Analogical Proof Planning

Toby Walsh
Department of Artificial Intelligence
Edinburgh University
80 South Bridge, Edinburgh
T.Walsh@uk.ac.ed

September 6, 1998

Abstract

Proof planning is a powerful technique for theorem proving. Proof plans are descriptions of common proof strategies; each consists of a sequence of more primitive methods. This paper explores how the methods developed in one domain can be "creatively" applied to new problems and new (sometimes analogous) domains.

1 Introduction

More than 150 years ago, Lady Lovelace wrote "The Analytical Engine has no pretensions to *originate* anything" (her italics) [Tur63]. Since the Analytical Engine is, in principle, a universal Turing machine, her comment could be applied to all computers. AI, especially traditional symbolic AI, continues to face such criticism. Indeed, Karl Popper has argued *"... there is no such thing as a logical method of having new ideas, or a logical reconstruction of this process ..."* [Pop59]. Some of the reasons for such critical attitudes towards (symbolic) AI include the brittleness, domain specificity, and predictability of (symbolic) AI programs. The aim of this paper is to show that symbolic AI programs can, if written well and given appropriate representations, behave in ways that appear "creative"; that is, they can behave with less brittleness, more domain independence, and more unpredictability than is generally assumed.

This paper focuses on mathematical creativity. Although problem solving in mathematics is usually well defined, there is considerable scope for creativity. For example, in the 18th century Euler discovered a novel method for summing an arithmetic series; that is, he discovered a new method for re-expressing an arithmetic series in terms of its first term, the difference between successive terms, and the number of terms in the series.

Consider the arithmetic series,

$$S = a + (a+d) + (a+2d) + ... + (a+nd)$$

That is, an arithmetic series of $n+1$ terms whose first term is a and in which the difference between successive terms is the constant d. Euler's insight was to invert this series,

$$S = (a+nd) + ... + (a+2d) + (a+d) + a$$

Adding these two series together gives,

$$\begin{array}{rcllllllll} 2S & = & a & + & (a+d) & + & (a+2d) & + ... + & (a+nd) & + \\ & & (a+nd) & + & (a+(n-1)d) & + & (a+(n-2)d) & + ... + & a & \end{array}$$

Reordering this expression by collecting together those pairs of terms which are $n+1$ terms apart (vertically adjacent in the above equation), gives $n+1$ terms, each equal to $2a+nd$. Thus,

$$2S = (n+1)(2a+nd)$$

That is,

$$S = \frac{n+1}{2}(2a+nd)$$

Remarkably, Euler was just seven years old when he found this proof. The aim of this research is to try to reproduce such behaviour on a computer.

This paper is structured as follows. Section 2 introduces proof planning, a technique for describing proof strategies. Section 3 illustrates this idea by means of an example. In sections 4 to 6, problem solving strategies developed in one domain are mapped across onto new and analogous domains. Finally, section 7 gives some conclusions.

2 Proof Planning

To be able to discuss mathematical problem solving, we need a way of describing proofs, and proofs strategies. **Proof plans** have been developed in Alan Bundy's DReaM group (Discovery and Reasoning in Mathematics) as a way of specifying high-level proof strategies [Bun88]. Proof planning has been successfully applied to many different domains including inductive theorem proving, database integrity maintenance, and hardware configuration.

Proof plans are built in terms of **methods**, meta-level descriptions of compound proof steps. Methods encode discrete proof "chunks" (*eg.* a useful strategy in equation solving is to collect together occurrences of the unknown); every method has some **preconditions**, a set of conditions necessary for the method to apply (*eg.* unknown occurs more than once), and some **postconditions** that describe the result of the method (*eg.* unknown now only occurs once). Each method also has an **input** and **output**; these are schematic representations of the goal formula before and after application of the method. A particular method can capture many different proof "chunks" as it includes parameters that need to be instantiated to give an object-level proof (*eg.* the *number* of occurrences of the unknown).

A proof planner, called Clam, has been implemented in PROLOG that puts methods together to give proof plans [BvHFHS90]. Each proof plan built by Clam is tailored to the particular theorem being proved, with the preconditions of methods at the end of the plan being satisfied by the postconditions of methods earlier in the plan. For reasons of efficiency and brevity, methods are often only partial specifications of problem solving strategies; proof plans therefore need to be executed to create an object-level proof. Each method has an associated **tactic**, a program which tries to perform the associated object-level reasoning. The execution of tactics can occasionally fail; this gives proof planning a certain unpredictability.

Proof planning moves the search for a proof from the object-level to the meta-level, often resulting in great reductions in search. Proof planning can be thought of as a *creative* process since, for every new problem, methods are put together in a new way which is tailored to the problem at hand. In section 3, I'll look at proof planning within a single domain, whilst in sections 4 and 5 I'll look at how proof planning can map across onto new and analogous domains.

3 An Example

To illustrate some of the ideas behind proof planning, I'll give an example of building a proof plan to sum a series. That is, as in the Euler example, finding an expression for the sum of a series just in terms of primitive arithmetic operations like addition and multiplication. Such a **closed form** expression should not involve summation either explicitly (*ie.* it should not contain expressions mentioning "$\sum$") or implicitly (*eg.* it should not contain symbols like "$+ \ldots +$"). Many of the methods used in this (and the next two sections) for summing series are described in more detail in [WNB92]. For the sake of simplicity, I'll consider just finite series. By Dirichlet's theorem, most of the methods could, however, also be used for summing absolutely convergent infinite series.

By repeated application of the rule,

$$\frac{\mathrm{d}(u+v)}{\mathrm{d}x} = \frac{\mathrm{d}u}{\mathrm{d}x} + \frac{\mathrm{d}v}{\mathrm{d}x}$$

The derivative of a sum can be shown to be the sum of the derivatives,

$$\frac{\mathrm{d}}{\mathrm{d}x}(\sum u) = \sum \frac{\mathrm{d}u}{\mathrm{d}x}$$

Thus, one trick for summing a series is to find the sum of the integral of the series and then differentiate the result. For example, consider the series,

$$\begin{aligned} S_1 &= 1 + 2.x + 3.x^2 + ... + (n+1).x^n \\ &= \sum_{i=0}^{n} (i+1).x^i \end{aligned}$$

The **integrate** method transforms this series as follows,

$$\begin{aligned} S_1 &= \sum_{i=0}^{n} (i+1).x^i \\ &= \sum_{i=0}^{n} \frac{\mathrm{d}x^{i+1}}{\mathrm{d}x} \\ &= \frac{\mathrm{d}}{\mathrm{d}x}(\sum_{i=0}^{n} x^{i+1}) \end{aligned}$$

The sum inside the derivative is a geometric progression, a well known standard result; it can be easily solved by the **standard form** method (see next section) giving,

$$S_1 = \frac{\mathrm{d}}{\mathrm{d}x}(x.\frac{x^{n+1}-1}{x-1})$$

Differentiating this expression is now a purely algorithmic task. The proof plan for summing this series is thus the integrate method followed by the standard form method. Note how this simple two step proof plan is able to describe a complex and much longer proof.

Using PROLOG style notation, the **integrate** method can be represented by the following frame:

Name:	`integrate(X,Pos)`
Input:	`Goal`

Preconditions:	`exp_at(Goal,Pos,sum(I,A,B,U)),`
	`integrate(U,X,V),`
	`I \= X,`
	`simplify(U,V)`

Postconditions:	`replace(Pos,Goal,`
	`dif(X,sum(I,A,B,V)),NewGoal)`

Output:	`NewGoal`

The name of the method, `integrate(X,Pos)` includes two parameters, `X` and `Pos` which are the variable of integration and the position in the goal formula of the series to be summed. The input `Goal` is the theorem we wish to prove. In the previous example, this theorem is,

$$\exists s \ . \ s \ = \ \sum_{i=0}^{n}(i+1).x^i$$

In proving this theorem, we actually find a **witness** for s; that is, a closed form sum for the series[1], $\sum_{i=0}^{n}(i+1).x^i$.

The precondition `exp_at(Goal,Pos,sum(I,A,B,U))` finds a subexpression of the `Goal` at position `Pos` which is a series to be summed. The expression `sum(I,A,B,U)` is the internal representation for the sum of terms of the form `U` for `I=A` to `I=B`. That is,

$$\sum_{I=A}^{B} U$$

When we sum the series,

$$\sum_{i=0}^{n}(i+1).x^i$$

This precondition will instantiate I to i, A to 0, B to n and U to $(i+1).x^i$.

The precondition `integrate(U,X,V)` then integrates the individual terms, `U` in this series; `V` is the result of the integration, whilst `X` is the variable of integration. In our example, V is instantiated to x^{i+1}, and X to x. The precondition `I \= X`, checks that the variable of integration is not also the (bound) variable of summation.

The precondition `simplify(U,V)` restricts the application of the method to those sums in which the integral of the terms in the series, `V` are "simpler" to sum than the original terms, `U`. In our example, V (which is instantiated to x^{i+1}) is considered a simpler expression than U (which is instantiated to $(i+1).x^i$). It is possible to give a formal account of this intuitive concept in terms of a Knuth-Bendix term order (see [WNB92] for more details). Without this precondition, the integrate method would always be applicable; this would cause much unnecessary search. The motivation behind this precondition is that we only want to use this method when it makes the series easier to sum.

The postcondition `replace(Pos,Goal,dif(X,sum(I,A,B,V)),NewGoal)` substitutes the derivative of the new series for the original series, giving the transformed

[1]Note that this theorem has a trivial proof as one witness for s is simply the expression $\sum_{i=0}^{n}(i+1).x^i$. However, this is not closed form since it involves the non-primitive operation of summation. At the meta-level, we therefore impose the extra condition that the witness for s be closed form.

goal, `NewGoal`. The expression `dif(X,sum(I,A,B,V)` is the internal representation for the derivative,

$$\frac{\mathrm{d}}{\mathrm{d}X}(\sum_{I=A}^{B} V)$$

In our example, this will be instantiated to,

$$\frac{\mathrm{d}}{\mathrm{d}x}(\sum_{i=0}^{n} x^{i+1})$$

As explained in the last section, proof planning can occasionally be unpredictable because, for reasons of efficiency, methods are often only *partial* specifications of problem solving strategies. Consider, for example:

$$S_2 = \sum_{i=0}^{n} i.e^{\pi.i}$$

where π is 3.1415, and e is 2.7183. The proof plan built for this problem is entirely analogous to the previous one for S_1. It includes the rather novel task of differentiating with respect to π. That, is:

$$\sharp\sharp\sharp\sharp \qquad S_2 = \frac{\mathrm{d}}{\mathrm{d}\pi}(\sum_{i=0}^{n} e^{\pi.i}) \qquad \sharp\sharp\sharp\sharp$$

This occurs because the preconditions to the **integrate** method merely check that we don't differentiate with respect to the (bound) index of summation. Actually, differentiating (and integrating) with respect to the constant π gives the correct answer since the series could be generalised (replacing π with x), summed and then specialised (replacing x with π).

4 Mapping between Domains

Methods are often general purpose proof strategies; many can therefore be mapped (sometimes with changes) onto new domains. In this section, I'll give an example of one method, the **standard form** method which was developed for summing series, but that can be easily mapped across to several other domains.

The standard form method uses **rippling**, a method itself developed for another domain, inductive theorem proving [BvHSI90]. Rippling is based upon the observation that we frequently want to transform one expression into something very similar. Consider for example, the following subgoal from the last section:

$$S_3 = \sum_{i=0}^{n} x^{i+1}$$

We can compare this goal with the standard result for the sum of the geometric progression:

$$\sum_{i=0}^{n} x^i$$

The metaphor is a Scottish loch; the reflection of the mountains in the loch represent the goal, whilst the mountains themselves represent what we know, the standard result. The reflection is not perfect because of the extra "+1" which appears in the goal. Such an expression is called a wavefront; we represent it with a box:

$$\sum_{i=0}^{n} x^{\boxed{\underline{i}+1}}$$

The underlined expression within the box represents that part of the reflection which mirrors the mountain exactly. If we delete everything in the box which is not underlined, we get a perfect reflection. The wavefront in the box is like a ripple on the surface of the loch which prevents a perfect reflection. The standard form method moves this wavefront upwards (that is, higher up the expression) till it is out of the way leaving behind (some function of) the standard result.

The standard form method annotates the goal with the wavefronts which are to be moved out of the way by the rippling method. In this case, the standard form method adds the following annotations:

$$S_3 = \sum_{i=0}^{n} x^{\boxed{\underline{i}+1}}$$

To move this wavefront upwards and out of the way, we need various rules for manipulating exponentials and sums; these are called **wave rules**:

$$\begin{aligned} x^{\boxed{\underline{i}+1}} &\Rightarrow \boxed{x.\underline{x^i}} \\ \sum \boxed{x.\underline{u}} &\Rightarrow \boxed{x.\underline{\sum u}} \end{aligned}$$

Notice how each of these rules, when viewed left to right, moves the wavefronts (the boxes) higher up and towards the outside of the expression. The rippling method uses these wave rules to move the wavefront on the sum upwards eventually leaving a function of the sum of a geometric progression.

$$\begin{aligned} S_3 &= \sum_{i=0}^{n} x^{\boxed{\underline{i}+1}} \\ &= \sum_{i=0}^{n} \boxed{x.\underline{x^i}} \\ &= \boxed{x.\underline{\sum_{i=0}^{n} x^i}} \end{aligned}$$

We can now replace $\sum x^i$ by the standard result for the sum of a geometric progression, giving the answer:

$$S_3 = x.\frac{x^{n+1}-1}{x-1}$$

The standard form method can be mapped across to work in many other domains: products, derivatives, integrals ... For instance, to transform the standard form method from sums to integrals we merely need to replace the wave rules for manipulating sums by some (not completely analogous) wave rules for manipulating integrals and to replace the standard results for summing series by some standard results for integration. This transformed standard form method can cope with a wide variety of integrals. As an example, it builds the following proof:

$$\begin{aligned} I_1 &= \int e^{\boxed{\underline{x}+1}}\,\mathrm{d}x \\ &= \int \boxed{e.\underline{e^x}}\,\mathrm{d}x \\ &= \boxed{e.\underline{\int e^x\,\mathrm{d}x}} \\ &= e.e^x \end{aligned}$$

This proof is entirely analogous in strucuture to that for the last series, S_3. However, since some of the rules for manipulating integrals are very different to those for manipulating sums, the transformed standard form method can produce proofs of a very different structure to those produced by the original standard form method for summing series. In mapping the standard form method between domains, we *merely* needed to change the rules used. As I'll show in the next section, the mapping between domains can be more complicated than this.

5 Mapping between Analogical Domains

Analogy plays an important rôle in mathematical creativity [Pol65]. Mapping methods onto analogical domains therefore seems an interesting exercise. Unlike much analogical reasoning [Gen89, Kli71] this involves the mapping not of object-level terms but of meta-level methods. Consider, for example, the **telescope** method for summing series. In this method, one term in the series cancels against a successive term by subtraction, leaving just the two end terms. For example:

$$\begin{aligned} \sum_{i=1}^{n}\frac{1}{i.(i+1)} &= \sum_{i=1}^{n}\frac{1}{i}-\frac{1}{i+1} \\ &= (\frac{1}{1}-\frac{1}{2})+(\frac{1}{2}-\frac{1}{3})+(\frac{1}{3}- \quad .... \quad -\frac{1}{n})+(\frac{1}{n}-\frac{1}{n+1}) \end{aligned}$$

Now, like Euler, we can reorder this series to give:

$$\begin{aligned}
\sum_{i=1}^{n} \frac{1}{i.(i+1)} &= \frac{1}{1} + (\frac{1}{2} - \frac{1}{2}) + (\frac{1}{3} - \frac{1}{3}) + \quad .... \quad + (\frac{1}{n} - \frac{1}{n}) - \frac{1}{n+1} \\
&= \frac{1}{1} - \frac{1}{n+1} \\
&= \frac{n}{n+1}
\end{aligned}$$

The telescope method can be mapped onto an analogous method for calculating products in which one term in a product is cancelled against a successive term by division, leaving just the end terms of the product. For example,

$$\begin{aligned}
\prod_{i=1}^{n} (1 + \frac{1}{i}) &= \prod_{i=1}^{n} \frac{i+1}{i} \\
&= \frac{2}{1}.\frac{3}{2}.\frac{...}{3}. \quad .... \quad .\frac{n-1}{....}.\frac{n}{n-1}.\frac{n+1}{n}
\end{aligned}$$

Reordering this product gives:

$$\begin{aligned}
\prod_{i=1}^{n} (1 + \frac{1}{i}) &= 1.\frac{2}{2}.\frac{3}{3}. \quad .... \quad .\frac{n-1}{n-1}.\frac{n}{n}.n+1 \\
&= n+1
\end{aligned}$$

In this analogical mapping, addition maps onto multiplication, subtraction onto division, and the summation operator onto the product operator:

$$\begin{aligned}
+ &\Longrightarrow * \\
- &\Longrightarrow / \\
\sum &\Longrightarrow \prod
\end{aligned}$$

The cancellation of repeated addition by subtraction thus maps onto cancellation of repeated multiplication by division. The mapping of the telescope method from sums to products thus requires both the mapping of rules used by the method (*cf.* the mapping of the **standard form** method) and the mapping of object-level terms within the (preconditions and postconditions of the) method:

Name:	`telescope`
Input:	`Goal`
Preconditions:	`exp_at(Goal,Pos,sum(I,A,B,U)),` `rewrite(U,V(I) - V(I+1))`
Postconditions:	`replace(Pos,Goal,V(A)-V(B+1),NewGoal)`
Output:	`NewGoal`

$$\Vert$$
maps onto
$$\Downarrow$$

Name:	`telescope`
Input:	`Goal`
Preconditions:	`exp_at(Goal,Pos,prod(I,A,B,U)),` `rewrite(U,V(I) / V(I+1))`
Postconditions:	`replace(Pos,Goal,V(A)/V(B+1),NewGoal)`
Output:	`NewGoal`

The precondition, `rewrite(U,V(I) - V(I+1))` determines whether the terms in the series being summed can be rewritten into an appropriate difference. Note that `V` is a higher-order variable. If this is so, the postcondition, `replace(Pos,Goal,V(A)-V(B+1),NewGo` replaces the series by the expression `V(A)-V(B+1)` where `A` and `B` are the lower and upper limits of the series.

Similarly, the precondition, `rewrite(U,V(I) / V(I+1))` determines if the terms in the product can be rewritten into an appropriate fraction. If this is so, the postcondition, `replace(Pos,Goal,V(A)/V(B+1),NewGoal)` replaces the series by the expression `V(A)/V(B+1)` where `A` and `B` are the lower and upper limits of the product.

6 Another Analogical Mapping

As a final illustration of this idea of mapping between analogical domains, I'll show how the method of integration by parts can be mapped onto an analogical method for summation. This mapping highlights how summation is very much the discrete analog of the continuous notion of integration. The mapping calls upon the upper difference operator, $\triangle$ which was implicitly used in the telescope method. The upper difference operator is defined by,

$$\triangle\ v(x) \quad = \quad v(x+1) - v(x)$$

This is the discrete analog of the (continuous) differentiation operator,

$$\frac{\mathrm{d}}{\mathrm{d}x}\ v(x) \quad = \quad \lim_{\delta x \to 0} \frac{v(x+\delta x) - v(x)}{\delta x}$$

Integration by parts uses the identity,

$$\mathrm{d}(u.v) \quad = \quad u.\mathrm{d}v + v.\mathrm{d}u$$

Integrating both sides and subtracting gives,

$$\int u \, \mathrm{d}v \quad = \quad u.v - \int v \, \mathrm{d}u \tag{1}$$

The **integration by parts** method uses this equation to transform one integral, $\int u \, \mathrm{d}v$ into a (hopefully) simpler integral, $\int v \, \mathrm{d}u$.

Now, an analogous equation can be found for summation and the upper difference operator. Let E be the shift operator. That is,

$$\mathrm{E}v(x) \quad = \quad v(x+1)$$

By expanding out terms,

$$\triangle\, u.v \quad = \quad u. \triangle v + \mathrm{E}v. \triangle u$$

Summing both sides and subtracting gives,

$$\sum u. \triangle v \quad = \quad u.v - \sum \mathrm{E}v. \triangle u \tag{2}$$

Aside from the shift operation, this is directly analogous to (1). We can therefore map the integration by parts method onto an analogous **summation by parts** method. In this analogical mapping, integration maps onto summation and derivatives onto upper differences:

$$\begin{array}{rcl} \int \quad \mathrm{d}x & \Longrightarrow & \sum \\ \frac{\mathrm{d}}{\mathrm{d}x} & \Longrightarrow & \triangle \end{array}$$

This requires both the mapping of rules used by the method (as in the mapping of the standard form method) and the mapping of object-level terms within the preconditions and postconditions of the method (as in the mapping of the telescope method).

As an example of a proof constructed by the summation by parts method, consider,

$$S_4 \quad = \quad \sum_{i=0}^{n} i.H_i$$

Where H_m is the m-th Harmonic number,

$$H_m \quad = \quad \sum_{i=1}^{m} \frac{1}{i}$$

And where we will extend the concept of closed form sum to include functions of such Harmonic numbers.

Let $u = H_i$ and $\triangle v = i$.

Thus, $\triangle u = \frac{1}{i+1}$ and $v = \frac{i.(i-1)}{2}$.

Hence,

$$\begin{aligned}
S_4 &= \sum_{i=0}^{n} i.H_i \\
&= \left[\frac{i.(i-1)}{2}.H_i\right]_0^{n+1} - \sum_{i=0}^{n} \frac{i.(i+1)}{2}.\frac{1}{i+1} \\
&= \frac{n.(n+1)}{2}.H_{n+1} - \sum_{i=0}^{n} \frac{i}{2} \\
&= \frac{n.(n+1)}{2}.H_{n+1} - \frac{n.(n+1)}{4} \\
&= \frac{n.(n+1)}{2}.(H_{n+1} - \frac{1}{2})
\end{aligned}$$

This proof is entirely analogous to that built by the integration by parts method for the integral,

$$\begin{aligned}
I_2 &= \int_1^n x.\ln(x)\,\mathrm{d}x \\
&= \left[\frac{x^2}{2}.\ln(x)\right]_1^n - \int_1^n \frac{x^2}{2}.\frac{1}{x}\,\mathrm{d}x \\
&= \frac{n^2}{2}.\ln(n) - \int_1^n \frac{x}{2}\,\mathrm{d}x \\
&= \frac{n^2}{2}.\ln(n) - \left[\frac{x^2}{4}\right]_1^n \\
&= \frac{1}{2}.(n^2.\ln(n) - \frac{n^2}{2} + \frac{1}{2})
\end{aligned}$$

Note how $\ln(x)$ in integration is analogous to H_i in summation. This is because $\ln(x)$ behaves very similarly in the domain of integration to H_i in the domain summation; for example, $\int \frac{1}{x}\,\mathrm{d}x$ equals $\ln(x)$ whilst $\sum \frac{1}{i}$ equals H_i. Thus, the analogy must also map logarithms in integration onto Harmonic numbers in summation. It would be very interesting to see how much further this analogy between integration and summation can be taken, and, in particular, to see how many other methods from the domain of integration can be mapped onto methods for summation.

7 Conclusions

The application of old methods to new problems and new domains generates behaviour which might be thought of as "creative". Mapping methods onto new (and possibly analogous) domains seems a powerful technique for generating new problem solving strategies. Some of the advantages of this approach include:

Great applicability. Methods can describe very general reasoning strategies; they can therefore be transformed to work in a wide range of different domains. For example, the standard from method can sum series, and be transformed to calculate products, derivatives and integrals.

Ability to fail. Since methods are often only partial specifications of proof strategies, they can fail, sometimes in interesting ways. This also allows analogies to break in a natural way.

Implicitness. The mapping between domains need not be supplied explicitly. For example, in transforming the standard form method from sums to integrals, we merely replace the rules for manipulating sums by some new rules for manipulating integrals; the standard form method is sufficiently powerful in its control of the reasoning to "handle" the mapping between the domains implicitly.

8 Acknowledgements

The author is supported by a SERC PostDoctoral Fellowship. All the members of the Mathematical Reasoning group in Edinburgh and the Mechanized Reasoning group in Trento are thanked for their contributions to this work. Especially thanks go to Alan Bundy, and Fausto Giunchiglia. Alex Numes has implemented these and other methods for summing series in the Clam proof planning system.

References

[Bun88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120, Springer-Verlag, 1988. Longer version available from Edinburgh as Research Paper No. 349.

[BvHFHS90] A. Bundy, van Harmelen. F., C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648, Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.

[BvHSI90] A. Bundy, F. van Harmelen, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 132–146, Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449.

[Gen89] D. Gentner. The mechanisms of analogical learning. In S. Vosmiadou and A. Ortony, editors, *Similarity and Analogical Reasoning*, pages 199–241, Cambridge University Press, 1989.

[Kli71] R.E. Kling. A paradigm for reasoning by analogy. *Artificial Intelligence*, 2, 1971.

[Pol65] G. Polya. *Mathematical discovery.* John Wiley & Sons, Inc, 1965. Two volumes.

[Pop59] K.R. Popper. *The Logic of Scientific Discovery.* Hutchinson, London, 1959.

[Tur63] A.M. Turing. Computing machinery and intelligence. In *Computers and Thought*, McGraw-Hill Book Company, 1963.

[WNB92] T. Walsh, A. Nunes, and A. Bundy. The Use of Proof Plans to Sum Series. To appear in D. Kapur, editor, *11th International Conference on Automated Deduction*, Springer-Verlag, 1992. Lecture Notes in Artificial Intelligence.