Analogy for Automated Reasoning

Analogy for Automated Reasoning

Stephen Owen

Hewlett Packard Laboratories
Stoke Gifford, Bristol

ACADEMIC PRESS
Harcourt Brace Jovanovich, Publishers
Boston San Diego New York
London Sydney Tokyo Toronto

ACADEMIC PRESS LIMITED
24–28 Oval Road, London NW1 7DX

This book is printed on acid-free paper.

United States Edition Published by
ACADEMIC PRESS, INC.
1250 Sixth Avenue, San Diego, CA 92101

ISBN 0-12-531715-8

Library of Congress Cataloging-in-Publication Data is available

Printed in Great Britain by
St Edmundsbury Press Limited, Bury St Edmunds, Suffolk

90 91 92 93 9 8 7 6 5 4 3 2

Editor's Note

By now it is a truism in artificial intelligence (AI) and cognitive science that metaphor and its parent, analogy, are ubiquitous in human cognitive behavior. A central problem in analogy is how to match a given situation with situations stored in memory so that the most analogous one can be brought to the fore. A deep logical problem exists here, as suggested by Watanabe's Theorem of the Ugly Duckling (Watanabe, 1969), to wit: "Any pair of two objects are as similar to each other as are any other pair of two objects, insofar as the degree of similarity is measured by the number of shared predicates." Is the degree of analogy between two situations inherent in a given pair of situations, a function of the context and the goal of the agent, or some combination of the two? The first alternative, which would lead to matching routines that are functions of only the descriptive vocabulary for situations, is ruled out if one supports Watanabe's theorem. Belief in the second alternative means that, given any two situations, one can find goal descriptions of the agent for which the two situations are the most analogous. Of course, the third alternative appears suitably moderate, being a *via media* position, but figuring out how to combine the two alternatives is actually quite difficult.

Steve Owen's work on analogy is a good step forward in attempts to understand how analogies work. By concentrating on a well-defined set of goals (proving theorems) and a well-defined domain (several areas of mathematics), he has built a powerful experimental system that can be used to investigate various matching heuristics. One of the uses of analogies in problem solving is that once an analogy is found, the solution of the old problem is somewhat transferable to the solution of the new problem. Owen also investigates the range of options available for controling this solution transfer process.

Another aspect of Owen's book that workers in analogy will find useful is his analysis of previous work in this field. His framework enables him to cast the matching and solution-transfer strategies of earlier researchers in a more uniform language so that they may be compared experimentally.

It seems to me that complex problems such as analogy can be investigated only by experiments in different domains, experiments of appropriate size and scope that do not trivialize the problem of finding and using analogies. Owen's work is an excellent example of such much-needed research. I am happy to present it to the AI and cognitive science community as part of the Perspectives in Artificial Intelligence series.

—B. Chandrasekaran

References

Watanabe M. S., *Knowing and Guessing*, John Wiley & Sons, New York, 1969.

Foreword: Alan Bundy

Human thought and reasoning is replete with analogies: from the use of metaphor in everyday speech to the association of ideas that inspires a new theory. It has long been a dream of researchers in artificial intelligence to build a computer program that uses analogy, so that we may better understand how human analogical reasoning is possible, and so we can apply it to solve technological problems. In recent years this interest has grown significantly with the work of Gentner, Carbonell, Holyoak, Winston and many others.

There have been many attempts to build an analogical reasoning program, but they have all been rather unsatisfactory. It is almost impossible not to 'cheat' by building into the program the very analogy you wish it to discover. Firstly, you think of an analogy between two situations, for instance, the particles in an atom *vs* the planets in the solar system, or radiation attacking a cancerous growth *vs* armies attacking a city. Secondly, you represent these situations symbolically in your program. Thirdly, your program 'discovers' this analogy by some kind of matching process between the two representations. Lastly, it demonstrates its analogical reasoning by applying a solution known in one situation to an 'unsolved' problem in the other.

One is never sure what understanding has been gained in such demonstrations. There is always the suspicion that the symbolic representations of the two situations were consciously or unconsciously chosen to simplify the task of the analogical matcher. It is unclear how the program would fare give some huge database of situations provided by some third party and asked to find an analogy suitable for solving some previously unsolved problem. Would the best analogies prove elusively beyond the abilities of the matcher? Would the matcher become bogged down in the computational complexity of finding all possible connections between all pairs of situations? We do not know because we do not have access to a suitably encoded huge database. Nor is one likely to become available in the foreseeable future.

There is one exception to this. The world of mathematics provides a potentially huge database of situations in the form of different mathematical theories, theorems and proofs. It is also especially rich in the potential for analogies within and between these theories. Moreover, an independent encoding of these situations is readily available from the work of mathematical logicians, and it is a relatively simple matter to encode large numbers of them in a machine readable form. This mathematical world provides a domain in which analogical reasoning mechanism can be thoroughly tested and compared.

Despite these methodological advantages the domain of mathematics has been only a minor player in the history of automated analogical reasoning. The exceptions are researchers like Kling, Munyer and Bledsoe. Their goal has been the use of analogies between formulae to guide the search for the unknown proof of a new conjecture using the known proof of an old theorem. To find and apply such analogies they have built fuzzy matchers which look for near isomorphisms between the new conjecture and the old theorem. These matchers all have rather an *ad hoc* flavour. They relax the normal constraints of isomorphism in whatever ways occurred to the human designer and were needed to find the analogies they knew were present.

Steve Owen has built on all this work and has made a significant step forward. He has surveyed and analysed these previous matchers and discovered the underlying principles on which they are based. This has enabled him to build a general purpose, modular matcher, which calls on a set of heuristics. These heuristics can be readily modified, enabling the experimental investigation of different combinations of heuristics. Owen has conducted thorough experiments with this machinery to find which combinations of heuristics perform best in the long run.

Once a match has been found between the old theorem and the new conjecture it can be used to map the old proof into a new proof. Unfortunately, life is not quite this simple; a step of the old proof may not be a legal proof step when mapped across. For instance, mapping an axiom to a new formula may not produce another axiom. The mapped old proof must be considered as a rough plan, which requires filling out to become a proof of the new conjecture. Owen has also applied his analytical and experimental methodology to this problem. He has surveyed previous work on plan application and investigated the choices available for implementing plan appliers. Using this analysis he has built a modular system which allows different options to be rapidly chosen and compared. He has then conducted his usual thorough testing of the different options in order to assess their relative merits.

At last we have a basis for the empirical study of analogical reasoning. Owen has shown us how to test the properties of analogy finding matchers and plan appliers on a body of independent data. His modular matcher and plan applier enable us to formulate and test different hypotheses about the mechanisms of analogical reasoning. I believe this book will represent a major milestone in the computational investigation of analogical reasoning and become required reading for the new generation of researchers that it will inspire.

Alan Bundy
University of Edinburgh

Chapter 1

Introduction

1.1 The goal

The goal of the research described in this book is to enhance automatic problem solving systems with **analogical ability**: that is, a capacity to use a known solution to a problem to aid in the solution of a similar problem.

The goal is primarily a technical one. However, much of the inspiration for the endeavour comes from the accepted human ability to reason by analogy — this apparently natural ability, 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 need to refine this definition, for example to exclude reasoning by inductive generalisations, which certainly fits the definition given. However, the definition given forms the focus of this book.

Motivation for the research comes from a desire to build more powerful and adaptable problem solving systems. An automatic problem solver with analogical ability would be an **extensible reasoning system**: that is, a system able to increase its problem solving power over the course of experience in solving specific problems. As more problems were solved, the system would extend its knowledge base and power through being able to use analogies with a larger set of past problems. We would thus hope to see increased problem solving power as one direct result of the analogical ability. However, it may be that analogical ability proves most useful in other ways: knowledge of useful analogies in a domain can indicate directions for generalisation. Such generalisations can represent major breakthroughs in the understanding of a domain. In this way analogical ability can be of more significance than simply in improving a system's ability to answer questions: it can help to ask new questions.

Such a system remains a long term aim. Many problems will have to be overcome before one is built; some of these problems are addressed in this book, some are not. Section 1.6 discusses the scope and limitations of the book.

The work described in this book is not intended primarily to represent a model of human analogical reasoning, despite being motivated by the corresponding human ability. The goal is to build computer systems which can use analogy. However, the work is influenced by the models of analogical reasoning which have been proposed by cognitive psychologists (we discuss these in the next chapter); and it may well be that the research will prove useful to psychologists by feeding back experience of the technical aspects of analogical reasoning.

1.2 The Basic APS framework

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.**

As indicated above, the Basic APS framework is not intended as a model of human analogical reasoning (although it bears resemblances to many such models); but it describes how APS researchers have attempted to introduce analogy into automatic problem solvers.

The Basic APS framework leaves open the issue of where the base problem comes from and how it is found. In almost all analogy research done so far, the base problem is provided by the user of the analogy system. A realistic analogical reasoning system will have to find promising base problems

$$\begin{tikzcd}
 & \text{Base} & & \text{Target} \\
\text{Problem} & P_2 \arrow[rr, leftrightarrow, "match"] & & P_1 \\
\text{Solution} & S_2 \arrow[dr, "construct"'] & & S_1 \\
\text{Plan} & & X \arrow[ur, "apply"'] &
\end{tikzcd}$$

Figure 1.1: Basic APS

for itself from among a large knowledge base of known and solved problems — we refer to this problem as the **base filtering problem.** Many analogy researchers have included base filtering as part of their model of analogical problem solving, particularly those concerned with providing a model of human analogical reasoning. It augments the Basic APS model as a prior stage.

1.3 Examples

We discuss here some examples of problem solving analogies. These are the kinds of analogy which we might wish an analogy system to handle. They give a feel for the potential utility of an analogy in guiding the search of an automatic problem solver, and also for the task that we face in building an analogy system.

Elementary number theory

Consider the following simple results of elementary number theory:

$$\mathit{even}(x) \wedge \mathit{even}(y) \rightarrow \mathit{even}(x \cdot y) \qquad (1.1$$

$$odd(x) \wedge odd(y) \rightarrow odd(x \cdot y) \tag{1.2}$$

In a mathematics textbook, it is common for a result like the first to be derived in the text just after the concepts of *even* and *odd* have been defined, and then for the second to be set as an exercise for the student.

We can represent the similarities and differences between the problems by the following **symbolic correspondence**:

$$\begin{tikzcd}[column sep=tiny]
even \arrow[d, no head] & (x) \arrow[d, no head] & \wedge \arrow[d, no head] & even \arrow[d, no head] & (y) \arrow[d, no head] & \rightarrow \arrow[d, no head] & even \arrow[d, no head] & (x \arrow[d, no head] & \cdot \arrow[d, no head] & y) \arrow[d, no head] \\
odd & (x) & \wedge & odd & (y) & \rightarrow & odd & (x & \cdot & y)
\end{tikzcd}$$

This correspondence indicates the structural isomorphism between the problems, and the consistent mapping of symbols

$$\begin{array}{ccc}
x & \longleftrightarrow & x \\
y & \longleftrightarrow & y \\
even & \longleftrightarrow & odd \\
\cdot & \longleftrightarrow & \cdot
\end{array}$$

between the two. The mapping indicates differences as well as similarities between the problems. In embarking on a target proof by analogy to the base proof, the hope is that the differences do not seriously affect the *form* of the base proof. The use of the word 'hope' in the previous sentence is significant: we are not usually sure that the analogy will work before trying it out; often, the similarity between the problem statements is misleading, and the attempt to use the base proof as a guide for the proof of the target fails. In the next chapter, we argue that this uncertainty is an important aspect of analogical reasoning. In this case, the analogy turns out to be a good one, as we now describe.

The derivation of 1.1 might go as follows:[1]

> Expand the assumptions $even(x)$ and $even(y)$ by the (recently introduced) definition of *even*:[2]
>
> $$x = 2 \cdot a \quad \text{(for some } a\text{)}$$

[1] We use a somewhat informal style to present this example – just enough detail to illustrate the analogy. When we return to the example, later in the book, we use a refutation style for the example.

[2] We use the following definition of *even*:

$$\forall x \,.\, even(x) \leftrightarrow \exists y \,.\, x = 2 \cdot y$$

$$y = 2 \cdot b \quad \text{(for some } b\text{)}$$

Then multiply these equations:

$$x \cdot y = (2 \cdot a) \cdot (2 \cdot b) \tag{1.3}$$

Then rearrange 1.3 to get:

$$x \cdot y = 2 \cdot (a \cdot (2 \cdot b)$$

and lastly reapply the definition of *even*, this time contracting it:

$$even(x \cdot y)$$

Formula 1.2 can be proved in a similar way (i.e. 1.1 and 1.2 are analogous):

Expand the assumptions *odd*(x) and *odd*(y):

$$x = 2 \cdot a + 1 \quad \text{(for some } a\text{)}$$

$$y = 2 \cdot b + 1 \quad \text{(for some } b\text{)}$$

Then multiply as before:

$$x \cdot y = (2 \cdot a + 1) \cdot (2 \cdot b + 1)$$

Then rearrange (but this part is harder than before):

$$x \cdot y = 2 \cdot (a \cdot (2 \cdot b + 1) + b) + 1$$

Lastly reapply the definition of *odd* as before, obtaining:

$$odd(x \cdot y)$$

In overall shape, and many details, these two proofs are very similar. Just as the first proof helps the student to find the second, we would hope that an analogy system would be able to use the first as guidance in its search for a proof of the second. Intuitively, the guidance is encapsulated in expressions such as 'multiply as before': at each intermediate stage, a problem solver (human or machine) would find that were many possible steps that *could* be made. The base proof can be used to suggest which one of these *should* be made. Notice that the analogy does not amount simply to applying the same sequence of operators that worked in the base (although this does sometimes work). For example, expansion and contraction of the definition of *even* is replaced by those for *odd* — here the proofs are *correspondingly different*. More seriously, the rearrangement stages of the two proofs are quite different:

for the target this stage involves six basic steps, whereas there was just one for the base. For this stage, the base proof is not much help, and the student (or machine) must do more work. So the proofs are not *precisely* analogous, but enough so for the analogy to be useful. The analogist[3] must be able to follow the base proof where it is useful, while being prepared to fill in gaps where it is not. It is a challenge for the designers of analogy systems to achieve this behaviour. We describe how this has been attempted in a later chapter.

Before we move on to the next example, notice one more thing: we described the base proof in a way that also validly describes the target proof (once it is found): there are corresponding stages, and the description of the 'rearrange' stage is chosen so that it applies to the corresponding target stage. Sometimes humans are given similar descriptions to work with, which may be helpful; sometimes there are no such descriptions, or the descriptions given turn out to be misleading. Indeed, sometimes a person is able to form a common description as a result of applying the analogy; these descriptions can be an important part of the learning gained from the analogical problem solving.

Boolean algebra

In Boolean algebra, there is a well known *duality* between meet ($\cup$) and join ($\cap$).[4] Consider, for example, the following two theorems:

$$\begin{tikzcd}
\textsc{Base} & x \arrow[d, dash] & \cup \arrow[d, dash] & x \arrow[d, dash] & = \arrow[d, dash] & x \arrow[d, dash] \\
\textsc{Target} & x & \cap & x & = & x
\end{tikzcd}$$

The base states that $\cup$ is **idempotent**; the target that $\cap$ is also. The structural correspondence shown indicates the analogy. Figure 1.2 shows refutation proofs of both theorems side by side.[5] All the steps of the proofs

[3]We use the term analogist to refer to the person or machine who is reasoning by analogy.

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

[5]For those not familiar with refutation proofs, here is a brief explanation. If the goal is to prove a universally quantified statement such as $x \cup x = x$, we first assume that it is false; that is, that there exists an a such that $\neg a \cup a = a$ (first line of base proof). We then draw consequences from this assumption until a contradiction is reached (*nil*, last line). This means that our assumption of the existence of such an a was false; or, that the theorem is true. Note that the constant a introduced is *arbitrary*; that is, we are not allowed to assume anything about it during the proof.

are shown. It is easy to see that the proofs are structurally isomorphic, with a mapping that includes and extends that of the problem statements:

$$\begin{array}{ccc} = & \longleftrightarrow & = \\ \cup & \longleftrightarrow & \cap \\ \cap & \longleftrightarrow & \cup \\ 0 & \longleftrightarrow & 1 \\ 1 & \longleftrightarrow & 0 \\ - & \longleftrightarrow & - \end{array}$$

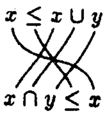
We can regard the application of the analogy as being the extension of the initial match between the problem statements to the complete proofs, including the corresponding axioms that were applied. The analogy is perfect, unlike that of the first example given. If the problem solver was aware of the formal duality within Boolean algebra, it could infer that the analogy was bound to succeed without going through the application stage. If the solver was not aware of the formal duality, it would need to go through the application. In fact the success of the analogy could be used as a clue to the existence of the duality (this is similar to, though more formal than, the suggestion of the proof descriptions by the analogy in the first example).

Lattice theory

We give a brief example to illustrate that even in perfect analogies we must be prepared to permute the arguments to predicates or functions in a match. Suppose that we are defining lattice operations in a partially ordered set with finite least upper and greatest lower bounds. We might define $\cup$ and $\cap$ as follows:

$$x \cup y = lub(x, y)$$
$$x \cap y = glb(x, y)$$

where *lub* and *glb* are the least upper bound and greatest lower bound functions respectively. Consider the analogy between the following properties of the newly defined $\cup$ and $\cap$:



As above, this is a perfect analogy, and is part of the same duality as the previous example (we do not show the isomorphic proofs here). But notice that to describe the problems as isomorphic we need a concept of isomorphism, which allows arguments of functions and predicates to be permuted.

$$\begin{tikzcd}
\textsc{Base}\text{: } x \cup x = x & \textsc{Target}\text{: } x \cap x = x \\
\neg a \cup a = a \arrow[d, dash] & \neg a \cap a = a \arrow[d, dash] \\
\neg a \cup a = a \cup 0 \arrow[d, dash] & \neg a \cap a = a \cap 1 \arrow[d, dash] \\
\neg a \cup a = a \cup (x \cap \bar{x}) \arrow[d, dash] & \neg a \cap a = a \cap (x \cup \bar{x}) \arrow[d, dash] \\
\neg a \cup a = (a \cup x) \cap (a \cup \bar{x}) \arrow[d, dash] & \neg a \cap a = (a \cap x) \cup (a \cap \bar{x}) \arrow[d, dash] \\
\neg a \cup a = (a \cup a) \cap 1 \arrow[d, dash] & \neg a \cap a = (a \cap a) \cup 0 \arrow[d, dash] \\
\neg a \cup a = a \cup a \arrow[d, dash] & \neg a \cap a = a \cap a \arrow[d, dash] \\
\textit{nil} & \textit{nil}
\end{tikzcd}$$

Figure 1.2: Isomorphic proofs in Boolean algebra

We will see that a number of analogy researchers have defined the concept of analogy match to exclude argument permutation. In such accounts, therefore, even perfect analogies like the current one are excluded.

Notation for matches The current example indicates the need for a more detailed representation of symbolic correspondences than the notation

$$\leq \longleftrightarrow \leq$$

which we have used so far. In this example, the arguments to $\leq$ are permuted and we may wish to encode this important information in the correspondence. We therefore sometimes use the more detailed but less evocative notation

$$(\leq, \leq, [(1,2), (2,1)])$$

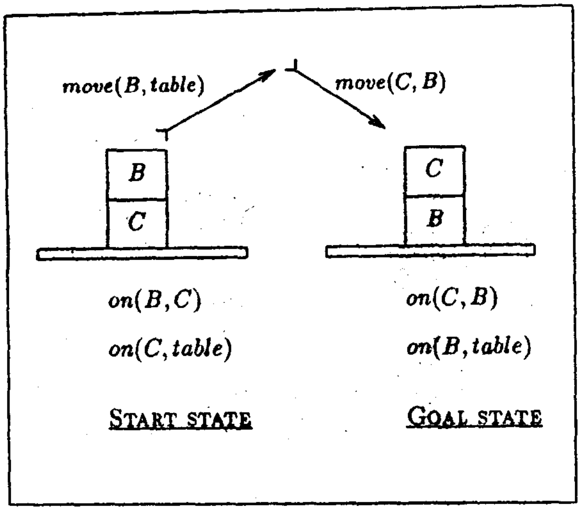


Figure 1.3: Reversing a stack of two

to indicate the argument pairings in an association. The third component of the triple above gives the argument pairings — in this case the first argument of the left symbol with the second of the right, and vice versa. The above association has different implications for an analogy system from

$$(\leq, \leq, [(1,1),(2,2)])$$

in which argument order is preserved. This is why the more elaborate notation is sometimes necessary. We will switch fairly freely between the two notations in this book. It is important that the reader understands the equivalence.

The blocks world

For our next example, we visit the blocks world. Suppose that we know how to reverse the positions of two blocks, one on top of the other. The start and goal states and the operations necessary for this are displayed in Figure 1.3.

Suppose now that we are set the task illustrated in Figure 1.4. We are given a stack of three and asked to construct a configuration in which the stack is reversed. We can describe this problem as 'reverse a stack of three blocks'

just as we could have described the base problem as

'reverse a stack of two blocks'

This strongly suggests an analogy as follows (here we use a correspondence between English sentences to stand for one between corresponding formal descriptions):

$$\begin{tikzcd}
\text{reverse} \arrow[d, no head] & \text{a} & \text{stack} \arrow[d, no head] & \text{of} & \text{two} \arrow[d, no head] & \text{blocks} \arrow[d, no head] \\
\text{reverse} & \text{a} & \text{stack} & \text{of} & \text{three} & \text{blocks}
\end{tikzcd}$$

In a way, using these descriptions for the problems directly is cheating or, at least, assuming sophistication in the problem solver: these descriptions implicitly make a generalisation which encompasses the analogy. As with the previous examples, we would like an analogy system to be able to cope without being provided with the generalised description, rather the straightforward ones given in Figures 1.3 and 1.4.

It is clear that there is no isomorphism between the descriptions of Figures 1.3 and 1.4 since they have different sizes and numbers of variables. This makes the construction of the analogy harder than the earlier examples. We might attempt to associate the whole of the base with parts of the target (i.e. pick two blocks in the target to correspond to those in the base); but this approach fails to express the full analogy between the problems. In Chapter 4, we describe mechanisms by which the problems can be re-expressed into formal versions of the English forms given above as part of an attempt to construct an analogy match between them.

If such re-expression can be performed on the problems, it is important also to re-express the known base solution in terms similar to the new version of the base problem. Specifically, we would like to describe the base solution in terms of operations on a stack of blocks (in this case, two) rather than in terms of applications of basic operations to specific blocks, as it is initially presented to us. This will enable us to construct a corresponding plan for the solution of the target. It turns out that these feats of re-expression can be performed on this example and the target problem solved by application of the plan.

This example is an instance of an important feature of analogical reasoning: the perception of an analogy between two problems (or situations) often involves re-expressing their descriptions. To put it another way: the