Finding and Using Analogies to Guide Mathematical Proof

Stephen G. Owen

PhD University of Edinburgh 1987

I DECLARE THAT THIS THESIS HAS BEEN COMPOSED BY MYSELF AND THAT THE WORK DESCRIBED IN IT IS MY OWN:

(Stephen Owen)

Acknowledgements

I thank my supervisor, Alan Bundy, for encouraging me to pursue analogical reasoning as my thesis topic; his many comments and suggestions have been crucial in the direction and focusing of my research.

My thanks also go to Chris Miller, my second supervisor, for his help with the project. His thesis on analogy was the starting point of my interest in the subject.

In addition to those mentioned above, various people read, and made useful comments on, the draft of this thesis — Steve Todd and John Moroz from HP, Paul Brna and Pete Madden from Edinburgh.

As well as reading the draft, Paul Brna has been most helpful to me in bridging the gap between Bristol and Edinburgh, and with his command of Latex.

I am also most grateful to HP Labs Bristol, for their understanding and generosity in allowing me time and resources to complete the thesis.

Lastly, I thank my wife, Lucy, for her patience and understanding while I have been writing up.

While doing the research for this thesis, I was supported by an SERC Studentship.

Abstract

This thesis is concerned with reasoning by analogy within the context of automated problem solving. In particular, we consider the provision of an analogical reasoning component to a resolution theorem proving system. The framework for reasoning by analogy which we use (called Basic APS) contains three major components — the finding of analogies (analogy matching), the construction of analogical plans, and the application of the plans to guide the search of a theorem prover. We first discuss the relationship of analogy to other machine learning techniques. We then develop programs for each of the component processes of Basic APS.

First we consider analogy matching. We reconstruct, analyse and crticise two previous analogy matchers. We introduce the notion of analogy heuristics in order to understand the matchers. We find that we can explain the shortcomings of the matchers in terms of analogy heuristics. We then develop a new analogy matching algorithm, based on flexible application of analogy heuristics, and demonstrate its superiority to the previous matchers.

We go on to consider analogical plan construction. We describe procedures for constructing a plan for the solution of a problem, given the solution of a different problem and an analogy match between the two problems. Again, we compare our procedures with corresponding ones from previous systems.

We then describe procedures for the execution of analogical plans. We demonstrate the procedures on a number of example analogies. The analogies involved are straightforward for a human, but the problems themselves involve huge search spaces, if tackled directly using resolution. By comparison with unguided search, we demonstrate the dramatic reduction in search entailed by the use of an analogical plan.

We then consider some directions for development of our analogy systems, which have not yet been implemented. Firstly, towards more flexible and powerful execution of analogical plans. Secondly, towards an analogy system which can improve its own ability to find and apply analogies over the course of experience.

Published Papers

A paper entitled "Heuristics for Analogy Matching" was presented at the European Conference in Artificial Intelligence, Brighton 1986, by the author of this thesis. This paper covers the work described in Chapter 3 of this thesis. A reprint of the paper may be found at the end of the thesis.

Table of Contents

1. Introduction **1**

1.1 Analogical problem solving 1

1.2 Some Examples 5

1.3 Related Work 9

1.3.1 Less relevant work 9

1.3.2 More relevant work 11

1.4 Scope and Structure of Thesis 16

1.4.1 Scope 16

1.4.2 Structure 20

1.5 Conclusion 20

2. Analogy in the Context of Human and Machine Learning **21**

2.1 Introduction 21

2.2 Philosophers'/Psychologists' Accounts 22

2.3 Hesse's Theory 23

2.4 Psychological Theories of Simple Analogies 25

2.5 More recent psychological accounts 27

2.6 Summary 28

2.7 Machine Learning Systems 29

2.7.1 LEX 30

2.7.2 LEX2 30

2.7.3 LP 31

2.7.4 LEX vs APS 31

2.7.5 LEX2 vs APS 33

2.7.6 APS vs LP 33

2.8 Summary 35

3. Analysis of Existing Analogy Matchers **37**

3.1 Introduction — Motivation for Reconstructions 37

3.2 Descriptions of algorithms and reconstructions 39

3.2.1 Munyer 39

3.2.2 Kling 51

3.3 Summary of reconstructions 66

3.3.1 Heuristics used and justifications 66

3.3.2 Conclusion 74

4. An Improved Analogy Matcher **76**

4.1 Introduction 76

4.2 Design of the matcher 78

4.2.1 Dual use of heuristics 78

4.2.2 Design criteria 80

4.2.3 The development of a partial match 84

4.2.4 Co-ordination of matching between different partial matches 100

4.3 Summary of new matcher 104

4.3.1 Additional features 106

4.4 Performance 108

4.5 Assessment of new matcher 114

4.6 Areas for improvement 117

4.7 Limitations of matching on given problem statements alone . . . 120

4.8 Summary 124

5. How to Map a Proof Using an Analogy **126**

5.1 Introduction 126

5.2 Negating and partitioning the mapping 129

5.3 Analogue Construction Rule 131

5.3.1 Comparison with Munyer's AIR 137

5.4 Analogous Operator Identification 139

5.4.1 Kling's EXTENDER 139

5.4.2 An improved AOI 143

5.5 Mapping Inference Positions 153

5.6 Summary and Conclusions 155

6. The application of analogical plans **157**

6.1 A resolution/paramodulation system 158

6.2 A framework for analogy application 160

6.3 Issues in plan construction and application 161

6.3.1 What information should the plan contain ? 161

6.3.2 Should the plan forming and validating procedures be separate or interleaved ? 163

6.3.3 Summary of possibilities 165

6.3.4 Special Unification 167

6.4 Performance 169

6.4.1 Examples 169

6.4.2 Results 171

6.4.3 Discussion 180

6.4.4 Comparison with unguided search and existing analogy systems 185

6.5 Conclusion 186

7. Further Work **188**

7.1 Introduction 188

7.2 The patching of analogical plans 189

7.2.1 Flawed analogical plans 190

7.2.2 Example flawed plans and patches 192

7.2.3 How to implement patching strategies 196

7.2.4 Coordination of patching strategies 204

7.2.5 Comparison with Kling, Munyer and Carbonnell 207

7.2.6 Summary 209

7.3 Improving Analogical Reasoning By Learning Global Analogies and Semantic Types 210

7.3.1 Semantic types and dualities 211

7.3.2 Justification for Use 216

7.3.3 Connection between types and dualities 217

7.3.4 Learning Dualities 220

7.3.5 Learning Types 233

7.3.6 The use of dualities 235

7.3.7 Conjectures and refutations — alternative uses of analogy 238

7.3.8 Summary 240

8. Conclusion **242**

8.1 Motivation 242

8.2 Achievements 243

8.3 Limitations 246

8.4 Summary 257

A. Comparison of the three matchers **263**

A.1 Introduction 263

A.2 Examples 265

A.3 Conclusion 280

B. Sample output of matcher and application system **282**

B.1 Matcher 282

B.2 Application system D 286

List of Figures

1–1 Basic APS 3

2–1 Analogy and Generalisation 30

3–1 Algebraic type structures 65

3–2 Heuristics used 67

6–1 Implemented systems 165

6–2 Key to following figures 176

6–3 PROBLEM 1 177

6–4 PROBLEM 2 178

6–5 PROBLEM 3 179

6–6 PROBLEM 5 181

6–7 PROBLEM 6 182

6–8 PROBLEM 7 183

7–1 Extra step inserted 196

7–2 Conjecture made 197

7–3 Plan step ignored 198

7–4 Schematic Patches 199

7–5 An example hierarchy 212

7–6 An alternative hierarchy 213

7–7 Algebraic hierarchies 218

7–8 Extended algebraic hierarchies 218

7–9 Arithmetic hierarchy 219

7–10 Analogous proofs 228

7–11 Analogous proofs with lemma 231

7–12 Proof of lemma 232

Chapter 1

Introduction

This chapter has three main purposes:

- *To introduce the problem addressed in this thesis, namely that of problem solving by analogy, including the terminology associated with it, and to give some examples of the kind of analogies with which the thesis is concerned.*
- *To explain the scope and structure of the thesis.*
- *To review previous work in the area, and explain how it relates to the work described in the thesis.*

1.1 Analogical problem solving

The subject of this thesis is problem solving by analogy, in particular the provision of an analogical reasoning component to automated problem solving systems. The goal is primarily a technical one, if only vaguely specified: to enable an automated problem solving system to exploit its knowledge of the solution to one problem when faced with a similar problem to solve. However, much of the inspiration for the endeavour comes from the well-accepted human ability to reason by analogy — it is this apparently natural ability which, when set alongside the inability of even a powerful theorem prover to make use of its experience, motivates the study of analogical reasoning in Artificial Intelligence (AI).

The previous paragraph gives an informal definition of analogical problem solving: the solution of a problem using knowledge of the solution to a similar problem as a guide. As we shall see, we will require to refine this definition later, for example to exclude reasoning by inductive generalisations, which certainly fits the definition given. However, the definition given forms the focus of this thesis.

The provision of such analogical ability to an automated problem solver would be a major step towards an ambitious goal: to develop an **extensible reasoning system** — one which would improve its own problem solving ability over the course of its experience. Previous solutions would be retained for use as analogues in the solution of new problems; as experience developed, the chances of finding a good analogy would increase, and problem solving ability would be enhanced. Such a system remains a long term aim. Many hard problems will have to be overcome before one is built; some of these problems are addressed in this thesis, some are not. Section 1.4 discusses the scope and limitations of the thesis.

Figure 1–1 illustrates schematically the operation of an analogical reasoning component. The problem solver is set problem P_1 to solve (the **target problem**). It finds, or is directed towards, another problem P_2 (the **base problem**) whose solution S_2 it knows. A **match** is found between the two problem statements; the match represents the similarities, and perhaps differences, between the problems. Given such an analogy match, the system constructs out of S_2 a **plan**, X, for the solution of the target problem; as noted above, an analogy is used to *guide* the problem solver; the guidance is encapsulated in the analogical plan. The system then attempts to **apply** the plan, which means following the guidance encoded in the plan, as far as possible, in the hope of constructing a solution to the target problem. In the language of search, the problem solver uses the analogical plan to choose which steps to take in attempting to solve P_1. If the analogy is a good one, the steps suggested will, on the whole, be the right ones, and the search required to solve P_1 will have been greatly reduced. We will call this approach to analogical problem solving (APS) **Basic APS**.

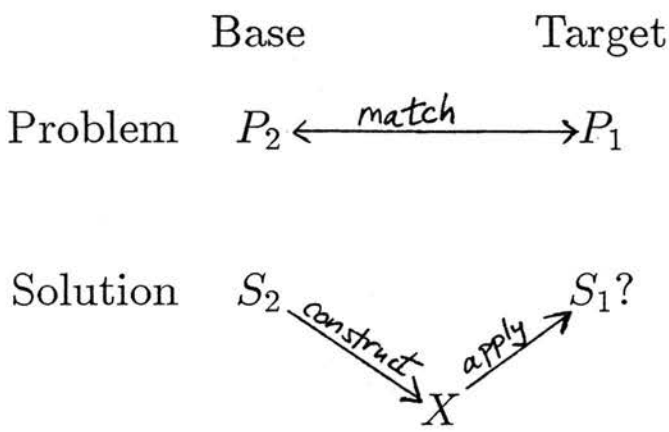


Figure 1–1: Basic APS

Diversity of analogy

It is well accepted that people are able to use analogies in diverse ways, not all fitting into Basic APS. We briefly consider some alternatives; each possibility has its counterpart among potential applications of analogy within AI.

Conjectures are often made by analogy: given an analogy between two domains and a known property of one of them, we might suspect there to be an analogous property in the other; we could conjecture an analogous property to hold in the latter domain, and then attempt to verify the conjecture by analogy to the demonstration of the known property. This can give a useful focus to the exploration of an unfamiliar domain. A discovery and reasoning system such as that proposed in [Bundy 85] might well be able to make use of such a conjecture making and verifying ability.

Similarly, definitions are frequently made by analogy: given an analogy between an unfamiliar domain which is being explored and a better understood domain, we might make a definition of a new concept in the former domain by analogy to one which has proved useful in the latter. We hope that the analogy remains fruitful, and the new concept leads to interesting results. Again, such an ability might prove useful in a discovery and reasoning system.

Another major area where analogies are frequently used by humans is in

teaching and explanation: we often explain an unfamiliar situation by analogy to another situation more familiar to the recipient of the explanation. For example, we might explain electricity using the flow of fluids as an analogy. Explanation by analogy has been proposed as a potentially useful part of the explanation facilities of expert systems and tutoring systems.

An ability to perform Basic APS can also be embedded within larger scale reasoning activity. For example, when attempting to solve a problem, a subproblem may be produced which we can solve by analogy, while using different techniques on the rest of the problem; we may use different analogies on different subproblems. We may also solve a subproblem by analogy to another subproblem of the same overall problem.

Emphasis on Basic APS

There are many uses of analogy made by humans, and many corresponding potential uses within AI. This thesis focusses on the Basic APS model described above; in addition, in Chapter 7 we do consider conjecture by analogy in some detail, but no such facility has been implemented yet. We concentrate on Basic APS for two main reasons: firstly, it is probably the simplest, in structure, of the uses of analogy; thus it is natural to consider it first. Secondly, although Basic APS is only one among many possible activities with analogy, it seems to have a central position among them. The acid test of an analogy comes down to whether problems on one side can be solved in similar ways to those on the other. This is just what is directly addressed in Basic APS. For example, the usefulness of an explanation made by analogy to the person receiving it depends on whether, in future, the analogy will enable him to tackle new problems better than he could have done without it; that is, whether new problems can be solved in similar ways to corresponding ones in the familiar domain. Similarly, the usefulness of a definition made by analogy depends on whether it leads to interesting theorems on one side whose proofs are suggested by those on the other side. Thus, a system which could exploit definitions (and likewise conjectures) made by analogy would need to be competent at Basic APS. It is more obvious that systems which embed

Basic APS within a richer framework rely on the power of Basic APS for their own power.

1.2 Some Examples

We give here some examples of the kind of analogies which we would wish an analogy system to handle. There are several purposes in giving examples at this point. First, to give a feel for the task that we face in building an analogy system; in particular to realise the kind of variation between problems which even a fairly modest analogy system will have to cope with. Second, so that we can define and illustrate the essential notion of an **analogy match**; all AI accounts of analogy, including this one, have the same basic idea about what an analogy match is, although there are important differences in how the idea is developed. Third, so that we can understand the limitations of some of the work on analogy reviewed in section 1.3, when applied to our domain of mathematics.

Consider first the following two theorems of Boolean algebra[1]:

$$x \cup 0 = x \qquad \text{and} \qquad x \cap 1 = x$$

These are quite separate problems, involving different functions and constants. However, they have the same structural form. The structural correspondence extends from the theorems to their proofs (which we do not show here); while different sequences of axioms are applied, the proofs have the same structural similarity that the theorems do — that is, the proofs are analogous. A person learning about Boolean algebra soon finds that proofs of theorems such as the second of those above can be directly modelled on known proofs of corresponding theorems such as the first. This is an example of a very close analogy — the

[1]Free variables are implicitly universally quantified in the examples throughout the thesis.

proofs proceed in *precisely* corresponding ways — and we would certainly hope that an analogy system would be able to find and exploit it.

We can express the structural similarity that exists between the above theorems by defining a correspondence between symbols occurring in the formulae, which preserves the syntactic structure of the formulae. Such a correspondence is graphically illustrated as follows:

Example 1

$$\begin{array}{c} x \cup 0 = x \\ \| \; \| \; \| \; \| \; \| \\ x \cap 1 = x \end{array}$$

The notion of a symbolic correspondence intuitively expresses well the similarities (and differences) between the two formulae; we call symbolic correspondences between formulae **analogy matches**. The basic notion of analogy matches as symbolic correspondences between formulae is common to all of the AI accounts of analogy; the nature of the correspondences differs between different accounts, though, as we will see in section 1.3.

We can also express the match between the formulae above as a symbolic correspondence without reference to the formulae:

$$\begin{array}{c} x \longleftrightarrow x \\ = \longleftrightarrow = \\ \cup \longleftrightarrow \cap \\ 0 \longleftrightarrow 1 \end{array}$$

It is often easier to think of analogies in this way; we can clearly compute the latter presentation from the former, but not vice-versa, in general. The structural correspondence discarded from the second presentation is an important factor in the analogy.

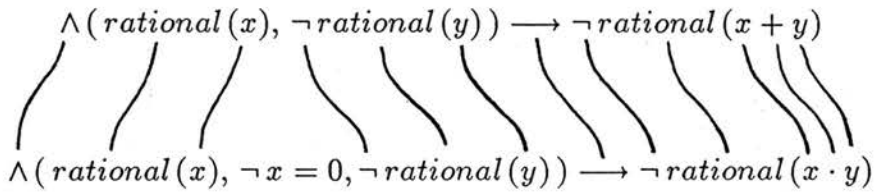
As a second example, consider the following two propositions of elementary real number theory:

1. The sum of a rational number and an irrational number is irrational.

2. The product of a non-zero rational number and an irrational number is irrational.

A potential analogy is immediately suggested by reading these two propositions together. The following match between formal representations of the propositions makes the correspondence explicit:

Example 2



The first proposition can be thought of as one about addition, and the second as a corresponding one about multiplication. It turns out that these propositions can in fact be proved in corresponding ways (again, we do not give the proofs here, but leave them until Chapter 6). However the correspondence is not so precise as it was in the first example — some of the symbols of one of the propositions are left unmatched, those that make up the 'non-zero' condition. This is because there is nothing in the other proposition which corresponds to the non-zero condition — it is part of the difference between the propositions. There is a corresponding mismatch between the proofs: at least one extra step needs to be put into the proof of the second proposition to handle the non-zero condition (see Chapter 6). However, the mismatches between the propositions and between the proofs do not seriously disrupt our intuitions that there is an analogy here (apart from the extra step, the proofs correspond closely). Many intuitively close analogies turn out to involve unmatched symbols, sometimes, as here, unmatched atoms, sometimes unmatched arguments to predicates or functions. Thus we take the view that an analogy system will have to allow matches of this form if it is to be useful.

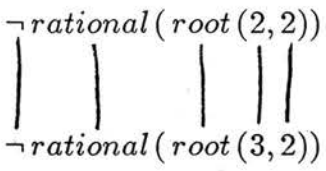
To illustrate another property of analogy matches, we consider an example involving one of the most famous theorems in mathematics:

1. The square root of 2 is irrational.

2. The cube root of 2 is irrational.

Again, we can immediately recognize a potential analogy, which, given formal representations, we can make explicit as follows:

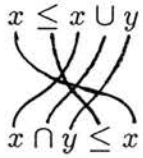
Example 3



Our suspicion is again borne out by the fact that the two propositions can be proved in very similar ways. However, notice that the match between the propositions is, in a sense, inconsistent; that is, a symbol on one side, 2, is associated with two *different* symbols on the other side, 2 and 3. Just as with unmatched symbols, there is nothing about this situation which prevents a person from recognising the analogy. Inconsistencies frequently occur in analogies. We therefore take the view that we cannot rule out inconsistent matches, such as this one, from our analogy system without seriously restricting its generality.

Another frequent feature of analogy matches, illustrated by the following example, is that predicates and functions which are associated can have their arguments permuted by the match:

Example 4



Here the arguments to $\leq$ are swapped in the match. The analogy is closely related to the first example given above, and is just as close. In general, the ordering of arguments is the result of an arbitrary choice, and there is no reason why analogy matches should preserve the order.

1.3 Related Work

Now that we have introduced the problem to be addressed in this thesis, we look at previous work which is relevant to the problem. In the previous section, we looked at examples which indicated the variety of matches which an analogy system will need to be able to find. When looking at other work on analogy, we will be particularly concerned with the variety of matches permissible by the systems.

There is much work on analogy in the literature; some in AI, some on the borders between AI and psychology, some in philosophy, particularly the philosophy of science. We will not attempt a comprehensive review of all work that has any connection with analogy. We will first mention some of the AI accounts which are less closely related to the concerns of this thesis; we chose these because of their influence on the field. We then mention in more detail the AI accounts which are more relevant to the thesis. Those accounts which are particularly relevant are discussed in much more detail in subsequent chapters. We leave the work in psychology and philosophy to be mentioned in later chapters where it is relevant.

[Miller 82] gives a more detailed account of some of the accounts, and also includes some we do not mention.

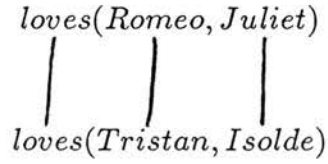
1.3.1 Less relevant work

Many of the attempts to model analogy in AI have not been concerned with *reasoning* by analogy (i.e. using analogies to solve problems in a domain); they have mainly restricted themselves to the initial matching stage of the analogy process. This does not in itself make them irrelevant to this thesis, as we will be much concerned with analogical matching in Chapters 3 and 4. However the notions of analogy matches and algorithms for constructing them have tended to be over-simplified in these accounts. This seems to be because they operate in

domains which have no accepted conventions for formal representation; thus the authors have defined their own somewhat idealised representations within which the analogies are simple. Within problem solving domains, such as mathematics, the problem representations are largely dictated by convention, and thus cannot be devised specially by the designer of an analogy system.

Winston

Winston [Winston 80] has considered analogies between simple representations of the plots of stories. He restricts his analogies to be 1-1 correspondences between objects from the plots, which are considered analogous by virtue of bearing some of the same relations to each other in one story as in the other. For example, the following match between a pair of propositions would contribute to one of Winston's analogies:



Both of these requirements on analogies are too restrictive for the types of analogy with which we are concerned in this thesis: examples 2 and 3 indicate that analogical correspondences need not be bijective; example 1 shows that the relations and functions which give an analogy strength need not be the same on one side as the other.

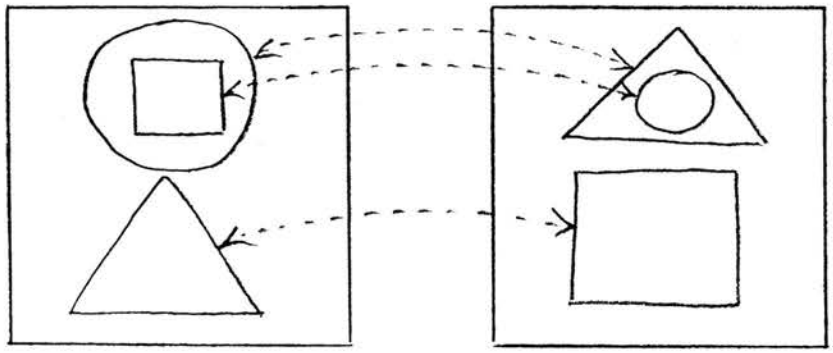
Evans

[Evans 68]

Evans wrote a program which attempted to solve geometric analogy problems of the kind which occur in intelligence tests. These problems test the ability to perceive relations and analogies, or rather the ability to perceive the same analogy as the person who set the problem. The analogies are not used for any external purpose. Evans' concept of an analogy match is similar to Winston's: a bijective relation between objects such that identical relations hold between corresponding objects. An example of such a match is the following

$$\begin{array}{cc} on\_top(circle_1, triangle_1), & inside(square_1, circle_1) \\ | \quad | \quad | & | \quad | \quad | \\ on\_top(triangle_2, square_2), & inside(circle_2, triangle_2) \end{array}$$

which represents the following correspondence between geometrical figures:



Thus Evans' notion of analogy match is again too restrictive for the purposes of this thesis. However, Evans' early work on analogy nevertheless is an important part of the development of analogy within AI – as far as the present author is aware, Evans was the first person to articulate analogies as symbolic correspondences between formal representations. This notion of what an analogy is exists implicitly in the work of both Hesse [Hesse 63] and Polya [Polya 54], but is not developed to the stage of being made explicit.

1.3.2 More relevant work

Inderkhya

Inderkhya [Indurkhya 85] describes a knowledge representation scheme, very similar to first-order logic, and defines a notion of analogy within it. While Inderkhya does consider the problem of reasoning with an analogy, both his problems and his concept of analogy are idealised: he requires an analogy to be an isomorphism between formulae; i.e. predicates and objects may be associated with different predicates and objects (unlike in the accounts mentioned above), but only in a strictly consistent manner. There can be no permutation of arguments between

associated terms and formulae, and there can be no unmatched arguments. Thus most of the examples given above cannot be captured in Inderkhya's system.

Carbonnell

Carbonnell has discussed how a problem solving system can be augmented with an analogy component. In [Carbonnell 83b], he proposes a model essentially the same as the Basic APS model of Figure 1–1 except that the plan construction phase is omitted – the base solution is taken as the plan for the target. He gives no definition of, nor algorithm for finding, analogies. His main concern seems to be with the plan application stage. He proposes various operators which may be applied to a faulty plan in the hope of transforming it into a valid target solution. Examples of these are 'insert an operator application at a particular point in the plan' and 'swap a consecutive pair of operator applications in the plan'. However, he gives no details as to how these solution transformation operators are to be controlled; this must be a serious problem, since the branching rate at the solution transformation level is very much greater than that at the original state space level. He does not even give any concrete examples of the proposed behaviour of his system. He seems to refer in places to an implemented system; however he gives no technical details of how the system works.

In [Carbonnell 83a], Carbonnell proposes a new model for reasoning by analogy within an automated problem solver, for which he coins the phrase **derivational analogy**. This is different from the Basic APS model and his previous model in the following respect: instead of looking for analogies between the problem representations alone, analogies are sought between initial segments of problem-solving activity between the target and the base; that is, the target problem is attacked by the problem solver (presumably with some general-purpose search technique); the entire trace of the problem solver's search at any stage is retained, including all failed branches and intermediate states; matching takes place between this structure and corresponding initial traces for candidate base problems; once an adequate match is found, the later stages of the process proceed in a similar way to Carbonnell's earlier model.

The motivation behind this model seems to be that an initial segment of problem-solving activity contains more information about the form of a potential solution than the problem does alone. This is certainly an attractive idea. But, once again, Carbonnell gives no examples or technical details with which his model can be tested. In particular he does not address the following issues which would arise in an implementation of his model:

- How can the enormous quantity of information needed for matching be stored and accessed efficiently? This applies both to the trace of the attempt at a target solution and particularly to the corresponding traces for all possible candidate base solutions which must be retained.

- How can such large structures, once retrieved, be matched in reasonable time? Note that there are bound to be many irrelevant differences between the target and base traces, arising, for example, from extra operators on one side which apply during the search but lead to dead ends.

- Why should an initial trace of a search be a good indication of the form of a successful solution? We only need to resort to analogy because we have a search problem; this means, by definition, that we cannot be confident that a particular partially developed search path will ever lead to a solution. Thus using such a search path to chose candidate base solutions seems problematic.

Carbonnell criticises analogy systems which base their decision of which base problem to pursue entirely on the comparison between problem representations. His criticism certainly carries some weight. However, Basic APS systems do not necessarily have this property, despite the initial appearance that they do: the development of an analogical plan can be frequently reassessed during plan application; if the plan is doing well it can be developed further; if not, or if there is a higher-rated alternative plan, the current plan can be suspended in favour of the other. Thus the process of choosing the right base problem can extend into the application phase, and thus be sensitive to more than just the original

problems. In Chapter 7, we consider the design of such systems in more detail.

We now turn to the attempts at analogy between mathematics problems.

Brown

Brown [Brown 77] considers how to exploit analogies in a slightly different way from the Basic APS model. He considers analogies between theories or domains rather than isolated pairs of problems. Given an analogy between theories and a target problem in one of the theories, a corresponding base problem in the other theory is *constructed* using the analogy. This process replaces the matching stage of the Basic APS model. Otherwise, Brown's model is the same as Basic APS.

Despite the differences, the issues that arise in Brown's model are the same as those in Basic APS, with which we will be concerned for most of this thesis – matching, plan construction and validation. [Brown 77] contains some interesting ideas, which we will refer to later in the thesis. He is certainly aware that analogy matching involves permuting and omitting arguments where necessary. However, it is clear that he is not describing an implemented system – his account has many gaps and assumptions in it. Thus it has not been as valuable to the project described in this thesis as has the work of Munyer and Kling described below.

Kling

Kling [Kling 71] reports an analogical reasoning system for use with an existing resolution theorem proving system. It is, as far as the present author is aware, the earliest attempt to automate the *use* of analogies to solve problems. Kling was the first to introduce the paradigm for reasoning by analogy which we have called Basic APS into AI. Kling was mainly concerned with the analogies that exist between different branches of abstract algebra, particularly group theory

and ring theory. The fact that these analogies are not perfect isomorphisms has forced Kling to take a more flexible notion of analogy match than many subsequent authors have done. We defer discussion of this notion, and Kling's matching algorithm, until Chapter 3.

It is perhaps misleading to describe the later stages of Kling's system as the construction and application of analogical *plans*: Kling uses an analogy to find analogues for the axioms which were used in the base proof; the axiom base of the theorem prover is then restricted to these analogues, and the theorem prover attempts to prove the target by its normal uniform search procedure. Thus the order of application of the axioms and any intermediate steps generated in the base proof are discarded. In AI, the term plan is usually taken to mean a structured entity which represents an intention to proceed in a particular way, including some notion of order. We need to generalise this definition to include Kling's candidate analogue axiom sets as plans: they represent an intention to proceed in a particular way, namely to attempt a proof of the target using only the analogous axioms, but without any specification of order. We will consider Kling's application routines in greater detail in Chapter 5.

Munyer

Munyer [Munyer 81] describes a system for reasoning by analogy. Munyer's system also fits into the Basic APS structure. Munyer, like Kling, is concerned with analogies between problems in mathematics. This has led him also to adopt a flexible view of an analogy match, and to face a correspondingly difficult analogy matching problem. Again, we postpone more detailed discussion of this until the next chapter.

Munyer describes two analogy application systems, the **implicit planning method** and the **explicit planning method** (the latter was not implemented). In both of these, the base solution is used to construct a plan for the solution of the target. These plans are constructed out of the intermediate steps which were generated in the base solution; the order in which they occur in the base

solution is also used in the plans; the axioms (or operators, in Munyer's terminology) which were applied in the base solution are ignored. It is interesting that Munyer's and Kling's analogical plans make use of disjoint sets of information from the base solutions. We will discuss Munyer's application procedures in more detail in later chapters.

The work of Kling and Munyer (and to a lesser extent Brown) has been of great assistance to the project described in this thesis; much of what follows is an analysis of their programs, and many of the ideas developed in later chapters have been suggested by the shortcomings which emerge. That their work has been so relevant is surely related to the fact that they consider analogies between problems in mathematics, and how to exploit them effectively. This has lead them to approach the difficulties inherent in these "naturally occurring" analogies.

1.4 Scope and Structure of Thesis

1.4.1 Scope

It has already been explained that this thesis concentrates on Basic APS. We note here some other aspects and restrictions of the project described in the thesis.

Mathematics as a source of analogies

We will be almost entirely concerned with analogies between problems in mathematics. Mathematics is a good domain for studying analogical problem solving for a number of reasons. Firstly, mathematics is accepted as providing a rich source of analogies, ranging from close and obvious to subtle and obscure. Therefore, we can build systems to deal initially with simple analogies, and then extend them to cope with more difficult examples. Secondly, there are well-developed formalisms and conventions for expressing mathematics problems formally, which

include precise definitions of what constitutes a solution to a problem. This means that we have a success criterion for the use of an analogy, namely whether or not it leads to a solution to the target problem. Since the formalisation of mathematics is accepted, we do not need to invent a formalism and representational conventions before we can work with analogies. This removes the need to tackle representation issues at the outset, and, perhaps more importantly, prevents us from representing our problems in an idealised and simplified way, in order to allow an over-simplified notion of analogy to be used.

Resolution in first-order logic

We use first-order logic in which to express theories, problems and solutions. The automated problem solving system which underlies the analogical reasoning system which we develop is a **resolution theorem prover** [Chang & Lee 73]. Both these choices were made on grounds of convenience, and the analogical techniques which we develop are not closely tied to either choice. That is, it would be straightforward to adapt the techniques for use in a different family of first-order theorem provers, such as natural deduction; it would also be quite straightforward to adapt the techniques for use in a different logical formalism, such as higher-order logic.

The notion of solution, or proof, which we use is that of the basic resolution proof (see Chapters 5 and 6). This comes fairly directly from the notion defined in logic textbooks — a sequence of formulae, each member of which is either an axiom or follows from formulae earlier in the list by logical inference. Thus we consider proofs to be 'flat' entities, not annotated or structured in any way. This is just the form in which proofs are produced by conventional theorem provers. Some automated reasoning systems, such as planning systems, may produce solutions which are structured according to the way in which they were constructed; for example, a solution may give information about the plans which were used to find it. Clearly, such annotated solutions can be flattened into the basic form. We discuss in Chapter 7 how such annotations may be used in the

application of analogies; however no annotations are used by the implemented systems described in this thesis.

Difficulty of analogies

Just as mathematics problems vary enormously in difficulty, there is a wide range of analogies from very simple to subtle and difficult. The approach taken in this thesis is that an automated analogy system should be able to handle easy analogies before it can handle difficult ones. A system which can make use of a seemingly difficult analogy, but is unable to handle much simpler ones is suspect. There appears to be no system reported in the literature which can handle even very close analogies of the type that arise between mathematics problems. This fact is not always evident from the claims made by researchers for their systems, but will become clear as we analyse the relevant systems in more detail.

The procedures implemented and described in this thesis were aimed first of all at close analogies, and then augmented so as to deal with successively more difficult cases. It is always important that the effective behaviour on the easier examples is maintained.

The extent of the development of the techniques has been limited within the project described in this thesis; the analogies which are exploited by the systems described here are all straightforward for a person. We consider in detail how our systems will need to be further improved to cope with more ambitious analogies in Chapter 7.

It is important to distinguish the difficulty of an analogy between a pair of problems and the difficulty of the problems themselves. The difficulty of an analogy refers to the process of finding and exploiting the analogy; the difficulty of the problems refers to solving them from scratch, without the use of an analogy. While the analogies considered in this thesis are straightforward, some of the problems are quite hard, both for people and for a resolution theorem proving system. In fact, some of the closest analogies we will look at, between problems in Boolean algebra, involve some of the hardest problems which we consider.

An analogy system can fail to make use of an analogy (recognised by its user) in one of two ways: firstly, it may fail to notice the analogy at all, a failure at the matching stage; secondly, it may notice and construct the analogy in the matching stage, but be unable to validate the analogical plan which results. The latter stages of the analogy process, the application of analogical plans, has been least developed in this project. Many analogies which are constructed by the matching stage do not lead to successful solutions. Thus the further work chapter (Chapter 7) concentrates on the further development of plan application.

Filtering base problems

An important issue which we do not consider in this thesis is the retrieval of candidate base problems from a potentially large base of stored problems and solutions. It is accepted by many authors that the analogy matching procedures which construct analogies are too expensive to be applied exhaustively throughout such a knowledge base. Some kind of filtering mechanism would be needed to cut down the possible base problems to a 'short-list' which could be passed to the full analogy matching procedures for further analysis. Unfortunately, there are no solutions to this problem in the literature, so it remains an area for future work.

Post-generalisation

Another potentially useful component of an analogical reasoning system would be an ability to **generalise** pairs of problems (and their solutions) which led to successful analogical solutions; i.e. to abstract out the similarities between the problems and solutions which had been exploited, while discarding the differences. This would amount to a post-processing step on analogical solutions. This is another area which is not addressed by this thesis. It is clearly necessary to have a competent Basic APS system before a generalisation component would be of use.

1.4.2 Structure

The thesis is mainly structured around the component processes of the Basic APS model. Successive chapters deal with analogy matching (Chapters 3 and 4), plan construction (Chapter 5) and plan validation (Chapters 6 and 7).

In Chapter 2, we discuss analogical problem solving as a form of machine learning, and how it relates to existing machine learning techniques. It is important that we consider other forms of machine learning for, while analogy works in a different way, the overall goals are often the same. Chapter 2 is also an appropriate place to discuss some of the work done on analogy in philosophy and psychology, and its relevance to the goals of this thesis.

1.5 Conclusion

We have introduced the problem of analogical reasoning within the context of automated problem solving, introduced the terminology used in this thesis, and given some examples to illustrate the nature of simple analogies between mathematics problems, which we would hope an analogy system could exploit. We have also discussed some limitations in the scope of the thesis, in particular some aspects of reasoning by analogy which we do not address.

We have also briefly reviewed some previous work related to the thesis, but have left the most interesting accounts to be analysed in more detail in later chapters.

The source code for all of the programs developed in this thesis is reproduced in [Owen 88].

Chapter 2

Analogy in the Context of Human and Machine Learning

The main purpose of this chapter is to put analogical reasoning in the context of alternative machine learning techniques, and to understand the trade-offs between them. To this end, we start by reviewing some work in the philosophy of science which considered the issue in the human case.

2.1 Introduction

Analogical reasoning, in the sense introduced in the previous chapter, is a form of machine learning — a successful analogical reasoning system will improve its performance over time by learning from examples which it has experienced. There are other forms of machine learning which have the same goal, but which attempt to achieve it in different ways. Broadly, these are generalisation techniques. Thus, just as we should take account of existing work in analogical reasoning, we should also take account of that in the other areas. In this chapter we compare analogical reasoning with other forms of machine learning.

In making this comparison, some work which has been done in philosophy and psychology on the relationship between argument from analogy and argument from generalisations will be useful. Therefore, we first discuss this work.

While discussing this work, we take the opportunity to give a brief review of psychological accounts of analogy. This review is not part of the main development of the current thesis, as it turns out that psychologists' theories do not provide much computational insight into the problems we are concerned with.

Then we will turn to machine problem solving, and will try to understand the relationship and trade-offs between analogy and other machine learning techniques, using the corresponding human situation as an analogy.

2.2 Philosophers'/Psychologists' Accounts

Analogy has been a recognised tool in human investigation for thousands of years. Aristotle proposed a theory of analogy. However, in this discussion, we will go back no further then Hesse's theory [Hesse 63], published in 1963. We truncate the investigation so recently for two reasons: firstly, the analysis of older theories of analogy, particularly medieval and ancient ones, while fascinating in itself, becomes embroiled in discussion of the semantics of the terms which were used; such discussions are best carried out by linguists. Secondly, Hesse herself has studied the older theories in some detail; for the purposes of this thesis, we may assume that Hesse's theory has been informed by any insights to be had from the older theories.

Hesse was interested in analogy as a tool in scientific investigation, a use for analogy closely related to our own. We next look briefly at two accounts proposed by psychologists, those of Rumelhart and Abrahamson [Rumelhart & Abrahamson 73] and Sternberg [Sternberg 77], which are concerned with analogy problems such as occur in IQ tests. We then consider two more recent psychological accounts, those of Gentner [Gentner 82] and Gick and Holyoak [Gick & Holyoak 80], which are concerned with the use of analogy to solve problems; both of these have some relevance to this thesis.

2.3 Hesse's Theory

Hesse's theory of analogy [Hesse 63] is concerned with the use of analogical models to make predictions in science. When a scientist is investigating some new and unfamiliar system (the target system), with only a few properties of the system known to him, he may use an analogy with a more familiar system (the base system) in order to predict new properties of the unfamiliar system; these predictions can then be tested by experiment.

Famous examples of scientific models, quoted by Hesse, are Newton's particle theory of light and Rutherford's solar system model of the atom.

Typically, the base domain is well enough understood that its basic mechanisms are known; that is, there is a successful theory which predicts the observed properties. The model is used to assert properties of the target because corresponding properties are known to be true of the base.

Hesse analyses the situation in the following way: the model is initially proposed on the basis of *recognisable similarities* between *observable properties* of the base and target systems; say the base has properties

$$\{B_1, \ldots, B_r\}$$

which are recognisably similar to known properties of the target

$$\{B'_1, \ldots, B'_r\}.$$

This part is known as the **positive analogy**. There are also properties of each domain known not to hold in the other; say

$$\{A_1, \ldots, A_n\}$$

in the base, and

$$\{C'_1, \ldots, C'_m\}$$

in the target. This part is known as the **negative analogy**. There are moreover properties of the base of which it is not known whether corresponding properties

hold in the target (remember that the target is sparsely understood). This part is known as the **neutral analogy**. The terms positive, negative and neutral analogy are due to Hesse.

Say that property R is in the neutral analogy. The use of the analogy is to assert that R' holds in the target. But there will be many properties which are presently in the neutral analogy but will be false in the target. How is it that the properties to assert and investigate are chosen from the neutral analogy? The fact that the base domain is well understood means that there will be **causal relations** known between the properties of the base. The properties from the neutral analogy which are chosen to form hypotheses for the target are those which are causally connected with properties from the positive analogy.

Perhaps a direct causal connection between the positive analogy and R is known. This would amount to the knowledge of (or belief in) a general law

$$B_1 \ \& \ \ldots \ \& \ B_r \longrightarrow R.$$

Such a law could (and should) be used to *infer* R' in the target by direct inference, without using the analogy at all. But (Hesse claims) such general laws are not often known; what is known is that

$$B_1 \ \& \ \ldots \ \& \ B_r \ \& \ A_1 \ \& \ \ldots \ \& \ A_n \longrightarrow R,$$

but precisely which factors are responsible for R in the base cannot be determined. For example, the motion of elastic balls may be taken as a model for the propagation of sound in air; the fact that balls rebound off hard surfaces may be correlated with (i.e. is causally connected to) their basic properties, using, say, Newtonian mechanics; but

> "... we cannot empirically separate the characteristics which are in common between the throwing of a ball and the uttering of a sound, and those which are different, in order to infer a general causal relation applicable also to sound: "throwing is correlated with rebound," in such a way that it is independent of the occurrence of the other characters of the ball." ([Hesse 63])

Thus the echoing of sound could be predicted (or explained) *by analogy* to the bouncing of balls, but not using a general theory, given the existing state of understanding.

So Hesse's argument is that while arguing by general law is preferable to arguing by analogy when it applies, the latter is applicable in many more situations than the former. According to Hesse, analogies tend to be "pre-theoretic", with respect to the target domain anyway; a successful analogy will lead to a deeper understanding of the target domain and hence, perhaps, to a causal theory of the target which will replace the analogy. Thus analogical models can be seen as complementary to general theories.

Hesse is rather vague about what constitutes the "recognisable similarity" on which her theory is based. Notice that her theory requires not only that we should be able to *recognise* the similarity between analogous properties A and A', but also that, given a property R in the neutral analogy, we can assert the analogous property R' in the target – i.e. we need to be able to *construct* the analogue of a property.

2.4 Psychological Theories of Simple Analogies

Simple analogies are problems of the form

$$A : B :: C : X$$

(which reads "A is to B as C is to X"), where A, B and C are given and X is to be determined, usually chosen from a small set of alternatives. This form of problem is common in intelligence tests. Psychologists became interested in providing models for the solution of such problems apparently because they believed this would help them understand the nature of intelligence. With simple analogies, the problem is recognising the analogy — we are not *using* the analogy to solve another problem.

Rumelhart and Abrahamson [Rumelhart & Abrahamson 73] give a model for the solution of simple analogies. In this model, the terms are represented as points in a multi-dimensional space, the dimensions being properties relevant to the terms in question, such as size, ferocity etc. for animals.

$$A : B :: C : X \quad \text{if} \quad \underline{B}\text{-}\underline{A}=\underline{X}\text{-}\underline{C}.$$

That is, the analogy relation holds if the differences in the pre-defined dimensions are the same on each side. Thus, the analogy is defined purely in terms of intrinsic properties of the entities in question rather than external relationships between them. So a simple analogy such as

$$DOG : CAT :: CAT : MOUSE$$

where the common relationship is a tendency for one term to chase the other, could not be represented adequately in this model. Similarly, scientific analogies, such as those studied by Hesse, typically involve systems of external relationships between entities. So the representational system chosen by Rumelhart and Abrahamson is inadequate for most interesting analogies. Perhaps the fact that the simple analogies are not used for anything, and thus that there is no objective criterion for their success, led Rumelhart and Abrahamson to propose such a restricted model. Also spatial models were fashionable in psychology at that time.

Sternberg [Sternberg 77] gives another model for the solution of simple analogies; in this model, terms are represented by some (unspecified) semantic memory, a semantic network for instance. An analogy is represented by the same relation occurring between different objects. This definition for analogy is too restrictive for the analogies which we are seeking in problem solving, in which relations on one side may match with different relations on the other. However, Sternberg's representation language can at least express a wider range of relationships between objects than Rumelhart and Abrahamson's. Again, not surprisingly, a model of simple analogies does not shed much light on the *use* of analogy in problem solving.

2.5 More recent psychological accounts

More recently, psychologists have moved on from simple analogies to consider the role of analogy in problem solving.

As mentioned above, Hesse's theory does not define the notion of similarity on which it is based. Gentner [Gentner 82] presents a theory of analogy, the **structure/mapping theory**, which attempts to define a notion of similarity (or analogy) between physical systems. This theory can therefore be seen as an extension of Hesse's. The notion of similarity which Gentner uses is similar to that used by Sternberg and mentioned above, except that it is more fully and clearly defined. Gentner uses a kind of semantic network to represent physical systems. An analogy between two such networks is defined to be a mapping between the objects of the networks so that the same relations hold between corresponding objects. This definition is too restrictive for the purposes of problem-solving by analogy, as pointed out above. Gentner also considers unary relations to be irrelevant to analogies, and so does not try to find analogues for them [1] ; she justifies this by saying that unary relations usually refer to superficial properties of the objects (such as the yellowness of the sun), which are therefore not part of the causal structure of the base domain, and so will never be helpful in making predictions in the target. In automatic problem solving, however, unary relations (represented by unary predicates) seem to be as useful in looking for analogies as any other kind. In the following example, the *only* relations present are unary[2]:

$$even(x) \wedge even(y) \longrightarrow even(x * y)$$

$$odd(x) \wedge odd(y) \longrightarrow odd(x * y)$$

---

[1] Brown [Brown 77] takes exactly the opposite view, namely that unary relations should form the cornerstones of analogical matches.

[2] We discuss the application of this analogy in Chapter 7.

These two factors (the insistence on identical non-unary relations, and the ignoring of unary relations) together make Gentner's definition of analogy too restrictive for the purposes of this thesis. Gentner's structure mappings are clearly very much dependent on the particular way in which the systems in question are represented; a complicated, real-world system such as the solar system could be represented in many different ways; Gentner has chosen the one that fits the best with her representation of the atom, i.e. she has, perhaps subconsciously, anticipated the intended analogy in forming her representations of the physical systems.

Furthermore, Gentner's theory is not computational, i.e. she does not suggest mechanisms for the construction of her structure mappings.

The work of Gick and Holyoak [Gick & Holyoak 80] addresses the issue of the level of representation of the base and target systems which Gentner avoided. They show how people represent the systems in such a way as to maximise the similarity between them (their notion of similarity, given representations, is very similar to Gentner's structure mapping). They liken this ability to change representation to a theory of text comprehension, in which a story can be described in many different ways, each being a summary of the story at some level of generality. While explaining the problem clearly, they do not suggest any computational mechanisms for either constructing the representations, or for matching them once constructed.

2.6 Summary

Hesse's theory of the use of analogy to make predictions in science is interesting and useful for the purposes of this thesis because it considers the applicability of prediction by analogy relative to argument from general laws or theories. This issue will be seen, later in this chapter, to arise again in the context of automated problem solving systems. Subsequent psychological theories have tended to ignore this issue, and to treat analogy as an isolated phenomenon.

Both Gentner and Gick & Holyoak raise interesting and relevant issues in analogy – of the notion of similarity, and the representation of the analogues – but neither account provides computational answers.

2.7 Machine Learning Systems

We now discuss analogy in the context of automatic problem solving – the subject of this thesis. As explained in the previous chapter, the goal of APS is to enable a problem solving system to improve its performance by exploiting its past experience in solving similar problems. Thus APS is a form of machine learning. There are other machine learning techniques which have the same overall goal, although their means of achieving it is different from APS. In assessing the power and applicability of APS, therefore, we should compare it with other machine learning systems. In this section, we compare APS to three existing machine learning systems, LEX [Mitchell 78], LEX2 [Utgoff 83] and LP [Silver 85]. The three systems are chosen as representative of different kinds of machine learning — LEX embodies an inductive form of learning, LEX2 uses an analytical, explanation-based technique, and LP is an example of learning by meta-level analysis.

All three are spontaneous generalisation systems; that is, given an example or examples of problem solving behaviour, they form a generalisation of both the problem and (aspects of) the solution. Faced with a new problem, they look for a generalisation which subsumes it (strict subsumption); when one is found, they attempt to instantiate the generalised solution to solve the given problem. In contrast, APS makes no spontaneous generalisation, but attempts to match old problems with new ones by *flexible* matching. These different approaches to exploiting past experience are illustrated schematically in Figure 2–1. In each case, there is a notion of the potential generality of the process – that is, across what sort of variation in problems can old experience be exploited. As well as the potential generality across problems, there is the issue of the potential variation

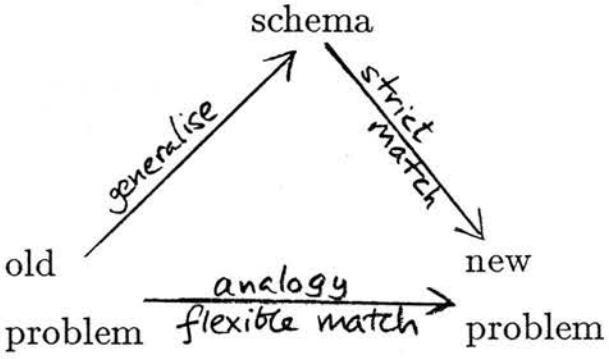


Figure 2–1: Analogy and Generalisation

in the form of the solutions. These two issues are crucial to the assessment of the generality (in terms of applicability) of the techniques. We compare the existing systems to the hypothetical Basic APS system with particular regard to these issues.

Firstly, we give a brief description of the existing systems.

2.7.1 LEX

LEX [Mitchell 78] used a technique called **version spaces** to learn heuristics for the application of particular rules in a symbolic integration system. The generalisation is based on the taking of least upper bounds in a fixed, user-supplied **type-hierarchy** for the domain. Positive instances for a particular operator are problems whose shortest solutions begin with an application of the operator. Given two positive instances, a generalisation is formed, which subsumes both, by replacing subterms at which they differ by the least upper bound of the differing function symbols in the hierarchy.

2.7.2 LEX2

LEX2 [Utgoff 83] works in the same domain as LEX, symbolic integration. It uses a technique called **goal regression** to find the most general form of a problem which can be solved by a particular sequence of operators. Its generalisations

stay within the **object language** — that is the language in which the examples and operators are expressed.

2.7.3 LP

LP [Silver 85] learns schemata for the solution of equations from single examples presented by the user. It first characterises the original equation, and the user's solution in terms of a sophisticated description language, quite separate from the object language — this language is termed the **meta-language**, since it describes object level terms. LP then uses a technique called **precondition analysis** to construct a schema by performing a simple kind of goal regression in the description language. LP's schemata are expressed in the meta-language.

2.7.4 LEX vs APS

A LEX generalisation has the same top-level structure as the problems which gave rise to it; it differs from the problems in that instances of particular functions have been replaced by typed function variables. The part of the original problems which has not been generalised (the greater part in most cases) must match the new problem exactly. More complicated types of matches, such as

$$\begin{array}{r} (x+y)-x=y \\ \backslash\backslash\backslash\ |\ |\ |\ / \\ x=0 \vee (x*y)/x=y, \end{array}$$

where there is an extra condition on one side, are not allowed. Matches in an APS system may also permute arguments and omit function symbols. Thus the range of problem generality in an APS system is potentially much greater than that in LEX.

With respect to the solution generality, LEX only makes generalisations relating to the application of *single* operators; thus it cannot capture variations in the solution which may correspond to variations in the problem. For example,

LEX could potentially handle the following problem variation (within Boolean algebra),

$$\begin{array}{c} x \wedge x = x \\ \| \quad \| \quad \| \quad \| \quad \| \\ x \vee x = x \end{array}$$

(assuming it knew the appropriate types), but could not represent the fact that *correspondingly different* operators need to be applied in the solution of the problems.

Thus, in both respects, APS offers greater generality than LEX, and would thus be applicable in many more situations. Referring to Hesse's theory described above, we can think of LEX as forming general laws inductively from instances which it has observed. That is, it abstracts out certain of the properties of the instances (in Hesse's terminology), and forms general laws (or rather beliefs) involving those that remain. However, its single method of generalising through a type hierarchy means that it can only abstract out certain sorts of property (the identity of certain functions). If a potential generalisation involves abstracting out other sorts of property, such as the order of arguments, LEX will not be able to capture it.

Furthermore, even within its range of variation, the reliability of LEX generalisations is questionable. As Hesse points out, arguing by general laws (or beliefs) involves claiming a causal link between the form of problem (expressed by the type hierarchy) and the form of solution (in this case, which operator was applied first). In the case of LEX, this link is not based on any theory of the problem solving process, as it is in, say, LP (see below), but is represented by a domain-level belief that the type hierarchy gives good dimensions for generalisation. While this may be true for some sets of problems, it is unlikely to be true throughout a non-trivial domain, and, in any case, strongly depends on the user supplying a *good* hierarchy. Thus, the causal links will not be strong enough to support reliable generalisations. The effect of this would be that LEX would make serious over-generalisations after extended experience.

In summary, APS offers greater scope for variation in both problem and solution, and thus will be more widely applicable than LEX's inductive generalisation, as predicted by Hesse's theory. Furthermore, it is questionable whether LEX's inductive technique is powerful enough to support valid generalisation.

2.7.5 LEX2 vs APS

LEX2 uses a form of **explanation-based generalisation**. That is, its generalisation from a solution to a problem is obtained by answering (at some level) the question "why did this solution work?" LEX2 is very cautious – its answer is simply that it worked because the sequence of operators applied was seen to solve the problem (an answer which would be considered bloody-minded of a human). Its generalisation technique (goal regression) looks for the most general problem which could have been solved by the same sequence of operators. Thus, its generalisations are very strong (theorems, in fact), but the range of problem and solution variation is limited: new problems must match perfectly with old ones, except for subterms which have been generalised to variables, and new solutions must be identical to old ones. Thus APS will be more widely applicable than LEX2. This is not to say that LEX2 is not a useful system; as Hesse points out, arguing by generalisation, where applicable, is preferable to arguing by analogy. Generally speaking, in a given non-trivial domain, there will be a trade-off between the strength and the generality of the generalisations: LEX2's are cautious – that is, not very general, but very strong.

2.7.6 APS vs LP

LP's generalisation technique, like LEX2's, is a form of explanation-based generalisation. The difference is that LP's answer to the question "why did this solution work?" is much less cautious and is based on sophisticated domain knowledge; it gives its answer, not in terms of the particular operators which were applied, but in terms of the effect which they had on the terms to which they were applied, expressed in a meta-level description language. Thus the gen-

eralisations of which LP is capable are much greater than those of LEX2; on the other hand, the generalisations are not as reliable, and schema application may fail.

The potential variation in problem obtainable by LP's meta-level technique goes beyond simple syntactic similarity; this is because LP's schemata are expressed in terms of the description language, and will be applied to new problems which satisfy a particular description. But, the descriptors in question, which may say, for example, that the equation must contain a pair of identical subterms, do not characterise the *syntactic* form of equations to which they apply. That is, we cannot write down the most general equation which satisfies a particular meta-level description. So the range of variation is much greater on the problems. Similarly, the solutions can vary considerably as well: the operators applied in the new situation do not have to be the same as the old ones, but can be any from a class of operators capable of producing similar meta-level effects. LP can also insert extra operators in the application of its schemas.

While the problem variation in APS is wider than that in LEX and LEX1, it is still syntactically based, and thus cannot express the leaps of generalisation available to LP. Similar remarks are true of the solution variation. Note that it is still possible, in theory, for APS to exploit variation which LP cannot: if the syntactic match between problems does not preserve the meta-level descriptors, LP would not regard the new problem as an instance of the schema derived from the old. Furthermore, an LP schema is only applicable within the domain (equation solving) in which it was formed; APS can potentially work across domains – indeed this is perhaps the most interesting application for analogy.

The success of LP's schemas is clearly dependent on the accuracy of the meta-language – that is, on the extent to which the (meta-level) form of a problem determines the (meta-level) form of the solution. In LP's domain of equation-solving, the meta-language seems extremely appropriate: it represents a theory of *how* to solve equations, which enables large generalisations to be made.

In the terms of Hesse's theory, there is a theory which underlies and explains the observed (meta-level) properties independently of the other characteristics

of the equations. This makes LP's generalisation, when it applies, preferable to APS. As Hesse shows, models or analogies are useful when general theories are not available. Thus, in domains which are not as well understood as equation-solving, APS will become more attractive since LP's techniques will not work.

2.8 Summary

The above analysis shows that APS has a role to play in an advanced learning and reasoning system. The only generalisation system we have discussed which can produce greater variation in both problem and solution, LP, relies on sophisticated domain knowledge. Where such knowledge is lacking, such as when a new domain is being explored, APS could provide a means of exploiting past experience. The following spectrum illustrates our conclusions:

analogy		generalisation
knowledge-free	———————————	knowledge-based
empirical		analytical

On the left hand side, in the absence of sufficient domain-specific knowledge, analogy will be the preferred technique for learning from past experience; on the right hand side, in the presence of sophisticated domain knowledge, generalisation techniques such as LP's precondition analysis will be preferred.

The question naturally arises of whether we could have a learning system in the middle of the spectrum; that is to say, one which combines a certain amount of spontaneous generalisation with a certain amount of flexible, analogical matching. If the meta-language for the domain was not fully developed, analogical techniques could supplement analytical techniques in the following ways:

- If the meta-language was not as general as it could be, i.e. there was more scope for generalisation in the domain than it knew, flexible matching could

be used between schemas and new problems; this would have the effect of "stretching" the schemas a bit further. The question would then arise of whether the results of the attempted applications could be used to increase the power of the meta-language so that it could express the greater generalisation. If this could be done, we would have a powerful system, capable of directing its generalisation language by experience; analogical reasoning would be a step towards reasoning by generalising, just as reasoning from models is a step towards the construction of theories in science. However, this seems to be an extremely difficult thing to do.

- If the meta-language was over-general, and was producing schemas which failed much of the time, analogical matching between the object-level problem statements could be used to select the most promising of a number of schemas which matched at the meta-level. The results of this could perhaps be used to refine the meta-language, although this again seems to be very difficult.

Chapter 3

Analysis of Existing Analogy Matchers

This chapter is about **analogy matching**, *i.e. finding analogies. We argue that analogy matching algorithms must include heuristic beliefs about promising matches, implicitly or explicitly. The matching algorithms of Munyer [Munyer 81] and Kling [Kling 71] are analysed; the analysis is done on the basis of Prolog reconstructions of the algorithms and their performance. We abstract the heuristics which are used in the algorithms and discuss their validity. We are able to explain the shortcomings of the algorithms, which are made clear by the reconstructions, in terms of the inflexible way in which they use their heuristics.*

3.1 Introduction — Motivation for Reconstructions

There seems to be no general, formal rule which tells us what constitutes a good analogy match. It is in the nature of analogy that it is partly empirical — we only know for sure whether a particular match is good or not after the application procedure has used it in trying to solve the target problem. Hesse's theory of

analogy, discussed in the last chapter, helps us to understand this: analogy (as opposed to generalisation) is useful when the causality in a domain is not well understood; that is, when we cannot say, in general terms, why the solution to the base worked; so, according to Hesse, we turn to analogy precisely when we are not *sure* whether we can apply our old experience to a new situation.

Therefore analogy matching algorithms use **heuristic criteria** to guide them in searching for good analogies. The heuristics that a particular matcher uses determine its idea (or rather its designer's) of what constitutes a promising analogy.

But the descriptions given in the literature of existing matchers do not make it at all clear what heuristics they are using — the authors generally do not say what heuristics their matchers are using, where they are using them, or why they are using them. There has been little discussion of the assumptions which matchers make. This makes it very hard to compare the existing matchers as to their performance — how much they achieve relative to the information they are given (see the discussion about semantic types below) — and hard also to extend and improve them.

The matchers of Kling and Munyer have been reconstructed (both in Prolog) in order to analyse their performance and underlying assumptions. The rationale behind such rational reconstructions of previous work is as follows: firstly, the discipline of having to transform the sometimes ambiguous descriptions of the algorithms given by their authors into computer programs forces one to examine the algorithms much more closely than one might do otherwise; secondly, having a working version of an algorithm makes it easy to test it on different examples from those around which it was designed, and thus to assess its generality.

The matchers of Kling and Munyer were chosen for reconstruction as they are the only complete algorithms reported in the literature which attempt to handle the sort of problem variation which occurs between analogous problems in mathematics. Such variations include **function** and **relation** mismatches, where, say, a relation on one side is matched with a different relation on the other; **inconsistent** matches, where a symbol on one side is matched with more

than one on the other; **unmatched subterms**, where a subterm of one of the problem representations is not matched; **structural mismatches**, where individual symbols, rather than complete subterms, are left unmatched.

Brown [Brown 77] makes some suggestions about how to construct analogies between **theories**, for example between two-dimensional and three-dimensional geometry; Brown's ideas do not apply directly to the construction of individual analogies between pairs of problems. Furthermore, his techniques are only vaguely specified, and do not really constitute a single algorithm.

3.2 Descriptions of algorithms and reconstructions

3.2.1 Munyer

The matching algorithm reported in [Munyer 81] takes two logical terms as input and outputs an **analogy match** between them. We have already introduced the notion of an analogy match (in Chapter 1); we now define it more precisely: an analogy match is a set of **positional associations** between symbols in the logical terms; ie consists of elements of the form

$$((symbol_1, position_1), (symbol_2, position_2))$$

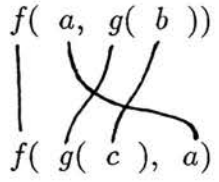
where $symbol_1$ has $position_1$ (the usual sequence of argument positions) in $term_1$ and likewise for $symbol_2$, $position_2$ and $term_2$. The two symbols in their respective positions are regarded as being *analogues* in the analogy represented by the match. For example, given the terms

$$f(a, g(b)) \quad and \quad f(g(c), a)\ ,$$

Munyer's algorithm would return the match

$$\{ ((\ f,\ [\ ]\ ),\ (\ f,\ [\ ]\ )),\ ((\ a,\ [1]\ ),\ (\ a,\ [2]\ )),$$
$$((\ g,\ [2]\ ),\ (\ g,\ [1]\ )),\ ((\ b,\ [2,1]\ ),\ (\ c,\ [1,1]\ ))\ \}$$

which can be graphically represented as:



Following Munyer, we have called the terms input to the system **logical terms**. They are well-formed in the sense of being representable as trees in the usual way (so that each symbol has a unique position in a term and heads a unique subterm) but are not necessarily first-order, since variables can have arguments (i.e. there can be variable functions and predicates). Munyer's matcher makes no distinction between predicates and functions. If logical *formulae* are to be matched, the logical connectives in them are treated as predicate/function symbols just like any normal predicates or functions in the formulae. So the matcher operates with an unrestricted and rather uniform syntax. The only information about the symbols that the matcher uses is whether or not functions or predicates are commutative.

The algorithm

Munyer's matcher works in three stages, constructing intermediate matches from the first two. They are: grounding, deleting and adding. The matching strategy may be briefly described as follows: in the first stage, an initial match of *a priori* plausible associations is constructed; in the second stage, this match is whittled down to a subset of associations which fit together well; in the third stage, holes are filled in the whittled down match.

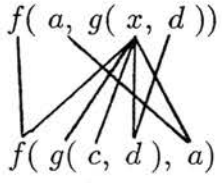
Grounding In this stage, an initial map is created in which

- Any pair of identical symbols, one from each term, are associated.
- Any variable in either term is associated with all symbols in the other term.

Deleting This stage takes as input the output of the previous stage, ie two terms and an initial map between them, and deletes some (or all or none) of the associations in the initial match. The choice of which maps to delete is made on the basis of how well each map **supports** the maps which 'directly dominate' it in the existing match. To explain what 'directly dominate' means, note that the associations in a match can be partially ordered in the following way: for symbols a and b in a term, define $a > b$ if b occurs in the (unique) subterm headed by a; for associations (a_1, a_2) and (b_1, b_2), (a_1, a_2) **dominates** (b_1, b_2) if $a_1 > b_1$ and $a_2 > b_2$; the relation of domination puts a partial order on the associations in a match. Now we can define the relation of direct domination as follows: (a_1, a_2) **directly dominates** (b_1, b_2) (in a match M) if (a_1, a_2) dominates (b_1, b_2) and there is no (c_1, c_2) in M, dominated by (a_1, a_2), which dominates (b_1, b_2).

The procedure is first to mark certain associations as being exempt from deletion, and then to delete those which have not been exempted and which share a node[1] with another association.

To be exempt from deletion, a match must give **support** to an association which directly dominates it. The notion of support is defined so that one association supports another if together they tend to preserve the structure of the terms being matched. For example, in the match



the (x, c) and (d, d) maps are directly dominated by the (g, g) map and both give it support. The (x, d) map, however, which is also directly dominated by (g, g), does not give it support.

---

[1]Munyer uses the term node to refer to an occurrence of a symbol at a position in a term.

Support is a relative concept; that is, map1 supports map2 if there is no map3 **competing with** map1 which supports map2 better. The definition of 'competes' is important : in the above example (x, c) competes with (x, d) at x, (x, d) competes with (d, d) at d, whereas (x, c) and (d, d) do not compete at all (all this with respect to the directly dominating map (g, g)).

The formal definition is that two maps compete on the left (w.r.t a map which directly dominates them both) if their left nodes occur in the same argument position of the left node of the dominating map. (Similarly for competition on the right).

A map supports a directly dominating map on the left if it has minimal **left penalty** among all the maps with which it competes on the left. Munyer defines the left penalty of a map (l_2, r_2) w.r.t a directly dominating map (l_1, r_1) as the sum

$$C.P. + P.P.$$

where $C.P.$, the containment penalty, is the number of unmapped symbols above l_2 and below l_1 in the tree structure of the left term, and $P.P.$, the permutation penalty, is 1 if (l_2, r_2) permutes arguments of (l_1, r_1) and neither l_1 or r_1 is commutative and 0 otherwise. If the minimum is not unique, all maps achieving it are deemed to give support.

In the above example, (x, c) and (x, d) compete on the left w.r.t. (g, g); both have $C.P. = 0$, (x, c) has $P.P. = 0$ and (x, d) has $P.P. = 1$, so (x, c) supports (g, g) on the left and (x, d) doesn't.

The notion of supporting on the right is defined analogously using the notion of competing on the right.

Finally, Munyer defines a map to be exempt from deletion if it gives another map *either* left support *or* right support.

After the marking stage has been completed, those maps which have not been marked and which share a node with another association are removed from the match. This removal stage (but not the marking stage) is iterated,

as an association which was removed may have been the only one which another association supported and so the latter should be removed also.

The motivation behind this complicated procedure seems to be that we should retain a set of associations which preserve the structure of the terms (this is behind the notion of direct dominance), and remove any alternative sets (this is behind the notion of competition).

Adding This stage takes as input the result of the deleting stage and adds an association between any pair of nodes in the two terms which are both unmapped in the existing match but whose parent nodes are mapped to each other.

Also, if the head symbols of the two terms are both unmapped an association between them is added.

This step is not iterated.

Heuristics

What are the analogy heuristics which are being used in this algorithm? That is to say, what does Munyer believe about analogies that led him to design the algorithm in the way which he did?

In the grounding stage, all possible identical associations are put in the initial match: thus Munyer must believe that identical associations make good analogues, and matches containing high proportions of identical associations make good analogies. Let us call this belief the **identical symbols heuristic.**

In the deleting stage, the initial match is whittled down to a subset of the original; this is done so that the associations in the subset tend to support each other well, in the sense of respecting the structure of the terms which are being matched. Thus, Munyer must, in addition, believe that matches which respect the structure of the terms make promising analogies. Let us call this belief the **partial homomorphism heuristic** (a homomorphism, in algebra, is a mapping which respects, in a formal sense, the structure of the entities which it relates).

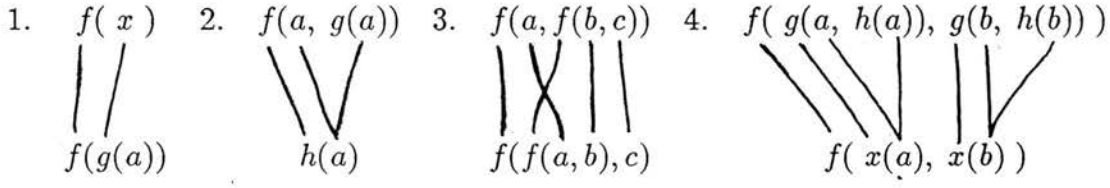
In the adding phase, associations are added to the match which fit in well, again in a structural sense, with those left after deletion. So this stage also makes use of the partial homomorphism heuristic.

Reconstruction

The reconstruction of Munyer's algorithm in Prolog proved to be fairly straightforward, although the details of the deleting stage are complicated and make the program larger than might be expected. The reconstruction has not been particularly concerned with efficiency, which could no doubt be improved, but the algorithm seems to be unavoidably expensive (we analyse this further below).

Performance and modifications

Among the examples of analogy matches which Munyer claims his matcher finds are the following four[2]:

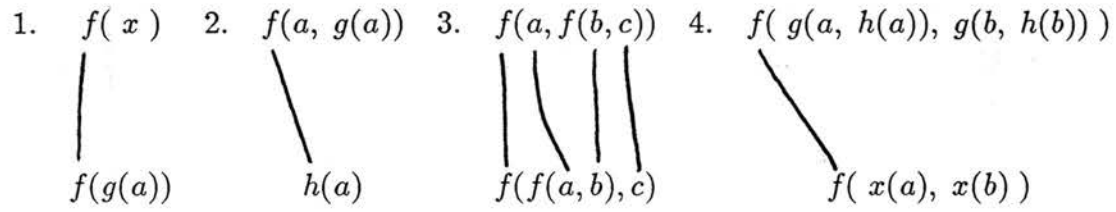


(Note the second order function variable, x, in the fourth example). Munyer does not go into why these matches are suitable analogies, but lists them as the types of matches which he thinks a matcher should be able to construct. The first example represents a first order unification : Munyer views analogy as "a cousin of unification" and thus the matcher should unify two terms when this is possible. The second example represents a second order mismatch, the similarity being that both terms are functions of a . The third example represents analogy modulo associativity, and the fourth a second order unification.

---

[2]These examples are chosen from [Munyer 81] in order to illustrate the problems with his original algorithm — they are not a random sample.

Initially, these examples were considered not so much for their intrinsic interest (they are not examples from real problems, and Munyer's explanations for them seem doubtful) as for whether the reconstruction of the algorithm constructed the intended matches. The matches produced by the reconstruction on the same examples are as follows:



(These matches were subsequently confirmed going through the algorithm by hand). These results are different from those that Munyer claims for his algorithm. We now analyse why the matcher does not get the intended answers, and propose modifications to it so that it does.

Number 3 — improvement A. Analysis of the third example suggested that it is wrong to count only **unmapped** intermediate nodes in computing the containment penalty — this is why the internal (f, f) map is deleted in favour of the (a, a) and (c, c) maps. If this restriction was dropped, (f, f) would have equal left penalty with (c, c) and equal right penalty with (a, a) and would thus be exempted.

This change was made to the program, and the new version was tested on Munyer's examples. The only change in result was in the third example above, for which the desired match was found. Since the performance wasn't worse on any of the examples, we will call this an improvement to the algorithm — improvement A.

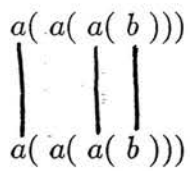
Numbers 1. and 4. — improvement B Note that the first example should produce a first-order unification. But Munyer gives a sketch of a proof that his algorithm is complete for first order unifiable terms i.e. that it finds the

most general unifier (m.g.u.) between such terms. This 'proof' must be wrong somewhere since the first example above had first order unifiable terms. The idea of the proof is that, given two unifiable terms, after the grounding stage some of the associations are 'correct', that is they are part of the valid unification, and some are not. The correct maps will have penalty 0. Munyer claims that any maximal incorrect map will have non-zero penalty and so will be deleted.

This claim is false both for the original algorithm and for the improvement A version. However, it turns out that the claim is made true if, in addition to improvement A, the condition for exemption from deletion, in the deleting stage of the algorithm, is strengthened in the following way: a map is exempt if it supports another map *both* on the left *and* on the right, as opposed to the previous condition which only required *either* one *or* the other.

This further change was made to the program, and the new version was tested on Munyer's examples. Again the only changes were improvements. The new version got the 'right' answers on the first and fourth of the examples given above. We therefore call the second change improvement B. Improvement B without improvement A reverted to the 'wrong' answer on the third example.

Number 2. — improvement C But, even with both improvements, the algorithm is not first order complete: incorrect associations are sure to be deleted, but correct ones are not sure to be retained. All four versions of the program (generated by including or omitting improvements A and B independently) produce the following match between the terms $a(a(a(b)))$ and $a(a(a(b)))$:



Note the missing $((a,[1]),(a,[1]))$ map. So the algorithm is not complete even for *identical* first order terms. There is an easy way to stop the correct maps being deleted: exempt any map from deletion which is maximal in the match w.r.t. the **partial dominance** relation. (An association (a_1, b_1) partially dominates

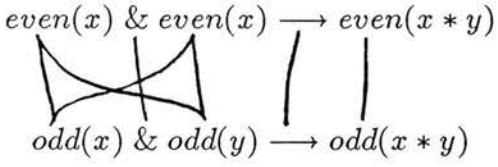
another (a_2, b_2) if *either* (a_1, b_1) dominates (a_2, b_2) *or* $a_1 = a_2$ and $b_1 > b_2$ *or* $b_1 = b_2$ and $a_1 > a_2$.)

It would too lenient to exempt all maximal maps under dominance, since this would exempt incorrect maps as well. This is why partial dominance is needed. With this change made to the program, it was tested again on Munyer's examples. It got the right answers on all of those listed above. The one remaining example from [Munyer 81] on which the new algorithm did not get the answer that Munyer claimed is the example with which Munyer illustrates his algorithm! We call this change improvement C.

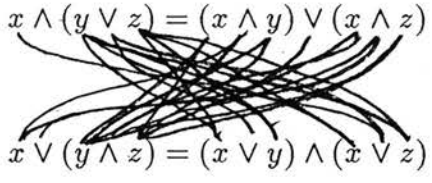
It may be that Munyer had in fact incorporated the three improvements into his algorithm, but forgot to update his written description of it. Otherwise, it seems odd that he should quote examples that his matcher could not carry out.

Assessment of the algorithm

The first thing which is notable about Munyer's algorithm, or at least the reconstruction of it, is it's enormous computational cost. For example, in constructing the match



the program took 40 seconds of c.p.u. time (interpreted Quintus Prolog on a Sun3 workstation); and on the match



(27 associations!)

took 185 seconds. While the reconstruction of the algorithm has not been particularly concerned with efficiency, the high cost seems to be an inherent feature of the structure of the algorithm, in particular the treatment of variables. In

the grounding stage, associations are placed between a variable and all symbols in the other term, on the basis that no a priori restriction can be placed on the possible bindings for a variable. In the above examples, there were 67 and 146 associations respectively after the grounding stage. In general, the number of maps at this stage would be of the order of the square of the length of the terms.

The deleting stage then has to perform fairly complex computations on this set of associations, and this is where most of the c.p.u. time is used. The complexity of the deleting stage is of the order of the square of the size of the match produced by the grounding stage. The adding stage is linear in the size of the intermediate match. Therefore, the whole algorithm is n^4 in the size of the original terms. Unlike many "worst case" analyses, this is also a typical case analysis, as the algorithm does not exploit features of particular terms to cut down its computation. Munyer's justification for his bottom-up approach to matching (as opposed to the top-down approach of unification algorithms) is that, since an analogy match may disrupt the structure of the terms, no top-down restrictions should be placed on associations. This philosophy contrasts with that of Kling [Kling 71], who uses the notion of semantic type to constrain the possible mappings between variables (see below). Kling's approach is to enforce type correspondence if possible, but to allow a small amount of deviation in order to find interesting analogies; in any case, in Kling's matches, variables are only mapped to other variables. Munyer has no type restrictions, and furthermore allows the binding of variables in his matches. This enables him to find a much wider range of analogies, in theory anyway, but he pays for this generality in c.p.u. time.

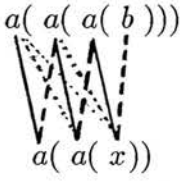
Having observed the inherent complexity of Munyer's approach, there are ways in which the algorithm could be guided which would not violate the approach:

- consistency: no attention is paid, during matching, to the consistency of the match, particularly the variable bindings which it entails. Matches with inconsistent variable bindings are rejected at a later stage of the analogy process (see [Munyer 81]). But the complexity would be reduced if only

consistent sets of variable bindings were considered in the first place. This would involve the development of several possible matches between a pair of terms which Munyer, for some reason, seems to want to avoid. The (relative) consistency of the predicate and variable mappings could be used to prune unpromising matches.

- syntactic types: Munyer's matcher blurs some syntactic distinctions which analogies tend to respect, for example the special status of the propositional connectives, and the distinction between predicates and functions: in all the analogies considered in this thesis, propositional connectives are only mapped to other logical connectives, predicates are only mapped to other predicates, and functions to other functions. One way of exploiting these syntactic type distinctions would be to construct only those associations which respect the syntactic types during grounding, and put in those which don't, if at all, during adding. These proposals introduce more typing into the representation, but only at a very general, domain-independent level.

The issue was raised above of the construction of more than one match between a pair of terms, which Munyer's algorithm cannot do. The following intermediate match after the grounding stage illustrates the usefulness of such a facility:



There are two main competing 'chains' of supporting associations, one drawn as dotted lines, and one as continuous lines. Munyer's original algorithm would delete both chains iteratively from the left, finishing the deleting stage with a null match. The version with improvement C would retain the continuous chain, because it connects the head function symbols, delete the dotted one, and thus come up with the first order unification. But there really seems to be a disjunction here — either of the chains represents a plausible match which

could be investigated further. The present criterion for deciding between them is unsatisfactory as it does not relate to the structure of the various possibilities. It seems sensible to split the match up into the disjuncts, at this stage, and investigate them separately.

Analysis in terms of heuristics

In the previous section, we saw how Munyer's matcher could be improved in certain ways, without departing from the overall, bottom-up approach which Munyer takes. However, there is a serious problem with Munyer's algorithm which will not be solved by any of the remedies which have been suggested. The problem can be understood in terms of the way in which the matcher uses its analogy heuristics: the **identical symbols** heuristic and the **partial homomorphism** heuristic. The matcher relies entirely on the identical symbols heuristic to propose initial associations (apart from variable bindings). Non-identical associations, which are not variable bindings, can only be added by the adding stage. The effect of this is that the matcher is unable to construct useful matches which involve a significant number of non-identical associations. The two examples given above to illustrate the computational cost of the algorithm also illustrate this defect. There are strong matches between the problem pairs, which Munyer's algorithm misses — this is because they involve significant non-identical associations. In the second example, the identical symbols heuristic is positively misleading, since the terms do contain common function symbols, but these are not associated in the strong match.

Generally, the identical symbols heuristic can only be considered as a fairly weak heuristic (see section 3.3.1 for further discussion of this, and of other heuristics). The partial homomorphism heuristic is stronger, however (see section 3.3.1). But no amount of clever computation at the deleting stage, which is based on the latter heuristic, can rectify the errors in the initial match constructed by the grounding stage, which is based on the former. Most interesting analogies will contain a significant number of non-identical associations; thus the heavy reliance of Munyer's matcher on the identical symbols heuristic will re-

strict the generality of the matcher, preventing it from finding many interesting and easy analogies.

Summary

Munyer's algorithm, even with the improvements which have been made to it, does not provide an adequate analogy matcher, even for simple analogies. The broad diagnosis of the problem with the algorithm is that it relies too heavily on a weak heuristic, the identical symbols heuristic. Quite apart from this, the algorithm is extremely expensive computationally, which makes the prospects for it scaling up to larger examples remote.

3.2.2 Kling

Kling's algorithm for matching up pairs of problem descriptions is called INITIAL MAP, and is part of his analogy system ZORBA. INITIAL MAP accepts as input two theorem descriptions, each of one of the forms

$$H_1 \,\&\, H_2 \,\&\, \ldots \,\&\, H_n \;\rightarrow\; C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_m \quad \text{or} \quad C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_m,$$

where each of the H's and C's is of the form

$$P(x_1, \ldots, x_r),$$

P being a predicate and the x's variables. The variables are implicitly universally quantified. Thus there are no functions or constants. Note that this is a more restricted syntax than that accepted by Munyer's matcher. Kling claims that this subset of predicate calculus is suitable for mathematics. This seems an odd thing to say, as normal mathematics has as many constants and functions in it as anything else. It is true that a much wider class of sentences can be transformed into semantically equivalent sentences of the above form (skolemize the existential variables; if P is a (unary) predicate, f a function symbol and $t_1, \ldots, t_n$ terms, replace $P(f(t_1, \ldots, t_n))$ by $f'(t_1, \ldots, t_n, x) \rightarrow P(x)$, where x is a new variable and f' is the functional relation associated with f (similarly for

n-ary functions); iterate the last step until there are no functions or constants (0-ary functions) left). What comes out of this procedure is a very unnatural looking sentence, and, importantly for analogy matching, the functionality of the new predicates is not represented in the syntax. Note also that, unless the functionality of the predicates is stated explicitly in the form of axioms, the theory which results from this transformation is weaker than the original — in particular it is decidable.

However, the theorems of abstract algebra on which Kling bases his matcher do fit naturally into this form.

INITIAL MAP also has access to **semantic templates** for each of the predicates in the sentences to be matched. For example, the predicate *group* has semantic template

$$structure(set, operation).$$

The intended meaning of this (although Kling does not explain it) seems to be that a group is a *structure* consisting of a *set* and an *operation*.

INITIAL MAP outputs an **analogy match** between the sentences, which consists principally of an association between the predicates in the sentences, and, as a by-product, an association between the variables. In Kling's system ZORBA, this match is then passed to the EXTENDER module, which applies the match to the axioms used in the proof of one of the theorems in order to find analogous axioms, extending the match in the process. The latter stages will be analysed further in Chapter 5.

The algorithm

The matching strategy of INITIAL MAP (I-M) is as follows: assume the hypotheses of one theorem match those of the other, and similarly for the conclusions (so the match breaks up into two sub-matches); do one of these first, and then the other, using the intermediate match obtained from the first to help the second; use the semantic templates to constrain the association of atoms

between the theorems, and use the existing predicate and variable associations to choose where there is ambiguity in the semantic templates.

Kling doesn't say in what order the two sub-matches are attempted. In the reconstruction described here, whichever of the conclusions and the hypotheses contains fewest atoms is attempted first, on the basis that it is usually easier to match smaller structures.

So, if the theorems to be matched are

$$A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_n \rightarrow B_1 \,\&\, B_2 \,\&\, \ldots \,\&\, B_m$$
$$\text{and } C_1 \,\&\, C_2 \,\&\, \ldots \,\&\, C_r \rightarrow D_1 \,\&\, D_2 \,\&\, \ldots \,\&\, D_s$$

and $m + s < n + r$, the module SETMATCH is called first on the conclusions, then the hypotheses.

SETMATCH breaks up into two submodules, SINGLEMATCH and MULTIMATCH, where SINGLEMATCH is first called to pair up the atoms in the sets which have unique semantic templates, and then MULTIMATCH is called to pair up the rest using the partial match generated by SINGLEMATCH to guide it.

When either SINGLEMATCH or MULTIMATCH decides to associate a pair of atoms, a procedure called ATOMMATCH is called to extract predicate and variable correspondences from the atoms, which are added to the evolving match. If the atoms have identical semantic templates, the associations are just those which preserve argument order. But Kling also wants to be able to find analogies in which, for example, groups are associated with rings — that is, $group(Set, Op)$ is associated with $ring(Set1, Op1, Op2)$ — and these predicates do not have the same semantic templates. Their templates are $structure(set, operation)$ and $structure(set, operation, operation)$ respectively. Kling's solution to this problem is as follows: he allows a variable in one of the atoms to be associated with a consecutive sequence of variables in the other as long as (a) all the types involved are the same and (b) the correspondences preserve argument order. In this case,

he would get

$$group \leftrightarrow ring, \quad Set \leftrightarrow Set1 \quad and \quad [Op] \leftrightarrow [Op1, Op2].$$

If conditions (a) and (b) cannot be satisfied, ATOMMATCH fails.

The job of SINGLEMATCH and MULTIMATCH, then, is to decide on pairs of atoms to pass to ATOMMATCH. SINGLEMATCH uses the following criteria in its decision:

1. If there is only one unpaired atom left on each side, associate them by default.

2. Associate pairs of atoms having the same semantic template.

3. Associate pairs of atoms whose predicate symbols are associated in the existing match.

4. Associate pairs of atoms whose predicate symbols have the same type, where this gives a unique association.

1 is checked after any association is made. Otherwise 2, 3 and 4 are checked in that order.

If SINGLEMATCH cannot pair up all the atoms with unique semantic templates on each side, it fails. If it can, the partial match produced is passed on to MULTIMATCH, which considers the atoms on each side which have the same semantic templates as others. First, the atoms are grouped into their semantic blocks; then blocks are paired up using the following criteria in the same way that SINGLEMATCH used its criteria:

1. If there is only one unpaired block on each side, associate them by default.

2. Associate pairs of blocks which contain atoms whose predicate symbols are associated by the existing match.

3. Associate pairs of blocks whose atoms have predicates with the same semantic type.

When a pair of blocks is associated by the above criteria, the submodule MULTIMATCH1 is called to pair up the atoms within the blocks. MULTIMATCH1 has a single criterion for doing this, based on the existing variable associations: if there is a pair of variables associated by the existing match, each of which occurs in only one atom in the appropriate block, then pair up the two atoms.

This completes the description of INITIAL_MAP.

Heuristics

What are the analogy heuristics which are used in Kling's algorithm? The most obvious guidance is given to the matcher by the semantic templates; associations are made which preserve the semantic type of predicates whenever possible. Thus Kling must believe that matches which tend to preserve semantic type make promising analogies. Let us call this the **semantic type heuristic**.

As well as the semantic types of predicates (for example, *structure* and *relsructure*[3]), the templates give types for the argument positions to the predicates (for example, set and operation). These argument types are used in the procedure ATOMMATCH, which insists that variable associations preserve the argument types. We call this the **argument type heuristic**. These types seem to be of a different kind to the semantic types of the predicates (although Kling does not discuss the meaning of either); the argument types could well be unary predicates from the object language, or the types within a typed logic, whereas the semantic types denote heuristic sets of predicates from the object language. This is why the two heuristics are distinguished here.

Thirdly, the **partial homomorphism heuristic**, used by Munyer, is also used by Kling; the overall top-down structure of the algorithm ensures that the symbol associations made in a match preserve the syntactic structure of the terms being matched; furthermore, the default rules in SINGLEMATCH and

---

[3] *relstructure* seems intended to denote a substructure.

MULTIMATCH are also based on the belief that analogy matches ought to be structure-preserving — the association between atoms made by default fills a structural hole in the associations which have already been made. The default rules in INITIAL MAP are similar in spirit to Munyer's adding stage.

SINGLEMATCH, MULTIMATCH and MULTIMATCH1 all contain criteria for pairing up atoms, or blocks of atoms, on the basis of existing associations in the intermediate match; pairings are made so that the associations are likely to be repeated — that is, an attempt is made to keep the match as **consistent** as possible. Thus, Kling must believe that consistent matches make promising analogies. Let us call this the **consistent translation heuristic**.

Lastly, note that INITIAL MAP never considers associating symbols unless they have the same **syntactic type**; i.e. propositional connectives are only ever associated with propositional connectives, and similarly for predicates and variables (unlike Munyer's algorithm, INITIAL MAP makes no attempt to bind variables in matching). This is not done by explicit criteria, but is built into the structure of the algorithm; let us call this the **syntactic type heuristic**. Given the restricted syntax which the matcher accepts, we can think of this heuristic as being implied by the partial homomorphism heuristic (or vice-versa); however, in more general situations, the two would be separate, which is why they are listed separately here.

Reconstruction

The reconstruction of INITIAL_MAP in Prolog turned out to be straightforward — considerably easier than Munyer's algorithm.

Performance and modifications

The reconstruction was first tested on the examples from abstract algebra that Kling gives in [Kling 71]. The performance was satisfactory on most of these.

The following is one of Kling's theorem pairs and the resulting match obtained by the program:

HYPOTHESES	*HYPOTHESES*
$group(g,*)$	$ring(r,*,+)$
$propernormal(m,g,*)$	$properideal(n,r,*,+)$
$factorstructure(x,g,m)$	$factorstructure(y,r,n)$
$simplegroup(x,*)$	$simplering(y,*,+)$
CONCLUSION	*CONCLUSION*
$maximalgroup(m,g,*)$	$maximalring(n,r,*,+)$

$$
\begin{aligned}
maximalgroup &\longleftrightarrow maximalring \\
m &\longleftrightarrow n \\
g &\longleftrightarrow r \\
[*] &\longleftrightarrow [*,+] \\
factorstructure &\longleftrightarrow factorstructure \\
x &\longleftrightarrow y \\
propernormal &\longleftrightarrow properideal \\
group &\longleftrightarrow ring \\
simplegroup &\longleftrightarrow simplering
\end{aligned}
$$

SETMATCH is first called on the conclusions, and SINGLEMATCH immediately defaults to ATOMMATCH, since there is only one atom in each conclusion. The semantic templates of the conclusions are not identical, but they do match in the weak sense outlined above, giving the first four elements of the match. SETMATCH is then called on the hypotheses, with the partial match being passed along. The atoms in the hypotheses which have unique semantic templates,

$$\{propernormal(m,g,*),\ factorstructure(x,g,m)\}$$

from theorem 1, and

$$\{properideal(n,r,*,+),\ factorstructure(y,r,n)\}$$

from theorem 2, are passed to SINGLEMATCH.

Condition 2 from SINGLEMATCH pairs up the factorstructure atoms and passes them to ATOMMATCH which adds two more associations to the analogy. Then the default condition in SINGLEMATCH pairs up the remaining two atoms

and passes them to ATOMMATCH which adds one more association to the match.

Next, MULTIMATCH is called on the two atom partitions which have identical semantic templates,

$$\{group(g,*),\ simplegroup(x,*)\}$$

and

$$\{ring(r,*,+),\ simplering(y,*,+)\}.$$

These are paired up by the default condition, and are passed to MULTIMATCH1 to pair up the atoms within the partitions. The existing $g \leftrightarrow r$ variable association is used to pair up one atom from each partition, and the $x \leftrightarrow y$ association to pair up the remaining atoms, giving the final match.

While the program performed as desired on Kling's algebra examples, the heavy use of the semantic templates was worrying. All of Kling's theorem pairs come from essentially the same global analogy between group theory and ring theory. It seemed that it was Kling's knowledge of this global analogy which led him to choose the particular semantic templates which he did for the examples. Thus it is no surprise and no great achievement that the algorithm finds the correct matches in these cases. This issue is discussed further below.

After going through Kling's algebra examples, the reconstruction was tried on some new examples. One of these was the following pair [4]:

$$-(x+y) = -x + -y \quad and \quad 1/(x\,.\,y) = 1/x\,.\,1/y$$

---

[4]Note that the second equation is not true if interpreted within the real numbers; we have omitted the conditions which would make it true in order to make it as easy as possible for the matcher. In any case, the equation is true of the non-zero reals

When transformed into Kling's syntax, these become:

Example A

$$
\begin{array}{ll}
(times(x,y,z) & (plus(x,y,z) \\
\& \; inv(z,z_1) & \& \; minus(z,z_1) \\
\& \; inv(x,x_1) & \& \; minus(x,x_1) \\
\& \; inv(y,y_1) & \& \; minus(y,y_1) \\
\& \; times(x_1,y_1,r)) & \& \; plus(x_1,y_1,r)) \\
\quad \rightarrow equal(z_1,r) & \quad \rightarrow equal(z_1,r)
\end{array}
$$

where new functional predicates are introduced to replace the functions. Very general types were chosen — 'relfn' for all the functional predicates and 'rel' for equal/2. This is a very easy analogy, involving 1-1 associations of atoms, predicates and variables; however, the matcher was unable to find the correspondence. On investigation, it can be seen that the problem comes when MULTIMATCH is called on the hypotheses. Condition 3 (see above) does not apply, since both blocks in each partition have predicates of the same type, and thus MULTIMATCH fails. There seems to be no reason why MULTIMATCH should not have the extra semantic template condition that SINGLEMATCH does (condition 2); this would solve the present problem, and lead to the correct match. This change was made to the program. Furthermore, performance is not impaired on any of the previous examples. It seems odd that Kling should have omitted this condition, since it seems a natural consequence of his matching strategy; but, as mentioned above, Kling admits that the matcher was designed around two of the algebra examples, so it may be just that the extra condition was not necessary for these.

Another example on which the matcher was tried was:

$$rational(p) \; \& \; irrational(q) \rightarrow irrational(p+q)$$

$$rational(p) \; \& \; irrational(q) \; \& \; unequal(p,0) \rightarrow irrational(p \,.\, q).$$

This is one of the examples considered in Chapter 1, except that irrational is used instead of not(rational) and unequal instead of not(equal), since the matcher

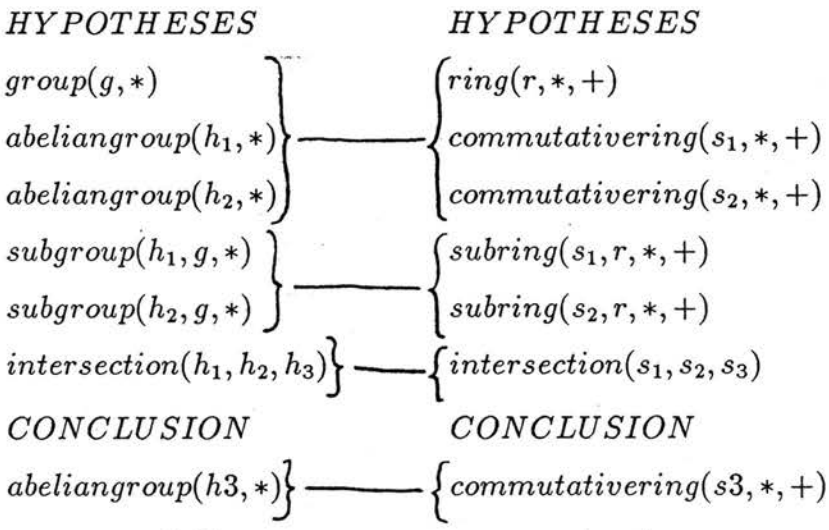
cannot handle negation. After transformation, the sentences become:

$rational(p)$	$rational(p)$
$\& \; irrational(q)$	$\& \; irrational(q)$
$\& \; plus(p, q, r)$	$\& \; unequal(p, 0)$
	$\& \; times(p, q, r)$
$\rightarrow irrational(r)$	$\rightarrow irrational(r).$

General semantic templates were again used. The matcher fails on this example, since it requires a 1-1 association of atoms in order to succeed. The failure occurs in SINGLEMATCH on the hypotheses; the plus atoms are associated by the second condition, leaving the unequal condition on the right hand side and nothing on the left hand side, which causes failure. But sometimes (as in this example) it seems right to leave an atom unmatched — extra conditions are common in analogies. A reasonable rule would be to allow SINGLEMATCH to succeed if one set is empty and the other contains just one atom. With this change implemented, the matcher goes on to find the right match in this example. Again, performance on previous examples is not impaired. More liberally, we could allow any number of unmatched atoms within SINGLEMATCH, but this was not implemented.

Another example where the performance of the matcher was not as good as

expected is shown below:



$$
\begin{aligned}
abeliangroup &\leftrightarrow commutativering \\
h3 &\leftrightarrow s3 \\
[*] &\leftrightarrow [*,+] \\
intersection &\leftrightarrow intersection \\
h1 &\leftrightarrow s1 \\
h2 &\leftrightarrow s2 \\
subgroup &\leftrightarrow subring \\
g &\leftrightarrow r
\end{aligned}
$$

The match is almost complete, but lacks the expected $group \leftrightarrow ring$ association. First, the conclusions are SETMATCHed; the intersection atoms are then SINGLEMATCHed; MULTIMATCH pairs up the semantic blocks as shown, and passes them to MULTIMATCH1; this is first applied to the three-atom blocks, pairing up the abeliangroup and commutativering atoms using existing variable associations, but the remaining atoms, $group(g,*)$ and $ring(r,*,+)$ cannot be paired up by MULTIMATCH1 — this is the cause of the incompleteness in the final match; MULTIMATCH is then called on the two element blocks and finishes the matching.

There seem to be two possible solutions to this problem:

- MULTIMATCH1 could be given a default condition, like those in SINGLEMATCH and MULTIMATCH. The group and ring atoms would then be ATOMMATCHed by default and the final match would be complete.

- The applications of MULTIMATCH1 to the various pairs of semantic blocks could be interleaved in the following way: if MULTIMATCH1 can't

pair up some of the atoms between two blocks, it should leave them unmatched temporarily, and work on another pair of blocks; having matched as many atoms as it can within the new pair, it should return to the old pair and try to use newly added variable associations to pair up more atoms. In this example, when MULTIMATCH1 is applied to the second pair of blocks, the $g \leftrightarrow r$ association is added, which enables the group and ring atoms from the first pair to be associated.

Both of the above changes will be applicable in situations where the other one is not. So, although only one is needed for this example, it is reasonable to propose both changes. Of the two, the second seems more reliable — a more sophisticated application of the existing matching strategy. The problem with having many defaults in the matcher is that it will tend to match pairs of short sentences, almost entirely by default, which are not in fact very similar. The matcher will not be able to discriminate so well. Having said that, Kling uses defaults heavily anyway, and adding one to MULTIMATCH1 seems consistent with his matching strategy. Again, it is probably just because Kling's target examples didn't need it that he did not put in the default himself. As before, these changes would not affect the matcher's performance on previous examples, although they were not actually made to the reconstruction.

Further criticisms

The following are further criticisms of INITIAL MAP which are not based on failure in any one example, but have arisen from scrutiny of the matching procedure over previous examples:

- As explained above, ATOMMATCH only makes variable associations which preserve argument order. While this is understandable if the predicates of the atoms are the same, it is not otherwise — the relative order of arguments between two different predicates is the result of arbitrary decisions made when the domain was formally represented, and should therefore have no influence over analogy matching. Furthermore, if the predicates

are commutative in some of their arguments, the particular presentations chosen for the atoms determine the possible variable correspondences that ATOMMATCH can make; that is, no account is taken of the commutativity which might allow ATOMMATCH to permute the order of the variables in the atoms. In the examples given above, where functional predicates have occurred, we have presented the atoms so that corresponding variables (in the intended match) occur in the same argument positions; thus, we have made it as easy as possible for the matcher. Had we not done this, the deficiencies of ATOMMATCH would have been compounded with the other problems which were discussed.

- MULTIMATCH1 makes use of existing variable associations only, and in a fairly simple way — this is all that was necessary for Kling's target algebra examples. There seems to be no reason why it should not use existing predicate associations as well, as SINGLEMATCH and MULTIMATCH do. This would make it more powerful. The way in which MULTIMATCH1 uses the variable associations could also be strengthened so that it can make use of multiple associations, like $* \leftrightarrow [*, +]$, as well as single associations.

- Conversely to the previous point, there seems to be no reason why SINGLEMATCH and MULTIMATCH should not use existing variable associations, as well as predicate associations, to pair up atoms and blocks of atoms.

A more serious problem with the matcher is shown up by the following example:

Equation	In Kling's syntax
$\sqrt{x} + a = b$	$root(x, z)\ \&\ plus(z, a, y) \rightarrow equal(y, b)$
$\sqrt{x+1} + a = b$	$plus(x, 1, y)\ \&\ root(y, z)\ \&\ plus(z, a, w) \rightarrow equal(w, b)$

In the first sentence, all the atoms have unique semantic templates, whereas in the second the two plus atoms are in a semantic block together. Thus, SINGLEMATCH pairs up the two root atoms and the two equal atoms, and leaves

the plus atom in the top sentence un-matched. Then MULTIMATCH is called between a single block on one side and nothing on the other, and so fails.

It is the basic SINGLEMATCH/MULTIMATCH distinction which is at fault here: where there is an atom which ought to be unmatched, its semantic template may coincide with that of another which ought to be matched by SINGLEMATCH, which will prevent the latter from ever being considered by SINGLEMATCH. It seems likely that this situation will occur often in non-trivial analogies, and thus that the matching strategy of INITIAL MAP will have to be changed to cope.

Assessment

Given the user-specified semantic templates, INITIAL MAP performs quite well on the algebra examples around which Kling designed it. Its top-down matching strategy, and the heavy guidance it receives from the semantic templates, make INITIAL MAP efficient — much quicker than Munyer's algorithm.

However, as indicated above, the heavy and inflexible reliance on semantic templates is worrying:

- The expectation that matches will preserve type is built in to the overall structure of the matcher, in particular the SINGLEMATCH/MULTIMATCH distinction. As explained above, this rigid structure restricts the generality of the matcher, causing it to fail on fairly easy examples, even when the intended match would preserve types. In order to avoid this problem, the main procedures of the algorithm would have to be rewritten.

- It is a consequence of the design of the algorithm that almost all of the predicate associations which are made will preserve semantic type; it is possible for non-type preserving associations to be added by the default rules, but not significantly many. But we can think of the types as (partially) encoding potential analogies; for example, the type structures shown in Figure 3–1 suggest an analogy which includes the associations $group \leftrightarrow ring$ and

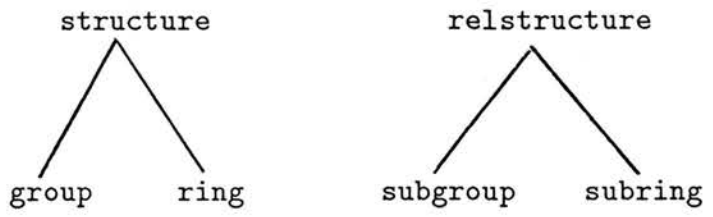


Figure 3–1: Algebraic type structures

subgroup ↔ *subring*. So INITIAL MAP is unable to extend significantly the analogies which are encoded in its type structure. Thus it is doubtful whether the matcher can be said to discover analogies at all — its use of types seems to involve a circular argument. As explained above, on the algebra examples, the matcher repeatedly reconstructs parts of the same global analogy between group theory and ring theory, which is the analogy suggested by the semantic templates. The problems with the use of the semantic types in matching are discussed further in section 3.3.1.

Analysis in terms of heuristics

From the point of view of the discovery of new analogies, the shortcomings of Kling's matcher can, like Munyer's, be understood in terms of the way in which it uses its analogy heuristics. The matcher relies on the **semantic type heuristic** to propose initial associations between atoms, and only uses the other heuristics to add new associations on the basis of the existing ones. Thus, just as Munyer's matcher is unable to break free of its reliance on the identical symbols heuristic, Kling's is unable to break free of the semantic type heuristic.

But some analogies will preserve a given set of types, some will respect them (in the sense of inducing a consistent mapping between types in the base and target), and others will do neither. We would like a matcher to be able to use the type information in a flexible way, along with other criteria for analogy, so that matches which respect types may be preferred, but those which don't will not be ruled out.

Summary

Kling's matcher has turned out to be inadequate for many of the simple analogies which we would hope that a matcher could find. Perhaps this is not surprising, since the matcher was designed around just a few algebra examples, for which it was provided with much of the information about the intended analogies.

We have seen that there are various straightforward alterations which can be made to the algorithm to improve its performance while not affecting its overall strategy. However, it has also become clear that, in order to improve the algorithm much further, its reliance on the semantic type heuristic, including the SINGLEMATCH/MULTIMATCH distinction, which is the basis of the algorithm, would have to be dropped. This would entail a complete redesign of the matcher. The distinction, and hence the matching strategy, is based on the rigid use of types and 1-1 associations between atoms; where these cannot be guaranteed, the matcher will be inadequate.

3.3 Summary of reconstructions

3.3.1 Heuristics used and justifications

Figure 3–2 lists the heuristics which have been found to underlie the matchers of Kling and Munyer, and which matchers use each of them.

We next consider what justification there is for making use of each of these heuristics.

Partial homomorphism Firstly, note that almost all computational or quasi-computational accounts of analogy are based around a notion of structural matching, which is what the partial homomorphism heuristic recommends. For example, in her **structure-mapping theory**, considered in Chapter 2, Gentner introduces the notion of the **systematicity** of a structure-mapping; this is the

Heuristics	Munyer	Kling
partial homomorphism *(strong)*	✓	✓
consistent translation *(strong)*	✗	✓
semantic type *(weak/circular)*	✗	✓
identical symbols *(weak)*	✓	✗
syntactic type *(strong)*	✗	✓
argument type *(variable)*	✗	✓

Figure 3–2: Heuristics used

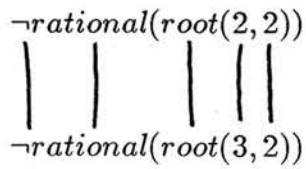
extent to which matched objects are mutually constrained by identical relations. Apart from the strange restriction to identical relations, Gentner does not consider the looser forms of structural correspondence which arise in mathematics, and which both Munyer and Kling are concerned with. Both these restrictions of the structure-mapping theory seem related to the fact that Gentner designed the representations of the physical domains herself, and thus was able, by anticipating the analogy, to finesse away many of the matching problems. Interestingly, the partial homomorphism heuristic is the only one shared by both Kling's and Munyer's matchers.

But why is it a good idea to look for matches which respect the structure of the terms being matched? Although most authors have considered structural matching crucial to analogy, there has been very little discussion of its justification. However, within automated problem solving, we can understand why structural matching is important. The goal of analogy is to use the base proof to guide the search for a target proof. The base proof consists of a sequence (or tree structure) of applications of axioms or lemmas to intermediate clauses,

starting with the problem and ending with a recognised goal state (the empty clause, for instance). The applicability of an axiom to a step is determined by unification, a strict structural matching procedure. The hope in analogy is that we will be able to make corresponding steps in the target. In the target too, we will need structural matches between axioms and intermediate steps. If we can find a structurally similar problem to the target problem, we can reasonably hope that, if we can also find structurally similar axioms to those used in the base, there will be a good chance of being able to make corresponding steps in the target — that is, that the corresponding clauses will unify and produce corresponding inferences. If this is so throughout the proof, the application procedure (see Chapter 6) will have no difficulty in solving the target problem by analogy to the base problem. Of course, many things can go wrong: we may not be able to find similar axioms to apply in the target, or, even if we can, the corresponding steps may not go through because of the loose structural matches (see Chapter 7 for a discussion of how the application procedure can get into difficulties). But a structural match between problems is still an important factor. To put it another way, if we do *not* have a structural match between problems, it is almost certain that the application of the analogy will be difficult.

Therefore, we consider the partial homomorphism heuristic to be strong. It is important that our notion of structural similarity be flexible, in order to encompass the sorts of structural matches between even closely analogous problems which occur in mathematics.

Consistent translation heuristic Again, the notion of consistency of the translation implied by an analogy match is taken to be important in almost all accounts of analogy, but, again, without much attempt at justification. Many accounts, such as Gentner's [Gentner 82] and Winston's [Winston 80] require that the mapping be strictly consistent. But others, such as that of Gick and Holyoak [Gick & Holyoak 80], accept that many analogies contain some inconsistencies, and that to insist on strict consistency would considerably restrict generality. In Chapter 1, we saw that simple problem matches such as



involve inconsistencies. Kling's matches can have inconsistent predicate mappings, via the default rules in SINGLEMATCH and MULTIMATCH, although this happens rarely in practice[5]. Munyer's matcher pays no explicit attention to consistency, although its reliance on the identical symbols heuristic means that inconsistencies can only be introduced at the adding stage.

So why should we prefer consistent matches at all? Again, by considering automated problem solving, we can understand why we would like matches to be as consistent as possible. In the discussion on the importance of structural similarity in predicting inferences to make on the target problem, we omitted another important feature of unification: it is not only a strict structural match, but also a strict match on the identities of predicates, functions and constants. Since, in most interesting analogies, there will be non-identical associations, there is a danger that target inferences will fail to go through, not because of structural mismatches, but because of symbol mismatches. But, if the mapping is consistent, we will have greater confidence that this will not happen — if there is a consistent translation of symbols between base and target, where symbols were the same in the base, they will be the same in the target. Although we would like a strictly consistent match for this reason, restricting a matcher to such matches would restrict its generality. More realistically, we would like our match to have as few inconsistencies as possible, but we may have to put up with some.

So we consider the consistent translation heuristic to be strong, but we must be flexible about its application.

---

[5]In fact, inconsistent predicate mappings are rejected by Kling's system ZORBA, since the application stage of the system cannot cope with them

Syntactic type heuristic It is implicit in all existing computational accounts of analogy that syntactic types will be respected by analogy matches — predicates are never associated with functions, and propositional connectives are only ever associated with each other. Perhaps even this should not be treated as an absolute criterion — it would be quite possible for a predicate on one side to be associated with a function on the other. But in practice, it seems a reliable heuristic. The application of an analogy which did not respect syntactic types would be difficult.

We can therefore consider this to be a strong heuristic.

Argument type heuristic Kling insists that matches preserve the types of the arguments of predicates. Whether or not this is a good heuristic depends on the particular types which are used; with Kling's algebra predicates, the main argument types are *set* and *operation*. We would certainly feel more confident in being able to apply an analogy which matched sets with sets and operations with operations, since the axiomatisations are likely to involve checks on these types. Thus, in this case, we could consider the argument type heuristic to be quite strong. However, in some other case, argument types may not be a good clue to analogy at all. Thus, the strength of the heuristic is variable. Brown [Brown 77] uses just this notion of type in his matching procedures; however, Brown's matches do not necessarily preserve the types, indicating the variability of the heuristic.

Identical symbols heuristic This heuristic is used heavily in Munyer's matcher, as explained above, but not at all in Kling's. Some accounts of analogy, such as those of Gentner [Gentner 82] and Sternberg [Sternberg 77] have insisted that all associations between predicates (though not constants) be identical. Munyer's matcher does allow non-identical associations to be added at the adding stage, although, because of the structure of the algorithm, this has limited effect.

While it is clear that most interesting analogies involve a significant proportion of non-identical associations (some authors define analogy as necessarily

involving differences in the surface representations), the heuristic nevertheless has some value: we should not forget that the aim of an analogy system is to solve the target problem; it should use the most similar problem which it can find as base. Other things being equal, the presence of identical associations can only be a good thing; it will increase our confidence that we can find structurally similar axioms to apply in the target (the *same* axioms, with luck).

But, bearing in mind that many examples involve genuine translations, we can only consider the identical symbols heuristic to be relatively weak.

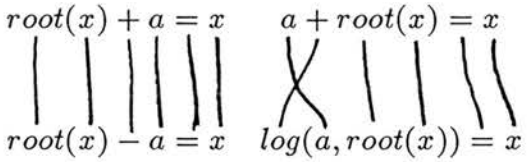
Semantic type heuristic The discussion of this heuristic has been left until last because it is the most problematical to assess. The reason is that it is the only one that requires extra information to be provided to the analogy matcher, apart from the problem statements, and other information available in the object language, such as argument types. Thus we need to assess not only whether the preservation of semantic type is a useful property for an analogy match, but where the type information comes from and what information it represents.

The term "semantic type" is taken in AI to refer to groups of symbols from the object language which are believed to be related to each other in some (usually unspecified) way.[1] Often, the semantic types are organised into a **hierarchy**, determined by the subset relations between the groups. In automated problem solving, type hierarchies have been extensively used in the field of **inductive generalisation** [Mitchell 78], discussed in the previous chapter. It is generally recognised that semantic types are heuristic devices, rather than conforming to a formal definition; but little attention has been paid to the origin of the hierarchy, although some work has been done on modifying the hierarchy over the course of problem solving experience [Utgoff 83].

So it is initially unclear what sort of information is being provided to an analogy matcher in the guise of semantic types. However, by returning to the automated problem solving argument expounded above, we can identify the conditions under which a set of semantic types would be *useful* in guiding a matcher; i.e. conditions on the types which would make the semantic type heuristic suc-

1. In the sequel, the term "semantic type" simply refers to such groupings of symbols.

cessful. It is clear in the argument that, in order to apply an analogy easily and successfully, the analogy system needs a structural correspondence not only between the problem statements, but also between the axioms which were used in the base proof and other axioms from the axiom base; it is under these conditions that we can reasonably expect to make a corresponding inference in the target to one which was made in the base. The partial homomorphism and consistent translation heuristics tell us nothing about how likely it is that we will be able to find analogues for the axioms; the identical symbols heuristic does (if the symbols are the same, we can reasonably guess that the same axioms will apply), although it only handles the case where most of the symbols are mapped to themselves. In more interesting cases, such as Kling's algebra examples, where most of the associations are non-identical, the heuristics above tell us nothing about potential analogues for the axioms. For example, given the solution of the equation $a + root(x) = x$ as target problem[6], we would not be able to distinguish between the following two potential matches with base equations:



That is, in terms of structural correspondence, consistency and the identity or difference of matched symbols, the two matches above are identical; we call this inability to distinguish between potential matches the **multiple matches problem**. Intuitively, however, the first of the two seems more promising; this is because the functions $+$ and $-$ are thought to be more closely related than are $+$ and *log*. This feeling is born out by the fact that the first two can be solved in very similar ways (move the a to the right hand side, square to remove the *root*, and solve the resulting quadratic in x), whereas this simple strategy does not apply to the logarithmic equation. This is the sort of guidance we would

[6] $\sqrt{}$ is written as *root* to make the syntactic structure clear.

hope to get from the semantic type heuristic; the first match is chosen because it preserves the type of arithmetic operators, whereas the second does not.

From the point of view of the analogy system, what makes the first match above fruitful is the presence of analogues for the axioms used in the solution of the base equation; in particular, the identity $(x+y)-y=x$ was applied to move the a to the right hand side, in the base; the target proof proceeds analogously in the presence of an analogous identity $(x-y)+y=x$.

In general, we claim

> The extent to which semantic types are useful in analogy matching is the extent to which symbols of the same type have analogous axioms/theorems associated with them.

To put it another way,

> The extent to which semantic types are useful in analogy matching is the extent to which they encode potential analogies within or between domains.

This analysis introduces a dilemma: if the types do not encode analogies, they will not be useful in analogy matching; if they do, their use in analogy matching apparently is based on a circular argument, i.e. the matcher is being told, albeit in disguised form, about the analogies which it is supposed to discover for itself. However, we can avoid the circularity by one or both of the following means:

- The semantic types should be used *flexibly* by the matcher, so that it is able to extend and/or override the type information, if other of the analogy heuristics suggest that this is a good thing to do.

- The types could be learnt automatically in some way. Since, as we have seen, useful types encode analogies, the types could represent a succinct summary of the knowledge which the analogy system has gained about analogies within the domain in question. This would provide a means for

the analogy system to improve its ability to reason analogically within the domain over the course of its experience. This issue is taken up again in Chapter 7.

But types could perhaps be derived in some other way as well; while not relating directly to potential analogies, they might nevertheless provide clues to the matcher.

The first alternative really ought to be chosen whether or not the second is pursued: if learnt semantic types are used inflexibly, they will never be extended or overridden, which will prevent any further learning from taking place.

Kling's matcher does neither of these things: the types are provided by the user, and they are used inflexibly. Thus the suspicion that Kling's use of semantic types (in his algebra examples, anyway) amounts to a circular argument is confirmed.

In conclusion, the semantic type heuristic can provide useful guidance to an analogy matcher, without involving a circular argument. But it can only be considered as a fairly weak heuristic compared to the partial homomorphism heuristic and the consistent translation heuristic, from the point of view of the discovery of *new* analogies. From the point of view of finding a successful analogy, whether a new one or not, the usefulness of the heuristic depends on the reliability of the types. The techniques proposed in Chapter 7 suggest ways of assessing the strength of semantic types; this information could be fed back to the analogy matcher, which would weight the heuristic accordingly.

3.3.2 Conclusion

As the arguments presented above show, all of the heuristics which have been found to underly the analogy matchers of Kling and Munyer have validity, and can provide useful guidance to a matcher. The problem with the particular algorithms which we have considered, then, is not the heuristics which they use, but the way in which they use them. Each matcher relies far too heavily on

the weak heuristic which it uses, the identical symbols heuristic, in the case of Munyer, and the semantic type heuristic, in the case of Kling; these are used as the source of initial, plausible associations. The stronger heuristics are only used conditionally — to propose or retain an association on the basis of an existing association. This strategy for the use of the heuristics restricts the generality of the matchers considerably — they are only able to find matches which conform closely to the weak heuristics.

Quite apart from this, we have seen that the existing matchers are not particularly successful at finding even matches which do conform to the weak heuristics. Their performance has been improved somewhat by the alterations that have been described. But Munyer's algorithm appears unavoidably expensive, and Kling's is too strongly based around the enforcement of types even for some matches which do preserve types, but which involve unmatched atoms.

Chapter 4

An Improved Analogy Matcher

In this chapter we describe a new approach to analogy matching, and a particular matcher which exemplifies it. The approach can be considered to be a generalisation, in several important respects, of Kling's algorithm. We first motivate the approach and the algorithm by reference to the shortcomings of existing algorithms, as described in the last chapter. The features of the algorithm are introduced gradually, each being motivated as it is introduced. A brief summary of the algorithm is given in section 4.3. The performance of the new matcher is then illustrated on some examples, and compared to Munyer's and Kling's algorithms. Finally, we consider the limitations of the approach to analogy matching taken by all of the matchers considered so far, including the new one — namely matching between the problem statements almost in isolation.

4.1 Introduction

In the last chapter we analysed the shortcomings of the matching algorithms of Munyer and Kling, particularly with respect to their strategy for applying their analogy heuristics. The overall conclusion was that neither of the algorithms provides a robust and useful analogy matcher. The broad diagnosis of

their problems is that they rely too heavily and inflexibly on weak heuristics, the **identical symbols** heuristic in Munyer, and the **semantic type heuristic** in Kling. Thus, while certain modifications have been suggested to the algorithms which improve their performance to some extent, the analysis in terms of heuristics suggests that a more fundamental redesign is necessary, in order to produce a matching algorithm which can be a useful part of an analogy system.

The broad diagnosis of the problem, too much reliance on weak heuristics, suggests a cure: namely, to use the heuristics, particularly the weak ones, more flexibly, allowing each to be overridden if others suggest that this would lead to a promising match. Furthermore, it is clear from the previous chapter firstly that different people have different ideas about the analogy heuristics that should be used, and secondly that definitive answers as to the relative strengths of the heuristics are hard to come by. Thus as well as making more flexible use of the heuristics, it is desirable to be flexible about which heuristics to use, and what relative significance to give them. The latter two features tend to follow from the former, since, if the heuristics are being used flexibly, this suggests that the design of the matcher is not biased towards particular weak heuristics, and therefore is relatively easy to modify by either changing the heuristics which are used, or changing the relative significance of existing heuristics.

So we have the following requirements for an improved matcher:

- The matcher should use its heuristics flexibly, rather than being biased in design towards any single weak heuristic.

- It should be easy to modify or extend the matcher, on the basis of changing beliefs about the heuristics; this may include tailoring the matcher to a particular domain, by the addition of domain-specific heuristics.

- The matcher should accept any terms from the language of the problem solver on which the analogy system is based: Kling's restriction to a subset of first order logic, containing no functions or constants, would be a severe restriction on a practical analogy system.

In the next section, we introduce the design features of the new matcher gradually, explaining the motivation for each. Then, in section 4.3, we give an algorithmic summary of the matcher.

4.2 Design of the matcher

4.2.1 Dual use of heuristics

Each analogy matcher involves a notion of the space of a priori possible matches between terms. The analogy heuristics have two main uses:

(a) To guide the matcher towards promising matches — i.e. to prune the search in the space of possible matches.

(b) To assess the strength, or promise, of completed matches between terms.

The algorithms discussed so far have been wholly concerned with (a). Munyer has a separate algorithm for computing the strength of completed analogy matches (see below); Kling does no assessment.

It would be possible to take the following brute force approach to analogy matching: generate all possible matches between a pair of terms, assessing each and returning the best one or few; i.e. search the space of possible matches exhaustively, only using the heuristics in (b). Winston [Winston 80] employs this approach, although he admits that it would be inadequate for larger examples than those which he has considered so far.

In order to assess whether Winston's approach would be feasible for the sort of analogies we have in mind, we should consider the size of the space of possible matches associated with such analogies. We can attempt to estimate the size of the space between the following formulae from Boolean algebra:

$$x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$$

$$x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$$

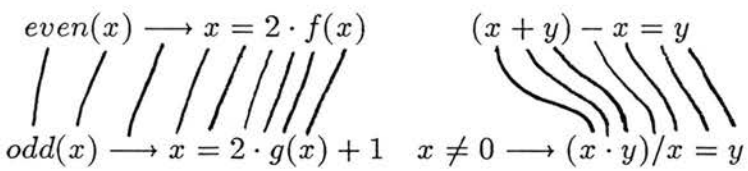
A match is an association between symbols in the two formulae; i.e. a subset of the cartesian product of the sets of symbol occurrences in the formulae. Each formula contains 13 symbols, so the cartesian product contains $13 \cdot 13 = 169$ elements, and thus has 2^{169} subsets. To use the brute force approach, we would clearly need to put restrictions on the type of associations acceptable as analogy matches. What restrictions are reasonable?

Winston restricts to 1-1 (i.e. consistent) associations of objects in his representation system; but we have seen that consistency can only be used as a relative criterion without restricting the generality of the matcher. Thus, we cannot make this restriction. We have likewise seen that we cannot restrict to identity pairings between predicates and functions.

It would be reasonable to restrict to associations which respect syntactic type; in this case, this restriction would bring the space down to $2^{(1^2)} \cdot 2^{(12^2)} = 2^{145}$ (the single predicate $=$ could only be mapped to itself). We could also restrict to matches which respect the structure of the terms in the following weak sense:

$$\text{for local maps } (a_1, b_1) \text{ and } (a_2, b_2), \text{ if } a_1 \geq a_2 \text{ then } b_1 \geq b_2.$$

This notion of preservation of structure leaves room for much structural deformation in a match, such as the following matches



There are degrees of greater structural preservation within this class of matches. Munyer's algorithm can, in theory, construct matches which do not satisfy this restriction. But he gives no real examples of such matches — all of his mathematical examples do satisfy the condition. We regard this as a reasonable restriction for an analogy matcher working within a problem solving system; the matching approach described in the next section only produces matches of this form. But it is hard to estimate the reduction in the space of matches produced by this restriction. We can give a worst case analysis for terms of size n (i.e. n symbol

occurrences): for given n, two flat terms of size n produce the maximal number of matches, since there is minimal term structure to restrict possibilities; the number is

$$2 \cdot \sum_{r=1}^{n} \frac{n!}{r! \cdot (n-r)!} \cdot \frac{n!}{(n-r)!} \; + \; 2 \cdot n.$$

So the number of matches is, in the worst case, worse than factorial in the size of the terms. A program which enumerated the possible matches produced 12 possible matches between $f(a)$ and $g(b)$, 88 between $x \cup y$ and $x \cap y$, and ran out of space when called on the terms $x \cap (y \cup z)$ and $x \cup (y \cap z)$. Thus, while we cannot estimate the size of the space of matches between the original terms analytically, it is clear that it will be far too big to be searched exhaustively — apart from enumerating the possibilities, the brute force approach would have to assess each according to the criteria, which in itself requires significant computation (see below).

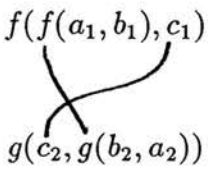
So we can conclude that guidance through the space of possible matches is necessary; i.e. that the heuristics need to be used in (a) as well as (b).

4.2.2 Design criteria

One part of Kling's matcher which seems quite promising is his pairing up of the arguments to logical conjunction, which are always atoms in his restricted syntax. Although he relies too heavily on semantic templates to suggest the pairings, which contributes to the overall inflexibility of his matcher, the associativity and commutativity of conjunction are implicit in the process, so that there are no preconceptions as to which atoms are paired with which — the pairings are determined by analysing the atoms using heuristics. By contrast, Kling's procedure ATOMMATCH, for extracting variable associations from a pair of atoms which have been associated, is based on arbitrary preservation of argument order, which is totally unsatisfactory.

The matcher developed in the current section makes the following generalisation of the promising part of Kling's algorithm: a flexible, heuristic analysis of possible argument (or subterm) pairings should be done at *every* stage of

matching, whether at the propositional or functional level. This generalisation will enable the matcher to work with an unrestricted syntax. The analysis is done with **pairing criteria**, which are based on the analogy heuristics. The criteria are coordinated flexibly to produce subterm pairings. For example, if the following terms are to be matched, the matcher might make the argument pairings shown:



In general, a given set of argument or subterm pairings will lead to a corresponding set of matching subproblems. If, for the moment, we represent a matching subproblem as a pair (T_1, T_2) of subterms to be matched, the argument pairings shown above cause the matching problem

$$(f(f(a_1, b_1), c_1),\ g(c_2, g(b_2, a_2)))$$

to be replaced by the two subproblems

$$(f(a_1, b_1),\ g(b_2, a_2)) \quad \text{and} \quad (c_1,\ c_2)$$

So we are led to a **problem refinement** view of analogy matching: the initial problem is to match the two terms or formulae which are given; as matching proceeds, subproblems, to match pairs of subterms of the original terms, are produced, and recursively refined. Subproblems are solved without producing more subproblems when one of the subterms in question is a constant or a variable, or when no pairings are suggested by the refinement procedure. Thus, instead of Kling's one-pass matching procedures, the whole matching process needs to be recursive, calling the same procedure on subterms which have been associated.

As matching proceeds, symbolic associations will be made (between the head function/predicate symbols (hfs's) of subterms which are associated); these associations will be added to an evolving **mapping**, which will in turn be used to

make new refinements. In the example above, the association[1]

$$(f, g, [(1,2),(2,1)])$$

would be added to the mapping when the argument pairings are made.

Notice that this approach to matching will only produce matches which satisfy the structural preservation constraint discussed above. This constraint thus represents an assumption behind the approach to matching which is developed in this section. The **partial homomorphism heuristic** is therefore built into the structure of the algorithm. However, as noted above, the restriction still allows considerable scope for structural deformations, which encompassess the types of structural mismatch which have been suggested by example.

Two more observations need to be made before the structure of the matcher can be described.

Firstly, it is important to realise that, in contrast to unification algorithms, it cannot be predicted, in advance, on which of the subproblems progress can next be made. Thus a fixed flow of control (for example top-down left-to-right), in which each matching subproblem is either successfully refined or the whole match fails, is not suitable for a heuristic analogy matcher[2]. This suggests that the matching subproblems should be organised as an **agenda**; if no refinement is suggested for a subproblem, it should be moved to the end of the agenda, and the next subproblem analysed. Matching should only terminate when no progress can be made on any of the subproblems on the agenda. When a subproblem is successfully refined, the sub-subproblems which are produced are added to

---

[1]Associations which have been written previously as $f \leftrightarrow g$ are in this chapter written (f,g,P), in order to show the argument pairings explicitly as the list P. P contains elements of the form (i,j) meaning that the ith argument of f is paired with the jth argument of g.

[2]Kling's matcher was criticised in the previous chapter for its inflexible co-ordination of the procedure MULTIMATCH1 within MULTIMATCH.

the front of the agenda. Since the existing mapping is used by the refinement procedure, we may (and often will) find that a matching subproblem which had been analysed without success previously, will be returned to with an extended mapping and will be refined successfully. This is the benefit of the agenda-based refinement.

The agenda of matching subproblems in a partial match are conjunctive subproblems — that is, they represent parts of a single match, and we hope to be able to solve them all together with a common mapping. We call this kind of agenda an *and*-agenda, to contrast it with the alternative type (*or*-agenda) which will be introduced below.

The second observation is that the heuristic analysis of a subproblem from the agenda may suggest more than one way of refining the subproblem, i.e. more than one set of subterm pairings. Each such set can then be investigated separately. Therefore, whenever this happens, the entire agenda should be **split** into a set of new agendas, each arising from one set of pairings. For example, on the following problem

$$f(x, f(y, z)) = f(f(x, y), z)$$

$$g(x, g(y, z)) = g(g(x, y), z)$$

the matcher might decide to make the two sets of argument pairings [(1,1),(2,2)] and [(1,2),(2,1)]; the partial match would be split into two, containing the new sets of subproblems

$$\{\,(\,f(x, f(y, z)),\ g(x, g(y, z))\,),\ (\,f(f(x, y), z),\ g(g(x, y), z)\,)\,\}$$

and

$$\{\,(\,f(x, f(y, z)),\ g(g(x, y), z)\,),\ (\,f(f(x, y), z),\ g(x, g(y, z))\,)\}$$

respectively. Thus, the current state of matching between a pair of terms involves, in general, a *set* of partial matches, each represented as an and-agenda together with an evolving mapping. In the current example, the mappings associated with the offspring partial matches would be produced by adding the

associations

$$(f, g, [(1,1),(2,2)])$$

and

$$(f, g, [(1,2),(2,1)])$$

respectively to the previous mapping.

As we shall see, each partial match can be assessed as to its promise, via its mapping. Since we are looking for strong matches, we develop the most promising of the partial matches first. After the strongest has been developed a certain amount, we re-assess its mapping, compare it with the rest, and develop the (possibly) new strongest partial match. Thus the set of partial matches between terms is handled as an agenda as well; this agenda is an *or*-agenda, since we only need a single match between the terms in order to progress to the later stages of the analogy process; the partial matches on the top-level or-agenda do not share any information.

Thus, the overall structure of the matcher will be an or-agenda, which consists of partial matches between the terms, each of which itself consists of an and-agenda of pairs of subterms still left to be matched coupled with an evolving mapping

While this structure may seem rather complex at first sight, it has been derived from some simple and natural properties that an analogy matcher should have. We emphasise this basic simplicity as an attractive feature of the approach to matching which we are taking. In the next two sections we add detail to the structure: in section 4.2.3, we describe the procedures involved in refining a partial match; in section 4.2.4, we describe how matching is coordinated *between* partial matches — i.e. how the top-level or-agenda is developed.

4.2.3 The development of a partial match

In this section we describe how a given partial match is developed — that is, how one or more of the matching subproblems which make up the partial match

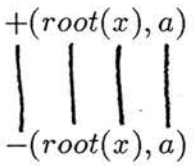
is refined, and how offspring partial matches are constructed from a refinement. Specifically, we describe

- The heuristic criteria which are used to propose argument or subterm pairings,
- How the application of these criteria is co-ordinated over a partial match, and how their suggestions are evaluated to decide on which pairings to make,
- How offspring partial matches are constructed once the pairings are decided on.

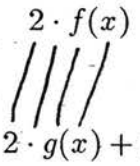
It is an important feature of the new matcher that the heuristic criteria are separate from the procedures that make use of them. This separation facilitates modification of the matcher, by introducing new criteria (possibly based on new heuristics), adjusting the relative importance of existing criteria, or altering the strategy for coordinating the criteria.

Pairing criteria

In Kling's matching algorithm, all the pairing criteria used by SINGLEMATCH, MULTIMATCH and MULTIMATCH1 are *argument* pairing criteria; that is, given an association of compound terms, it is assumed that the immediate subterms of the terms will match, as in the following example:



While this is true for most simple analogies, matches such as the following are also quite common:



Here, the two terms have been associated, but instead of the arguments being paired in some way, the whole of one of the terms is matched with a subterm of the other. We would like a flexible matcher to be able to find such matches. This is why the pairing criteria are not solely *argument* pairing criteria, but may be used to indicate other sorts of subterm pairings.

The pairing criteria perform tests on the two terms involved in a matching subproblem; the results of the tests can be thought of as *suggestions* for pairings of subterms, usually argument pairings; however, the job of assessing the information and deciding on which pairings to make, if any, is left to the control procedures — depending on the results of its test, a particular criterion may be used to suggest pairings of arguments, or the pairing of the whole of one term with a subterm of the other.

We list the criteria which are used in the current matcher below, together with examples of their use. We then discuss each, and say which analogy heuristic it is based on. All but one of the criteria are completely specified; one however, criterion 3, is only partially specified, and will be described in detail in the remarks which follow.

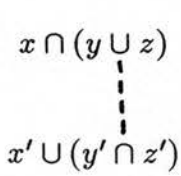
Pairing Criteria

Given a matching subproblem, consisting of a pair of terms to be matched, and the existing mapping:

1. Suggest pairs of arguments to the terms which have head function symbols (h.f.s.'s) which are associated in the existing mapping.

   For example, if the existing mapping contained the association $(\cup, \cap, P)$, this criterion would suggest the pairing (2,2) on the following subproblem[3]

[3]We are using the convention that the upper formula in a diagram corresponds to the left element in associations, and the lower corresponds to the right. Pairing suggestions are illustrated with dashed lines, in this section, to distinguish them from actual pairings.



2. If the hfs's of the terms are associated in the existing mapping, suggest the previous argument pairings. If either of the hfs's in question is commutative, return all possible argument pairings.

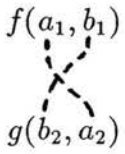
   For example, suppose the existing mapping contains the association

   $$(f, g, [(2,1),(1,2)])$$

   Then the pairings

   $$\{(1,2),(2,1)\}$$

   would be suggested on the following subproblem



3. Suggest pairs of arguments to the terms in the match which have similar syntactic structure, as long as almost all of the arguments can be paired in this way.

   This criterion is further specified and discussed below, where an example is given.

4. Suggest pairs of non-head symbols in the arguments of the terms which are associated by the existing mapping.

If $(x, x, term) \in \alpha$[4], the pairing ([2,1],[1,2,1]) would be suggested on the following subproblem

$$2 \cdot f(x)$$
$$\vdots$$
$$2 \cdot g(x) + 1$$

Note that the complete subterm positions at which the symbols occur are retained in the pairing suggestion; this is so that the criterion can be used to suggest pairings other than argument pairings.

5. Suggest pairs of arguments of the terms which have the same hfs.

   The suggestions {(1,1),(2,2),(2,3)} would be made by this criterion on the subproblem

$$\wedge(rational(x), \neg rational(y))$$
$$\wedge(rational(x), \neg rational(y), \neg x = 0)$$

6. If the terms have the same hfs (and the same arities), which is not known to be commutative, suggest argument pairings which respect argument order (i.e. the pairings $\{(1,1),(2,2)\ldots(n,n)\}$); if they have the same hfs which is known to be commutative, suggest all argument pairings.

   On the (schematic) subproblem

$$\begin{array}{ccc} A & \longrightarrow & B \\ \vdots & & \vdots \\ A' & \longrightarrow & B' \end{array}$$

---

[4]The current mapping is referred to as α throughout this chapter. The expression *term* in the argument pairing slot of an association indicates that the symbols have no arguments paired — i.e. the association is terminal in the match. This will usually be because at least one of the symbols in question is atomic, and thus has no arguments.

this criterion would suggest the pairings {(1,1),(2,2)}.

Remarks on the pairing criteria:

1. This criterion is based on the **consistent translation heuristic**; argument pairings are suggested which, if made, would maintain the consistency of the existing mapping. If pairings suggested by this criterion are overruled, the mapping is very likely to become inconsistent. Thus, we regard this as a strong criterion. Kling uses similar criteria in SINGLEMATCH and MULTIMATCH.

2. Again this is based on the **consistent translation** heuristic; an association of predicates or functions which is repeated in a match with *different* argument pairings is regarded as a (mild) form of inconsistency. If either of the symbols in question is commutative, the particular pairings used before are not relevant, so all possible pairings are suggested; it would be possible to make no suggestions in this case, and this was done with an earlier version of the criterion, but this had the effect of discriminating against commutative functions; so it was replaced with the present version.

3. This criterion is based on the **partial homomorphism** heuristic: pairings are suggested which would allow a close structural match, not paying attention to other factors such as consistency. This criterion was added to the matcher after the others had been implemented. Originally, the partial homomorphism heuristic was only used in the overall structure of the matcher, as explained above. The performance of the original matcher was found to be poor on simple examples such as the following one

$$\begin{array}{c} f(f(a_1,b_1),c_1) = f(a_1,f(b_1,c_1)) \\ \| \; \| \; \| \; \| \; \| \; \| \; \| \; \| \; \| \; \| \; \| \\ g(g(a_2,b_2),c_2) = g(a_2,g(b_2,c_2)) \end{array}$$

where there is a close structural correspondence and consistent mapping, but few initial clues, such as the identity of symbols, to get the match

started. Thus an explicit criterion, based on the partial homomorphism heuristic, was suggested, which could be used to propose new pairings.

So far, we have only specified the *intention* of the criterion, to look for structural similarity between arguments of the terms. The following other considerations motivated the design of the syntactic structure criterion:

- If arguments can be unambiguously paired by analysing their syntactic structure to a certain level, there is no point in analysing them any further, at that stage. For example, in matching $x \cap 1 = x$ and $x \cup 0 = x$, analysis at level 1 (ie just arities) indicates the pairings (1,1) and (2,2), so there is no need for the criterion to go further.
- Unless almost all of the arguments of the terms can be paired off on the basis of syntactic structure, no pairing suggestions should be made at all. This is because, even between terms with little overall syntactic similarity, there are likely (by coincidence) to be some pairs of arguments with the same structure; this should not be regarded as evidence for their pairing.

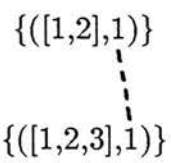
The criterion which has been implemented is given below, and is illustrated on the subproblem

$$\wedge(rational(x), \neg rational(y))$$

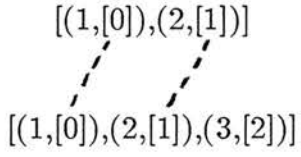
$$\wedge(rational(x), \neg rational(y), \neg x = 0)$$

Syntactic structure criterion Sort the arguments of each of the terms into blocks of equal arity; then pair up the arity blocks on one side with those of the same arity on the other; on the example we would get[5]

[5]i.e. both the arguments of the first term have arity 1 as do all those of the second term.



For each pairing of blocks, if each block contains just one argument mark these as paired (this does not apply in the present case); if the blocks contain more than one argument, attempt to pair up the arguments within the blocks on the basis of the arities of *their* arguments; with each argument in a block associate the bag of the arities of its arguments, and mark as paired each pair of arguments within the given blocks with the same bag of arities. In this case, we get



If, by the above procedure, all of the arguments to the terms can be paired up (perhaps ambiguously), except for at most one argument on one side, then return the pairings as suggestions. If not, return no suggestions. In the present case, there is just one unpaired argument, so the pairings {(1,1),(2,2)} are returned.

This criterion only looks to depth two; if there are sets of arguments which have the same structure at this level, but which could be differentiated at a deeper level, the pairings are still made.

The above is just one way of implementing a syntactic structure criterion, and is not meant to be taken as the definitive version. Experience with the matcher without it has indicated the need for *some* syntactic structure criterion.

4. This criterion is based on the **consistent translation** heuristic, and is a weaker form of criterion 1. It is weaker because the symbols which are associated by the existing analogy may not be in corresponding positions in the arguments, and thus we are not so sure that the argument pairings

which are suggested would lead to preservation of the consistency of the mapping. This criterion is designed to be a cheap form of look-ahead. Kling uses a similar pairing criterion as the basis of MULTIMATCH1. This criterion may be used to suggest structural mismatches, such as that shown above, as well as argument pairings.

5. This criterion is based on the **identical symbols heuristic**; we propose argument pairings which will lead to identical associations being added to the mapping. Just as the identical symbols heuristic was seen to be a weak heuristic, this criterion is a weak criterion, compared to those discussed above.

6. This is again based on the **identical symbols** heuristic, and thus is also a weak criterion; "identical association" in the context of an analogy mapping means having identical argument pairings as well. If the hfs is known to be commutative, there should obviously be no inclination towards identical pairings, and so all pairings are suggested. Kling uses a similar criterion, as the basis of ATOMMATCH, but for *all* pairs of predicates, not just identical pairs.

General remarks on criteria. Note that there is no criterion based on the semantic type heuristic; as discussed in the previous chapter, there are problems of circularity associated with the use of this heuristic. While the heuristic may be used to give the matcher clues to analogies, it is important for the matcher to be able to work without any type information. It would be easy to add a criterion, similar to criterion 5, which would find arguments whose hfs's have the same (or similar) type. The use of the semantic type heuristic, being weak, could also be restricted to the assessment stage (b) of matching, to choose among the matches produced by stage (a); this use for the heuristic is also suggested by the fact that the types would, in general, be organised in a **hierarchy**, and the semantic judgement between a pair of symbols would be a degree of semantic closeness, rather than an all or nothing judgement. We discuss further how criteria based on the semantic type heuristic could be added to the matcher in Chapter 7.

The coordination of the criteria in making pairings

We now describe how the pairing criteria are co-ordinated across a partial match to make a refinement. Recall that a partial match is an and-agenda of matching subproblems, together with a mapping. We need to decide

1. Which subproblem to apply the criteria to.

2. Given pairing suggestions, how to derive the pairings which are actually made.

3. Given the pairings which are made, how to update the partial match to produce offspring matches (including perhaps some complete matches).

At any stage, the answer to 1 is simple — choose the subproblem at the top of the agenda; the order of the agenda is a consequence of the procedures described below. The following procedure would be an obvious way of tackling 2:

> Apply all of the criteria to the subproblem. Put all of the evidence together to get a measure of strength (weighted according to the strength of the criteria) for each suggested argument pairing. Find consistent (1-1) subsets of the pairings which are relatively strong, the strength of a subset being the sum of the strengths of its elements. Return these strong subsets as the plausible argument pairings.

A control strategy such as this would have the following deficiencies:

- Some of the criteria are considerably stronger than others; if strong criteria suggest argument pairings unambiguously, there would be no point in applying the weaker criteria, since, in this case, they would not affect the overall judgement about pairings; doing so would be wasteful, particularly considering that one of the weaker criteria, criterion 4, will involve considerable computation.

- If only weak criteria suggest pairings, or if the strong criteria are ambiguous, it may be wise to suspend the current subproblem, making no pairings and moving it to the back of the and-agenda; stronger criteria may be active and unambiguous at another subproblem. The pairings suggested at the other subproblem would be more reliable, and, once made, the associations added to the mapping might lead to clearer pairing information at the original subproblem. The simple control strategy does not allow this flexibility.

- It would be sensible to delay splitting a partial match as long as possible; i.e. to see if a single set of pairings is suggested at any subproblem, before splitting the match. This would avoid the proliferation of weak partial matches in the top level or-agenda. Again, the simple control strategy would not be able to do this.

These factors suggest a control regime for match development under which stronger criteria are considered first throughout the agenda before the weaker criteria are considered anywhere. Furthermore, the weak criteria would be considered at all subproblems before any splitting of the partial match was done. Thus we are led to to a three-level organisation of criteria:

Level	Criteria
1	1,2,3
2	(1,2,3),4,5,6
3	(1, .., 6), splitting

Thus we deem criteria 1,2 and 3 to be stronger than the others; this judgement is based on the strength (decided in Chapter 3) of the heuristics on which the criteria are based; the only exception is criterion 4 which, although based on a strong heuristic, the consistent translation heuristic, is put in at level 2; this is because, as explained above, the criterion is based more loosely on the heuristic than is so with the other criteria.

To develop a match, we first attempt to make a refinement at level 1; if successful, the and-agenda and mapping are updated according to the new subterm

pairings; if unsuccessful (i.e. no clear pairings can be made at any matching subproblem at level 1), we attempt a refinement at level 2; similarly, if this is successful, the agenda and mapping are updated; if not, an attempt is made to refine at level 3.

Thus, there are two possible results to developing a partial match: (1) a refinement is made at some level, and new matches are produced as a result; some of these may be complete, having empty and-agendas, and some still partial; each new partial match would then be available for further development (2) no refinement is made, in which case matching terminates; any subproblems remaining on the agenda are transferred to the mapping — i.e. we regard the entire subterms as being associated as wholes.

Pairing evaluation We now describe how the pairing suggestions made, at a level, are analysed to find clearly suggested pairings.

Level 1. First, some weakly suggested pairings are removed from the list obtained from criteria 1 2 and 3 — those that share an argument position with another pairing which is suggested by a larger set[6] of criteria. Those pairings which remain and are now **unambiguous** (i.e. do not share arguments with others that remain), are returned as the clearly suggested pairings.

The rationale behind this procedure is as follows: the criteria which are being used often suggest many different argument pairings; those which are less well suggested than others with which they compete are removed to cut down the level of 'crosstalk'. After this has been done, any which remain and do not compete with any other which remain, may be taken to be reliable (clearly suggested) pairings, and should thus be made. Note that the pairings made here may not be complete, i.e. there may be arguments left in the terms which have not been paired.

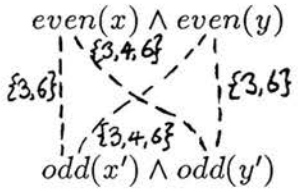
---

[6]By larger set, we mean strict superset.

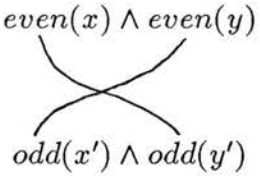
If terms in the subproblem each have all but one of the arguments paired, the remaining two are also paired. This is a default rule, similar to rules used by both Kling and Munyer.

If there are no clearly suggested pairings, the subproblem is put at the back of the agenda, the next is chosen, and the procedure is repeated. This continues until either some clear pairings are found (a refinement is made), or all subproblems have been analysed at level 1 (no refinement is made).

Level 2. The level 2 procedure is an extension of that at level 1; pairing suggestions from criteria 4, 5 and 6 are added to any remaining from level 1 (multiple suggestions from criterion 4 are counted multiply). The main procedure is exactly that for level 1. To illustrate, suppose we had the following subproblem, with pairings labelled with the sets of criteria suggesting them:



The first stage removes the pairings (1,1) and (2,2), leaving



The two remaining pairings are both unambiguous and so would be returned as the clearly suggested pairings.

In addition to the level 1 procedure just illustrated, suggestions made by criterion 4 may be used to derive a **structural mismatch** pairing, i.e. a pairing of one of the terms from the subproblem with a subterm of the other. A pair of symbol positions returned by criterion 4 may be thought of as suggesting a structural mismatch if the two symbols are at different depths in the terms; if the positions are P_1 and P_2, $|P_1| < |P_2|$, and P is the initial segment of P_1 of length

$|P_2| - |P_1|$, the pair of associated symbols suggest that the whole of the first term be paired with the subterm of the second at position P. This is illustrated by the following example:

$$2 \cdot f(x)$$
$$\vdots$$
$$2 \cdot g(x) + 1$$

If the variable x is associated with itself in the current mapping, criterion 4 will return the pair of x's at positions P_1=[2,1] and P_2=[1,2,1] respectively. Then P=[1], so the pairing of the whole of the first term, $2 \cdot f(x)$, with the subterm of the second at position [1], $2 \cdot g(x)$, is indicated. The rule for suggesting structural mismatches takes into account *all* the symbol pairs returned by criterion 4: a mismatch is only suggested if all the pairs give the same position for P. In the example above, x is the only symbol present which is in the existing mapping, so the rule would fire, and suggest the structural mismatch pairing. The rule is perhaps rather strict, as a potential mismatch may be ruled out by 'crosstalk' between more than one occurrence of a matched symbol. A more flexible rule could be devised if this problem turned up in practice.

The structural mismatch rule is an 'optional extra' in the matcher, which gives it the ability to construct a wider range of matches.

Again, a refinement is initiated when new pairings are suggested by the evaluation routine at some subproblem. Otherwise, the match passes to level 3.

Level 3. Level 3 involves no further calculation of criteria; the first subproblem (if any) which contains pairing information from levels 1 and 2 is taken from the agenda; the suggested argument pairings are split up into maximally consistent subsets[7]. Only those subsets having maximal size are retained, each of these producing a set of argument pairings. If there is no pairing information at any subproblem in the agenda, no refinement is made.

---

[7]By consistent, we mean that each argument is paired with at most one argument from the other term.

Updating the agenda and the mapping, given pairings

When the procedures just described result in pairings being made at a subproblem in a match, the pairings made are used to update the and-agenda and mapping of the match. We describe these procedures now.

When more than one set of pairings is made, the updating procedures described below are called on each such set in turn, to produce a corresponding set of offspring matches.

The agenda. Given a set of pairings, the agenda is updated in the following two ways:

- New subproblems may be added to the front of the agenda, corresponding to the subterms which are paired. A new subproblem is added for each new pairing of *compound* subterms. If one or both of the subterms in a pair is atomic, the association between them is added directly to the mapping, as described below, and no new subproblem is put on the agenda; this is because there is no more matching to be done between these subterms.

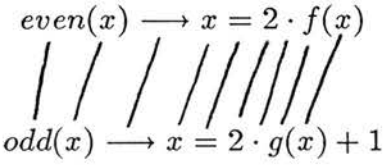
- The subproblem at which the pairings were made may be removed from the agenda. This happens if the pairings made are **complete** — that is, one of the terms in the subproblem has all its arguments paired, or if there is a structural mismatch. If the pairings are incomplete, it is retained, to be analysed again at a later time, in the hope of being able to complete the pairings; in this case, the pairing suggestions made for the subproblem are retained for analysis at lower levels.

The mapping. If the subproblem giving rise to the pairings is having its first argument pairings made, the association between the hfs's of the terms in the subproblem is added to the mapping together with the argument pairings which are being made. We could add the association as soon as the subproblem is put on the agenda, rather than waiting for pairings to be made; we do not do this for two reasons: firstly, the subproblem may lead to a structural mismatch, which does not involve an association between the hfs's of the terms; secondly,

the terms of the subproblem might not match syntactically at all, and would be shifted directly to the mapping when matching terminates (i.e. they would be considered part of the difference between the original terms in the analogy).

If the subproblem has already had some pairings made, and more are being added, the association between the hfs's will already be in the mapping, and is merely updated with the extra argument pairings.

In the case of a structural mismatch, the association between the entire terms in the subproblem is added to the mapping, with no argument pairings. The motivation for this is that, with structural mismatches such as



we want to consider $2 \cdot g(x) + 1$ as a potential analogue for $2 \cdot f(x)$; if we did not add the association between the whole terms, we would only be able to consider $2 \cdot g(x)$ as analogue for $2 \cdot f(x)$.

The mapping is also extended whenever subterms are paired at least one of which is atomic; the association is simply that between the subterms, with no argument pairings (i.e. with *term* in the pairing slot).

Completed matches There are two situations under which an offspring match is deemed to be complete:

- If the and-agenda is empty after it has been updated. This is the 'better' form of completion, as it means that the original formulae have been completely matched.

- If no pairings are made at the previous stage, i.e. no further matching is suggested between subterms which remain on the agenda. In this case, if we are to regard the original formulae as analogous, we must regard the remaining subterms as being associated as wholes, without further

decomposition of their association. The subproblems which remain on the agenda are therefore transferred to the mapping as associations.

4.2.4 Co-ordination of matching between different partial matches

In the previous section, we looked in detail at how a given partial match is refined, producing, in general, a set of new partial matches. We now describe the top-level of the matching algorithm — how the development of the various partial matches is co-ordinated. We have already explained (section 4.2.2) that the various partial matches being developed are organised in an or-agenda. The ordering of the agenda is based on an assessment of the promise of the matches — the assessment procedure is described below.

Assuming the assessment procedure, the development of the or-agenda of partial matches is straightforward: the top match (ie the most promising one) is taken from the agenda; the match refinement procedure is called to make a refinement to the match, producing a set of offspring matches. Each of the offspring matches is then developed as far as possible at level 1. The resulting matches, some complete and some still partial are assessed; the partial ones are **merged** with the or-agenda (i.e. inserted in order of strength); the complete ones are kept in a separate list; the set of matches (both complete and partial) is then pruned by removing all matches with strength less than a dynamic threshold — the threshold is calculated by subtracting a fixed number (0.125 in the current version) from the strength of the strongest complete match.

The above procedure is repeated until the or-agenda becomes empty.

One aspect of the procedure requires further explanation — the further development of offspring matches at level 1, after the initial refinement has been made. A single stage of development of a match may thus consist of more than one refinement. The reason for this is that a refinement at level 1 can be thought of as an inevitable consequence of the initial refinement, and involves no further splitting of the match; it is thus sensible to perform all such refinements before

re-assessing the match. This is not an important feature, but does limit the amount of time spent processing the top-level agenda and in match assessment.

Match assessment The assessment of a match is done using its associated mapping, and is based directly on the analogy heuristics. The mapping is assessed according to each heuristic separately, and the overall assessment is determined from the individual assessments. More specifically, the mapping is rated according to its degree of conformation with a heuristic on a scale of 0 to 1. The overall assessment is a weighted average of these numbers; the weights determine the relative significance given to the heuristics. The heuristics which are incorporated into the current assessment procedure are the following:

1. Partial homomorphism: as explained earlier, a degree of structural preservation is ensured by the design of the matcher, but there is still scope for significant variation within this — individual symbols or entire subterms may be unmatched. The proportion of symbols which have been matched to symbols which have been left unmatched is taken as a measure of the structural closeness of the match.

2. Consistent translation: we assess the degree of consistency of a mapping, by counting the number of inconsistencies as a proportion of the size of the mapping, and scaling the answer so that it lies between 0 and 1. If l is the length of the mapping, and n is the number of inconsistencies, the assessment is

$$\frac{1}{1+3\cdot(n/l)}.$$

Inconsistencies such as $\{a \leftrightarrow b, a \leftrightarrow c\}$, where a, b and c are atomic and distinct, in a mapping are easy to count. It is more difficult if compound terms are involved in the associations, which, as explained above, can happen. The general definition of an inconsistency which is used by the current assessment procedure is as follows:

The distinct associations

$$s_1 \leftrightarrow t \text{ and } s_2 \leftrightarrow v,$$

> where s_1, s_2, t and v are possibly compound terms, are regarded as inconsistent (from left to right) if t and v are distinct and s_1 and s_2 share at least one symbol. Inconsistency from right to left is defined analogously.

This definition of an inconsistency, which subsumes the normal one for atomic associations, is perhaps rather strict; we might find that some mappings are penalised more than they intuitively ought to be, although this problem has not yet arisen in practice with the current matcher.

When an association in the mapping involves unmatched arguments, these are taken into account when counting inconsistencies: if a symbol occurs in an unmatched term, and is also associated in the mapping, we regard this as a mild form of inconsistency; half an inconsistency is added for each such situation.

Before a mapping is checked for inconsistency, any associations in it which arose from subterms at which there was a structural mismatch are removed; for each such association, one inconsistency is counted. The reason for this exception to the procedure is illustrated by the example of a structural mismatch given above:

$$\begin{array}{ccc} even(x) & \longrightarrow & x = 2 \cdot f(x) \\ \Big| & & \Big| \\ odd(x) & \longrightarrow & x = 2 \cdot g(x) + 1 \end{array}$$

In the mapping, we would have the associations $2 \cdot f(x) \leftrightarrow 2 \cdot g(x) + 1$, $2 \leftrightarrow 2$, $x \leftrightarrow x$, $f \leftrightarrow g$ and $\cdot \leftrightarrow \cdot$; the first of these is the mismatch association which would be removed before counting inconsistencies, and would contribute one to the number of inconsistencies. This is correct intuitively, since there is an inconsistency implied by this mapping, namely (from left to right) as to what is the analogue of the term $2 \cdot f(x)$ — it could be either $2 \cdot g(x)$ or $2 \cdot g(x) + 1$ — but only one. If the mismatch association were not removed, it would lead to four inconsistencies in this case, which would be excessive. The general problem with mismatch associations is that they

tend to be inconsistent (in the sense defined above) with their own daughter associations.

3. Identical symbols: the proportion of the associations in the mapping which link identical symbols is computed as the extent to which the mapping conforms to the identical symbols heuristic.

The overall assessment of a mapping is given by

$$\alpha \cdot PH + \beta \cdot CT + \gamma \cdot IS,$$

where PH, CT and IS are the contributions from the partial homomorphism, consistent translation and identical symbols heuristics respectively, and

$$\alpha + \beta + \gamma = 1.$$

In the current version, $\alpha = 0.5$, $\beta = 0.4$ and $\gamma = 0.1$. These numbers reflect the relative significance attached to the corresponding heuristics.

Note that the semantic type heuristic is not used at the moment. This is firstly because the **multiple matches problem**, referred to in the previous chapter, has not arisen with the modest size of knowledge base which has been used with the current matcher, and secondly because the matcher should be able to work without the guidance of semantic types.

Examples of assessments of matches are given in section 4.4.

Comparison with Munyer's assessment

Munyer's direct assessment of completed matches is, like his matching algorithm, based on the partial homomorphism and identical symbol heuristics. Oddly, Munyer's procedure makes no penalty for unmatched symbols, and so would not rate the structural mismatches we have been considering any less highly than a structural isomorphism.

Another unsatisfactory aspect of Munyer's procedure is that he penalises associations which permute the argument positions of their parent associations,

even if the parent functions are distinct. As explained above with reference to Kling's procedure ATOMMATCH, there is no rational basis for this.

4.3 Summary of new matcher

We have now completed the description of the matcher. Since we have chosen to introduce the features gradually, with explanation and examples, the description has been rather spread out. Therefore, we give here a concise summary of the structure of the matcher.

Data structures

State of matching: a pair (C,O), where O is an or-agenda, whose nodes are partial matches, ordered by their assessments, and C is a list of completed matches, again ordered by their assessments.

Partial match: a quadruple (A,M,L,S), where A is an and-agenda of matching subproblems (ordered by (1) recency of addition and (2) success/failure of refinement); M is a mapping; L is a level (1, 2 or 3); S is the assessment of the partial match.

Completed match: a pair (M,S), where M and S are as above.

Matching subproblem: a quadruple (T1,T2,P,I), where T1 and T2 are the subterms to be matched, P records argument pairings of T1 and T2 which have already been made and I records any pairing information which has been passed down from analysis of the node at higher levels.

Mapping: a list of pairs (F1,F2,P), where F1 and F2 are symbols from the terms, and P gives pairings of the arguments to F1 and F2.

Initialisation

Initially, state of matching=(ϕ,{({(T1,T2,ϕ,ϕ)},ϕ,1,u)}), where T1 and T2 are the entire terms to be matched; i.e. there is one partial match, which consists of a single subproblem (the entire problem), empty mapping and no assessment;

the initial level is 1.

Algorithm

Repeat main cycle until state of matching=(C,ϕ); return the completed matches C as the result of the procedure.

Main cycle *(given state of matching, (C,O), returns new state of matching (C',O'))*: remove top partial match, (A,M,L,S) from O; call the development routine, producing sets of daughter matches (C'', N''), C'' complete and N'' partial; assess these (as described above), and merge them with (C, O), adding any completed daughters to C and the rest to O; compute the strongest completed match, and remove all matches (complete or partial) having strength more than a preset threshold below the strongest; return new state of matching (C', O').

Development routine *(given partial match, (A,M,L,S), returns new sets of matches (C,N), where C are complete and N are partial)*: attempt to make a refinement to the partial match, calling the refinement procedure at decreasing levels starting at L, until either a refinement is made at some level or the levels are exhausted; if a refinement is made, producing sets of new matches (C', N'), C' complete and N' partial, call the refinement procedure at level 1 repeatedly on each partial match from N', until either the match becomes complete or no more progress can be made at level 1; return the resulting sets of complete and partial matches as (C, N); if no refinement is made, move the nodes from A to the mapping M and return $(\{(M', S)\}, \phi)$ as (C, N), where M' is the extended mapping; i.e. mark the match as complete.

Refinement procedure *(given partial match (A,M,L,S), returns new sets of matches (C,N), where C are complete and N are partial)*: attempt to make a refinement to (A, M, L, S) by calling the subproblem analysis procedure repeatedly on subproblems from A (from the top), until either new pairings are made at some subproblem, or all subproblems from A have been tried; if new pairings are made at some subproblem, call the updating procedures (as described above) on each set of new pairings, producing new sets of complete and par-

tial matches (C, N); if no new pairings are made at any subproblem, return $(\phi, \{(A', M, L, S)\})$, where A' is the agenda updated by any pairing information at level L.

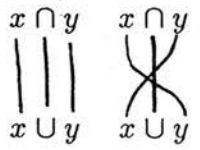
Subproblem analysis procedure *(given matching subproblem $(T1, T2, P, I)$, mapping M and level L, returns a set of sets of new argument pairings S)*: call the pairing criteria associated with L on those arguments of $T1$ and $T2$ not already paired according to P; add pairing information obtained to that in I and pass to the pairing evaluation procedure associated with L (as described above); if new pairings are suggested, return these as S; if not, return ϕ.

4.3.1 Additional features

- If we want to match up a pair of terms and we already know some associations which we want to be in the mapping, we can start the matching off with a non-empty mapping slot, containing the associations which we want to appear in the match. This facility allows guidance to be given to the matcher, and will prove useful in later stages of the analogy process (see Chapter 5).

- If the propositional structures of the terms being matched are not the same, they are both rewritten into conjunctive normal form (i.e. each term is transformed into an equivalent one which is a conjunction of disjunctions of literals), and matching proceeds on the normal forms. The idea behind this is that a potential analogy may be obscured by superficial differences in the propositional presentation of formulae; if both formulae are normal-formed, such differences will be removed. Other kinds of superficial differences between formulae are considered in section 4.7.

- When the matcher produces multiple matches between a pair of terms, some of them may be trivial variants of each other: for example, the following two matches



differ only in their association of variables and the argument pairings to the association $\cap \leftrightarrow \cup$. But the variable associations are equivalent up to renaming of variables (i.e. are α-variants of each other), and the functions $\cap$ and $\cup$ are known to be commutative, and so the argument pairings are irrelevant. Thus, only one of the matches needs to be retained.

Generally, two matches are considered variants of each other if (1) their variable associations are α-equivalent and (2) their non-variable associations are identical modulo commutativity. After two terms are matched, the set of matches which has been produced is analysed, and any variants are removed.

- The top-level or-agenda allows partial matches between a problem and a number of other problems to be considered alongside each other. The initial agenda contains a node for each of the alternative base problems. As matching proceeds, the promising matches with the promising base problems will tend to rise to the top of the agenda; the rest will fall to the bottom, and may well be cut off by the threshold. Thus it will not be necessary to match each candidate base problem to completion in order to find the best.

4.4 Performance

We illustrate the performance of the new matcher on some examples:

Example 1

Formulae to be matched[8]:

$$rational(x) \ \wedge \ \neg rational(y) \longrightarrow \neg rational(x + y)$$

$$rational(x) \ \wedge \ \neg rational(y) \ \wedge \ \neg x = 0 \longrightarrow \neg rational(x \cdot y)$$

The matcher starts with a single node, containing the whole formulae. The level 1 criteria are first checked; none of these makes any suggestions, so the match goes to level 2. Of the level 2 criteria, both 5 and 6 suggest the argument pairings (1,1) and (2,2). These suggestions are undisputed, so the argument pairings are made; two new nodes are added, containing the hypotheses and the conclusions respectively, and the original node is deleted, having had complete pairings made; the association ($\longrightarrow$,$\longrightarrow$,[(1,1),(2,2)]) is added to the mapping, and the match returns to level 1.

The hypotheses node is analysed first at level 1; criterion 3 suggests the pairings [(1,1),(2,2)], which are undisputed and therefore made; two new nodes are added to the agenda, and the hypotheses node is deleted, as one side has

---

[8]The hypothesis of the second formula is written with $\wedge$ as an infix operator for the sake or readability; the term presented to the matcher is actually

$$\wedge(rational(x), \neg rational(y), \neg x = 0)$$

This convention will be repeated elsewhere.

had all its arguments paired; the association (∧,∧,[(1,1),(2,2),(u,p(3,¬$x = 0$))]) is added to the mapping[9], and the match stays at level 1.

The new node $(rational(x), rational(x))$ is analysed next: criterion 3 suggests the undisputed pairing [(1,1)], which is made; the node is deleted, and associations (*rational*,*rational*,[(1,1)]) and $(x, x, term)$ are added to the mapping, the latter arising automatically from an atomic node. In a similar way, the hypotheses $\neg rational(y)$ and $\neg rational(y)$ are paired up using criteria 1, 2 and 3, and adding the associations (¬,¬,[(1,1)]) and (y,y,term) to the mapping.

On the conclusions, still at level 1, refinement proceeds via criteria 1 and 3. On the node $(x+y, x*y)$, criterion 1 suggests [(1,1),(2,2)], on the basis of the existing variable associations, and criterion 3 suggests [(1,1),(1,2),(2,1),(2,2)]: the more weakly suggested pairings, (1,2) and (2,1) are removed, and the remaining ones are made, which completes the matching.

Thus, the following single match is found:

$$rational(x) \wedge \neg rational(y) \longrightarrow \neg rational(x+y)$$

$$rational(x) \wedge \neg rational(y) \wedge \neg x = 0 \longrightarrow \neg rational(x \cdot y)$$

Assessment: 0.850

Example 2

Formulae to be matched:

$$\sqrt{x} + a = b$$

$$\sqrt{x+1} + a = b$$

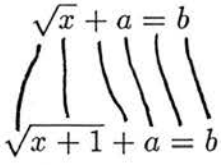
---

[9]When the argument pairing list contains a pair of the form (u,p(I,T)), this means that the Ith argument of the right symbol, T, was unmatched; similarly, the notation (p(I,T),u) refers to the left symbol having an unmatched argument.

This example was used, in the last chapter, to illustrate the inflexibility of Kling's matcher.

Firstly, at level 1, the initial node has pairings [(1,1),(2,2)] made by criterion 3, adding the association (=,=,[(1,1),(2,2)]). The (2,2) pairing yields an atomic association (b,b,term), which is added to the mapping. The node arising from the (1,1) pairing has pairings [(1,1),(2,2)] made by criterion 3 again, adding the association (+,+,[(1,1),(2,2)]) to the mapping. The (2,2) pairing yields another atomic association (a,a,term). The other new node, $(root(x), root(x+1))$, leads to no suggestions at level 1. Therefore, since it is the only node left, the match goes to level 2; the same node is analysed by the level 2 criteria, of which criterion 6 suggests the pairing (1,1), which is made, yielding an atomic association (x,x+1,term), which completes the matching.

Thus, the following single match is produced:



Assessment: 0.795

On the two examples considered so far, most of the matching takes place at level 1, with criterion 3 doing most of the work. This is because there is a close structural correspondence in each case. However, with criterion 3 removed from the matcher, the same matches would have been produced. In this case, most of the matching would be done at level 2, with applications of criteria 5 and 6 replacing those of criterion 3.

In the next two examples, the structural correspondence is not so close.

Example 3

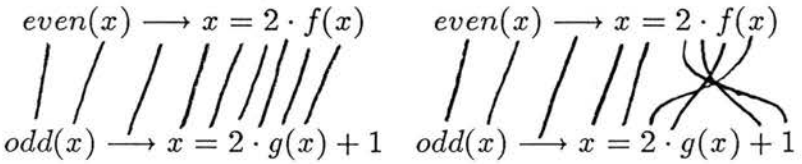
The formulae to be matched are:

$$even(x) \longrightarrow x = 2 \cdot f(x)$$

$$odd(x) \longrightarrow x = 2 \cdot g(x) + 1$$

On the initial node, criterion 3 suggests the pairings [(1,1),(2,2)], which are made. On the node arising from the hypotheses, the same criterion suggests the pairing (1,1), which yields an atomic association (x,x,term). On the node arising from the conclusions, criterion 1 suggests (1,1) and criterion 3 suggests [(1,1),(2,2)]. The pairings [(1,1),(2,2)] are therefore made, yielding an atomic association and a new node $(2 \cdot f(x), 2 \cdot g(x) + 1)$.

The new node yields no suggestions at level 1, so the match goes to level 2. Here, criterion 4 suggests the argument pairing (2,1); on evaluation of the pairing suggestions, the structural mismatch rule fires, as explained in section 4.2.3. Thus the match is split into two: the first new match results from the argument pairings [(1,2),(2,1)] (the (1,2) pairing is added by the default rule); the second new match results from the structural mismatch. These two matches are developed separately, and lead to the two matches shown below:



0.815 0.796

The assessments which are produced for the matches indicate that the first match is the preferred one[10].

[10]Although the assessments seem very close together, the difference between the first and the second is quite significant; experience with the matcher has shown that, as a consequence of the way assessments are calculated, reasonable matches start at around 0.6.

Example 4

HYPOTHESES	HYPOTHESES
group(g,*)	ring(r,**,++)
propernormal(m,g,*)	properideal(n,r,**,++)
factorstructure(x,g,m)	factorstructure(y,r,n)
simplegroup(x,*)	simplering(y,**,++)
CONCLUSION	CONCLUSION
maximalgroup(m,g,*)	maximalring(n,r,**,++)

This is one of Kling's algebra examples. As explained in the previous chapter, Kling's matcher leans heavily on the semantic templates to match up these formulae.

The matching with the new matcher proceeds as follows: no pairings are suggested for the initial node at level 1, so the matcher goes to level 2. Criterion 5 suggests the pairing (1,1), and criterion 6 the pairings [(1,1),(2,2)]. Thus, these pairings are made, and the matcher returns to level 1.

No progress is made on either of the two new nodes at level 1 (on the conclusion node, criterion 3 suggests all possible pairings, but there are no undisputed ones), so the matcher goes back to level 2. On the hypotheses, criterion 5 suggests the pairing (3,3), and criterion 6 suggests all possible pairings, since conjunction is commutative (symmetric); the assessment procedure removes the more weakly suggested pairings that compete with (3,3); of the remainder, (3,3) is the only undisputed pairing and is therefore made. The hypothesis node is retained on the agenda, since it has not had complete pairings made. The match goes to level 1, but, again, no progress is made there, so it returns to level 2. Criterion 6 suggests the pairings [(1,1),(2,2),(3,3)] at the node $(factorstructure(x,g,m), factorstructure(y,r,n))$; these suggestions are assessed along with those remaining at the node from criterion 3, which had suggested all possible pairings; the pairings [(1,1),(2,2),(3,3)], being the more strongly suggested, are made, and lead to atomic associations, (x,y), (g,r) and (m,n), which are added to the mapping. The match returns to level 1, where the

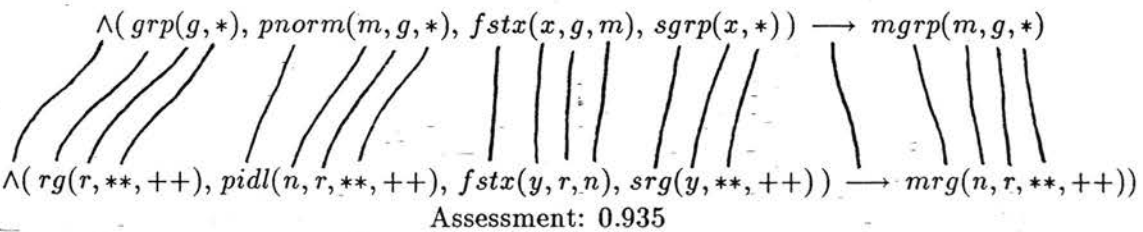
conclusion node $(maximalgroup(m, g, *), maximalring(n, r, **, ++))$ has pairings [(1,1),(2,2)] made by criterion 1 (criterion 3 again suggests all possible pairings). No further progress is made at level 1.

At level 2, the hypothesis node is analysed again: criterion 6 again suggests all possible pairings. Criterion 4 suggests the pairings [(1,1),(1,2),(2,1),(2,2),(2,2),(4,4)], on the basis of the existing variable associations (note that (2,2) is suggested twice by criterion 4). After weakly suggested pairings have been removed, the pairings [(1,1),(2,2),(3,3)] remain, and, being mutually consistent, are made (the structural mismatch rule does not fire).

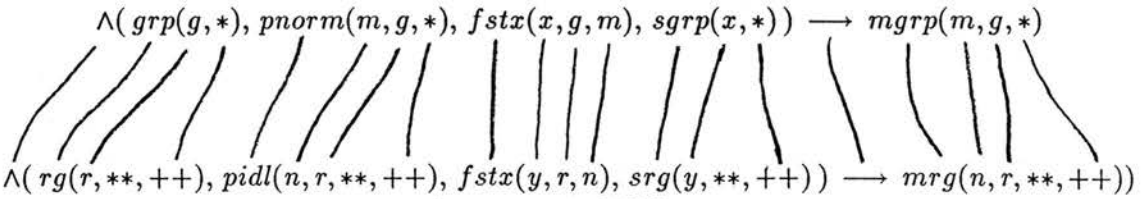
The new node, $(simplegroup(x, *), simplering(y, **, ++))$ is analysed at level 1, and the pairing (1,1) is made on the basis of criterion 1 (criterion 3 again suggests all possible pairings); similarly, the other new nodes, $(propernormal(m, g, *), properideal(n, r, **, ++))$ and $(group(g, *), ring(r, **, ++))$ are refined at level 1 by criterion 1; no further progress is made at level 1.

At level 2 the five remaining nodes, are re-analysed, but there are no clearly indicated pairings. The match therefore goes to level 3; the match is split up on the basis of the node $(group(g, *), ring(r, **, ++))$: the pairing (1,1) has already been made; the remaining possible pairings, (2,2) and (2,3), are both suggested by criterion 3; each of these pairings leads to a new match.

Each of these new matches is developed further. In each case, the matching is completed at level 1 by criterion 1. Therefore, two matches are constructed, which are shown below.



Assessment: 0.935



Assessment: 0.935

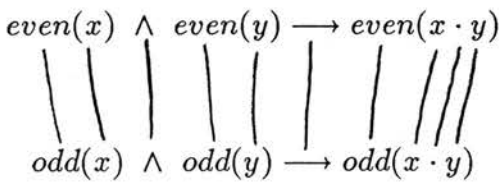
The only difference between the two matches which are produced is that, in one, the group operator is associated with the ring addition, whereas, in the other, it is associated with the ring multiplication. The matcher cannot distinguish between these possibilities and gives them the same assessment (0.935).

It is interesting that this example does not actually need the semantic templates in order to be matched. On other of Kling's algebra examples, the matcher produces more than two consistent matches; in some of these, set variables are matched with operator variables. Kling avoids these by insisting, in ATOMMATCH, that argument pairing respects argument types. The argument type heuristic is not used in the present matcher; it could be added as one of the pairing criteria to prune the matches which do not preserve the argument types; alternatively, it could just be added to the assessment phase, so that the type-respecting matches are preferred.

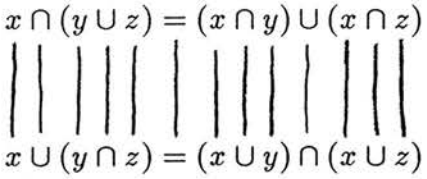
4.5 Assessment of new matcher

We claim that, in terms of performance, the current matcher is a considerable improvement over those of Munyer and Kling. The flexible use which the current matcher makes of the analogy heuristics enables it to find a wider range of matches than either Munyer's or Kling's. We summarize important aspects of the greater flexibility below; Appendix A contains a collection of examples, together with the performance of the three matchers on them.

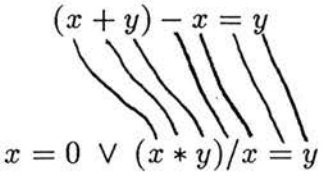
As explained in the previous chapter, Munyer's matcher is unable to construct matches which involve significant proportions of non-identical associations, such as the following ones, which are constructed by the current matcher:



(number theory)

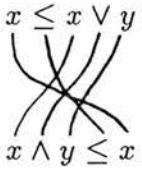


(Boolean algebra)



(arithmetic)

Because of the arbitrary nature of its procedure ATOMMATCH, Kling's matcher is unable to construct matches which involve any permutation of arguments, such as the following one from Boolean algebra, constructed by the current matcher:



Kling's matcher arbitrarily enforces preservation of (relative) argument position. If the functions (predicates in Kling's syntax) are commutative, this restriction means that Kling's matcher will only consider one possibility for pairing them up; this makes the performance of the matcher very sensitive to minor changes in presentation, such as which way round the arguments to a commutative function are written. The current matcher takes commutativity into account, and thus is not affected by these minor presentation changes.

We have seen that Kling's matcher can be modified to make it able to construct matches involving unmatched conditions and arguments, but only if the extra conditions have unique semantic templates. Otherwise, the SINGLEMATCH/MULTIMATCH distinction, which is the basis of the algorithm, is violated. This arbitrary restriction is removed in the current matcher.

The current matcher can be thought of as a generalisation of Kling's: it shares the basic top-down approach to matching. It is a generalisation in that it accepts an unrestricted syntax, removes arbitrary restrictions of Kling's matcher as explained above and makes more flexible use of the analogy heuristics. Furthermore, the agenda-based refinement architecture of the current matcher makes it more persistent, and therefore powerful, in matching than Kling's matcher.

The current matcher is able to construct essentially the same matches, together with others in some cases, as Kling's on the abstract algebra examples, without making any use of the semantic templates on which Kling's matcher relies. The other matches which the current matcher produces are usually just as strong, from a purely syntactic point of view, as the intended matches, but some variable associations do not preserve the argument types of their containing predicates. As has been argued above, the use of these templates represents a circular argument, as they encode most of the information about the global analogy between group theory and ring theory. We also argued, however, that there is no reason why an analogy matcher should not make use of global analogies which it knows, in the guise of semantic types or perhaps some more explicit representation, in order to construct matches between individual problems, as long as no pretence is made that the matcher is *discovering* the analogy. If the presenty matcher was guided by semantic templates (that is, predicate types and/or argument types) the unintended matches would be blocked, or at least be given lower strength.

Naturalness and flexibility

A complete description of the matcher is certainly complicated. However, the matcher is based around some simple requirements

- The central procedure of matching is a recursive pairing analysis, based on analogy heuristics.

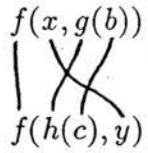
- The matcher should be flexible with respect to the heuristics.

The structure of the matcher follows directly from these requirements: the or-agenda of partial matches, the and-agenda of matching subproblems and their development, the separate pairing criteria. On this basis, we claim that the matcher embodies a natural approach to analogy matching.

The lower level procedures, such as the evaluation of pairing suggestions, the particular pairing criteria used and the assessment procedure, are not based so directly on the initial motivation. The choices made for these procedures make the current matcher one of a class of matchers of similar structure; other members of the class are obtained by varying the lower level procedures. For example, new criteria can be added, based on new heuristics; similarly, the pairing evaluation and match assessment procedures can be easily changed to produce alternative matchers — the next section discusses some directions for change. We regard this flexibility, particularly with respect to the analogy heuristics, as an important advantage of the current matcher over previous ones; as we found in the previous chapter, Munyer's and Kling's algorithms were strongly biased, in their structure, towards weak heuristics; this limited their development.

4.6 Areas for improvement

First-order unifiability We saw in the previous chapter that Munyer's algorithm can be modified so that it becomes complete for first order unifiable terms, i.e. finds a match that represents the most general unifier between such terms. The current matcher is not complete in this way. For example, the current matcher constructs the following match:



These two terms are unifiable, with common instance $f(h(c), g(b))$. The current matcher does not find the unifier. While a unification with a previous problem is no guarantee of being able to solve the given problem

analogously, it might well help, and is certainly relevant information. One way of getting round this problem would be to extend the syntactic structure criterion (criterion 3) so that it took account of variables, and the possibility of their being bound in a match; this would, however, considerably reduce the discriminating power of the criterion. Another would simply be to have a unification check separate from the analogy matching algorithm. The latter suggestion has the drawback that matches which were 'nearly' unifications would be unlikely to be found.[11]

Completeness A more ambitious aim than that just discussed would be to have a matcher which conforms to a more general completeness criterion for analogy matches (rather than for the subclass of first-order unifications). That is, we would have a precise definition for what constitutes an acceptable match, and would be able to prove that the matcher would be able to construct any match of that type. For example, the definition of acceptability could be based on the match assessment procedure — a match would be acceptable if and only if it had assessment greater than a fixed number. Bundy [Bundy 87] has argued for the desirability of such a criterion.

The current matcher clearly fails to satisfy any completeness criterion which includes first-order unifiability. With criterion 3 extended, the matcher would have a better chance of conforming to a completeness criterion. An exhaustive matcher, which constructed and checked all possible matches between a pair of formulae would naturally satisfy a completeness criterion. The problem would be to find a balance between thoroughness and reasonable efficiency; a matcher, such as the current one, based around the application of opportunistic pairing criteria, will tend not to satisfy uniform completeness criteria.

---

[11] Note that Munyer's matcher is also unlikely to find near unifications if they involve function mismatches.

Pairing evaluation The evaluation of pairing suggestions, the pruning of weakly suggested pairings and the rule for suggesting structural mismatches are all quite simple in the current matcher. The methodology for development of the matcher has been to make the procedures only as sophisticated as necessary to perform as desired on the examples being considered.

The present rule for pruning weakly suggested pairings is simply to remove those competing with others which are suggested by a strict superset of criteria. This may lead to good matches being missed because necessary argument pairings were not made early on in the matching process. While there will always be a danger of this in a matcher which uses heuristic criteria to prune the analogy search space, the danger would perhaps be lessened if the pruning rule was made more flexible; in particular, it could be made responsive to the strengths of different criteria, rather than just their identity; it could then prune pairings which were significantly weaker than others.

Similarly, the rule for assessment of pairing suggestions could be made more flexible; for example, pairings might only be made at level 1 if there was good overall pairing information, rather than, say, just one criterion firing between one pair of arguments; in cases such as the latter, the refinement would be delayed until level 2.

The rule for suggesting structural mismatches is quite fussy at the moment: if there is 'crosstalk' in the suggestions from criterion 4, the rule may well not fire in cases of potential mismatches.

However the present simple procedures have proved effective in producing a matcher of greater power and generality than previous ones; this fact indicates the strength gained from the agenda-based refinement architecture, and the generality gained from flexible application of the heuristics.

4.7 Limitations of matching on given problem statements alone

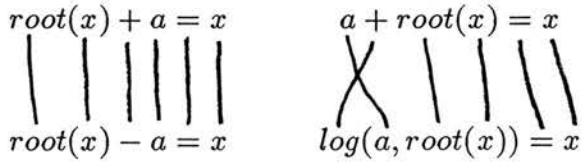
All the matchers which we have discussed construct matches between problem statements as presented to them; they find *structural* correspondences between the given problem statements without much regard to the 'meanings' of the symbols, or the context in which the problems arise. In fact, all of the matchers do take some account of the properties of the symbols in matching: Kling's uses semantic templates, which represent global analogies; Munyer's matcher and the current matcher both take account of the commutativity or otherwise of symbols in matching. Furthermore, the current matcher can, if necessary, change the propositional form of formulae in trying to match them up. Nevertheless, the view of analogy embodied in the matchers relates basically to the problem statements in isolation. In this section, we consider some limitations of this approach to analogy in mathematical problem solving.

The philosophy of analogy matching on the syntactic structure of the formulae representing problems is based on the hope that the analogies between problems will be apparent in their syntax, or at least that there will be sufficient clues in the syntax for the solution of the base problem to be of use in solving the target problem. There are two main limitations to this philosophy:

- Superficial difference problem. Analogies between problems are often obscured, from the syntactic point of view, by superficial differences in their representations; by superficial, we do not mean that they are necessarily trivial, but merely that the differences are not crucial to the form of the solutions.

- Multiple matches problem. There may be an analogy between two problems which is clear in the syntax, but there may be many other equally good syntactic matches with the target problem, and no syntactic way of distinguishing between them.

Multiple matches problem. This problem was referred to in the previous chapter, in discussion of the justification of the semantic type heuristic; it was argued that this heuristic can be used to alleviate the problem; for example, it would be used to prefer the first of the following two matches to the second, on the basis that + and − are of closer semantic type than + and *log*:



When searching for an analogue in a large knowledge base of solved problems, this problem could occur often.

Another, perhaps complementary, solution to the multiple matches problem would be just to try to apply all of the syntactically promising matches, in some interleaved manner, and discard those that did not turn out to be useful.

Thus, this limitation of problem-based analogy matching does not seem to present severe difficulties.

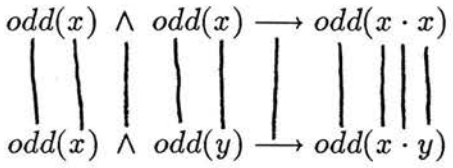
Superficial difference problem. The superficial difference problem is more challenging. Some superficial differences, such as the argument order of commutative functions and the propositional form of the formulae, really are trivial, and are already coped with adequately by the current matcher. However, others, such as the following example from elementary number theory, are more difficult:

$$odd(x) \longrightarrow odd(x^2)$$

$$odd(x) \ \wedge \ odd(y) \longrightarrow odd(x \cdot y)$$

Intuitively, these are analogous problems; in fact, one is a special case of the other. Furthermore, we could reasonably expect the solution to one to be of use in trying to solve the other. However, there is no close syntactic correspondence between the formulae which would enable a syntactic matcher to treat them as analogous. However, by applying the rules $P \wedge P \longleftrightarrow P$ and $x^2 = x \cdot x$ to the

first of the formulae, we can transform it into an equivalent formula which makes the analogy clear in the syntax:



As to the usefulness of this match, it would clearly be easier if the first of the original pair was the target problem, since the base solution could be used without modification; it would be more difficult if the base problem was the one which was rewritten, since the existing base proof might have to be altered significantly before it could be used. However, in either case, the correspondence between the problems is certainly *relevant*, and we would like an analogy matcher to be able to find it.

In order to be able to notice the similarity, the matcher would clearly need access to the rules with which the problems are made syntactically similar, in this case $P \wedge P \longleftrightarrow P$ and $x^2 = x \cdot x$, and would need to be able to apply them to one or both of the formulae. What is called for seems related to **unification modulo a theory** [Siekmann 86], in which terms are rewritten using equations from the theory, if necessary, in order to allow unification to proceed. We would need **analogy modulo a theory**. The difficulty with looking at the problem in this way is that, unlike in the case of unification, we do not have a precise specification for what we are looking for when we rewrite the terms. This would compound with the search problem already inherent in the rewriting.

The superficial differences discussed so far, whether easy or difficult, are solved by re-representing one or both of the formulae to be matched in order to make the analogy clear. The re-representations are achieved by making inferences within the object language — propositional inferences, applications of commutativity for certain functions, or, more generally, application of any axioms from the theory. However, it is possible to make more radical re-representations than these in order to remove superficial differences: the object level formulae may be described in a **meta-language**; object level formulae which are

very different syntactically may nevertheless have similar meta-level descriptions. This kind of re-representation is used by the schema learning system LP; as discussed in Chapter 2, LP makes use of a sophisticated meta-language to describe the equation solving process. The same meta-level descriptors, such as *common_subterms*, may apply to two equations which do not appear at all similar syntactically. The schemas which LP constructs from worked examples, which are expressed in the terms of the meta-language, will thus match with equations very different syntactically from the equation solved in the worked example.

The hope embodied in this use of the meta-language is that equations which have similar meta-level descriptions can be solved in similar ways; that is, that the differences in the object level equations are superficial. It was argued in Chapter 2 that this allows greater leaps of generalisation than would be possible by object-level manipulations alone. In the case of LP, the meta-level matching between schemas and new equations is strict — the equations must satisfy the preconditions of the schema. This strict use of schemas assumes that the fixed meta-language generalises the worked examples as far as possible. This depends on the meta-language being very well developed for the domain in question. In an advanced learning and reasoning system, we might well find that a meta-language had developed partially for a domain, i.e. that it expressed certain generalisations within the domain, but not others. In this situation, we would want to allow flexible matching at the meta-level as well, in order to extend schemas to new situations not envisaged in the existing meta-language. Thus, we could use the existing analogy matching algorithms, but lift them to the meta-level. The matching process would then be making use of the analysis of the domain represented by the meta-language to remove 'superficial' differences between old and new problems.

4.8 Summary

In this chapter, we have developed a new analogy matching algorithm. The new matcher makes flexible use of the heuristics which it embodies. There is only one heuristic implicit in the structure of the algorithm, the partial homomorphism heuristic, which is a strong heuristic; moreover, there is considerable scope for structural variation in the algorithm.

The matcher is also easily modifiable in the ways called for in section 4.1: the procedures for applying the heuristic criteria are separate from the criteria themselves; new heuristic criteria can be added easily (the late addition of criterion 3 exemplifies this), and the relative significance of existing criteria can be altered (criteria can be switched between levels 1 and 2, and the weightings given by the assessment procedure can be altered).

The matcher also accepts an unrestricted syntax, as called for in section 4.1.

The performance of the new matcher has been seen to be a considerable improvement on that of the existing matchers: we have seen, by general argument, that the new matcher produces a wider class of common matches; this has further been illustrated on particular examples, and is illustrated on many more in Appendix A.

We therefore have some confidence that the current matcher can be a useful component of an analogy system. Subsequent chapters, in which the matcher is repeatedly called on in such a system, will add to this confidence.

We have also indicated ways in which the procedures for co-ordinating the criteria may need to be made more flexible, for more difficult examples of analogy.

Lastly, we have considered some of the limitations to the problem-based approach to analogy taken by all the matchers we have considered. We have seen that the apprehension of an analogy often crucially involves re-representations of one or both of the problems. While we have not suggested any computational solutions to the problems of re-representation, we have argued that a structural

match is still involved on the transformed problems. Thus the techniques for structural matching will still be a necessary part of a more sophisticated analogy system.

Chapter 5

How to Map a Proof Using an Analogy

In this chapter we consider the next stage of Basic APS, in which a plan for the solution of the target problem is constructed from the known base solution and the initial match between problems. In particular, we describe how to construct analogues for steps made in the base solution; and how to find analogues for axioms applied in the base solution. Both processes are compared (in the latter case on the basis of a reconstruction) with the corresponding components of Kling's and Munyer's systems. We also discuss the mapping of the inference positions from the base solution.

5.1 Introduction

The previous two chapters have been concerned with the finding of analogies via analogy matching. In this chapter, we consider the next stage of the analogy framework which was described in Chapter 1 — the construction of a plan for the solution of the target problem, given a match with a base problem, and the solution to the base problem. We can think of the plan construction phase as being separate from the plan application phase, described in Chapter 6 — given a match with a base problem we could construct a complete plan before

trying to apply any of it. However, we will see that, in practice, it may be wise to interleave the construction and application stages, constructing a part of the plan, attempting to apply it, then constructing some more, and so on. Therefore, in this chapter, we describe the procedures for constructing individual parts of the plan, and leave the discussion of how the procedures are co-ordinated with the application of the plan until Chapter 6.

As explained in Chapter 1, the analogy system which is described in this thesis is based on a resolution problem solving system. Therefore, a base solution will be a resolution proof of the base problem. Such a proof contains the following three types of clauses:

- Axioms. These are axioms of the theory which are used in the proof.
- Goal clauses. These are the clauses which are produced by negating, normal-forming and skolemising the conjecture which is being proved.
- Intermediate steps. These are clauses obtained by the resolution or paramodulation[1] of previous clauses in the proof.

In addition, we will know the **inference positions** of the inferences which were made in the base proof; i.e. the positions within their clauses of the literals which were unified, in a binary resolution step, and the subterm position at which an equality was applied, in a paramodulation step. In order to map the base solution over into a plan for the solution of the target, we must be able to map over each of the three types of clauses; we also consider how to map over the inference positions, in order to predict how to make corresponding target inferences to those which were made in the base.

While we are concerned here with a resolution system, the same types of clauses and connection positions will arise in any automated problem solving

[1]Paramodulation [Bundy 83] is an inference rule which formalises substitution by equality.

system. The techniques described in this chapter will thus be applicable to other kinds of problem solving system.

For the goal clauses, we use the structural correspondence from the match between the problem statements to find analogous goal clauses from the target problem. This process is straightforward, and will not be described further here.

For the intermediate steps, we want to construct their analogues as best we can according to the current match; there is no *a priori* restriction on the form of the analogues; i.e. we are solving problems of the form

A is to B as C is to X

where A and C are the problem statements[2], B is an intermediate step in the base solution, and X is the (unconstrained) unknown. Given the existing match between A and C, we *apply* the match to B to get X. This application is done by the **analogue construction rule** (ACR) described in section 3.

For the axioms, we have a similar problem to the one above, except that X is constrained to be an axiom (or theorem) from the knowledge base. The 'free' application of the ACR is not guaranteed to result in an axiom, so we need some way of finding the axiom, or axioms, which is closest to being the analogue of B. This is done by the **analogous operator identification** routine (AOI), described in section 4.

In section 2, we take care of a slight complication to the story, arising from the refutation method of resolution systems: we describe how the initial mapping needs to be negated and partitioned before it is used in mapping parts of the base proof.

---

[2]More generally, A and C may stand for whatever has been matched so far to produce the current mapping — this may include previous axioms and intermediate steps as well as the original problem statements.

5.2 Negating and partitioning the mapping

Refutation problem solving systems, such as resolution, prove that a conjecture follows from some axioms by showing that, in the presence of the axioms, the negation of the conjecture leads to a contradiction. The initial match between problems is between the skolemised forms of the *un*-negated conjectures; the goal clauses, being skolemised forms of the negated base conjecture, will contain skolem terms where the matched clause contains variables, and vice-versa. Thus, the initial mapping will not apply directly to the goal clauses. The same is true of the intermediate steps, which will contain skolem constants and variables introduced by the goal clauses. However, the negated forms may be obtained from the positive forms by a consistent replacement of variables and skolem terms by skolem terms and variables respectively: for example, if the positive form is

$$p(x_0, y, z_0(y)),$$

where x_0 is a skolem constant and $z_0(y)$ is a skolem function, arising from the unskolemised conjecture

$$\exists x\ \forall y\ \exists z\ \ p(x, y, z),$$

then the negated form is

$$\neg p(x, y_0(x), z),$$

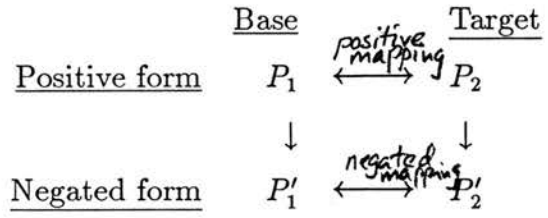
arising from the negated conjecture

$$\forall x\ \exists y\ \forall z\ \ \neg p(x, y, z).$$

The associated replacement substitution is

$$\begin{aligned} x_0 &\longrightarrow x \\ z_0(y) &\longrightarrow z \\ y &\longrightarrow y_0(x) \end{aligned}$$

By composing with the replacement substitutions for the base and target conjectures, we transform the initial mapping between the positive forms into a negated mapping between the negated forms, as shown below



It it is the negated mapping which is used in mapping the goal clauses and intermediate steps. For example, if the positive base and target conjectures (from Boolean algebra) were

$$x \cup x = x \quad \text{and} \quad x \cap x = x$$

respectively[3], the initial mapping between them was

$$\{x \leftrightarrow x,\ \cup \leftrightarrow \cap,\ = \leftrightarrow =\},$$

and the negated forms were

$$\neg a \cup a = a \quad \text{and} \quad \neg b \cap b = b,$$

the replacement substitutions would be

$$\{x \longrightarrow a\} \quad \text{and} \quad \{x \longrightarrow b\},$$

and the negated mapping would be

$$\{a \leftrightarrow b,\ \cup \leftrightarrow \cap,\ = \leftrightarrow =\}.$$

The part of the initial mapping which is transformed by the above procedure, that involving variables and skolem terms, is relevant for mapping goal clauses and intermediate steps, but not base axioms. Therefore, the negated mapping is partitioned into two subsets, one containing associations free of skolem terms and variables, and the other containing all other associations. In the above example, we would have

$$\{\cup \leftrightarrow \cap,\ = \leftrightarrow =\}$$

---

[3]The meet and join functions of Boolean algebra are written $\cap$ and $\cup$ respectively to distinguish them from the logical connectives $\wedge$ and $\vee$.

and

$$\{a \leftrightarrow b\}.$$

An alternative to negating and partitioning the mapping would be to negate and skolemize the problem statements before matching them. We do not do this for two reasons: firstly, it is much easier for a user to interpret a match between positive forms of the problem statements, since these directly express what is being asserted; secondly, and more importantly, if an analogy is successful we may wish to store the match for later use in guiding other analogy attempts (Chapter 7 discusses this in detail); the match would be used in its positive form; thus, we would have to invert the mapping at some stage anyway.

5.3 Analogue Construction Rule

As explained in the introduction to this chapter, the ACR is given the existing match, and a clause from the base proof, and has to construct the (best guess at the) analogue of the clause according to the mapping.

The basic idea behind the ACR is simple: view the existing match as a mapping from base to target, and apply the mapping to the clause to get its analogue. For example, supposing the match contained the associations[4]

$$(\cup, \cap, id),\ (\cap, \cup, id),\ (\leq, \leq, [(1,2),(2,1)]), (=, =, id),\ (a, b, term)\ \textit{and}\ (0, 1, term),$$

and the clause from the base proof to be mapped was

$$a \cup (a \cap 0) = a.$$

---

[4]Recall that the third component of the associations give argument pairing information about the symbol pair; 'id' (short for identical) means that argument order is preserved, 'term' (short for terminal) that there are no arguments to be paired, and otherwise the pairing is given by a list of pairs (i,j), meaning that argument i on the left is paired with argument j on the right.

Then the correct analogue would clearly be

$$b \cap (b \cup 1) = b.$$

If the base clause was

$$a \leq a \cup a,$$

then the analogue would be

$$b \cap b \leq b$$

However, the following cases may arise, which the ACR will have to cope with:

1. The base clause contains symbols which are not mapped in the existing match. This will happen often, as proofs almost always contain symbols which do not occur in the original problem statement.

2. The base clause contains a symbol which is mapped to more than one other by the existing match (i.e. the match is inconsistent with respect to this symbol).

The ACR described below treats these cases as follows:

1. The symbols are copied over to the target, with identical pairings if they are not atomic. The results are flagged as being default symbols, as this fact is relevant for plan application. For example, with the mapping used above, the analogue of the clause

   $$a \cup \overline{a} = 1$$

   would be

   $$b \cap def(\overline{b}) = def(1).$$

   The unary function symbol def is special syntax introduced to flag symbols which have been mapped by default; $def(T)$, for some term T, indicates that the head function symbol of T has been mapped by default. Depending on how the plan application procedures treat the default symbols

in a plan, their use may involve an application of the **identical symbols heuristic**; in fact, the application procedures described in the next section do not attach significance to the identity of the symbol which is mapped by default, just to the fact that it is a default symbol, so the heuristic is not used in this case; however, different application procedures might attach significance to the identity of default symbols.

2. All analogues are constructed and are returned either as a list, or as a more compact notation, if this is possible (see below). The list can be thought of as a disjunction of possibilities, and the plan application procedures of the next chapter treats it in this way. For example, if the mapping used above in addition contained the association $(a, c, term)$, the analogue of

$$a \cup (a \cap 0) = a$$

would be

$$[b, c] \cap ([b, c] \cup 1) = [b, c].$$

We next give the formal definition of the ACR which has the behaviour called for above. The definition is by recursion on the structure of the term whose analogue is being constructed. Firstly, some notation used in the definition: for a term or symbol T and a match α, let

$$T_\alpha = \{V | (T, V, term) \in \alpha\};$$

i.e. T_α is the set of terms to which T is directly mapped by α.

Analogue Construction Rule

Given a term T and a mapping α, ACR produces the result of applying the mapping to the term, T^α.

Base case. T is atomic: if $T_\alpha \neq \phi$ then $T^\alpha = T_\alpha$ (if T^α contains just one element, it is represented as the element, to improve readability); if $T_\alpha = \phi$ then $T^\alpha \doteq def(T)$.

Recursive case. T is not atomic, $T^\alpha = T_\alpha \cup {}_\alpha T$, where ${}_\alpha T$ is as defined

below[5] (again, if T^α contains just one element, it is represented as the element): if

$$T = f(a_1, a_2, \ldots a_n),$$

let $a_1^\alpha, a_2^\alpha, \ldots a_n^\alpha$ be the results of the recursive calls to the ACR. There are three subcases to the recursive case:

(a) If $f_\alpha = \phi$ then

$$_\alpha T = \{def(f(a_1^\alpha, a_2^\alpha, \ldots a_n^\alpha))\}.$$

(b) If f_α contains just one element, g, where $(f, g, P_{fg}) \in \alpha$, then

$$_\alpha T = \{g({a_{\sigma(1)}}^\alpha, {a_{\sigma(2)}}^\alpha, \ldots {a_{\sigma(n)}}^\alpha)\},$$

where σ is the argument mapping implied by P_{fg} [6] If the previous argument pairings of f and g, P_{fg}, leave an argument unmapped on the left (represented as $(p(i, t'), u)$ in P_{fg}, meaning that the ith argument, t', was unmatched previously), then for any j such that a_j unifies with t', a_j^α is flagged in $_\alpha T$ in the form $bdef(a_j^\alpha)$ to signify the fact that it may need to be omitted; in addition, if f is not known to be commutative, $bdef(a_i^\alpha)$ is added as a final argument to $_\alpha T$ (it would otherwise be erased by the above definition); if argument i is unmapped on the right (represented as $(u, p(i, t'))$ in P_{fg}), then the term t' is introduced in the ith argument position of $_\alpha T$; again, it is introduced in the form $bdef(t')$ to signify its special status. (These complicated conditions involving $bdef$ are explained in the remarks below.)

---

[5] T_α is the direct analogue, if any, of T, and $_\alpha T$ is the composite analogue, formed from the analogues of the constituents of T.

[6] If the previous occurrences of the (f, g) association have consistent argument pairings, σ is derived from these; if the previous occurrences are inconsistent, the first set of pairings in P_{fg} is used for σ.

(c) If f_α contains more than one element, say $[g_1, g_2, \ldots]$, the pairings of f with each of the g's are checked to see if they are consistent; if they are, then

$$_\alpha T = \{poss([g_1, g_2, \ldots], [a_{\sigma(1)}{}^\alpha, a_{\sigma(2)}{}^\alpha, \ldots])\},$$

where σ is the (consistent) argument mapping derived from the P_{fg}'s. This is the compact notation referred to above (the meaning of *poss* is explained in the remarks which follow). If the pairings are not mutually consistent,

$$_\alpha T = \{\ g_1(a_{\sigma_1(1)}{}^\alpha, a_{\sigma_1(2)}{}^\alpha, \ldots a_{\sigma_1(n)}{}^\alpha),\ g_2(a_{\sigma_2(1)}{}^\alpha, a_{\sigma_2(2)}{}^\alpha, \ldots a_{\sigma_2(n)}{}^\alpha), \ldots\ldots\},$$

i.e. just the set of the individual analogues. The same conditions on unmatched arguments in (b) apply to this case as well.

Remarks

- For most composite terms, $T_\alpha = \phi$; i.e. it is rare for composite terms to have direct analogues in the mapping. Thus, usually $T^\alpha = {}_\alpha T$. Where a composite term does have direct analogues, these are considered to be alternatives to the composite analogues derived by recursion on the structure of T, and so the union of the possibilities is taken.

- $poss(L, Args)$, where L is a list of function or predicate symbols stands for the list of terms constructable from symbols in L applied to $Args$; the notation is more compact that the list of the possibilities, and makes the analogue easier to interpret.

- The treatment of associations which include unmatched arguments is illustrated by the following example: Suppose the terms

$$angle(a, b, c) \quad \text{and} \quad segment(a, c)$$

have been matched together previously (say, within the problem statements), and have given rise to the association

$$(angle, segment, [(1, 1), (p(2, b), u), (3, 2)])$$

in the existing match; if, later, we want to construct the analogue (from left to right) of the term

$$angle(d, b, e),$$

the ACR would produce

$$segment(d, e, bdef(b)).$$

If we wanted to construct the analogue (from right to left) of

$$segment(x, z),$$

the ACR would produce

$$angle(x, bdef(b), z).$$

Terms headed by the unary $bdef$ function symbol should be thought of as optional. This means that they are deemed to unify with another term if they unify (in the normal sense) either with or without the term inside the $bdef$. The motivation behind the conditions involving $bdef$ is that we want to allow arguments to be erased or introduced again, where this has happened before. However, it would be hard to be confident as to just when arguments would be introduced or erased; the symbol $bdef$ is introduced and interpreted to allow flexibility — i.e. arguments may or may not be unmatched again. For a non-commutative symbol, an argument position which has been unmatched in the past is a candidate for being erased by the ACR, so $bdef$ is used for such arguments; for commutative symbols, particular argument positions are of no significance, so another condition is added which looks for arguments which unify with previously erased arguments, and flags these with $bdef$ as candidates for erasure. Flagging extra arguments with $bdef$ merely increases the generality of the analogue produced by the ACR, so the particular conditions under which $bdef$ is used are not critical. The true significance of $bdef$ (and the other special syntax introduced by the ACR) will only be clarified when the procedures which make use of the analogue steps are described in the next chapter.

- The ACR as defined above produces a structural correspondence between a term, T, and its analogue T^{α}, as a side effect (each symbol is associated with the symbol to which it is mapped). The structural correspondence is used in mapping inference positions (see below).

- The exotic cases which make the definition complicated, such as what to do if there is more than one prior pairing, and if arguments have been unmapped previously, are necessary for the definition to be complete; however, they do not, in practice, arise very often.

5.3.1 Comparison with Munyer's AIR

Munyer's analogy system contains a procedure which has the same purpose as the ACR described in the previous section: to predict an analogous step in the target to one which was made in the base. Munyer calls the procedure the **analogical inference rule**. He uses the term 'inference rule' since he regards the construction of an analogue as an approximate form of inference; however, the use of the procedure corresponds to that of the ACR, since approximate inferences need to be validated before they can be accepted.

Munyer's definition for his AIR is a complex case analysis; on investigation, it turns out to contain numerous errors and inconsistencies in the cases, and cannot be considered to be a coherent algorithm; the planning method which makes use of the AIR was never implemented by Munyer, and perhaps the AIR itself was never implemented. It is clear from the examples that Munyer gives that he intends the AIR to do the same sorts of things as the ACR described in the previous section. However, there are also overall differences which are discussed below:

- Munyer's AIR uses no special syntax to represent the results of unmatched symbols and inconsistencies; for example, if a symbol is unmatched, it will be copied over in the analogue, another application of the **identical symbols heuristic**. This will undoubtedly lead to problems for the plan

application procedure: whenever a symbol is introduced by a step of the base proof, it will be assumed to be preserved by the analogy; whenever this is not so, the plan will be flawed and will need to be corrected at some stage; plan correction (see Chapter 7) is a difficult enterprise, involving extra search. Thus it is sensible to avoid as many flaws in the plan as possible; the ACR does this, for unmapped symbols, by flagging the result using the special unary function def; the application procedure is then able to match the default symbol with any symbol in order to allow the step to go through.

- Munyer's AIR makes use of what he calls **inference links** to communicate the analogy between successive steps of the base proof; i.e. the analogue for a symbol is determined by the analogue for a symbol from the previous step to which it is (inference) linked. This contrasts with the approach taken in this thesis, and by Kling, in which the analogy is conveyed by an evolving global mapping. Munyer gives no account of the meaning of the links — almost all of them, in his mathematical examples, are between identical symbols and represent shared structure in the base proof. The effect of the links seems to be to subdivide occurrences of a symbol in a term according to the occurrences, if any, to which they are linked in the previous term. The main potential benefit seems to be that inconsistencies in the mapping can be avoided in this way. However, the value of this is doubtful: while it avoids the introduction of disjunctions into the plan, this will have the effect of causing the plan to be unnecessarily flawed if there is a genuine inconsistency.

  It would be easy to change the ACR to account for inference links, if this was thought sensible: whenever there is a check of the form $(f, g, P_{fg}) \in \alpha$ in the above definition, it would be replaced by a check of the form $linked(f, f')$ *and* $(f', g, P_{f'g}) \in \alpha$, where the symbols f, f' and g would be particular occurrences of the symbols concerned, and would thus include the position of occurrence in a named clause. The main difficulty involved with the introduction of inference links into the analogy system would be

updating and propagating the links in the base proof: given this, their incorporation into the ACR would be relatively straightforward.

5.4 Analogous Operator Identification

As explained above, the purpose of analogous operator identification (AOI) is to find plausible analogues (from the axiom/theorem base) for axioms which were used in the base proof, given a mapping derived from the problem statements. In addition, as we shall see below, the AOI can produce an extension to the initial mapping between problems. As noted above, the analogue of an axiom, A, with respect to the existing mapping α, A^{α}, will not in general be in the knowledge base. This is why a separate procedure is required for the AOI. Kling's analogy system, ZORBA, forms unstructured plans for the solution of the target problem which consist entirely of potential analogues for the axioms used in the base — ZORBA makes no use of intermediate steps.

We first analyse Kling's procedure EXTENDER, for finding the analogues of axioms. The analysis involves a reconstruction of EXTENDER, and assessment of its performance on some examples. In the following sections we propose an AOI which is an improvement on Kling's, and discuss its performance.

5.4.1 Kling's EXTENDER

We first give a brief description of Kling's algorithm:

Description

Given a mapping, α, and a set of axioms whose analogues are sought, first find the subset, $SOME_{\alpha}$, which consists of those axioms *some* (not all) of whose symbols are mapped by α. For each $A \in SOME_{\alpha}$, form a **description** of

A, descr(A). descr(A) is a list of the predicates [7] that occur in A, flagged by the polarities of the positions in which they occur — *pos* if all occurrences are positive, *neg* if they are all negative, and *impcond* if there are some of each. Thus if A is

$$\neg subgroup(h, g, *) \vee group(g, *)$$

then descr(A) is

$$[pos(group), neg(subgroup)].$$

Then map descr(A) to get descr(A)$^\alpha$ (this type of mapping is obviously much simpler than that described in the previous section). Express $descr(A)^\alpha$ as

$$\overline{descr(A)^\alpha} \cup Defs,$$

where Defs consists of that part of $descr(A)^\alpha$ which was mapped by default, and $\overline{descr(A)^\alpha}$ is that part which was positively mapped. Then return as possible analogues for A those axioms B such that

$$\text{(a)} \quad descr(B) \supseteq \overline{descr(A)^\alpha}$$

and (b) $|descr(B) \cap Defs|$ is maximal over all axioms satisfying (a). We can think of (a) as a filter, and (b) as a metric which is applied to axioms which pass through the filter. The filter (a) is an application of the **consistent translation** heuristic — we look for axioms which contain symbols which are associated with ones in the base axiom. The metric (b) is a direct application of the **identical symbols** heuristic — analogous axioms are chosen which will lead to identical associations being added to the analogy.

Given such a B, the existing mapping is extended in the following way: form the sets

$$descr(B) \setminus \overline{descr(A)^\alpha} \text{ and } Defs = descr(A)^\alpha \setminus \overline{descr(A)^\alpha}.$$

---

[7] Recall that Kling's clauses contain no functions or constants — in a less restricted syntax, the descriptions would obviously include the other symbols (see below).

For any pair of identical features (including the polarity signs) between these two sets, add the corresponding predicate association to the mapping, and delete the features from the sets. Out of the remainder, for any pair of features of the same polarity, whose predicates are of the same (unique) semantic type, add the predicate association to the existing mapping.

The procedure is repeated for each extended mapping, until the set $SOME_\alpha$ is empty.

Reconstruction

The procedure described above was reconstructed in Prolog, and its performance was analysed. The following difficulties with the routine became apparent:

1. It is likely that Kling's routine will fail to find analogues for certain of the axioms used in the base proof. Any axiom that has all its predicates mapped by the initial mapping will never be considered by the procedure. This is presumably an oversight on Kling's part; the situation perhaps never arose in the examples which he was considering. It is straightforward to change the procedure so that it takes account of such axioms, and this was done for the reconstruction.

2. Axioms whose descriptions contain only one feature, will tend to give rise to many possible analogues, most of them obviously unsuitable. For example, 12 analogues were found for the axiom $x = x$, out of a base of just 19 axioms, including such unlikely candidates as $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$. There is clearly something wrong with the metric for this to happen. As the axiom base expanded, this problem would become increasingly severe.

3. When an association is added to the mapping, no argument pairing information is included, because all such information is lost in forming the descriptions. For example, if the axioms in question were $x \leq x \cup y$ and $x \cap y \leq x$, the extended match *ought* to include the association

$(\leq, \leq, [(1,2),(2,1)])$, but Kling's procedure would produce $(\leq, \leq, id)$ by default. This could be important at later stages of the Basic APS process[8].

4. The extension procedure is heavily geared towards adding identical associations, both in the metric for choosing possible analogues, and in the procedure for extending the mapping based on them. For example, if A is

$$\neg rat(x) \vee \neg rat(y) \vee rat(x - y),$$

and '$-$' is not in the initial map, then the axiom B,

$$\neg rat(x) \vee \neg rat(y) \vee y = 0 \vee rat(x/y),$$

would not be considered as a possible analogue, since A itself has a higher default match than B. But if the initial mapping contained the association $(+, \cdot, id)$, then B would actually turn out to be the correct analogue for A. It is interesting that EXTENDER relies heavily on the identical symbols heuristic, while INITIAL_MAP (see Chapter 3–2) made no use of it at all, relying on the semantic type heuristic instead. The overall problem with both procedures is too much reliance on weak heuristics.

5. If A is

$$x \cap (y \cup z) = (x \cap y) \cup (x \cap z),$$

α contains

$$(\cap, \cup, id), (\cup, \cap, id),$$

and B is

$$x \cup (y \cap z) = (x \cup y) \cap (x \cup z),$$

then $descr(A) = descr(B) = descr(A)^{\alpha} = [pos(=), pos(\cap), pos(\cup)]$; so Kling's procedure, based on the abstracted descriptions, has no way of

---

[8]This point does not apply to Kling's own system, since it only uses the identity of analogous axioms, and not their structure; but it will apply to the application systems described in chapter 6.

distinguishing between A and B as analogues for A. Yet, given the initial mapping, it is clear that B is the analogue of A (in fact $B = A^{\alpha}$, according to the ACR defined in the last section).

Some of the problems listed above are caused by the particular filter and metric which Kling uses: 1 and 2 are clearly of this type; 4 is also partly, since the filter could be changed so as not to bias towards identical associations. However, some of the problems indicate limitations of the reliance on abstracted descriptions, which take no account of the *structure* of the axioms: it is the structure which indicates argument pairings (3), which causes non-identical associations to be added to the mapping (4), and which enables us to distinguish between axioms which contain the same symbols (5).

Conclusions Kling's EXTENDER is clearly deficient in two respects: firstly, his metric for chosing possible analogous axioms is unsatisfactory because it can cause many obviously unsuitable analogues to be found, particularly for small axioms, and leans too heavily on the identical symbols heuristic, causing the correct analogue often to be missed; secondly EXTENDER is based on abstracted descriptions of the clauses, and thus cannot take account of the structure of the clauses, which is important for analogy.

5.4.2 An improved AOI

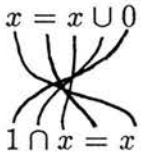
Value of structural match

Kling's EXTENDER routine seems to be based on the expectation that finding analogues for axioms should be easier than the initial analogy matching; i.e. that the knowledge of part of the analogy should make it easier to extend it further, and that the extension can be done solely on the basis of the abstracted descriptions of the axioms. However, we have seen that structural correspondence is relevant in deciding analogues for axioms. We summarise the reasons for this:

- The notion of structural similarity (the partial homomorphism heuristic) is crucial to the use of analogy in automated problem solving, as discussed in Chapter 3–2. In order to make use of this strong heuristic effectively, we need a structural match between base axioms and their potential analogues. It is possible to get an idea of whether two axioms might possibly match well together structurally by analysing their abstracted descriptions (see below), but this is only a weak use of the heuristic.

- A structural match enables the existing mapping to be extended in a principled way, with non-identical argument pairings if indicated. The use of the strong partial homomorphism heuristic would remove the need to rely on the weak identical symbols heuristic to propose extensions to the mapping.

- If we have a structural match between a pair of axioms, we can map over the inference position, at which the base axiom was used in the base proof, to the target axiom. For example, if we have the following match between axioms from Boolean algebra



  and we know that the upper axiom was applied by paramodulation from left to right in the base, we can predict that the lower one will be applied by paramodulation from right to left in the target.

Value of flexible filter

Kling's use of a filter based on abstracted descriptions has been seen to be both faulty in itself and limited in its discriminatory ability, being based on abstracted descriptions; while the latter point is true for any matching based on abstracted descriptions, a more flexible and less biased filter would nevertheless be a valuable part of an AOI: since some of the symbols in a base axiom will

probably already have been mapped, we can use their analogues to 'get a handle' on potential analogous axioms; this will enable us to cut down the number of candidates considerably out of the potentially very large knowledge base. Structural matching, as called for above, is bound to be more expensive than feature matching; without some way of pruning possible analogues, we would find that the AOI was extremely expensive.

However, we need a more flexible filter than Kling's, which does not, for example, allow inconsistencies to be introduced into the mapping; we would also want to remove the heavy reliance on the identical symbols heuristic.

Two-stage AOI

We are therefore led to a two-stage AOI:

1. A flexible filter, based on abstracted descriptions, is first called between a base axiom, and all clauses from the knowledge base; it returns a (short) ordered list of candidate analogues for the base axiom.

2. The candidate list is used to form the initial state of the matching and-agenda as described in Chapter 4; each partial match is started off with the existing mapping in its mapping slot, instead of the empty mapping. The analogy matching algorithm (described in chapter 4) is then called to produce an ordered list of completed matches. The candidate analogues involved in the matches, together with the matches themselves and their assessments, are returned as the result of the AOI. Each candidate has an associated extension to the mapping — those associations added in the course of matching, which were not in the initial mapping.

Of the two stages, the second is much more complex; however, the complexity involved is that which has already been discussed in the design of the matching algorithm, so there is not much more that needs to be said here. Note that the facility of the matcher to be provided with guidance in the form of expected associations is being made use of here; the existing mapping is put in the initial

mapping slot; this makes the matching faster and more accurate than it would be if the matching was done from scratch. It also has the virtue of guiding the AOI towards axioms which involve mappings which are consistent with the existing mapping; this is important, since we are looking for (relatively) consistent mappings throughout the analogy process. But we also do not rule out the introduction of inconsistencies into the mapping, if this is suggested by other analogy heuristics. The AOI inherits the flexibility of the matcher with respect to the heuristics.

The filter stage is important, for the reasons given above, and we next consider its design.

A flexible filter. We first change the definition of the description function so that repeated occurrences of symbols are repeated in the descriptions (thus $x \cup x = x$ no longer has the same description as $x \cup (y \cup z) = (x \cup y) \cup z$). We then define a similarity function between descriptions which takes account of the symbols which do not match as well as the ones which do.

Since we are dealing with an unrestricted syntax, the descriptions are defined to be three lists, one each for predicates, functions and constants. So if A is $x \cup 0 = x$ then

$$descr(A) = (\ [=],\ [\cup],\ [0]\ ).$$

(Unlike Kling, we do not include polarities in the description, although it would be simple to add them if it proved useful.) The similarity function $d(D_1, D_2)$ between two descriptions of this form is defined to be the sum of a similarity function f between individual components across the three components. The function f is defined below.

Suppose we have descriptions D_1 and D_2, and that D_1 is a mapped description; D_1 may thus contain default symbols, flagged by def, and optional symbols, flagged by $bdef$. Let these be $Defs$ and $Bdefs$ respectively, and let

$$Maps = D_1 \setminus (Defs \cup Bdefs)$$

be those symbols in D_1 which were positively mapped. Then

1. $f(D_1, D_2)$ should depend positively on $d_1(D_1, D_2) = |D_1 \cap D_2|$; i.e. the number of positively mapped symbols; note that default and optional symbols are excluded from this, since there are no defaults in D_2; this is an application of the **consistent translation heuristic** — we prefer axioms which may enable repeated associations.

2. $f(D_1, D_2)$ should depend negatively on the number of symbols in the descriptions which do not match. There are two qualifications to this. Firstly, default symbols should be excluded from the unmatched symbols, since they may match, without inconsistency, with symbols in D_2; furthermore, for each default symbol, two should be subtracted from the unmatched total, since the symbol with which a default might match should not be penalised either. Secondly, optional arguments should not count as unmatched, since they are optional; furthermore, for each optional symbol in $Bdefs$ which matches with a symbol in D_2, two should be subtracted from the unmatched total, since we can then consider the optional symbol to be included and to match with the other. These qualifications are designed to view the match as optimistically as possible. When all this is included, the expression obtained is

   $$d_2(D_1, D_2) = |D_1 \cup D_2| - |D_1 \cap D_2| - 2 \cdot |Defs| - |Bdefs| - |Bdefs' \cap D_2|$$

   where $Bdefs'$ is $Bdefs$ with the $bdef$ tags stripped off.

   This is also an application of the **consistent translation heuristic**, from the negative point of view — d_2 represents an estimate of the number of inconsistencies which would be introduced by matching with a particular axiom.

3. $f(D_1, D_2)$ should depend negatively on the difference in size between the descriptions. As above there is a qualification arising from $Bdefs$; since symbols in $Bdefs$ are optional, and we should be optimistic, we should retain just as many optional symbols in order to minimise the difference in sizes. Thus we define $d_3(D_1, D_2)$ as follows:

   $$|D_2| - |D_1| \quad \text{if} \quad |D_1| \leq |D_2|$$

$$|D_1| - |Bdefs| - |D_2| \quad \text{if} \quad |D_2| \leq |D_1| - |Bdefs|$$

$$0 \quad \text{otherwise}$$

This number represents a minimum on the number of unmapped symbols in an analogy match between the clauses giving rise to the descriptions. This is an application of the **partial homomorphism heuristic**: d_3 gives a lower bound on the number of unmatched symbols which would result from matching with a particular axiom.

4. $f(D_1, D_2)$ also depends positively on

$$|Defs' \cap D_2|$$

where $Defs'$ is $Defs$ with the def tags stripped out. That is, we reward default symbols which match symbols in D_2. This is an application of the **identical symbols** heuristic — we marginally prefer matches which promise identical associations.

All of the applications of analogy heuristics above are weak ones, even though some of the heuristics are strong; this is because the quantities can only estimate indirectly the prospects for a given heuristic being well satisfied by a match with the axiom. The filter which has been implemented has

$$f(D_1, D_2) = 3 \cdot d_1(D_1, D_2) - d_2(D_1, D_2) - 3 \cdot d_3(D_1, D_2) + d_4(D_1, D_2).$$

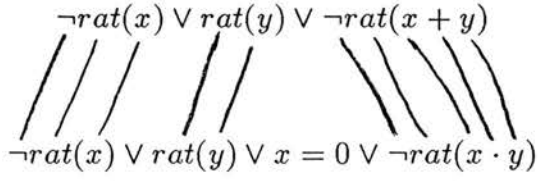
The weights attached to the four constituents reflect their relative importance.

The filter uses the similarity function just described together with a preset threshold — it finds the maximal similarity between the mapped description and the axioms in the knowledge base, and returns all axioms whose similarity with the mapped description differs from the maximum by less than the threshold. A small threshold will result in fewer axioms being returned, and thus in less work for stage 2, but risks missing plausible analogues. A large threshold has the reverse characteristics. 5 has proved a suitable threshold for the filter, and is used in the examples that follow.

Examples

We now illustrate the behaviour of the new AOI routine on some examples. For both of the example problems considered below, the same knowledge base was used, consisting of 59 axioms and theorems from real number theory and Boolean algebra. The axioms/theorems for the two domains were mixed in order to simulate a large knowledge base, where many different sorts of axioms and theorems could be applied.

Example 1 Given the following initial match between problems



we use the new AOI to find plausible analogues for the axioms used in the proof of the upper clause:

$$A_1\text{:}\quad \neg rat(x) \vee \neg rat(y) \vee rat(x - y)$$

$$A_2\text{:}\quad (x + y) - x = y$$

For A_1, the filter returns the following candidates, with associated scores:

$$A_1\text{:}\quad \neg rat(x) \vee \neg rat(y) \vee rat(x - y) \qquad (10)$$

$$A_3\text{:}\quad \neg rat(x) \vee \neg rat(y) \vee y = 0 \vee rat(x/y) \qquad (9)$$

$$A_4\text{:}\quad \neg rat(x) \vee rat(y) \vee \neg rat(x + y) \qquad (9)$$

The matching stage produces three matches:

Axioms	Strength	Map extension
$A_1 \leftrightarrow A_1$	0.929	$\{- \leftrightarrow -\}$
$A_1 \leftrightarrow A_3$	0.811	$\{- \leftrightarrow /\}$
$A_1 \leftrightarrow A_3$	0.811	$\{- \leftrightarrow /\}$

The difference between the two matches with A_3 is that the association $- \leftrightarrow /$ has arguments preserved in one and swapped in the other. In the context of the solution of the target, A_3 will turn out to be the correct analogue (see Chapter 6); the match with A_1 has greater strength than that with A_3 because it is a better structural match, and it has more identical symbols. However, the match with A_3 is still returned as a plausible analogue; Kling's EXTENDER returns A_1 as the only analogue for itself, and would thus be led into trouble on this example.

For A_2, the filter returns the following 9 candidates:

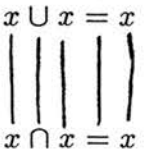
$$
\begin{array}{lll}
A_5\text{:} & x \cdot y = y \cdot x & (5) \\
A_6\text{:} & x \cdot y = x \cdot y & (5) \\
A_7\text{:} & \neg x \mid y \vee y = x \cdot h(x,y) & (4) \\
A_8\text{:} & x^2 = x \cdot x & (4) \\
A_9\text{:} & x = 0 \vee (x \cdot y)/x = y & (4) \\
A_{10}\text{:} & x \cdot 1 = x & (2) \\
A_{11}\text{:} & x \cap \overline{x} = 0 & (1) \\
A_2\text{:} & (x + y) - x = y & (1)
\end{array}
$$

Given these candidates, the matching stage produces 3 matches:

Axioms	Strength	Map extension
$A_2 \leftrightarrow A_2$	0.867	$\{+ \leftrightarrow +, - \leftrightarrow -\}$
$A_2 \leftrightarrow A_9$	0.833	$\{- \leftrightarrow /\}$
$A_2 \leftrightarrow A_{10}$	0.751	$\{(x + y) - x \leftrightarrow x \cdot 1\}$

The first of the matches is marginally preferred because the inconsistency which it entails ($+ \leftrightarrow \cdot$ previously, and $+ \leftrightarrow -$ now) is outweighed by the unmapped symbols in the other match. In the context of the target problem, the second match will turn out to be correct (see Chapter 6). Kling's EXTENDER returns 11 possible analogues from the knowledge base, including the correct one.

Example 2 Given the following initial match between problems,



we use the new AOI to find plausible analogues for the following axioms from the proof of the upper clause (see Chapter 6 for the full base proof):

$$B_1\text{:}\quad x \cup 0 = x$$

$$B_2\text{:}\quad x \cap \overline{x} = 0$$

$$B_3\text{:}\quad x = x$$

For B_1, the filter returns the following candidates, with associated scores:

$$B_4\text{:}\quad x \cap 1 = x \quad (6)$$

$$B_2\text{:}\quad x \cap \overline{x} = 0 \quad (3)$$

$$B_5\text{:}\quad x \cap 0 = 0 \quad (3)$$

$$B_6\text{:}\quad x + 0 = x \quad (2)$$

$$B_1\text{:}\quad x \cup 0 = x \quad (2)$$

The matching stage produces three matches:

Axioms	Strength	Map extension
$B_1 \leftrightarrow B_5$	0.980	$\{0 \leftrightarrow x,\ x \leftrightarrow 0\}$
$B_1 \leftrightarrow B_4$	0.960	$\{0 \leftrightarrow 1\}$
$B_1 \leftrightarrow B_1$	0.867	$\{0 \leftrightarrow 0, \cup \leftrightarrow \cup\}$

The second of these turns out to be correct in the context of the target proof. The second is a unification modulo commutativity and a single function mismatch. Kling's EXTENDER returns B_2 and B_5 as the possible analogues; it is unable to find B_4, the correct analogue, because of its heavy reliance on the identical symbols heuristic — the association with B_4 introduces a new non-identical association $0 \leftrightarrow 1$ into the mapping.

For B_2, the filter returns the following 8 candidates, with associated scores:

$$B_2: \quad x \cap \overline{x} = 0 \qquad (6)$$

$$B_7: \quad x \cup \overline{x} = 1 \qquad (4)$$

$$B_8: \quad x^2 = x \cdot x \qquad (3)$$

$$B_6: \quad x + 0 = x \qquad (2)$$

$$B_9: \quad (x + y) - x = y \qquad (2)$$

$$B_{10}: \quad x \cap y = y \cap x \qquad (2)$$

$$B_1: \quad x \cup 0 = x \qquad (2)$$

$$B_4: \quad x \cap 1 = x \qquad (2)$$

The matching stage produces two matches:

Axioms	Strength	Map extension
$B_2 \leftrightarrow B_7$	0.967	$\{\cap \leftrightarrow \cup, - \leftrightarrow -, 0 \leftrightarrow 1\}$
$B_2 \leftrightarrow B_2$	0.880	$\{\cap \leftrightarrow \cap, - \leftrightarrow -, 0 \leftrightarrow 0\}$

The first of these will turn out to be correct; the second involves an inconsistency as to the analogue of $\cap$ from right to left. Kling's EXTENDER returns B_2 as the only analogue for itself, and thus would, once again, be led into trouble. The cause of the problem is again the reliance on the identical symbols heuristic.

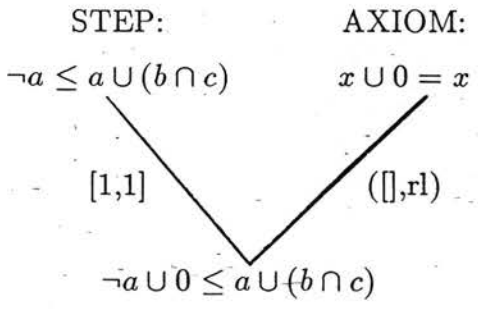
For B_3, the filter returns just one candidate, B_3 itself, which leads to a single match. Note that this was the example which led Kling's EXTENDER to propose 17 different analogues.

Conclusions The examples given above show how the two stages of the AOI combine to find analogues which match well according to the analogy heuristics and the initial mapping between the problems. The flexibility of both the filter and the matcher enable good analogues to be found, even if this involves extending the mapping with non-identical associations. The map extension is based on the same general principles as the initial map formation between problems, rather than special purpose procedures, as with Kling's EXTENDER. The examples given above are typical, and were not carefully chosen so that Kling's procedures failed on them.

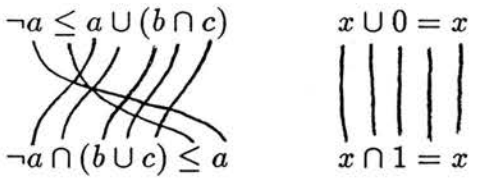
It would be quite possible to use each stage of the AOI without the other: we could try matching against all clauses in the knowledge base without first filtering; or we could just use the filter, and return the candidate list as the plausible analogues without using the matcher to distinguish between them. The problem with the first of these strategies would be that the AOI would be very expensive, and would become even more so as the knowledge base expanded. The problem with the second is that the AOI will not be so discriminating, and will not return structural correspondences; however, the latter problem will be mitigated by the fact that the candidates will be discriminated anyway when an attempt is made to apply them in plan application; the question is whether the extra cost of attempting and failing to apply bad analogues is greater than the cost of matching to find the promising ones beforehand. We cannot settle this question here, but will return to it again in Chapter 6.

5.5 Mapping Inference Positions

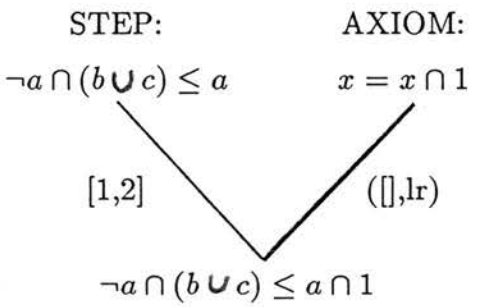
The ACR and AOI procedures described above are used to map the intermediate steps and axioms respectively of the base proof. However, important constituents of the base proof which are not covered by these procedures are the inference positions - i.e. the positions of the unified literals in their clauses in the case of binary resolution, and, in addition, the position of the rewritten subterm and the direction of application of the equality in the case of paramodulation. We can use an analogy to predict the positions of the corresponding inferences in the target proof - the analogous positions. For example, suppose we are considering the following base inference, an application of paramodulation:



This means that the subterm at position [1,1] in the step, a, is rewritten by the equation $x \cup 0 = x$ used from right to left. Suppose also that we have mapped the step and the axiom, using the ACR and AOI, in the following way:



Then, using the structural correspondences shown [9], obtained from the mapping procedures, we can map the entire step, including inference positions to:



The inference positions, [1,1] and ([],rl), are mapped via the structural correspondences shown; without the correspondences, the positions could not be mapped. The value of the mapped positions is well illustrated by the current example: there are six ways of applying the axiom $x = x \cap 1$ to the clause $\neg a \cap (b \cup c) \leq a$; we can use the mapped positions to try the indicated way first; in the context of a developing analogy, we have good grounds for expecting the indicated inference to be the one most likely to lead to success; if it does not, we can return and try other ways. If the positions were not mapped, we would not have grounds for choosing between the six possible inferences; over several successive steps, this could lead to an unmanageable search space, even assuming that the analogous axioms were the correct ones.

---

[9]Recall that the ACR, described in section 3, produces a structural correspondence between a term and its analogue as a side effect

5.6 Summary and Conclusions

In this chapter, we have discussed analogical plan formation; in particular, we have described procedures for mapping the various components of resolution proofs — goal clauses, intermediate steps, axioms and inference positions; while we have been concerned with resolution, the techniques described will apply to most forms of inference system.

The main procedures described are the analogue construction rule (ACR), for constructing the analogues of intermediate steps, and the analogous operator identification (AOI), for finding plausible analogues for axioms used in the base proof. We have seen that special syntax is useful, for the ACR, to represent uncertainty in the plan caused by symbols having no known analogues, or more than one known analogue. We have compared the ACR to Munyer's analogical inference rule (AIR); the latter procedure, while it appears to do the same kind of thing as the ACR, is too ambiguously and inconsistently described to be analysed in detail; however, we have seen that it is unable to represent uncertainty in a plan, which will lead to many plans being unnecessarily flawed.

We have described an AOI procedure which was motivated by the shortcomings of Kling's procedure EXTENDER, for finding analogues for base axioms; we have seen that the new AOI is a considerable improvement over Kling's procedures, both in the filter and the use of structural matching. The matching algorithm described in the previous chapter is used again in the AOI.

We have also considered the mapping of inference positions from base to target, and have seen the potential value of it, in terms of ordering the inferences which can be made at a step.

While the performance of the proof mapping procedures has been seen to be satisfactory on individual parts of certain proofs, their main validation comes from their co-ordination, along with the plan application procedures, to find

complete solutions to target problems. This validation is provided by the results of the next chapter.

Chapter 6

The application of analogical plans

In this chapter we investigate the use of analogical plans to guide the search within a resolution theorem proving system. The plans are constructed using the procedures described in the previous chapters, in particular the **analogue construction rule** *(ACR) and* **analogous operator identification** *routine (AOI). We consider various ways of coordinating the procedures in the construction and validation of analogical plans; the different approaches which can be taken lead to a family of analogy application systems which have been implemented. The performance of these systems is assessed on some examples of simple analogies; the systems are compared with each other, and with an unguided, breadth-first search theorem prover. The results demonstrate the ability of the application systems to exploit simple analogies — while the analogies are simple, in an intuitive sense, the target problems are hard, in the sense of involving large search spaces. The relative performance of the systems also enables us to draw some conclusions about the wisdom of the choices which were made in their design; these conclusions have a bearing on the assessment of existing analogy systems. We first describe the resolution system on which the analogy systems are based.*

6.1 A resolution/paramodulation system

All of the analogy systems which we will describe in this chapter make use of a resolution theorem proving system in order to validate analogical plans. See [Bundy 83] for an introduction to resolution theorem proving. The system can, in addition to binary resolution, apply the paramodulation rule of inference (essentially the replacing of equals by equals). The proofs which the analogy system accepts as base proofs and those which it produces as target proofs are thus resolution/paramodulation proofs.

The basic procedures of the system are those which apply the inference rules to a pair of clauses: the binary resolution procedure takes as input two clauses (skolemised disjunctions of literals; literals being either atoms or negated atoms) with a pair of literal positions; resolution of the clauses on the specified pair of literals is attempted; if successful the derived clause is returned as output. If the literal positions are unspecified, all possible ways of resolving the clauses are attempted, and the set of possible resolvents is returned as output.

The paramodulation procedure also takes a pair of clauses as input, with a pair of literal positions, a direction (left-to-right or right-to-left) for the first position, and a subterm position for the second; paramodulation of the second clause by the equality in the specified literal position of the first clause, in the specified direction at the specified literal and subterm position, is attempted; if successful the derived clause is returned as output. As with binary resolution, if the inference positions are not specified, paramodulation is attempted in all possible ways between the clauses, the set of derived clauses being returned as output.

The ability to specify the inference positions to the inference rules is important for the analogy systems, since they try, if possible, to make inferences in the target at analogous positions to those which occured in the base. As explained at the end of the previous chapter, this ability can considerably reduce the branching rate.

In addition to the basic inference rules, there is an unguided breadth-first search theorem prover which makes use of the inference rules. This theorem prover is based on the **set of support** strategy (see [Bundy 83]); the set of support consists of all clauses which are either goal clauses or which have goal clauses as ancestors in their derivations; inferences (by binary resolution or paramodulation) are made between a clause from the set of support and some other clause (whether on the set of support or not); i.e. axioms are never resolved or paramodulated with each other directly.

The breadth-first strategy works as follows: initially the depth of the search is zero and the goal clauses are assigned depth zero; at a particular depth, each clause from the set of support at that depth is resolved and paramodulated with all other clauses from the search space; any inferred clause which is subsumed by a previously inferred clause is deleted; similarly, any previously inferred clause which is subsumed by a newly inferred clause is removed from the search space; each inferred clause which is retained is assigned depth one greater than the current depth. This procedure is repeated for successively increasing depths until either the empty clause is derived, the machine runs out of space, or the program is stopped by the user.

In addition to the basic inference procedures and uniform theorem prover, there is a simple interactive proof construction routine; the purpose of this is to enable new proofs to be added to the data base of potential base proofs relatively easily; the representation for proofs used by the analogy systems is quite complicated — a proof is a list of nodes, each of which consists of a clause (an axiom, goal or intermediate step), pointers to the parent nodes in the proof, pointers to all daughter nodes in the proof, and inference positions for each daughter; the proof construction procedure takes care of the details. The user is prompted to input new clauses (goals or axioms) or to suggest an inference between existing clauses in the proof; if the user requests an inference he is prompted for the names of the parent clauses, the inference rule and the inference positions desired; given these, the system attempts the inference; if successful, the inferred clause is given a name, added to the proof, and all relevant information is added

to the existing nodes; if unsuccessful, the user is informed of the fact and asked to try again.

6.2 A framework for analogy application

The application systems which are described in this chapter are all based on the same framework for the application of analogical plans: the steps in the plan are verified successively, starting from the negated goals and continuing to the derivation of the empty clause in the base proof. Where there is more than one possibility at a position in the plan, for example if a base axiom has several candidate analogues, each possibility is tried in turn; if a possibility fails — either the axiom cannot be applied successfully, or, if it can, the plan fails at a later stage — another candidate is tried. Thus the strategy for plan execution is depth-first search through the various choice points in the plan. Note that the choices will, in general, have been ordered according to an estimate of their plausibility (by the strength of analogy matches, for example). The most plausible will be tried first. The ordering may be fixed when the plan is constructed, or derived dynamically as application proceeds. Thus, the strategy is something between depth-first and best-first search.

This simple framework will prove adequate for the close analogies which we consider in this chapter; in the next chapter, we consider how the framework will have to be improved to allow the successful application of more difficult analogies. The conclusions about the wisdom of the choices will still apply to the extended framework.

6.3 Issues in plan construction and application

6.3.1 What information should the plan contain ?

The tools so far described make the following information available for the plan for the solution to the target problem:

- Goal analogues: analogues for the goal clauses used in the base proof.
- Intermediate steps: analogues for steps in the base proof, produced by the ACR.
- Candidate axioms: sets of candidate analogues for axioms used in the base proof, produced by the AOI.
- The tree structure of the plan, including the inference rules applied (binary resolution or paramodulation) and the inference positions, as described above.

Existing analogy systems use plans involving different constituents: Kling's system ZORBA [Kling 71] only makes use of the axioms which were applied in the base proof; no attention is paid to the intermediate steps from the base proof, or to the structure of the base proof. The program synthesis system PRL [Constable & others 86] contains a small analogy component which also pays no attention to intermediate steps from the base — it uses axiom plans, like ZORBA, but also takes account of the structure of the base proof. Munyer [Munyer 81] describes two analogy application systems neither of which pays any attention to the axioms from the base proof; the analogical plans in these systems consist principally of intermediate steps, together with the structure of the base solution (apparently, no inference positions from the base are used, though).

So, previous authors have differed in their opinions about the nature of analogical plans, just as they have in the heuristics which their systems use; furthermore, as in the case of the matching algorithms (see Chapter 3), there has

been no explicit discussion of the relative usefulness of the various possible constituents of analogical plans. In this chapter, we initiate such a discussion. We consider the following possibilities for the composition of the plan:

- Axiom Plan. The plan contains a candidate set of analogues for each axiom which was used in the base proof, produced by a call to the AOI routine. Intermediate steps are not used at all. This means that only the inference positions in the axioms can be used, since those in the intermediate steps depend on the structural correspondence given by the ACR. In terms of the framework described above, local failure of the plan and backtracking is caused by the failure to apply any of the possible axioms at a step. That is, there are no intermediate steps which could be used to prune the search. However, the checking with intermediate steps (the special unification procedure described below) turns out to be an expensive process. So, while the search space may be larger with an axiom plan, it may nevertheless be searched more quickly. The (one-sided) inference positions can be used to order the possible inferences between an axiom and a step - that is, inferences (if any) at the specified axiom position can be tried first, others being tried on backtracking if these fail.

  As explained above, Kling's system ZORBA, and the analogy facility of PRL, both use axiom plans. Easy analogies tend to suggest axiom plans — solving the target problem merely requires applying the same, or similar, axioms to those used in the base, in the same order; checking with intermediate steps does not seem to be necessary. This view is criticised in the conclusion to this chapter.

- Step Plan. The plan contains an analogue for each of the intermediate steps which were passed through in the base proof, constructed by the ACR. No attention is paid to the axioms which were used. This means that only the inference positions in the intermediate steps can be mapped, since those in the axioms depend on the AOI. A step is verified by trying to make an inference with any axiom from the axiom base, and then checking that

the inferred clause matches with the next mapped step. Since the mapped step may contain some of the special symbols produced by the ACR, which have special meanings, the matching process needs to be an extension of ordinary unification, and is explained below. Thus, the intermediate steps are used to prune the search. Again, the (one-sided) inference positions are used to order the possible inferences between an axiom and an intermediate step, those at the specified position being tried first.

As explained above, Munyer's application systems make use of step plans.

- Full Plan. Both candidate axioms and intermediate steps are used, calling both the AOI and the ACR as appropriate. We can either use (a) the two-stage AOI or (b) just the filter stage, as suggested in the previous chapter. In the case of (a), all of the inference positions in the base proof can be mapped and included in the plan. In the case of (b), just the positions at the intermediate steps can be mapped, since the filter stage of the AOI alone does not produce structural correspondences between a base axiom and its potential analogues. A step is verified by attempting an inference with a candidate analogue, and, if successful, checking that the inferred clause matches with the next mapped step, using the special unification described below. Any inference positions which are available are used to order the inferences, as before.

6.3.2 Should the plan forming and validating procedures be separate or interleaved ?

- Separation. The entire plan is constructed before any validation is attempted. This type of system was assumed, for the sake of discussion, in the previous chapter. It is conceptually easier to grasp a plan which is entirely formed before it is validated. However, a great deal of time may be spent mapping the base proof only to find that the validation procedure fails on the first few steps. Kling's system ZORBA separates the construction and validation stages.

- Interleaving. Steps of the base proof are only mapped as they are needed for making target inferences. This type of system would have the benefit of failing relatively quickly on a bad analogy. Munyer's application systems interleave plan construction and validation.

A related issue is whether the mapping which is passed to the ACR and AOI should be extended in the course of plan construction/validation — extension of the mapping can come from either the matching of the AOI or the special unification of inferred clauses with intermediate steps (see below); using the extended mapping will, in general, make subsequent matching in the AOI and special unification quicker and more accurate. However, if the construction and validation processes are separated, different possible candidate analogues for axioms will, in general, lead to different extensions to the mapping; over the course of mapping the axioms in the base proof, this could lead to exponentially many extensions to the initial mapping. Therefore, in the separated systems which have been implemented, the initial mapping is used throughout the proof mapping process. On the other hand, if the construction and validation processes are interleaved, it is sensible to extend the mapping as they proceed; after backtracking, axioms and intermediate steps will have to be remapped anyway, so the most up to date mapping may as well be used. Therefore, in the interleaved systems which have been implemented, the mapping is extended at each step.

If a step plan, or a full plan of type (b), is chosen as answer to the first question, it seems advisable to interleave formation and verification of the plan rather than to have them separate; this is because the plan formation process, in this case, is essentially just the ACR and possibly the axiom filter, both of which are relatively cheap; thus it is no great cost to have to re-map the steps after backtracking; this enables the special unification to be called with as few special symbols in the mapped step as possible. If the whole plan is formed in advance, with the initial mapping, there will be many defaults throughout the plan, which will repeatedly slow down the special unification procedure, which is much more expensive when many special symbols are present.

	Separate	Interleaved
Axiom Plan	A	B
Full Plan (a)	C	D
Full Plan (b)	-	E
Step Plan	-	F

Figure 6–1: Implemented systems

6.3.3 Summary of possibilities

Figure 6–1 shows the various possible systems within the framework being discussed, with those which have been eliminated marked with dashes. The six systems which have been implemented are given letters so that they can be referred to in the discussion which follows. The various systems are compared with each other and with an unguided breadth-first search resolution procedure. There follow descriptions of two of the algorithms, A and D, to summarise how the choices made fit into the application framework which we are considering.

- **A: Axiom plan, construction/verification separate**

  1. Map base proof: call AOI on base axioms, to produce candidate sets, and map inference positions.
  2. Apply plan:

     - (a) find an active[1] node; if there is one go to (b). If there is no active node, and the nil clause has been derived, exit with success; if nil has not been derived, attempt to refute the clause which has been derived in place of nil, together with the original goal clauses, by breadth-first search.

[1] An active node is an node each of whose parents has either already been derived, is an axiom or is a goal clause.

- (b) attempt an inference between the current axiom in the candidate set of axioms and the other parent in the specified axiom position (if both or neither of the parents is an axiom the procedure is suitably modified). If successful, update the plan with the newly inferred clause and go to (a). If unsuccessful, or on backtracking, try to make an inference at any positions. If successful, update the plan and go to (a). If unsuccessful, or on backtracking, make the next axiom in the candidate list current and try again. If all candidates are unsuccessful, fail and backtrack to the previous step.

- **D: Full plan, interleaved construction/verification.**
  Initially, the plan is just the base proof, and the current mapping is the initial mapping.

  - (a) find an active node ..... (same as for A).
  - (b) map the parent clauses, inference positions and daughter clause, calling the ACR and the AOI as appropriate with the current mapping; try to make an inference at the specified positions (both axiom and step) between current axiom and the other mapped parent; check that the inferred clause matches the mapped daughter, calling the special unification procedure; if successful, update the plan, extend the mapping and go to (a); if not, or on backtracking, try the inference at any positions, again checking with the mapped daughter; if this fails, or on further backtracking, make the next candidate axiom current and try again. If the axiom set is exhausted, fail and backtrack to the previous step.

Each of the other systems implemented is a straightforward variant of one of the above.

In system D, stage (b), if the analogue of a base axiom is in the axiom base (ignoring default symbols), it is tried as the analogue first; the full AOI is only called on backtracking after subsequent failure. This is merely an efficiency

measure, and does not affect the functionality of the application system — the exact analogue would have been chosen first anyway, since it would match with base axiom with strength 1. Note that this efficiency measure is only open to the interleaving systems B, D and E.

6.3.4 Special Unification

As mentioned above, there is a special unification procedure to check whether an inferred clause matches with an intermediate step mapped by the ACR. Firstly, we explain the need for a special unification. Recall that the steps mapped by the ACR may contain the following special symbols:

- The function symbol def; $def(T)$ in a mapped step means that the head function symbol of T, f say, is not in the current match, i.e. f is a default symbol in the clause. When matching against an inferred clause, f should be allowed to match with any function symbol. If f matches with g, the association (f, g) can be added to the existing mapping. In this way the default is overwritten; in future calls to the ACR, occurrences of f will be replaced by g without the use of def.

- The function symbol $bdef$; $bdef(T)$ in a mapped step means that T is an "optional" argument. The step should be allowed to match with an inferred clause whether or not T is matched. For example the clause

$$[(a * b)/b = a, bdef(a = 0)]$$

should match with both

$$[(a * b)/b = a]$$

and

$$[(a * b)/b = a, a = 0].$$

A $bdef$ term is allowed to match with any number of terms on the other side (i.e. more than just one); furthermore, if the term contains variables,

these are not shared between different bindings. Such terms are produced by the ACR if symbols have had unmatched arguments in the past.

- The function symbols *poss* and *list*; these represent multiple possibilities arising from inconsistencies in the match (see the previous chapter for more details). They should be allowed to match if any of the possibilities would succeed. Thus

$$list([f(a,b), f(a,a), g(6)])$$

would match with

$$f(x,x)$$

by virtue of the second possibility. Similarly

$$poss([f,g],[a,b])$$

should match with

$$g(a,b).$$

A special unification procedure, which takes account of these possibilities, has been implemented. The structure of the procedure is the same as that of ordinary unification; the main difference is that, in addition to a list of variable bindings, a list containing information about the special syntax, if any, in the terms is maintained; this list records whether a default symbol has been overwritten yet, and, if so, what it is matched to; whether an optional argument has been included or omitted; whether a list of possibilities, either in the form *list* or *poss* has been restricted by the matching so far. In ordinary unification, whenever a variable is encountered, the variable binding list is consulted to see whether it has been bound; in the special unification, whenever a special syntax symbol is encountered, the special syntax list is consulted to see whether it has been overwritten, modified or omitted, as the case may be.

In addition to the above features, the procedure unifies modulo commutativity, for all known commutative functions and predicates; this enables minor patches to be made to the plan as a side effect of the unification. The procedure

produces a list of unifiers; each unifier consists of the usual variable substitution together with any extension made to the analogy by overwriting defaults. Whether or not the latter is made use of depends on choices made elsewhere in the particular application system.

6.4 Performance

The six systems were tested on seven examples, two from Boolean algebra, one from rational number theory and four from arithmetic.

6.4.1 Examples

Boolean algebra

The following axioms from Boolean algebra were used:

1. $x \cap 0 = 0$
2. $x \cap 1 = x$
3. $x \cup 0 = x$
4. $x \cup 1 = 1$
5. $x \cap (y \cap z) = (x \cap y) \cap z$
6. $x \cup (y \cup z) = (x \cup y) \cup z$
7. $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$
8. $x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$
9. $x \cap y = y \cap x$
10. $x \cup y = y \cup x$
11. $x \cap \overline{x} = 0$
12. $x \cup \overline{x} = 1$
13. $\neg x \leq y \vee \neg y \leq x \vee x = y$
14. $x \leq x \cup y$
15. $x \cap y \leq x$
16. $x \leq y \cap z \vee \neg x \leq y \vee \neg x \leq z$
17. $\neg y \leq x \vee \neg z \leq x \vee y \cup z \leq x$
18. $x \leq x$

together with the reflexivity of equality. 1-12 are standard Boolean algebra axioms; 13-17 relate the partial order in the algebra, $\leq$, to the lattice operations. The two examples used were:

- Problem 1. Prove $x \cup x = x$ by analogy to a given (non-standard) proof of $x \cap x = x$.

- Problem 2. Prove $x \cap x = x$ by analogy to a given (standard) proof of $x \cup x = x$.

The standard proof uses the standard Boolean algebra axioms; the non-standard proof uses the extra axioms 13-17. While the goals are essentially the same, the given proofs are different.

Rational Number Theory

The single example used was

- Problem 3. Prove $\neg rational(x) \vee rational(y) \vee x = 0 \vee \neg rational(x \cdot y)$ by analogy to a given proof of $\neg rational(x) \vee rational(y) \vee \neg rational(x + y)$.

A large axiom base was used, consisting of 45 axioms, some relevant to the theory, others not. This was done in order to test the systems on a large base, where the branching rate for a resolution system would be very high.

Arithmetic

The following axioms for arithmetic were used:

1. $x + y = y + x$
2. $x \cdot y = y \cdot x$
3. $x + y + z = x + (y + z)$
4. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
5. $x + 0 = x$
6. $x \cdot 1 = x$
7. $x + min(x) = 0$
8. $x \cdot inv(x) = 1 \vee x = 0$
9. $x - y = x + min(y)$
10. $x/y = x \cdot inv(y) \vee y = 0$
11. $x \cdot y = x \cdot y$
12. $x + y = x + y$

together with reflexivity for equality; min/1 represents unary minus, to distinguish it from binary subtraction; similarly inv/1 represents inversion. The problems set were:

- Problem 4. Prove $x/x = 1 \vee x = 0$ by analogy to a given proof of $x - x = 0$.
- Problem 5. Prove $(x \cdot y)/x = y \vee x = 0$ by analogy to a given proof of $(x + y) - x = y$.
- Problem 6. Prove $\neg inv(x) = inv(y) \vee x = y \vee x = 0 \vee y = 0$ by analogy to a given proof of $\neg min(x) = min(y) \vee x = y$.
- Problem 7. Prove $\neg inv(x) = inv(y) \vee x = y \vee x = 0 \vee y = 0$ by analogy to another proof of $\neg min(x) = min(y) \vee x = y$.

6.4.2 Results

The following table gives times for the solution of the various problems by the various systems. "fail" means that the program either ran out of space, or was interrupted having got irreperably bogged down. The times refer to compiled Quintus Prolog on a Sun 3 workstation.

Problem	A	B	C	D	E	F
1	42s	26s	48s	27s	11s	14s
2	1m29s	1m50s	1m1s	47s	17s	2m11s
3	1m11s	2m5s	1m19s	2m7s	42s	1m37s
4	23s	17s	25s	20s	9s	25s
5	fail	fail	45s	27s	23s	57s
6	fail	fail	1m55s	1m3s	57s	30m
7	fail	fail	3m45s	2m16s	2m33s	fail

- Problem 1. All the programs find the same proof for the target, without any backtracking, ie the plan suggests the right inference in every case. Figure 6–3 shows the base and target proofs[2] (Figure 6–2 explains the notation used in all of the application diagrams which follow). This example, and problem 2, are examples of 'perfect' analogies, where the target and base proofs are isomorphic, differing only in the identity of the symbols in the goals, axioms and steps. It is important for analogy systems to be able to solve easy analogies like this one; if they cannot, their claims on harder analogies become suspect; we argue below that previous analogy systems would not be able to solve problems such as 1 and 2.

  The axiom plan systems (A and B) find the proof faster than the corresponding full plan systems (C and D); the latter are slowed up by the checking with mapped steps, which is not necessary in this case since the right inferences are always suggested anyway. The full plan (b) (E) system is the quickest of all; this is because the axiom filter which it uses is successful in reducing possibilities, and the extra work which other systems do to match axioms fully is not necessary. The step plan system (F) is also quick; the cost of trying to apply axioms and matching the results with the

[2]The labels on the steps in the proof show the number of the axiom which was used, and the inference rule which was used ('binres' for binary resolution and 'paramod' for paramodulation). The plan steps are not shown in the figures.

intermediate steps turns out to be less than the cost of the full matching within AOI.

Also, the interleaving systems (B and D) are quicker than the corresponding separate systems (A and C); their use of the extended match pays off, and there is no backtracking which would force them to re-match axioms.

- Problem 2. Again, all programs succeed and find the same proof for the target. The full plan systems do so without backtracking, but the axiom plan systems (A and B) and the step plan system (F) do have to backtrack. The extra search made by A, B and F is shown with dotted lines in Figure 6–4; dead ends in the search are represented by triangles; for F, the extra search is caused by the application of axiom 4 at the second step, which matches with the corresponding plan step (not shown in the figure) but leads to a dead end; for A and B, the search arises because there are four ways to apply axiom 11, in the specified position, at the fourth step, the third of which leads to the solution; the first two inferences lead to dead ends after one more step. This extra search is responsible for the slower times for A and B; the interleaving system, B, is slowed down more by the extra search than the separate system A, since it involves re-matching axiom 2 for each branch.

  For the same reasons as before the full plan (b) system (E) is quickest of all, while the step plan system (F) is slowest — in this case much time is spent trying to apply the wrong axioms.

- Problem 3. Again, all programs succeed and find the same proof (see Figure 6–5). However, all require some backtracking to do so; in all cases, axiom 1 is tried first at the first step, and leads to failure; the full plan and step plan systems fail after two steps, since the inferred clauses do not match with the plan; the axiom plan systems do some extra search since they do not have the intermediate steps to prune it; this leads to dead ends in each case after one more step. System A, which does not extend the mapping as application proceeds, makes a similar mistake again at the

third step (preferring the association $- \leftrightarrow -$ to $- \leftrightarrow /$), and does some extra search, applying axiom 2 in seven ways before backtracking onto the solution path.

The backtracking causes the interleaving systems to be slower than the corresponding separate systems on this example, the cost of re-matching axiom 2 outweighing the advantage of using the extended match.

Again the full plan (b) system is quickest of all; the lack of inference positions in the axioms does not lead to any extra search.

This example is somewhat more difficult than the previous ones, in that the target and base proofs are not isomorphic — some clauses in the target have extra conditions, and the target proof contains extra steps; the last two steps in the target are made by breadth-first search after the plan has been exhausted without the empty clause being derived.

- Problem 4. This problem turns out to be very easy for all the systems, which find the proof (not shown) with no backtracking. The relative speeds are similar, with similar reasons, to those for Problem 1.

- Problem 5. The axiom plan systems both fail on this example, becoming bogged down in fruitless inferences. The other systems succeed, finding the proof shown with unbroken lines in Figure 6–6; the axiom plan systems make the wrong inference (shown with dotted lines) at the second step, get bogged down in the resulting search and so do not manage to backtrack to the second step and correct the inference. The wrong inference is made because of the lack of the inference position in the step, which is what enables the other systems to avoid the problem.

  For the full plan (a) systems, the interleaving one (D) is quicker than the separate one (C), since there is no backtracking to offset the advantage of using the extended match. Again, the full plan (b) system is quickest of all, for the same reasons as before.

This example, and the two which follow, are similar to problem 3, in that the proofs are not isomorhpic, the plan being patched by search at the end; they turn out to be more difficult because the choice of the wrong axiom can have more severe consequences in a long proof containing many applications of paramodulation (particularly for the axiom plan systems).

- Problem 6. The axiom plan systems again fail on this example, being unable to recover from the application of the wrong axiom at the first step. The other systems make the same mistake, but are able to prune the fruitless search by the special unification with plan steps. Systems C and F both do some more extra search later on in the solution, shown with dotted lines in figure 6–7; the use of the extended match in the AOI enables systems D and E to avoid the extra search.

  Again, the interleaving systems prove to be much quicker on this more difficult example. The step plan system (F) is extremely slow; it is slowed down particularly on steps which contain variables, as many clauses paramodulate with these.

- Problem 7. The full plan systems are the only ones able to solve this problem. The axiom plan systems cannot recover from applying the wrong axiom at the first step (as all the systems do), getting bogged down again in fruitless search, and unable to prune it with the plan steps. The step plan system (F) does not recover from making the wrong inference at the fourth step. Figure 6–8 indicates the search spaces.

  Of the successful full plan (a) systems, the interleaving one, D, is much quicker; this is because, having backtracked to the first step and corrected the inference, no further backtracking is required - the use of the extended mapping means that the next application of axiom 12 in the base proof is mapped correctly first time; whereas the separate system, C, makes the same mistake again with the next application of axiom 12, and a similar mistake with axiom 1, each of which leads to some extra search.

1. Steps in complete proofs are indicated by clauses joined with unbroken lines:

$$\neg a \cup a = a$$
$$\Big|$$
13,binres
$$\Big|$$
$$\neg a \cup a \leq a \vee \neg a \leq a \cup a$$

The label indicates that axiom 13 was applied using binary resolution.

2. Search steps not part of a completed proof are indicated with dashed lines:

$$\neg(a \cdot b) \cdot inv(a) = b \vee a = 0$$
$$\vdots$$
A,B:2,paramod
$$\vdots$$
$$\circ$$

In addition to the axiom and inference rule, the label on a search step indicates which of the implemented systems made the step (in this case, A and B). Where the systems are not indicated, they are the same as on the previous step. For search steps, the axiom number and/or inference rule may be omitted from the label.

3. Dead ends in the search are indicated by the symbol $\triangle$.

4. Failed searches are indicated by the symbol $\bowtie$.

Figure 6–2: Key to following figures

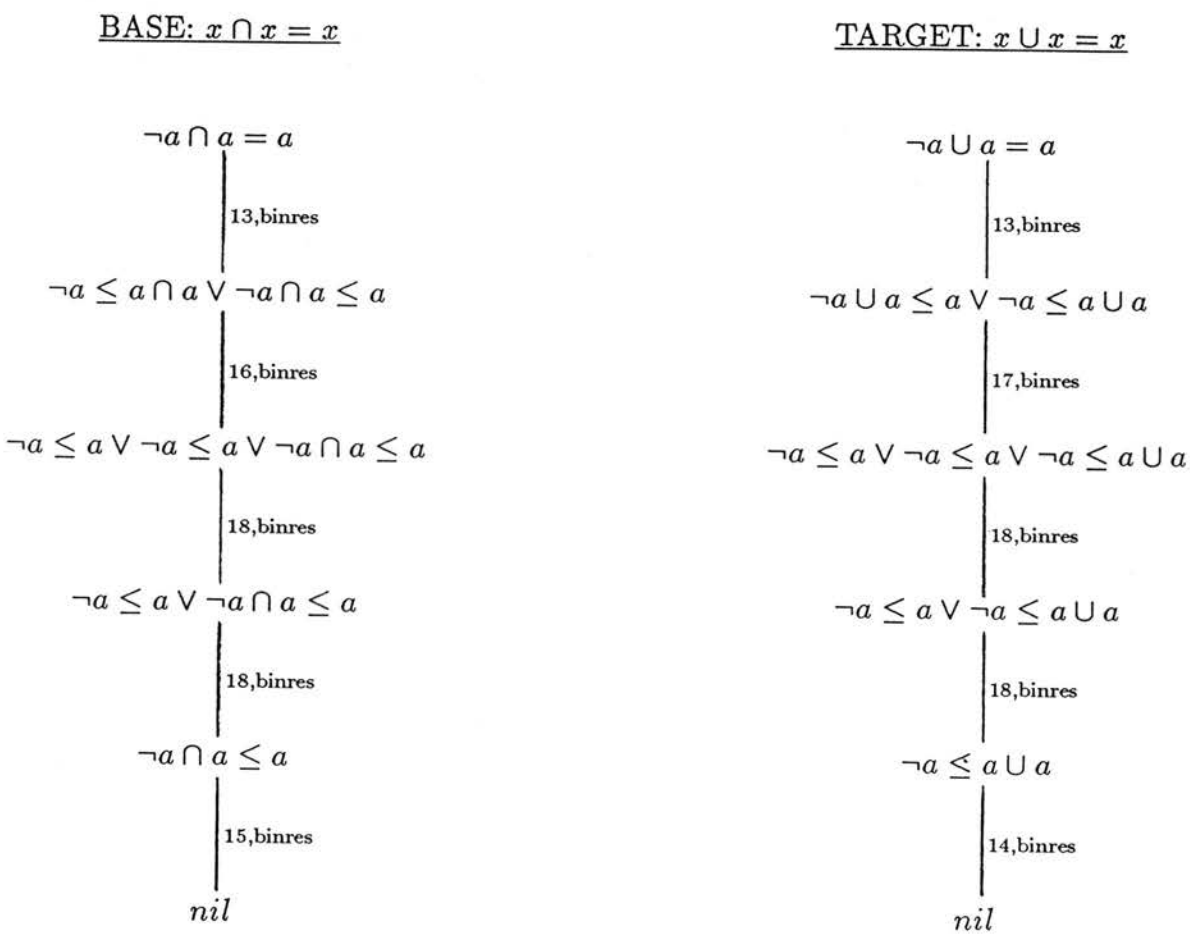


Figure 6–3: PROBLEM 1

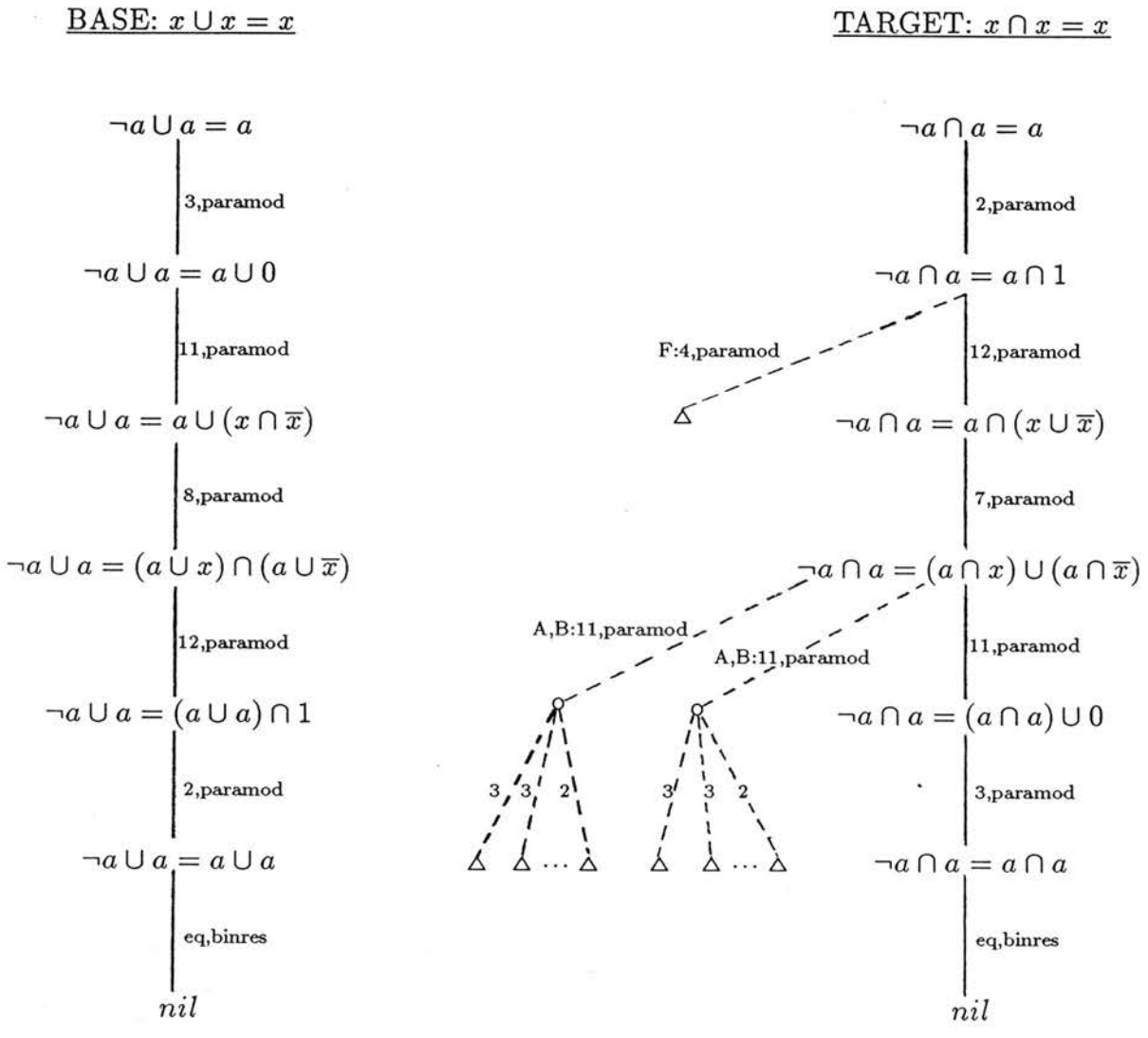
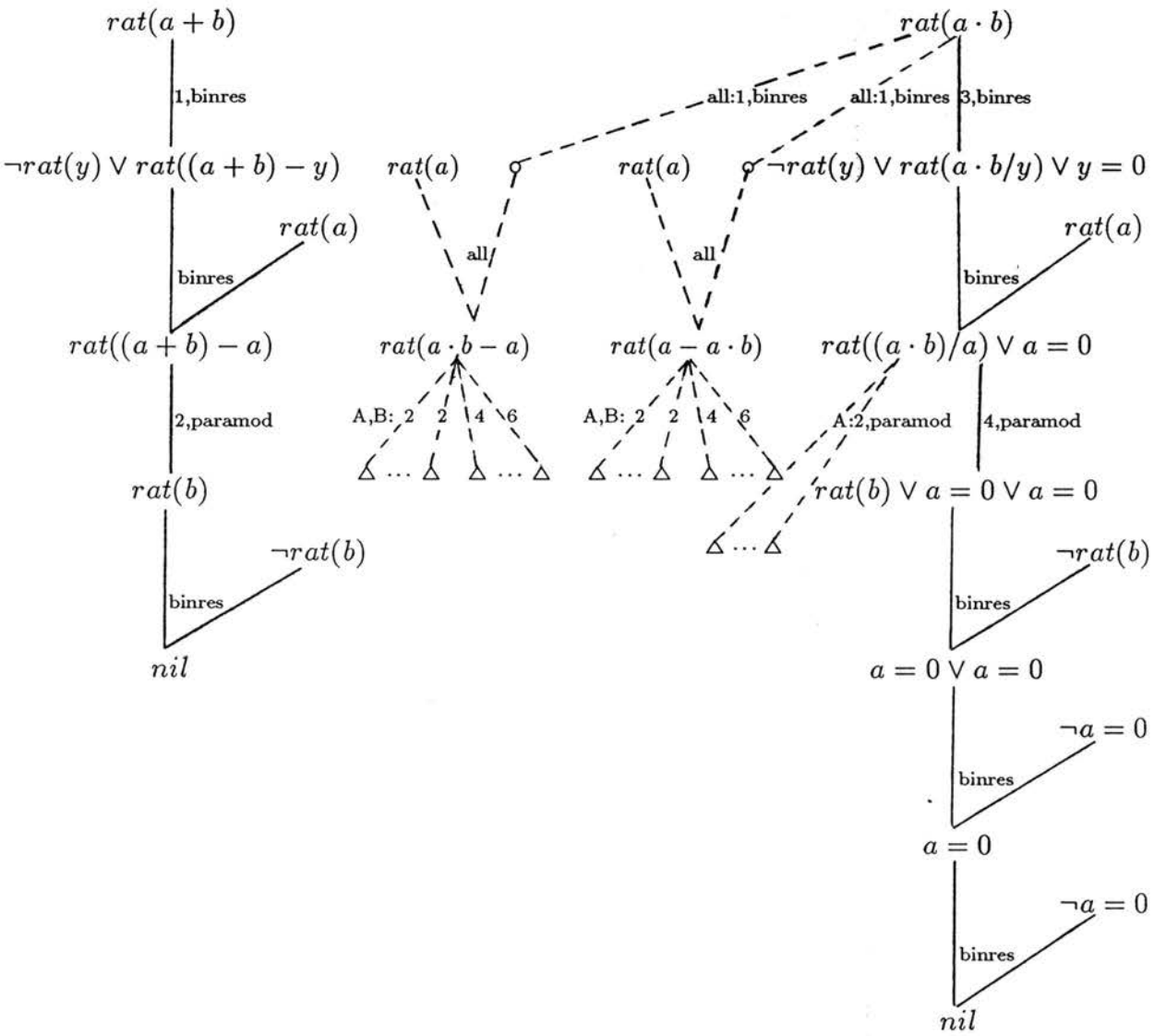


Figure 6–4: PROBLEM 2

BASE: $\neg rat(x) \vee rat(y) \vee \neg rat(x+y)$ TARGET: $\neg rat(x) \vee rat(y) \vee x = 0 \vee \neg rat(x \cdot y)$



1. $\neg rat(x) \vee \neg rat(y) \vee rat(x-y)$
2. $(x+y) - x = y$

1. $\neg rat(x) \vee \neg rat(y) \vee rat(x-y)$
2. $(x+y) - x = y$

Figure 6–5: PROBLEM 3

6.4.3 Discussion

Generally speaking, the analogy systems handled these simple analogy problems quite well. We constrast their performance with that of the unguided breadth-first search theorem prover in the next section. We firstly consider the relative performance of the various systems.

Full vs step vs axiom plans The most clear answer to any of the issues raised earlier is that the full plan systems were superior to the axiom plan systems and step plan systems.

Their superiority over the step plan system is simply accounted for by the pruning of search enabled by the AOI. Although the special unification with intermediate steps enables the step plan system F to prune the search, the preceding diagrams do not show the many attempts which F makes to apply all possible axioms, which are blocked by the special unification; this accounts for the relative slowness of F, even when little extra search is shown on the diagrams.

The axiom plan systems work well for the easy examples, but their performance deteriorates rapidly on the more difficult examples. The full plan (a) systems are more expensive on the simple examples, but prove much more powerful on the more difficult examples. There seem to be two main factors in the shortcomings of the axiom plan systems:

- The lack of inference positions in the steps leads to a higher branching rate for the axiom plan systems than for the others; the full plan systems benefit from being able to map inference positions in both axioms and intermediate steps; the lack of axiom positions in the step plan system turns out to be less important than the lack of step positions in the axiom plan system. This is because paramodulation, which is mainly responsible for the high branching rate, is usually applied with the equality as axiom and the intermediate step as the rewritten clause. The branching rate usually arises from being able to apply the equality at various different subterms of the rewritten clause; thus is it particularly useful to be able to

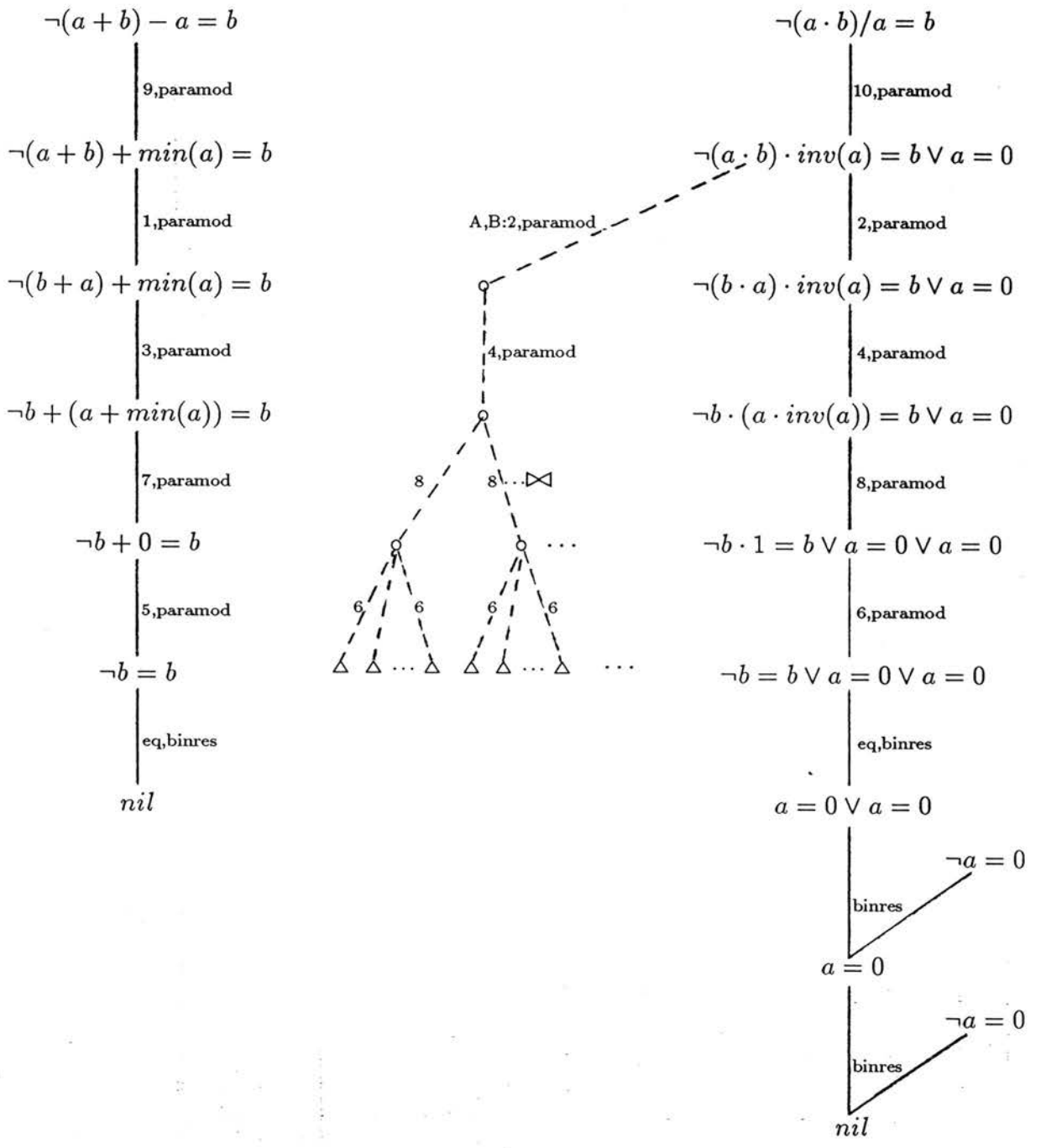


Figure 6–6: PROBLEM 5

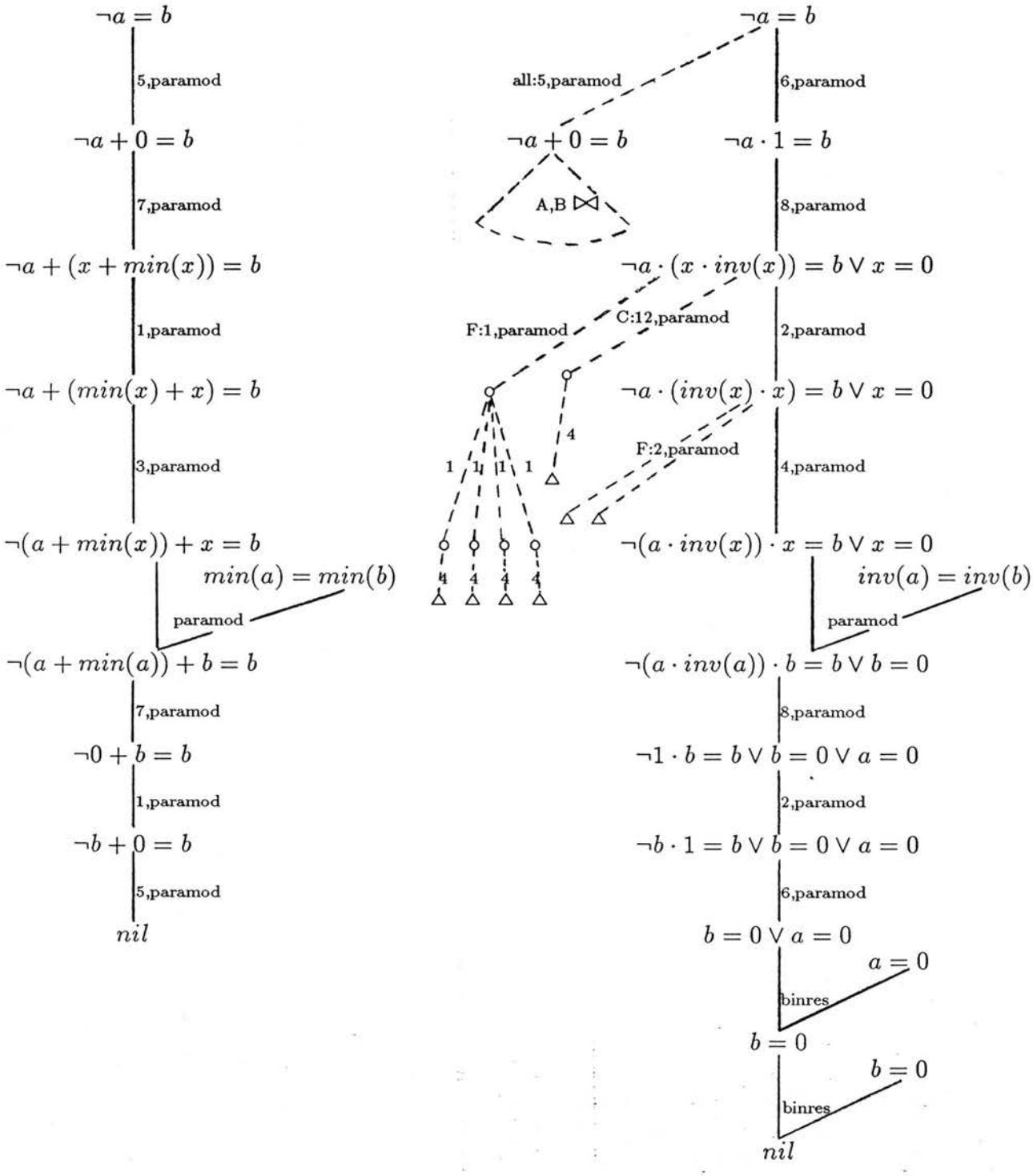


Figure 6–7: PROBLEM 6

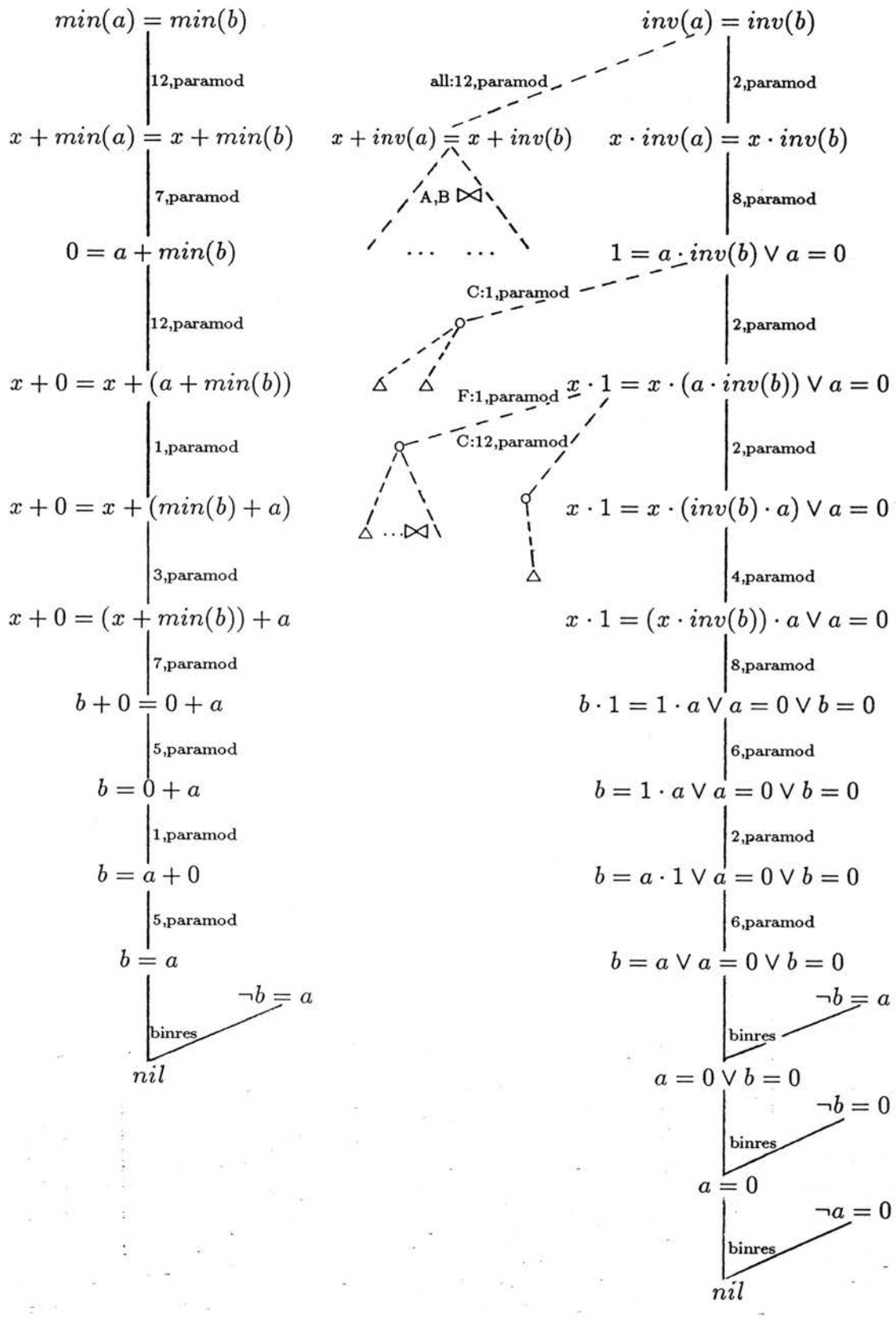


Figure 6–8: PROBLEM 7

map the intermediate step inference positions in order to specify a preferred rewriting.

- The axiom plan systems cannot use intermediate plan steps to reject certain inferences and so reduce the branching rate, as the other systems can. This inability to prune the search is what leads to the failure of the axiom plan systems on some of the examples. In the full plan and step plan systems, the cost of matching inferred clauses with plan steps, using the special unification procedure, turns out to be easily outweighed by the benefit of pruning the search which it brings.

Separation vs interleaving The results indicate that the extra matching involved in interleaving pays off, in terms of avoiding extra search, and quicker axiom matching and special unification. Of full plan (a) systems, the interleaving one, D, performs better, on the whole, than the separate one, C. The extra cost of re-matching axioms after backtracking is outweighed by the advantages of using the extended match; the extended match not only makes matching with plan steps more efficient, but can avoid the need to backtrack by dynamically ordering the candidate analogues according to their consistency with the **current** match. The separate system, which computes analogues for all the axioms before any inference, can perform extra backtracking. This is most evident in Problem 6, in which, as Figure 7 shows, system C makes two further incorrect inferences which system D is able to avoid by learning from its earlier mistake.

On most of the problems, the full plan (b) system (E) is the most efficient; the extra time spent in matching candidate inferences with mapped steps, using the special unification, is outweighed by the time spent by the full plan (a) system in calling the full analogy matching algorithm to distinguish further among the candidate analogues for axioms. On the more difficult examples (5, 6 and 7), there is not much difference between D and E. We cannot conclude from these results whether or not the two-stage AOI is preferable to the one-stage version; with the full plan (b) system, the special unification is effectively providing the second stage of the AOI; on more difficult examples, which require patching, (see

the next chapter), it may be useful to have strong information about analogues for the axioms independently of our *immediate* ability to apply them successfully — the next plan step may be misleading, for example. In situations such as this, the two-stage AOI may well be superior to the one-stage version.

6.4.4 Comparison with unguided search and existing analogy systems

The same problems were also given to the unguided breadth-first search theorem prover described earlier; it was unable to solve any of them within 30 minutes. Furthermore, on only one of the problems (problem 1) had the prover managed to search beyond depth 2; since the proofs for the seven problems lie at depth at least 5, there was no point in letting the prover run longer. The average branching rate (including ancestor resolution and paramodulation) for problem 1 was around 30 (817 mutually non-subsuming clauses were derived at depth 2); assuming that this branching rate is maintained at lower levels, the search space at level 5 would have size approximately $2 \cdot 10^7$.

For problem 3, with a large axiom base, the branching rate was about 50 for the first level, and over 500 for the next level (the increase in branching rate is caused by the introduction of variables into inferred clauses, which causes them to paramodulate with many other clauses).

For the arithmetic problems, the average branching rate turned out to be around 50, which gives an estimated size for the search space to depth 10 (as in problem 6) of 10^{17}.

So it is clear that the analogy application systems are effective in guiding search in otherwise explosive spaces. The explosive nature of the search is indicated by the fact that the axiom plan systems are sometimes unable to recover from a single incorrect inference. While the analogies involved are intuitively close, the base and target proofs are significantly different, involving application of different axioms, and sometimes different numbers of steps.

Moreover, existing analogy systems would not be able to apply these analogies: Kling's system ZORBA, apart from the weaknesses of its matching, forms unstructured axiom plans; the branching rate for these will be high even if the right analogues for the base axioms are found — certainly much higher than for the structured axiom plan systems considered above. Munyer's planning methods make heavy use of his analogy matching algorithm, discussed in Chapter 3; this algorithm has been seen to be inadequate for the type of analogies considered in this chapter, which involve many non-identical associations; as a result, Munyer's systems would not be able even to get started on these examples; moreover, Munyer's explicit planning method, which, although not implemented, he regards as an improvement on his implicit planning method, uses the **analogical inference rule** (AIR), discussed in the previous chapter, to construct its plans; on the examples in this chapter, many symbols are introduced into the proof which do not occur in the original problem statements; as explained in the previous chapter, the AIR maps these symbols over identically into the plan; this will cause the plan to contain many unnecessary flaws and require it to be patched; in corresponding situations, the step plan and full plan systems described above would use the default symbol to represent the uncertainty, allowing the special unification to succeed and extend the mapping.

6.5 Conclusion

In this chapter, we have seen how the techniques developed in earlier chapters can be used to guide search effectively within a resolution theorem proving system. We have seen the explosive nature of unguided breadth-first search within the domains we have considered, and have illustrated the dramatically reduced search spaces involved in the application of analogies.

We have also compared the effectiveness of different approaches to the formation and validation of analogical plans; we have concluded that the full plan systems, which make use of both the axioms and intermediate steps from the

base proof, are more effective than either the axiom plan or the step plan systems. Furthermore, the extra work involved in interleaving the construction and validation of plans has proved to be worthwhile. We have not considered, in this chapter, any examples which, while intuitively close analogies, cannot be solved within the simple framework we have considered; we consider some such examples in the next chapter, and discuss how the application framework will have to be extended to cope with them. The conclusions we have drawn will also apply to the extended framework, i.e. the superiority of the full plan and interleaving systems does not depend on the simplicity of the examples which we have considered so far.

Chapter 7

Further Work

In this chapter we propose and discuss some extensions and improvements to the analogy systems which have been described in the previous three chapters. In particular, we consider the patching of analogical plans and how an analogy system can become better at finding and using analogies over its experience. These extensions have not yet been implemented, and so are discussed as further work.

7.1 Introduction

We will discuss two directions for improvement of the analogy systems described in the previous chapters: the first (section 7.2) is towards more sophisticated plan application systems than were described in the previous chapter; the second (section 7.3) towards an analogy system which can improve its own performance over the course of its analogical problem solving experience. These directions for improvement are independent, in that we could pursue one without pursuing the other, but complementary.

In the next section we consider the **patching** of analogical plans. The motivation for this is that the analogy systems described in the previous chapter can only cope with fairly straightforward analogies. Many matches which are constructed by the matcher described in Chapter 4, while intuitively good analogies,

lead to plans which cannot be applied by the present application systems. We will define a notion of **correctness** for analogical plans, and will see that the present application systems are unlikely to be able to apply plans which are not correct. The purpose of the patching techniques discussed in the next section is to enable the application system to apply incorrect (or **flawed**) plans.

In the following section, we consider how an analogical problem solver could improve its performance by learning and exploiting global analogies (or **dualities**) and semantic types. The motivation for this is two-fold: firstly, the analogy systems so far described would be unable to benefit from a previous successful attempt at analogy when faced with a new analogy, even if the two situations involved essentially the same analogy; that is, the analogy systems exhibit an inability to learn from their analogical problem solving (APS) experience, just as theorem provers cannot learn from successful proofs when faced with similar problems. Secondly, the systems described so far fit into the basic APS framework introduced in Chapter 1. While this is one approach to analogy, and a natural one, it is not the only one. It would be desirable for an analogy system to be able to find and use analogies in a variety of ways. In section 7.3.7, we see how the knowledge of global analogies suggests new ways of using analogy.

7.2 The patching of analogical plans

The purpose of this section is to consider how the analogy application systems discussed in the previous chapter will have to be improved to cope with a wider range of analogies. The limitations of the present systems are caused by the fact that they are unable to apply *flawed* plans successfully, except in certain restricted cases. We first define what we mean by a flawed plan, in the context of the analogical plan language which we are using, and explain how the present systems fail on most such plans (and how they succeed on a few of them).

We then look more closely at particular examples of flawed plans, and what an application system would have to do to cope with them.

7.2.1 Flawed analogical plans

Firstly, we review the structure of the analogical plans which are constructed by the plan construction procedures described in Chapter 5. An analogical plan is a binary tree structure[1] isomorphic to the base proof from which it was derived. Just as a base proof tree contains three types of nodes, **goal** nodes, **axiom** nodes and **intermediate step** nodes, an analogical plan contains corresponding types of nodes; however, where an axiom node was labelled by the single axiom that was applied in the base proof, an axiom node in the plan will be labelled by a *list* of candidate analogous axioms; where an intermediate step node from the base proof was labelled by a clause from the object language, the corresponding node in the plan will be labelled by a formula from the extended plan language described in Chapter 5; the special function symbols, *list*, *poss*, *def* and *bdef* represents different kinds of uncertainty in the plan (see Chapter 5).

We can define an analogical plan to be **correct** if there is a choice of axioms from the axiom nodes which allows each of the inferences suggested by the plan to go through, with the inferred clause matching the plan formula for the result, in the sense of the special unification procedure described in the last chapter, and the empty clause being derived at the last step. A plan is **flawed** if it is not correct.

To put it another way, we can think of the analogical plan as representing a large number of possible attempted proofs of the target problem: a plan is correct if one of these attempts is a valid proof of the target, and it is flawed otherwise.

The plan application strategy of the systems described in the previous chapter is to verify the plan steps successively, choosing axioms from the axiom sets, making inferences, and checking that the inferred clauses correspond to the plan

---

[1]Plans are binary trees because we are working in a binary resolution system. The techniques would extend to non-binary forms of resolution, and the corresponding analogical plan trees would no longer have to be binary.

according to the special unification procedure. Thus it seems that these simple strategies will never be able to apply flawed plans successfully. In fact, a certain type of flawed plan can be applied by the present systems: when all of the plan steps go through successfully, except that the final clause derived is not the empty clause, the system attempts to finish the target proof by refuting the remaining clause in a breadth-first search. Problems 3, 4, 5 and 6, from the previous chapter, involve flawed plans of this type which were applied successfully.

A more serious type of flaw is when a step in the plan will not go through for any choice of axioms at previous nodes, either because none of the candidate axioms will apply, or, if some do, the inferred clauses do not match with the next intermediate plan step. This type of flaw will cause failure in all the application systems of the previous chapter.

Of course, the plan might not be appropriate for the target problem at all, in which case it is right that it should fail; but many intuitively close analogies lead to plans which are flawed, in the strong sense (examples of these are given in the next section). As humans, we feel that these analogies provide useful guidance towards the solution of the target problem. Only fairly straightforward analogies lead to plans which are correct, in the sense defined above. Thus an analogy system which can handle more interesting and difficult analogies will have to be able to cope with flawed plans.

For a flawed plan to be validated, it must be corrected or **patched**. That is, inferences other than those suggested by the plan must be made at some stage during application. As the examples given in the next section will show, flaws in analogical plans are often "local" problems — that is, extra inferences can be made which will allow the plan to be picked up again after one or a few steps. If the flaw can be successfully patched, the straightforward application can take over again.

7.2.2 Example flawed plans and patches

The question facing the designer of an improved application system is "what to do when the next step won't go through ?". This happens if (a) the next indicated axiom will not apply to the step, or, if it does, (b) the inferred clause does not match with the next plan step. There are many possibilities that come to mind — try a different candidate axiom, backtrack to a previous step, try to add steps to make the axiom apply (if (a)), try to add steps to match with the plan step (if (b)), try to miss out plan steps, try a different plan altogether, try to develop some other part of the plan, and others. The plan need not necessarily be flawed for this situation to arise — it may be that a wrong choice of axiom has been made somewhere in the application. The first two possible actions mentioned above, choosing a different axiom and backtracking to a previous step, attempt to follow up this possibility. But these are the only actions which the present systems can try. An improved application system will not immediately know whether a local failure arises because the plan is flawed, or because a wrong choice has been made. It will not be feasible, in general, to search the space indicated by the plan exhaustively before deciding on what patching action to take. Therefore, the application system will have to consider the patching options along with the first two backtracking options.

The problem is which action to try first, and how to coordinate the investigation of the different possibilities. Before we consider the issues of control, we examine the possible causes of the local failure of a plan, assigning blame to particular parts of the plan, or choices made during its application. Each type of cause suggests a way of overcoming the local failure. We give examples of the different types.

- **Wrong Axiom.** Perhaps the wrong axiom has been chosen out of the candidate set produced by the AOI, either at the present step or at a previous step. The analogy search spaces illustrated in the previous chapter show cases where the axiom chosen first leads to a dead end, and it is

another axiom from the candidate set, less favoured initially, which, when applied, leads to the solution.

The cure for this problem is to try a different candidate axiom, either at the current step, or at a previous step.

- **Wrong Plan.** Perhaps the entire plan which is being followed is inappropriate for the target problem, and should be abandoned in favour of another. In a realistic analogy system, this is bound to happen often because of the empirical nature of analogy.

  The cure for this is to stop developing the current plan and start on (or pick up again) another plan.

The above are the only situations which the present analogy systems can deal with. The system is exploring the wrong branch of the analogy search space, and should backtrack to another branch. For the rest of the reasons for failure, we assume that the plan application is basically on the right track, in terms of the inferences that have been made so far. The failures are classified according to whether the step and the axiom suggested by the plan are "right" or "wrong"—that is, whether they turn out to form part of a solution to the target problem. These distinctions are not clear cut; for example, an axiom suggested by the plan may turn out to be useful in achieving the next step, but only along with the application of other axioms; this would somewhat dilute its rightness. Here is the classification:

- **Step right, axiom right.** The hypothesised axiom and plan step for the next inference are right (in the sense that they both occur in the intended patch), but extra steps need to be inserted before and/or after the axiom is applied in order to allow the step to go through. Figure 7–1 shows an example of a plan step which occurs in the attempt to prove the additive closure of the binary rationals by analogy to that of the ordinary rationals, i.e.

$$binrat(x) \wedge binrat(y) \longrightarrow binrat(x + y)$$

by analogy to

$$rat(x) \wedge rat(y) \longrightarrow rat(x + y).$$

The suggested analogue, $A1'$, for the base axiom $A1$, does not apply at the expected step because the unification of the literals fails (symptom (a)). The problem is not that either the axiom $A1'$ or the next plan step is wrong, but that an extra step (boxed in Figure 7–1) needs to be added before $A1'$ is applied. When this step is made, $A1'$ applies, the inferred clause is found to match with the plan step, and the plan application proceeds, the problem having been overcome.

A similar thing can happen with symptom (b), where the axiom applies, but extra steps need to be added **after** the axiom in order to match the inferred clause with the plan step. It may also be necessary to insert steps both before and after the axiom in order to allow the step to go through.

So the cure here is to make extra inferences before and/or after the suggested axiom is applied (depending on which of (a) and (b) has happened), in order to infer a clause which matches with the next plan step.

- **Step right, axiom wrong.** It could be that, while the next plan step turns out to be right, none of the candidate axioms is right. This would happen if the analogue of the base axiom is not in the axiom base, and needs to be derived from the axioms which are in the base as a lemma. Similarly, the base axiom may itself have been a lemma, and thus it would not be surprising that the analogue would have to be proved as a lemma. Figure 7–2 shows an example of this from Boolean algebra where, in the base $x \cap x = x$ is proved by using the lemma $x \cap (x \cup y) = x$. Supposing that the analogous lemma $x \cup (x \cap y) = x$ has not yet been derived, it is conjectured, being the analogue to the base lemma, and an attempt made to prove it by analogy to the proof of the base lemma.

  The cure for this problem is to construct the analogue of the base axiom, conjecture it to be a theorem, attempt to derive it from the given target axioms, and then use it to make the next plan step. If a lemma was applied

in the base, it would be reasonable to attempt the proof of the conjecture by analogy to the proof of the lemma in the base. The conjecture proving stage could be omitted, with the conjecture being recorded as a hypothesis for the target problem. Bledsoe [Bledsoe 86] suggests something similar to this patching strategy.

Alternatively, a number of steps (which would represent the proof of the lemma) could be inserted directly into the plan, without explicitly proving the lemma.

- **Step wrong, axiom right.** Perhaps the fault is not with the axiom, but with the plan step, which is misleading. Figure 7–3 shows an example of this[1] which occurs in the attempt to prove

$$odd(x) \wedge odd(y) \longrightarrow odd(x \cdot y)$$

by analogy to

$$even(x) \wedge even(y) \longrightarrow even(x \cdot y).$$

The boxed plan step proves to be misleading, while the analogous axiom proves to be (weakly) useful, along with other additional axioms. The step needs to be ignored, and the plan is picked up again at the next step after extra inferences are made.

So the cure here is to make inferences, with the candidate analogous axioms and perhaps others, and attempt to match the inferred clauses with steps later in the plan.

- **Step wrong, axiom wrong.** Perhaps the next inference in the base proof is not needed in the target, and both the axiom and the plan step should be missed out. Figure 7–1 with base and target swapped provides an example of this; there is an extra step in the (new) base which is not needed in the target.

In order to patch in this situation, the following base axiom should be applied, and the result matched with the following base step. More gen-

1. Some of the axioms used in the proofs in Figure 7-3 are not shown in the Figure.

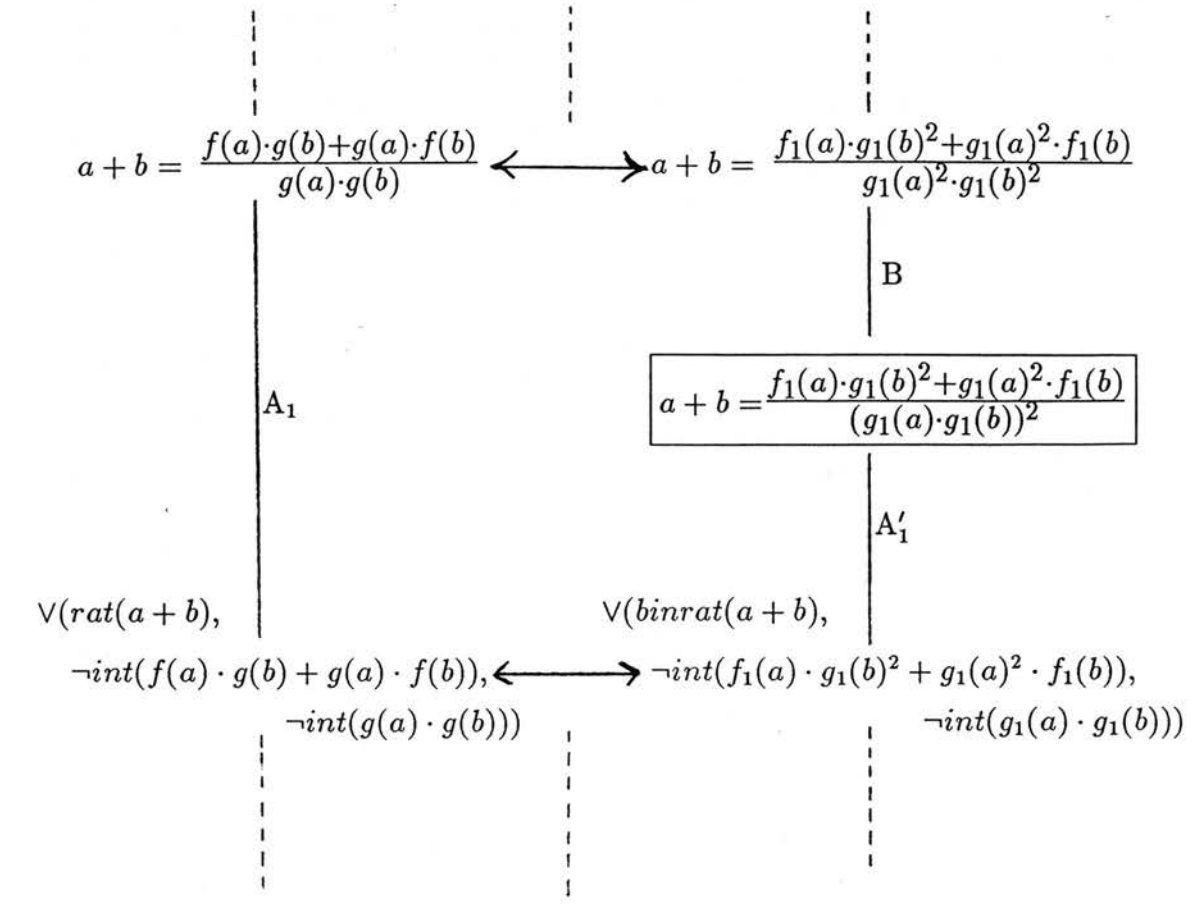


Figure 7–1: Extra step inserted

erally, inferences may be made from the current step, and the plan picked up again at a later stage.

7.2.3 How to implement patching strategies

The patching strategies described above fall into the following categories:

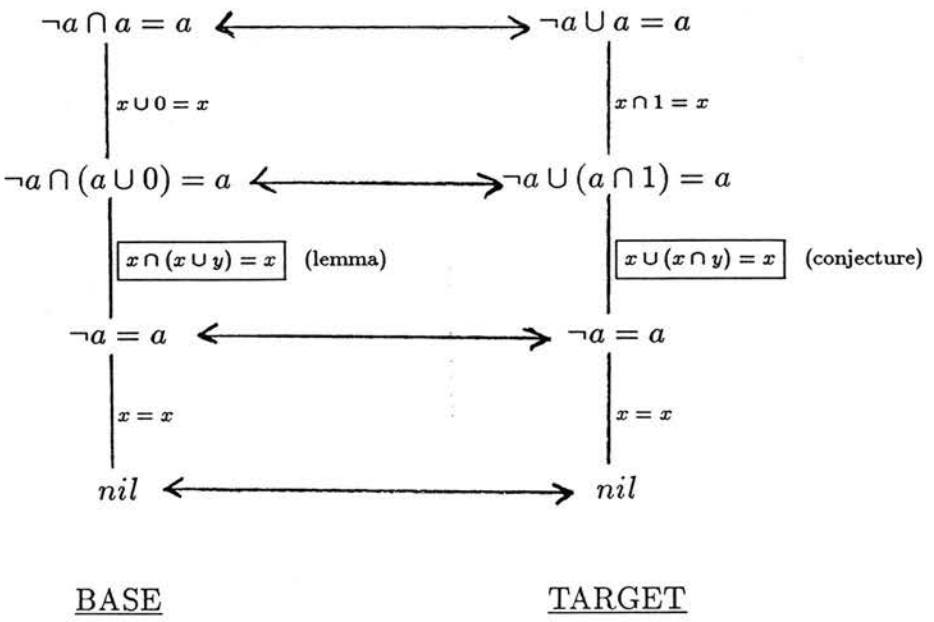


Figure 7–2: Conjecture made

- Patch by backtracking[2].
- Patch by conjecture.
- Patch by search.

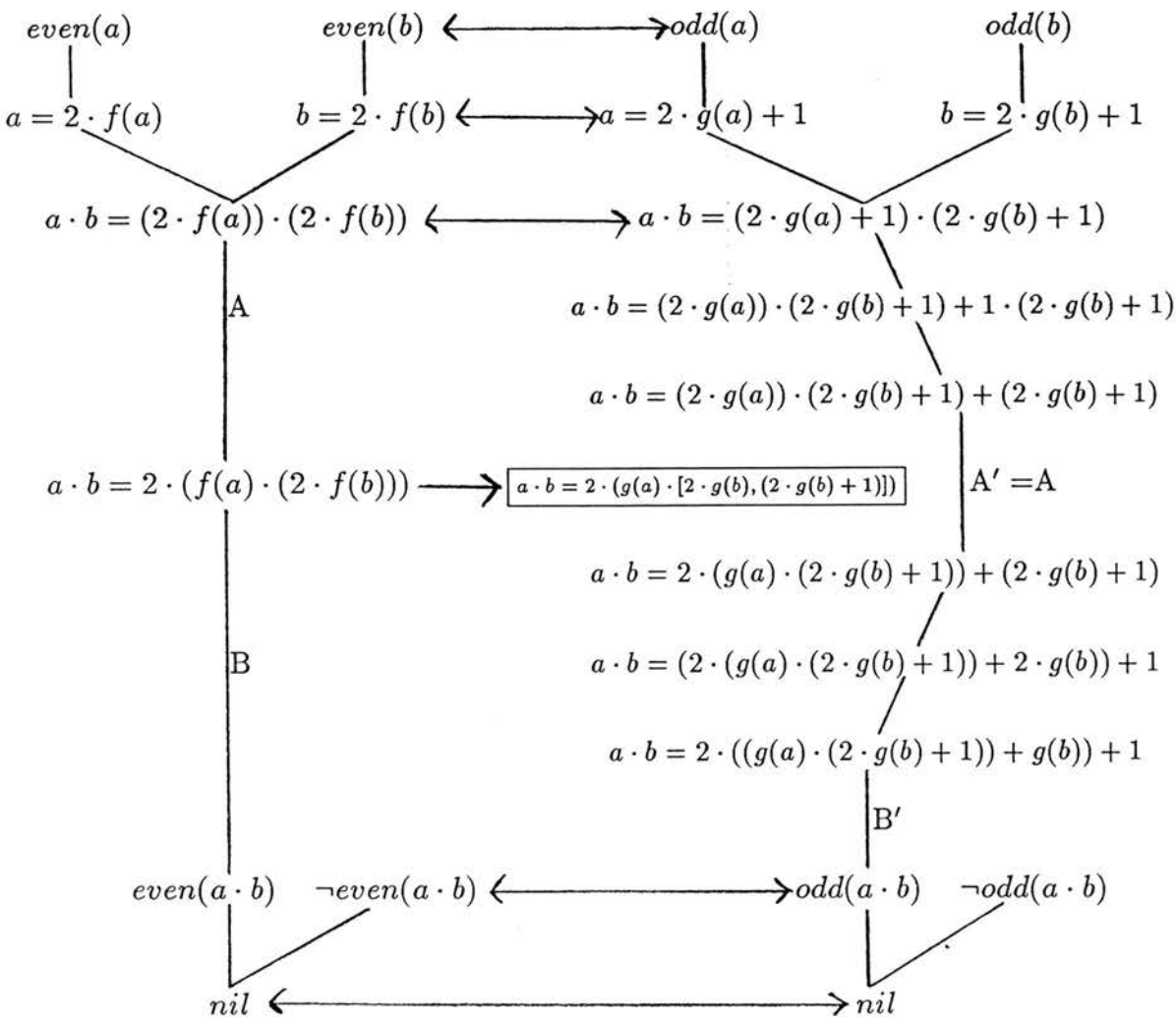
The first two of these have already been described. The most problematical is the third; we discuss it in more detail.

Patching by search

Note that, while we use the word search, we do not necessarily mean unguided search — the patching techniques which we will discuss are still guided by the

---

[2]Strictly speaking, it is wrong to refer to backtracking as a form of patching, since the plan need not be flawed for it to be necessary. However, we classify it along with the genuine forms of patching, since, as far as the application system is concerned, it is just another way to attempt to recover from a local failure.



A : $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

B : $\vee(\neg x = 2 \cdot y, even(x))$

A′ = A

B′ : $\vee(\neg x = 2 \cdot y + 1, odd(x))$

BASE

TARGET

Figure 7–3: Plan step ignored

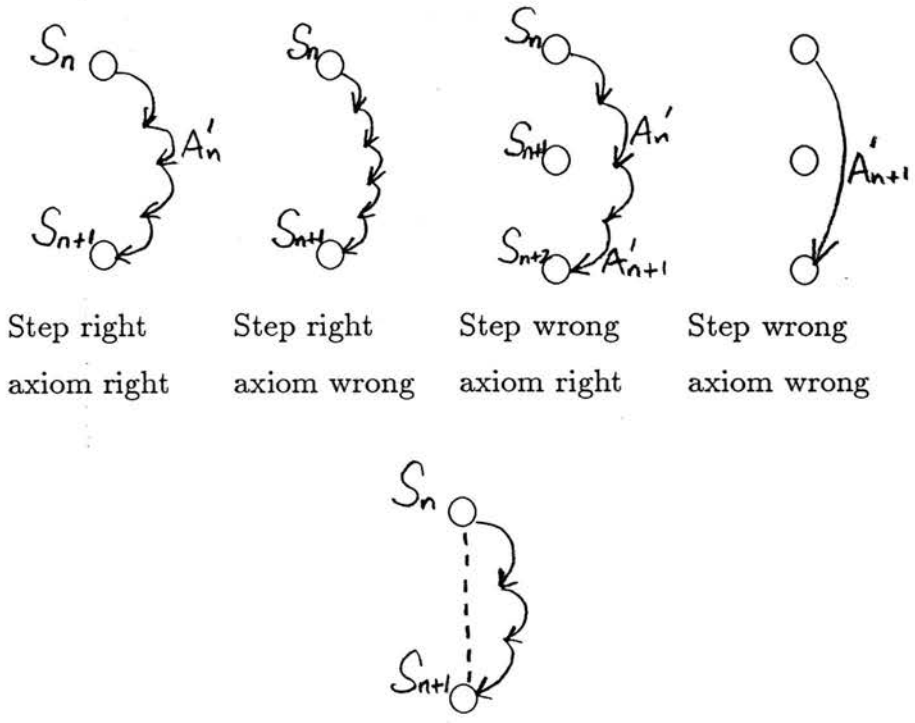


Figure 7–4: Schematic Patches

analogical plan, but not as closely as normal plan development. The top part of Figure 7–4 schematically represents the four examples of patching by search discussed in the previous section.

The application system could treat these types of patches as separate and search for them separately. However, we can consider them all to be subsumed by the lower half of Figure 7–4; the plan is picked up again at some later step, which could be the next one; the axioms which are applied to make the patching steps may or may not be candidate analogues for those in the base proof.

In order to search for this type of patch, inferences should be made from the current step, and inferred clauses should be matched against the following few plan steps. The application system could use unguided, breadth-first search to attempt this kind of patch — if the patches required were very small, this would perhaps be successful. However the example shown in Figure 7–3 suggests that

a more directed approach will be necessary to make the patches — six inferences are made before the plan is picked up again.

Search techniques We outline two methods for guiding the search for a patch, using the base solution as a guide — while the plan derived from the base has broken down, we still may be able to extract some assistance from the base proof in trying to patch the plan.

- Candidate analogues for the axioms used in the base in the next few steps could be given higher priority than others; i.e. we expect that these axioms may be applied *somewhere* during the patch.

- A form of means/ends analysis (MEA) could be used to select low priority axioms to apply (i.e. axioms other than the high priority ones indicated by the AOI). In Figure 7–1, for example, the candidate axiom will not apply because the unification of

$$\frac{f_1(a) \cdot g_1(b)^2 + g_1(a)^2 \cdot f_1(b)}{g_1(a)^2 \cdot g_1(b)^2} \quad \text{and} \quad \frac{x}{y^2}$$

fails. From the unification algorithm it could be seen that the failure is caused by the two subterms

$$g_1(a)^2 \cdot g_1(b)^2 \quad \text{and} \quad y^2.$$

An attempt could be made to rewrite the first so that it unifies with the second. This attempt could be further directed by considering the head symbols, multiplication and exponentiation, and how a term having the first as head symbol may be rewritten into one having the second. Equalities in the axiom base which produce the same replacement of head symbols could be used in the attempt to make the terms unifiable. In this case, the axiom

$$(x \cdot y)^2 = x^2 \cdot y^2,$$

assuming it was in the axiom base, would be selected and applied left to right, obtaining

$$\frac{f_1(a) \cdot g_1(b)^2 + g_1(a)^2 \cdot f_1(b)}{(g_1(a) \cdot g_1(b))^2}.$$

The candidate axiom now applies and the inferred clause matches with the plan step, thus completing the patch described in Figure 7-1.

The above technique applies to cases of symptom (a), where the patch is made in order to apply the axiom. A similar thing can be done with cases of symptom (b), where the axiom applies but doesn't match with the plan step. In this case it is the special unification procedure which will need to be amended in order to suggest **how** the two terms fail to match. Otherwise, the process would be the same as that for symptom (a).

A similar kind of MEA was used by LP [Silver 85] in order to patch the application of its schemas to new equations; but, instead of single axioms and their effect on head function symbols, LP used *methods* and their effect on *meta-level descriptions* (LP's methods can be thought of as meta-level operators, associated with collections of object level axioms).

Co-ordination of techniques The two methods of guiding the search for a patch to the plan could be combined in the following procedure:

- Decide how many steps ahead will be searched towards.
- Given these steps, form the sets St and Ax, of subsequent steps and candidate analogues for the corresponding axioms applied.
- Attempt to derive a clause which matches with a step from St by breadth-first search, where expansion of a node is given by the following procedure: try to apply each axiom from Ax to the current clause, calling the MEA procedure if necessary. For each new clause derived in this way, try to match it against each step in St, calling the MEA procedure if necessary. The procedure succeeds if/when an inferred clause matches against an element of St.

Directing the search for a patch ... One clear difficulty with this procedure is how far ahead to look in the plan. This will not be a problem with small

plans, as all steps can reasonably be included. But, bearing in mind that the MEA procedure may be quite expensive, it would be useful to direct the search towards likely plan steps, and away from unlikely ones.

... Towards likely steps. Obviously, the steps nearest to the point of failure will always be plausible. The following heuristic could be employed to select other likely steps:

> Subsequent steps where symbols which had been introduced into the proof at an earlier stage (i.e. symbols not occurring in the original problem statements) have just been removed should be regarded as plausible.

The motivation for this heuristic is that, where the analogy breaks down locally, we may be able to identify a subsequent step whose achievement can be thought of as the *purpose* of the intervening steps in the base; thus, while the intervening steps may not go through in the target, if we can reach the analogous goal in the target in a different way, we can pick the plan up again there. These intermediate goals can be thought of as the completion of stages of the base proof; steps where auxiliary symbols have just been removed are likely candidates for intermediate goals, since they can represent the completion of an inference concerning symbols we knew about before, which used the auxiliary symbols for the subproof; that is, these steps can be thought of as places where the base proof was made simpler.

For example, in Figure 7–3, the base step $even(a \cdot b)$ would be picked out as a likely stage, since the symbols f and 2 have just been removed.

... Away from unlikely steps. It may also be possible to direct search *away* from certain steps which can be recognised as being implausible. For example, in Figure 7–3, the misleading plan step shown could be recognised as such by the following means: it could be demonstrated that it does not follow from the previous step using axioms from the underlying theory, by construction of a counter example; this in itself does not rule the step out as being a useful one, since goal clauses, as well as axioms, could be used in a patch towards it. But

it does suggest that the step will be difficult to achieve, and so may be regarded as implausible.

... With bidirectional plan development. Another way of directing the search for a patch towards promising steps in the plan would be to work backwards from the end of the plan, attempting to validate a final segment of the plan; if this could be done, it would give the searching mechanism a firm step to aim at — it would know that, if the step could be reached, the rest of the plan would go through. For example, in Figure 7–3, the final two plan steps could be made backwards, producing the clause

$$a \cdot b = 2 \cdot x + 1$$

in place of the misleading plan step shown. This clause could give useful guidance via the MEA procedure outlined above. This proposal would require the reasoning system to be able to make inferences backwards; that is, given an inferred clause and one of the parent clauses, to construct a clause which could act as the other parent in the inference. In general, this would be difficult to achieve, possibly involving multiple possible results. More generally, validation of the plan could be taken up at *any* step. As well as the ability to make inferences both forwards and backwards, this would require the ability to reason directly with clauses in the extended plan language. This would not present many problems — the ordinary unification procedure within the inference rules would be replaced by the special unification procedure described in the last chapter.

An alternative approach

When attempting to patch by search, the various techniques tried would become successively more loosely based on the analogical plan, and successively closer to general theorem proving techniques. An alternative approach would be to design domain-specific patching procedures which were based on a meta-level analysis of the base solution, and the current state of search. The purpose of a meta-level analysis would be to abstract out the important features of the base solution (i.e. those responsible for its structure). An attempt could be made to

construct a sequence of steps in the target which had analogous features. The hope would be that, while the solutions may not be perfectly analogous at the lowest level, they may be so at a slightly higher level of abstraction.

This way of attempting patches is intuitively appealing since it seems that the clue to the use of an analogy often lies in finding the level of description at which the analogy holds. As discussed in Chapter 2, Gick and Holyoak regard the ability to change levels of description as an important part of analogical reasoning. However, finding a computational model of this process which would be of practical use in patching analogical plans will be difficult: it is not clear where the description language which expresses the crucial features comes from; if it is provided by the user there is a danger of circularity, rather similar to the problems with semantic types in Kling's matcher; that is, that the description language has been designed with the particular analogy in mind; alternatively, if a single language can abstract out the important features of a wide range of solutions, this suggests that the domain is so well understood that we would not have to bother with analogy in order to solve problems; techniques such as precondition analysis [Silver 85], discussed in Chapter 2, could be used to learn solution schemas instead. As argued in Chapter 2, a certain lack of understanding of the domain is necessary in order to make analogy an attractive method of solving problems, as opposed to methods of spontaneous generalisation.

From a pragmatic point of view, this direction for analogy research is probably best pursued within a context where meta-level planning is being used anyway, for separate purposes. This could help to avoid the circularity problems associated with the approach. Work being carried out on proof plans for guiding inductive proofs [Bundy & Wallen 86] might provide such a context.

7.2.4 Coordination of patching strategies

The procedure for coordinating the various patching strategies described above should take account of the following points:

- We do not want to spend too much time attempting patches by search or conjecture at genuine dead ends; that is to say, we would like to preserve the backtracking behaviour on the easy examples discussed in the previous chapter; i.e. if there is an alternative, and quite plausible, candidate axiom either at the present or at a previous step, we should backtrack to it before spending a great deal of time (if any) attempting to patch the current branch.

- On the other hand, we do not (necessarily) want to search exhaustively through all the candidate axiom sets in the plan before attempting any other sort of patch; some of the candidate axioms will not be very plausible, and should only be tried after some attempt is made to patch with the more plausible alternatives.

- For similar reasons, we would not want to spend too much time on patching by search without trying patching by conjecture, and vice-versa.

- As more time is spent on a particular patching strategy, without success, our expectation that it will ever lead to success will diminish; thus other strategies, initially less favoured, become more attractive by comparison. However, of course, we may want to return to the present strategy later.

These points clearly indicate that a fixed backtracking control structure for the patching strategies is insufficient. The present systems are based on Prolog backtracking; even with the restricted forms of patching of which they are capable (backtracking to another axiom, another step, or another plan), the Prolog control structure is not really suitable. We are forced to consider all candidate axioms at a step (which may involve much search), even the least plausible ones, before we can return to a previous step. Yet an untried candidate axiom may be waiting at the previous step which is just (or nearly) as plausible as the one which was originally applied there.

An agenda-based system is needed to support the kind of flexibility called for above. Each patching strategy would be given a numerical assessment of the

expectation of its success, on encountering a local failure. The most promising strategy is pursued first; after a while, it is re-assessed; if it has not been successful, its assessment may drop below that of another strategy, which would then be pursued instead.

There is some evidence to be had initially for the relative promise of the strategies, which can be used to set their initial assessments:

- if there are untried candidate analogues at previous steps which have match assessment nearly as high as those already tried, backtracking is suggested. The assessment for backtracking would be directly related to the most likely alternative axiom at some previous step.

- if there are no good untried analogues at present or previous steps, patching by conjecture is suggested.

- if the local failure occurs with symptom (b), i.e. the candidate axiom applied but the inferred clause did not match with the plan step, patching by MEA search is suggested, rather than patching by conjecture.

The above evidence is not very strong however, and certainly not strong enough to fix a permanent ordering on the strategies. The evidence is used to put an initial ordering on the strategies, which may be altered later.

Note that the agenda may include nodes from previous steps, as well as those from the latest step; i.e. the present step could be the result of a patch made at a previous step; so, if the present step proves fruitless, alternative strategies at the previous step may need to be investigated. It seems wise to reward a node for the number of plan steps which have been made to reach it; that is, we should be prepared to put more effort into patching (by conjecture or search) after many plan steps have gone through successfully.

A problem with the agenda-based approach would be the number of partially developed plans which would have to be stored, putting a strain on memory. It would be possible to implement a sort of plan-sharing between nodes on the

agenda, where the plan for a node could be computed from that for another, if the first node was ever made active. Additionally, if the space consumed by the system rose above a threshold, pruning of the agenda could be cued, in which less promising nodes would be deleted to reclaim space.

7.2.5 Comparison with Kling, Munyer and Carbonnell

As described in Chapter 1, Kling's approach to the search for a target proof by analogy to a given base proof is simply to use a uniform search procedure, but to restrict its axiom set to the set of candidate analogues for the axioms which were applied in the base. If we are to consider the candidate set as representing an unstructured plan, the plan can only be flawed if the target theorem is not actually provable from the candidate set. Thus Kling avoids all flaws resulting from the structure of the plan and the intermediate steps. This may give Kling's system an advantage on very seriously flawed plans, where structured application systems would struggle to find patches, being misled by the structure of the plan. However, on correct and moderately flawed plans, structured application systems would have a great advantage: recall that, on problems 5, 6 and 7 of the previous chapter, the axiom plan systems fail, being unable to search exhaustively, in reasonable time, the portion of the analogy search space from which they need to backtrack; this indicates that the search spaces within even correct analogical plans can be large if intermediate steps are not used; they will be much larger still if the order of axiom application is ignored as well, as is the case with Kling's system.

Munyer's application system ([Munyer 81]) is able to use breadth-first search in order to overcome gaps in an analogy (flaws, in our terminology). This is similar to the "patch by search" strategy discussed above, except that Munyer does not attempt to guide the search in any way.

Carbonnell [Carbonnell 83b] describes a number of "transformations" which can be made to a base solution in the hope of turning it into a solution to the target problem. Examples of such transformations are

- Swap two operators in the solution sequence.
- Reverse the solution sequence.
- Insert an operator into the solution sequence.
- Delete an operator from the solution sequence.

The various transformations give rise to a search space of possible operator sequences which can be obtained from the initial base solution. A goal, or success, state is any operator sequence which is a valid solution to the target problem. This space is known as the "T-Space". Carbonnell proposes search in the T-space as a means of analogy application.

In Chapter 1, we remarked that Carbonnell's system contains no solution mapping stage; that is, the initial state of the application phase, the base solution, which can be thought of as the plan for the target solution, will involve vocabulary (function, predicate and constant symbols) from the base problem rather than that of the target. If the target vocabulary is different from that of the base, as in most of the examples considered in this thesis, the application stage will be faced with an enormous, and largely unneccessary, search problem: each axiom applied in the base will have to be replaced by an analogous but different one in the target, by the transformation rules; additionally, any genuine patching required will have to be done as well by more applications of transformation rules. This appears to result in a T-space search of more than twice the depth of the final solution to the target. Furthermore, note that the branching rate in the T-space will be much greater than that in the object search space, since some of the operators are parametrised by at least one object-level operator.

It may be that Carbonnell intends his system to be used within a domain on analogies which involve no significant translation from base to target; for these types of analogies, the plan construction phase shrinks in importance. However, Carbonnell presents his system as a general model for reasoning by analogy, and, as such, it is open to the criticism made above: the most interesting analogies are

those involving superficially different entities; these are bound to involve genuine translations between base and target.

However, we may nevertheless regard Carbonnell's T-space search as a model of plan patching during application, and thus compare it with the proposals made earlier in this section: given a local plan failure, we could invoke a search in the T-space to patch the plan. The T-space approach corresponds to what we have called patching by search above, and would certainly be an alternative to the MEA procedure outlined above. As presented in [Carbonnell 83b], the T-space search is not directed in the same way as the MEA procedure (towards the step in the plan where failure occurred, towards analogues for base axioms and towards likely steps where the plan can be picked up again). However, it would certainly be possible to define T-space operators, and heuristics for searching the T-space, which would emulate the MEA procedure. Thus there is no fundamental difference of approach between the two techniques. Without guiding heuristics, the T-space search still looks explosive. Carbonnell does not consider specific ways of guiding the search.

7.2.6 Summary

We have discussed how and why straightforward application of analogical plans, of the type considered in the previous chapter, can break down. We have considered what kind of remedial action needs to be taken in order to recover from the local failure and resume normal plan application. We have described various kinds of patches which can be made to analogical plans, and have addressed the problem of the control structure which coordinates the investigation of the various possibilities. We concluded that an agenda-based application system would be necessary to allow the necessary flexibility, and discussed some of the issues in the design of such a system.

7.3 Improving Analogical Reasoning By Learning Global Analogies and Semantic Types

In this section we consider how an analogical reasoning system of the type described in this thesis could improve its ability to reason by analogy in a domain, by learning global analogies within the domain or between domains. The motivation for this is that the analogical reasoning systems so far described exhibit a similar kind of dumbness as ordinary theorem provers, one level up: that is, APS systems are called for so that theorem provers will be able to learn from their experience, and not repeat exactly the same behaviour when faced with a problem similar to one which they have solved before. However, the analogical reasoning systems, as described so far, in searching for and then applying an analogy which was essentially the same as one which had been encountered before, would go through the same steps of matching, plan construction and application as they would have done had they never encountered the previous analogy. Just as superficially different problems can sometimes be solved in analogous ways, superficially different examples of analogical reasoning, where the respective base problems and target problems may be very different, can involve essentially the same analogy.

We claim that, by abstracting out important aspects of successful attempts to solve problems by analogy, we can re-use them beneficially in new attempts at APS. The aspects which will be abstracted will amount to a description of the analogy which was found and used, without details which are specific to the problems that were involved. Since the description is not tied to particular problems, but could potentially apply to any from the domain in question, we can think of it as describing a global analogy within or between domains. Thus, our technique for transferring experience over different instances of APS is to learn global analogies, which we hope will be relevant to many different situations.

In Chapter 3, we criticised Kling's analogy matching algorithm because of

its use of **semantic types**. Recall how analogy matchers, such as Kling's, can make use of semantic types:

> Associations between symbols of the same (or similar) semantic type are preferred over those of disparate types.

How strong the preference is and how a given matcher implements it are separate matters. In Kling's case, the preference is very strong (in fact proscriptory) and is built into the structure of his algorithm. We argued (Chapter 3) that the types which Kling provided for the matcher encoded a global analogy between group theory and ring theory; the matcher merely reconstructed parts of this global analogy between individual problems from group theory and ring theory respectively. However, we also argued that the use of semantic types would be valid as long as the types could be learnt automatically in some way. In this section, we consider the construction or refinement of type hierarchies corresponding to the global analogies that are learnt over the course of APS experience.

We then consider how the global analogies, or semantic types, can be used within the Basic APS framework as described in the previous chapters; that is, how they can improve the performance of the analogical reasoning system. This includes their use in matching, as discussed earlier, but also extends to other parts of analogical problem solving.

We also discuss how the knowledge of global analogies, independent of particular pairs of problems, would enable a broader and more varied approach to the construction and use of analogies; we discuss how this more varied approach reflects better the use of analogies by humans than does Basic APS alone. However, we argue that the Basic APS process, which has been the main concern of this thesis, is a necessary part of the alternative uses of analogy.

7.3.1 Semantic types and dualities

Before we describe the techniques for learning global analogies and semantic types, we define more closely the structures which we intend to learn.

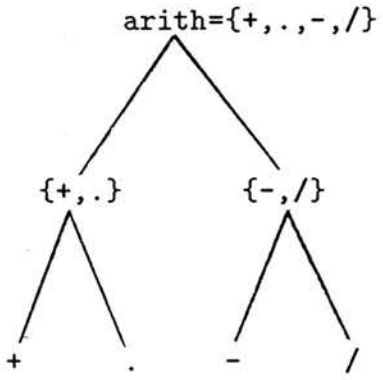


Figure 7–5: An example hierarchy

Semantic Types

The simplest definition of a **type hierarchy** is as a **tree**, the leaves[3] of which are labelled with symbols (function symbols, for instance) from the object language, and each non-terminal node is labelled with a set of symbols — the set containing just those symbols which label leaves below it in the tree. Figure 7–5 shows an example hierarchy. (This example is intended to illustrate the definition — no significance should be attached to the particular choices of node made here.) However, it is sometimes useful to consider more general structures than these, such as is shown in Figure 7–6. so some researchers [Plotkin 81] have defined a type hierarchy as an **upper semilattice**. This definition is motivated by the desire to make the version spaces/focusing algorithm [Mitchell 78] work (i.e. for there to be unique least upper bounds in the hierarchy), rather than by any direct reasons why concepts should be clustered in this way. For example, the dotted line in Figure 7–6 would not be allowed, but there seems to be no good reason why the resulting concept structure should not occur.

---

[3]The **leaves** of a tree are nodes which have no daughter nodes.

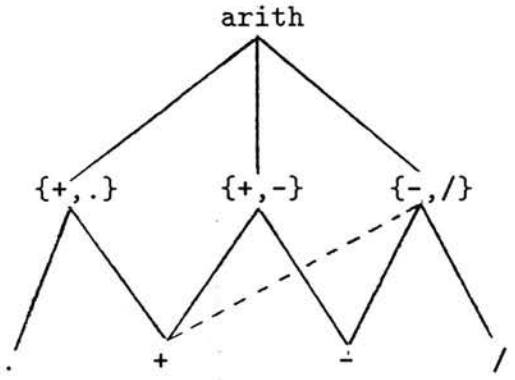


Figure 7–6: An alternative hierarchy

Dualities

We have argued that we should learn *global* analogies within or between domains; that is, analogies that are not tied to particular pairs of problems, as those which we have considered so far are. We will use the term **duality** to refer to such a global analogy, in order to distinguish it from the local analogies between pairs of problems. We give the following somewhat vague definition of a duality within a domain:

> A duality is (a) an association between some of the symbols in the domain together with (b) an association between axioms/theorems of the domain such that associated axioms or theorems are approximately mapped to each other by the symbol association.

Remarks

- The definition is phrased for dualities *within* a domain. Dualities *between* domains are defined in the same way, except that, of course, corresponding symbols or clauses must belong to corresponding domains. All of what follows applies equally well to this kind of duality.

- The definition is vague because we have not said what "approximately mapped" means. However, the notion is the same as that encountered in

analogies between individual problems. We could define a measure of the closeness of a duality, reflecting the degree of correspondence between the associated axioms, based on the assessment procedure for analogy matches described in Chapter 4.

Examples

A: Real number theory.

(a)
$$\begin{array}{ccc} + & \longleftrightarrow & \cdot \\ - & \longleftrightarrow & / \\ 0 & \longleftrightarrow & 1 \end{array}$$

(b)
$$\begin{array}{rcl} x+0=x & \longleftrightarrow & x\cdot 1=x \\ x+y=y+x & \longleftrightarrow & x\cdot y=y\cdot x \\ x-x=0 & \longleftrightarrow & x=0 \ \vee \ x/x=1 \\ & \text{etc.} & \end{array}$$

B: Boolean algebra.

(a)
$$\begin{array}{ccc} \cup & \longleftrightarrow & \cap \\ \cap & \longleftrightarrow & \cup \\ 0 & \longleftrightarrow & 1 \\ 1 & \longleftrightarrow & 0 \end{array}$$

(b)
$$\begin{array}{rcl} x \cup y = y \cup x & \longleftrightarrow & x \cap y = y \cap x \\ x \cap 1 = 1 & \longleftrightarrow & x \cup 0 = 0 \\ x \cup 0 = x & \longleftrightarrow & x \cap 1 = x \\ & \text{etc.} & \end{array}$$

C: Trigonometry.

(a)
$$\begin{array}{ccc} sin & \longleftrightarrow & cos \\ cos & \longleftrightarrow & sin \end{array}$$

(b)
$$\begin{array}{rcl} sin^2(x) = 1 - cos^2(x) & \longleftrightarrow & cos^2(x) = 1 - sin^2(x) \\ d/dx(sin(x)) = cos(x) & \longleftrightarrow & d/dx(cos(x)) = -sin(x) \end{array}$$

D: Abstract algebra.

(a)
$$\begin{array}{rcl} group & \longleftrightarrow & ring \\ subgroup & \longleftrightarrow & subring \\ normal\ subgroup & \longleftrightarrow & ideal \\ & \text{etc.} & \end{array}$$

(b)
$$\begin{array}{l} \neg subgroup(h,g) \vee group(g) \\ \qquad \longleftrightarrow \quad \neg subring(h,g) \vee ring(g) \\ \qquad \text{etc.} \end{array}$$

Remarks on examples.

- Example A shows the need for the word "approximate" in the definition — one of a pair of associated sentences has a condition which the other doesn't have.

- There is no need for a duality to apply to all axioms/theorems on each side; we fully expect that some properties of one side will not have close analogues (or analogues at all) on the other. For example the equation

$$sin(2 \cdot x) = 2 \cdot sin(x) \cdot cos(x)$$

  does not have a syntactically close analogue in C.

- Example B is an example of a formal duality, where the symbol mapping represents a perfect symmetry in the axiomatisation of the domain, which ensures that, for any theorem in Boolean algebra, its dual under the mapping is also a theorem. Other dualities may be regarded as partial symmetries.

The idea behind the definition of dualities given is to be able to represent, and hence exploit, situations where a part of a theory has similar structure to another part (or a part of a different theory). Formal dualities, such as B, are a limiting case of dualities, and do not seem to arise nearly so often as partial dualities do. Thus we claim it is worthwhile to study the more general kind, involving approximate mappings.

We discuss the exploitation of dualities in section 7.3.6; to give a brief idea here, they can be used in matching (in a similar, although potentially more powerful, way to semantic types), in plan construction (by implicitly extending the mapping between problems, thus making the plan more accurate) and in making and proving conjectures.

For partial dualities, successful exploitation will involve taking advantage of the duality where it applies, while not wasting too much time trying to use it where it doesn't.

7.3.2 Justification for Use

In Chapter 3. we argued that the semantic type heuristic would be effective in analogy matching if the types encode global analogies (what we are now calling dualities). The means of encoding provides the basic connection between types and dualities (which we will explore further in section 7.3.3):

> Symbols which are associated by a duality should have the same or similar semantic type (i.e. are sibling leaves, or close to being sibling leaves, in the hierarchy).

Recasting the effectiveness argument in terms of dualities (i.e. using the connection just given), we get a similar argument in justification of the use of dualities in matching:

> Analytical justification. Pairs of symbols associated by a duality are plausible as analogues because they have structurally similar axioms/theorems associated with them: thus, given that the dual symbols occur in structurally similar formulae, and regarding a solution as a sequence of structural rearrangements leading from a start to a finish, each arising from an axiom or theorem, it is likely that the two formulae will have similar solutions.

We could also take a more pragmatic approach to the justification for dualities, as in the following argument:

> Empirical justification. Analogies between individual problems often contain part of a global analogy (a duality) as well as more problem-specific elements: the associations
>
> $$+ \longleftrightarrow \cdot \ , - \longleftrightarrow / \quad \text{and} \quad 0 \longleftrightarrow 1$$
>
> often form part of specific analogies between problems, together with associations which do not belong to the global analogy. For this reason, symbols which are associated by a known duality are plausible as analogues within specific problems.

Just as it is possible to take both an empirical and an analytical approach to the justification of the use of dualities, we will see below that we can take both an empirical and an analytical approach to learning them.

Comparison with version spaces

As a brief digression, let us look at the success criterion for the use of the semantic type hierarchy in the inductive generalisation system LEX [Mitchell 78], discussed in Chapter 2: the hierarchy is used, by the **version spaces** technique, to **bias** the program towards making some generalisations, those that involve types from the hierarchy, rather than others. Once made, a generalisation will be applied to new problems containing symbols of the appropriate types in the appropriate places, i.e. if corresponding function symbols between the new and old problems are of similar types. The technique will be successful if the new problems, which match the generalisation, can be solved in suitably similar ways. This will be true if the types encode global analogies within the domain; i.e. the success criterion for the use of types in version spaces is the same as that in analogy matching.

This point is significant as it indicates that hierarchies may be shared between the different techniques. A refinement or addition to the hierarchy made by one of the techniques would be available for use by the other. Both uses for type hierarchies have associated methods for learning and refining them: [Utgoff 83] and [Wielemaker & Bundy 85] describe learning methods for generalisation; section 7.3.5 describes learning methods for analogy.

7.3.3 Connection between types and dualities

In Chapter 3 we introduced the basic connection between semantic types and dualities:

> Symbols which are associated by a duality are given the same or similar semantic type in the hierarchy.

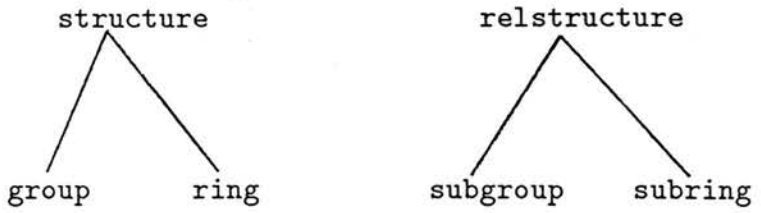


Figure 7–7: Algebraic hierarchies

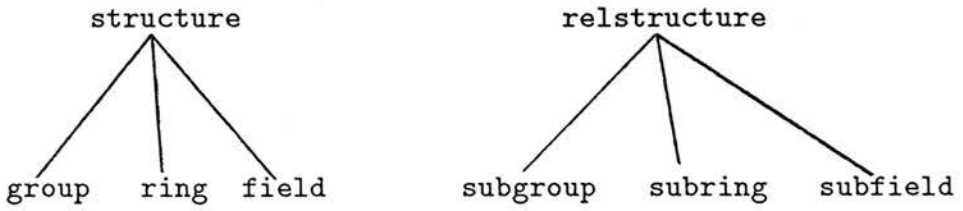


Figure 7–8: Extended algebraic hierarchies

Now that we have been more specific about the nature of the types and dualities, we can look at the connection in more detail.

Notice, first of all, that some of the information in a duality is lost when it is represented as types. For example, the duality

$$\begin{aligned} group &\longleftrightarrow ring \\ subgroup &\longleftrightarrow subring \end{aligned}$$

might be represented as in Figure 7–7. Analogous symbols are given the same type, but the dependency between the associations in the duality (i.e. if groups are seen as rings then subgroups are seen as subrings) is lost in the type structure. In Section 5, we will see that this information could be relevant for inductive generalisation.

On the other hand, dualities, as presented up to now, cannot directly model type situations such as shown in Figure 7–8. where a node has more than two descendants. We need to extend the definition to where the associations (of symbols, or axioms/theorems) could be ordered tuples. For example, a three-way symbol association is represented as (s_1, s_2, s_3); clearly the more expressive

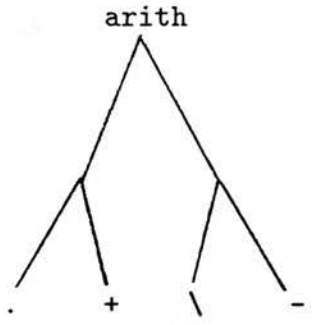


Figure 7–9: Arithmetic hierarchy

notation $x \leftrightarrow y$ available for binary associations cannot be used for ternary (or higher) associations.

Furthermore, dualities do not explicitly represent the hierarchical structure of types: suppose we had the fragment of type structure shown in Figure 7–9: The level 1 nodes correspond to the duality

$$\begin{array}{ccc} + & \longleftrightarrow & \cdot \\ - & \longleftrightarrow & / \end{array}$$

or

$$\begin{array}{c} (+,\cdot) \\ (-,/) \end{array}$$

and the *arith* node corresponds to the duality

$$\begin{array}{c} (+,-,\cdot,/) \\ (-,+,/,\cdot) \end{array}$$

together with the four-way associations of axioms and theorems that justify this duality.

But the dualities do not, as presented, explicitly represent the subset connection between the two. However, such connections could easily be derived in going from dualities to types.

7.3.4 Learning Dualities

Outline of learning methods

In the last section, two arguments were given justifying the use of dualities in providing *a priori* plausibility judgements in analogy matching: one was empirical, claiming that dualities frequently recur as parts of matches between individual problems; the other was analytical, arguing that two symbols having structurally similar axioms/theorems associated with them suggests that they make good analogues.

These two lines of argument directly indicate two different ways that an analogy system can learn dualities within a domain (or between domains), over the course of its APS experience.

Empirical method The first method, the empirical one, would work by examining analogy matches which the system had constructed when solving problems by analogy with other problems. The method would look for recurring submatches and for the operators (axioms or theorems) which had been found to be analogous by the application routine. The recurring submatches would form the (a) part of the dualities, and the associated operators would form the (b) part.

Analytical method The second method, the analytical one, would work by examining the knowledge base (composed of axioms, theorems and possibly unproved conjectures) and constructing explicit analogy matches between the constituents. From the successful matches, the method would group together those that were consistent with each other. The union of the matches in a group would form the (a) part of a new duality, and the pairs of axioms/theorems which gave rise to the matches would form the (b) part.

Requirements for the learning methods

The above are rough outlines of how the two methods would work. Before considering them in more detail, we consider some aspects of dualities which the methods will have to cope with.

Dualities are statistical Within $arith = \{+, \cdot, -, /\}$, we can find evidence for different dualities which are inconsistent with each other. Apart from duality A of Section 2, we could find evidence for

$$\text{(a)} \quad \begin{array}{ccc} \cdot & \longleftrightarrow & \cdot \\ + & \longleftrightarrow & - \\ sin & \longleftrightarrow & sin \\ cos & \longleftrightarrow & cos \end{array}$$

(b)

$$sin(x+y) = sin(x) \cdot cos(y) + cos(x) \cdot sin(y)$$
$$\longleftrightarrow \quad sin(x-y) = sin(x) \cdot cos(y) - cos(x) \cdot sin(y)$$
$$x \cdot (y+z) = x \cdot y + x \cdot z$$
$$\longleftrightarrow \quad x \cdot (y-z) = x \cdot y - x \cdot z$$

and for

$$\text{(a)} \quad \begin{array}{ccc} + & \longleftrightarrow & + \\ \cdot & \longleftrightarrow & / \end{array}$$

$$\text{(b)} \quad (x+y) \cdot z = x \cdot z + y \cdot z \quad \longleftrightarrow \quad z = 0 \vee (x+y)/z = x/z + y/z$$

If we tried hard enough, we could probably find evidence for almost any possible association within arith. We must be wary of this when looking for dualities. There is no reason why mutually inconsistent dualities should not be constructed and used, but there would be a danger of being swamped by too many. We would construct those which either (a) recur frequently (relative to others) or (b) are supported by at least several inter-related axiom/theorem matches such as duality A of section 2. It may also be useful to weight certain sentences, perhaps on syntactic grounds, as being important (for example, a statement of the form $f(x,x) = const$ might be regarded as important) and to favour dualities involving important sentences.

Rewriting problem Even if a strong duality "exists" in a domain, it is likely that the axioms will not be presented in a form which makes the duality clear, and thus amenable to the analytical method. In cases where the duality is clear, this is often because the provider of the axioms knows about the duality, and deliberately presents the axioms in an elegant, symmetrical form. We would like the learning technique to work in unfamiliar domains which had not been tidied up in this way. (In fact, the technique, where successful, would have the effect of doing the tidying up itself.) What would happen would be that axioms on one side would correspond to theorems on the other which would have to be proved from the (non-dual) axioms. In simple cases, it might be feasible to perform the inference when looking for matches within the analytical method (that is, if the necessary theorems had not already been proved). But, in general, this would present a massive search problem, and a rather unmotivated one, since the system would have little reason to believe that there was a duality there to be found until it had performed the inference.

In general, the establishment of the duality could involve an arbitrarily difficult re-expression of the axioms on one side, which would be beyond the scope of our techniques. However, if the difference between the forms of the axioms was relatively superficial, we might be able to combine the two learning techniques to overcome the difference. If a potential duality is masked by superficial differences between the axioms, we should still find that analogous *theorems* are provable. If these proofs are successfully achieved by analogy, we would find that the plan would have been patched: where a single axiom had been applied on one side, several axioms, or a single lemma (perhaps conjectured by analogy), would have been applied on the other side. Analysis of the trace of the successful APS by the empirical method would then suggest that the analytical method should attempt a corresponding rewriting in the axiom base in looking for its direct matches.

This suggests a control strategy for the two learning methods something like as follows:

1. Use the analytical method, without trying to do much, if any, inference. (This will only pick up obvious fragments of dualities).

2. Use any dualities derived from 1 to help with solving problems by analogy.

3. After some successful attempts had been made, use the empirical method to suggest new dualities.

4. Attempt to justify and extend these new dualities by the analytical method. Where extra steps were needed on one side of an analogy, use this to suggest what inferences to make.

5. Go to 2.

A more sophisticated approach would be to perform an agenda-based best first search, making progress on any of 1 to 4, whichever seemed most promising.

Making conjectures and refutations An addition to the analytical method as described so far is its use in making conjectures. Suppose the empirical method suggests a duality with some axioms/theorems to back it up. At stage 4 above, the analytical method tries to extend the duality to other axioms/theorems associated with the symbols already involved in the duality: that is, it tries to find new axiom/theorem pairs which have analogy matches consistent with the existing duality. However, it may find, say, a theorem on one side with no apparent analogue on the other. If the duality is strong enough, the method could construct the analogue of the given theorem via the analogue construction rule (see Chapter 5), and conjecture it to be true. It would pass the problem to the application routine, which would try to prove the conjecture *by analogy* to the proof of the theorem from which it was constructed.

It would be necessary to use a falsification routine (search for counter-examples) to prune out obviously false conjectures before much effort was expended in trying to prove them. If the conjecture turned out to be false, this information would be passed back to the duality, where the fact would be recorded. An attempt

could also be made to modify the conjecture so that the proof would go through. Negative information is important in dualities as well as positive information; negative information is usually provided by omission: when we give the duality $sin \longleftrightarrow cos$ it is implicit that it cannot be extended to, say, (sin, cos, tan) i.e. that the analogous conjectures for tan do not in general hold. However, in order to distinguish cases when conjectures had been falsified from those where they hadn't been made at all, we will need to add negative information explicitly as part of the dualities.

An example

We illustrate this technique on a database containing axioms for arithmetic. We consider three 'scenarios' arising from different possible axiomatisations of arithmetic.

Suppose that a four-way duality connecting the four arithmetic operators is known to the system; it could have been provided by the user or derived from the axioms/theorems shown (either at stage 1 or stage 3):

(a) $(+, -, \cdot, /)$
$(-, +, /, \cdot)$

(b) $(rat(x) \wedge rat(y) \rightarrow rat(x + y)$,
$rat(x) \wedge rat(y) \rightarrow rat(x - y)$,
$rat(x) \wedge rat(y) \rightarrow rat(x \cdot y)$,
$\neg y = 0 \wedge rat(x) \wedge rat(y) \rightarrow rat(x/y)$)

(possibly others)

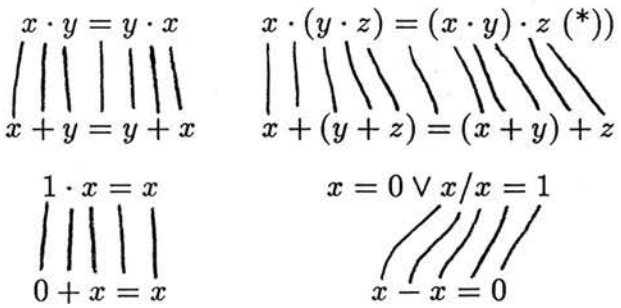
(rat stands for *rational*)

(It should perhaps be possible for the user to specify a duality without a (b) part and force the system to consider it even if there is no apparent evidence for it; this would amount to specifying the type *arith* with no justification for why it is a type.)

Scenario 1

At stage 4 of the control strategy, the analytical method tries to extend the

duality, by looking at axioms/theorems involving arithmetic operators and trying to find their analogues. Suppose for now that the axiom base has been set up symmetrically with respect to + and ·; we would have among the axioms and theorems:



The analogy matcher would produce the matches shown above. No obvious analogues for (*) involving - and / have been found, and similarly for the rest. So the following conjectures are made:

$$x - y = y - x \qquad x - (y - z) = (x - y) - z$$

$$bdef(y = 0) \vee x/y = y/x \qquad bdef(y = 0) \vee x/(y/z) = (x/y)/z$$

and so forth

The $bdef$ terms are produced by the ACR as a result of the extra condition in one of the original analogy matches; recall the interpretation of these terms, normally enforced by the special unification procedure: the $bdef$ term can stand for any number (usually 0 or 1) of occurrences of terms which match its argument, with the convention that variables are not shared between different occurrences, or with the rest of term; thus

$$bdef(y = 0) \vee x/y = y/x$$

would match with all of the following

$$x/y = y/x \ , \quad y = 0 \vee x/y = y/x \quad and \quad x = 0 \vee y = 0 \vee x/y = y/x \ .$$

In the case of proving or refuting conjectures, corresponding interpretations would be made; for example, in this case, the refutation procedure would only succeed if all the variables in the counter-example failed the extra condition.

The falsification routine should be able to refute each of these conjectures easily. So these sentence forms are not true for the whole of *arith*; this indicates that there is a more specific duality within *arith*, which would then be constructed:

$$\text{(a)}\quad \begin{array}{ccc} + & \longleftrightarrow & \cdot \\ - & \longleftrightarrow & / \\ 0 & \longleftrightarrow & 1 \end{array} \qquad \text{(b)}\quad \begin{array}{ccc} 0+x=x & \longleftrightarrow & 1\cdot x=x \\ x+y=y+x & \longleftrightarrow & x\cdot y=y\cdot x \\ & & \text{etc.} \end{array}$$

together with the information that this duality does not extend to the whole of arith. Even this duality is not perfect: the axiom $x \cdot (y + z) = x \cdot y + x \cdot z$ has no obvious analogue among the axioms. The best guess at an analogue, $x + (y \cdot z) = (x + y) \cdot (x + z)$, would be rejected by counter-example.

Scenario 2

Now let us retract the assumption that the axiom base had been set up symmetrically, and consider a slightly more difficult situation where the axiom base had not been set up quite symmetrically. Suppose we had:

$$\begin{array}{ccc} & 1\cdot x=x & \\ \text{but} & x+0=\times & \end{array} \qquad \text{and} \qquad \begin{array}{ccc} & x\cdot(y\cdot z)=(x\cdot y)\cdot z & \\ \text{but} & x+(y+z)=z+(x+y) & \end{array}$$

as axioms.

In fact, the matcher, given access to the commutativity axioms for + and ·, would be able to straighten the matches out itself. However, for the sake of illustration, let us assume that it could not. Furthermore assume that the matches that were found by the analytical method at stage 4 were not strong enough to activate the conjecture-making process; whether or not this was so could depend, among other things, on whether the system had "spare time" which it could use making and testing conjectures.

The system returns to stage 2, i.e. solving problems by analogy. Suppose the system has a proof of

$$(y + x) - x = y$$

in its knowledge base and, when asked to prove

$$x = 0 \vee (y \cdot x)/x = y$$

decides to do it by analogy to the stored result. Assuming that the application procedure was able to do this, we might have the analogous proofs for the two statements shown in Figure 7–10. (Again, the existing application routines take into account any known commutativity of functions, and so would not need to make even this simple patch in this case; again, for the sake of illustration, we assume that the ability to deal with commutativity has been "turned off".) The third step in the base proof corresponds to two steps in the target. At stage 3 of the learning process, the duality

(a)
$$\begin{array}{ccc} + & \longleftrightarrow & \cdot \\ - & \longleftrightarrow & / \\ 0 & \longleftrightarrow & 1 \end{array}$$

(b)
$$\begin{array}{ccc} (y+x)-x = y+(x-x) & \longleftrightarrow & (y \cdot x)/x = y \cdot (x/x) \vee x = 0 \\ x - x = 0 & \longleftrightarrow & x/x = 1 \vee x = 0 \\ y + 0 = y & \longleftrightarrow & y \cdot 1 = y \end{array}$$

would be suggested, where $y \cdot 1 = y$ is a theorem, proved during the course of the larger theorem, which is the analogue of the single axiom on the left hand side.

At stage 4, the analytical method would extend the duality with

$$\begin{array}{ccc} x \cdot y = y \cdot x & \longleftrightarrow & x + y = y + x \\ 0 + y = y & \longleftrightarrow & 1 \cdot y = y \\ & \text{etc.} & \end{array}$$

where $0 + y = y$ would be conjectured and proved, perhaps by analogy to the proof of $y \cdot 1 = y$.

Thus the asymmetry in the knowledge base would be removed, the duality would have been learnt and would be available for future use in the ways described in Section 6.

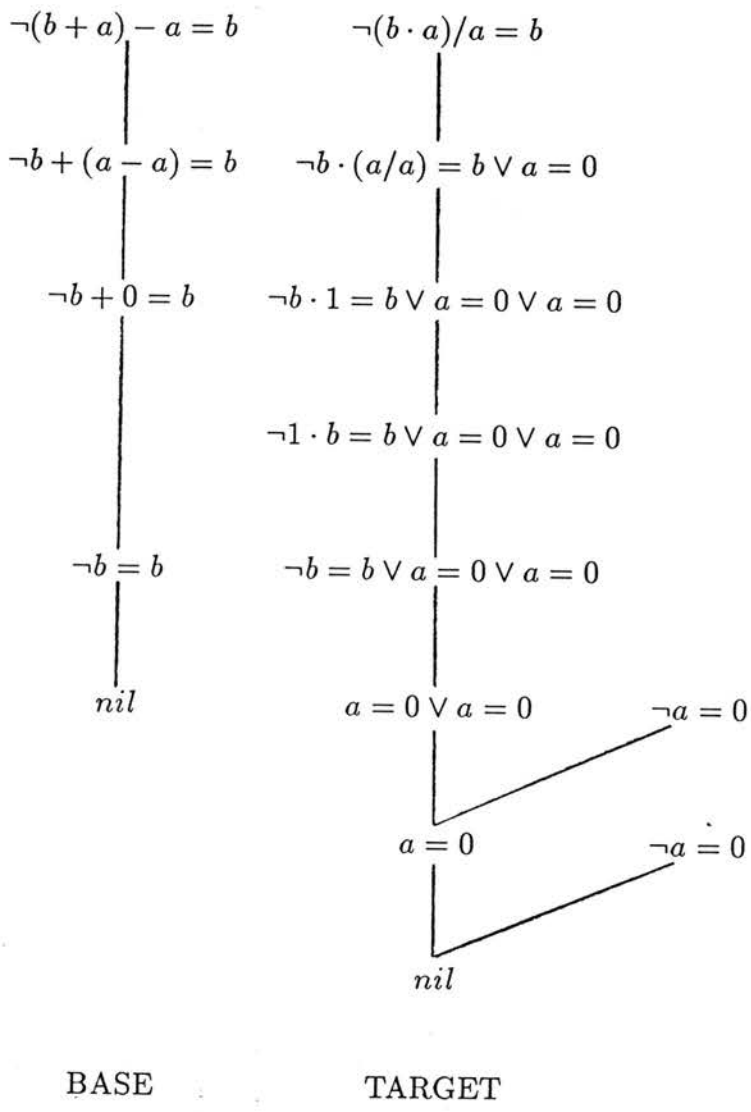


Figure 7–10: Analogous proofs

Scenario 3

Finally, we consider a much more serious case of misleading asymmetry in the knowledge base. Suppose that the axiom base for arithmetic had been set up in a different way from that given and used in section 5.1.3 of the previous chapter, where analogous axioms for addition and multiplication were used. Suppose that the additive axioms

$$7. \quad x + min(x) = 0$$
$$9. \quad x - y = x + min(y)$$

were retained, but the analogous multiplicative axioms

$$8. \quad x.inv(x) = 1 \vee x = 0$$
$$10. \quad x/y = x.inv(y) \vee y = 0$$

were replaced by the superficially non-analogous

$$8'. \quad x/x = 1 \vee x = 0$$
$$10a'. \quad inv(x) = 1/x \vee x = 0$$
$$10b'. \quad x.(y/z) = (x.y)/z \vee z = 0$$

The equivalence between the two sets of multiplicative axioms (modulo the rest of the multiplicative axioms) is sufficiently obscure for it to be out of the question to make the necessary inferences during matching.

Suppose the system was asked to prove

$$\neg x \cdot y = 1 \vee x = inv(y) \vee y = 0$$

and decided to do it by analogy to a stored proof of

$$\neg x + y = 0 \vee x = min(y)$$

using the following match constructed by the matcher

$$\neg x \cdot y \doteq 1 \vee x = inv(y) \vee y = 0$$
$$\neg x + y = 0 \vee x = min(y)$$

Assuming that the application system was able to do this (a big assumption, since a major patch is required), we might have the corresponding proofs shown in Figure 7–11. The axioms applied have been left out at the easy steps, those that go through without patching. Corresponding to the two applications made in the base of the axiom

$$7. \quad x + min(x) = 0$$

there are applications of a *lemma* in the target,

$$x \cdot inv(x) = 1 \vee x = 0 \ ,$$

which would have been conjectured by the existing analogy, and proved, as in Figure 7–12 (or perhaps left as an unproved assertion). Thus, we assume that "patching by conjecture" has been used. Alternatively, "patching by search" could have been used, with extra steps being put in the target proof (effectively, the proof of the lemma would need to be transcribed into the plan for the target, twice.)

With the analogies between the two theorems, and between the axiom and the corresponding lemma having been established, there would be enough evidence for the empirical method to propose the duality

$$\begin{array}{rcl} + & \longleftrightarrow & \cdot \\ min & \longleftrightarrow & inv \\ 0 & \longleftrightarrow & 1 \end{array}$$

In order to extend the duality, the analytical method would need to conjecture and prove analogues for axioms 9, 8′, 10a′ and 10b′. If this could be done, the equivalence of the alternative multiplicative axiomatisations would effectively have been proved.

The proofs of the conjectures made by analogy in this example are quite hard, and would need a quite powerful theorem prover to be found automatically. This

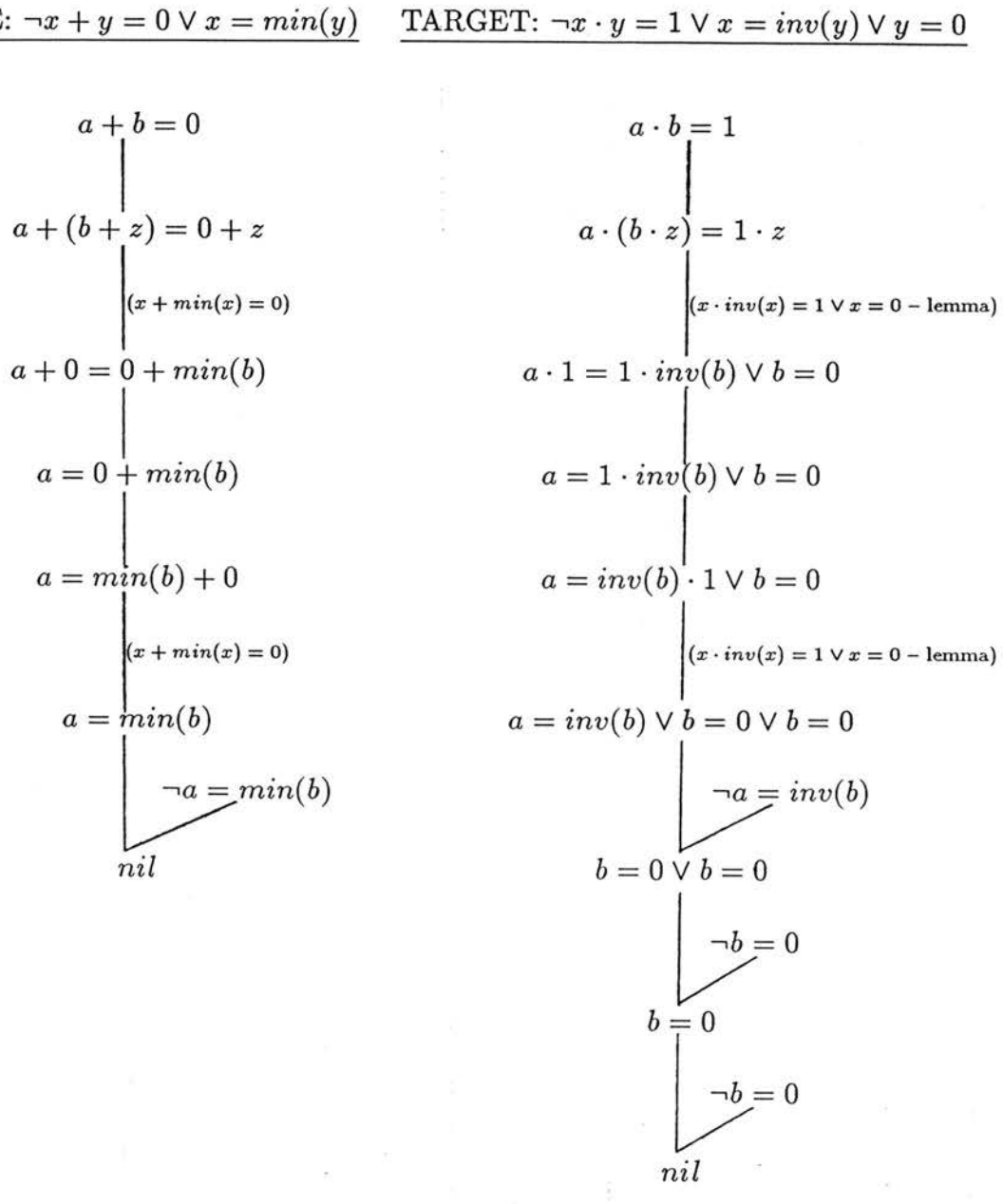


Figure 7–11: Analogous proofs with lemma

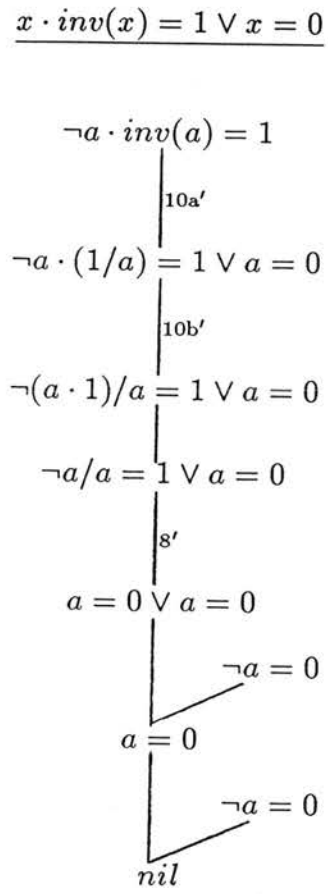


Figure 7–12: Proof of lemma

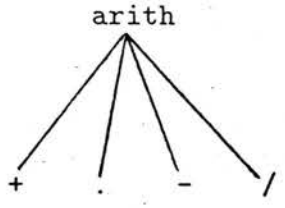
is simply because the equivalence between the two axiomatisations is not at all obvious.

The control strategy proposed for the coordination of the learning methods is just one way it might be done, and the details should not be taken too seriously; the important point is that the two methods should be able to help each other, especially when one gets stuck — the empirical method suggests symbol mappings, axiom associations and inferences to the analytical method, while the latter makes conjectures and extends the proposed dualities so that the empirical method becomes better at solving problems by analogy.

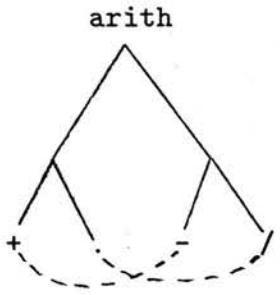
7.3.5 Learning Types

We have argued that dualities provide a means for type formation, and that they provide the justification for the use of type hierarchies in analogy matching and inductive generalisation. In order for the creation of types to be of much use, the new types will have to be assimilated into the existing hierarchy, which may have been prespecified by the user, previously learnt, or derived by some other means.

We should also note that the remarks about the statistical nature of dualities, made in Section 3.4.2, apply equally to types — the types that we form should correspond to *strong* dualities. Consider the arithmetic example of the last section. Having done all the work to learn about the new duality, the corresponding refinement to the type structure is easy. Initially, we had the type structure:

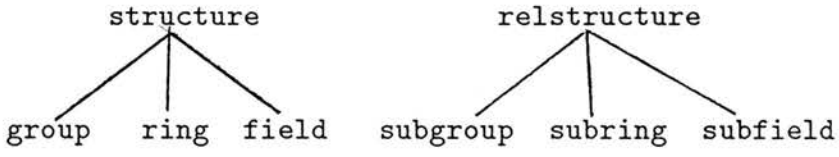


corresponding to the known duality (or this type could have been specified by the user.) Following the principle that symbols associated in a duality should be given the same type, we obtain —



from the new duality. The dotted lines indicate that the connected symbols were on the same side of the duality; this is the dependency information that is traditionally excluded from the hierarchy. However, the following example of inductive generalisation indicates that it might be wise to retain the dependencies:

Type hierarchies:




```
subgroup(h ,g) ---> group(g)
```

Positive instances:

```
subring(h, g) ---> ring(g)
```

The version space technique, briefly described in section 7.3.2, would generalise them to

$$p_1(h, g) \longrightarrow p_2(g)$$

where $p_2 \in \{group, ring, field\}$

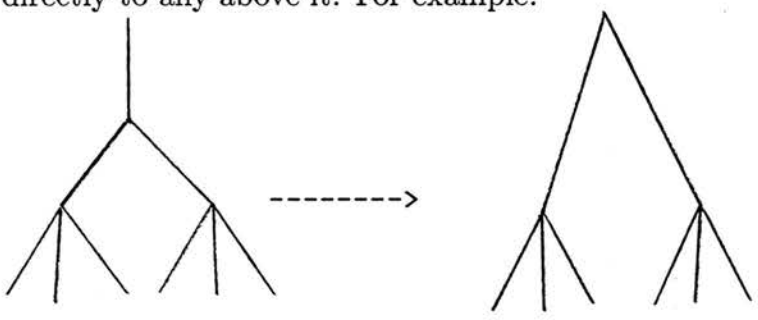
and $p_1 \in \{subgroup, subring, subfield\}$ independently.

This is a large over-generalisation. Attention to the dependencies indicates the generalisation

$$p_1(h, g) \longrightarrow p_2(g)$$

where $(p_1, p_2) \in \{(group, subgroup), (ring, subring), (field, subfield)\}$,

which is the intuitively reasonable generalisation to make. It may be that types which were once thought significant become superceded by stronger types for which evidence accumulates. Just as we have a notion of strength for dualities, we will map over a corresponding notion of strength for types. If a type is too weak, we may wish to remove it from the hierarchy altogether, linking any types below it directly to any above it. For example:



By being able to both add and delete types from the hierarchy, the system will be able to adjust its own bias without necessarily diluting it.

7.3.6 The use of dualities

The main claim of this section is that learning dualities within a domain will allow an analogy system to get better at analogical reasoning within the domain over time. Just as analogical reasoning itself is learning from past problems, the learning and use of dualities amounts to learning from past analogical reasoning.

We now consider how dualities, and semantic types derived from them, once learnt, can assist the processes of analogical reasoning. We also consider how dualities can be useful outside the framework of Basic APS with which we have been mainly concerned in this thesis: we briefly mention some alternative uses for analogy within an automated reasoning system, which, together with Basic APS, may begin to reflect the variety of uses made for analogy by humans.

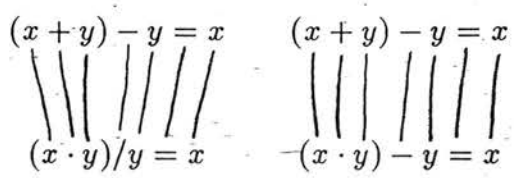
Within matching

One of the original motivations for the introduction of dualities was the desire to make the use of the **semantic type heuristic** within matching "respectable".

The idea behind the semantic type heuristic was to favour associations between symbols of the same, or similar, semantic type over those of disparate types. Expressing this in terms of dualities, we should favour the association of pairs of symbols which are associated often in known dualities more than those which are not. Obviously the **strength** of the dualities (as discussed above, involving the consistency of the mapping, and the number of associated axioms) which involve a given symbol pair would contribute to the judgement of semantic closeness.

The flexible matching algorithm, described in Chapter 4, makes the incorporation of extra heuristic criteria straightforward. Firstly, the match assessment procedure would be modified to reflect the degree of semantic closeness of symbols in the mapping, whether arising from types, dualities or both. Since all candidate matches are reassessed after each stage of match development, this would have the effect of moving matches consistent with the types and dualities towards the top of the agenda, and thus preferring them for development. Additionally, a heuristic development criterion could be added to the set which the matcher already uses — that is, semantic closeness could be used as a criterion for proposing new associations for a developing match, rather than just assessing those which had been made. The criterion, being a default criterion, would presumably be put in at level 2, along with the identical symbols criterion which it generalises.

Furthermore, just as the dependency information within dualities is relevant to inductive generalisation, it is relevant to analogy matching as well. Rather than just making *a priori* plausibility judgements for isolated pairs of symbols, we could also regard associations as more plausible if they occur in a match with other associations from the *same* duality than if the dualities are separate. Thus the first of the following two matches would be judged to be more plausible than the second:



A refinement to this would be to increase the plausibility attached to the first match if the solution to the base problem involved the use of axioms or lemmas which were known to be associated in the duality, and to decrease it if not.

Within Application

Dualities contain information about analogous axioms or theorems. So, if a duality is involved in a particular match between problems, and the solution to the base problem involved application of some of the axioms or theorems (as lemmas) from the duality, then the duality can be used to suggest the probable analogues for these in the solution to the target. This would avoid having to search for these analogues during the application phase, and so would speed up the process.

In a similar way, if a symbol occurs in the solution to the base problem which does not occur in the problem statement (and thus is not in the problem match) but does occur in a duality involved in the match, then its analogue can be predicted from the duality. Where applicable, this replaces the default clause in the analogue construction rule, and is likely to save time in application by making the right prediction straight away. This use for dualities would find application in problem 3 of the previous chapter: as the right hand side of Figure 6–5, of that chapter, shows, all of the application systems make the wrong inference at the first step, and only recover after varying amounts of fruitless search. The problem is caused by a marginal preference for the association

$$- \longleftrightarrow -$$

over

$$- \longleftrightarrow /$$

given an initial mapping containing the associations

$$\begin{array}{ccc} rat & \longleftrightarrow & rat \\ + & \longleftrightarrow & \cdot \end{array}$$

This preference would be reversed given knowledge of the duality

$$
\begin{array}{ccc}
+ & \longleftrightarrow & \cdot \\
- & \longleftrightarrow & / \\
0 & \longleftrightarrow & 1
\end{array}
$$

since the dependency implicit in the duality would suggest / as analogue for $-$.

7.3.7 Conjectures and refutations — alternative uses of analogy

(The title of this section is taken from [Popper 68].) We have already seen that the ability to make and test conjectures is a necessary part of duality formation, as well as being interesting and useful in its own right. It is possible to use the same mechanism in the problem solving phase; given a problem to solve which has no close analogue among previously solved problems, but which involves symbols from a duality, we could apply the mapping from the duality to the given problem in order to construct a new problem; we would then attempt to solve the new problem and, if successful, try to map the solution back to a solution to the original problem. This is the paradigm for reasoning by analogy used by Brown [Brown 77]. Similarly, if the dual problem was rejected by counter-example, we could try to construct a counter-example to the target problem by analogy.

It would be useful to have some measure of the relative difficulty of the two sides of a duality in order to make the technique successful — we would want to map problems from the harder side to problems on the easier side, and then to use the solutions to the easier problems as models for the solutions of the harder ones. For example, we might map a problem in three dimensional geometry to its analogue in two dimensional geometry rather than the other way around. One way of deriving such a measure would to be to consider the relative complexity of the solutions to pairs of dual problems which had previously been encountered; this would involve the number of steps in the solution, the number of applicable operators and other relevant factors.

A more radical departure from the Basic APS framework, but one which has been suggested as an important use for analogy by humans, is to use a

hypothesised analogy to refine the axiomatisation of a domain; that is, rather than taking the axiomatisations and the problems as given, and constructing an analogy based on these, we would take a complete axiomatisation of the base domain, a partial/provisional axiomatisation of the target, (part of) an intended global analogy, or duality, between the domains, and some properties of the base domain which we wanted to hold in the target; the aim of the process would be to extend or refine the axiomatisation of the target domain and the analogy mapping, so that the specified properties of the base would have analogues in the target. We could approach this vaguely specified, but interesting, problem as follows: we take the proof of one of the specified properties in the base and attempt to transfer it to the target, using the analogy mapping provided to construct a plan for the target proof. As we attempt to validate the plan, we will probably be unable to find analogues for the axioms used in the base; we will use the patch by conjecture strategy to assert analogous axioms in the target, except that, rather than conjecturing them as lemmas, we will propose them as axioms. In the process, the duality between the domains will presumably be extended; its extension may involve the introduction of new symbols into the target as analogues for known symbols in the base. This can therefore be thought of as a kind of definition by analogy.

Examples of this kind of use of analogy are, by definition, hard to find in standard mathematical literature, since, by the time textbooks are written, the domain axiomatisations have been fixed, and the exploratory processes involved in arriving at a good axiomatisation are not discussed. However, by looking at the historical development of certain branches of mathematics, we can see how axiom systems have evolved by analogy to other areas. For example, the theory of vector spaces developed from coordinate geometry to basis-free axiomatisations by analogy with existing branches of abstract algebra.

Notice that, while these more varied uses of analogy extend the Basic APS framework, the process of solving a problem by analogy to another one is still at their heart; the differences lie in where the problems and underlying axiomatisations come from. We therefore suggest that a competent Basic APS system

is a pre-requisite for the more exotic uses of analogy. This is why we have been almost entirely concerned with Basic APS in this thesis.

Inductive generalisation

Inductive generalisation programs such as LEX are based on type hierarchies. The learning techniques described in this chapter provide mechanisms for the creation/refinement of type hierarchies. Moreover, by the discussion in Section 3.2, the hierarchies produced in this way will be particularly suited to inductive generalisation. Thus it seems like a good idea for the analogy system to suggest new types to the generalisation system.

One problem might be that, since we have put no restrictions on the type structure derived from dualities, it might turn out not to be an upper semilattice ie least upper bounds may not be unique. One way of getting around this would be to pick out a maximal substructure which is an upper semilattice. The choice could be made on the basis of the strengths of the types. However, the problem really seems to lie with the generalisation technique which requires an upper semilattice; the solution could be to extend the version spaces technique to deal with more general structures.

7.3.8 Summary

We have argued for the learning of global analogies, or dualities, as a means of transferring experience from successful attempts at solving problems by analogy, to subsequent attempts. We have outlined two complementary learning techniques, one analytical and one empirical, for constructing dualities; we have also considered how the techniques can be coordinated, with each other and with normal analogical problem solving. Using the link between global analogies and semantic types, established in Chapter 3, we have discussed the creation and refinement of type hierarchies, based on dualities which have been learnt.

We then considered how dualities and semantic types, once learnt, can be used to improve performance within the Basic APS framework. We also dis-

cussed some alternative uses of analogy which are suggested by the use of global analogies, which persist over varying problem situations. We argued that basic APS is a crucial component in the extended systems.

Chapter 8

Conclusion

This chapter summarises the work described in the thesis. In particular we will review the achievements, limitations and motivation for our study of analogical reasoning. In discussing the achievements, we will consider how far we have got towards fulfilling the motivation, and in what respects we have got further than previous work. In discussing the limitations, we will consider how much further there is to go in order to fulfill the motivation; we will summarise the avenues for future research which we have discussed in previous chapters.

8.1 Motivation

As we have explained earlier, the motivation for our study of analogical reasoning is to enable problem solvers to learn from their experience. In Chapter 1, we called a system which can do this an **extensible deductive system.** As discussed in that chapter, learning from experience via analogy can take a number of forms: the Basic APS model with which we have been concerned in this thesis; definition by analogy, conjecture by analogy, generalisation from analogy; we have argued, in Chapter 1, that a competent Basic APS system is a significant and necessary step towards these alternative uses of analogy. The potential of analogical reasoning can only be fully realised within a discovery and reasoning

system such as that proposed in [Bundy 85], in which many different discovery and reasoning activities, including definition, conjecture and generalisation, would be co-ordinated.

8.2 Achievements

In this thesis we have developed a system which can effectively exploit a range of analogies to control the search of a conventional theorem prover. This has involved making developments in each of the component processes involved in the Basic APS model. The following sections describe these for each.

The work of this thesis has been grounded in analysis and reconstruction of previous analogy systems, in particular those of Kling and Munyer. Over the course of previous chapters, we have seen how these systems are unable to handle even very close analogies of the kind that occur in mathematics. The system described in this thesis is able to exploit at least such simple analogies, and can also apply some analogies which are not quite so close.

Another contribution of this thesis is the analysis of analogy systems in terms of the analogy heuristics which they make use of. The assumptions and heuristics behind the system developed have been made clear. This makes the system easier to understand, and easier to develop, for example by changing the significance attached to heuristics, or adding new ones which are appropriate for a particular domain. By contrast, the previous systems we have analysed have been presented, by their authors, in obscure ways: the underlying assumptions have not been brought out from their algorithms. We have been able to shed light on the systems, and understand their limitations, by analysing them in terms of the analogy heuristics.

Matching

The matching stage has formed the focus for the identification of the analogy heuristics. In Chapter 3, we identified the heuristics underlying existing anal-

ogy matchers and discussed their justification. We have shown that the existing matchers rely too heavily on weak heuristics, which seriously restricts their generality.

In Chapter 4, we described the design and implementation of a new analogy matching algorithm. The algorithm is based around the flexible application of analogy heuristics to guide the matching process. We have demonstrated the greater power and flexibility of the new matcher, compared with the old ones.

As a consequence of its design, the new matcher lends itself to modification and extension; the algorithm implemented is just one of many possible within the same framework.

Plan construction

In Chapter 5, we described procedures for constructing a plan for the solution of the target problem, given the solution to the base problem, and a match between the two problems. The procedures are able to produce plans which make use of all available information from the base proof, including both axioms and intermediate steps; previous systems have only made use of part of the information.

The analogous operator identification routine (AOI) finds a candidate set of analogues for an axiom applied in the base proof. We have seen that the AOI improves considerably on Kling's corresponding EXTENDER routine, which was seen to rely too heavily on a weak heuristic (but strangely a different weak heuristic than the one relied on by his matcher), as well as having other curiosities in its design. The AOI uses an initial filter to cut down possibilities, and then calls the full matcher to sort between the candidates. The filter is designed to be flexible and modular with respect to the heuristics. The second stage clearly inherits the flexibility of the full matcher.

The analogue construction rule (ACR) produces a plan step for each intermediate step in the base proof. It improves on Munyer's corresponding component in the following ways: it is able to express uncertainty in a plan step, arising from

either an inconsistency or a gap in the existing analogy match; in corresponding situations, Munyer's procedure would be forced to choose a single possibility, since his plan language does not include any features for expressing uncertainty. The effect of this is that Munyer's plans would often be unneccessarily **flawed** (in the sense defined in Chapter 7). Munyer's ACR makes heavy use of the weak **identical symbols heuristic**, which is not used by the new ACR.

The plan construction procedures developed in this thesis also construct analogues for the **inference positions** of the base plan — i.e. the literal and subterm positions at which the base inferences were made. The structural correspondence of the analogy match is used to produce hypothesised inference positions in the plan. We saw in Chapter 6 that these inference positions provide useful guidance for the plan application procedure.

Plan application

In Chapter 6, we demonstrated the ability of our analogy system to solve a range of problems by analogy to given solved problems (in fact we considered various application systems built in different ways out of the components developed in earlier chapters). We demonstrated how the application system was able to apply analogical plans effectively. We also demonstrated the dramatic reduction in search made possible by the guidance of the analogical plan, compared to a uniform search technique.

The application routine includes procedures for exploiting the uncertainty in analogical plans: the ability to try alternative analogous axioms when previous choices lead to failure, and the use of an extended unification procedure to test derived clauses against corresponding plan steps which involve uncertainty.

Our experimentation with a number of application routines, some involving different types of plans, has led us to conclude that full plans are the most effective overall; that is, the effort of constructing a full analogical plan is paid back by greater reduction in search. We have also concluded that the plan construction and application routines should be interleaved.

8.3 Limitations

The limitations of the work described in this thesis fall into two main categories:

- Limitations within the Basic APS framework; i.e. in what respects we have fallen short of building a powerful Basic APS system which can realise the goal of an extensible deductive system.

- Limitations of the Basic APS framework; i.e. how would a powerful Basic APS system fall short of realising the potential for analogical reasoning in an advanced discovery and reasoning system.

The second of these categories was discussed in Chapter 1; since we have said very little, in the intervening chapters, about alternative uses for analogy, we do not have much to add to these here. To recap, some of the potential alternative uses for analogy are **conjecture** by analogy, **definition** by analogy, **explanation** by analogy; in addition, we indicated how Basic APS could be embedded within a larger-scale problem solving activity — for example, using Basic APS **recursively** on subproblems generated during the application of analogical plans; using Basic APS to solve subproblems of a problem by analogy to subproblems of the *same* problem. This latter possibility is appealing, since it corresponds to a common technique in mathematics problem solving: solving a subproblem, then recognising that further subproblems are analogous to the first, and hence simplifying the overall solution.

Some of the proposals made in the previous chapter have a bearing on the alternative uses for analogy. In order to allow a Basic APS system to get better at finding and using analogies in a domain, we were led to consider global analogies, which are persistent across instances of Basic APS. As well as (potentially) improving performance of Basic APS, the knowledge of global analogies indicates alternative uses of analogy — in particular conjecture and definition by analogy as discussed in the previous chapter. Furthermore, we discussed how patching

by conjecture would be a potentially useful part of a flexible analogy application system. Thus, it seems that improvements in a Basic APS system will necessarily involve increasing its range of activities beyond the Basic APS framework. These tentative conclusions reflect the argument put forward in [Bundy 85] — namely, that separate discovery and reasoning activities can become more powerful by co-operating with each other.

We now turn to the first of the categories mentioned above — the limitations of our work within the Basic APS framework. Overall, the obvious limitation is that our system is only able to exploit quite straightforward analogies. As discussed above, the main bottleneck is in the application system, which has not been developed as much as the matching algorithm within this thesis. In the following sections, we describe limitations of each of the components of Basic APS which we have developed, and suggest corresponding areas for future development.

Matching

As discussed at the end of Chapter 4, there are various limitations of the current matcher:

Unification The current matcher is not guaranteed to find a first-order unification between two formulae, if one exists. This is a clear defect — while a first-order unification between the base and target is no guarantee of the base solution being of use for the target, the information must be considered relevant for the analogy system. An unsatisfactory solution to the problem would be to append a first-order unification algorithm onto the matcher; this is unsatisfactory because we would probably still miss matches which were "nearly" first-order unifications, which violates the flexible match approach behind the matcher. The most direct approach to the problem would be to change the **syntactic structure criterion** (see Chapter 4) to take account of possible variable bindings. The danger of this approach would be that the discriminatory power of the criterion would be diminished, leading to an explosion of candidate matches,

which would bog the matcher down. Whether this would be so is an issue for further work.

Pairing selection The pairing selection criteria are simple in the current matcher. That is, given possible pairings proposed by the criteria, sets of pairings which are actually made are arrived at in a fairly simple way. While this has proved adequate for the analogies we have considered in this thesis. it is likely that more sophisticated procedures will be needed for further development of the matcher. In particular, a more flexible weighting scheme could be useful in giving some pairings more weight than others.

Syntactic types The matcher takes no account of *syntactic* types, other than variables, in considering potential pairings or assessing evolving matches. That is, it does not distinguish between predicates, functions and constants. While this has not been a significant limitation for the matches considered in this thesis, it may prove useful to enable the matcher to prefer associations which respect syntactic types. This ability would be more significant within richer formalisms which have a greater variety of syntactic types than skolemised first-order logic; for example logics with sorts and sorting operations, and logics with extra quantifiers such as modal logics.

There are two ways in which we could build such discrimination into the matcher: in the assessment procedure, where matches which did not respect syntactic types would be penalised; or, more directly, in the match refinement procedure, by blocking pairings which do not respect syntactic types from ever being made.

Adding domain-specific information In the long term, the most important area for development of the matcher will be in the exploitation of domain-specific information to guide matching. The power of the current matcher is limited in that it makes use of very little domain-specific information. We first discuss with some short-term possibilities, and go on to discuss more ambitious and longer-

term possibilities.

Semantic types and dualities

We have already discussed (in Chapters 4 and 7) the feasibility and potential benefits of allowing the matcher to prefer matches consistent with known semantic types and dualities in a domain (or between domains). This provides a channel for learning from past analogies.

Given the machinery for generating and refining types and dualities, their incorporation into the matcher would be straightforward. Given that they represent a fairly weak heuristic, it would be appropriate to use them only in the match assessment procedure.

Matching modulo a theory

The **superficial difference problem**, discussed in Chapter 4, will clearly be an obstacle to the development of the matcher. To recap, it is where a potential analogy is obscured by superficial differences in presentation between the base and target problems. To overcome the problem, the matcher will need to have access to, and apply, the domain-specific rewritings which remove the difference. The difficulty of doing this automatically has already been discussed. Perhaps the most tractable approach would be to attempt to extend the way the current matcher treats propositional equivalences; if the propositional structures of the two formulae are not identical, they are both normal-formed, and matching is resumed; if the propositional differences between the formulae were superficial, they will be removed by this process. The availability of suitable normal forms will be a limiting factor in such an endeavour.

It is tantalising that humans seem to be able to see through superficial differences without even trying to, in many cases. Perhaps this results from a more general ability to switch between alternative representations of a situation. No such ability has been emulated so far in AI.

A particular type of superficial difference is that resulting from a definition being expanded on one side but not on the other. The current matcher takes no

notice of possible expansions, or contractions, of definitions. With definitions, a normal form does exist, namely the formula with *all* definitions expanded. This would not be a suitable solution for analogy matching for two reasons: first, the complexity of the matching problem could increase enormously; second, differences might well be *introduced* between the formulae, resulting from, say, functions which occurred in analogous positions but which had different definitions.

Meta-level matching

The current matcher is limited to matching problems at a single, basic, level of representation. It is commonly argued that a key to problem solving power lies in the ability to maintain alternative levels of representation; in particular, control of inference resulting from *meta-level* descriptions of object level problems has been demonstrated in a number of domains [Bundy et al 79,Sterling et al 82, Silver 85]. If the meta-level inference philosophy is successful, it is likely that multiple representations of problems will be common in future automated reasoning systems.

In earlier chapters, we have argued that analogy matching should take account of relevant meta-level descriptions of problems, as well as the object level. We have argued that the matching techniques themselves are not at fault, but that they should be applied at the more abstract levels of representation. This will become an important area for the development of analogical matching, when such meta-level descriptions become common in reasoning systems. We give an example which illustrates one of the matching issues which will arise — the coordination of the matching between different levels. Consider the following two equations (each to be solved for the unknown x):

$$sin(2 \cdot x) + sin(3 \cdot x) + sin(4 \cdot x) = 0$$

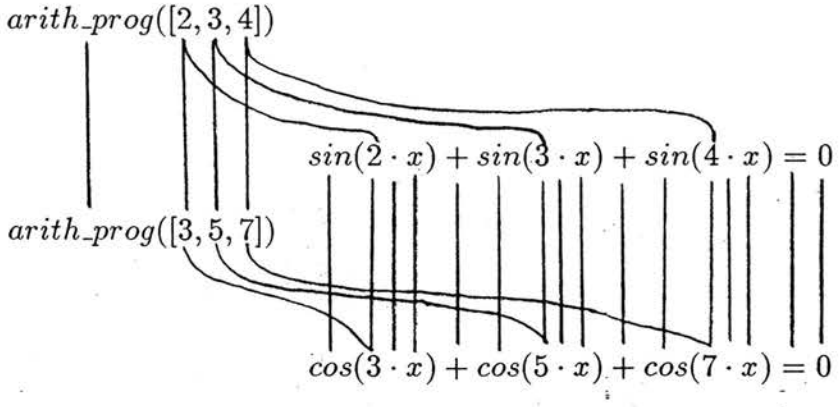
$$cos(3 \cdot x) + cos(5 \cdot x) + cos(7 \cdot x) = 0$$

These equations are analogous in that they can both be solved by the same method, which relies on the fact that they both have trigonometric terms whose

arguments are in arithmetic progression. The solution method proceeds by adding the first and the third terms, using the identity for the sum of two sines, factoring out a resulting common factor (this is why the arithmetic progression is important), splitting the equation, and solving the resulting equations by isolating a single occurrence of x. If we present this pair to the analogy matcher, six matches are produced, resulting from the six possible pairings of the terms between the equations; of these, the one involving a $3 \leftrightarrow 3$ association would be marginally preferred by the identical symbols heuristic. The matcher is unable to see that the arithmetic progression of arguments is the crucial factor (how would it know?). But suppose that the first of the equations was the base, and that the system had inferred that the arithmetic progression property was important and represented it as follows:

$$sin(2 \cdot x) + sin(3 \cdot x) + sin(4 \cdot x) = 0 \quad \text{where} \quad arith\_prog([2,3,4])$$

The *arith_prog* term can be thought of as a meta-level description of the object-level equation. The knowledge of the relevant property, expressed at the meta-level, can now be used to guide the object level matching by choosing between the six candidate matches: the matcher would try to extend the object level matches to the meta-level by finding meta-level properties of the target equation which correspond with that of the base *consistently* with the object-level match. This would pick out two of the six as being better than the rest. One of these is shown below:



Thus, knowing (or believing) a relevant feature of the base can guide the matcher towards promising matches.

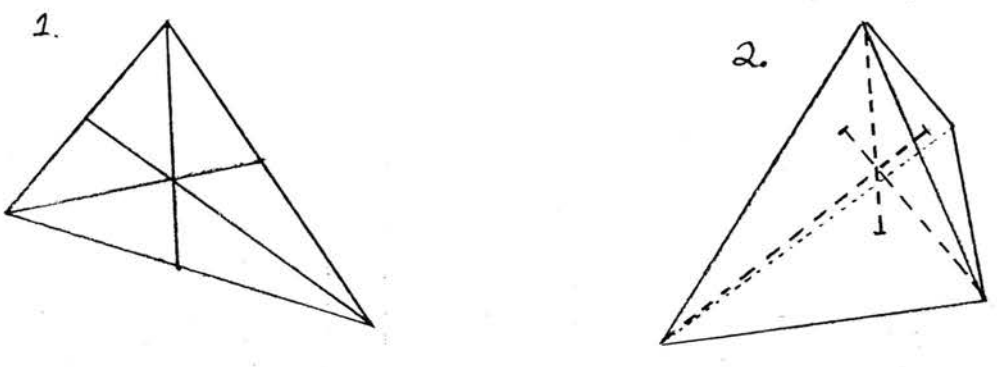
In a more developed system, features of the target could be derived automatically and matched directly with corresponding features of the base.

To end this section, we give an example in which there is a close meta-level match between problems but no obvious object-level match. Consider the following two problems from geometry:

1. The three lines joining the vertices of a triangle to the midpoints of the opposite sides intersect in a single point (the orthocentre).

2. The four lines joining the vertices of a tetrahedron to the orthocentres of the opposite faces intersect in a single point.

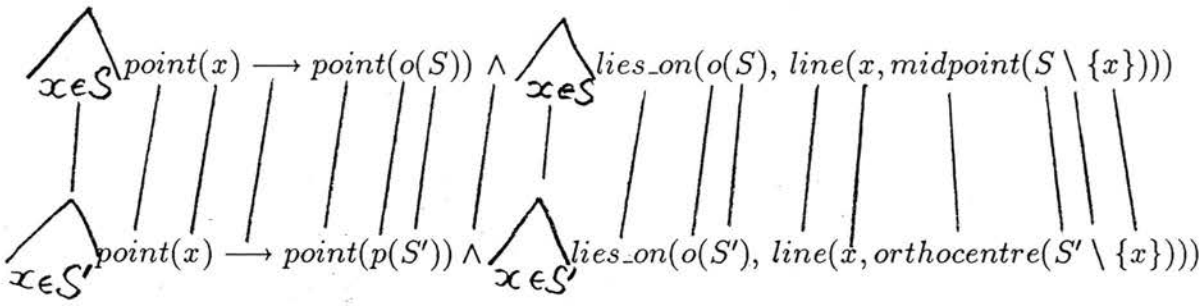
These problems can be illustrated as follows:



These problems would normally be considered analogous since (a) the second seems to be an extension of the first from two to three dimensions and (b) they can be solved in intuitively similar ways. We can represent the problems as follows:

HYPOTHESES	HYPOTHESES
$point(x)$,	$point(x)$,
$point(y)$,	$point(y)$,
$point(z)$	$point(z)$,
	$point(w)$
CONCLUSIONS	CONCLUSIONS
$point(o(x,y,z))$,	$point(p(x,y,z,w))$
$lies\_on(o(x,y,z),$ $line(z, midpoint(x,y)))$,	$lies\_on(p(x,y,x,w),$ $line(w, orthocentre(x,y,z)))$,
$lies\_on(o(x,y,z),$ $line(x, midpoint(y,z)))$,	$lies\_on(p(x,y,z,w),$ $line(x, orthocentre(y,z,w)))$,
$lies\_on(o(x,y,z),$ $line(y, midpoint(z,x)))$	$lies\_on(p(x,y,z,w),$ $line(y, orthocentre(w,z,x)))$,
	$lies\_on(p(x,y,z,w),$ $line(z, orthocentre(w,x,y)))$

(The functions o and p are introduced by skolemisation.) However, although there is a clear similarity between these formulae, we cannot give a symbolic correspondence without omitting parts of the target; such matches, while not necessarily useless, do not express the full correspondence between the problems. To do that, we need to realise that the four points in the target are treated in a corresponding way to the three points in the base when viewed as *sets*:



How could we hope to derive the latter expressions and the resulting match automatically? It is clearly necessary to have the ability to re-represent object-level geometry problems in terms of sets. The alternative representation can be

regarded as either a meta-level description of the object-level formulae, or as an equivalent object-level representation (this depends on how the knowledge in the geometry system is structured); if we choose the former, we describe the problem as one of meta-level matching; if we choose the latter it becomes analogy modulo a theory. It is clear that these two concepts can be different ways of expressing the same thing.

One approach to constructing the new descriptions, and the match between them, would be by analysing how an object-level match fails to define a close correspondence: i.e. certain of the atoms are unmatched; this could suggest ways of coalescing the formulae, using the set language, so as to remove the defects from the match. We could imagine that the final descriptions and correspondence could be derived by repeatedly applying this process to parts of the match until the entire match has been "straightened out". We should not over-simplify what would undoubtedly be an extremely difficult procedure. However, perhaps this would be a way of approaching the meta-level matching problem.

To sum up this section: there are many analogies which we cannot handle using the existing matcher at the object level alone; this is often because there is no adequate object-level correspondence. However, we can often construct a good symbolic match between alternative (perhaps meta-level) descriptions of the problems; thus the matcher will need to be integrated with an ability to switch between descriptions in order to capture these analogies; the nature of this integration is an important but difficult area for future research. We have suggested the interleaving of matching and re-description as an approach to the problem.

Plan construction

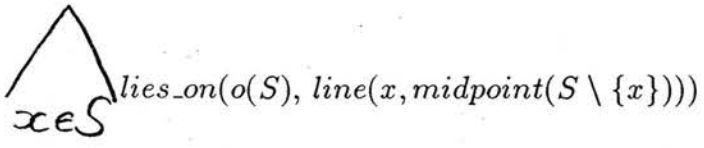
Given the use of the matcher within the AOI, the plan construction routines are the most straightforward and least problematic of the stages of Basic APS. Any limitations of the matcher will result in limitations of the AOI; hence all of the discussion of the previous section refers to the AOI stage as well as to initial matching. In a system able to reason at more than one level of description, as

referred to in the previous section, the plan construction procedures would need to be extended to deal with the alternative descriptions.

In order to construct a plan corresponding to a meta-level match, we would need to be able to relate the base solution to the meta-description of the base problem; that is, we need to be able to see the solution in the same terms as the problem; in some cases we may be able to re-express the solution completely at the meta-level, in which case the analogy task simply moves to the meta-level, with the existing techniques being applied.

Alternatively, the meta-level "proof", while coherent in itself, may not represent a complete proof of the formula, but rather an *explanation* of the object level proof. LP's analysis of a base proof resulted in plans of this form. In this case, both the object-level and meta-level parts of the base solution are eligible to be mapped into the target plan. Whether or not they can be would depend on whether there was a match, between the problems, at the object-level, the meta-level, or both.

Another possibility might be that the meta-level descriptions associated with the base problem are just annotations to the solution, without containing operators at the meta-level. For example, with the geometric analogy discussed above, the system would need to be able to relate the base solution to the set notation used in the meta-description; this would probably amount to chunking the solution into three parts, each a subproof proving one of the conjuncts in the conclusion; each subproof would be annotated with the vertex, from the set S, corresponding to the conjunct; the annotations would be related in a *member_of* fashion to the conclusion:



In order to construct a target plan, analogous chunks would be constructed for each element of the analogous set S'; within each chunk, extended versions of

the ACR and AOI would be applied to fill out the chunk in the target plan. For example, the ACR would need to compose the object annotations, the meta-level match and the target annotations to obtain plan steps for the target. The extended procedures would have to "understand" the relationship of the set notation of the meta-level to the object-level formulae.

Plan application

Plan application is the bottleneck of the present analogy system — many intuitively promising matches lead to plans which cannot be applied. Examples of these were given in the previous chapter. As discussed in the previous chapter, the application system is limited in that it cannot **patch** analogical plans significantly in applying them. It is clear that a powerful analogy application system will need to apply plans flexibly. The addition of patching ability is thus an important area for future research.

The current application system is inflexible in another way as well: a given plan is pursued completely, leading either to success or failure, before any alternative plans are considered. Similarly, a given choice of axiom in the plan is pursued completely before any other candidate axioms are tried. It would be desirable to remove both of these inflexibilities by controlling the plan application procedure from an agenda of partially developed plans. This area for future research has been discussed in the previous chapter.

The current application system is only able to make use of single level plans. It thus cannot make use of any structuring or annotation which might be available in the plan. We briefly extend our discussion of what a multi-level analogy system might look like, to consider how it might apply plans.

As discussed in the previous section, there would be a variety of possible forms that meta-level plans could take: the meta-level component could be very strong, in the limit representing a complete proof plan at the meta-level — the plans used by LP were of the strong form; or it may be restricted to annotations on an object level proof plan. The equation solving and geometry examples considered

above would probably be of this form. The general idea behind two level plans would be that the meta-level component represents important features of the plan, whereas the object level part contains more details, but is more likely to need to be changed. The application procedure would need to coordinate the two levels, while giving greater precedence to the meta-level. For example, the meta-level steps might be validated in turn; for each, the validation would first be tried with the object-level chunk corresponding to the meta-level step; if this failed uniform search could be tried for the particular meta-level step; if this failed a patch could be attempted to the meta-level plan. Thus the structuring of the plan would be used to add flexibility to the application process.

Again, this is only a rough discussion of a hypothetical situation; there are many variable factors which prevent us from being more precise. However, the model that we suggest is, once again, one of the existing analogy techniques being applied and co-ordinated across the different levels of description.

8.4 Summary

In this and previous chapters, we have discussed two main areas for extensions to the work of this thesis:

- Embedding the Basic APS system in a system able to use analogy in more ways than Basic APS alone.

- Improving the Basic APS system itself.

In this chapter, we have concentrated on the second of these two areas. We have identified limitations of the present system and indicated how they might be tackled. In the short term, work can be done to make the Basic APS system more flexible, particularly the application stage. The ability to learn from past analogies, as discussed in Chapter 7, would also be a useful and feasible extension in the short term.

In the long term, we believe that the most fruitful area for developing the analogy system will be in adding the ability to switch between representation levels, and all that this entails for the analogy process. We have argued that the techniques developed in this thesis will still be present in such powerful analogy systems; but the techniques will be applied to different levels of description, as necessary, and coordinated between the levels.

Analogy systems capable of finding and using analogies at multiple levels are a long term goal. Many difficult issues would need to be addressed, such as how re-representation (whether of the rewriting or the meta-description form) can be done automatically. It is understandable how such a system would be capable of powerful analogical reasoning — the more different ways in which you can look at a pair of problems, the better your chance of finding a correspondence between them. Some of the most difficult issues, those relating to multiple representations — their development, the connections between different levels, and their co-ordination with problem solving techniques — apply more generally than to just analogy. Development of analogy systems in this direction may have to wait until automated systems are available which represent even a portion of these capablities.

Bibliography

[Bledsoe 86] W Bledsoe. Appendix to grant application. 1986.

[Brown 77] R.M. Brown. *Use of Analogy to Acheive New Expertise.* Master's thesis, M.I.T., 1977.

[Bundy & Wallen 86] A. Bundy and L.A. Wallen. Proving properties of logic programs: summary of progress. In *Proceedings of Alvey Logic Programming Club*, 1986.

[Bundy 83] A. Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1983.

[Bundy 85] A. Bundy. Discovery and reasoning in mathematics. In *IJCAI-85*, 1985.

[Bundy 87] A Bundy. A correctness criterion for analogy matchers. 1987.

[Bundy et al 79] A. Bundy, L. Byrd, G. Luger, C. Mellish, R. Milne, and M. Palmer. *Solving Mechanics Problems Using Meta-level Inference.* Research Paper 112, Dept Artificial Intelligence, Edinburgh, 1979. In proceedings of IJCAI-6.

[Carbonnell 83a] J Carbonnell. Derivational analogy. In *AAAI-83*, 1983.

[Carbonnell 83b] J Carbonnell. Learning by analogy: formulating and generalizing plans from past experience. In *Machine Learning*, Tioga, 1983.

[Chang & Lee 73] Chang and Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[Constable & others 86] R.L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[Evans 68] (see foot of page)

[Gentner 82] D. Gentner. Are scientific analogies metaphors? In *Metaphor : Problems and Perspectives*, 1982.

[Gick & Holyoak 80] M. Gick and K. Holyoak. Analogical problem solving. *Cognitive Psychology*, 1980.

[Hesse 63] M.B. Hesse. *Models and Analogies in Science.* Notre-Dame, 1963.

[Indurkhya 85] B Indurkhya. *A Computational Theory of Metaphor Comprehension and Analogical Reasoning.* PhD thesis, Boston University, 1985.

[Kling 71] R.E. Kling. A paradigm for reasoning by analogy. *Artificial Intelligence*, 2, 1971.

[Miller 82] C. Miller. *Analogy and Mathematical Reasoning — A Survey.* Master's thesis, Edinburgh University, 1982.

[Mitchell 78] T.M. Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Stanford University, 1978.

[Evans 68] T.G. Evans. A Program for The Solution of Geometric-Analogy Intelligence Test Questions. 1968
In Semantic Information Processing, Minsky (ed

[Munyer 81] J.C. Munyer. *Analogy as a Means of Discovery in Problem-Solving and Learning.* PhD thesis, Univ. Calif. Santa Cruz, 1981.

[Owen 88] "Prolog Code for an Analogy System", AI Dept., Edinburgh Univ., 1988

[Plotkin 81] G Plotkin. Notes on focusing. 1981.

[Polya 54] G. Polya. *Induction and Analogy in Mathematics.* Princeton, 1954.

[Popper 68] K.R. Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge.* Harper-Row, 1968.

[Rumelhart & Abrahamson 73] Rumelhart and Abrahamson. A model for analogical reasoning. *Cognitive Psychology*, 7, 1973.

[Siekmann 86] (see foot of page)

[Silver 85] B. Silver. *Meta-level inference: Representing and Learning Control Information in Artificial Intelligence.* North-Holland, 1985.

[Sterling et al 82] L. Sterling, A. Bundy, L. Byrd, R. O'Keefe, and B. Silver. *Solving Symbolic Equations with Press.* Research Paper 171, Dept Artificial Intelligence, Edinburgh, 1982. In proceedings of EUROCAM-82.

[Sternberg 77] R. Sternberg. Component processes in analogical reasoning. *Cognitive Psychology*, 9, 1977.

[Utgoff 83] P.E. Utgoff. Adjusting bias in concept learning. In *Proceedings of the Eighth IJCAI*, International Joint Conference on Artificial Intelligence, 1983.

[Siekmann 86] J.H. Siekmann. Unification Theory. in Proceedings of ECAI-86, 1986

[Wielemaker & Bundy 85] J. Wielemaker and A. Bundy. *Altering the Description Space for Focusing.* Research Paper 262, Dept Artificial Intelligence, Edinburgh, 1985.

[Winston 80] P.H. Winston. Learning and reasoning by analogy. *Communications of the A.C.M.*, 23(12):689–703, December 1980.

Appendix A

Comparison of the three matchers

A.1 Introduction

Munyer's matcher, and the matcher developed in this thesis accept the examples given below in the form in which they are presented. Kling's matcher, however, only accepts formulae which do not contain function symbols. Therefore, the formulae have been transformed into a suitable form before presentation to Kling's matcher. For example, the formula

$$even(x) \longrightarrow x = 2 \cdot f(x)$$

is transformed to

$$(\cdot(x_3, 2, x_2) \wedge f(x_2, x) \wedge even(x)) \longrightarrow x = x_3$$

where $\cdot$ and f are now predicates. Any match obtained between the resulting formulae can be interpreted as one between the original formulae (the variables which have been introduced correspond to subterm positions; thus associations between these variables correspond to positional associations). The matches below are presented in this interpreted form, as this facilitates comparison with the other matchers.

For the abstract algebra examples (17-20) the semantic templates from [Kling 71] were used, although we also say what happens when the templates are weakened;

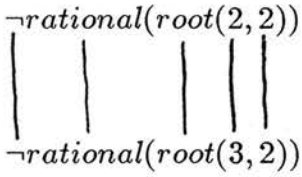
for the other examples, general semantic templates were chosen — $relfn$ for functional relations, rel for other relations and $prop$ for negation and where a relation is treated as a function due to being nested inside propositional functions.

We start with very simple examples, and gradually move on to more difficult ones. The philosophy is the same throughout: between each pair of formulae there is at least one correspondence which is, to a greater or lesser extent, intuitively obvious. We regard the matchers as being successful if they find the intended matches. As analogies become more obscure, this means of assessment becomes increasingly suspect. However, as will be seen below, the matchers are clearly distinguishable even on simple analogies. Therefore we claim that a more sophisticated analysis is not necessary in this case.

Choice of examples The examples used here have not been chosen specially to show the previous matchers in a bad light. They have been chosen as illustrative of the kind of simple analogies which we would expect a matcher to be able to find. For example, examples 2,3,5,6 and 7 are perfect isomorphisms from Boolean algebra. We have argued that any analogy matcher should be able to find the analogies here. Many other examples of this type could be listed, with similar results. Examples 4, 8 and 11 are variations on the same analogy (discussed in [Carbonnell 83b]) — again they have not been chosen carefully. Examples 3,9,10 and 12 are examples and variations from [Munyer 81]. Examples 15 and 16 are taken from [Silver 85], where they were discussed in the context of schema learning. Examples 17-21 are taken from [Kling 71], and include the examples around which Kling designed his algorithm. Other examples, which have been provided by the present author, have been chosen because they represent intuitively clear analogies, some of which have been used in the solution of target problems in Chapter 6.

A.2 Examples

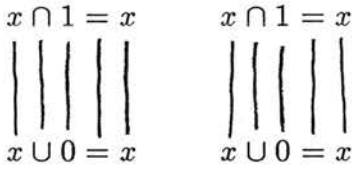
Example 1



All three matchers

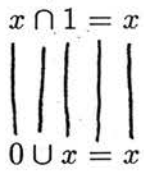
All three matchers find the match shown above, which is the intended one.

Example 2



Kling and Owen Munyer

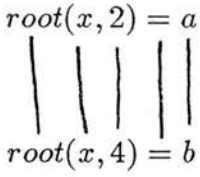
Kling's and Owen's[1] matchers find the intended match; Munyer's finds a different match which does not seem to represent an intuitive correspondence between the problems. Note that the symbols ∩, ∪ and = are commutative; if arguments to these are permuted, Munyer's and Owen's matchers would find the same symbolic correspondence, since they take the commutativity into account. However, Kling's matcher would find a different correspondence, for example



[1]In this appendix, we refer to the matcher described in Chapter 4 as Owen's matcher.

Whenever commutative symbols are present, Kling's matcher is very sensitive to the order of presentation; it will only find the intended match if the formulae are presented so as to make the match clear. We have frequently presented formulae in this way, in this thesis; but, for the other matchers, this has been purely for the sake of presentation; for Kling's, it is crucial to the matcher. This makes it difficult to compare Kling's matcher with the other two. For the remaining examples, we present Kling's matcher with the formulae in a form which makes the intended correspondence as clear as possible; the resulting matches produced by Kling's matcher should therefore be judged with this qualification in mind. Where this qualification is relevant, we label Kling's match with a "q".

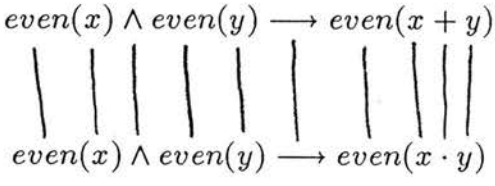
Example 3



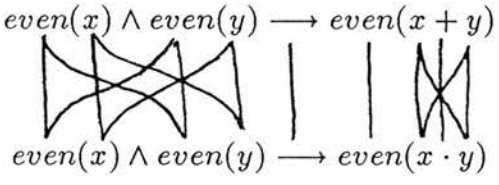
All three matchers

All matchers find the match shown above (remember, however, the qualification on Kling's result).

Example 4



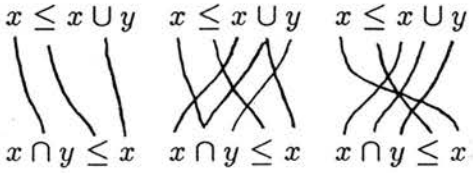
Kling(q) and Owen



Munyer

Owen's matcher also constructs the symmetric match in which x and y are swapped, and the arguments to $\cdot$ and $+$ are permuted, and then deletes it as being a variant of that shown above. Munyer's matcher again produces an unlikely match — it nearly constructs both plausible matches together, but fails on the conclusions.

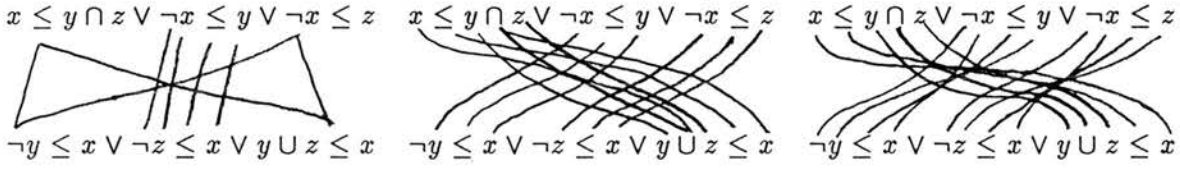
Example 5



Munyer Kling Owen

Munyer's matcher finds a plausible match, although not the intended one. Kling's matcher finds an implausible match. The current matcher finds the intended match.

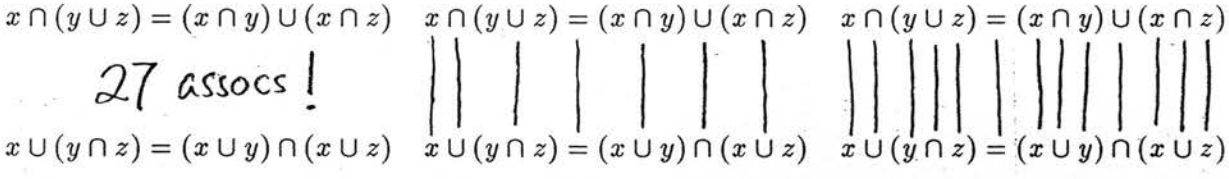
Example 6



Munyer Kling(q) Owen

Owen's matcher finds the intended, consistent match; the others both involve inconsistencies. Kling's suffers from being unable to permute arguments to $\leq$; Munyer's again seems to involve part of several possible matches.

Example 7

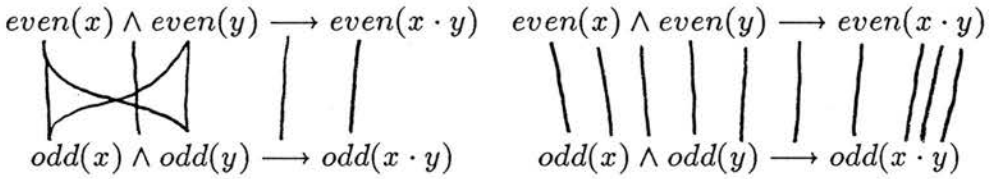


Munyer Kling(q) Owen

Munyer's match is completely implausible; Kling's is incomplete because of the deficiencies of MULTIMATCH1; Owen's matcher constructs two equally good

consistent isomorphisms, and discards one as being a variant. Note that Kling's match is particularly susceptible to the presentation here, with so many commutative symbols. In fact, the order of presentation dictates, in a very simple way, what match is constructed.

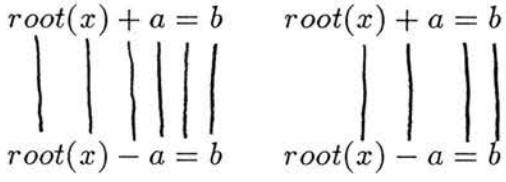
Example 8



Munyer Kling(q) and Owen

Munyer's match is again implausible; Kling constructs the match indicated by the presentation; Owen's matcher constructs both plausible matches and discards one as a variant.

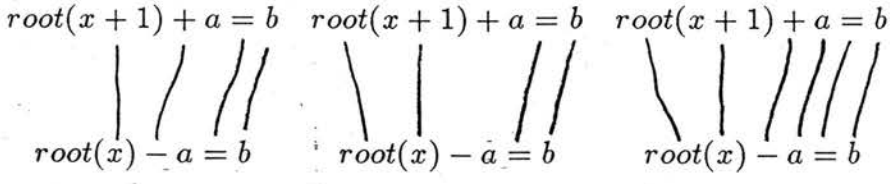
Example 9



Kling(q) and Owen Munyer

Kling and Owen find the intended match. Munyer's match lacks some associations.

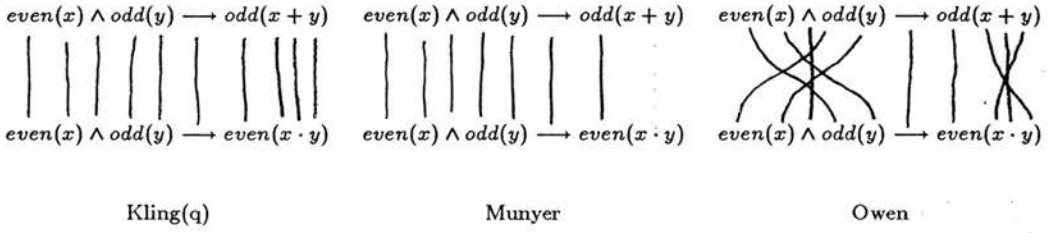
Example 10



Munyer Kling(q) Owen

Munyer's match is deficient in a similar way to the previous example. Kling's match lacks associations because of the SINGLEMATCH/MULTIMATCH distinction. Owen's matcher finds the intended match.

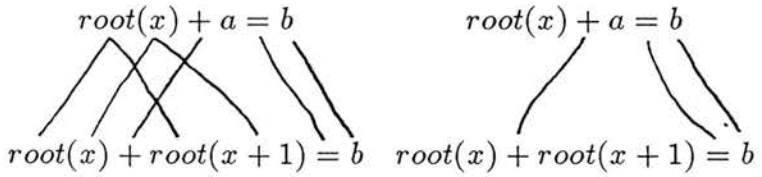
Example 11



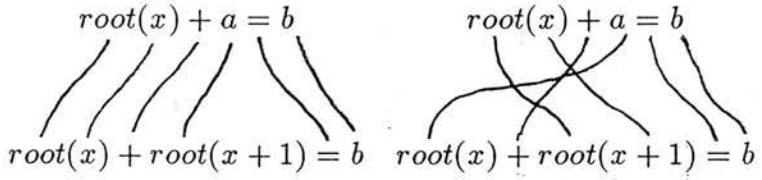
Kling(q) Munyer Owen

Munyer's match is along the right lines, but has some associations missing. In general, Munyer's matcher is bad at matching non-identical symbols, for the reasons explained in Chapter 3. Kling finds a plausible match, but it involves an inconsistency. Owen's matcher finds the intended, consistent match.

Example 12



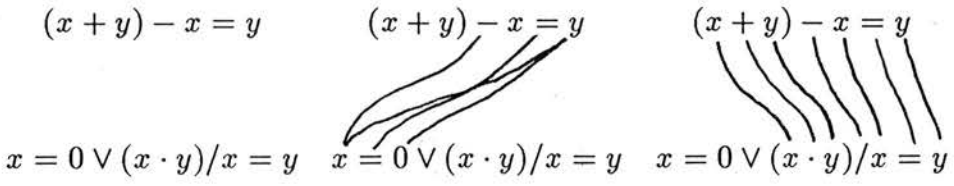
Munyer Kling



Owen Owen

In this example, intuitions are not so strong as to what is a sensible match (or if there is one at all). Kling and Munyer both find partial matches. Owen's matcher finds two plausible matches (of equal strength).

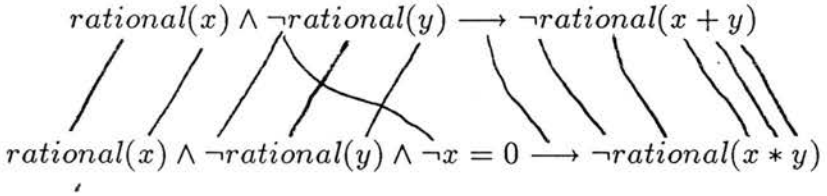
Example 13



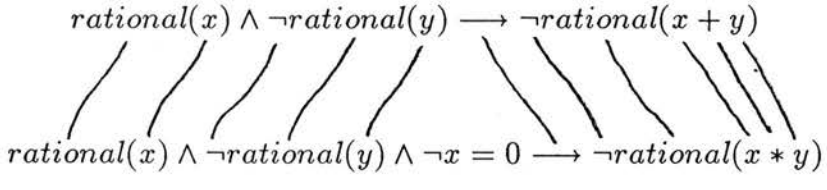
Kling Munyer Owen

Kling returns the null match[2]. Munyer's match is implausible. Owen's matcher returns the intended match, which involves an unmatched atom, $x = 0$, on one side.

Example 14



Munyer



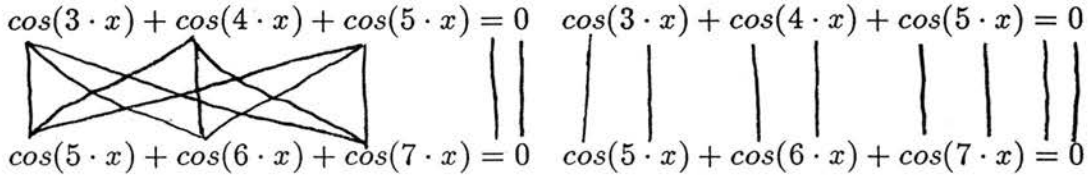
Owen

Kling's matcher fails on this example. Munyer's returns an implausible match. Owen's matcher finds the intended match; this is a consistent mapping, but leaves an unmatched condition, $x = 0$, on one side.

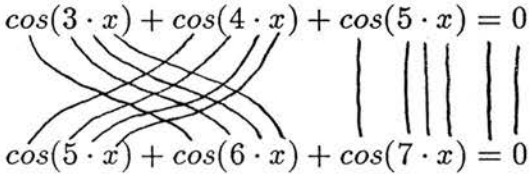
---

[2]For Kling's matcher, returning a null match is distinct from failing to return a match at all; this is not an interesting distinction, however.

Example 15



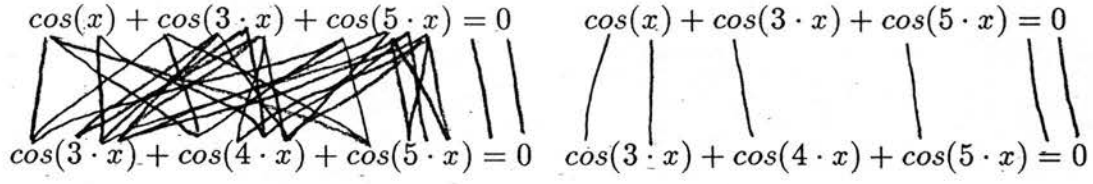
Munyer Kling



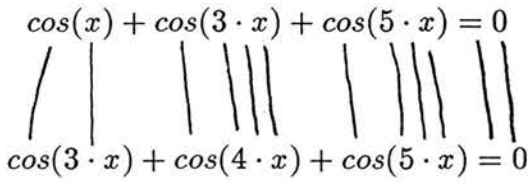
Owen(eg)

Munyer finds an implausible match; in this example, and in many others, Munyer's matcher finds matches which are unions of parts of plausible matches; but since there is no recognition of the different components, the entire match is implausible. Kling's match is partial. Owen's matcher actually finds six matches, of which that shown above is one. The other five correspond to the other five ways of pairing up the *cos* and *sin* terms; the two matches which pair 5 with itself are marginally preferred to the other, by the identical symbols heuristic. Note that, as explained in chapter 8, the successful matches would be the two which pair up the terms in corresponding arithmetic progression, but the matcher has no way of knowing this.

Example 16



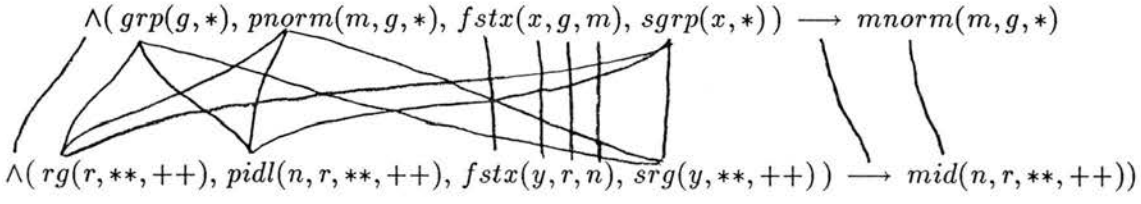
Munyer Kling(q)



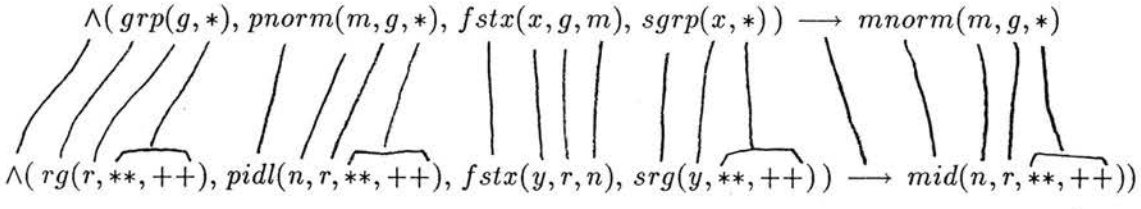
Owen(eg)

Munyer finds an implausible match. Kling finds part of a plausible match (the one indicated by the presentation), but is unable to complete it. Owen's matcher finds 12 matches; 6 of these correspond to those of the previous example, with x associated with itself wherever possible(such as (a) above); the other 6 have some occurrences of x bound to numbers. The first set have greater strength than the second and so are preferred.

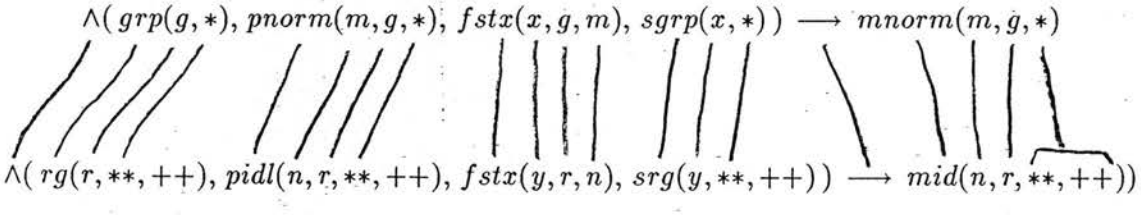
Example 17



Munyer



Kling



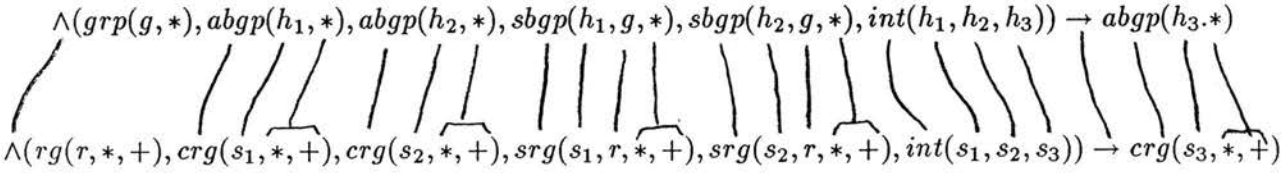
Owen (eg)

grp: group	*rg*: ring
pnorm: proper normal subgroup	*pidl*: proper ideal
sgrp: simple group	*srg*: simple ring
mnorm: maximal normal subgroup	*mid*: maximal ideal
fstx: factorstructure	

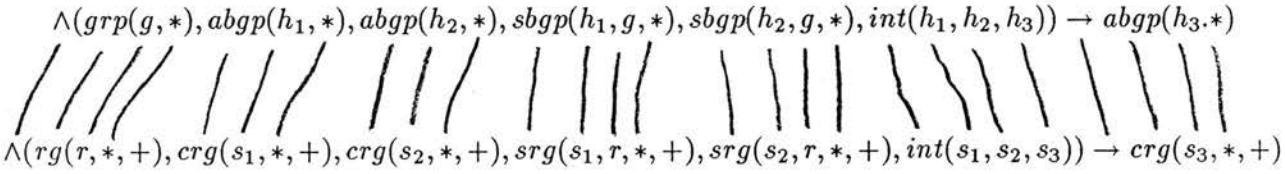
Munyer finds an implausible match. With the guidance of strong semantic templates, Kling finds the intended match. Owen's matcher finds two matches (displayed in chapter 4); these (one of which is shown above) are effectively equivalent to Kling's match.

The semantic templates consist of types for each predicates, and types for each argument position of each predicate. If the predicate type information is removed (i.e. all predicates are given the same type), Kling's matcher finds the same match. If the argument type information is also removed (i.e. if Kling's matcher is given no more guidance than Owen's matcher), the matcher fails.

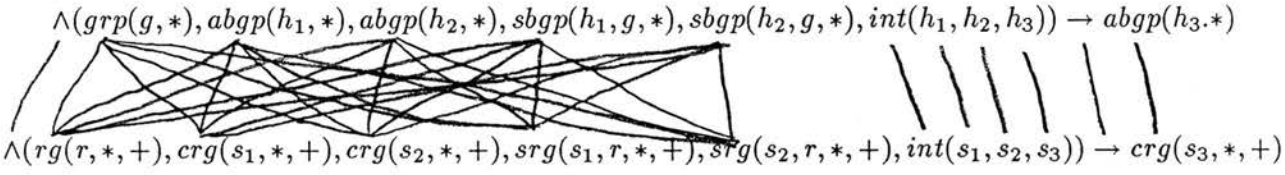
Example 18



Kling



Owen(eg)

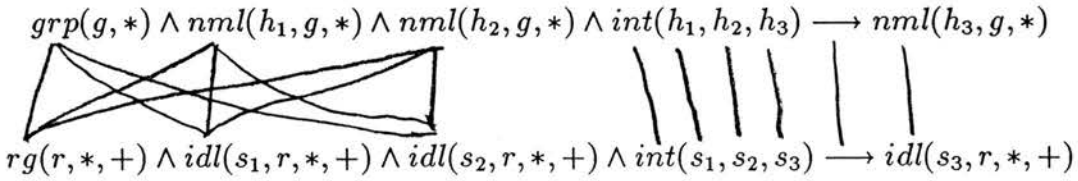


Munyer

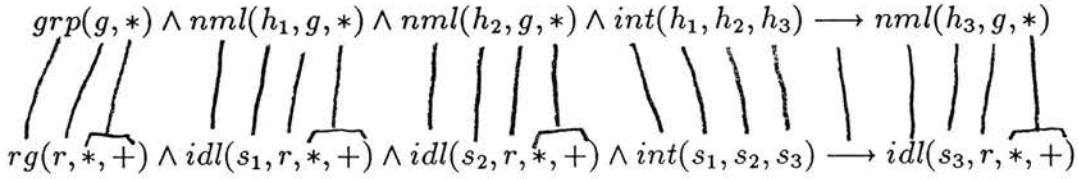
abgp: abelian group	*crg*: cummutative ring
pnorm: proper normal subgroup	*pidl*: proper ideal
sbgp: subgroup	*srg*: subring
int: intersection	

Kling's match is close to the intended one but lacks the expected $group \leftrightarrow ring$ association. Without predicate types, Kling's matcher fails. Munyer's match is implausible. Owen's matcher finds four matches, Of these two are equally strong; these are the intended matches (one is shown above, the other is the same except that $*$ is associated with $+$ instead of $*$). These two matches are the only consistent mappings in which all of the first formula is matched. The other two are weaker and involve some inconsistencies in the variable associations.

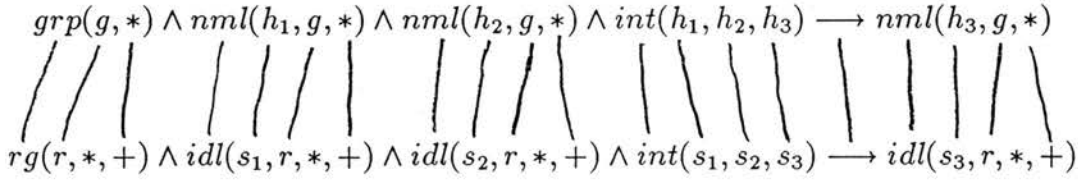
Example 19



Munyer



Kling

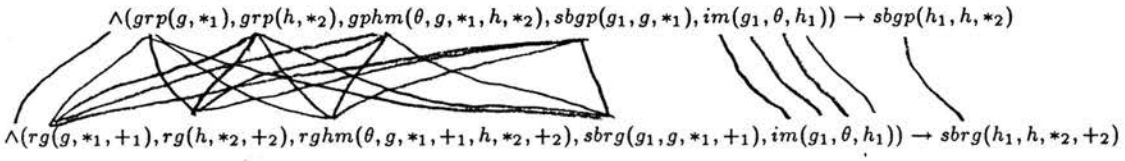


Owen(eg)

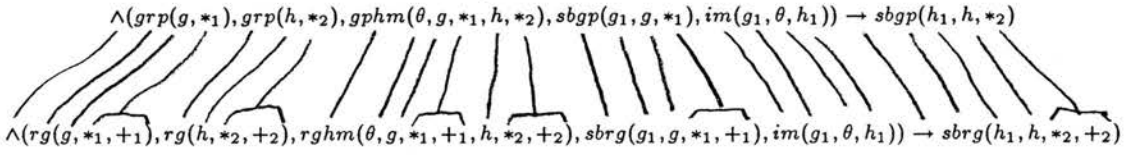
nml: normal subgroup *idl*: ideal

Munyer's match is again implausible. Kling finds the intended match. With just argument types, it still finds the intended match, but without these it fails. Owen's matcher finds six matches, one of which is shown above. These matches are equally strong — they are the six consistent mappings which map all of the first formula. Without argument type information and preference, these matches cannot be distinguished further.

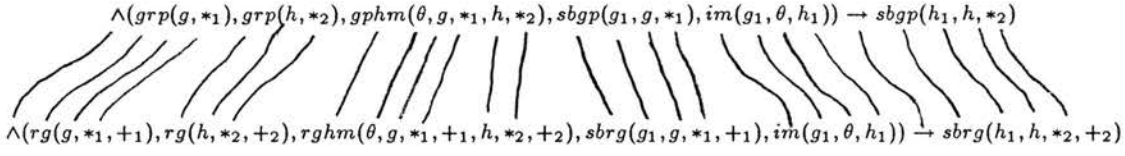
Example 20



Munyer



Kling

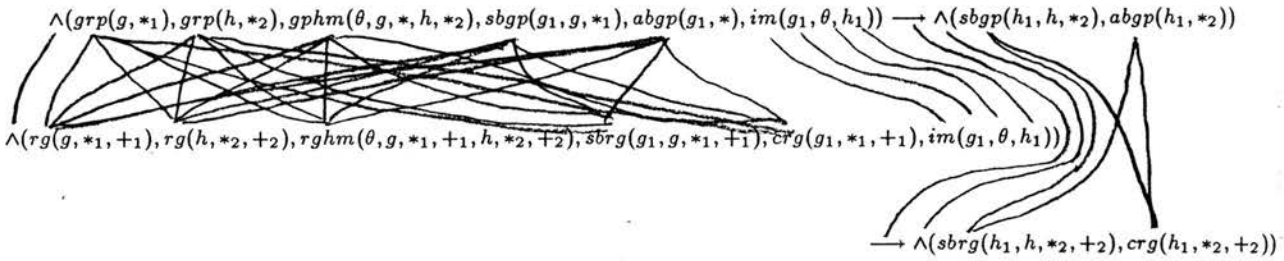


Owen (eg)

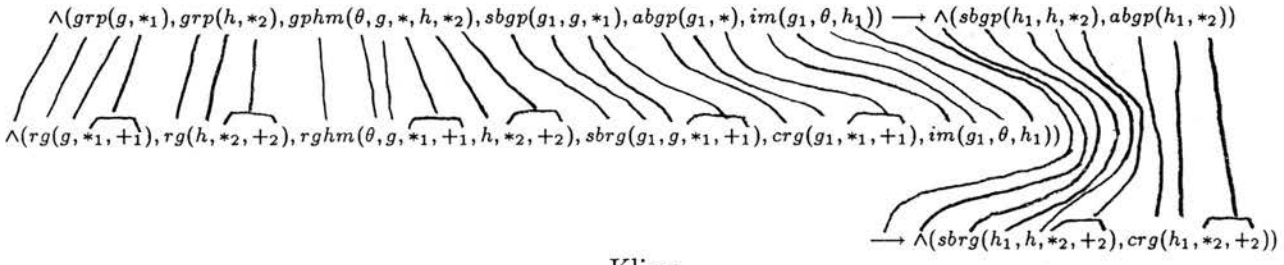
gphm: group homomorphism *rghm*: ring homomorphism
im: image

Kling finds the intended match. Without predicate types he finds the same match; without argument types as well, he fails. Munyer's match is again implausible. Owen's matcher again finds six matches, one of which is shown above, which once again represent the six consistent mappings which map the whole of the first term. Once again, these are equally strong, unless argument types are used to distinguish between them.

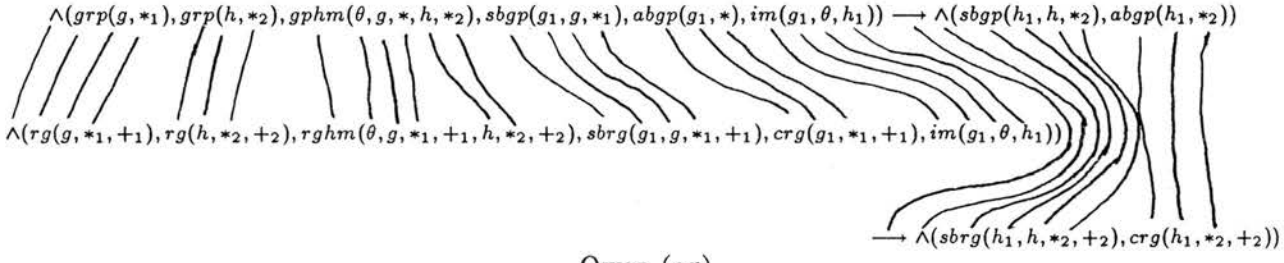
Example 21



Munyer



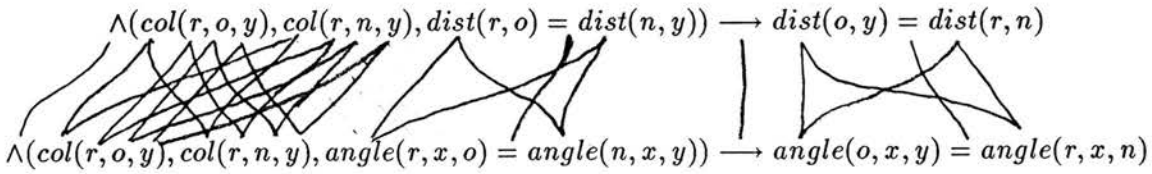
Kling



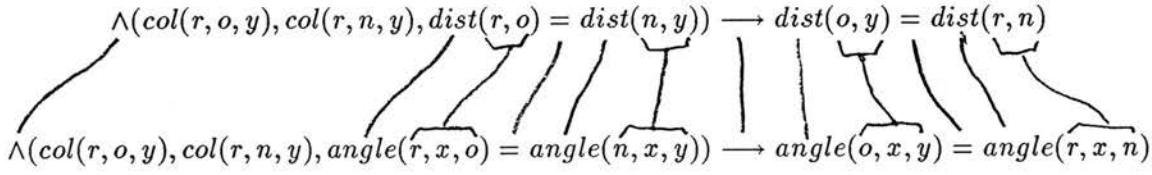
Owen (eg)

Munyer's match is again implausible. Kling finds the intended match. Owen's matcher finds six matches. Two of these are stronger than the rest, and are the intended matches, corresponding to Kling's. The other four are weaker, involving inconsistent variable associations. Without predicate types, Kling's matcher fails.

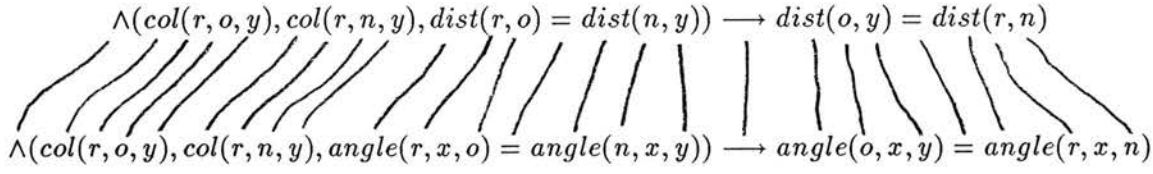
Example 22



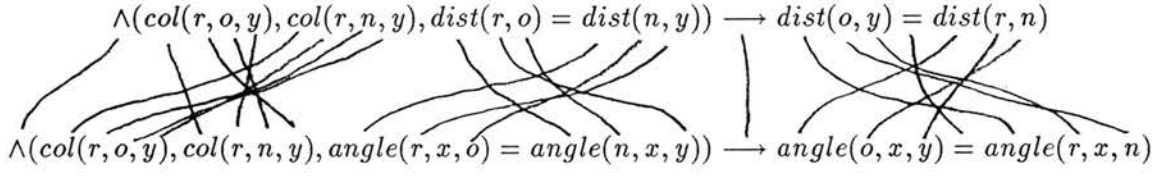
Munyer



Kling



Owen

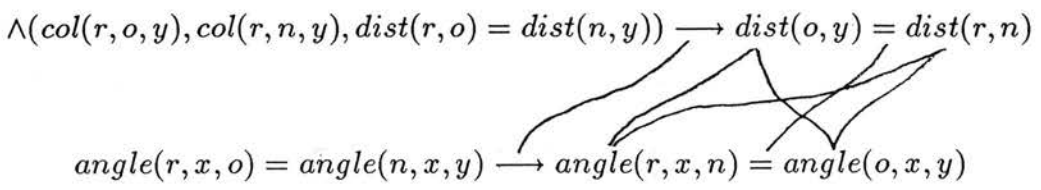


Owen

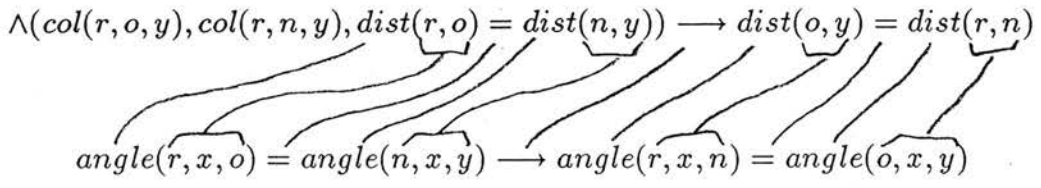
dist: distance *angle*: angle
col: colinear

Munyer's match is implausible. Kling's is along the right lines, but his matcher fails to develop the match to the point of variable associations. Owen's matcher finds two matches, which are the intended ones. Once again, they are the only two consistent mappings which map the whole of the first term.

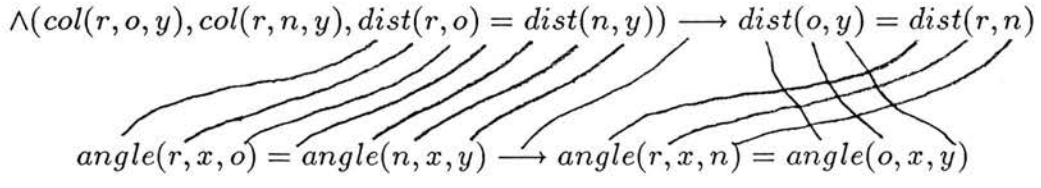
Example 23



Munyer



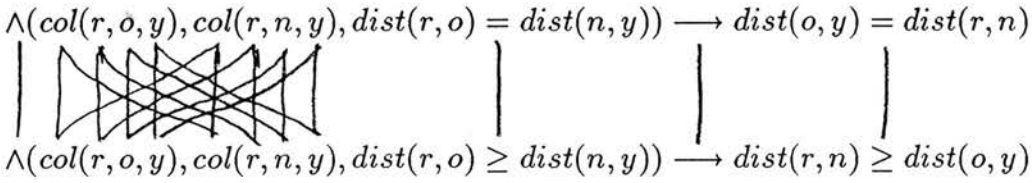
Kling



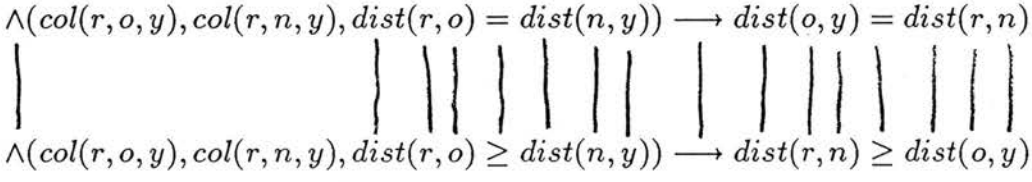
Owen (eg)

Munyer's match is implausible. Kling's match again does not include variable associations. Owen's matcher finds 11 matches, four of which are stronger than the rest, and equally strong as each other; one of these is shown above. The other strong matches contain different (but still consistent) variable mappings. As with other examples, it was one of these strong matches which was intended. However, we must accept that the other three are equally good. The weaker matches contain inconsistent variable associations.

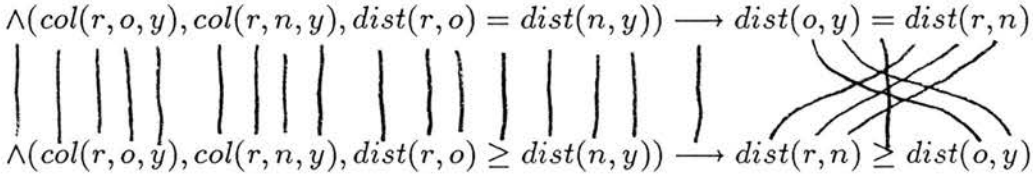
Example 24



Munyer



Kling



Owen

Munyer's match is implausible. Kling's is along the right lines but incomplete. Owen's matcher finds one match, which is the intended one.

A.3 Conclusion

The examples given in this appendix illustrate the superiority of Owen's matcher over the previous ones. The flexibility and persistence of Owen's matcher enables it to construct intuitively correct matches in a wide range of examples (and to find other, equally good, matches which had not been foreseen). We have already discussed the deficiencies of the previous matchers in depth; we merely remark here how the major ones have been illustrated by the examples of this appendix:

Munyer

- Inability to construct plausible matches involving significant proportions of non-identical associations (examples 2,5-8,13,17-24).

- Inability to consider more than one match between a pair of terms (examples 4,6,7,8,12,15,16,17-24).

Kling

- Inability to permute arguments inside atoms (examples 5,6 and all those marked "q").

- Rigid use of semantic types (examples 10,12,13,14).

Appendix B

Sample output of matcher and application system

In this appendix, we give some sample output from the analogy matcher and one of the application systems developed in this thesis. The application system we illustrate is system D of Chapter 6. The text is exactly that produced by the systems with the following qualifications:

- We have annotated the text with explanatory comments, preceded by the sign %.
- Lists of associations in matches have been alligned to improve readability.
- Some sections have been removed, where they are repetitive; this is indicated by text within angular brackets < .... >.

B.1 Matcher

We give output from two examples, examples 14 and 17 from Appendix A.

Example 14 In this example, matching is completed with a single development of the top-level and-agenda. Each match found (in this case one) is presented with its strength and its constituent associations. For each association, a list of argument pairings is returned. The process of match development for this example is described in detail in Section 4.4.

```
MATCHING
     imply(&(rational(x), not(rational(y))), not(rational(x + y)))
AND
     imply(&(rational(x), not(rational(y)),
                                not(x = 0)), not(rational(x * y)))

MATCHING COMPLETE

1  match.

Match 1 :
               Strength  0.84965;    Assocs  12
Mapping :
        Association          Argument pairings
         + <----> *          [[p(2, 2), p(1, 1)]]
         y <----> y          [term]
       not <----> not        [[p(1, 1)]]
         x <----> x          [term]
  rational <----> rational   [[p(1, 1)]]
         & <----> &          [[p(1, 1), p(2, 2), p(u, p(3, not(x = 0)))]]
     imply <----> imply      [[p(1, 1), p(2, 2)]]
```

Example 17 This example takes several stages of development of the and-agenda before matching is completed. The partial match is split in the last stage, and leads to two completed matches. Again, the details of the development are given in Section 4.4.

```
MATCHING
     imply(&(group(x, w1), propernormal(y, x, w1),
               factorstructure(z, x, y), simplegroup(z, w1)),
                   maximalgroup(y, x, w1))
AND
     imply(&(ring(x, w1, w2), properideal(y, x, w1, w2),
               factorstructure(z, x, y), simplering(z, w1, w2)),
                         maximalring(y, x, w1, w2))

DEVELOPING THE MATCHING AGENDA .....
% matching takes more than one stage
NEW STATE OF AGENDA:

Best completed matches so far:
                                % no complete matches yet
```

```
Partial matches so far:
                                % one partial match
Match 1 :                       % with two associations
              Strength  1.0;     Assocs  2
---------------------------------
DEVELOPING THE MATCHING AGENDA .....
                                % develop partial match further
NEW STATE OF AGENDA:

Best completed matches so far:
                                % still no complete match
Partial matches so far:
                                % seven associations added
Match 1 :
              Strength  0.988889;    Assocs  9
---------------------------------
DEVELOPING THE MATCHING AGENDA .....

NEW STATE OF AGENDA:

Best completed matches so far:

Partial matches so far:

Match 1 :
              Strength  0.975;    Assocs  16
---------------------------------
DEVELOPING THE MATCHING AGENDA .....

NEW STATE OF AGENDA:          % match split into two, and both
                              % offspring completed
Best completed matches so far:

Match 1 :
              Strength  0.934546;    Assocs  20

Match 2 :
              Strength  0.934546;    Assocs  20

Partial matches so far:       % no more partials, so matching complete
---------------------------------
MATCHING COMPLETE
```

```
2 matches.

Match 1 :
              Strength  0.934546;    Assocs  20
Mapping :
            Association                Argument pairings
            w1 <----> w2               [term]
          group <----> ring            [[p(1, 1), p(2, 3),
                                          p(u, p(2, w1))]]
   propernormal <----> properideal     [[p(2, 2), p(1, 1), p(3, 4),
                                          p(u, p(3, w1))]]
    simplegroup <----> simplering      [[p(1, 1), p(2, 3),
                                          p(u, p(2, w1))]]
   maximalgroup <----> maximalring     [[p(2, 2), p(1, 1), p(3, 4),
                                          p(u, p(3, w1))]]
              z <----> z               [term]
              x <----> x               [term]
              y <----> y               [term]
factorstructure <----> factorstructure [[p(3, 3), p(2, 2), p(1, 1)]]
              & <----> &               [[p(3, 3), p(4, 4), p(2, 2),
                                          p(1, 1)]]                x
          imply <----> imply           [[p(1, 1), p(2, 2)]]

Match 2 :
              Strength  0.934546;    Assocs  20
Mapping :
            Association                Argument pairings
            w1 <----> w1               [term]
          group <----> ring            [[p(1, 1), p(2, 2),
                                          p(u, p(3, w2))]]
   propernormal <----> properideal     [[p(2, 2), p(1, 1), p(3, 3),
                                          p(u, p(4, w2))]]
    simplegroup <----> simplering      [[p(1, 1), p(2, 2),
                                          p(u, p(3, w2))]]
   maximalgroup <----> maximalring     [[p(2, 2), p(1, 1), p(3, 3),
                                          p(u, p(4, w2))]]
              z <----> z               [term]
              x <----> x               [term]
              y <----> y               [term]
factorstructure <----> factorstructure [[p(3, 3), p(2, 2), p(1, 1)]]
              & <----> &               [[p(3, 3), p(4, 4), p(2, 2),
```

```
                                                          p(1, 1)]]
      imply <----> imply              [[p(1, 1), p(2, 2)]]
```

B.2 Application system D

We show the output from Problem 3 of Chapter 6, which was to prove

$$\neg rational(x) \vee rational(y) \vee x = 0 \vee \neg rational(x \cdot y)$$

by analogy to a given proof of

$$\neg rational(x) \vee rational(y) \vee \neg rational(x + y).$$

Figure 6–5, which illustrates the search of all the application systems, may be followed while reading the output below.

As Figure 6–5 shows, system D only solves the target problem after some search. The wrong axiom is chosen first at Step 1; this leads to local failure at Step 3; on backtracking, the same axiom (still wrong) is applied in a different way at Step 1, and likewise leads to local failure at Step 3; on backtracking this time, the correct axiom is chosen at Step 1, which leads to a solution without further incorrect steps begin made. Note the applications of the AOI, involving the axiom filter and then full matching, and that the AOI is called again at a step which is returned to after backtracking. Note also the applications of the special unification procedure, between candidate inferences and plan steps; it is this which is effective in controling the search.

```
MATCHING PROBLEM STATEMENTS ......

INITIAL MATCHING TOOK :
     2900ms

1 MATCH FOUND

Match 1 with present :
               Strength  0.850909;    Assocs  11
Mapping :

       Association          Argument pairings
       x <----> x           [term]
       + <----> *           [[p(2, 2), p(1, 1)]]
     not <----> not         [[p(1, 1)]]
```

```
       y <----> y          [term]
     rat <----> rat        [[p(1, 1)]]
      or <----> or         [[p(2, 2), p(3, 4), p(1, 1),
                                        p(u, p(3, x = 0))]]

NOW TRY TO USE STRONGEST MATCH TO PROVE TARGET

SKOLEM PART:                          % partition and negate the initial
       x <----> x          [term]     % mapping
       y <----> y          [term]
NON-SKOLEM PART:
       + <----> *          [[p(2, 2), p(1, 1)]]
     not <----> not        [[p(1, 1)]]
     rat <----> rat        [[p(1, 1)]]
      or <----> or         [[p(2, 2), p(3, 4), p(1, 1),
                                        p(u, p(3, x = 0))]]
NEGATED CLAUSE: &(rat(skx), not(rat(sky)), not(skx = 0), rat(skx * sky))
NEGATED MAPPING:
       a <----> skx        [term]
       b <----> sky        [term]

ATTEMPTING TO REFUTE:
     &(rat(skx), not(rat(sky)), not(skx = 0), rat(skx * sky))

FIRST STEP

Step 1                              % Exact analogue found for first axiom
EXACT ANALOGUE                      % hence no need to call AOI yet
     or(not(rat(x)), not(rat(y)), rat(x - y)) % NB WRONG AXIOM CHOSEN
FOUND FOR BASE AXIOM                          % LEADS TO DEAD END
     or(not(rat(x)), not(rat(y)), rat(x - y)) % AT STEP 3
HENCE TRY IT FIRST ....

Step 1
TRYING AXIOM:   or(not(rat(x)), not(rat(y)), rat(x - y))

     or(not(rat(y)), rat(skx * sky - y))
DERIVED BY binres BETWEEN                     % axiom applies, and result
     or(rat(skx * sky))                       % matches plan step (note
            AND                               % special syntax in plan
step
     or(not(rat(x)), not(rat(y)), rat(x - y))   % step)
```

```
AND MATCHES WITH PLAN STEP
     or(not(rat(y)), rat(def(skx * sky - y)), bdef(x109 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....

MOVING TO STEP 2                                % try next step of plan
                                                % which does not involve
     or(rat(skx * sky - skx))                   % an axiom -
DERIVED BY binres BETWEEN                       % goes through
     or(rat(skx))
             AND
     or(not(rat(y)), rat(skx * sky - y))
AND MATCHES WITH PLAN STEP
     or(rat(skx * sky - skx), bdef(x110 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....

MOVING TO STEP 3

APPLYING AXIOM FILTER ....              % no exact analogue, so AOI called

CANDIDATE ANALOGUES FOR:                % first the filter
     or(x + y - x = y):
          Strength: 3     Axiom: or(x * y = y * x)
          Strength: 3     Axiom: or(x * y = x * y)
          Strength: 3     Axiom: or(x + y - x = y)
          Strength: 3     Axiom: or(meet(x, - x) = 0)
          Strength: 2     Axiom: or(exp(x, 2) = x * x)
          Strength: 2     Axiom: or(not(divides(x, y)),
                                        y = x * h(x, y))
          Strength: 2     Axiom: or(x = 0, x * y / x = y)
          Strength: 2     Axiom: or(join(u, - u) = 1)
          Strength: 0     Axiom: or(x * 1 = x)
          Strength: -1    Axiom: or(x - y = x + min(y))

NOW USE THE FULL MATCHER ON THE CANDIDATES ....

3 MATCHES PRODUCED

Match 1 with or(x + y - x = y) :
               Strength  0.883871;    Assocs  7
Mapping :

        Association         Argument pairings
```

```
        + <----> +          [[p(2, 2), p(1, 1)]]
        x <----> x          [term]
        y <----> y          [term]
        = <----> =          [[p(1, 1), p(2, 2)]]
        + <----> *          [[p(1, 1), p(2, 2)]]
      not <----> not        [[p(1, 1)]]
      rat <----> rat        [[p(1, 1)]]
       or <----> or         [[p(u, p(3, x = 0)), p(1, 1),
                                        p(3, 4), p(2, 2)]]
        a <----> skx        [term]
        b <----> sky        [term]
        - <----> -          [[p(1, 1), p(2, 2)]]


Match 2 with or(x = 0, x * y / x = y) :
              Strength  0.779924;    Assocs  8
Mapping :

        Association         Argument pairings
       x <----> x           [term]
       - <----> /           [[p(1, 1), p(2, 2)]]
       y <----> y           [term]
       = <----> =           [[p(1, 1), p(2, 2)]]
       + <----> *           [[p(2, 2), p(1, 1)]]
     not <----> not         [[p(1, 1)]]
     rat <----> rat         [[p(1, 1)]]
      or <----> or          [[p(1, 2), p(u, p(1, x = 0))],
                             [p(u,p(3,x=0)),p(1,1),p(3,4),p(2,2)]]
       a <----> skx         [term]
       b <----> sky         [term]
       - <----> -           [[p(1, 1), p(2, 2)]]


Match 3 with or(x * 1 = x) :
              Strength  0.779109;    Assocs  6
Mapping :

         Association        Argument pairings
        x <----> 1          [term]
x + y - x <----> x * 1      [mismatch]
        y <----> x          [term]
        = <----> =          [[p(1, 1), p(2, 2)]]
```

```
        + <----> *          [[p(2, 1), p(1, 2)], [p(1, 1), p(2, 2)]]
      not <----> not        [[p(1, 1)]]
      rat <----> rat        [[p(1, 1)]]
       or <----> or         [[p(u, p(3, x = 0)), p(1, 1), p(3, 4), p(2,
2)]]
        a <----> skx        [term]
        b <----> sky        [term]
        - <----> -          [[p(1, 1), p(2, 2)]]

Step 3
TRYING AXIOM:   or(x + y - x = y)

     or(rat(skx * sky - (x + skx - x)))
DERIVED BY paramod BETWEEN          % axiom applies, but result does
     or(rat(skx * sky - skx))       % not match plan step, hence reject
             AND
     or(x + y - x = y)
BUT FAILS SPECIAL UNIFICATION WITH PLAN STEP
     or(rat(sky), bdef(x113 = 0))
HENCE WE REJECT THIS STEP
TRY AGAIN WITH SAME AXIOM .....


     or(rat(skx * (x + sky - x) - skx))
DERIVED BY paramod BETWEEN
     or(rat(skx * sky - skx))
             AND
     or(x + y - x = y)
BUT FAILS SPECIAL UNIFICATION WITH PLAN STEP  % same thing happens again
     or(rat(sky), bdef(x113 = 0))
HENCE WE REJECT THIS STEP
TRY AGAIN WITH SAME AXIOM .....

    <THREE MORE ATTEMPTS WITH SAME AXIOM ALL FAIL TO MATCH>

Step 3
NO MORE POSSIBLE INFERENCES WITH CURRENT AXIOM - TRY NEXT ONE

Step 3
TRYING AXIOM:   or(x = 0, x * y / x = y)

    <TEN ATTEMPTS WITH THIS AXIOM ALL FAIL TO MATCH PLAN STEP>
```

```
Step 3
NO MORE POSSIBLE INFERENCES WITH CURRENT AXIOM - TRY NEXT ONE

Step 3
TRYING AXIOM:   or(x * 1 = x)

    <FIVE ATTEPMTS WITH THIS AXIOM ALL FAIL TO MATCH PLAN STEP>

Step 3
NO MORE POSSIBLE INFERENCES WITH CURRENT AXIOM - TRY NEXT ONE
Step 3
NO MORE AXIOMS TO TRY AT THIS STEP - LOCAL FAILURE -  % all three axioms
          BACKTRACK TO PREVIOUS STEP                  % tried and failed
                                                % (because of wrong axiom
                                                % choice at Step 1
RETURNING TO STEP 2

Step 2
NO MORE AXIOMS TO TRY AT THIS STEP - LOCAL FAILURE -  % no more possibili
          BACKTRACK TO PREVIOUS STEP                  % here; i.e. infere
                                              % originally made at Step 1
has
RETURNING TO STEP 1                           % lead to failure

     or(not(rat(x)), rat(x - skx * sky))
DERIVED BY binres BETWEEN                       % another application of
     or(rat(skx * sky))                         % the exact analogue at
             AND                                % step 1 is successful
     or(not(rat(x)), not(rat(y)), rat(x - y))
AND MATCHES WITH PLAN STEP
     or(not(rat(y)), rat(def(skx * sky - y)), bdef(x109 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....        % unfortunately, the wrong
                                                % axiom can be applied in
MOVING TO STEP 2                                % another way; this will
                                                % again lead to failure
     or(rat(skx - skx * sky))
DERIVED BY binres BETWEEN
     or(rat(skx))
             AND
     or(not(rat(y)), rat(y - skx * sky))
AND MATCHES WITH PLAN STEP                          % goes through
```

```
     or(rat(skx - skx * sky), bdef(x126 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....

MOVING TO STEP 3

APPLYING AXIOM FILTER ....

CANDIDATE ANALOGUES FOR:
     or(x + y - x = y):
          Strength: 3     Axiom: or(x * y = y * x)
          Strength: 3     Axiom: or(x * y = x * y)
          Strength: 3     Axiom: or(x + y - x = y)
          Strength: 3     Axiom: or(meet(x, - x) = 0)
          Strength: 2     Axiom: or(exp(x, 2) = x * x)
          Strength: 2     Axiom: or(not(divides(x, y)),
                                                y = x * h(x, y))
          Strength: 2     Axiom: or(x = 0, x * y / x = y)
          Strength: 2     Axiom: or(join(u, - u) = 1)
          Strength: 0     Axiom: or(x * 1 = x)
          Strength: -1    Axiom: or(x - y = x + min(y))

NOW USE THE FULL MATCHER ON THE CANDIDATES ....

3 MATCHES PRODUCED

   < SAME AXIOMS AND SAME MATCHES PRODUCED AS AT STEP 3 BEFORE.
    AGAIN, EACH IS TRIED MULTIPLY AND FAILS TO MATCH WITH PLAN STEP
>

Step 3
NO MORE AXIOMS TO TRY AT THIS STEP - LOCAL FAILURE -  % another local
          BACKTRACK TO PREVIOUS STEP                  % failure as a result
of
                                                      % wrong axiom being
RETURNING TO STEP 2                                   % chosen at Step 1

Step 2
NO MORE AXIOMS TO TRY AT THIS STEP - LOCAL FAILURE -
          BACKTRACK TO PREVIOUS STEP

RETURNING TO STEP 1
```

```
EXACT ANALOGUE UNSUCCESSFUL, HENCE LOOK FOR OTHERS .....
                                        % now AOI called to look
APPLYING AXIOM FILTER ....              % for alternative analogues
                                        % for first base axiom

CANDIDATE ANALOGUES FOR:
     or(not(rat(x)), not(rat(y)), rat(x - y)):
       Strength: 10  Axiom: or(not(rat(x)), not(rat(y)), rat(x - y))
       Strength: 9   Axiom: or(not(rat(x)), not(rat(y)),
                                                  y = 0, rat(x / y))
       Strength: 9   Axiom: or(not(rat(x)), rat(y), not(rat(x + y)))

            % NB exact analogue returned by filter, but removed before
            % full matching as already has been tried

NOW USE THE FULL MATCHER ON THE CANDIDATES ....

2 MATCHES PRODUCED                      % the first of these turns out to
                                        % be correct
Match 1 with or(not(rat(x)), not(rat(y)), y = 0, rat(x / y)) :
              Strength  0.830909;    Assocs  11
Mapping :

        Association       Argument pairings
       y <----> y         [term]
       x <----> x         [term]
       - <----> /         [[p(1, 1), p(2, 2)]]
       + <----> *         [[p(1, 1), p(2, 2)]]
     not <----> not       [[p(1, 1)]]
     rat <----> rat       [[p(1, 1)]]
      or <----> or        [[p(3, 4), p(2, 2), p(1, 1),p(u,p(3,y=0))],
                          [p(u,p(3,x=0)), p(1, 1), p(3, 4), p(2, 2)]]
       a <----> skx       [term]
       b <----> sky       [term]


Match 2 with or(not(rat(x)), not(rat(y)), y = 0, rat(x / y)) :
              Strength  0.830909;    Assocs  11
Mapping :

        Association       Argument pairings
       y <----> x         [term]
       x <----> y         [term]
```

```
      - <----> /          [[p(1, 2), p(2, 1)]]
      + <----> *          [[p(1, 1), p(2, 2)]]
    not <----> not        [[p(1, 1)]]
    rat <----> rat        [[p(1, 1)]]
     or <----> or         [[p(3, 4), p(2, 1), p(1, 2), p(u,p(3,y=0))],
                          [p(u,p(3,x=0)), p(1, 1), p(3, 4), p(2, 2)]]
      a <----> skx        [term]
      b <----> sky        [term]

Step 1
TRYING AXIOM:   or(not(rat(x)), not(rat(y)), y = 0, rat(x / y))

     or(not(rat(y)), y = 0, rat(skx * sky / y)) % the right inference
DERIVED BY binres BETWEEN                       % is finally made at the first
     or(rat(skx * sky))                         % step - after this it's easy
             AND
     or(not(rat(x)), not(rat(y)), y = 0, rat(x / y))
AND MATCHES WITH PLAN STEP
     or(not(rat(y)), rat(def(skx * sky - y)), bdef(x142 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....

MOVING TO STEP 2

     or(rat(skx * sky / skx), skx = 0)          % right step is made
DERIVED BY binres BETWEEN                       % first time for this
and
     or(rat(skx))                               % and next step
             AND
     or(not(rat(y)), rat(skx * sky / y), y = 0)
AND MATCHES WITH PLAN STEP
     or(rat(skx * sky / skx), bdef(x144 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....

MOVING TO STEP 3

APPLYING AXIOM FILTER ....

CANDIDATE ANALOGUES FOR:
     or(x + y - x = y):
          Strength: 7     Axiom: or(x = 0, x * y / x = y)
          Strength: 3     Axiom: or(x * y = y * x)
          Strength: 3     Axiom: or(x / y = x * inv(y), y = 0)
```

```
        Strength: 3     Axiom: or(x / y = x / y)
        Strength: 3     Axiom: or(x * y = x * y)

NOW USE THE FULL MATCHER ON THE CANDIDATES ....

1 MATCH PRODUCED                                  % single candidate
                                                  % returned is the right
Match 1 with or(x = 0, x * y / x = y) :           % one
              Strength  0.843879;    Assocs  8
Mapping :

       Association      Argument pairings
     x <----> x         [term]
     y <----> y         [term]
     = <----> =         [[p(1, 1), p(2, 2)]]
     + <----> *         [[p(2, 2), p(1, 1)]]
   not <----> not       [[p(1, 1)]]
   rat <----> rat       [[p(1, 1)]]
    or <----> or        [[p(1, 2), p(u, p(1, x = 0))],
                        [p(u,p(3,x=0)), p(1, 1), p(3, 4), p(2, 2)]]
     a <----> skx       [term]
     b <----> sky       [term]
     - <----> /         [[p(1, 1), p(2, 2)]]

Step 3
TRYING AXIOM:   or(x = 0, x * y / x = y)

    or(rat(sky), skx = 0, skx = 0)
DERIVED BY paramod BETWEEN                % step 3 goes through this time
    or(rat(skx * sky / skx), skx = 0)     % right inference is made first
            AND
    or(x = 0, x * y / x = y)
AND MATCHES WITH PLAN STEP
    or(rat(sky), bdef(x148 = 0))
HENCE WE MAKE THIS STEP AND MOVE ON ....

MOVING TO STEP 4

    or(skx = 0, skx = 0)                  % again right inference
DERIVED BY binres BETWEEN                 % is made, leaving just
    or(not(rat(sky)))                     % the extra conditions
to
```

```
            AND                          % be refuted; i.e. the plan
has
     or(rat(sky), skx = 0, skx = 0)      % been applied; the proof is
AND MATCHES WITH PLAN STEP               % now finfished off with
     [or(bdef(x151 = 0)), nil]           % breadth-first search
HENCE WE MAKE THIS STEP AND MOVE ON ....

or(skx = 0)                          % proof finished off by search
  derived by binres

nil
  derived by binres

SUCCESS !! - ANALOGICAL PLAN VALIDATED

APPLICATION TOOK : 207650ms
```

HEURISTICS FOR ANALOGY MATCHING

Stephen Owen

Dept. Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland

Abstract

We analyse the heuristics that underlie analogy matching algorithms, in particular the matchers of Munyer and Kling. These matchers are reconstructed, tested, modified and assessed, and the underlying heuristics are abstracted. We propose a more principled approach to analogy matching, involving flexible application of the heuristic criteria.

Keywords

Analogy, Partial Matching, Problem Solving.

1 Introduction - Motivation for Analogy

The desirability of having computers which can reason by analogy has been argued for many times in the AI literature, usually by reference to the usefulness of the analogical abilities of humans. My area of interest is analogical reasoning within automated or interactive problem solvers working within mathematics. Given a problem to solve which is suitably similar or analogous to one which has previously been solved, such a system would use its experience of solving the previous problem to help it solve the given problem. The time taken to solve the given problem in this way could be much less than that if the analogy had not been used.

If previously solved problems were stored together with their solutions, and could be retrieved efficiently, the problem solver could become more efficient and powerful over the course of its problem solving experience.

Analogy would also give a useful way of providing advice to an interactive proving system such as PRL [Staff 85].

Matching and Application

There are two main stages involved in solving a problem by analogy: firstly finding and then using the analogy. Finding an analogy means retrieving a previously solved problem and constructing an **analogy match** between the two; this stage is known as **analogy matching**. Using the analogy means applying the match obtained in the first stage to the solution of the solved problem (the **base** problem), and using the result to guide the search for a solution to the given problem (the **target** problem); this stage is known as **analogy application**. The process is illustrated in figure 1.

1..1 Matching

The matching stage is performed by an **analogy matching algorithm**. This takes as input logical representations of the two problems and either outputs an **analogy match** between them, or fails. A match is a correspondence between symbols in the logical representations of the two problems. Previous systems have differed in their representation of matches: in Kling's system, ZORBA [Kling 71], the

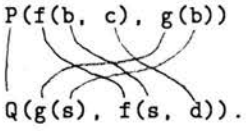
correspondence is between symbols alone, the information about where the symbols occur in the problem statements being discarded. Thus, when given the two statements,

```
P(f(b, c), g(b))        Q(g(s), f(s,d)),
```

the following match might be returned :

```
Predicates : { P <---> Q }
Functions  : { f <---> f, g <---> g }
Constants  : { b <---> s, c <---> d }.
```

(In the Kling-matches displayed below, the distinctions between predicate, variable and constant associations are not shown, in order to show the order in which associations are made.) So analogy matches in ZORBA are *translations*. In Munyer's system [Munyer 81], the information about where the symbols occur in the formulae is retained in the match. The match is a correspondence between *nodes* in the *trees* which represent the statements. Thus, given the same statements, his matcher might return



Munyer calls the individual associations in a match **local maps**.

1..2 Application

Analogy systems **apply** analogies by first constructing the analogues of either the steps in the base solution (Munyer), the axioms used in the base solution (Kling), or perhaps both (see [Owen 85a]), according to the match between the problem statements. This is done with an **analogue construction rule**, the purpose of which is to produce a best guess at what the analogue of a given term is under a given analogy match. The result of the analogue construction is then used to guide the search for a solution to the target problem in some way. In Kling's system, this latter stage is very simple - the axiom base used for the target proof attempt is restricted to the set of analogous axioms. Munyer's system is more ambitious in that it uses the analogues of the base steps as *plan* for the solution of the target. The details of application, while important, will not concern us in this paper.

2 Motivation for reconstructions

There seems to be no general, formal rule which tells us what constitutes a good analogy match. It is in the nature of analogy that it is partly empirical - that is, we only know for sure whether a particular match is good or not after the application procedure has used it in trying to solve the target problem. Therefore

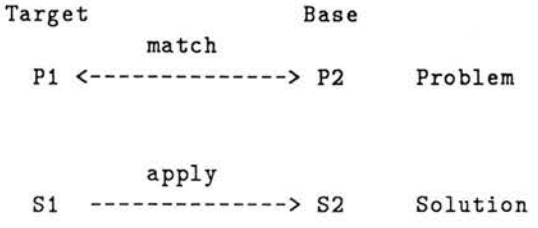


Figure 1: Overview

analogy matching algorithms use *heuristic criteria* to guide them in searching for good analogies. The heuristics that a particular matcher uses determine its *idea* (or rather its designer's) of what is a promising analogy.

But the descriptions given in the literature of existing matchers do not make it at all clear what heuristics they are using - there has been little discussion of the assumptions which matchers make. This makes it very hard to compare the existing matchers as to their performance - how much they achieve relative to the information they are given (see the discussion about semantic types below) - and hard also to extend and improve them.

I have reconstructed the matchers of Kling and Munyer (both in Prolog) in order to analyse their performance and underlying assumptions. The rationale behind such rational reconstructions of previous work is as follows: firstly, the discipline of having to transform the sometimes ambiguous descriptions of the algorithms given by their authors into computer programs forces one to examine the algorithms much more closely than one might do otherwise; secondly, having a working version of an algorithm makes it easy to test it on different examples from those around which it was designed, and thus to assess its generality.

Analogy matching, being in a poor state of explanation as described above, seemed a suitable area for rational reconstruction. More detailed reports of both reconstructions are available from the author.

3 Descriptions of algorithms and reconstructions

The first analogy matcher which I reconstructed was that of Munyer [Munyer 81].

3.1 Munyer

Munyer calls the terms input to his matcher **logical terms**. They are not necessarily first-order, since variables can have arguments (i.e. there can be variable functions and predicates). Munyer's matcher makes no distinction between predicates and functions. If logical *formulae* are to be matched, the logical connectives in them are treated as predicate/function symbols. So the matcher operates with an unrestricted and rather uniform syntax. Furthermore, it uses only the problem statements, and so has no semantics for the symbols in the statements. It is thus **purely syntactic**.

3.1.1 The algorithm

Munyer's matcher works in three stages, constructing intermediate matches from the first two. They are : grounding, deleting and adding.

Grounding

In this stage, an initial map is created in which

- Any pair of identical symbols, one from each term, are associated.
- Any variable in either term is associated with all symbols in the other term.

Deleting

This stage takes as input the output of the previous stage, ie two terms and an initial map between them, and deletes some (or all or none) of the local maps in the initial match. To explain how this stage works we need some definitions.

For symbols a and b in a term, define a > b if b occurs in the (unique) subterm headed by a. For local maps (a1,a2) and (b1,b2), (a1,a2) **dominates** (b1,b2) if a1 > b1 and b1 > b2. (a1,a2) **directly**

dominates (b1,b2) (in a match M) if (a1,a2) dominates (b1,b2) and there is no (c1,c2) in M, dominated by (a1,a2), which dominates (b1,b2).

If (a1,a2) directly dominates both (b1,b2) and (c1,c2), (b1,b2) and (c1,c2) **left-compete** (w.r.t. (a1,a2)) if b1 and c1 are in the same argument position of a1. (b1,b2) **left-supports** (a1,a2) if it has minimal **left penalty** among the maps with which it left-competes. The left penalty of (b1,b2) w.r.t. (a1,a2) is defined as

$$\text{C.P.} + \text{P.P.}$$

where C.P., the containment penalty, is the number of *unmapped* symbols between a1 and b1 in the tree structure of the left term, and P.P., the permutation penalty, is 1 if (b1,b2) permutes arguments of (a1,a2) and 0 if it doesn't. Right-competing and right-supporting are defined analogously.

Lastly, (b1,b2) **reinforces** (a1,a2) if it **either** left-supports **or** right-supports (a1,a2).

The deleting stage works by first calculating the reinforcement relations in the initial match, and then deleting any local maps which (a) share a node with another map **and** (b) do not reinforce any other map in the match. The latter stage is iterated, as a map which is deleted may be the only map which another had reinforced, so the latter should be deleted also.

The motivation behind this complicated procedure seems to be that we should retain a set of local maps which preserve the structure of the terms (this is behind the notion of direct dominance), and remove any alternative sets (this is behind the notion of competition).

Adding

This stage takes as input the result of the deleting stage and adds a local map between any pair of nodes in the two terms which are both unmapped in the existing match but whose parent nodes are mapped to each other.

Also, if the head symbols of the two terms are both unmapped a local map between them is added.

This step is not iterated.

3.1.2 Reconstruction

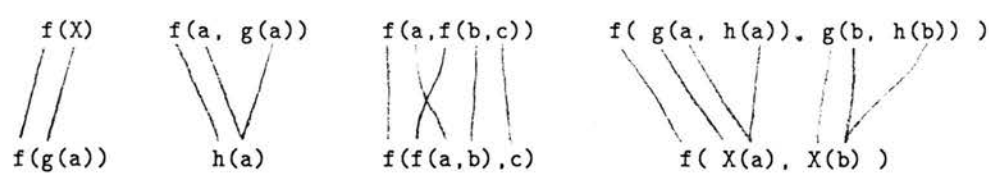
The reconstruction of Munyer's algorithm in Prolog proved to be fairly straightforward, although the details of the deleting stage are fiddly and make the program larger than might be expected. I have not been particularly concerned with efficiency, which I am sure could be improved, but the algorithm seems to be unavoidably expensive.

The structure of the entire program corresponds closely with the description of the algorithm given above, with predicates corresponding to the concepts of the algorithm (directly_dominates, perm_penalty, containment_penalty etc.). It has therefore been quite easy to modify the program when definitions in the algorithm have been changed (the program is **modular**).

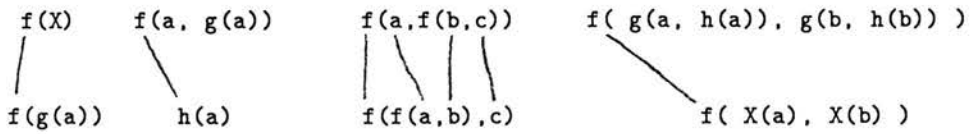
The program has 60 Prolog predicates and 132 clauses.

3.1.3 Performance and modifications

Among the examples of analogy matches which Munyer claims his matcher finds are the following four:



(Note the second order variable, X, in the fourth example). Munyer does not go into *why* these matches are suitable analogies, but lists them as matches which a matcher should be able to construct. The matches produced by my reconstruction on the same examples are as follows:



(I subsequently confirmed these matches by going through the algorithm by hand). Analysis of the third example suggested that it is wrong to count only **unmapped** intermediate nodes in computing the containment penalty - this is why the internal (f,f) map is deleted in favour of the (a,a) and (c,c) maps. If this restriction was dropped, (f,f) would have equal left penalty with (c,c) and equal right penalty with (a,a) and would thus be exempted.

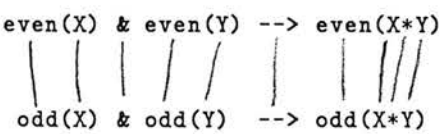
I made this change to the program, and tested the new version on Munyer's examples. The only change in result was in the third example above, for which the desired match was found. Since the performance wasn't worse on any of the examples, I will call this an improvement to the algorithm - improvement A.

Note that the first example should produce a first-order unification. But Munyer gives a 'proof' that his algorithm is complete for first-order unifiable terms (i.e. finds the m.g.u.). Analysis of the obviously faulty proof suggested that the proof would be valid if (B) the definition of reinforcement is strengthened so that (b1,b2) reinforces (a1,a1) if it supports (a1,a2) **both** on the left **and** on the right, and (C) any map which is maximal in the match w.r.t. **partial dominance** (a suitably defined refinement of dominance) is exempted from deletion.

With these changes made to the program, I tested it again on Munyer's examples. It got the right answers on all but one, the remaining one being the example with which Munyer illustrates his algorithm! Again, performance is not impaired on previously successful examples. I therefore call these changes improvements as well - improvements B and C. First-order completeness is finally achieved. (Without C, the algorithm is not even complete for *identical* first-order terms, a(a(a(b))) being a counter-example!)

3.1.4 Assessment of the algorithm

Munyer's matcher is extremely expensive computationally: since variables are associated with all symbols in the other term in grounding, the match which is passed to the deleting stage is usually large. In deleting, complex computations are performed on this match: this is where most of the c.p.u. time is spent. For example, in constructing the match



the program took 30 seconds of c.p.u. time. Considerable computational cost seems inherent to the structure of the algorithm. However, there are ways in which the matcher could be guided more than it is:

- consistency: no attention is paid, during matching, to the consistency of the match, particularly the variable bindings which it entails. Matches with inconsistent variable bindings are rejected at a later stage of the analogy process (see [Munyer 81]). But the complexity would be reduced if only consistent sets of variable bindings were considered in the first place. This would involve the development of several possible matches between a pair of terms which Munyer, for some reason, seems to want to avoid. The (relative) consistency of the predicate and variable mappings could be used to prune unpromising matches.

- syntactic types: Munyer's matcher blurs some syntactic distinctions which analogies tend to respect, for example the special status of the propositional connectives, and the distinction between predicates and functions. One way of exploiting these would be to construct only those local maps which respect the syntactic types during grounding, and put in those which don't, if at all, during adding.

3.1.5 General assessment of syntactic matching on terms

A purely syntactic analogy matcher, such as Munyer's is based on the hope that there will be enough clues about an analogy in the syntax for the solution of the base problem to be of use in the search for a solution to the target. In studying Munyer's matcher, the following problems with this philosophy have become apparent:

- Sometimes the analogy is not apparent in the syntax, for one of the following reasons:
  - **Ambiguous syntax problem**: there are several plausible syntactic matches and no syntactic clue as to the 'correct' one: in the following example, the correct match is given, which pairs cosine terms whose arguments are in corresponding arithmetic progression, but note that a syntactic matcher would have no reason for distinguishing this map from the other possible pairings of terms :

```
cos(x) + cos(3*x) + cos(5*x) = 0
cos(3*x) + cos(5*x) + cos(7*x) = 0.
```

  - **Superficial difference problem**: the syntactic structures of the terms are substantially (but unimportantly for the analogy) different:

```
even(X)   --->  even(X^2)
even(X) & even(Y) ---> even(X*Y).
```

    A purely syntactic matcher is not able to 'see through' the irrelevant differences in representation (e.g. X^2 for X*X). Yet the analogy is clear to a human.

- **Multiple matches problem**: even if the analogy seems clear in the syntax, and one would expect a syntactic matcher to find it, for example

```
a + root(X) = X
a - root(X) = X,
```

  it may well be that other, non-analogous, formulae would match just as well:

```
a + root(X) = X
log(a, root(X)) = X,
```

  for example, would be an equally valid match - the two pairs are syntactically indistinguishable - but would not be a fruitful one (the equations cannot be solved in the same way).

Of these three problems, we could live with the first two - we would just miss a proportion of useful matches. The multiple matches problem is more serious: as the knowledge base of solved problems expands, the chances of finding a fruitless match would increase until it was not worth the effort of the application routine.

The problems show that syntactic matching is not strong enough: it can't distinguish different plausible matches with the same term or with different terms.

3.1.6 Possible solutions

Adding extra (semantic) information along with the terms could help to solve the problems of the last section:

Ambiguous syntax

The method of solution for the base problem relies on the cosine arguments being in arithmetic progression. Numerical features of problem statements are often crucial to their solutions. By applying **constraint back-propagation** [Utgoff 83] (or related techniques), these features could be extracted from the base solution and used to *guide* matching - fruitless matches could be rejected quickly in favour of those which (approximately) preserve the features of the base.

Superficial difference

In order to 'see through' irrelevant syntactic differences between formulae, the matcher will need to construct analogies **modulo a theory**. In the example given above, it would need to know that X^2 = X*X and that A&A <—> A. This is very similar to 'built-in' unification [Raulefs 78]. The incorporation of such ability into a matcher would not be easy, involving extra search; it would also permit extra matches to be found, while not, in itself, blocking any; and so it would be best used along with the suggestions for **guiding** the matcher made above and below.

Multiple matches

It might be possible to use knowledge about the symbols in the terms to distinguish the promising matches from the bad ones. One fairly obvious way of doing this would be to use a **type hierarchy** so that, the closer two symbols were in the hierarchy, the more plausible they would be considered as analogues.

Kling uses 'semantic templates' in his matcher, which seem very similar to a type hierarchy. I went on to a reconstruction of Kling's matcher, with particular reference to its use of semantic types, which I describe next.

3.2 Kling

Kling's analogy matcher, INITIAL_MAP, is part of his analogy system ZORBA. INITIAL_MAP accepts as input two theorem descriptions, each of one of the forms

$$H_1 \& H_2 \& \ldots \& H_n \rightarrow C_1 \& C_2 \& \ldots \& C_m \quad \text{or} \quad C_1 \& C_2 \& \ldots \& C_m,$$

where each of the H's and C's is of the form

$$P(x_1, \ldots, x_r),$$

P being a predicate and the x's variables. The variables are implicitly universally quantified. Thus there are no functions or constants. Note that this is a more restricted syntax than that accepted by Munyer's matcher. Kling's claim that this syntax is suitable for mathematics is surely unfounded, but the algebra theorems around which Kling designed his matcher do fit naturally into this form.

INITIAL_MAP also has access to **semantic templates** for each of the predicates in the sentences to be matched. For example, the predicate group/2 has semantic template

$$structure(set, operation).$$

3.2.1 The algorithm

I.M. matches the hypotheses of the theorems with each other and similarly the conclusions; it does one of these first, then the other, using the match obtained from the first to guide the second.

Kling doesn't say in what order the two sub-matches are attempted. In my reconstruction, whichever of the conclusions and the hypotheses contains fewest atoms is attempted first, on the basis that it is usually easier to match smaller structures.

So, if the theorems to be matched are

$$A_1 \& A_2 \& \ldots \& A_n \rightarrow B_1 \& B_2 \& \ldots \& B_m \quad \text{and} \quad C_1 \& C_2 \& \ldots \& C_n \rightarrow D_1 \& D_2 \& \ldots \& D_m$$

and m+s < n+r, the module SETMATCH is called first on the conclusions, then the hypotheses.

SETMATCH breaks up into two submodules, SINGLEMATCH and MULTIMATCH. SINGLEMATCH pairs up the atoms in the sets which have unique semantic templates, and then MULTIMATCH pairs up the rest using the partial match from SINGLEMATCH as guidance.

Whenever a pair of atoms is associated, ATOMATCH is called to extract predicate and variable correspondences, which are added to the evolving match. If the atoms have identical semantic templates, the correspondences added are just those which preserve argument order. But Kling also wants to be able to find analogies in which, for example, group(Set, Op) is associated with ring(Set1, Op1, Op2), and these predicates do not have the same semantic templates. Their templates are structure(set,operation) and structure(set,operation,operation) respectively. Kling's solution to this problem is a hack. He allows a variable in one of the atoms to be associated with a consecutive sequence of variables in the other as long as (a) all the types involved are the same and (b) the correspondences preserve argument order. In this case, he would get

```
group <--> ring,   Set <--> Set1  and  [Op] <--> [Op1,Op2].
```

If conditions (a) and (b) cannot be satisfied, ATOMATCH fails. I describe this as a hack because there is no reason, in general, why associations should preserve argument order - this depends on arbitrary choices made in formally representing the theorems.

The job of SINGLEMATCH and MULTIMATCH, then, is to decide on pairs of atoms to pass to ATOMATCH. SINGLEMATCH uses the following criteria in its decision:

1. If there is only one unpaired atom left on each side, associate them by default.
2. Associate pairs of atoms having the same semantic template.
3. Associate pairs of atoms whose predicate symbols are associated in the existing match.
4. Associate pairs of atoms whose predicate symbols have the same type.

1 is checked after any association is made. Otherwise 2, 3 and 4 are checked in that order.

If SINGLEMATCH cannot pair up all the atoms with unique semantic templates on each side, it fails. If it can, the partial match produced is passed on to MULTIMATCH, which considers the atoms on each side which have the same semantic templates as others. First, the atoms are grouped into their semantic blocks; then blocks are paired up using the following criteria in the same way that SINGLEMATCH used its criteria:

1. If there is only one unpaired block on each side, associate them by default.
2. Associate pairs of blocks which contain atoms whose predicate symbols are associated by the existing match.
3. Associate pairs of blocks whose atoms have predicates with the same semantic type, where this gives a unique association.

When a pair of blocks is associated by the above criteria, the submodule MULTIMATCH1 is called to pair up the atoms within the blocks. MULTIMATCH1 has a single criterion for doing this, based on the existing variable associations: if there is a pair of variables associated by the existing match, each of which occurs in only one atom in the appropriate block, then pair up the two atoms.

This completes the description of INITIAL_MAP.

3.2.2 Reconstruction

The reconstruction of INITIAL_MAP in Prolog turned out to be straightforward - considerably easier than Munyer's algorithm. The code has 46 predicates and 102 clauses. The modules in Kling's description of the algorithm translated easily into Prolog predicates.

3.2.3 Performance and modifications

I first tried the matcher on the algebra examples that Kling gives in [Kling 71]. There are several different ways of translating these (given in natural language) into logic. If 'corresponding' ways for analogous theorem pairs are not chosen, I.M. will almost certainly fail (there would be a superficial difference problem).

The performance of the reconstruction was satisfactory on most of the algebra examples that Kling gives in [Kling 71]; where there were representation choices, the matcher was successful on at least one of them. The following is one of Kling's theorem pairs and the resulting match obtained by the program:

```
HYPOTHESES                          HYPOTHESES
group(g,*)                          ring(r,**,++)
propernormal(m,g,*)                 properideal(n,r,**,++)
factorstructure(x,g,m)              factorstructure(y,r,n)
simplegroup(x,*)                    simplering(y,**,++)

 CONCLUSION                          CONCLUSION
maximalgroup(m,g,*)                 maximalring(n,r,**,++)
```

```
                    MATCH

    maximalgroup <-------> maximalring
               m <-------> n
               g <-------> r
             [*] <-------> [**,++]
 factorstructure <-------> factorstructure
               x <-------> y
   propernormal <-------> properideal
          group <-------> ring
    simplegroup <-------> simplering
```

The reader is encouraged to work through this example by hand.

While the program performed as desired on Kling's algebra examples, the heavy use of the semantic templates worried me. All of Kling's theorem pairs come from essentially the same global analogy between group theory and ring theory. It seemed to me that it was Kling's knowledge of this global analogy which led him to choose the particular semantic templates which he did for the examples. Thus it is no surprise and no great achievement that the algorithm finds the correct matches in these cases. There is a strong suspicion of circularity about this.

After going through Kling's algebra examples, I tried the matcher on some new examples. One of these was the following pair:

```
-(x + y) = -x + -y        and    1/(x.y) = 1/x .1/y
```

When transformed into Kling's syntax, these become:

Example A

```
(times(x,y,z)                    (plus(x,y,z)
& inv(z,z1)                      & minus(z,z1)
& inv(x,x1)                      & minus(x,x1)
& inv(y,y1)                      & minus(y,y1)
& times(x1,y1,r))                & plus(x1,y1,r))

--> equal(z1,r)                  --> equal(z1,r)
```

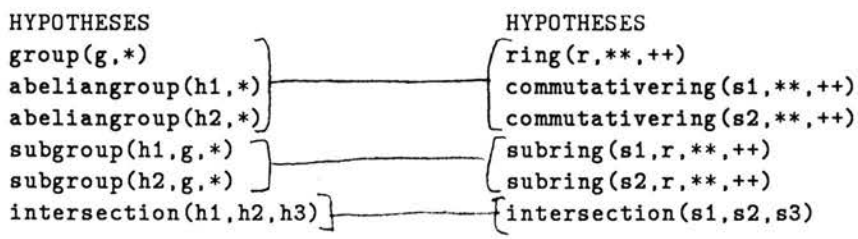
where new functional predicates are introduced to replace the functions. I chose very general types - 'relfn' for all the functional predicates and 'rel' for equal/2. I.M. fails on this easy example, because MULTIMATCH fails on the hypotheses, not being able to pair up the semantic blocks with condition 3. If MULTIMATCH is given the extra semantic template condition that SINGLEMATCH has, the problem would be solved. With this change made to the program, performance is not impaired on any previous examples.

Another example which I tried the matcher on was:

```
  rational(p)                      rational(p)
& irrational(q)                  & irrational(q)
& plus(p,q,r)                    & unequal(p,0)
                                 & times(p,q,r)
--> irrational(r)                --> irrational(r).
```

General semantic templates were again used. The matcher fails on this example, since it requires a 1-1 association of atoms in order to succeed. The failure occurs in SINGLEMATCH. Since extra conditions on one side often occur in analogies, allowing SINGLEMATCH to succed if there is one atom left on one side seems sensible. With this change made, the matcher finds the right match, and, again, performance on previous examples is not impaired.

Another example where the performance of the matcher was not as I expected is shown below:




```
abeliangroup <-------> commutativering
          h3 <-------> s3
         [*] <-------> [**,++]
intersection <-------> intersection
          h1 <-------> s1
          h2 <-------> s2
    subgroup <-------> subring
           g <-------> r
```

The match lacks the expected group <—> ring association. This is because MULTIMATCH1 cannot pair up the group and ring atoms within the first pair of semantic blocks shown.

There seem to me to be two possible solutions to this problem:

- MULTIMATCH1 could be given a default condition, like those in SINGLEMATCH and MULTIMATCH. The group and ring atoms would then be ATOMATCHed by default and the final match would be complete.
- The application of MULTIMATCH1 to the pairs of semantic blocks could be more flexible, allowing it to interleave work on the various pairs, so that if it gets stuck on one, it can move on to another one and return to the first if progress is made on the second.

The second modification is more principled than the first (defaults should be avoided if possible). Again, performance is not impaired on previous examples.

3.2.4 Assessment

SINGLEMATCH, MULTIMATCH and MULTIMATCH1 all try to pair up parts of the sentences by reference, in part, to the existing partial match. But all of the tests are *monotonic* in the existing match - that is, they force an association if some related association has already been made. The existing match is never used to explicitly *inhibit* an association, which might be a good thing to do if the association in question would be inconsistent with the match. The lack of lateral inhibition in INITIAL_MAP may be the result of the fact that it never had to search for analogies in a knowledge base of solved problems (Kling fed it with pairs of problems which he knew were analogous). In the more realistic situation, it is important for the matcher to fail on non-analogous problems as well as to succeed on analogous ones, and this is where consistency checks (and other strength criteria) could be useful.

A more serious problem with the matcher is shown up by the following example:

Equation	In Kling's syntax
`root(x) + a = b`	`root(x,z) & plus(z,a,x) & equal(x,b)`
`root(x+1) + a = b`	`plus(x,1,y) & root(y,z)` `& plus(z,a,x) & equal(x,b)`

In the first sentence, all the atoms have unique semantic templates, whereas in the second the two plus atoms are in a semantic block together. Thus, SINGLEMATCH pairs up the two root atoms and the two equal atoms, and leaves the plus atom in the top sentence un-matched. Then MULTIMATCH is called between a single block on one side and nothing on the other, and so fails.

It is the basic SINGLEMATCH/MULTIMATCH distinction which is at fault here: where there is an atom which ought to be unmatched, its semantic template may coincide with that of another which ought to be matched by SINGLEMATCH, which will prevent the latter from ever being considered by SINGLEMATCH. It seems likely that this situation will occur often in non-trivial analogies, and thus that the whole matching strategy of INITIAL_MAP will have to be changed to cope.

3.2.5 Assessment of the use of semantic types in matching

The idea behind the use of semantic types in analogy matching is that symbols of the same/similar type make good analogues. Thus types are used to give *a priori* plausibility judgements for analogies. Semantic types can be thought of as expressing information about global analogies within the domain; i.e. symbols of the same/similar type are likely to have structurally similar axioms/operators associated with them. Thus, given that the symbols occur in structurally similar formulae, and regarding a solution as a sequence of structural rearrangements leading from a start to a finish, each arising from an axiom or operator, it is likely that the two formulae will have similar solutions. Thus semantic types will be effective, when used along with the **partial homomorhism heuristic** (see below).

However, as noted earlier, there is a danger of circularity in the use of types i.e. that the types encode the analogies that the matcher is supposed to find for itself. This will be so if (a) the types are specified

by the user *and* (b) the matcher does not go beyond the analogies encoded in the types. These conditions are, unfortunately, met by Kling's matcher, which repeatedly reconstructs the global analogy between group theory and ring theory which is represented by the types.

To avoid (b), the matcher would need to use the types more flexibly than Kling does - it ought not to be based around the enforcement of type preservation, as I.M. is. A more structured type **hierarchy** would be helpful for this, as it would give a notion of *similar* type.

To avoid (a), the analogy system could create/refine its type hierarchy over the course of its analogical problem solving experience: roughly, the new types would be sets of symbols which had often been associated in previous analogies. In [Owen 85b], the learning of types using analogies is gone into in detail.

4 Summary and conclusions

Neither of the algorithms described provides a robust and useful analogy matcher. Munyer's algorithm, even with the improvements, is dogged by the fact that it is purely syntactic, and thus suffers from all the problems of synactic matching discussed above. Kling's algorithm has no solution to the first two problems, and its solution to the third is inflexible and circular. Furthermore, Kling's matcher is incapable of producing matches which involve (a) any argument permutation and (b) matched predicates with one or more unmatched arguments, both of which occur often in analogies.

The purpose of this paper is to describe heuristics for analogy matching. The following five heuristics are the basis of the matchers which have been reconstructed:

- partial homomorphism heuristic: an analogy match is likely to be a partial homomorphism between the terms to be matched i.e. the associations in a match are expected to respect the structure of the matched terms. Kling uses this heuristic in ATOMATCH, where variable associations which preserve argument order are extracted from prior associations of the containing predicates. Munyer uses the heuristic both in the deleting and adding stages, which are based around the idea that associations in a match should reinforce each other if the pairs of symbols they connect are in the same relative positions within their terms.

- syntactic type heuristic: symbols which are associated in a match are likely to be of the same syntactic type. This heuristic is built in to the structure of Kling's algorithm, which never considers associating symbols unless they have the same syntactic type. Surprisingly, Munyer makes no use of this heuristic.

- consistent/efficient translation heuristic: the translation between symbols implied by an analogy match ought to be consistent, in the sense that the mapping should be as near to 1-1 as possible, and efficient, in the sense of matching relatively large structures with a relatively small mapping. (This heuristic is used in the related field of **inductive generalisation** [Dietrich 79].) Kling uses a form of the heuristic in SINGLEMATCH, MULTIMATCH and MULTIMATCH1, when atoms are paired whose predicates or variables are already associated in the match; i.e. the expected consistency of the match is used to enforce correspondences which in turn increase the efficiency of the match. An additional use of the heuristic, not made by Kling, would be to explicitly inhibit associations inconsistent with the developing match. Munyer does not use this heuristic at all during matching.

- identical symbols heuristic: symbols which are associated in a match are (*a priori*) likely to be identical. Munyer uses this heuristic in order to get his algorithm going, and then proceeds with the partial homomorphism heuristic. Kling does not use this heuristic.

- semantic type heuristic: symbols which are associated in a match are (*a priori*) likely to have the *same* (Kling) or *similar* (more general) semantic type. This heuristic dominates Kling's algorithm, being the main source of possible predicate associations. As discussed above, this heuristic is important but problematic.

These heuristics all have validity. My criticism of the matchers discussed here is that they are used rather inflexibly (particularly the last two). It seems that any heuristic criterion will only be valid some of the time. Therefore it would be better to use the heuristics flexibly within a best partial match framework, such as that proposed by Hayes-Roth [HayesRoth 78]. The matcher would prefer, and be guided towards, matches which respect the heuristics, but would not be precluded from finding those which didn't. This is particularly important for the learning of types, which can only be achieved when existing type information is overridden.

References

[Diettrich 79] T.G. Diettrich and R.S. Michalski. Learning and generalization of characteristic descriptions: evaluation criteria and comparative review of selected methods. In *Proceedings of the Sixth IJCAI*, International Joint Conference on Artificial Intelligence, 1979.

[HayesRoth 78] F. Hayes-Roth. The role of partial and best matches in knowledge systems. In *Pattern-Directed Inference Systems*, Academic Press, 1978.

[Kling 71] R.E. Kling. A paradigm for reasoning by analogy. *Artificial Intelligence*, 2, 1971.

[Munyer 81] J.C. Munyer. *Analogy as a Means of Discovery in Problem-Solving and Learning.* PhD thesis, Univ. Calif. Santa Cruz, 1981.

[Owen 85a] S.G. Owen. *Analogy in Artificial Intelligence - Thesis Proposal.* Working Paper 176, Edinburgh University, 1985.

[Owen 85b] S.G. Owen. Getting better at analogy. 1985.

[Raulefs 78] Siekmann J. Szabo P. Raulefs, P. and E. Unvericht. A short survey on the state of the art in matching and unification problems. *AISB Quarterly*, issue 32:17–21, December 1978.

[Staff 85] The PRL Staff. *Implementing Mathematics with the NUPRL proof development system.* 1985.

[Utgoff 83] P.E. Utgoff. Adjusting bias in concept learning. In *Proceedings of the Eighth IJCAI*, International Joint Conference on Artificial Intelligence, 1983.