Technical Report

UCAM-CL-TR-327
ISSN 1476-2986

Number 327

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

A new application for explanation-based generalisation within automated deduction

Siani L. Baker

February 1994

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

ISSN 1476-2986

A New Application for Explanation-Based Generalisation within Automated Deduction[*]

Siani L. Baker
Computer Laboratory, University of Cambridge

Abstract

Generalisation is currently a major theorem-proving problem. This paper proposes a new method of generalisation, involving the use of explanation-based generalisation within a new domain, which may succeed when other methods fail. The method has been implemented for simple arithmetical examples.

Key words. Generalisation, proof transformation, automated theorem proving.

[*]Research funded by the SERC (grant RF/1389).

Contents

1 Introduction

1.1 Generalisation

Generalisation is a powerful tool in automated theorem proving with a variety of rôles, such as enabling proofs, defining new concepts, turning proofs for a specific example into ones valid for a range of examples and producing clearer proofs. Van der Waerden's account of how the proof of Baudet's conjecture was found [25] illustrates how generalisation lies at the heart of mathematical discovery, and how generalised theorems may be easier to prove than the original goal (because the induction hypothesis is also made stronger when the goal is strengthened by generalisation). If induction is blocked for an expression[1], generalisation may be used as a step to convert this expression into a new, more general, expression which may be proved by induction. In order to verify a goal Φ, one may instead prove its generalisation Ψ. Indeed, it might actually be *necessary* to adopt this approach in cases where Φ is a theorem of a system $\mathcal{S}$ but it is not the case that Φ may be proven by induction within $\mathcal{S}$ (although Ψ may be found such that Ψ is provable by induction within $\mathcal{S}$, and such that Ψ is a generalisation of Φ). Heuristics are needed for the suggestion of Ψ since there is no appropriate algorithm for finding suitable generalisations.[2] A suitable suggested generalisation must be just general enough to provide a proof by induction whilst not being too general, for not only may generalisations Ψ be computed such that it is not the case that Ψ is provable by induction, but overgeneralisations may also be computed (where Ψ is not a theorem of $\mathcal{S}$).

Although generalisation is an important problem in theorem-proving, it has by no means been solved. It is important and still being investigated for reasons which relate to cut elimination and the lack of heuristics for providing cut formulae. A cut elimination theorem for a system states that every proof in that system may be replaced by one which does not involve use of the cut rule[3]. Uniform proof search methods can be used for logical systems, in sequent calculus form, where the cut rule is not used. In general, cut elimination holds for arithmetical systems with the ω-rule, but not for systems with ordinary induction. Hence in the latter, there is the problem of generalisation, since arbitrary formulae can be cut in. This makes automatic theorem-proving very difficult, especially as there is no easy or fail-safe method of generating the required cut formula.

When discussing generalisation I have so far referred to goal generalisation, which is the proof step described above which allows the postulation of a new theorem as a substitute for the current goal, from which the latter follows easily (for example,

[1]Induction is said to blocked if, after all available symbolic evaluation has been carried out, the induction conclusion is still not an instance of the induction hypothesis, and hence remains unprovable.

[2]Of course, an algorithm could be used which tried every possible option, but the search required might be infeasibly large, and the conclusion that there was no suitable generalisation would not be allowed: this trivial approach is not what is required.

[3]See [22], for example.

backward application of the ∀-elim rule of a natural deduction calculus). Yet generalisation may be carried out on proofs as well: it has a different emphasis, namely on providing a more general proof, and although goal generalisation may involve generalisation of the proof of a theorem in order to generalise the theorem itself, this need not be the case.

1.2 The Proposed Generalisation Strategy

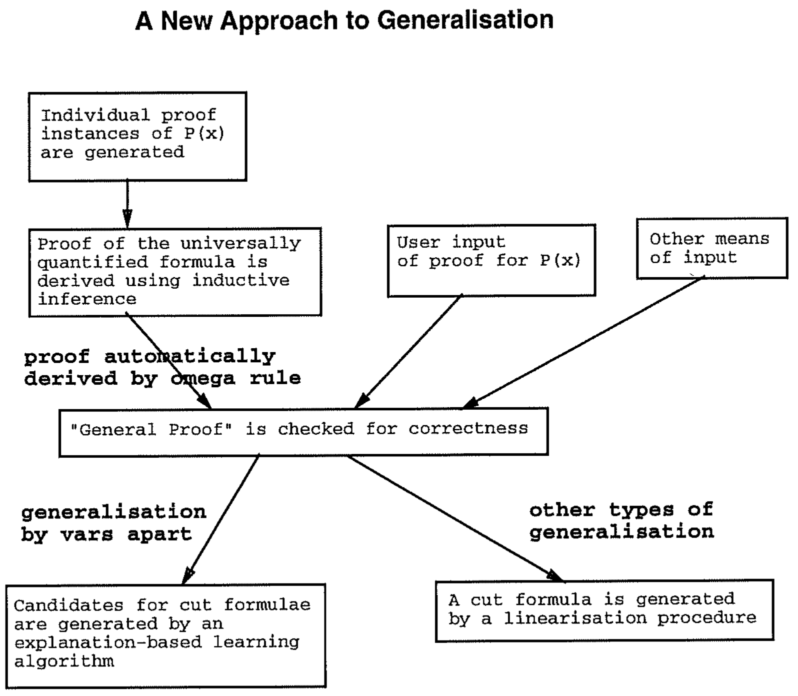
A New Approach to Generalisation
Individual proof instances of P(x) are generated
Proof of the universally quantified formula is derived using inductive inference
User input of proof for P(x)
Other means of input
proof automatically derived by omega rule
"General Proof" is checked for correctness
generalisation by vars apart
other types of generalisation
Candidates for cut formulae are generated by an explanation-based learning algorithm
A cut formula is generated by a linearisation procedure

Figure 1: Generalisation Strategy

Generalisation of proofs has the advantage that more information may be available than for goal generalisation. The method proposed in this paper exploits this fact by enabling goal generalisation by means of carrying out explanation-based generalisation on proofs. The problem of generalisation is tackled by use of an alternative (stronger) representation of arithmetic, in which proofs may be more easily generated. The "guiding proofs" in this stronger system may succeed in producing proofs in the original system when other methods fail (cf. Table 2). This method has been automated for simple arithmetical examples and results in the suggestion of an appropriate cut formula. More specifically, in order to carry out generalisations of the form $\forall x A(x)$ to $\forall x Q(x)$, individual proof instances of $A(x)$ are generated, and then generalised to a proof of the arbitrary case $A(r)$ using some inductive inference process (which presents algorithms to obtain a general pattern from individual instances). The latter proof can be generalised via explanation-based generalisation to the most general proof using the same rule set. The new proof can be shown to be of a linear form. Hence the formula for which this is a proof provides a suitable cut formula for $A(x)$, since it satisfies the properties of being a more general form of which A is a specific case, and such that induction may be performed upon it. Figure 1 represents the overall strategy for generalisation: a "general proof" (of $A(r)$) is provided by some means (one possible option being an automatic derivation using the ω-rule), and a suitable cut formula is suggested by inspection of this proof.

The following section presents ω-proofs, from which it is possible to read off appropriate cut formulae, and the semi-formal system of arithmetic in which they are derived. Section 3 discusses how the explanation-based generalisation method may be applied to these proofs in order to suggest cut formulae. Section 4 discusses how conventional inductive proofs are related to ω-proofs, and in particular how the explanation-based generalisation method linearises ω-proofs, and hence provides a cut formula which is provable by induction. Finally, conclusions are given.

2 The Constructive Omega Rule

In order to describe the generalisation method proposed, it is first necessary to provide a description of the 'stronger' system mentioned above. A suitable rule other than induction which might be added to Peano's axioms to form a system formalising arithmetic is the ω-rule:

$$\frac{A(0), A(\underline{1}) \dots A(\underline{n}) \dots}{\forall x A(x)}$$

where $\underline{n}$ is a formal numeral, which for natural number n consists in the n-fold iteration of the successor function applied to zero, and A is formulated within the language of arithmetic. This rule is not derivable in Peano Arithmetic (PA)[4], since for example, for the Gödel formula $G(x)$, for each natural number n, $PA \vdash G(\underline{n})$ but it is not true that $PA \vdash \forall x G(x)$. This rule together with Peano's axioms gives a complete theory — the usual incompleteness results do not apply since this is not a formal system in the usual sense.

However, this is not a good candidate for implementation since there are an infinite number of premises. It would be desirable to restrict the ω-rule so that the infinite proofs considered possess some important properties of finite proofs. One suitable option is to use a **constructive** ω**-rule**. The ω-rule is said to be constructive if there is a recursive function f such that for every n, $f(n)$ is a Gödel number of $P(n)$, where $P(n)$ is defined for every natural number n and is a proof of $A(\underline{n})$ [24]. This is equivalent to the requirement that there is a uniform, computable procedure describing $P(n)$, or alternatively that the proofs are recursive (in the sense that both the proof-tree and the function describing the use of the different rules must be recursive) [27], which is the basis taken for implementation (as opposed to a Gödel numbering approach). The sequent calculus enriched with the constructive ω-rule (let us call it $PA_{c\omega}$) has cut elimination, and is complete [23]. Moreover, since the ω-rule implies the induction rule, $PA_{c\omega} + IND$ is a conservative extension of $PA_{c\omega}$. There are many versions of a restricted ω-rule; this one has been chosen

[4]See for example [22] for a formalisation.

because it is suitable for automation. Note that in particular this differs from the form of the ω-rule (involving the notion of provability) considered by Rosser [20] and subsequently Feferman [9]. Implementation of a proof environment with the constructive ω-rule is described in [2]; this provides a basis for the implementation of the generalisation method described in this paper.

In the context of theorem proving, the presence of cut elimination for these systems means that generalisation steps are not required. In the implementation, although completeness is not claimed, some proofs that normally require generalisation can be generated more easily in $PA_{c\omega}$ than PA.

2.1 Automation of the ω-rule

The constructive ω-rule may be used to enable automated proof of formulae, such as $\forall x\ (x + x) + x = x + (x + x)$, which cannot be proved in the normal axiomatisation of arithmetic without recourse to the cut rule. In these cases the correct proof could be extremely difficult to find automatically. However, it is possible to prove this equation using the ω-rule since the proofs of the instances $(0 + 0) + 0 = 0 + (0 + 0), (1+1)+1 = 1+(1+1), \ldots$ are easily found, and the general pattern determined by inductive inference. Automated proof in such a system might be seen as a goal in itself, but the concern of this paper is rather how it is possible to use this system as a guide to the provision of difficult proofs in more conventional systems.

One way in which the constructive ω-rule may be put into effect is to require that there is an enumeration of the derivations which prove the premises — for example one could code proofs by numbers, by means of a primitive recursive function which generates them. But I have not used such a traditional representation; it was sufficient for my purposes to provide (for the nth case) a description for the ω-proof of A, namely $P(n)$, in a constructive way (in this case a recursive way), which captures the notion that each $P(n)$ is being proved in a uniform way (from parameter n). This is done by manipulating $A(\underline{n})$, where $\forall x A(x)$ is the sequent to be proved, and using recursively defined function definitions of PA as rewrite rules, with the aim of reducing both sides of the equation to the same formula. The recursive function sought is described by the sequence of rule applications, parametrised over n. In practice, the first few proofs will be special cases, and it is rather the correspondence between the proofs of $P(99)$, say, and $P(100)$, which should be captured.

The ω-proof representation represents $P(n)$, the proof of the nth numerator of the constructive ω-rule, in terms of rewrite rules applied $f(n)$ or a constant number of times to formulae (dependent upon the parameter n). (For more technical details regarding the representation of ω-proofs, see [2] and [3].) As an example, the implementational representation of the ω-proof for $\forall x\ (x + x) + x = x + (x + x)$ takes the form given in Figure 2 (although it may be represented in a variety of ways) presuming that, within the particular formalisation of arithmetic chosen, one is given the axioms of addition of Figure 2.

By $s^n(0)$ is meant the numeral $\underline{n}$, ie. the term formed by applying the successor function n times to 0. The next stages use the axioms as rewrite rules from left to right, and substitution in the ω-proof, under the appropriate instantiation of

Axioms

$$\begin{aligned} 0+y &= y & (1)\\ s(x)+y &= s(x+y) & (2)\end{aligned}$$

Proof

$$\begin{array}{ll}
 & (\underline{n}+\underline{n})+\underline{n}=\underline{n}+(\underline{n}+\underline{n})\\
\underline{n}\equiv s^n(0) & (s^n(0)+s^n(0))+s^n(0)=s^n(0)+(s^n(0)+s^n(0))\\
USE\ (2)\ n\ TIMES\ ON\ LEFT\ ([1,1]) & s^n(0+s^n(0))+s^n(0)=s^n(0)+(s^n(0)+s^n(0))\\
USE\ (1)\ ON\ LEFT\ ([2,2,1,1]) & s^n(s^n(0))+s^n(0)=s^n(0)+(s^n(0)+s^n(0))\\
USE\ (2)\ n\ TIMES\ ON\ RIGHT\ ([2]) & s^n(s^n(0))+s^n(0)=s^n(0+(s^n(0)+s^n(0)))\\
USE\ (1)\ ON\ RIGHT\ ([2,2,2]) & s^n(s^n(0))+s^n(0)=s^n(s^n(0)+s^n(0))\\
USE\ (2)\ n\ TIMES\ ON\ LEFT\ ([1]) & s^n(\underbrace{s^n(0)}_{\lambda}+\underbrace{s^n(0)}_{\lambda})=s^n(\underbrace{s^n(0)}_{\lambda}+\underbrace{s^n(0)}_{\lambda})\\
 & EQUALITY
\end{array}$$

Figure 2: An ω-Proof of $\forall x\ (x+x)+x=x+(x+x)$

variables, with the aim of reducing both sides of the equation to the same formula. The subpositions to which the rewrite rules are applied are given in parentheses, according to the *exp_at* notation of Clam [26]. The ω-proof represents, and highlights, blocks of rewrite rules which are being applied. Meta-induction may be used (on the first argument) to prove the more general rewrite rules from one block to the next: for example, $\forall n\ s^n(x)+y=s^n(x+y)$ corresponds to n applications of axiom (2) above.

The processes of generation of a (recursive) ω-proof from individual proof instances, and the (metalevel) checking that this is indeed the correct proof have been automated (see [4]). Any appropriate inductive inference algorithm[5], such as Plotkin's least general generalisation [19], or that of Rouveirol [21], could be used to guess the ω-proof from the individual proof instances.

Note that such inductive inference algorithms for generating generalisations produce a proof for an arbitrary instance: it is suggested below how explanation-based generalisation can be applied using this information to find the proper induction formulae for inductive theorem provers. The problems involved in the inductive inference process differ from those involved in (goal) generalisation because inductive inference generalises a proof of $A(k)$ where k is some number to a proof of $A(n)$, whereas generalisation involves finding a formula C such that $C\vdash A$ and in addition such that C may be proven by induction. Inductive inference may be carried out by a relatively simple algorithm (such as updating a guess). However, there is no such known algorithm for goal generalisation.

In the implementation, the theorem under consideration is proved for a few cases,

[5]Explanation-based learning permits learning from a single example, whereas inductive learning usually requires many examples to learn a concept.

and then learning induction is used to guess the ω-proof from these cases; next it is verified that the guessed ω-proof is correct.[6] As an alternative, the user may bypass this whole stage by specifying the ω-proof directly. Further details of the algorithms and representations used, together with the correspondence between the adopted implementational approach and the formal theory of the system are described in [3].

The next section discusses how ω-proofs may be generalised.

3 Explanation-Based Generalisation Applied to ω-Proofs

There is a class of proofs which are provable in PA only using the cut rule but which are provable in $PA_{c\omega}$ [3, 14]. I shall consider whether the proof in $PA_{c\omega}$ suggests a proof in PA, and in particular, what the cut formula would be in a proof in Peano Arithmetic.

3.1 The Problem of Generalisation

To illustrate the general principle, consider the simple example for $\Psi \equiv \forall x\ (x+x)+x = x+(x+x)$. The problem is to find Φ such that Φ should be a more general version of the goal Ψ, in order to prove $\Phi \vdash \Psi$, but on the other hand it should be suitable to give an inductive proof of Φ.

$$\frac{\Phi \vdash \forall x\ (x+x)+x = x+(x+x) \qquad \vdash \Phi}{\vdash \forall x\ (x+x)+x = x+(x+x)}\,CUT$$

Ordinary induction does not work on Ψ, primarily because the second, third and sixth terms in the step case may not be broken down by the rewrite rules corresponding to (1) and (2) above, and hence fertilisation (substitution using the induction hypothesis) cannot occur. Hence it is necessary to use the cut rule.

3.2 An Initial Solution

In order to suggest a cut formula from an ω-proof, one method already proposed is to see what remains unaltered in the nth case proof, and then write out the original formula, but with the corresponding term re-named [5]. So, for the example in Figure 1, the variable corresponding to λ would be rewritten as y. In this case, this would give

$$\Phi \equiv \forall x \forall y\ (x+y)+y = x+(y+y).$$

Φ could then be proved by induction on x. Note that what is meant by 'unaltered' is defined by what is unaffected in structure by the rewrite rules. This procedure has been automated (all that is required is detection of the unaltered terms), and

[6] Verifying that the guessed ω-proof works for all n involves mathematical induction at the meta-level; however, the induction required for this differs from that required to prove the original theorem directly, and is usually simpler, so there is no circularity.

so the cut formula may be produced automatically. This method of generalisation will allow the proof of some theorems which pose a problem for other methods, such as $x \neq 0 \rightarrow p(x) + s(s(x)) = s(x) + x$, where p is the predecessor function (detailed comparisons of this 'unaltered term' method with other generalisation methods with regard to this example are given in [5]).

3.3 Generalisation using Explanation-Based Generalisation

A new development of this solution, which produces a more general generalisation, is to look at the rules of the ω-proof, and work out what the most general statement could be which was proved using these rules. This process has been applied in various other domains (see for example [7]), and is the approach of explanation-based generalisation (denoted 'EBG' as an abbreviation). EBG is a technique for formulating general concepts on the basis of specific training examples, first described in [16]. The process works by generalising a particular solution to the most general possible solution which uses the rules of the original solution. It does this by applying these rules, making no assumptions about the form of the generalised solution, and using unification to fill in this form.

Original Problem	**Rule Used**	**Generalisation**
$\int 5.x^2 dx$		$\int S.x^{K \neq -1} dx$
$\Downarrow$	$Rule1 : \int R.F(x)dx \Rightarrow R.\int F(x)dx$	$\Uparrow$
$5.\int x^2 dx$		$S.\int x^{K \neq -1} dx$
$\Downarrow$	$Rule2 : \int x^{R \neq -1} dx \Rightarrow \frac{x^{R+1}}{R+1}$	$\Uparrow$
$5.\frac{x^3}{3}$		$S.\int G(x)dx$

Figure 3: An Example of Explanation-Based Generalisation in LEX2

Figure 3 illustrates the EBG method, where the domain comprises (partial) integration proofs [17], and constraint back-propagation is used to enable explanation-based generalisation. With reference to this figure, the most general form of the expression for which the rules would be applicable may be calculated as follows: The form of an expression to which *Rule*1 is applicable must be $\int S.G(x)dx$, and afterwards $S.\int G(x)dx$, for any rational number S and first-order function G. For *Rule*2 to be applicable, $G(x)$ must be $x^{K \neq -1}$, for any rational number K. Any such information derived will then be filtered back to the original expression, to give a most general generalisation (using the rules used on the original problem) of $\int S.x^{K \neq -1} dx$.

Figure 4 shows how the EBG method applied to an individual example $A(k)$ fails to provide a generalisation appropriate for direct use as a cut formula for $A(x)$. In particular, the individual proof of $P(k)$ $k \in nat$ will *at best* generalise to $Q(k, y)$ where Q is a specialised form of a suitable generalisation (for the proof of $A(k)$ may be specialised with respect to k, and not general enough to capture for instance the correspondence between the proof for $P(k)$ and $P(k+1)$. The

Original Problem	**Rule Used**	**Generalisation**
$(\underline{5}+\underline{5})+\underline{5}=\underline{5}+(\underline{5}+\underline{5})$		$(\underline{5}+Y)+K=\underline{5}+(Y+K)$
$\Downarrow$	$Rule1: s(X)+Y \Rightarrow s(X+Y)$	$\Uparrow$
$\Downarrow$	$Rule2: 0+Y \Rightarrow Y$	$\Uparrow$
$EQUALITY$		$EQUALITY$

Figure 4: EBG on Proof of an Individual Example

rules of ω-proof	generalised ω	proof	instantiations
(2) *n times at* [1, 1]	$fn0([s^n(X)+Y\|K])$	$=W$	*original*
(1) *once at* [2, 2, 1, 1]	$fn0([s^n(X+Y)\|K])$	$=W$	
(2) *n times at* [2]	$fn0([s^n(Y)\|K])$	$=W$	$X=0$
(1) *once at* [2, 2, 2]	$fn0([s^n(Y)\|K])$	$=s^n(P+Q)$	$W=s^n(P)+Q$
(2) *n times at* [1]	$s^n(Y+K)$	$=s^n(Q)$	$P=0,\ fn0=+$
$EQUALITY$	$s^n(Y+K)$	$=s^n(Y+K)$	$Q=Y+K$

Figure 5: Illustration of Explanation-Based Generalisation on Rules of ω-Proof

final result of EBG does not lead to the required generalisation: it is necessary to generalise this again. One needs also to introduce a correctness check for the suggested generalisation. In addition, the required generalised proof will involve rewrite rules applied a parametrised number of times (because of the structure of the database). However, by providing a *correct* initial proof using parametrised rules, it is possible to carry out EBG directly in order to produce a suitable cut formula (cf. Figure 5).

The EBG method is applied in this instance to a new domain, namely that of ω-proofs. As an illustration of the method, let us apply explanation-based generalisation to Figure 1, to give the process shown in Figure 5. The right hand column is the instantiations of variables, which are finally to be filtered back up into the original expression. Essentially what is happening is that each application of the rewrite rules of the original ω-proof is matched with the latest line of the new ω-proof to see the necessary form of a generalised ω-proof. If (2) is applied m times, this will match with the form

$$s^m(X)+Y \Rightarrow s^m(X+Y)$$

Nothing more is supposed about the original form of the ω-proof than that it is of the form $U=W$. The rule application blocks on the left hand side of this figure are identical with those of the ω-proof given in Figure 1. The procedure is to form the most general ω-proof which could use those same rules to achieve equality. Hence, these same rewrite rules are applied at the specified subpositions to give a new ω-proof. In so doing the structure of U and W is revealed. For instance, the fact that rule (2) may be applied n times at subposition [1,1] of $U=W$ reveals that U must be of the form $fn0([s^n(X)+Y|K])$ (which represents some functor $fn0$ of as yet

$$\begin{aligned}
\forall x\; x + s(x) &= s(x + x) && (3)\\
\forall x\; (x + x) + x &= x + (x + x) && (4)\\
\forall x\; x + s(x) &= s(x) + x && (5)\\
\forall x\; x.(x + x) &= x.x + x.x && (6)\\
\forall x\; (2 + x) + x &= 2 + (x + x) && (7)\\
\forall x\, \forall y\; (x + y) + x &= x + (y + x) && (8)\\
\forall x\; x \neq 0 \rightarrow p(x) + s(s(x)) &= s(x) + x && (9)\\
\forall x\; even(x + x) &= true && (10)
\end{aligned}$$

Note that induction is blocked for the above expressions, but they may all be proved by the EBG method proposed and a correct cut formula produced as appropriate.

Table 1: Some Examples of Theorems Proved

unknown arity with initial argument $s^n(X)+Y$ and additional arguments K) before the rule application, and of the form $fn0([s^n(X+Y)|K])$ afterwards. This process is repeated until all the given rules are exhausted. Finally, the left-hand side and the right-hand side of the ω proof are unified (since the original proof resulted in equality). Throughout this process, information will have been obtained regarding the structure of some of the postulated variables in this new ω proof, such as that presented in the final column of Figure 5.

Feeding such variable instantiation information back to the original expression $U = W$ shows that it must be of the form:

$$(\underline{n} + Y) + K = \underline{n} + (Y + K)$$

This gives the most general generalisation as being

$$\forall x\, \forall y\, \forall z\; (x + y) + z = x + (y + z)$$

By means of this method, the generalisations are found by recognising patterns, and a uniform approach for generalisation is provided.

Although the heuristic of replacing unaltered terms is suitable for implementation (and was successfully implemented), the method of explanation-based generalisation extends this idea to provide a uniform algorithm based on the underlying structure of the proof. The implementation of EBG, in which the previous "ω-rule" environment was extended to include generalisation tactics, follows the unification process described above, and thus subsumes the implementation of the heuristic method. [4] provides full details of the implementational environment, which will automatically suggest cut formulae for examples such as (3)–(9) of Table 1.[7] Most of these

[7]Definitions of the predicates involved may be found in [3].

examples involve generalisation of variables apart, or else generalisation of common subexpressions. In these cases both the heuristic of replacing unaltered variables and the explanation-based generalisation method work fairly straightforwardly. Although the examples listed in the table are of a similar simple form, these methods may also be applied to complicated examples containing nested quantifiers, etc., for the ω-rule applies to arbitrary sequents. Example (8) provides an instance of nested use of the ω-rule, which carries through directly. For example (10), the cut formula of $even(2.x)$ could possibly be extracted by a user from the form of the ω-proof (see [5]), which is an improvement over other generalisation methods. However, in some cases where an ω-proof may be provided, it is not clear what the cut formula might be.

3.4 Comparison with Related Work

These methods involving manipulation of the ω-proof should be compared with current generalisation methods. Of these, perhaps the most famous is that implemented by Boyer and Moore in their theorem-prover NQTHM [6]. The main heuristic for (goal) generalisation is that identical terms occurring on both the left and right side of the equation are picked for rewriting as a new variable (with certain restrictions). This may be a quick method if it happens to work, but may also entail the proofs of many lemmas, which might need to be stored in advance in anticipation of such an event in order to be more efficient. The problems inherent in Boyer and Moore's approach have led Raymond Aubin to extend their work in this field [1]. Aubin's method is to "guess" a generalisation by generalising occurrences in the definitional argument position, and then to work through a number of individual cases to see if the guess seems to work. If it does work, he will look for a proof. If it does not, then he will "guess" a different generalisation. However, Aubin's solution does not work in all cases. In particular, if a constructor such as a successor function appears in an original goal, together with individual variables, Aubin's method may result in over-generalisation or indeed no solution at all. These methods are used as the basis for generalisation in many different theorem-proving systems. Related work in proof generalisation has been carried out by Masami Hagiya, who has considered generalization of proofs in type theories. Hagiya approaches the problem of proof generalisation by synthesising a general proof from a concrete example proof by higher-order unification in a type theory with a recursion operator [12]. Rather than the ω-rule, he uses ordinary recursion terms for representing inductive proofs. In order to make recursion terms more expressive, he has extended the calculus with implicit arithmetical inferences [11]. However, the degree of automation provided is not very high, and he has not addressed the issue of ensuring that the proofs from which one generalises are in the form required to produce a suitable result.

The generalisation method proposed in this paper provides a uniform approach and does not have to check extra criteria, nor work through individual examples. Moreover, it is not possible to overgeneralise to a non-theorem (the method is sound but not complete — it does not always provide a solution, nor necessarily the best solution possible). Table 2 provides a comparison between cut formulae suggested

Methods	Selection of Types of Example	
	$x.(x+x) = x.x + x.x$	$(x+x)+x = x+(x+x)$
Boyer & Moore	*fail*	$y+x = x+y$
Aubin	$x.(y+y) = x.y + x.y$ 1.	$(x+y)+y = x+(y+y)$
EBG Method	$x.(y+z) = x.y + x.z$	$(x+y)+z = x+(y+z)$
	$x+s(x) = s(x)+x$	$even(x+x)$
Boyer & Moore	$x+y = y+x$	*fail*
Aubin	*fail*	*fail*
EBG Method	$x+s(y) = s(x)+y$	$even(2.x)$ 2.
	$x \neq 0 \rightarrow p(x)+s(s(y)) = s(x)+y$	$x.(y.0) = y.(y.0)$
Boyer & Moore	*fail*	*fail*
Aubin	*fail*	*fail*
EBG Method	$x \neq 0 \rightarrow p(x)+s(s(y)) = s(x)+y$	ω*-proof only* 3.

1. Solution achieved by testing various possibilities after initial failure of method.
2. Generalisation algorithm fails, but this cut formula is 'suggested' from the ω-proof.
3. Generalisation algorithm fails, but user might be able to suggest a generalisation from the ω-proof.

Table 2: Cut Formulae Suggested Using Different Generalisation Methods

by the various methods, and demonstrates that the EBG guiding method works much better on some examples.

By the EBG method, the most general generalisation of an expression is achieved, in a uniform manner. This is a new and desirable achievement: Hesketh writes [13, P271] that for her generalisation method "The generalisation from $\forall x.x + (x+x) = (x+x)+x$ will be found as $\forall x \forall y.x + (y+y) = (x+y)+y$ not $\forall x \forall y \forall z.x + (y+z) = (x+y)+z$ but I know of no theorem prover that can find this last generalisation in a principled way, ie. other than with just trial and error." Note also how if an ω-proof is provided, even without a generalisation being suggested, something has still been achieved, in the sense that a pattern might still emerge for the user. Thus the method may still be useful within a co-operative environment if it breaks down (cf. note 3., Table 2). This contrasts with alternative methods of generalisation, which do not provide much information if they fail. Moreover, because the suggested method explicitly exploits general patterns, it has a higher-level structure and thus greater potential for extension than other more special-purpose approaches.

The EBG method proposed in this section will succeed in the sense that there does exist some ω-proof such that a correct cut formula could be found by EBG (so long as inductive proof by generalisation apart, that is, generalisation by means of renaming some occurrences of the same variable in an expression, is possible). However, it will not necessarily work if generalisation apart is not appropriate for the example under consideration.

Note that it is possible to carry out explanation-based generalisation on proofs in other systems such as Peano arithmetic, in order to give a more general expression. However, if inductive proof is not possible in these systems (because induction is blocked) and hence application of the cut rule is needed in order to obtain a proof, then such a method will not work. However, it is possible that an ω-proof may be returned, and in this case explanation-based generalisation may be applied to the latter in order to obtain a solution.

The explanation-based generalisation method proposed in this paper can be used in a more general context for lemma generation, regardless of the way an ω-proof was obtained, so long as one can represent in the particular system of interest the notions of nth-successor, parametrised applications of rewrite rules and also the correctness check. Hence, although the learning device for cut formulae is not reliant upon the given proof system involving use of the ω-rule, in practice suitable proof environments such as HOL [10] would require extension, and moreover the user would have to input the proof directly unless some other method of generation could be provided. Hence, if the ω-proof were represented directly in higher-order logic, one would still be faced with the problems of application of rewrite rules a parametrised number of times, and a method of provision of the ω-proof (solved by inductive inference in the case of $PA_{c\omega}$).

4 Linearisation

This section briefly explains what linearisation is, and then discusses how conventional inductive proofs are related to ω-proofs in the context of EBG. It shall be shown that EBG linearises ω-proofs (and therefore, the initial formula of the generalised ω-proof may be proven by induction). When the EBG method (relying on generalisation of variables apart) is not appropriate, there are other ways of linearising ω-proofs.

The general form of a linear proof involving a single use of the constructive ω-rule is given in Figure 6,[8] with the proviso that there may be additional leaves if the $P(i)$ consist of branching proof rules. Any proof which is of the form of Figure 6 (with additional leaves in the case of inductive proofs with branching in the step proof) shall be called *linear*, with the constraints that $A(\underline{k})$ is reduced to $A(\underline{k-1})$ in a uniform way for each k, and that the proof is cut-free. In the case of the constructive ω-rule operating on $\forall x\ A(x)$, the $A(i)$ are uniformly generated, and there will be a relationship between their proofs — otherwise, this may not be the case. An ω-proof is a parametrised version of an arbitrary subtree of a proof in $PA_{c\omega}$ of a universal statement; thus, a "linear" ω-proof will correspond to the kth subtree of Figure 6.

It shall be shown that ω-proofs may be turned into a linear form, which will suggest that induction can be done on the original formula (that is to say that this formula is a suitable cut formula). Hence it is necessary to recognise Q such

[8]Use of two inductions should be compared with P. Madden's transformations which linearise proofs defined by functions with two inductive variables [15].

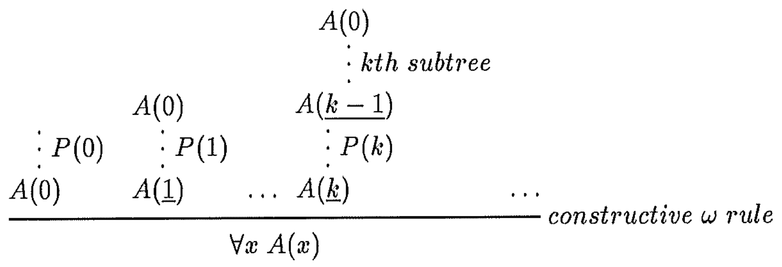

Figure 6: Form of Linear Proof in $PA_{c\omega}$

that an original ω-proof for $P(x)$ may be transformed into a "linearised" ω-proof of form $Q(s(n))$ reducing in a uniform manner (via rewrite rule block i) to $Q(n)$ to $Q(n-1)$ down to $Q(0)$, which reduces to an equality using a rewrite rule block j. The required proof of $\forall x\ P(x)$ in PA is:[9]

$$\dfrac{\forall x\ Q(x) \vdash \forall x P(x) \qquad \dfrac{\begin{array}{c} CLOSES\\ \vdots\, j\\ \vdash Q(0)\end{array} \qquad \begin{array}{c} CLOSES\\ \vdots\, i\\ Q(r)\vdash Q(s(r))\end{array}}{\vdash \forall x\ Q(x)}\, ind(x)}{\vdash \forall x\ P(x)}\, cut$$

If the ω-proof is of a linear form, it is the case that $\forall x\ P(x)$ may be proved in this manner. The fact that a proof of $\Gamma, A(\underline{k}) \vdash A(s(\underline{k}))$ exists provided there is a derivation of $\Gamma \vdash A(s(\underline{k}))$ from $\Gamma \vdash A(\underline{k})$ (in the sense of there being a natural deduction proof of the former from the latter, possibly using additional axioms) forms an analogue of the deduction theorem.[10]

Theorem 1 (Deduction Theorem for Sequents) *The sequent $\Gamma, A \vdash B$ is provable in $PA_{c\omega}$ if and only if there is a derivation in $PA_{c\omega}$ of*

$$\begin{array}{c}\Gamma\vdash A\\ \vdots\\ \Gamma\vdash B\end{array}$$

Proof 1 *$\Rightarrow$: Suppose $\Gamma, A \vdash B$ is provable in $PA_{c\omega}$. The cut rule may be used to derive $\Gamma \vdash B$ from $\Gamma \vdash A$ and $\Gamma, A \vdash B$. Hence, given $\Gamma, A \vdash B$, then there is a derivation of $\Gamma \vdash B$ from $\Gamma \vdash A$. By the cut elimination theorem, there must be a cut-free derivation (in $PA_{c\omega}$) of $\Gamma \vdash B$ from $\Gamma \vdash A$.*
$\Leftarrow$: Suppose that there is a proof in $PA_{c\omega}$ of $\Gamma \vdash B$ from $\Gamma \vdash A$. By using the same rules, there must be a proof of $\Gamma, A \vdash B$ from $\Gamma, A \vdash A$. However, $\Gamma, A \vdash A$ is an axiom. Hence, there is a proof in $PA_{c\omega}$ of $\Gamma, A \vdash B$. □

Theorem 2 (Linearisation Theorem) *Q can be proved using a single induction if and only if there is a linear ω-proof of Q*

[9]The stages i and j apply rewrite rules on a formula, rather than structural manipulations on a sequent.

[10]If $\Gamma, B \vdash_K A$, then $\Gamma \vdash_K B \to A$, for some logical system K [8, P127].

Proof 2 $\Rightarrow$: *By Theorem 1.*

$\Leftarrow$: *Given an ω-proof of the form:*

$$\begin{array}{c} \dfrac{Q(s(k))}{Q(k)}\, i(k) \\ \vdots \\ \dfrac{Q(0)}{\mathit{equality}}\, j \end{array}$$

then $Q(s(\underline{k}))$ reduces to $Q(\underline{k})$ for all numerals such that $k \leq n$, where n was arbitrary — hence k is arbitrary. The rules $i(k)$ here form a repeated rule-block (parametrised over k): this is what has been described above as a linearisable ω-proof. By the soundness of the ω-proof representation with respect to $PA_{c\omega}$ (cf. [3]), there is a proof in $PA_{c\omega}$ of

$$\begin{array}{c} \vdash Q(\underline{k}) \\ \vdots \\ \vdash Q(s(\underline{k})) \end{array}$$

and by Theorem 1, there is a proof of $Q(k) \vdash Q(s(k))$, for each numeral k. By the form of the original linear proof, each proof segment $P(k)$ is obtained by taking the same shape of proof with a numerical parameter that is replaced by the appropriate value of k. So, there is a uniform correspondence between how the different instances of the numeral k are treated in each proof of $Q(k) \vdash Q(s(k))$, which suggests that the free variable version $Q(r) \vdash Q(s(r))$ is provable. The final lines of the general proof provide a proof of $Q(0)$, and therefore the prooftree in PA using induction may be completed. □

The argument above suggests that if the EBG method linearises the ω-proof, then it also suggests a suitable cut formula. It remains to be shown that the EBG method does linearise the ω-proof. Recall that the ω-proof of the generalisation is the EBG proof with the final (instantiated) result. By doing EBG, the parts of a proof which should not be altered when using induction are renamed. This puts the ω-proof into a linear form, in that (for inital line $P(n)$) $P(n)$ reduces to $P(n-1)$, whereas it did not before. More specifically, if differentation of variables apart is a suitable method of generalisation, the initial theorem ($P(n)$) must be of the form $Q(\vec{n})$ where each argument of Q is n (although there might be additional free variables in Q which are not being represented in this notation). Q may have an arbitrary number of argument positions, but for clarity let us consider the simple example when there are three, and when $Q(n_1, n_2, n_3)$ reduces in the ω-proof to $Q(n_1 - 1, n_2, n_3)$. The second and third arguments will renamed by the EGB process, thus giving a linear form. A corresponding argument applies given different argument positions of the altered variable, or varied arities of Q. Thus, EBG is a way of linearising the ω-proof, and if a solution is possible by generalisation of variables apart, then this method will find it (so long as a correct ω-proof is given initially). In conclusion, when a cut formula is suggested by the EBG method proposed, it will have been shown via linearisation of the ω-proof that induction may be successfully carried out upon it. Hence, it is not necessary to actually carry out the induction to know that the proof may be completed by using the cut rule and induction.

The EBG proof provides a most general generalisation of the initial formula which can be proved using the rules given in the ω-proof. However, in some cases (eg. $\forall l\ len(rev(l)) = len(l)$), such a generalisation will be the same, and still not provable by induction, since what is needed is the addition of new structure (ie. an 'accumulator'). However, this additional structure is apparent in the ω-proof, and by means of linearising the ω-proof by looking for repeated structure while allowing some generalising, a correct cut formula may again be suggested [3]. The general proof is put into a form such that there is a repeated rule block i: thus the rth line after r uses of i is $Q(r)$, such that $Q(r)$ reduces to $Q(r-1)$. In this way, correct cut formulae are suggested for many difficult examples [3]: in particular, one generalisation provided by this method of $\forall a\, \forall l\ len(rev(l) <> a) = len(rev(a) <> l)$ (from $len(rev(l)) = len(l)$) is a better result (since it only requires one induction) than the only alternative suggestion provided for this example of $\forall a\, \forall l\ len(rev(l) <> a) = len(a <> l)$[11] (requiring two inductions).

5 Conclusions

In summary, an ω-proof is constructed using some inductive inference process (the individual proofs being easily generated using basic tactics), or obtained by some other means. Unification is used for the stage of constructing a generalised ω-proof and information about unification is fed through various stages of the construction using back propagation. The result is a proof of a generalisation of the original expression, which is linear in form, and hence suggests an appropriate cut formula. The approach also applies generally to other data-types. Not only is it the case that certain new structural patterns may be seen in the ω-proof which may guide generalisation, but also that the general representation of an arbitrary object of that type (eg. $s^n(0)$ for natural numbers, $[x_1, x_2, \ldots x_m]$ for lists, etc.) enables the structure of that particular data-type to be exploited, in the sense that rewrite rules may be used which would not otherwise be applicable.

The new method for generalisation which has been proposed is robust enough to capture in many cases what the alternative methods can do (in some cases with less work), plus it works on examples on which they fail. Implementation of this method has been carried out within the framework of an interactive theorem-prover with Prolog as the tactic language, in which the object-level logic is replaced by classical and constructive theories of arithmetic [2]. Current work involves implementation of an analogous generalisation tactic for the Isabelle proof environment [18]. This approach works for theories other than arithmetic and logics other than a sequent version of the predicate calculus, and may rather be regarded as suggesting a general framework. So long as a procedure for constructing a proof for each individual of a sort is specified, universal statements about objects of the sort could be proved.

[11]cf. [13].

Thus it appears that the approach described in this paper may be an aid to automated deduction, and provides a mechanism for guiding proofs in more conventional systems.

Acknowledgements I would like to acknowledge the help of Alan Smaill, and many others, from the Mathematical Reasoning Group in Edinburgh University, together with the Isabelle group at the Computer Laboratory, Cambridge University.

References

[1] Aubin, R.: 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)

[2] Baker, S., Smaill, A.: A proof environment for arithmetic with the omega rule. Research Paper 645, Dept. of Artificial Intelligence, Edinburgh (1993) To appear in *Proceedings of Computer Science Logic '93* Swansea, Springer Lecture Notes in Computer Science.

[3] Baker, S.: *Aspects of the Constructive Omega Rule within Automated Deduction.* PhD thesis, University of Edinburgh (1992)

[4] Baker, S.: CORE manual. Technical Paper 10, Dept. of Artificial Intelligence, Edinburgh (1993)

[5] Baker, S., Ireland, A., Smaill, A.: On the use of the constructive omega rule within automated deduction. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning – LPAR 92, St. Petersburg,* Lecture Notes in Artificial Intelligence, Springer-Verlag **624** (1992) 214–225

[6] Boyer, R.S., Moore, J.S.: *A Computational Logic.* Academic Press ACM monograph series (1979)

[7] Donat, M.R., Wallen, L.A.: Learning and applying generalised solutions using higher order resolution. In E. Lusk and R. Overbeek, editors, *Lecture Notes in Computer Science* Springer-Verlag **310** (1988) 41–60

[8] Dummett, M.: *Elements of Intuitionism.* Oxford Logic Guides. Oxford Univ. Press, Oxford (1977)

[9] Feferman, S.: Transfinite recursive progressions of axiomatic theories. Journal of Symbolic Logic **27** (1962) 259–316

[10] Gordon, M: HOL: A proof generating system for higher-order logic. In G. Birtwistle and P.A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis* Kluwer (1988)

[11] Hagiya, M.: A Typed λ-Calculus for Proving-by-Example and Bottom-Up Generalisation Procedure. Algorithmic Learning Theory 93, Lecture Notes in Artificial Intelligence **744** (1993)

[12] Hagiya, M.: Programming by example and proving by example using higher-order unification. *10th Conference on Automated Deduction* Lecture Notes in Artificial Intelligence **448** (1990) 588–602

[13] Hesketh, J.T.: *Using Middle-Out Reasoning to Guide Inductive Theorem Proving.* PhD thesis, University of Edinburgh (1991)

[14] Kreisel, G.: On the Interpretation of Non-Finitist Proofs. Journal of Symbolic Logic **17** (1952) 43–58

[15] Madden, P.: The specialization and transformation of constructive existence proofs. In Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* Morgan Kaufmann (1989)

[16] Mitchell, T.M.: Toward combining empirical and analytical methods for inferring heuristics. Technical Report LCSR-TR-27, Laboratory for Computer Science Research, Rutgers University (1982)

[17] Mitchell, T.M., Kellar, R.M., Kedar-Cabelli, S.T.: Explanation-based generalisation: a unifying view. Machine Learning **1(1)** (1986) 47–80

[18] Paulson, L.: Natural Deduction as Higher Order Resolution. *Journal of Logic Programming* **3** (1986) 237–258

[19] Plotkin, G.: A note on inductive generalization. In D Michie and B Meltzer, editors, *Machine Intelligence* **5** Edinburgh University Press (1969) 153–164

[20] Rosser, B.: Gödel-theorems for non-constructive logics. JSL **2** (1937) 129–137

[21] Rouveirol, C.: Saturation: Postponing choices when inverting resolution. In *Proceedings of ECAI-90* (1990) 557–562

[22] Schwichtenberg, H.: Proof theory: Some applications of cut-elimination. In Barwise, editor, *Handbook of Mathematical Logic* North-Holland (1977) 867–896

[23] Shoenfield, J.R.: On a restricted ω-rule. Bull. Acad. Sc. Polon. Sci., Ser. des sc. math., astr. et phys. **7** (1959) 405–7

[24] Takeuti, G.: *Proof theory* North-Holland, 2 edition (1987)

[25] Van der Waerden, B.L.: How the proof of Baudet's conjecture was found. In L. Mirsky, editor, *Papers presented to Richard Rado on the occasion of his sixty-fifth birthday*, Academic Press, London-New York (1971) 252–260

[26] van Harmelen, F.: The CLAM proof planner, user manual and programmer manual: version 1.4. Technical Paper TP-4, DAI, Edinburgh (1989)

[27] Yoccoz, S.:. Constructive aspects of the omega-rule: Application to proof systems in computer science and algorithmic logic. Lecture Notes in Computer Science **379** (1989) 553–565