A Science of Reasoning

Alan Bundy

DAI Research Paper No.

April 26, 2004

Submitted to "Computational Logic: Essays in Honor of Alan Robinson", MIT Press, forthcoming.

Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland

A Science of Reasoning *

Alan Bundy

Abstract

This paper addresses the question of how we can understand reasoning in general and mathematical proofs in particular. It argues the need for a high-level understanding of proofs to complement the low-level understanding provided by Logic. It proposes a role for computation in providing this high-level understanding, namely by the association of *proof plans* with proofs. Proof plans are defined and examples are given for two families of proofs. Criteria are given for assessing the association of a proof plan with a proof.

1 Motivation: the understanding of mathematical proofs

The understanding of reasoning has interested researchers since, at least, Aristotle. Logic has been proposed by Aristotle, Boole, Frege and others as a way of formalising arguments and understanding their structure. There have also been psychological studies of how people and animals actually do reason. The work on Logic has been especially influential in the automation of reasoning. For instance, resolution, [Robinson 65], the paradigm technique for automatic reasoning, was based on the work of logicians such as Herbrand. Logic has been used for the representation of knowledge in artificial intelligence, where it has inspired the invention of new kinds of logics[1], *e.g.* for non-monotonic reasoning.

In this paper we argue that Logic is not enough to understand reasoning. It provides only a low-level, step by step understanding, whereas a high-level, strategic understanding is also required. Robinson has expressed this requirement with the slogan:

Proof = Guarantee + Explanation

where the "Guarantee" is provided by the logical, low-level, step by step check of soundness, and the "Explanation" is a high-level outline showing how the parts of the proof relate to each other. Many commonly observed phenomena of reasoning cannot be explained without such a high-level understanding. Furthermore, automatic reasoning is impractical without a high-level understanding.

We propose a science of reasoning which provides both a low- and a high-level understanding of reasoning. It combines Logic with the concept of *proof plans*, [Bundy 88]. We illustrate this with examples from mathematical reasoning, but it is intended that the science should eventually apply to all kinds of reasoning.

*The research reported in this paper was supported by SERC grant GR/E/44598 and an SERC Senior Fellowship to the author. I would like to thank other members of the mathematical reasoning group at Edinburgh for feedback, especially Frank van Harmelen, Colin Phillips, Mitch Harris and Toby Walsh.

[1] We adopt the convention of using uncapitalised 'logic' for the various mathematical theories and capitalised 'Logic' for the discipline in which these logics are studied.

2 The Role of Logic

A proof in a logic is a sequence of formulae where each formula in the sequence is either an axiom or is derived from earlier formulae in the sequence by a rule of inference. Each mathematical theory defines what it means to be a formula, an axiom and a rule of inference. For instance, many mathematical theories use the *rewriting* rule of inference:

$$\begin{array}{c} \Phi[L'] \qquad\qquad L \Rightarrow R \\ \\ \Phi[R]\sigma \end{array}$$

where σ is a substitution such that $L\sigma \equiv L'$. A typical *rewrite rule* in a theory of natural number arithmetci might be:

$$even(s(s(u)) \quad \Rightarrow \quad even(u)$$

Axioms and rules of inference are sometimes divided into two sets: the theory specific ones and those of the underlying logic. Further examples of theory specific rewrite rules are given in table 1 and of theory specific rules of inference are given in figure 1

$v = v$	$\Rightarrow$	$true$	$\boxed{s(\boxed{u_1})} = \boxed{s(\boxed{u_2})}$	$\Rightarrow$	$u_1 = u_2$
$0 + w$	$\Rightarrow$	w	$\boxed{s(\boxed{u})} + v$	$\Rightarrow$	$\boxed{s(\boxed{u + v})}$
$even(0)$	$\Rightarrow$	$true$	$even(\boxed{s(s(\boxed{u}))})$	$\Rightarrow$	$even(u)$
$even(s(0))$	$\Rightarrow$	$false$			
$0 \times w$	$\Rightarrow$	0	$\boxed{s(\boxed{u})} \times v$	$=$	$(u \times v)\boxed{+v}$

The last three rows are formed from recursive definitions of arithmetic functions and predicates. On the left hand side are the base cases and on the right hand side the step cases. The boxes around some expressions define wave fronts *which are explained in §4.*

Table 1: Rewrite Rules

Thus Logic provides a low-level explanation of a mathematical proof. It explains the proof as a sequence of steps and shows how each step follows from previous ones by a set of rules. Its concerns are limited to the soundness of the proof, and to the truth of proposed conjectures in models of logical theories.

3 Evidence for Higher-Level Explanations

While Logic provides an explanation of how the steps of a proof fit together, it is inadequate to explain many common observations about mathematical proofs. These observations are so common that they have, as afar as we are aware, escaped the attention of psychologists, although they have been noticed by mathematicians reflecting on the processes of theorem proving, *e.g.* [Pólya 65]. We list them below and hope that other mathematicians will immediately recognise their validity.

- Mathematicians distinguish between understanding each step of a proof and understanding the whole proof. It is possible to understand at one level without understanding at the other — either way round. A step by step under-

$$\frac{\Gamma \vdash P(0), \qquad \Gamma, x' : pnat, P(x') \vdash P(\boxed{s(\underline{x'})})}{\Gamma, x : pnat \vdash P(x)}$$

$$\frac{\Gamma \vdash P(0), \quad \Gamma \vdash P(s(0)), \quad \Gamma, x' : pnat, P(x') \vdash P(\boxed{s(s(\underline{x'}))})}{\Gamma, x : pnat \vdash P(x)}$$

Two rules of inference are given of mathematical induction in natural number arithmetic. If the formulae above the line can be proved then those below the line can be deduced. We will normally be working backwards, so to prove the formula below the line we will prove those above. Each formula above the line will give rise to a case*. Those on the left are* base cases*, those on the right are* step cases*. The* Γ*s are additional hypotheses that are inherited by each case. In each rule,* x *is the* induction variable*. To prevent ambiguity it is renamed to* x' *in the step cases. In these step cases, the expression* $P(x')$ *is called the* induction hypothesis *and the expression* $P(\ldots x' \ldots)$ *is called the* induction conclusion*. The induction conclusion differs from the induction hypothesis by the boxed sub-expressions. We call these* wave fronts *for reasons which are explained in §4.*

Figure 1: Induction Rules of Inference

standings is adequately represented by a logical proof. We propose representing an overall understanding by associating a proof plan with each proof.

- Mathematicians recognise families of proofs which contain common structure. These families are sometimes named, *e.g.* diagonalization arguments, but, more often, are not. We propose representing such common structures with proof plans.
- Mathematicians use their experience of previously encountered proofs to help them discover new proofs. We speculate that the old and new proofs belong to the same family, and that a representation of their common structure, *e.g.* a proof plan, is used to guide the search for the new proof.
- Mathematicians distinguish between 'interesting' and 'standard' steps of a proof. We speculate that the 'standard' steps are associated with a previously known proof plan, whereas the 'interesting' steps are not.
- Mathematicians often have an intuition that a conjecture is true, but this intuition is fallible. One way of explaining this intuition is that they have already associated a proof plan with the conjecture, that the proof is likely to unpack into a proof, but that this unpacking may fail.
- Students of mathematics, presented with the same proofs, learn from them with varying degrees of success. One explanation is that the more successful students are equipped by previous experience with the concepts required to extract general proof plans from specific proofs.

4 Common Structure in Proofs

As an example of a common structure in a family of proofs, consider the two proofs in figures 2 and 3, which are derived using the rewrite rules in figure 1 as axioms.

Both proofs are by induction, but using different induction rules of inference. The two induction rules used are given in figure 1. The step cases of each induction proof consist of a sequence of steps we have labelled *ripple out* followed by a step we have labelled $fertilize$. In these two examples the $fertilize$ step consists of proving the induction conclusion by matching it to the induction hypothesis. More generally, $fertilize$ identifies sub-expressions of the induction conclusion that match the induction hypothesis and replaces them by $true$. The *ripple_out* steps can be seen as preparing the induction conclusion for $fertilize$ by moving the constructor function, s, from its innermost position around the induction variable, x', out, in stages, until it can be removed altogether. In other examples it may not be possible to remove all wave fronts, but only to move them to the outside of the induction conclusion.

The rewrite rules deployed by *ripple_out* are called *wave rules*. They have the general form:

$$\begin{aligned} & F(\boxed{S_1(\boxed{U_1^1,\ldots,U_1^{m_1}})},\ldots,\boxed{S_n(\boxed{U_n^1,\ldots,U_n^{m_n}})}) \\ \Rightarrow \quad & \boxed{T(\boxed{F(U_1^{q_1},\ldots,U_n^{q_n}),\ldots,F(U_1^{r_1},\ldots,U_n^{r_n})})} \end{aligned}$$

where $1 \leq q_i, r_i \leq m_i$ for $1 \leq i \leq n$. F, the S_is and T are terms with one distinguished argument. T may be empty, but F and the S_is must not be.

We will say that this rule ripples the S_is past F to become T. The sub-expressions which are being moved are called *wave fronts*, and are indicated by the boxes around them. In most wave rules n and the m_j are all 1, in which case this general form reduces to:

$$F(\boxed{S(\boxed{U})}) \quad \Rightarrow \quad \boxed{T(\boxed{F(U)})}$$

The first three rewrite rules on the right hand side of figure 1 are the wave rules used in the proofs of figures 2 and 3. The induction schema in each proof puts the induction conclusion into a form where *ripple_out* will succeed. To see this try proving the same theorems with y as induction variable, or use the induction schema of the first proof to prove the second theorem.

ripple_out works by identifying those sub-expressions of the induction conclusion which cause it not to match the induction hypothesis. These too are called *wave fronts*. We can mark them with boxes in the same way as the rule wave fronts. For the associativity of $+$ induction conclusion this gives:

$$\boxed{s(\boxed{x'})} + (y+z) \quad = \quad (\boxed{s(\boxed{x'})} + y) + z$$

Wave rules are applied to induction conclusions in such a way that *each wave front in a rule matches all or part of a wave front in the induction conclusion.* The step cases of the recursive definition of $+$ (see table 1) can be applied in this way to the two wave fronts in our example. The effect is to ripple the wave fronts past the $+$ signs that dominate them, but two new wave fronts are created further out.

$$\boxed{s(\boxed{x' + (y+z)})} \quad = \quad \boxed{s(\boxed{x'+y})} + z$$

Another application of the same wave rule brings both wave fronts to immediately below the $=$ sign.

$$\boxed{s(\boxed{x' + (y+z)})} \quad = \quad \boxed{s(\boxed{(x'+y)+z})}$$

$x : pnat$
$y : pnat$
$z : pnat$
$\vdash \underline{x + (y + z) = (x + y) + z}$

induction

$\vdash \overline{\underline{0 + (y + z)} = (\underline{0 + y}) + z}$

$2 \times base$

$\vdash \underline{\overline{y + z} = \overline{y} + z}$

sym_eval

$\Box$

$\overline{x' : pnat}$
$\overline{x' + (y + z) = (x' + y) + z}$
$\vdash \underline{s(x') + (y + z)} = (\underline{s(x') + y}) + z$

$2 \times wave$

$\vdash \overline{s(x' + (y + z))} = \underline{\overline{s(x' + y)} + z\}}$

wave

$\vdash \underline{s(x' + (y + z)) = \overline{s((x' + y) + z)}}$

wave

$\vdash \underline{\overline{x' + (y + z) = (x' + y) + z}}$

fertilize

$\Box$

ripple out

ind strat

This proof and the one in figure 3, are arranged in a tree to illustrate the dependencies between the formulae. The proofs procede backwards from theorems to axioms, i.e. *the logical direction of the proofs is from the tips to the root. A formula is associated with each non-tip node of the tree and a rule of inference and some axioms are associated with the arrows pointing from it. This formula is derived using the rule of inference and the axioms together with the formulae associated with its daughter nodes. The rule of inference and the axioms associated with each arrow are summarised in words like wave, $fertilize$,* etc. *These represent* tactics, *which are explained in §5 below. Each formula is a sequent of the form $H \vdash G$, where H is a list of hypotheses and G is a goal. Formulae of the form $X : T$ are to be read as "X is of type T". pnat is the type of Peano natural numbers. The top sequent is a statement of the theorem. Each subsequent sequent is obtained by rewriting some subexpressions in the one above it. A subexpression to be rewritten is underlined and the subexpression which replaces it is overlined and connected to it with an arrow. Only newly introduced hypotheses are actually written in subsequent sequents; sequents are to be understood as inheriting all those hypotheses above them in their proof.*

Figure 2: Proof of the Associativity of +

$x : pnat$ $y : pnat$	$\vdash \underline{even(x) \wedge even(y) \rightarrow even(x+y)}$	ind strat
	induction	
$\vdash \overline{\underline{even(0)} \wedge even(y) \rightarrow even(\underline{0+y})}$	$\overline{x' : pnat}$ $\overline{even(x') \wedge even(y) \rightarrow even(x'+y)}$	ripple out
$2 \times base$	$\vdash \overline{\underline{even(s(s(x')))} \wedge even(y) \rightarrow even(\underline{s(s(x'))+y})}$	
$\vdash \underline{\overline{true} \wedge even(y) \rightarrow even(\overline{y})}$	$2 \times wave$	
sym_eval	$\vdash \overline{even(x')} \wedge even(y) \rightarrow even(\overline{s(\underline{s(x')+y})})$	
$\square$	*wave*	
$\vdash \overline{\underline{even(s(0))} \wedge even(y) \rightarrow even(s(0)+y)}$	$\vdash even(x') \wedge even(y) \rightarrow \underline{even(s(\overline{s(x'+y)}))}$	
base	*wave*	
$\vdash \underline{\overline{false} \wedge even(y) \rightarrow even(s(0)+y)}$	$\vdash even(x') \wedge even(y) \rightarrow \underline{\overline{even(x'+y)}}$	
sym_eval	*fertilize*	
$\square$	$\square$	

Figure 3: Proof of the Even+ Theorem

Finally, an application of the substitution rule for = (see table 1) removes these wave fronts altogether (T is empty).

$$x' + (y + z) \quad = \quad (x' + y) + z$$

This expression is identical to the induction hypothesis.

Another example of common structure can be found in the two examples of equation solving in figure 4. In each example the equation is first rewritten into a homogeneous form in which all occurrences of the unknown, x, appear within identical terms; a new unknown, y, is defined equal to this term; the unknown of the equation is changed from x to y and, hence, made into a quadratic; and then this quadratic is solved for y.

	$4.\log_x 2 + \log_2 x = 5$		$\cos x + \sin^2 x = -1$
	homogenization		*homogenization*
	$\frac{4}{\log_2 x} + \log_2 x = 5$		$\cos x + 1 - \cos^2 x = -1$
	change of unknown		*change of unknown*
$y = \log_2 x$	$\frac{4}{y} + y = 5$	$y = \cos x$	$y + 1 - y^2 = -1$
isolation	*poly normal form*	*isolation*	*poly normal form*
$x = 2^y$	$y^2 - 5.y + 4 = 0$	$x = \cos^{-1} y + 2.\pi.n$	$y^2 - y - 2 = 0$
	quadratic		*quadratic*
	$y = 1 \vee y = 4$		$y = -1 \vee y = 2$

Figure 4: Equation Solving

Other authors have also identified common structure in families of proofs. For instance,

- [Bledsoe *et al.* 72] describes the common structure in theorems about limits of functions in analysis. This common structure was implemented as the *limit heuristic* and used to guide proofs of such theorems.
- [Wos & McCune 88] describes the common structure in attempts to find fixed points combinators. This common structure was implemented as the *kernel method* and used to guide the search for such fixed points.
- [Pólya 65] describes the common structure in ruler and compass constructions. This common structure was implemented by [Funt 73] and used to guide the search for such constructions.

One approach to capturing such common structure is to build a derived rule of inference out of a sequence of rules of inference and axioms (see, for instance, work on partial evaluation in logic programming, [Komorowski 82], explanation based generalization, [Mitchell *et al.* 86], or the Isabelle system, [Paulson 88]). It is not possible to capture the common structures observed here in a derived rule of inference. Note that different wave rules are used in different examples of *ripple_out*. Moreover, *ripple_out* is fallible; it will fail if a suitable wave rule is not available, *e.g.* there is no wave rule in figure 1 to ripple s past *even* in $even(s(x'))$. Similar remarks hold for homogenization.

5 Proof Plans

Common structure in proofs *can* be captured in *proof plans.* A proof plan consists of two parts: a *tactic* and a *method.*

A tactic is a procedure which constructs part of a proof by applying a sequence of rules of inference. High-level tactics are defined in terms of lower-level subtactics. The lowest level tactics will apply individual rules of inference. Note that tactics can only construct proofs by applying rules of inference, so that these proofs are guaranteed to be sound. The particular rules to be applied are determined dynamically after analysis of the current situation by the methods and the tactics. Tactics may be fallible, *i.e.* a tactic may not be able to determine an appropriate sequence of rules to apply, or one of its rules may be inapplicable.

A method is a partial specification of a tactic. It consists of preconditions which must be satisfied before the tactic is executed and some effects which will be true provided the tactic application is successful. Preconditions are normally checked in a partially instantiated state, with unbound variables being instantiated during checking. This has the side effect of analysing the current conjecture and, hence, determining the rules of inference that the tactic will apply.

Proof plans have been implemented in the Clam-Oyster system, [Bundy *et al.* 88a]. Oyster is a theorem prover for Intuitionist Type Theory. Clam is a plan formation program which has access to a number of general-purpose tactics and methods for inductive proofs. This system has been used to control the search for inductive proofs about natural numbers and lists. Clam constructs a special-purpose proof plan for each conjecture out of its methods and tactics. The tactic of the proof plan is then executed. It instructs Oyster to build a proof of the conjecture. The search for a proof plan at the meta-level is considerably cheaper than the search for a proof at the object-level, which makes proof plans a practical solution to the problems search control in automatic theorem proving.

Writing methods requires designing new terminology to represent the relevant common structures in proofs from the same family. For instance, we have identified rippling out as a common structure in inductive proofs, and have introduced the new terminology *wave rule* and *wave front* to describe this new structure. Clam represents this terminology as the signature of a *meta-level, logical theory.* The preconditions and effects of its methods are represented by formula in this theory. They describe the syntactic structure of formulae and proofs in the *object-level theory*, *i.e.* the theory in which it is proving the theory. Reasoning with the methods is a form of *meta-level inference.* The meta-level and object-level theories may be based on the same underlying logic — or they may not. The essential difference between them are their signatures, *i.e.* their predicates, functions, types, *etc.*

Clam has general-purpose tactics which produce the steps in figures 2 and 3. The labels on the arrows in these figures are the names of these tactics. In addition, there are composite, general-purpose tactics, *ripple_out* and *ind_strat*, which are defined in terms of these other tactics. *ind_strat* is responsible for almost the whole of both proofs (all but the *sym_eval* steps). Figure 5 gives examples of special-purpose tactics for the associativity and commutativity of $+$, built out of the general-purpose tactics: *ind_strat* and *sym_eval*. These two general-purpose tactics are enough to construct a surprisingly large number of special-purpose tactics for simple inductive proofs.

```
ind_strat(s(x), x:pnat) then        ind_strat(s(x), x:pnat) then
    [ sym_eval,                         [ ind_strat(s(y), y:pnat) then
      sym_eval                                [ sym_eval,
    ]                                           sym_eval
                                              ],
                                          ind_strat(s(y), y:pnat) then
                                              [ sym_eval,
                                                sym_eval
                                              ]
                                        ]
```

Figure 5: Special-Purpose Tactics for the Associativity and Commutativity of +

The preconditions of the method for *ind_strat* look ahead to the *ripple_out* tactic and the wave rules that it deploys. They use this analysis to suggest a form of induction that is likely to lead to a successful *ripple_out* (for details see [Bundy *et al.* 88b]).

A simplified declarative reading of the preconditions of *ind_strat* can be given in English as follows.

- Suppose that $Conj$ is the conjecture to be proved by induction on variable X using an induction rule of inference with wave front $IndTerm$.
- X must occur in $Conj$;
- For each occurrence of X in $Conj$, if X were replaced by $IndTerm(X')$ then there must be a wave rule which applies to $Conj$ in such a way that its wave front matches $IndTerm$.

Factors like multiple induction variables, rules with multiple wave fronts and $IndTerm$s that require rippling out in several stages, complicate these preconditions. We avoid these complexities here. The above reading conveys the basic idea.

These preconditions ensure that if an induction rule with wave front, $IndTerm$, is used then at least the first round of *ripple_out* will succeed. They are usually checked with $Conj$ instantiated to some specific conjecture, but X and $IndTerm$ uninstantiated. The process of checking instantiates X and $IndTerm$ to some values for which the preconditions are true — effectively analysing the conjecture to suggest a form of induction that will make *ripple_out* likely to succeed. In practice, the checking process is flexible so that partial success is accepted when total success is not possible. For instance, if the conjecture is $x + y = y + x$ then no choice of induction variable fully meets the preconditions, but x/X and $s(\ldots)/IndTerm$ is accepted as one of the two best available alternatives.

These tactics and methods are based on an analysis and rational reconstruction of the work of Boyer and Moore, [Boyer & Moore 79], and Aubin, [Aubin 75]. In particular, the term 'fertilize' is due to Boyer and Moore, and the term 'ripple out' is due to Aubin (although we have considerably extended the usage of these two terms). The idea of tactics came originally from the LCF system, [Gordon *et al.* 79]. Oyster is based on the Nuprl system, [Constable *et al.* 86]. The idea of formally specifying tactics using methods and reasoning with methods to form proof plans, is first described in [Bundy 88]. This idea developed from earlier work on meta-level inference in the Press system, [Bundy & Welham 81]. Press is a symbolic equation solver for equations such as those in figure 4. Polya advocates the use of plans in human problem solving in [Pólya 65] and gives examples of their use. This work is a major source of inspiration for proof plans.

6 The High-Level Understanding of Proofs

Thus a high-level explanation of a proof of a conjecture is obtained by associating a proof plan with it. The tactic of this proof plan must construct the proof. The method of this proof plan must describe both the preconditions which made this tactic appropriate for proving this conjecture and the effects of this tactic application on the conjecture. It must also describe the role of each sub-tactic in achieving the preconditions of later sub-tactics and the final effects of the whole tactic.

In fact, this association provides a multi-level explanation. The proof plan associated with the whole proof provides the top-level explanation. The immediate sub-tactics and sub-methods of this proof plan provide a medium-level explanation of the major sub-proofs. The tactics and methods associated with individual rules of inference provide a bottom-level explanation, which is similar to that already provided by Logic.

The general-purpose tactics and methods which we will use to build proof plans, and the association of proof plans with proofs will constitute the theories of our science of reasoning.

7 What is the Nature of our Science of Reasoning?

Before we can dignify this proposed study of the structure of proofs with the name *science* we must address a fundamental problem about the nature of such a science. Traditional sciences like Physics and Chemistry study physical objects and the way they interact. The subject of our proposed science is proof plans. But proof plans are not physical objects. If they can be said to exist at all it is in the minds of mathematicians proving theorems, teachers explaining proofs and students understanding them. Physicists assume that the electrons in the apple I am eating are essentially the same as the electrons in some distant star. But proof plans will differ from mind to mind and from time to time. There will be billions of such proof plans. Are we doomed merely to catalogue them all? Given the difficulty of discovering the nature of even one such proof plan, what a difficult and ultimately pointless task this would be. We would prefer to narrow our focus on a few representative proof plans. But on what basis could these few be chosen?

Fortunately, this is not a new problem. It is one faced by all human sciences to some extent and it is one that has been solved before. Consider the science of Linguistics. Judged by their utterances, each person has a slightly different version of what counts as a sentence. Furthermore, this changes over time. To have a science of Linguistics with billions of different grammars would be impractical, and would make testing hypotheses and repeating experiments impossible. Fortunately, these different grammars would all have a lot in common — even those from different languages.

The goal adopted by linguists is to try and capture this commonality. Their aim is to build a definitive grammar for each human language which will analyse all and only, the grammatical sentences of that language. They try to make these grammars as parsimonious as possible, so that they capture the maximum amount of generality within and between languages. There are rival grammars competing against each other, none of which yet fully meets all these criteria. Linguists do not claim that everyone or anyone has this target grammar stored in their head — nor, indeed, that anyone has a grammar at all — only that it *specifies* the grammatical sentences.

How can linguists tell what are the grammatical sentences? They must start with some empirical study of human utterances. If they are native speakers of the language under study, then they could confine these initial observations to their own utterances and to reflection on their own intuitions about. If they are not native speakers themselves, then they must observe some. However, all native speakers sometimes utter ungrammatical sentences, either through ignorance or error. Linguists must, therefore, look for a concensus and they must consult experts. Experts here are people who have studied the language in question and are recognised authorities on its grammaticality. These experts are particularly useful for distinguishing between utterances that are grammatical and those that are readily understood and widely used, but ungrammatical.

Many linguists believe that all human languages have a biologically based, common structure, *cf.* Chomsky's 'Universal Grammar', but even allowing for this, much of the grammatical structure of languages is a product of, essentially arbitrary, human decisions. Some of these human decisions have been codified and can be consulted as expert testimony, *e.g.* Fowler's "Modern English Usage". Thus linguistic grammars based on these codifications have a normative flavour; they act as authorities on what is to be regarded as grammatical. We will see this mixture of the empirical, reflective and normative basis of grammars, re-emerging in proof plans.

Another example is Logic itself. Again judged by the arguments people produce, the logical laws differ between minds and vary over time. Logicians do try to capture this variety, but confine themselves to a few logics which specify 'correct' arguments. As with grammatical sentences, correct arguments are identified by initial observation of arguments actually used and consultation with experts to decide which of these are correct.

However, the situation is not quite the same as the linguistic one, and neither is the solution. Firstly, the range of variation is less for logical laws than grammatical rules. Logical laws are universal across human cultures, and probably across all forms of cognitive activity. Most variation can be accounted for by error. In contrast, grammatical rules have an inherent arbitrariness, which accounts for the wide variety of equally grammatical languages. This makes Logic even more normative than Linguistics; it prescribes not just the arbitrary rules agreed by a particular language culture, but the universal laws of correct reasoning.

Secondly, everyone is a 'native speaker' of logical thought, and so can progress much further by self observation and self consultation than they could if studying language. Therefore, empirical study plays a much smaller role in Logic than in Linguistics, although I will argue that it does play some role. Logic is usually regarded as an *a priori* science whose laws can be discovered by a process of reflection on the use of language. However, this is not to say that the discovery of logical laws is an easy matter in which you introspect and they come to you. On the contrary our current logical laws were determined only after millenia of careful study. Moreover, rival systems have been proposed and fiercely argued for. This process did involve empirical study, for instance Frege's invention of the quantifiers $\forall$ and $\exists$ arose from a rational reconstruction of subtle arguments in analysis by Cauchy and Weierstrass.

I place our proposed science of reasoning between Linguistics and Logic. Proof plans are more universal than grammatical rules, but it is possible to associate different, equally appropriate proof plans with the same proof. The study of proof plans appeals both to an empirical study of the way in which mathematicians structure their proofs and to reflection on the use of logical laws to put together proofs out of parts.

Thus there are strong precedents for a science that takes mental objects as its domain of study and tames the wide diversity of exemplars by imposing a normative explanation informed by reflectvion and empirical study. It only remains to propose criteria for associating proof plans with proofs that will enable us to prefer one proof plan to another. This we can do by appealing to general scientific principles. Our proposals are given in the next section.

8 Criteria for Assessing Proof Plans

If there were no criteria for the association of proof plans with proofs, then we could carry out our programme by associating with each proof an *ad hoc* tactic consisting of the concatenation of the rules of inference required to reproduce it, and constructing an *ad hoc* method in a similar way. This would not go beyond the existing logical explanation.

The only assessment criterion we have proposed so far is *correctness*, *i.e.* that the tactic of the proof plan associated with a proof will construct that proof when executed. We now discuss some other possible criteria. Most of them are concerned with assessing the collection of general-purpose proof plans that we are allowed to draw on in constructing a special-purpose proof plan to associate with a proof. Thus they only indirectly affect the association of proof plans with proofs.

- A criterion which would exclude such *ad hoc* proof plans is *intuitiveness*, *i.e.* that the way in which the proof is structured by a proof plan accords with our intuitions about how we structure the proof. Unfortunately, this is a very subjective criterion. It offers no way of settling disputes between rival intuitions. Nor does it meet the criteria of rigour, formality, *etc.* which we should demand of a science of reasoning. We need some more objective criteria.

- We could try to put intuitiveness on a scientific basis by appealing to the criterion of *psychological validity*, *i.e.* we could conduct experiments on mathematicians producing or studying proofs to show that all, most or some of them also structured a proof in the way suggested by some proof plan. Such experiments might be a good vehicle for discovering new proof plans. However, in a normative science, psychological validity cannot be the sole or even the main criterion for evaluating theories. That is, a normative science of reasoning must not depend on the errors, bias or short-sightedness of some particular mathematician on some particular occasion, but should be the consensus view of a number of mathematicians after careful reflection and debate. For this maturation of an initial intuition we need some non-psychological criteria.

- We can borrow two criteria from the list of desirable properties of proof plans given in [Bundy 88], namely *expectancy* and *generality*. *Expectancy* means that there must be a basis for predicting the successful outcome of a proof plan. One way of formalising this is to demand that if the preconditions of a proof plan are met by the input to a tactic and this tactic succeeds then the effects of the proof plan are met by the output of the tactic.

- *Generality* means that a proof plan gets credit from the number of proofs or sub-proofs with which it is associated and for which it accounts. But expectancy and generality cannot be the only criterion, or we could associate with every proof a single, totally general proof plan whose tactic consisted of the exhaustive application of a uniform proof procedure. This tactic would find each proof, but only after an indefinite amount of search.

- So another criterion might be *prescriptiveness*, *i.e.* a proof plan gets more credit the less search its tactic generates and the more it prescribes exactly what rules of inference to apply. However, this criterion might tend to produce fussy, over-detailed proof plans. This tendency will be counteracted to some extent by generality, but we might also want to add other criteria.

- For instance, we might add the criterion of *simplicity*, *i.e.* a proof plan gets more credit for being succinctly stated.

- Or we might add the criterion of *efficiency*, *i.e.* that a proof plan gets more credit when its tactic is computationally efficient.

- We might also add the criterion of *parsimony*, *i.e.* that the overall theory gets more credit the fewer general-purpose proof plans are required to account for some collection of proofs.

Thus we might start designing proof plans using the criteria of intuitiveness and psychological validity as sources of inspiration, but then use the criteria of correctness, expectancy, generality, prescriptiveness, simplicity, efficiency and parsimony to revise them.

These criteria are just the sort one would apply to any scientific theory, that is we have merely applied the scientific method to the understanding of mathematical proofs. Note that these criteria do not provide absolute judgements on the association of a particular proof plan with a proof. They only provide grounds for preferring one association to another — and even this is not a total order, the criteria might disagree on some association. They, therefore, permit the association of more than one proof plan with a proof. This seems reasonable; one can discover the same proof by viewing it from more than one direction. Nor can we normally expect a precise, mathematical proof that one proof plan is better than another on some dimension. Often the best we can do is appeal to an informal argument that one proof plan is *usually* better than the other on some dimension.

9 Using these Criteria to Assess Proof Plans

As an example of the use of these criteria consider the following rival proof plans for the steps marked 'ripple out' in figures 2 and 3, above. The purpose of these steps is to prepare the way for the $fertilize$ tactic. $fertilize$ matches the induction hypothesis against sub-expressions of the induction conclusion and replaces these sub-expressions with *true*.

The heuristic associated with these 'ripple out' steps in [Boyer & Moore 79] is what they call *symbolic evaluation*, but others have called *unfolding*, [Burstall & Darlington 77]. We have implemented this heuristic as the *sym_eval* proof plan, and used it in other parts of inductive proofs. *sym_eval* replaces all defined terms by their definitions, until no further rewriting is possible. In particular, recursively defined functions are rewritten with the base or step cases of their recursive definitions.

The rival proof plan, proposed in [Bundy *et al.* 88b], is *ripple_out*. The idea is to rewrite the induction conclusion, selectively, using wave rules, in an attempt to create within the rewritten induction conclusion one or more sub-expressions which match the induction hypothesis.

Since the step cases of recursive definitions can often be interpreted as wave rules, these two proof plans often suggest the same steps. However, sometimes they differ, and we will argue that in these cases it is usually *ripple_out* that is right and *sym_eval* that is wrong. To define 'right' and 'wrong' we will appeal to the criteria of expectancy and generality, which were defined in the last section.

To establish the superior expectancy of *ripple_out* over *sym_eval* we must show that the context of their use gives a better basis for predicting the success of *ripple_out* than of *sym_eval*. The context of their use is just after induction has produced an induction conclusion differing from the induction hypothesis only by the insertion of a wave front around each occurrence of the induction variable. Success means that the induction conclusion has been rewritten so that it contains one or more sub-expressions which match the induction hypothesis. $fertilize$ can then replace these by *true*.

ripple_out has an associated story about how this success is to be achieved. The wave rules ripple the wave fronts out until they are wholly on the outside of the induction conclusion, or until no wave rule can be found for some wave front. Each ripple removes an old wave front and replaces it with a new one further out; the rest of the expression is left as it was. Since the original induction conclusion differed from the induction hypothesis only by the insertion of the wave fronts, then a simple inductive argument establishes that the final induction conclusion will have the same property. If *ripple_out* does not terminate unsuccessfully by failing to find an appropriate rule, then it will terminate successfully with all the wave fronts wholly on the outside of the induction conclusion. In this case the inner part of the induction conclusion will match the induction hypothesis and $fertilize$ will apply. Examples of this can be found in figures 2 and 3 in the example two paragraphs below.

No such clear story is available to explain how *sym_eval* prepares for $fertilize$. It does have some empirical success, but this can be explained by the observation that many of the rules it applies are wave rules, and so it often mimics *ripple_out*. So *sym_eval* fails the expectancy criteria but *ripple_out* passes it.

To establish the superior generality of *ripple_out* over *sym_eval* we must show that *ripple_out* can find more proofs than *sym_eval*. The first half of the argument is that there are lots of wave rules that are not step cases of recursive definitions, and so lots of rewritings that *ripple_out* can do, but *sym_eval* cannot. In particular, we can interpret previously proved theorems as wave rules and use them as lemmas for *ripple_out* in subsequent proofs. For instance, the theorem proved in figure 3 can be interpreted as a wave rule for rippling $+v$ past *even* to get $\wedge even(v)$.

$$even(u\boxed{+v}) \quad \Rightarrow \quad even(u)\boxed{\wedge even(v)} \tag{1}$$

This theorem can then be used as a wave rule in the proof of the theorem:

$$even(y) \quad \rightarrow \quad even(x \times y)$$

Here is the *ripple_out* sequence from the inductive proof of this theorem.

$$\begin{array}{rcl}
even(y) & \rightarrow & even(\boxed{s(}x'\boxed{)} \times y) \\
even(y) & \rightarrow & even((x' \times y)\boxed{+y}) \\
even(y) & \rightarrow & even(x' \times y)\boxed{\wedge even(y)}) \\
(even(y) & \rightarrow & even(x' \times y))\boxed{\wedge (even(y) \rightarrow even(y))}
\end{array}$$

The s is rippled past the $\times$ using the step case of the recursive definition of $\times$ (see figure 1). The $+y$ is rippled past the *even* using our new wave rule (1). The $\wedge even(y)$ is rippled past the $\rightarrow$ using a propositional wave rule. Neither of these last two steps would have been suggested by *sym_eval*, because the required rewrite rules would not have been in its repertoire. Note that the induction conclusion now contains a copy of the induction hypothesis to which $fertilize$ can be applied, *i.e.* *ripple_out* has succeeded in preparing the way for $fertilize$.

The second half of the argument is that the greater selectivity of *ripple_out* does not cause it to miss proofs that *sym_eval* finds. In fact, we will make a stronger argument; that *sym_eval*'s exhaustive unfolding of definitions can cause it to miss some proofs. The selectivity of *ripple_out* comes from its insistence that the wave fronts of the wave rules should match with wave fronts in the induction conclusion. Since the wave fronts of the induction conclusion are those places in which it mismatches the induction hypothesis, it is precisely these wave fronts which must be rippled out in preparation for $fertilize$. Rippling out other parts of the induction conclusion will only cause the induction conclusion not to match the induction hypothesis. To see this, consider the associativity of + theorem with y replaced by $s(y)$. Applying *sym_eval* to the induction conclusion of this theorem produces the following derivation:

$$\begin{aligned}
\boxed{s(\,\boxed{x'}\,)} + (\underline{s(y)+z}) &= (\boxed{s(\,\boxed{x'}\,)} + s(y)) + z \\
\boxed{s(}\,x' + \overline{s(y+z)}\,\boxed{)} &= \boxed{s(}\,x' + s(y)\,\boxed{)} + z \\
\boxed{s(}\,x' + s(y+z)\,\boxed{)} &= \boxed{s(}\,(x' + s(y)) + z\,\boxed{)} \\
x' + s(y+z) &= (x' + s(y)) + z
\end{aligned}$$

All the unfoldings use the step case of the recursive definition of +. Note that the final equation does not match the induction hypothesis. The problem is caused by the rewrite indicated by the under- and over-lining. This step unfolds an expression, $s(y)$, which does not contain a wave front. The first occurrence $s(y)$ is unfolded, but the second occurrence is not unfoldable. Thus the left hand side of the equation is unnecessarily messed up by *sym_eval*. *ripple_out* would not make this unnecessary step, and consequently *would* find the required proof.

How does this pair of proof plans fare on the other criteria advanced in §8?

- **Correctness**: Both *sym_eval* and *ripple_out* are correct for a wide range induction conclusion rewritings. We have argued above that *ripple_out* is more often correct for such rewritings than *sym_eval*.

- **Intuitiveness**: *ripple_out* accords better with my intuitions about the way inductive proofs are structured than *sym_eval* does, but there is obviously room for other views.

- **Psychological Validity**: I know of no experiments to try and what proof plans might be used by people doing inductive proofs.

- **Prescriptiveness**: Neither of the proof plans search, so they are equal on this criterion. One could overcome the overeagerness of *sym_eval* by making a search version which tried all possible partial *sym_eval*s. Then *ripple_out* would score best on this criterion.

- **Simplicity**: The extra machinery required to make *ripple_out* selective makes it slightly more complex than *sym_eval*. On the other hand the rewrite rules deployed by *ripple_out* can be described in a simpler way than those deployed by *sym_eval*.

- **Efficiency**: This selectivity machinery also makes each *ripple_out* rewrite more expensive than the corresponding *sym_eval* one.

- **Parsimony**: Since *sym_eval* is required for other parts of the proof and *ripple_out* is not, *sym_eval* scores higher on this criterion.

Thus *ripple_out* does not uniformly score better than *sym_eval*, but it scores better on the most important criteria, namely expectancy and generality. The arguments advanced have been informal, although the ones for expectancy and generality could be tightened up into formal mathematical proofs.

10 The Role of the Computer

So far we have not involved the computer in this methodological discussion. One might expect it to play a central role. In fact, computers have no role in the *theory*, but play an important *practical* role. *Computation* plays a central role in the theory, because the tactics are procedures and they are part of the theory of our science of reasoning. It is not, strictly speaking, necessary to implement these tactics on a computer, since they can be executed by hand. However, in practice, it is highly convenient. It makes the process of checking that the tactics meet the criteria of the §8 both more efficient and less error prone. Machine execution is convenient:

- for speeding up correctness testing, especially when the proof plans are long, or involve a lot of search, or when a large collection of conjectures is to be tested;
- to automate the gathering of statistics, *e.g.* on size of search space, execution time, *etc*;
- to ensure that a tactic has been accurately executed; and
- to demonstrate to other researchers that the checking has been done by a disinterested party.

In this way the computer can assist the rapid prototyping and checking of hypothesised proof plans. Furthermore, in its 'disinterested party' role, the computer acts as a sceptical colleague, providing a second opinion on the merits of hypothesised proof plans that can serve as a source of inspiration. Unexpected positive and negative results can cause one to revise ones current preconceptions.

Computer checking is particularly valuable to check the criteria of generality, prescriptiveness and correctness. It is tedious to apply a tactic to a large collection of conjectures to demonstrate generality. Hand checking for prescriptiveness and correctness are notoriously error-prone, since humans tend to overlook the branch points in the search that do not lead to proofs, especially when these start with obviously losing moves, and they tend to leap prematurely down obviously winning branches, overlooking mismatches between the proof plan and the current situation. Maybe they are being influenced by their own internal proof plans. Thus computer checking of proof plans can produce unexpected results which serve as a source of inspiration for a proof plan's further development.

Another role for the computer is to show the technological value of our science of reasoning. Since proof plans suggest a procedure for the discovery of proofs, the computer can be used to automate these proofs, with consequent applications to the automation of formal methods, knowledge-based systems, *etc.* Automatic reasoning systems which use proof plans to guide the search for a proof can use the preconditions and effects of the methods to explain the high-level structure of those proofs. This might be useful in presenting choices and/or justifying conclusions in an interactive reasoning system. It might also be used to help teach problem solving in a computer aided instruction system.

11 The Relation to Automatic Theorem Proving

Although our science of reasoning might find application in the building of performance automatic theorem provers, the two activities are not co-extensive. They differ both in their motivation and their methodology.

I take the conventional motivation of automatic theorem proving to be the building of theorem provers which are empirically successful, without any necessity to understand why. The methodology is implied by this motivation. The theorem prover is applied to a random selection of theorems. Unsuccessful search spaces are studied in a shallow way and crude heuristics are added which will prune losing branches and prefer winning ones. This process is repeated until the law of diminishing returns makes further repetitions not worth pursuing. The result is fast progress in the short term, but eventual deadlock as different proofs pull the heuristics in different directions. This description is something of a caricature. No ATP researchers embody it in its pure form, but aspects of it can be found in the motivation and methodology of all of us, to a greater or lesser extent.

Automatic theorem provers based on proof plans make slower initial progress. Initial proof plans have poor generality, and so few theorems can be proved. The motivation of understanding proofs mitigates against crude, general heuristics with low prescriptiveness and no expectancy. The 'accidental' proof of a theorem is interpreted as a fault caused by low prescriptiveness, rather then a lucky break. However, there is no eventual deadlock to block the indefinite improvement of the theorem prover's performance. If two or more proof plans fit a theorem then either they represent legitimate alternatives both of which deserve attempting or they point to a lack of prescriptiveness in the preconditions which further proof analysis should correct.

Some laws of diminishing returns will become active as the number of tactics increases, but these are potentially soluble by adjusting the balance between the different criteria. This may entail deeper analysis. For instance, it may become prohibitively expensive to choose between the many potential proof plans. This can be cured by trading the prescriptiveness of the tactics' preconditions for an improvement of their generality and simplicity, and hence reducing the number of tactics. On the other hand, the search engendered by the tactics may become prohibitively expensive, in which case we will want to increase the prescriptiveness of the tactics and their preconditions. The undecidability of most interesting mathematical theories means that we will always eventually encounter such barriers, but no one barrier should prove inherently insurmountable. A deeper analysis of the problem proofs, leading to an extension of our meta-language, should, in principle, enable us to define a revised collection of proof plans with an acceptable balance between the criteria.

Thus, we expect a science of reasoning will help us build better automatic theorem proving programs in the long term, although probably not in the short term.

Conventional theorem proving research is a valuable input to our science of reasoning. Its analyses and heuristics provide a starting point for the construction of proof plans. However, these heuristics are usually too crude to provide the deep understanding that our motivation demands. They must be developed using the criteria of §8.

12 Conclusion

In this paper we have proposed a methodology for reaching a multi-level understanding of mathematical proofs as part of a science of reasoning. The theories of this science consist of a collection of general-purpose proof plans, and the association of special-purpose proof plans with particular proofs. Each proof plan consists of a tactic and a method which partially specifies it. Special-purpose proof plans can be constructed by a process of plan formation which entails reasoning with the methods of the general-purpose proof plans. Ideas for new proof plans can be found by analysing mathematical proofs using our intuitions about their structure and, possibly, psychological experiments on third party mathematicians. Initial proof plans are then designed which capture this structure. These initial proof plans are then refined to improve their expectancy, generality, prescriptiveness, simplicity, efficiency and parsimony while retaining their correctness. Scientific judgement is used to find a balance between these sometimes opposing criteria. Computers can be used as a workhorse, as a disinterested party to check the criteria and as a source of inspiration.

The design of general-purpose proof plans and their association with particular proofs is an activity of scientific theory formation that can be judged by normal scientific criteria. It requires deep analysis of mathematical proofs, rigour in the design of tactics and their methods, and judgement in the selection of those general-purpose proof plans with real staying power. Our science of reasoning is normative, empirical and reflective. In these respects it resembles other human sciences like Linguistics and Logic. Indeed it includes Logic as a sub-science.

Personal Note

For many years I have regarded myself as a researcher in automatic theorem proving. However, by analysing the methodology I have pursued in practice, I now realise that my real motivation is the building of a science of reasoning in the form outlined above. Now that I have identified, explicitly, the science I have been implicitly engaged in for the last fifteen years, I intend to pursue it with renewed vigour. I invite you to join me.

References

[Aubin 75] R. Aubin. Some generalization heuristics in proofs by induction. In G. Huet and G. Kahn, editors, *Actes du Colloque Construction: Amélioration et vérification de Programmes*. Institut de recherche d'informatique et d'automatique, 1975.

[Bledsoe *et al.* 72] W. W. Bledsoe, R. S. Boyer, and W. H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence*, 3:27–60, 1972.

[Boyer & Moore 79] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979. ACM monograph series.

[Bundy & Welham 81] A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981. Also available from Edinburgh as DAI Research Paper 121.

[Bundy 88] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.

[Bundy *et al.* 88a] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. Research Paper 413, Dept. of Artificial Intelligence, University of Edinburgh, 1988. Appeared in Journal of Automated Reasoning, 7, 1991.

[Bundy *et al.* 88b] A. Bundy, F. van Harmelen, J. Hesketh, A. Smaill, and A. Stevens. A rational reconstruction and extension of recursion analysis. Research Paper 419, Dept. of Artificial Intelligence, University of Edinburgh, 1988. Also in the proceedings of IJCAI-89.

[Burstall & Darlington 77] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1):44–67, 1977.

[Constable *et al.* 86] R. L. Constable, S. F. Allen, H. M. Bromley, *et al. Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, 1986.

[Funt 73] B. V. Funt. A procedural approach to constructions in Euclidean geometry. Unpublished M.Sc. thesis, University of British Columbia, October 1973.

[Gordon *et al.* 79] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[Komorowski 82] H. J. Komorowski. Partial evaluation as a means for inferencing data structures in an applicative language: A theory and implementation in the case of PROLOG. In *Proceedings of the ninth conference on the Principles of Programming Languages (POPL)*, pages 225–267, Albuquerque, New Mexico, 1982. ACM.

[Mitchell *et al.* 86] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986. Also available as Tech. Report ML-TR-2, SUNJ Rutgers, 1985.

[Paulson 88] L.C. Paulson. Experience with Isabelle: A generic theorem prover. In *COLOG 88*. Institute of Cybernetics of the Estonian SSR, 1988.

[Pólya 65] G. Pólya. *Mathematical Discovery.* John Wiley & Sons, Inc, 1965. Two volumes.

[Robinson 65] J. A. Robinson. A machine oriented logic based on the resolution principle. *J Assoc. Comput. Mach.*, 12:23–41, 1965.

[Wos & McCune 88] L. Wos and W. McCune. Searching for fixed point combinators by using automated theorem proving: a preliminary report. Technical Report ANL-88-10, Argonne National Laboratory, 1988.