A. I. WORKING PAPER 132 October 1976

REASONING BY ANALOGY

A Progress Report

by

Richard Brown

ABSTRACT--Rather.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643. The views expressed are necessarily (and perhaps only) those of the author.

Brown? Isn't he working on resolution theorem proving? The blasphemer should be shot. Not only that -- he mumbles about geometry. Gelernter and Goldstein were bad enough, but this loser works with axiomatic geometry -- non-Euclidian axiomatic geometry. Ugh! Next thing you know, he'll be in Hilbert Spaces. As a matter of fact, he did mention Hilbert yesterday. God save us!

--overheard in a paranoid delusion

PREFACE

This report concerns some research I have been doing in the past year on the problem of reasoning by analogy. Specifically, I have been working on a process that will learn how to solve (very) simple problems in solid incidence geometry, given a predicate calculus description of solid geometry and an expert problem solver in plane incidence geometry. The research has entailed work on computational logic (which is to mathematical logic as computational linguistics is to linguistics), development of an "expert problem solver" formalism which is simple enough to be understood and manipulated by the analogy process, and finally the analogy process itself. This progress report will deal almost exclusively with the last topic.

There. I admit it. I make use of the predicate calculus. I have found that predicate calculus does not make your hair fall out, nor does it result in blindness.

Plane and solid incidence geometry are not Euclidian. As to what they *are*, I can give two very different answers to that question. I can either tell you what lines and planes *are*, but leave you in the dark as to how they are manipulated, or I can tell you how to manipulate them efficiently without ever saying what they *are*.

This report is mathematically challenging. There are unsolved problems in the mathematical and meta-mathematical background of our investigations. The domains of geometry we are interested in are non-trivial. This should not put the reader off -- we mortals will never delve too deeply in matters best left to the gods.

This report contains five sections, including the preface. In the introduction we will

take a quick look at the ground rules and framework of the research.

In the section on axioms we develop descriptions of two problem domains: plane geometry and solid geometry. This section is fairly heavy; you have been warned.

The next section shows the analogy process in action. Three solid geomery problems and their solutions are presented in detail. This section is the core of the report.

The final brief section is a summery. We will review the new ideas which have resulted from the research to date.

SETTING

Suppose one is given an expert problem solving system in some domain, say plane geometry. Suppose further that one has a desire for another expert problem solver in a different domain, say solid geometry. One would like to have this second expert constructed automatically by some process. Analogy is such a process.

An analogy is a map from one problem domain to another. It can usually be extended to become a map from an expert problem solver in the first domain to an expert problem solver in the image domain. It is this extended analogy in which we are exclusively interested.

For our purposes, an expert problem solver contains three ingredients: (1) A single data base into which only true assertions are entered. (2) A set of objects which are manipulated by the expert. An object has further structure: a type and a representation. (3) A set of pattern-invoked procedures for forward and backward chained deductions, and for representation manipulation.

For example, in solid geometry we have objects of type "point", "line", and "plane". The representation of an object of type "line" is an ordered list of objects of type "point". The representation of a "plane" is an unordered list of objects of type "point" and of type "line". Integers might be represented by their prime decomposition. In algebra one might have an object of type "finite abelian group" with a representation that is an unordered list of prime--power pairs. Another type of object might be a "finite field" with representation a prime group with a set of adjoined elements (or element--polynomial pairs) following. In solving cryptoarithmetic problems a fairly good representation for a symbol might be the set of digits that the symbol cannot be.

One of our major concerns in this research was to use analogy to "teach" an expert problem solver to use new special-purpose representations. One of the problems solved was to

develop the representation of a plane "by analogy" to the representation of a line.

For our expert problem solvers, a problem is a predicate calculus statement (containing only one IMPLIES) which is to be proven. A solution is a complete, rigorous proof of the statement. One may ask the expert problem solver to "expand" some step in a proof, so that a correct solution can be expanded to a proof where each step is justified by "given" or "by axiom such-and-such".

Expert Limitations

Even with this level of description we can say something about the problem solving capabilities of our experts. First, they cannot produce proofs "by contradiction". A simple problem that requires such a proof is

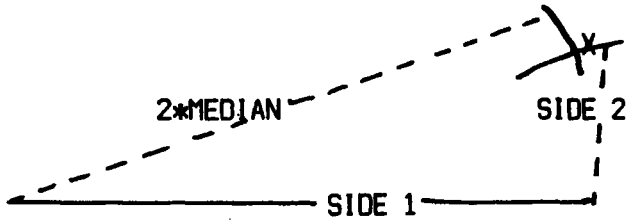
```
GIVEN:
(IMPLIES A (AND B C))
(IMPLIES B (OR E F))
(IMPLIES C (OR E G))
(IFF F (NOT G))

PROVE:
(IMPLIES A E)
```

A second result we can give concerns constructions: If we insist that all line representations be canonical (as do Goldstein, Nevins, and Ullman) then no constructions can be made. Furthermore, if constructions are disallowed, all problems become trivial in the sense that only a (small) finite number of true assertions can ever be made about a given configuration. Thus, in particular, we can show Nevins' discussion of forward chaining vs. backward chaining to be vacuous.

We informally define a class of constructions, the "trivial constructions", to consist of those whose form could constitute the name of the object constructed. (This definition can be

made precise in terms of representations and types.) For example, if A and B are points, then the line containing them can be named (LINE A B), so the LINE function performs a trivial construction. If two lines K and L happen to have a point in common (or can be proven non-parallel), then (INTERSECT K L) is also a trivial construction. We insist that our geometry expert be able to do trivial constructions.

Examples of non-trivial constuctions are locus constructions (the locus of all points equidistant from a given pair of points is a line or a plane) and the construction of the point X in the problem "Given two sides of a triangle and the median to the third, construct a triangle"



INTRODUCTION TO ANALOGY PROCESS

The analogy process has six steps:

1. Construct an analogy map from the "problem" domain to the image domain (i.e., from solid geometry to plane geometry).

2. Translate the original problem to a "similar" problem in the image domain. We don't want the solution of this "image" problem as much as we want the *way* the expert problem solver in the image domain solved the problem.

3. Solve the image problem. We will carefully observe how the problem was solved.

4. Apply the inverse map to the image solution. By solution we mean first the "answer" (a particular line or point, or perhaps a truth value), and then a proof that this solution is correct

5. Apply the inverse map to the image procedures responsible for the image solution. This gives us additional expertise in the "problem" domain, i.e., we have learned something from solving the problem.

6. Solve the original problem. In most cases, this is just to show off the new code in the "problem" domain expert.

In general, step 1 does not fully specify the analogy map. Step 4 is a debugging step which may further specify the analogy map. The inverse map is usually not one-to-one. Step 5 is the step we are really interested in. Analogy does not produce a solution -- it writes a program which in turn produces a solution (in step 6). One should keep in mind that, in response to one problem, several different analogies may be used in sequence on various sub-problems.

Domain Description

The analogy process makes use of expert problem solvers in two domains. It also requires descriptions of these two domains. The expert problem solver in the "domain of expertise" constitutes a highly biased partial description of its domain. Indeed, the purpose of "reasoning by analogy" is to learn those biases. However, the analogy process requires a complete description of the domain, and we would prefer a more neutral description.

There are two radically different forms a domain description can take: declarative and imperative. A declarative description is (by definition) complete. It is *almost* impossible to use such a description for problem solving.

> Green demonstrated that it is not *completely* impossible in his QA3 system. We exploit the problem solving/theorem proving duality he developed in the *correct* direction: using problem solving expertise to do theorem proving.

On the other hand, an imperative description of a domain can easily be used to solve problems in that domain. Unfortunately this type of description is generally incomplete, sometimes inaccurate, and (the main flaw) difficult to produce.

The Point of Analogy

The purpose of analogy is to translate a declarative description (useless but easy to give) to an imperative description (useful, but a pain to develop). It uses both declarative and imperative description of (presumably) analogous domains for inspiration.

RESULTS

1. We claim that an analogy is a map. Where does this map come from? We have a technique for generating the analogy maps directly from the axioms describing the domain.

2. The analogy process must be a good copier. If the imperatives associated with one domain are *completely* applicable to another domain, then they should be copied. In the first example we will show that our theory copies only what is both needed and applicable. Irrevelant junk is ignored.

3. The analogy process must be a good debugger. If imperatives in one domain are needed to describe a second, but those imperatives are not quite applicable, they must be adapted. The second example illustrates the way we associate "bugs" in proofs with "bugs" in programs. This aspect of our theory of analogy depends on having a strong notion of what constitutes a "proof".

4. Occasionally an expert problem solver will grind to a halt on a problem. This may be due to the problem being insoluble, or due to incompleteness of the imperative description. The latter signals a job for analogy. In order to get anywhere, we need to be able to *summarize* a log jam. The second example shows how this summary process works, and illustrates that the key idea is the use of *representations*.

5. The analogy process must be an innovator. Suppose that some aspect of a domain has *no* analogous aspect in our "domain of expertise". A specific example is encountered in problem 3. Our theory of analogy also solves this problem. The amazing thing about this particular dancing bear is not that it dances at all, but that it does so gracefully!

Many of our manipulations may appear to be overly syntactic. Do not be deceived! A major result (illustrated in the first example) is the lack of dependence on surface similarity (or dissimilarity) in domain descriptions.

> *In place of "point", "line", and "plane" we must at all times be able to say "beer mug", "table", and "chair".*
>
> --- David Hilbert

We are going to describe the two domain we deal with. In plane geometry we also know how to manipulate (solve problems involving) the entities described below. In solid geometry, we have *only* the declarations (axioms) given below.

The description is declarative. We know what IN-LN allows us to conclude, and we know some facts which allow us to conclude IN-PL is true of two objects. What does IN-PL check? The description doesn't really say: implementation is left to the reader. In this case the "reader" is our analogy process.

In what follows, we will blandly assume EQUAL and DISTINCT are understood. In fact, analogy can build the machinery to handle equality and non-equality, but the process is not very exciting.

> Axioms within each domain are grouped into classes I and II. Within each group they are numbered. Plane geometry axioms are preceeded with a P, while solid geometry axioms are preceeded with S. We will give the predicate calculus versions of the axioms used in the problems. The rest are given for completeness.

PLANE Geometry Axioms

P-I1.*Given two points, there is a line that contains them.*

```
(FORALL (A B) (IMPLIES (AND (PT A) (PT B))
                       (AND (LN (LINE A B))
                            (IN-LN (LINE A B) A)
                            (IN-LN (LINE A B) B))))
```

Note that we don't insist that the two points be distinct. This claims that two points determine at least one line.

P-I2. ***For every two distinct points, no more than one line contains them.***

```
(FORALL (A B)
     (IMPLIES (AND (DISTINCT A B)
                   (PT A)
                   (PT B))
              (NOT (EXISTS (X Y) (AND (DISTINCT X Y)
                                      (LN X)
                                      (LN Y)
                                      (IN-LN X A)
                                      (IN-LN Y A)
                                      (IN-LN X B)
                                      (IN-LN Y B))))))
```

P-I3a. ***Each line contains at least two points.***

P-I3b. ***There are at least three non-colinear points.***

We also have a set of axioms dealing with the concept of "order". These are included, even though our examples are in incidence geometry, because of the important role they play in the representation of a line.

P-II1a. ***The BTWN relation implies that the points are co-linear.***

```
(FORALL (A B C)
     (IMPLIES (AND (PT A) (PT B) (PT C) (BTWN A B C))
              (EXISTS (L) (AND (LN L)
                               (IN-LN L A)
                               (IN-LN L B)
                               (IN-LN L C)))))
```

P-II1b. ***The order of the BTWN arguments may be reversed.***

```
(FORALL (A B C)
        (IMPLIES (AND (PT A) (PT B) (PT C) (BTWN A B C))
                 (BTWN C B A)))
```

P-II2. ***For two points A and C there always exists at least one point B on the line containing A and C such that C lies between A and B.***

P-II3. ***Of any three points on a line there exists no more than one that lies between the other two.***

In addition to axioms, we also include definitions. In fact, we allow two kinds of definitions: new functions and predicates, and new object types, but the latter form of definition does not concern us here.

For various reasons, we disallow definitions which introduce any new knowledge -- definitions are strictly and purely notational. Enforcing this edict is harder than one might think. For example, we have the definition of INTERSECT:

P-DEF1. *Define a function INTERSECT. Insist that it take two distinct lines as arguments. If this condition is met, then the result is in both of the given lines.*

```
(DETERMINES (INTERSECT A B)
            ((LN A) (LN B) (DISTINCT A B))
            ((IN-LN A (INTERSECT A B))
             (IN-LN B (INTERSECT A B))))
```

The form for the function call is followed by a list of restrictions on the arguments (i.e., the function is only defined if the arguments meet these restrictions), and then a list of claims about the value returned by the function.

This definition requires some explanation. We have not declared the type of the result of applying the INTERSECT function to two distinct lines. We have only given the minimal properties we wish this returned object to have.

Suppose we were to say that it is a "point" and that under the assumptions given it is unique. Further suppose (AND (DISTINCT A B) (PT A) (PT B)). Then by P-I1 we have a line X = (LINE A B). Suppose there were a distinct line Y such that (IN-LN Y A) and (IN-LN Y B). Then (INTERSECT X Y) should "return" points A and B and by it being unique, A = B, contradiction! Thus such a line Y does not exist. Indeed, we know that it does not, but we have just proven this fact without using axiom P-I2. This should not be surprising because the contrapositive of P-I2 is the proof of uniqueness. In other words, our "definition" really contains an axiom. We must disallow such definitions.

Instead we insist that for any type Q (Q may be, for example, PT, LN, PL), if the intersection of two lines is of type Q (i.e., (Q (INTERSECT A B)) is true for lines A and B), then no other object of type Q can be in both lines. We can then prove that if two distinct lines have

two points X and Y in common, then X = Y. Hence the intersection of these two lines is the point X (or the same point under the name Y).

We are forced to define INTERSECT this way by purely logical considerations. However, this definition also makes reasoning about intersection by analogy easier. If we uniformly replace IN-LN by IN-PL, we will be able to prove in solid geometry that the intersection of two planes is not a point, and that it is a line. A very syntactic and natural transformation is all that is required to "lift" the definition.

SOLID Geometry Axioms

S-I1,S-I2,S-I3a,S-I3b,S-II1a,S-II1b,S-II2,S-II3 are all identical to P-etc.

S-DEF1. *We define a predicate of three arguments to be true if and only if the three arguments are points and there is no line containing all of them.*

```
(DEF-PRED (NON-LN A B C)
          (AND (PT A)
               (PT B)
               (PT C)
               (NOT (IN-LN (LINE A B) C))))
```

S-I4a.*For three non-colinear points there is always a plane containing them.*

```
(FORALL (A B C)
        (IMPLIES (NON-LN A B C)
                 (AND (PL (PLANE A B C))
                      (IN-PL (PLANE A B C) A)
                      (IN-PL (PLANE A B C) B)
                      (IN-PL (PLANE A B C) C))))
```

S-I4b.*Every plane contains at least one point.*

```
(FORALL (P) (IMPLIES (PL P)
                     (EXISTS (A) (AND (PT A) (IN-PL P A)))))
```

S-I5. *For three non-colinear points, no more than one plane contains them.*

S-I6. *If two points of a line are in a plane, then all points in that line are in the plane.*

S-I7. *If there is one point in two distinct planes, then there is a second distinct point also in both planes.*

Note that INTERSECT is not defined in solid geometry. The above five axioms *completely* describe what a plane is. They don't give even a hint about how planes should be represented or manipulated by a program -- analogy must figure that out for itself.

SEMANTIC TEMPLATES

The analogy program initially examines the solid geometry axioms above, and notes the presence of three unary predicates PT, LN, and PL. It then assumes that these are type-checking predicates with associated type names PT, LN, and PL. Having done this, we can go through the axioms to discover the argument types expected by the rest of the predicates and the functions:

```
(IN-LN LN PT)
(IN-PL PL PT)
(PLANE PT PT PT)
(LINE PT PT)
```

These patterns are called "semantic templates", after a similar but less powerful notion developed by Kling (Ph.D. Thesis "Reasoning by Analogy with Applications to Heuristic Problem Solving"). Similar information is collected from the solid geometry axioms. This information is used to develop the initial analogy map.

We admit that these semantic templates are derived using fairly syntactic operations. Suppose we change terminology, and in plane geometry call points "*beer mugs*", and lines "*tables*". We would then have (in plane geometry) type checking predicates (say) TBL and MUG, with templates

```
(ON-TABLE TBL MUG)
(TABLE MUG MUG)
```

PROBLEM 1

```
(FORALL (K L C) (IMPLIES (AND (LN L K)
                              (PT C)
                              (IN-LN K C)
                              (IN-LN L C))
                         (EQUAL C (INTERSECT K L))))
```

Several observations should be made. First, note that the LN predicate has two arguments. Input to the problem solver is processed by special procedures. Since we know that LN is a type checking predicate, we know how to deal with this form -- it is expanded to

```
(LN L), (LN K), (NOT (EQUAL L K))
```

The second observation is that INTERSECT has never been defined in solid geometry.

The bare-bones solid geometry expert knows how to deal with problems of the above form: first create objects named K, L, and C. Then assert (LN L) ... (IN-LN L C) into the data base. Finally, it tries to evaluate

```
(EQUAL C (INTERSECT K L))
```

by first evaluating the function call

```
(INTERSECT K L).
```

Since there is no procedure willing to perform this computation, the solid geometry expert reports a failure, and analogy takes over.

The most we can expect from analogy is:
1. Learn what INTERSECT means.
2. Learn how to implement a function that computes INTERSECT.
3. Learn how to react to assertions of the form
   (IN-LN line point).
4. Learn how to represent lines.

Although this is the most we can expect, analogy gives us a little bit more!

We note that the problem concerns (so far) only objects of type LN and of type PT. We further note that the plane geometry expert has objects with the same type names. We apply the SAME NAME heuristic and set up the analogy map

LN -> LN
PT -> PT

We then start dumping the contents of the data base out to the plane geometry expert. There are six assertions in the data base. No trouble arises until we encounter (IN-LN K C).

We examine the type requirements of the IN-LN predicate in both plane and solid geometry, find them compatible. We add

IN-LN -> IN-LN

to the analogy, completing the data base transfer.

Plane geometry is fully developed. When each assertion is entered in the data base, it is examined by a procedure (invoked by the pattern of the assertion). This has occured, and the data base is now empty. The assertions have been used to set up objects and their representations. We have

```
K       type=LN, representation=(C)
L       type=LN, representation=(C)
C       type=PT, no representation
OB1     type=DBUCKET, representation=(K L)
```

where the last object is a "distinctness bucket".

We now evaluate (INTERSECT K L), and get a return value C. We apply the inverse analogy map to the object C, and get the "upstairs" object C. We evaluate (EQUAL C C), get "TRUE" as a result. This indicates it would be worth while "lifting" the definition of INTERSECT and the proof that "C" is the correct value.

We lift the definition of INTERSECT by applying the inverse analogy map to it (trivial in this case). We then lift the proof, which depends on a random plane geometry theorem we will call P-THM22, the contrapositive of axiom P-I2.

```
1. (IN-LN K (INTERSECT K L)) definition of INTERSECT
2. (IN-LN L (INTERSECT K L)) definition of INTERSECT
3. (EQUAL C (INTERSECT K L)) P-THM22 applied to above
```

We need to know if this theorem is also true in solid geometry. We apply the inverse map to P-THM22, getting (in solid geometry)

```
S-THM15
(FORALL (A B X Y)
        (IMPLIES (AND (LN X) (LN Y) (DISTINCT X Y) (PT A) (PT B)
                      (IN-LN X A) (IN-LN X B) (IN-LN Y A) (IN-LN Y B))
                 (EQUAL A B)))
```

We try to prove this upstairs, and fail. We determine that analogy is not likely to be helpful (at least not this analogy map), so we resort to "brute force". We could *in principle* use some uniform proof procedure (e.g., resolution), but the cost is prohibitive. Instead we try to find a one-step proof by examining each axiom and previously proven theorem and its contrapositive to see if the desired result can be shown. We succeed with the contrapositive of S-I2.

> We claim that "brute force" will never get worse than a simple one-step deduction. We will never need to use anything like resolution theorem proving in the analogy process. We have already explained that the expert problem solver does not use theorem proving in any of its activity. Nor do we use theorem proving (in the general sense) in obtaining from these experts the proofs analogy uses.

Since we know S-THM15 to be true, we can complete the lifted proof and add the following to the analogy map for use in the next step.

S-THM15 -> P-THM22
INTERSECT -> INTERSECT
S-I2 -> P-I2

Beer mugs on Tables?

Using beer mugs and tables for points and lines respectively in the plane geometry axioms stops us from employing the SAME NAME heuristic. Oh well, we only have two choices for an initial map: PT->MUG or PT->TBL. Then LN must (presumably) go to the other. IN-LN then goes to a function of two arguments: TBL and MUG. Fortunately, there is no problem, since only ON-TABLE takes these two arguments. (Had there been more, the semantic template for INTERSECT derived from the problem statement would be brought into play). Thus PT->MUG and LN->TBL. Everything above then follows without alteration.

Messing around with names doesn't affect the analogy process as much as one might expect. (There is, however, a rather interesting possibility that the analogy process will try inventing projective geometry).

REPRESENTATION THEOREMS

We now lift all the programs which contributed lines to the lifted proof, and the correctness proofs associated with those programs (checking the lifted versions for correctness, naturally). Our loot includes a "representation theorem" which states that if the representation of a line L matches the pattern

(* X *) * will match any sequence

then we may conclude (IN-LN L X). The proof is by induction on the length of the representation (the details need not concern us). This theorem is a first step towards learning the representation for a line.

Another role "representation theorems" play concerns "extended" predicate forms. We find it convenient to write (IN-LN L A B) instead of saying both (IN-LN L A) (IN-LN L B). This

representation theorem will allow us to directly translate a statement

(IN-LN L A B C D)

to giving the object L of type LN the representation

(A B C D)

in the current solid geometry expert. The notion is that the last "argument" can be replaced by a list of arguments, provided that (1) a representation theorem exist for the particular predicate, and (2) the pattern of the representation theorem is being matched against the additional arguments. Thus, in plane geometry we have a representation theorem (also about the representation for lines) stating

(* X * Y * Z *) -> (BTWN X Y Z)

so that we can interpret

```
(BTWN A B C D) -> (BTWN A B C)
                  (BTWN A B D)
                  (BTWN A C D)
                  (BTWN B C D).
```

Similar translations operate in the "dumping" process, so that if we have an object L of type LN and representation=(E F G H) we write this out as

(IN-LN L E F G H)

in solid geometry, and as

(IN-LN L E F G H) and (BTWN E F G H)

in plane geometry. On read-in, if both assertions are present we can go directly to the representation without going through the "expansion" and deduction stages.

Lessons from Problem 1

What have we learned as a result of solving this problem? As much as could be expected. We have both a definition and code for intersections of lines (under some circumstances), We have the beginnings of code dealing with representations of lines. We also know what to do with IN-LN assertions of *multiple* point arguments.

PROBLEM 2

We now turn to our second problem. Let us assume that, by processes similar to that given above, all the necessary expertise in dealing with points and lines has been learned by the solid geometry expert. We wish to prove that

```
(FORALL (A B C P)
        (IMPLIES (AND (PT A B C)
                      (NOT (EQUAL (LINE A B) (LINE B C)))
                      (PL P)
                      (IN-PL P A B C))
                 (EQUAL (PLANE A B C) P)))
```

There are several points to note. We used an extended format for PT (which we know how to deal with) and for IN-PL (which we don't). There are trivial construction of both lines (understood) and planes (not understood). Finally, the problem mentions points, lines, and planes, so the analogy map must be non-trivial.

Processing is forced to halt when we try to assert

```
(IN-PL P A B C)
```

Recall that IN-PL is expected to have two arguments, the first of type PL and the second of type PT. This form does not match its semantic template.

We must use analogy to find what this assertion means. Having no good reason to abandon the analogy map used in problem 1, we continue with

PT -> PT
LN -> LN
etc.

To find a mapping of IN-PL, we ask "What predicate do we have in plane geometry that takes two arguments, one of them of type PT and the other one of another type?" One answer is IN-LN. But this has an object of type LN as its other argument, so on the basis of matching semantic templates we conclude

PL -> LN
IN-PL -> IN-LN

and add this to the map.

Construction of Analogous Problems Involves SUMMARIZING

We are forced to solve a problem in plane geometry to know how to treat the expression (IN-PL P A B C) in solid geometry. If we simply map *everything* we know down to plane geometry, we will end up with a contradiction: (IN-LN P A B C) and the claim (DISTINCT (LINE A B)(LINE B C)) conflict with axiom P-12. This means we must summarize the current situation into a sub-problem. The next question is "What sub-problem?" Upstairs we have objects:

```
A       type=PT
B       type=PT
C       type=PT
OB1     type=DBUCKET, rep=(A B C)
OB2     type=LN, rep=(A B)
OB3     type=LN, rep=(B C)
OB4     type=DBUCKET, rep=(OB2 OB3)
```

Since we are using a direct deduction system (as opposed to resolution) we can always add more assertions if a proof does not develop. We also have access to the current deduction "tree", so we can see if there are any interesting outstanding questions, should a proof fail to materialize downstairs. These two considerations tell us that postponing transfer of assertions to plane geometry won't cost us anything, and may be beneficial. We therefore set up a "distance" metric, and progressively make more assertions about more objects. We start off with the objects in the current interesting assertion (i.e., P, A, B, and C), their representations, and type declarations resulting from relevant distinctness buckets (i.e., OB1).

These assertions constitute the "given" portion of the sub-problem constructed by the summarization process. The "to prove" portion is supplied by the assertion solid geometry couldn't deal with. Continuing, we assert in plane geometry

```
(PT A B C)
(LN P)
(IN-LN P A B C)
```

The last assertion would give P the representation (A B C) if the assertion (BTWN A B C) were also present. Since it isn't present, we expand the IN-LN assertion to

(IN-LN P A) (IN-LN P B) (IN-LN P C)

The first two are then removed from the data base, while the object P is given the representation (A B). The third assertion causes an attempt to prove or disprove the three statements

(BTWN A B C) (BTWN A C B) (BTWN C A B).

Naturally no progress is made on any of these, so we continue summarizing.

We need to map the current upstairs goal

(EQUAL (PLANE A B C) P)

to an appropriate goal in plane geometry. Finding no exact match for the mapped semantic template, we note that the arguments to PLANE are of homogeneous type, and that the semantic template for LINE downstairs is also of homogeneous argument type, with the appropriate (under

the map) value type. Thus we add

PLANE -> LINE

with a note to *arbitrarily* drop the last argument.

Then, noting that (PLANE A B C) -> (LINE A B), it is trivial to evaluate the latter expression in plane geometry to get the object P, and by golly

(EQUAL P P)

The proof rests on LINELEMMA1 (proven by using P-I2):

```
(FORALL (P1 P2 L1 L2)
        (IMPLIES (AND (PT P1 P2)
                      (LN L1)
                      (LN L2)
                      (IN-LN L1 P1 P2)
                      (IN-LN L2 P1 P2))
                 (EQUAL L1 L2)))
```

The downstairs proof reads

```
1. (PT A B)          given
2. (LN L)            given
3. (IN-LN L A B)     given
4. (LN (LINE A B)) P-I1 applied to 1
5. (IN-LN (LINE A B) A B)   P-I1 applied to 1
6. (EQUAL L (LINE A B))     LINELEMMA1 applied to above
```

We have now solved the summarized problem. Unfortunately, if we try to "lift" the solution, we find it is not correct!

DEBUGGING the Analogy

The first hint of trouble occurs when we try to justify step 4* (PL (PLANE A B C)). To do this, we need to lift P-I1 as it was used in this step:

```
I1* (FORALL (A B C)
        (IMPLIES (AND (PT A) (PT B)) (PL (PLANE A B C))))
```

This theorem is not true, but no matter, because we try using brute force, and fail to prove I1*. Good -- we have detected a bug! We do find, however, that we can prove (PL (PLANE A B C)) in one step using S-I4a. The "lifted" portion of plane geometry allows us to prove in solid geometry that (NON-LN A B C). (The appropriate portion of plane geometry would be lifted by this exercise in any case).

We can now classify the "bug" in the analogy to be a MISSING-PREREQUISIT (after Sussman). With this in mind, we add

S-I4a -> P-I1

to the analogy map, and a line providing non-colinearity to the proof in solid geometry.

We proceed to lift step 5 as

(IN-PL (PLANE A B C) A B) *S-I4a applied to NON-LN*

after checking that S-I4a does indeed prove this. We now need to lift LINELEMMA1 (above) in preparation for lifting step 6. This lemma in turn depends on axiom P-I2.

Downstairs LINELEMMA1 is proven by using a refutation proof. The same is unfortunately true upstairs. We are looking for a one-step proof, so we don't need to use a resolution theorem prover. We simply translate the lifted version of LINELEMMA1 to disjunctive normal form, then compare this to all axioms (also in their disjunctive normal form). We discover that axiom S-I5 gives us the desired equality provided that the points are not collinear and that

all three points are in both planes. We just proved the former, and know the latter is true by S-I4a (appropriate assertions were made when it was applied).

We make note of a second MISSING-PREREQUISITE bug on our bug list, and give the proof for

```
PLANELEMMA1:
(FORALL (P1 P2 P3 PL1 PL2)
        (IMPLIES (AND (PT P1 P2 P3)
                      (NON-LN P1 P2 P3)
                      (PL PL1) (PL PL2)
                      (IN-PL PL1 P1 P2 P3)
                      (IN-PL PL2 P1 P2 P3))
                 (EQUAL PL1 PL2)))
```

and also add

PLANELEMMA1 -> LINELEMMA1
S-I5 -> P-I2

to the analogy map. Note we cleverly got back the proper number of points: LINELEMMA1 quantified two points, while PLANELEMMA1 quantified three!

This completes the proof. We are still not ready to lift code. We have one anomaly remaining: in plane geometry there is an outstanding question concerning the order of points A, B, and C in the "line" P. We can easily prove in solid geometry that

(NOT (OR (BTWN A B C) (BTWN A C B) (BTWN C A B))

We can conclude that there is not an "obvious" candidate in solid geometry for the BTWN relation with this analogy. We thus note an UNNECESSARY-PREREQUISIT bug on the bug list.

Lifting Code for Problem 2

We can now lift the code. Although the bugs were detected by logical means, they are noted with respect to the code fragments which gave rise to them. When we lift these questionable code fragments, the bug type tells us what actions need to be taken to repair the code.

We also lift a representation theorem about the representation of planes:

P's rep=(* X *) -> (IN-PL P X)

We learned from this example a little about representing planes, and how to construct a plane from given points. We also learned that there is no concept corresponding to BTWN which applies to points in a plane. It is important to remember in all this that by "learn" we always mean "write code for".

PROBLEM 3

So far, the problems have been interesting, but not spectacular in that the analogies were fairly obvious. The problem we will now solve has no obvious solution.

The problem involves the notion of a line being in a plane. We recognize that "IN" is a transitive, non-symetric binary relation in solid geometry: if A is IN B, and B is IN C, then A is IN C, but if A is in B, then B is not necessarily in A (almost certainly not). The crux of the problem is that there are *no* transitive, non-symetric binary relations in plane geometry (as we have described it). That we call "IN" by different names according to argument types just makes things worse. In this example we will find an analogy where none can reasonably exist.

The above example indicates we could replace both IN-LN and IN-PL in solid geometry with a single predicate IN without affecting the analogy process operation.

It is also worth pointing out that we never need some higher level descriptions like "transitive, anti-symetric binary relation."

Suppose we wish to introduce the notion of "a line being in a plane" to our budding solid geometry expert. We cannot say merely "if a point is in a line and that line is in a plane, then the point is in the plane" because that only tells us how to use a line being in a plane, not how to deduce it.

We might wish to add "a line is in a plane if all points on that line are in the plane", which is correct, but testing for this condition involves a proof by contradiction. As mentioned earlier, we dislike proofs by contradiction. Therefore we might try to add "a line is in a plane if two points of that line are in the plane". This, of course, duplicates axiom S-I6 in a definition, and thus cannot be allowed.

What we will do is similar to the device used with INTERSECT: we will claim that for line L and plane P, (IN-PL P L) is a predicate such that

```
(FORALL (A L P)
        (IMPLIES (AND (PT A) (LN L) (PL P)
                      (IN-LN L A) (IN-PL P L))
                 (IN-PL P A)))
```

In other words, the above is true "by definition." This is a "second-order" definition, because it implies

```
(FORALL (A L P ALPHA)
        (IMPLIES (IMPLIES (AND (PT A) (LN L) (PL P)
                               (IN-LN L A) (ALPHA P L))
                          (IN-PL P A))
                 (IMPLIES (ALPHA P L) (IN-PL P L))))
```

We have our definition. To find how the defined predicate is to be implemented, we use a clever trick: we pose the definition as a problem! We continue using the same analogy developed by problems 1 and 2, so that

PT -> PT
LN -> LN
PL -> LN
IN-PL -> IN-LN when applied to plane and point

The semantic type of IN-PL applied to plane and line under the current analogy is a predicate applied to two lines, i.e., the same semantic type as EQUAL (applied to lines). With the axioms given, it is also the only semantic template match. In fact, there are four reasons why EQUAL is a good choice for the analogy:

(1) Pragmatic. This choice works.
(2) Tradition. In algebra one investigates the structure of groups and rings by way of maps that send problematic substructures to identity elements, i.e., one imposes equivalence relations on the structure.
(3) Conjectural. Suppose we wanted to choose P to maximize the size (cardinality) of the set "Q such that for all X,Y (P X Y) implies (Q X Y)". I suggest that EQUAL would be one of the best choices. In other words, EQUAL "does more" than any other predicate.
(4). Philosophical. We really want to write a program. A common joke is that writing programs is the same as debugging a blank sheet of paper. We are essentially using EQUAL as a blank stimulus in the hopes of debugging the response.

So we add

IN-PL -> EQUAL when applied to planes and lines.

To make a long story short, the image problem is solved with the proof being "by definition of equality". When this justification (a reference to a second-order equality axiom) is lifted, we get a proof "by definition of IN-PL" which is indeed correct!

We now lift the programs. Ignoring the details, the downstairs program first searched P's representation for A, and failed. It then searched for any "line" EQUAL to P, found L, and proceeded to search L for A.

Thus the "lifted" program should first search P's representation for the object A, and failing that, search for any line (because L's inverse is a line) satisfying the relation (IN-PL P x) where x is a line, and then search there for A.

The downstairs program has already been lifted before by problem 2, and this is known to the analogy process. We therefore can implement the new procedural knowledge as a patch! What more could one ask for?

So we learn two things by solving this problem: first that the use of the "definition" should be in backward chaining rather than forward chaining, and second the point and details of the patch required to implement the new interpretation of IN-PL. Now if

(IN-PL plane line)

is ever asserted, we know what to do.

Problem 3, Continued

Thus armed, suppose we are given the problem

```
(FORALL (A B C P)
        (IMPLIES (AND (PT A B C)
                      (PL P)
                      (IN-PL P A B)
                      (IN-LN (LINE A B) C))
                 (IN-PL P C)))
```

Under the current analogy (which we might as well continue using) we will show (in plane geometry) that (IN-LN P C) because (EQUAL P (LINE A B)). Lifting this reasoning, we find that (IN-PL P C) because (IN-PL P (LINE A B)). Of course, this "proof" isn't too helpful, and we will need to use "brute force" to find the one missing step (S-16). In this case, the analogy process did not help solve the problem, but it did do something much better! In looking at the fine structure of the processing downstairs, we discover that the LINE procedure asserted (EQUAL P (LINE A B)). Therefore, the LINE procedure upstairs should make the appropriate IN-PL assertion in a

forward chaining deduction. Furthermore, the description of the "bug" in the lifted proof tells us the rest of what we require for patching and justifying (proving correct) solid geometry's new LINE procedure.

We now know imperatively what IN-PL means applied to planes and lines. If no natural analogy is available, the analogy process blithly uses an unnatural one.

Major Ideas

The three problems and solutions illustrate the major new ideas which have resulted from the research to date. These ideas are:

1. The use of semantic templates to initialize and extend the analogy map. To be truly useful, type information must be derived from the problem domain description. This represents a necessary correction to the technique developed by Kling.

2. The notion of an evolving analogy map. Winston introduced the idea of a "context" for learning. Our notion is an extension and refinement of his technique.

3. Our use of representation theorems, which are required for proofs of program correctness and for driving proofs through representations (not discussed), to assign an interpretation to extended predicate forms.

4. The notion that analogy is used to lift problem-solving procedures, not problem solutions. At best, the inverse (i.e., "lifted") problem solution gives us a "feel" for what the solution to the original problem should look like. Indeed, some feel that the role of analogy is only to obtain the *form* of the solution. Our view is more ambitious. Indeed, analogy does have this role, but, as we have seen, there is no reason to settle for frosting when we can have the cake as well.

5. The use of "bugs" in proofs to correct procedures being learned. The use of bug types and descriptions to patch programs is, of course, due to Sussman.

6. When we started this research, we assumed that at times we would be forced to do arbitrarily hard theorem proving (in a "dumb" way). It turned out that only one-step proofs were ever required. This is very fortunate, since it implies that all of the above applies if one uses "common sense" logic (which must include the intuitionistic portion of mathematical logic) instead of predicate calculus. Were we required to use, say, resolution theorem proving, this would not be so.

Role of Theorem Proving

Rarely does one want a theorem prover. One might therefore ask exactly how much of the theory of analogy presented above is predicated on the particular experts being theorem provers. In an important sense, this question is meaningless!

The expert problem solvers being discussed do not have proofs as their primary output.

Rather, functions have values as output, and predicates return truth values. The experts are programs, and find answers as efficiently as possible (without resorting to analytic techniques). Where, then, do the proofs we have been quoting come from?

Like all good programs, we have proven the correctness of the various routines in our expert problem solver. We have also developed a set of techniques whereby from these correctness proofs and the solution process we can generate a proof that the solution process did indeed produce the correct answer. It is this proof that the analogy process needs.

What this means is that if an expert problem solver can be proven correct (for some theory of the domain), then we can go from this proof that the program is correct to proofs that particular answers to particular questions are also correct. Thus any expert problem solver can be made into a theorem prover.

Finally, it should be noted that these ideas are independent of the formalism used to write the expert problem solver, and probably independent of the choice of the domains of plane and solid geometry.