2 TIMES[STAR,U,C,V] -IN[A,S] -IN[B,S] -IN[C,S]
-IN[SK29[C,B,A,STAR,S],S] -IN[SK30[C,B,A,STAR,S],S]
-IN[SK31[C,B,A,STAR,S],S]
-TIMES[STAR,B,C,SK31[C,B,A,STAR,S]]
-TIMES[STAR,A,SK31[C,B,A,STAR,S],SK30[C,B,A,STAR,S]]
-TIMES[STAR,A,B,SK29[C,B,A,STAR,S]] -ASSOC[STAR,S]

NEG[ASSOC] NEG[IN] IMPCOND[TIMES]

79. AX55-1 ASSOC[STAR,S] -RASSOC[STAR,S] -LASSOC[STAR,S]

NEG[LASSOC] NEG[RASSOC] POS[ASSOC]

80. AX56-1 TIMES[STAR,B,A,E] -IN[A,S] -IN[B,S]
-IDENTITY[STAR,E,S] -INVERSE[STAR,A,B]

NEG[INVERSE] NEG[IDENTITY] NEG[IN] POS[TIMES]

81. AX56-2 TIMES[STAR,A,B,E] -IN[A,S] -IN[B,S]
-IDENTITY[STAR,E,S] -INVERSE[STAR,A,B]

NEG[INVERSE] NEG[IDENTITY] NEG[IN] POS[TIMES]

82. AX57-1 INVERSE[STAR,A,B] IN[A,SK38[STAR,B,A]]

POS[IN] POS[INVERSE]

83. AX57-2 INVERSE[STAR,A,B] IN[B,SK38[STAR,B,A]]

POS[IN] POS[INVERSE]

84. AX57-3 INVERSE[STAR,A,B]
IDENTITY[STAR,SK39[STAR,B,A],SK38[STAR,B,A]]]

POS[IDENTITY] POS[INVERSE]

85. AX57-4 INVERSE[STAR,A,B]
-TIMES[STAR,A,B,SK39[STAR,B,A]]
-TIMES[STAR,B,A,SK39[STAR,B,A]]

NEG[TIMES] POS[INVERSE]

86. AX58-1 IN[C,S] -IN[A,S] -IN[B,S] -TIMES[STAR,A,B,C]
-CLOSED[STAR,S]

NEG[CLOSED] NEG[TIMES] IMPCOND[IN]

87. AX59-1 CLOSED[STAR,S] IN[SK40[S,STAR],S]

POS[IN] POS[CLOSED]

88. AX59-2 CLOSED[STAR,S] IN[SK41[S,STAR],S]

POS[IN] POS[CLOSED]

89. AX59-3 CLOSED[STAR,S]
TIMES[STAR,SK40[S,STAR],SK41[S,STAR],S?] K42[S,STAR]]

POS[TIMES] POS[CLOSED]

90. AX59-4 CLOSED[STAR,S] -IN[SK42[S,STAR],S]

NEG[IN] POS[CLOSED]

91. AX60-1 IN[X,C] -IN[X,A] -IN[X,B] -INTERSECTION[C,A,B]

NEG[INTERSECTION] IMPCOND[IN]

92. AX60-2 IN[X,B] -IN[X,C] -INTERSECTION[C,A,B]

NEG[INTERSECTION] IMPCOND[IN]

93. AX60-3 IN[X,A] -IN[X,C] -INTERSECTION[C,A,B]

NEG[INTERSECTION] IMPCOND[IN]

94. AX61-1 INTERSECTION[C,A,B] -IN[SK43[C,B,A],C]
-IN[SK44[C,B,A],A] -IN[SK44[C,B,A],B]

NEG[IN] POS[INTERSECTION]

95. AX61-2 INTERSECTION[C,A,B] -IN[SK43[C,B,A],C]
IN[SK44[C,B,A],C]

IMPCOND[IN] POS[INTERSECTION]

96, AX61-3 INTERSECTION[C,A,B] IN[SK43[C,B,A],B] -IN[SK44[C,B,A],A] -IN[SK44[C,B,A],B]

IMPCOND[IN] POS[INTERSECTION]

97, AX61-4 INTERSECTION[C,A,B] IN[SK43[C,B,A],B] IN[SK44[C,B,A],C]

POS[IN] POS[INTERSECTION]

98, AX61-5 INTERSECTION[C,A,B] IN[SK43[C,B,A],A] -IN[SK44[C,B,A],A] -IN[SK44[C,B,A],B]

IMPCOND[IN] POS[INTERSECTION]

99, AX61-6 INTERSECTION[C,A,B] IN[SK43[C,B,A],A] IN[SK44[C,B,A],C]

POS[IN] POS[INTERSECTION]

100, AX62-1 IN[A1,A] -MAP[F,A,B] -MAP1[F,A1,B1]

NEG[MAP1] NEG[MAP] POS[IN]

101, AX62-2 IN[B1,B] -MAP[F,A,B] -MAP1[F,A1,B1]

NEG[MAP1] NEG[MAP] POS[IN]

102, AX63-1 CLOSED[STAR,S] -WELLDEF[STAR,S]

NEG[WELLDEF] POS[CLOSED]

103, AX63-2 TIMES[STAR,A,B,SK45[B,A,S,STAR]] -IN[A,S] -IN[B,S] -WELLDEF[STAR,S]

NEG[WELLDEF] NEG[IN] POS[TIMES]

```
104.      AX64-1      WELLDEF[STAR,S]      -CLOSED[STAR,S]
-TIMES[STAR,SK46[S,STAR],SK47[S,STAR],C]

NEG[TIMES] NEG[CLOSED] POS[WELLDEF]


105.      AX64-2      WELLDEF[STAR,S]      -CLOSED[STAR,S]
IN[SK47[S,STAR],S]

POS[IN] NEG[CLOSED] POS[WELLDEF]


106.      AX64-3      WELLDEF[STAR,S]      -CLOSED[STAR,S]
IN[SK46[S,STAR],S]

POS[IN] NEG[CLOSED] POS[WELLDEF]


107. AX65-1 IN[SK48[X,B,A,F],B] -MAP[F,A,B] -IN[X,A]

NEG[MAP] IMPCOND[IN]


108. AX65-2 MAP1[F,X,SK48[X,B,A,F]] -MAP[F,A,B] -IN[X,A]

NEG[IN] NEG[MAP] POS[MAP1]


109. AX66-1 IN[Y,B] -HOM[PHI,A,B] -IN[X,A] -MAP1[PHI,X,Y]

NEG[MAP1] NEG[HOM] IMPCOND[IN]


110.      AX67-1      MAP1[PSI,Y,X]      -HOM[PHI,A,B]      -IN[X,A]
-MAP1[PHI,X,Y] -INVERSE[COMP,PSI,PHI]

NEG[INVERSE] NEG[IN] NEG[HOM] IMPCOND[MAP1]


111. AX68-1 NORMAL[M,G,STAR] -PNSUBGROUP[M,G,STAR]

NEG[PNSUBGROUP] POS[NORMAL]
```

112. AX68-2 -IDENTITY[STAR,M,G] -PNSUBGROUP[M,G,STAR]
NEG[PNSUBGROUP] NEG[IDENTITY]

113. AX68-3 -EQUALSET[M,G] -PNSUBGROUP[M,G,STAR]
NEG[PNSUBGROUP] NEG[EQUALSET]

114. AX69-1 INVERSE[STAR,G,F] -INVERSE[STAR,F,G]
IMPCOND[INVERSE]

115. AX70-1 INVERSE[COMP,F,G] -HOM[F,A,B] -HOM[G,B,A]
NEG[HOM] POS[INVERSE]

116. AX71-1 HOM[SK49[B,A,PHI],B,A] -HOM[PHI,A,B]
IMPCOND[HOM]

117. AX73-1 HOM[SK50[C,B,A,G,F],A,C] -HOM[F,A,B]
-HOM[G,B,C]
IMPCOND[HOM]

118. AX74-1 INTERSECTION[X,Z,Y] -INTERSECTION[X,Y,Z]
IMPCOND[INTERSECTION]

119. AX75-1 COMMRING[R,STAR,PLUS] -C1RING[R,STAR,PLUS]
NEG[C1RING] POS[COMMRING]

120. AX75-2 UNITRING[R,STAR,PLUS] -C1RING[R,STAR,PLUS]
NEG[C1RING] POS[UNITRING]

121. AX76-1 C1RING[R,STAR,PLUS] -COMMRING[R,STAR,PLUS]
-UNITRING[R,STAR,PLUS]

NEG[UNITRING] NEG[COMMRING] POS[C1RING]

122. AX77-1 DIST[STAR1,STAR2,S] -LDIST[STAR1,STAR2,S]
-RDIST[STAR1,STAR2,S]

NEG[RDIST] NEG[LDIST] POS[DIST]

123. AX78-1 CLOSED[STAR,S] -RASSOC[STAR,S]

NEG[RASSOC] POS[CLOSED]

124. AX78-2 TIMES[STAR,A,V,W] -IN[A,S] -IN[B,S] -IN[C,S]
-TIMES[STAR,A,B,U] -TIMES[STAR,U,C,V] -TIMES[STAR,B,C,W]
-RASSOC[STAR,S]

NEG[RASSOC] NEG[IN] IMPCOND[TIMES]

125. AX79-1 RASSOC[STAR,S] -CLOSED[STAR,S]
-TIMES[STAR,SK66[S,STAR],SK70[S,STAR],SK71[S,STAR]]

NEG[TIMES] NEG[CLOSED] POS[RASSOC]

126. AX79-2 RASSOC[STAR,S] -CLOSED[STAR,S]
TIMES[STAR,SK67[S,STAR],SK68[S,STAR],SK71[S,STAR]]

POS[TIMES] NEG[CLOSED] POS[RASSOC]

127. AX79-3 RASSOC[STAR,S] -CLOSED[STAR,S]
TIMES[STAR,SK69[S,STAR],SK68[S,STAR],SK70[S,STAR]]

POS[TIMES] NEG[CLOSED] POS[RASSOC]

128, AX79-4 RASSOC[STAR,S] -CLOSED[STAR,S]
TIMES[STAR,SK66[S,STAR],SK67[S,STAR],SK69[S,STAR]]

POS[TIMES] NEG[CLOSED] POS[RASSOC]

129, AX79-5 RASSOC[STAR,S] -CLOSED[STAR,S]
IN[SK68[S,STAR],S]

POS[IN] NEG[CLOSED] POS[RASSOC]

130, AX79-6 RASSOC[STAR,S] -CLOSED[STAR,S]
IN[SK67[S,STAR],S]

POS[IN] NEG[CLOSED] POS[RASSOC]

131, AX79-7 RASSOC[STAR,S] -CLOSED[STAR,S]
IN[SK66[S,STAR],S]

POS[IN] NEG[CLOSED] POS[RASSOC]

132, AX80-1 CLOSED[STAR,S] -LASSOC[STAR,S]

NEG[LASSOC] POS[CLOSED]

133, AX80-2 TIMES[STAR,U,C,V] -IN[A,S] -IN[B,S] -IN[C,S]
-TIMES[STAR,B,C,W] -TIMES[STAR,A,W,V] -TIMES[STAR,A,B,U]
-LASSOC[STAR,S]

NEG[LASSOC] NEG[IN] IMPCOND[TIMES]

134, AX81-1 LASSOC[STAR,S] -CLOSED[STAR,S]
-TIMES[STAR,SK63[S,STAR],SK62[S,STAR],SK64[S,STAR]]

NEG[TIMES] NEG[CLOSED] POS[LASSOC]

135, AX81-2 LASSOC[STAR,S] -CLOSED[STAR,S]
TIMES[STAR,SK60[S,STAR],SK61[S,STAR],SK63[S,STAR]]

POS[TIMES] NEG[CLOSED] POS[LASSOC]

136, AX81-3 LASSOC[STAR,S] -CLOSED[STAR,S]
TIMES[STAR,SK60[S,STAR],SK65[S,STAR],SK64[S,STAR]]

POS[TIMES] NEG[CLOSED] POS[LASSOC]

137, AX81-4 LASSOC[STAR,S] -CLOSED[STAR,S]
TIMES[STAR,SK61[S,STAR],SK62[S,STAR],SK65[S,STAR]]

POS[TIMES] NEG[CLOSED] POS[LASSOC]

138, AX81-5 LASSOC[STAR,S] -CLOSED[STAR,S]
IN[SK62[S,STAR],S]

POS[IN] NEG[CLOSED] POS[LASSOC]

139, AX81-6 LASSOC[STAR,S] -CLOSED[STAR,S]
IN[SK61[S,STAR],S]

POS[IN] NEG[CLOSED] POS[LASSOC]

140, AX81-7 LASSOC[STAR,S] -CLOSED[STAR,S]
IN[SK60[S,STAR],S]

POS[IN] NEG[CLOSED] POS[LASSOC]

141, AX83-1 COMMUTATIVE[STAR,G] -ABELIAN[G,STAR]

NEG[ABELIAN] POS[COMMUTATIVE]

142, AX83-2 GROUP[G,STAR] -ABELIAN[G,STAR]

NEG[ABELIAN] POS[GROUP]

143, AX86-1 TIMES[STAR2,B1,B2,B3] -MAP1[F,A1,B1]
-MAP1[F,A2,B2] -MAP1[F,A3,B3] -TIMES[STAR1,A1,A2,A3]
-GROUP[G,STAR1] -GROUP[H,STAR2] -HOM[F,G,H]

NEG[HOM] NEG[GROUP] NEG[MAP1] IMPCOND[TIMES]

144, AX87-1 IN[SK87[X],X] -NONEMPTY[X]

NEG[NONEMPTY] POS[IN]

145, AX90-1 MAP[COMP[F,INVERSE[F]],B,B] -HOM[F,A,B]

NEG[HOM] POS[MAP]

146, AX92-1 MAP_1[F,INVERSEF[Y,A,B,F],Y] -IN[Y,B] -MAP[F,A,B]

NEG[MAP] NEG[IN] POS[MAP_1]

147, AX92-2 IN[INVERSEF[Y,A,B,F],A] -IN[Y,B] -MAP[F,A,B]

NEG[MAP] IMPCOND[IN]

148, AX93-1 MAP_1[F,X,MAPF[X,A,B,F]] -IN[X,A] -MAP[F,A,B]

NEG[MAP] NEG[IN] POS[MAP_1]

149, AX93-2 IN[MAPF[X,A,B,F],B] -IN[X,A] -MAP[F,A,B]

NEG[MAP] IMPCOND[IN]

150, AX94-1 IN[A1,A] -MAP[F,A,B] -MAP_1[F,A1,B1]

NEG[MAP_1] NEG[MAP] POS[IN]

151, AX94-2 IN[B1,B] -MAP[F,A,B] -MAP_1[F,A1,B1]

NEG[MAP_1] NEG[MAP] POS[IN]

152, AX95-1 IN[X,A] -MAP[PHI,A,B] -MAP_1[PHI,X,Y]

NEG[MAP_1] NEG[MAP] POS[IN]

153, AX95-2 IN[Y,B] -MAP[PHI,A,B] -MAP1[PHI,X,Y]

NEG[MAP_1] NEG[MAP] POS[IN]

154, AX96-1 MAP_1[COMP[F,INVERSE[F]],Y,Y] -MAP[F,A,B] -IN[Y,B] -MAP1[F,INVERSEF[Y,A,B,F],Z]

NEG[IN] NEG[MAP] IMPCOND[MAP_1]

155, AX97-1 IN[Y,B] -MAP[PHI,A,B] -IN[X,A] -MAP1[PHI,X,Y]

NEG[MAP1] NEG[MAP] IMPCOND[IN]

156, AX98-1 IN[X,A] -MAP[PHI,A,B] -IN[Y,B] -MAP_1[PHI,X,Y]

NEG[MAP_1] NEG[MAP] IMPCOND[IN]

157, AX99-1 GROUP[G,STAR] -MAXIMAL[M,G,STAR]

NEG[MAXIMAL] POS[GROUP]

158, AX99-2 PNORMAL[M,G,STAR] -MAXIMAL[M,G,STAR]

NEG[MAXIMAL] POS[PNORMAL]

159, AX99-3 -PSUBSET[M,OTHERSET[STAR,G,M]] -PNORMAL[OTHERSET[STAR,G,M],G,STAR] -MAXIMAL[M,G,STAR]

NEG[MAXIMAL] NEG[PNORMAL] NEG[PSUBSET]

160, AX100-1 MAXIMAL[M,G,STAR] -GROUP[G,STAR] -PNORMAL[M,G,STAR] PNORMAL[OTHERSET[STAR,G,M],G,STAR]

IMPCOND[PNORMAL] NEG[GROUP] POS[MAXIMAL]

161, AX100-2 MAXIMAL[M,G,STAR] -GROUP[G,STAR] -PNORMAL[M,G,STAR] PSUBSET[M,OTHERSET[STAR,G,M]]

POS[PSUBSET] NEG[PNORMAL] NEG[GROUP] POS[MAXIMAL]

170. AX105-1 HOM[HOMMAP[STAR,N,G,X],G,X] -GROUP[G,STAR]
-NORMAL[N,G,STAR] -FACTS[X,G,N]

NEG[FACTS] NEG[NORMAL] NEG[GROUP] POS[HOM]

171. AX106-1 SUBGROUP[H,G,STAR] -GROUP[H,STAR]
-GROUP[G,STAR] -SUBSET[H,G]

NEG[SUBSET] NEG[GROUP] POS[SUBGROUP]

172. AX107-1 -IDENTITY[STAR2,Y,B] -HOM[F,A,B]
-GROUP[A,STAR1] -GROUP[B,STAR2] -MAP[F,X,Y]
IDENTITY[STAR1,X,A]

NEG[MAP] NEG[GROUP] NEG[HOM] IMPCOND[IDENTITY]

173. AX108-1 IDENTITY[STAR2,Y,B] -HOM[PHI,A,B]
-GROUP[A,STAR1] -GROUP[B,STAR2] -MAP[PHI,X,Y]
-IDENTITY[STAR1,X,A]

NEG[MAP] NEG[GROUP] NEG[HOM] IMPCOND[IDENTITY]

174. AX109-1 PSUBSET[Y,B] -HOM[PHI,A,B] -GROUP[A,STAR1]
-GROUP[B,STAR2] -MAP[PHI,X,Y] -PSUBSET[X,A]

NEG[MAP] NEG[GROUP] NEG[HOM] IMPCOND[PSUBSET]

175. AX110-1 NORMAL[Y,B,STAR2] -GROUP[A,STAR1]
-GROUP[B,STAR2] -HOM[PHI,A,B] -MAP[PHI,X,Y]
-NORMAL[X,A,STAR1]

NEG[MAP] NEG[HOM] NEG[GROUP] IMPCOND[NORMAL]

176. AX111-1 NORMAL[X,G,STAR] -PNORMAL[X,G,STAR]

NEG[PNORMAL] POS[NORMAL]

177. AX111-2 PSUBSET[X,G] -PNORMAL[X,G,STAR]

NEG[PNORMAL] POS[PSUBSET]

178. AX111-3 -IDENTITY[STAR,X,G] -PNORMAL[X,G,STAR]

NEG[PNORMAL] NEG[IDENTITY]

179. AX112-1 PNORMAL[X,G,STAR] -NORMAL[X,G,STAR] -PSUBSET[X,G] IDENTITY[STAR,X,G]

POS[IDENTITY] NEG[PSUBSET] NEG[NORMAL] POS[PNORMAL]

180. AX113-1 SUBSET[X,Y] -PSUBSET[X,Y]

NEG[PSUBSET] POS[SUBSET]

181. AX113-2 -EQUALSET[X,Y] -PSUBSET[X,Y]

NEG[PSUBSET] NEG[EQUALSET]

182. AX114-1 SUBSET[X,Y] -PSUBSET[X,Y]

NEG[PSUBSET] POS[SUBSET]

183. AX115-1 PSUBSET[X,Y] -SUBSET[X,Y] EQUALSET[X,Y]

POS[EQUALSET] NEG[SUBSET] POS[PSUBSET]

184. AX116-1 PSUBSET[X,Z] -PSUBSET[X,Y] -PSUBSET[Y,Z]

IMPCOND[PSUBSET]

185. AX117-1 SUBGROUP[X,G,STAR] -NORMAL[X,G,STAR]

NEG[NORMAL] POS[SUBGROUP]

162. AX101-1 GROUP[H,STAR] -SUBGROUP[H,G,STAR]

NEG[SUBGROUP] POS[GROUP]

163. AX101-2 GROUP[G,STAR] -SUBGROUP[H,G,STAR]

NEG[SUBGROUP] POS[GROUP]

164. AX101-3 SUBSET[H,G] -SUBGROUP[H,G,STAR]

NEG[SUBGROUP] POS[SUBSET]

165. AX102-1 GROUP[G,STAR] -SIMPLEGROUP[G,STAR]

NEG[SIMPLEGROUP] POS[GROUP]

166. AX102-2 -NORMAL[X,G,STAR] IDENTITY[STAR,X,G] -PSUBSET[X,G] -SIMPLEGROUP[G,STAR]

NEG[SIMPLEGROUP] NEG[PSUBSET] POS[IDENTITY] NEG[NORMAL]

167. AX103-1 SIMPLEGROUP[G,STAR] -GROUP[G,STAR] NORMAL[SK91[STAR,G],G,STAR]

POS[NORMAL] NEG[GROUP] POS[SIMPLEGROUP]

168. AX103-2 SIMPLEGROUP[G,STAR] -GROUP[G,STAR] -IDENTITY[STAR,SK91[STAR,G],G]

NEG[IDENTITY] NEG[GROUP] POS[SIMPLEGROUP]

169. AX103-3 SIMPLEGROUP[G,STAR] -GROUP[G,STAR] PSUBSET[SK91[STAR,G],G]

POS[PSUBSET] NEG[GROUP] POS[SIMPLEGROUP]

186, AX118-1 MAP[PHI,X,MAPF[X,B,A,PHI]] -HOM[PHI,A,B] -SUBSET[X,A]

NEG[SUBSET] NEG[HOM] POS[MAP]

187, AX118-2 SUBSET[MAPF[X,B,A,PHI],B] -HOM[PHI,A,B] -SUBSET[X,A]

NEG[HOM] IMPCOND[SUBSET]

188, AX119-1 IDENTITY[FACTSETOP[X,STAR,G,N],N,X] -NORMAL[N,G,STAR] -FACTS[X,G,N]

NEG[FACTS] NEG[NORMAL] POS[IDENTITY]

189, AX120-1 EQUALSET[X,G] -NORMAL[N,G,STAR] -FACTS[X,G,N] -IDENTITY[STAR,N,G]

NEG[IDENTITY] NEG[FACTS] NEG[NORMAL] POS[EQUALSET]

190, AX121-1 IDENTITY[STAR,X,G] -NORMAL[N,G,STAR] -FACTS[X,G,N] -EQUALSET[X,G]

NEG[EQUALSET] NEG[FACTS] NEG[NORMAL] POS[IDENTITY]

191, AX122-1 NORMAL[G,G,STAR]

POS[NORMAL]

192, AX123-1 EQUALSET[X,X]

POS[EQUALSET]

193, AX124-1 ABELIAN[X,STAR] -ABSUBGROUP[X,Y,STAR]

NEG[ABSUBGROUP] POS[ABELIAN]

194. AX124-2 SUBGROUP[X,Y,STAR] -ABSUBGROUP[X,Y,STAR]

NEG[ABSUBGROUP] POS[SUBGROUP]

195. AX125-1 ABSUBGROUP[X,Y,STAR] -ABELIAN[X,STAR] -SUBGROUP[X,Y,STAR]

NEG[SUBGROUP] NEG[ABELIAN] POS[ABSUBGROUP]

196. AX126-1 COMMRING[X,STAR,PLUS] -COMMSUBRING[X,Y,STAR,PLUS]

NEG[COMMSUBRING] POS[COMMRING]

197. AX126-2 SUBRING[X,Y,STAR,PLUS] -COMMSUBRING[X,Y,STAR,PLUS]

NEG[COMMSUBRING] POS[SUBRING]

198. AX127-1 COMMSUBRING[X,Y,STAR,PLUS] -COMMRING[X,STAR,PLUS] -SUBRING[X,Y,STAR,PLUS]

NEG[SUBRING] NEG[COMMRING] POS[COMMSUBRING]

199. AX128-1 SUBRING[M,R,STAR,PLUS] -IDEAL[M,R,STAR,PLUS]

NEG[IDEAL] POS[SUBRING]

200. AX128-2 IN[B_2,M] -IN[A,M] -IN[B,M] -IN[X,R] -TIMES[STAR,X,A,X_1] -TIMES[STAR,A,X,X_2] -INVERSE[PLUS,B_1,B] -TIMES[PLUS,B,B_1,B_2] -IDEAL[M,R,STAR,PLUS]

NEG[IDEAL] NEG[INVERSE] NEG[TIMES] IMPCOND[IN]

201. AX128-3 IN[X2,M] -IN[A,M] -IN[B,M] -IN[X,R] -TIMES[STAR,X,A,X_1] -TIMES[STAR,A,X,X_2] -INVERSE[PLUS,B_1,B] -TIMES[PLUS,B,B_1,B_2] -IDEAL[M,R,STAR,PLUS]

NEG[IDEAL] NEG[INVERSE] NEG[TIMES] IMPCOND[IN]

202. AX128-4 IN[X_1,M] -IN[A,M] -IN[B,M] -IN[X,R] -TIMES[STAR,X,A,X_1] -TIMES[STAR,A,X,X_2] -INVERSE[PLUS,B_1,B] -TIMES[PLUS,B,$B1$,B_2] -IDEAL[M,R,STAR,PLUS]

NEG[IDEAL] NEG[INVERSE] NEG[TIMES] IMPCOND[IN]

203. AX129-1 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS] -IN[SK98[PLUS,STAR,R,M],M] -IN[SK99[PLUS,STAR,R,M],M] -IN[SK96[PLUS,STAR,R,M],M]

NEG[IN] NEG[SUBRING] POS[IDEAL]

204. AX129-2 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS] TIMES[PLUS,SK94[PLUS,STAR,R,M],SK95[PLUS,STAR,R,M],SK96[PLUS ,STA?] R,R,M]]

POS[TIMES] NEG[SUBRING] POS[IDEAL]

205. AX129-3 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS] INVERSE[PLUS,SK95[PLUS,STAR,R,M],SK94[PLUS,STAR,R,M]]

POS[INVERSE] NEG[SUBRING] POS[IDEAL]

206. AX129-4 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS] TIMES[STAR,SK93[PLUS,STAR,R,M],SK97[PLUS,STAR,R,M],SK99[PLUS ,STA?] R,R,M]]

POS[TIMES] NEG[SUBRING] POS[IDEAL]

207. AX129-5 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS] TIMES[STAR,SK97[PLUS,STAR,R,M],SK93[PLUS,STAR,R,M],SK98[PLUS ,STA?] R,R,M]]

POS[TIMES] NEG[SUBRING] POS[IDEAL]

208. AX129-6 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS] IN[SK97[PLUS,STAR,R,M],R]

POS[IN] NEG[SUBRING] POS[IDEAL]

209. AX129-7 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS]
IN[SK94[PLUS,STAR,R,M],M]

POS[IN] NEG[SUBRING] POS[IDEAL]

210. AX129-8 IDEAL[M,R,STAR,PLUS] -SUBRING[M,R,STAR,PLUS]
IN[SK93[PLUS,STAR,R,M],M]

POS[IN] NEG[SUBRING] POS[IDEAL]

211. AX130-1 RING[R,STAR,PLUS] -MAXRING[A,R,STAR,PLUS]

NEG[MAXRING] POS[RING]

212. AX130-2 PIDEAL[A,R,STAR,PLUS] -MAXRING[A,R,STAR,PLUS]

NEG[MAXRING] POS[PIDEAL]

213. AX130-3 -PSUBSET[A,OTHERSET[STAR,PLUS,A,R]]
-PIDEAL[OTHERSE?] T[STAR,PLUS,A,R],R,STAR,PLUS]
-MAXRING[A,R,STAR,PLUS]

NEG[MAXRING] NEG[PIDEAL] NEG[PSUBSET]

214. AX131-1 MAXRING[A,R,STAR,PLUS] -RING[R,STAR,PLUS]
-PIDEAL[A,R,STAR,PLUS]
PIDEAL[OTHERSET[STAR,PLUS,A,R],R,STAR,PL?] US]

IMPCOND[PIDEAL] NEG[RING] POS[MAXRING]

215. AX131-2 MAXRING[A,R,STAR,PLUS] -RING[R,STAR,PLUS]
-PIDEAL[A,R,STAR,PLUS] PSUBSET[A,OTHERSET[STAR,PLUS,A,R]]
POS[PSUBSET] NEG[PIDEAL] NEG[RING] POS[MAXRING]

216. AX132-1 PSUBSET[X,B] -HOM[PHI,A,B]
-RING[A,STAR1,PLUS1] -RING[B,STAR2,PLUS2] -MAP[PHI,X,Y]
-PSUBSET[X,A]

NEG[MAP] NEG[RING] NEG[HOM] IMPCOND[PSUBSET]

217, AX133-1 IDENTITY[STAR2,Y,B] -RING[A,STAR1,PLUS1] -RING[B,STAR2,PLUS2] -HOM[PHI,A,B] -MAP[PHI,X,Y] -IDENTITY[STAR1,X,A]

NEG[MAP] NEG[HOM] NEG[RING] IMPCOND[IDENTITY]

218, AX134-1 IDENTITYLFACTSETOP1[X,STAR,PLUS,R,A],A,X] -IDEAL[A,R,S?] TAR,PLUS] -FACTS[X,R,A]

NEG[FACTS] NEG[IDEAL] POS[IDENTITY]

219, AX135-1 IDEAL[A,R,STAR,PLUS] -PIDEAL[A,R,STAR,PLUS]

NEG[PIDEAL] POS[IDEAL]

220, AX135-2 PSUBSET[A,R,STAR,PLUS] -PIDEAL[A,R,STAR,PLUS]

NEG[PIDEAL] POS[PSUBSET]

221, AX135-3 -IDENTITY[STAR,A,R] -PIDEAL[A,R,STAR,PLUS]

NEG[PIDEAL] NEG[IDENTITY]

222, AX136-1 PIDEAL[A,R,STAR,PLUS] -IDEAL[A,R,STAR,PLUS] -PSUBSET[A,R,STAR,PLUS] IDENTITY[STAR,A,R]

POS[IDENTITY] NEG[PSUBSET] NEG[IDEAL] POS[PIDEAL]

223, AX137-1 RING[R,STAR,PLUS] -SIMPLERING[R,STAR,PLUS]

NEG[SIMPLERING] POS[RING]

224, AX137-2 -IDEAL[Y,R,STAR,PLUS] IDENTITY[STAR,Y,R] -PSUBSET[Y,R] -SIMPLERING[R,STAR,PLUS]

NEG[SIMPLERING] NEG[PSUBSET] POS[IDENTITY] NEG[IDEAL]

225. $_{A}$X138-1 SIMPLERING[R,STAR,PLUS] -RING[R,STAR,PLUS] IDEAL[SK102[PLUS,STAR,R],R,STAR,PLUS]

POS[IDEAL] NEG[RING] POS[SIMPLERING]

226. $_{A}$X138-2 SIMPLERING[R,STAR,PLUS] -RING[R,STAR,PLUS] -IDENTITY[STAR,SK102[PLUS,STAR,R],R]

NEG[IDENTITY] NEG[RING] POS[SIMPLERING]

227. $_{A}$X138-3 SIMPLERING[R,STAR,PLUS] -RING[R,STAR,PLUS] PSUBSET[SK102[PLUS,STAR,R],R]

POS[PSUBSET] NEG[RING] POS[SIMPLERING]

228. AX142-1 COMMUTATIVE[STAR,R] -COMMRING[R,STAR,PLUS]

NEG[COMMRING] POS[COMMUTATIVE]

229. AX142-2 RING[R,STAR,PLUS] -COMMRING[R,STAR,PLUS]

NEG[COMMRING] POS[RING]

230. $_{A}$X143-1 COMMRING[R,STAR,PLUS] -COMMUTATIVE[STAR,R] -RING[R,STAR,PLUS]

NEG[RING] NEG[COMMUTATIVE] POS[COMMRING]

231. AX148-1 RING[A,STAR,PLUS] -SUBRING[A,B,STAR,PLUS]

NEG[SUBRING] POS[RING]

232. AX148-2 RING[B,STAR,PLUS] -SUBRING[A,B,STAR,PLUS]

NEG[SUBRING] POS[RING]

233, AX148-3 SUBSET[A,B] -SUBRING[A,B,STAR,PLUS]

NEG[SUBRING] POS[SUBSET]

234, AX158-1 -IDENTITY[PLUS2,Y,B] -HOM[F,A,B] -RING[A,STAR1,PLUS1] -RING[B,STAR2,PLUS2] -MAP[F,X,Y] IDENTITY[PLUS1,X,A]

NEG[MAP] NEG[RING] NEG[HOM] IMPCOND[IDENTITY]

235, AX159-1 -IDENTITY[STAR2,Y,B] -HOM[F,A,B] -RING[A,STAR1,PLUS1] -RING[B,STAR2,PLUS2] -MAP[F,X,Y] IDENTITY[STAR1,X,A]

NEG[MAP] NEG[RING] NEG[HOM] IMPCOND[IDENTITY]

236, AX160-1 IDEAL[BB,AA,STAR,PLUS] -RING[A,STAR,PLUS] -IDEAL[B,A,STAR,PLUS] -HOM[F,A,AA] -MAP[F,B,BB]

NEG[MAP] NEG[HOM] NEG[RING] IMPCOND[IDEAL]

237, AX161-1 MAP[F,A1,IMAGE[A1,B,A,F]] -HOM[F,A,B] -SUBSET[A1,A]

NEG[SUBSET] NEG[HOM] POS[MAP]

238, AX162-1 GROUP[B,STAR2] -HOM[PHI,A,B] -GROUP[A,STAR1]

NEG[HOM] IMPCOND[GROUP]

239, AX165-1 HOM[F,ASUB,BSUB] -HOM[F,A,B] -SUBSET[ASUB,A] -MAP[F,ASUB,BSUB]

NEG[MAP] NEG[SUBSET] IMPCOND[HOM]

APPENDIX C

ZORBA-I AS A USER SYSTEM

ZORBA-I has been implemented in LISP on a Digital Equipment Corporation PDP-10 Computer used in an interactive (time-sharing) mode. The (interpreted) system uses 100K of LISP words which are divided as follows:

25K	Basic LISP System;
10K	Special LISP Trace Package;
15K	(QA3) Functions used to maintain the data base, prenex wffs into clauses;
25K	ALGBASE loaded into QA3's data base;
15K	ZORBA-I including I/O for analogies;
10K	Freespace for running programs.
100K	Total

I designed ZORBA-I empirically. Algorithms were coded for the information flows I understood. When I was in doubt or didn't know how to handle a particular decision, I would program a break point that enables me to communicate with the LISP EVAL at that point in the program. I could interrogate the state of ZORBA-I, edit functions, execute various I/O operations, and continue running the program in order to design the needed sections of the ZORBA-I algorithms.

Without such flexible interactive facility, it is doubtful that ZORBA-I could have been developed at all.

In principle, ZORBA-I and QA3 are intimately linked while in practice they are barely connected.

In principle, the operating procedure for ZORBA-I would be:

(1) Load the ZORBA-I system on the PDP-10.

(2) Select a theorem T_A to prove and an analogous theorem T that

has been proved.

(3) Load the data-base, e.g. ALGRASE from disk or tape.

(4) Load the resolution tree (saved from the run of QA3 when T was proved) from disk or tape.

(5) Execute ZORBA-I[T_A;T] to create an analogy α.

(6) Delete all clauses from the data base which do not appear in the image of α^c.

(7) Call QA3 to prove T_A using the restricted data base.

Two practical reasons inhibit these last two steps in the implemented system:

(1) The version of QA3 that runs on the PDP-10 is incompatible with the QA3 memory structures which ZORBA-I inherited from its initial implementation on the SDS-940. Substantial work would need to be done to render the two compatible again.

(2) The full blown QA3 system would demand an additional 15K of code and possibly additional freespace. The resultant system would be ~115K in size and would exceed the memory capacity of the PDP-10 LISP system at Stanford Research Institute. In practice, QA3 could be loaded after ZORBA-I was run by deleting the ZORBA-I code after use. QA3's code would be deleted and ZORBA-I's code would be reloaded to run the next analogy problem. Consequently, ZORBA-I has been run as an independent system from QA3. Two theorems (T_2 and T_4) were run on the PDP-10 version of QA3 using the data bases output by ZORBA-I after it generated an analogy with T_1 and T_3 respectively. The QA3 search was, in fact, quite small (~ 100 clauses) and it

found the two proofs easily. Other theorems in the experimental set required special QA3 strategies that were not converted to the PDP-10. Consequently, the impact of ZORBA-I on restricting their QA3 search was not explictly tested.

REFERENCES

1. R. E. Kling, "Reasoning by Analogy as an Aid to Heuristic Theorem Proving," Presented at the IFIP Conference 1971, Ljubljana, Yugoslavia, August 23-28, 1971.

2. R. E. Kling, "A Paradigm for Reasoning by Analogy," to appear in Artificial Intelligence.

3. Max Wertheimer, Productive Thinking, (Harper, 1959).

4. G. Polya, Induction and Analogy in Mathematics, (Princeton University Press, Princeton, N. J. 1954).

5. G. Polya, Mathematical Discovery, Vols. I and II, (John Wiley and Sons, New York, 1962).

6. H. G. Birch, "The Relation of Previous Experience to Insightful Problem Solving," J. Comp. Psych., Vol.38, pp. 367-383 (1945).

7. J. R. Quinlan, "A Task-Independent Experience Gathering Scheme for a Problem Solver," Proceedings of the International Joint Conference on Artificial Intelligence, D. E. Walker and L. M. Norton, Eds., Washington, D. C., 7-9 May 1969.

8. J. R. Slagle and C. D. Farrell, "Experiments in Automatic Learning for a Multipurpose Heuristic Program," J. ACM, Vol. 14, No. 2, pp. 91-98 (1971).

9. D. A. Waterman, "Generalization Learning Techniques for Automating the Learning of Heuristics," Artificial Intelligence, Vol.I, No.1, Spring, (1970).

10. A. Newell, J. C. Shaw, and H. A. Simon, "Elements of a Theory of Human Problem Solving," Psychological Review, Vol.65, pp.151-166, (1958).

11. Rudolf Arnheim, Visual Thinking, (University of California Press, Berkeley, California, 1969).

12. D. Bobrow, "A Question-Answering System for High School Algebra Word Problems," AFIPS Conference Proceedings, Vol. 26, part I, (Spartan Books, New York, 1964).

13. J. M. Paige and H. A. Simon, "Cognitive Processes in Solving Algebra Word Problems" in Problem Solving: Research Method and Theory, B. Kleinmontz, Ed. (John Wiley and Sons, 1966).

14. John W. Gyr, John S. Brown, Richmond Wiley, and Arthur Zivian, "Computer Simulation and Psychological Theories of Perception," Psychological Bulletin, Vol. 65, No. 3, pp. 174-192, (1966).

15. T. G. Evans, "A Heuristic Program to Solve Geometric-Analogy Problems," Ph.D. Thesis, Department of Mathematics, M.I.T., (May 1963).

16. J. Becker, "The Modelling of Simple Analogic and Inductive Processes in a Semantic Memory System," Proceedings of the International Joint Conference on Artificial Intelligence, D. E. Walker and L. M. Norton, Eds., Washington, D. C., 7-9 May 1969.

17. N. Nilsson, Problem Solving Methods in Artificial Intelligence, (McGraw Hill Book Co., 1971).

18. A. Newell, J. Shaw, and H. A. Simon, "Empirical Exploration with the Logic Theory Machine," Computers and Thought.

19. R. Dreistadt, "The Use of Pictorial Analogies and Incubation in Obtaining Insights in Creative Problem Solving," Journal of Psychology, Vol. 71, No. 2, pp. 159-175 (1969).

20 G. A. Davis, "Current Status of Research and Theory in Human Problem Solving," Psychological Bulletin, Vol. 66, No. 1, pp.36-54 (1966).

21. A. DiVesta, "Transfer of Solution Rules in Problem Solving," J. of Ed. Psych., Vol. 58, No. 6, Part 1, pp. 319-326 (1967).

22. A. Newell, "On the Analysis of Human Problem Solving Protocols," Artificial Intelligence Report, Carnegie Institute of Technology, Pittsburgh, Pennsylvania (1966).

23. G. Polya, Induction and Analogy in Mathematics, Ibid., p. 13.

24. C. Green, "Applications of Theorem Proving to Problem Solving," Proceedings of the International Joint Conference on Artificial Intelligence, Ibid.

25. R. D. Charmichael, Introduction to the Theory of Groups of Finite Order, (Dover Publications, 1956).

26. E. J. Sandewall, "Concepts and Methods for Heuristic Search," Proceedings of the International Joint Conference on Artificial Intelligence, D. E. Walker and L. M. Norton, Eds., Washington, D. C., 7-9 May 1969.

27. A. Newell and H. A. Simon, "GPS, A Problem-Solving Program that Simulates Human Thought," Computer and Thought.

28. H. Gelernter, "Realization of a Geometry Theorem-Proving Machine," Computers and Thought, E. Feigenbaum and J. Feldman, Eds., (McGraw Hill Book Co., 1963).

29. R. E. Fikes, "REF-ARF: A System for Solving Problems Stated as Procedures," Artificial Intelligence, Vol. 1, Number 1, (Spring 1970).

30. C. Hewitt, "PLANNER: A Language for Theorem Proving in Robots," Proceedings of the International Joint Conference on Artificial Intelligence, D. E. Walker and L. M. Norton, Eds., Washington, D. C., 7-9 May 1969.

31. J. F. Rulifson, R. A. Waldringer, and J. A. Derksen, "A Language for Writing Problem-Solving Programs," to be presented at the IFIP Congress 1971, Ljubljana, Yugoslavia, August 23-28, 1971.

32. C. C. Green, "Theorem-Proving by Resolution as a Basis for Question-Answering Systems," Machine Intelligence 4, B. Meltzer and D. Michie, Eds. (American Elsevier, N. Y., 1969).

33. G. Ernst, "Some Issues of Representation in a General Problem Solver," Proceedings of the Spring Joint Computer Conference, 1967).

34. C. L. Chang, "The Unit Proof and the Input Proof in Theorem Proving," J. ACM, Vol. 17, No. 4, pp. 698-707.