On the Use of the Constructive Omega-Rule within Automated Deduction *

S. Baker, A. Ireland and A. Smaill †

Abstract

The cut elimination theorem for predicate calculus states that every proof may be replaced by one which does not involve use of the cut rule. This theorem no longer holds when the system is extended with Peano's axioms to give a formalisation for arithmetic. The problem of generalisation results, since arbitrary formulae can be cut in. This makes theorem-proving very difficult — one solution is to embed arithmetic in a stronger system, where cut elimination holds. In particular, this is the case given the addition of the constructive ω-rule: *if each $P(n)$ can be proved in a uniform way (from parameter n), then conclude $\forall n P(n)$.* This paper describes a system based on this rule, and shows that certain statements are provable in this system which are not provable in Peano arithmetic without cut. Moreover, an important application is presented in the form of a new method of generalisation by means of "guiding proofs" in the stronger system, which sometimes succeeds in producing proofs in the original system when other methods fail. The implementation of such a system is also described.

1 Introduction

This paper investigates how the ω-rule may be used in theorem-proving. This rule is of interest for several reasons. Firstly, it may be seen as an attempt to overcome the limitations of reasoning using a formal system that follow from Gödel's first incompleteness theorem. The theorem says that for any formal theory of arithmetic there will always be true statements for that are not theorems of this theory,

*This research was supported by an SERC studentship to the first author and by ESPRIT BRA 3245.

†Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN; Email: siani@uk.ac.edinburgh.

and hence we can never completely formalise arithmetic. Penrose[1], amongst others, has argued that it is mathmatical insight which allows us to see the truth of these true but unprovable statements, and hence that this notion of mathematical insight cannot be completely captured by any algorithm. Now, from an analysis of the incompleteness theorem it can be seen that the problem lies in the formalisation of induction, and that in fact a complete formalisation of arithmetic can be constructed from the Peano axioms and the ω-rule[2]. Unfortunately, the ω-rule cannot be effectively implemented, since it requires the proof of an infinite number of hypotheses. Mathematical induction is the usual computable approximation to the ω-rule. In this paper, we shall present the constructive ω-rule as an alternative approximation. Although this does not in itself circumvent Penrose's argument, the latter rule might perhaps be regarded as a closer approximation to the full ω-rule.

Secondly, the ω-rule allows us to do away with the cut rule. Now, it is a desirable property of a system that cut elimination is valid.[3] This is for reasons of consistency, restriction of the search space (for normal proofs only need be considered), and the derivations of important results such as the subformula property. It is therefore unfortunate that cut elimination is not valid for the usual formulation of arithmetic (namely Peano's axioms plus induction, here in a sequent calculus formulation, which we denote "PA"). Although use of the cut rule may lessen the length and obscurity of a proof, and in addition justify the use of lemmas, automation of the search for a proof may be made extremely difficult as arbitrary formulae may be cut in. The cut formula has to be provided by some sort of heuristic. Unfortunately, there is no easy or fail-safe method of generating the required cut formula. One approach is to try various more general forms of the goal, and hence the problem is termed that of generalisation. Much work has been done in this area, but relatively less in the direction of an alternative solution: essentially, it is the induction rule which is causing the problem, so the approach could be taken of supplementing the system or replacing this rule by some other rule, with the end result that arithmetic would be embedded in a stronger system where cuts may be eliminated. Proofs in this system might then be used to guide the original proof, by suggesting a cut formula. Thus investigation of such a system could be justified in terms of an approach to the problems of generalisation.

Another reason for investigating use of the ω-rule in theorem-proving would be that it is possible to prove more expressions in a system including it. Note that "PA + constructive ω-rule" is a stronger system than PA (according to [Shoenfield 59], it is complete). In addition, as shall be demonstrated, certain expressions can be more easily proved in a system with the constructive ω-rule than Peano Arithmetic, including some not provable at all in the latter without cut.

[1] See [Penrose 89].

[2] See [Orey 56]. Gödel's theorem does not apply here since this is not a formal system in the required sense.

[3] See [Schwichtenberg 77].

The following section introduces the constructive ω-rule and presents a representation of the general proof. Section 3 describes a logic which incorporates the constructive ω-rule, and discusses the question of correctness of the general proof. Section 4 presents a class of proofs which may be proved in the above logic but not in PA \ cut (by the latter we mean PA we with cut rule deleted). Section 5 proposes a method by which proofs in this logic might suggest generalisation in PA, assesses it, gives examples, and compares it with existing methods. Section 6 gives an overview of the implementation of such a system. Finally, related work is discussed and conclusions given.

2 The Constructive Ω-Rule

A suitable rule other than induction which might be added to Peano's axioms to form a system formalising arithmetic is the ω-rule — that rule of inference which allows the inference of the sentence $\forall x A(x)$ from the infinite array $A(n)$ $n \in \omega$ of sentences [Sundholm 78]. However, this is not a good candidate for implementation since there are an infinite number of premisses. A more suitable option is the **constructive ω-rule**, which states:

"If each P(n) can be proved in a uniform way (from parameter n), then conclude ∀nP(n)."

It is with this rule, rather than with the full ω-rule, that the following paper is concerned.

One use of the constructive ω-rule is 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. As mentioned above, 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 by generating proofs of $(0+0)+0 = 0+(0+0), (1+1)+1 = 1+(1+1), \ldots$ and working out the general pattern of these proofs. Note that the most suitable proof depends partly on the type of information available. The presence or ease of obtaining a large number of individual proof instances would encourage the use of such an approach, whereas without these it would be impossible. However, there is scope for automatic generation of these proofs.

What is meant by *general pattern* is best illustrated by examining the form of each individual proof of the equation given above. Let us presume that, within the particular formalisation of arithmetic chosen, one is given the axioms of addition $s(x+y) = s(x)+y$ (1) and $0+y = y$ (2). Each individual proof, for natural number n, will be as follows:

<u>Proof</u>

	$(n+n)+n = n+(n+n)$
$EXPAND\ \ n \rightarrow s^n(0)$	$(s^n(0)+s^n(0))+s^n(0) = s^n(0)+(s^n(0)+s^n(0))$
USE(1) n TIMES	$s^n(0+s^n(0))+s^n(0) = s^n(0+(s^n(0)+s^n(0)))$
USE(2)	$s^{2n}(0)+s^n(0) = s^n(s^n(0)+s^n(0))$
USE(1) n TIMES ON LEFT	$s^n(s^n(0)+s^n(0)) = s^n(s^n(0)+s^n(0))$
	$EQUALITY$

Note that the first step, that of expansion, is an informal one, since it is just useful notation for our own benefit to write $s^n(0)$ as n. By $s^n(0)$ we mean the term formed by applying the successor function n times to 0. The next stages use the axioms as rewrite rules, with the aim of reducing both sides of the equation to the same formula. The general pattern which we are looking is described by the sequence of rule applications, parametrised over n.

The criterion for uniformity of the procedure of proof using the constructive ω-rule is taken to be the provision of a general proof, namely of $P(n)$ in terms of n, each rule R being applied $f_R(n)$ times, for some f_R. It is an important notion that there must be a general proof, for this will guarantee the provision of a proof of $P(k)$, for any number k without having to specify an infinite number of individual cases. 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 we hope to capture.

Now, one way to put the constructive ω-rule into effect is to require that there is an enumeration of the derivations which prove the premisses — for example one could code proofs by numbers, by means of a primitive recursive function which generates them. But we are not going to use such a traditional representation; it is sufficient for our purposes to provide (for the nth case) a description for the general proof in a constructive way. Thus we extend our original system for arithmetic by enriching the syntax with names for the syntax of the original theory and representations of proofs in the original theory.

So, we are led to consider how the general proofs might be represented. They may be regarded as functions from the natural numbers to representations of proofs, ie. gen-pf: $\mathcal{N} \rightarrow \surd\nabla\wr\wr\{\int$, such that gen-pf(n) is a proof with conclusion P(n). An example representation, in the linear case, for the general proof for $P(n)$ is:

$$gen([R1(Pos1, f1'(n)), R2(Pos2, f2'(n)), \ldots]).$$

where the representation for each individual proof for $P(J)$ would be of the form:

$$proof(J, [Rule1_J(Pos1_J, T1_J), Rule2_J(Pos1_J, T2_J), \ldots])$$

This is a predicate, its first argument being the natural number J, and the other argument being an ordered list of predicates such that $RuleK_J$ is the Kth rule to be applied in the proof, and is applied TK_J times, with $PosK_J$ specifying the position at which it is applied. Note that we may have $RuleA_J = RuleB_J$.

In order to *automatically* recognise a general pattern, and hence form the basis for an implementation, upon input of individual proofs for $P(1), P(2), \ldots$, it is necessary to provide an algorithm to do this. This could be of the following general form: First pick some proof of P(J). In general it is best to have J as some fairly low number, but not too low or the proof might exhibit some individual variance. Then rewrite the individual representation of the proof to that of a general proof. Next inspect the representation of the proof of P(J+1), compare this with the general proof, and extend the general form where necessary. Continue in this manner; after a finitely large number of examples, if there is a general pattern to the proofs, then it will be seen to emerge and will be encapsulated by the final representation of the general proof. Note that the algorithm needs to be modified to allow the individual proofs to have rules applied in a different order, provided this gives the same end result. In addition, an internal tree structure needs to be recorded in order to account for branching rules.

Details of the algorithm used in the implementation will not be provided here, since the overall strategy described in this paper is not dependent upon it, and in fact any inductive learning program could be used. The algorithm used is not central to the concern of this paper, and indeed we do not even require a particularly clever algorithm, because it is the underlying structure of the general proof which we wish to exploit, rather than the way in which the general proof is obtained. Related work in the field of inductive inference (ie. presenting algorithms to obtain a general pattern from individual instances) includes that by Kinber and Brazma to formalise the process of generalisation [Kinber & Brazma 90], and Thomas and Jantke's work in generalisation of rewrite rules during the course of inductive completion [Thomas & Jantke 89].

3 An Implementable Logic Incorporating the Ω-Rule

In order to implement the constructive ω-rule and to investigate the nature of the "omega arithmetic" formed by replacing induction by the constructive ω-rule in the standard formalisation of arithmetic, we introduce a new quantifier $\triangle$ such that "$\vdash_{PA+\omega} \triangle x(\Gamma \vdash A(x))$" represents *"there is a general proof of* $\Gamma \vdash A(n)$*, for each* $n \in \omega$*, within the system M"*.

Thus, the constructive ω-rule has the form:

$$\frac{\vdash_{PA+\omega} \triangle x(\Gamma \vdash A(x)) \qquad PREMISS}{\Gamma \vdash_{PA+\omega} \forall x A(x) \qquad CONCLUSION}$$

The extended system has names for sequents from PA, and the occurrence of

"$\Gamma \vdash A(x)$" in the premiss is such a (parametrised) name. We give rules for the new quantifier, so that general proofs are constructed by using these proof rules.

For example, let us consider rules on the right, ie. those which have the form $\frac{\Gamma \vdash f'(P,Q)}{\Gamma \vdash f(P,Q)}$. We arrive at a rule involving $\triangle$ by arguing that a δ-quantified statement asserts the existence of a uniform proof, that each instance can be treated by using the original rule, and so that the general proof may mimic the rule. Thus we have the rule:

$$\begin{array}{ll} \mathit{PREMISS} & \vdash_{PA+\omega} \triangle x[\Gamma \vdash f'(P(x), Q(x))] \\ \cline{2-2} \mathit{CONCLUSION} & \vdash_{PA+\omega} \triangle x[\Gamma \vdash f(P(x), Q(x))] \end{array}$$

All the original rules have analogues in this way. Next we wish to propose some sort of induction rule to capture the idea, not covered in the preceding rules, that a proof for $\forall x P(x)$ may be obtained if a proof of P(s(x)) reduces to the proof of P(x). We simply use induction over natural numbers, but we distinguish the form used as we are now interested in statements about parametrised proof representations.

This is the point at which the introduction of this system is especially justified, for $A \vdash_{PA+\omega} B$ essentially means that the proof of B may be constructed by extending the proof of A. Instead of using rules on A, we use rules on B to reduce it to A, resulting in the axiom $A \vdash_{PA+\omega} A$ (which closes this branch). These rules convey the notion that B is an extension of A, rather than just that B may be proved, perhaps using information provided by A.

The rule $\triangle$-IND, says:

$$\frac{\vdash_{PA+\omega} \Gamma \vdash P(0) \qquad \Gamma \vdash P(n) \vdash_{PA+\omega} \Gamma \vdash P(s(n))}{\vdash_{PA+\omega} \triangle x(\Gamma \vdash P(x))}$$

At this stage the question might be raised that this all seems rather circular, for although we are considering using the ω-rule instead of induction, we are introducing $\triangle$-induction, which might leave us no better off. In fact this is not the case, since $\triangle$-induction is applied to simpler problems – the generalisation problem does not arise, and so we do gain after all.

3.1 On the Correctness of the General Proof

The final result of the stage described in section 2 is that, upon input of individual proof instances of an expression, a general proof will be produced. There is however a problem of how it could be known that the general proof was the right one — in other words, how it could be known that the next individual proof example selected would not falsify it. This necessarily involves some kind of reasoning about proof trees as a check to ensure that the proposed general rule applications do indeed give a proper proof when applied to the general case of the sequent to be proved. Our strategy is to validate general proofs by reasoning about them using the rules of the extended logic described in the previous subsection.

We may define the general proof as a recursive function which, when given a number, returns a partial proof, eg. genpf:$\mathcal{N} \rightarrow \sqrt{\dashv\nabla\sqcup\rangle\dashv\updownarrow}\ \sqrt{\nabla\wr\wr\{\int}$, where genpf(n) is a partial proof-tree for P(n). There will be two sorts of operations to extend the tree: applying a rule R at some tree position Pos(n) will give an updated tree with a single branch added (type 1), and applying a rule R some function of n times at a given position will return a new tree with that number of extra stages (type 2). These may then be tacked together to form the final general proof. $\triangle$-induction may be used to justify type 2 applications. Thus, given $genpf : \mathcal{N} \rightarrow \sqrt{\nabla\wr\wr\{\int}$, if we wish to prove $\forall n \Phi(genpf(n))$, $\triangle$-induction will ask us to prove $\Phi(genpf(0))$ and, given a proof of $\Phi(genpf(x))$, to show that there is a proof of $\Phi(genpf(s(x)))$, where x is any natural number.

So, the general idea of the solution is to take the nth case, and apply the rules in the manner indicated by the general proof, in the hope of reaching truth or equality (as in the example on P3). But how is it actually possible to carry out this checking within the framework of a computer program? In particular, how can we apply a rule n times, when we do not know the value of n? If we keep applying a rule transformation, we will not know when to stop.

Let us imagine we are trying to prove $\forall x P(x)$ by means of the constructive ω-rule. We have obtained a proposed general proof via the previous stages of the implementation. We look at a random case for the number n, namely $P(n)$. Firstly, we substitute $n \rightarrow s^n(0)$, and similarly for other numbers. Recall that the general solution is stored in terms of rules of the form "apply R f(n) times at position P_0". This will mean that a subexpression, SE(0), of $P(s^n(0))$ will be transformed by the rule to SE(f(n)), which will then be substituted for SE(0) at position P_0 in $P(s^n(0))$.

This may be automated by applying the rule R a few times in order to *guess* the general pattern in terms of m, the number of times the rule is applied. For example, if SE(0) $= s^n(0) + y$ and R is $s(x) + y \rightarrow s(x + y)$,

$$\begin{array}{rrcl@{\quad}c@{\quad}l}
 & SE(1) & \equiv & s(s^{n-1}(0)+y) & & \\
 & SE(2) & \equiv & s(s(s^{n-2}(0)+y)) & \equiv & s^2(s^{n-2}(0)+y) \\
 & SE(3) & \equiv & s(s(s(s^{n-3}(0)+y))) & \equiv & s^3(s^{n-3}(0)+y) \\
GUESS & SE(m) & \equiv & s^m(s^{n-m}(0)+y) & &
\end{array}$$

To allow for constants, a list of possible guesses should be made for each SE(r), and the intersection of these given as SE(m).

This guess is then proved using induction over the proof tree representations. If this cannot be done since the guess was wrong, move back to the previous stage and try again with a more complicated SE(m), dependent on a larger number of base examples. Note that a correct guess will be proved by the $\triangle$-induction because SE(s(r)) is reducible to SE(r) by applying R in reverse, and tidying up the expression using definitions such as $s^n(0) = s(s^{n-1}(0))$.

The whole process is then repeated with the new expression and rule from the proposed general proof. Substitution of the transformation given by the last rule provided by the general proof should result in a $PA + \omega$-axiom, such as $\vdash X = Y$, where Y is reducible to X by various syntax definitions. If it does not, then the general proof is not a correct one as there must be cases in which it could be falsified.

The proof tree in $PA + \omega$ representing this general-proof checking stage for the above substitution in $OldExp$ to obtain $NewExp$ is as follows:

$$\dfrac{\dfrac{\vdash_{PA+\omega} m = f(n) \qquad \dfrac{\dfrac{}{\vdash SE(0) = SE(0)}\,axiom \qquad \dfrac{CLOSES}{\vdash SE(r) = SE(0) \vdash_{PA+\omega} SE(s(r)) = SE(0)}\,rev-rule}{\vdash_{PA+\omega} \triangle m(\vdash SE(m) = SE(0))}\,\triangle - ind}{\vdash_{PA+\omega} \vdash SE(f(n)) = SE(0)}\,subst \qquad \vdash_{PA+\omega} \vdash NewExp}{\vdash_{PA+\omega} \vdash OldExp}\,subst$$

4 Comparison of Systems

This section investigates whether there are any expressions provable in $PA + \omega \setminus cut$ but not in $PA \setminus cut$. We will consider the class of proofs, such as $(x + x) + x = x + (x + x)$, which are provable only using cut.[4] For example, these would include the standard proofs that induction necessitates use of the cut rule in arithmetic. (Note that the relations discussed are necessarily those of a most basic nature, and the theory would have to be developed to take account of case splits and conditionals in the proofs.) It is not the case that a proof of P(s(x)) may be given in terms of an extension to the proof of P(x), eg.

$$\begin{array}{rcll}
(s(x) + s(x)) + s(x) & = & s(x) + (s(x) + s(x)) & \equiv \mathbf{P(s(x))} \\
s(x + s(x)) + s(x) & = & s(x + (s(x) + s(x))) & \\
s((x + s(x)) + s(x)) & = & s(x + s(x + s(x))) & \\
(x + \underbrace{s(x)}_{\lambda}) + \underbrace{s(x)}_{\lambda} & = & x + s(x + \underbrace{s(x)}_{\lambda}) & \neq \mathbf{P(x)}
\end{array}$$

By rules (1) s(x + y) = s(x) + y, and (2) 0 + y = y, if we have (p + q), we can never alter q. Thus, the terms λ cannot be reduced. Hence, we can only hope to make x disappear. This will only work if x = 0 (by use of (2)), or if x is written as $s^n(0)$, and then reduced. In this case we will never be able to reduce P(s(x)) to P(x). This is basically also an informal proof of why propositions of this type may not be proved without the use of cut.

[4]Only an informal proof will be attempted within the scope of this paper.

Induction in PA does not work, because P(s(x)) may not be given in terms of P(x):

$$\frac{\vdash (0+0)+0 = 0+(0+0) \qquad (r+r)+r = r+(r+r) \vdash (s(r)+s(r))+s(r) = s(r)+(s(r)+s(r))^{\dagger}}{\vdash \forall x (x+x)+x = x+(x+x)}$$

† is not provable without using cut, as shown above.

However, △-induction does work, although standard induction does not, for the following crucial reason. The proof of the successor case is not an extension of the proof of the case of its predecessor, as the latter may not be substituted into the former, but instead we may use the rules for the latter as a "guide" for the rules of the former. So the proof of the successor case is described by a uniform transformation on the proof of the previous case, but this transformation is not restricted to extension of the previous proof.

To illustrate:

By the use of △-induction, we wish to prove:

$$\underbrace{(n+n)+n = n+(n+n)}_{\mathcal{A}} \vdash_{PA+\omega} \underbrace{(s(n)+s(n))+\overbrace{s(n)}^{\mathcal{C}} = s(n)+(s(n)+s(n))}_{\mathcal{B}}$$

We cannot substitute $\mathcal{A}$ directly in a proof of $\mathcal{B}$, because terms like $\mathcal{C}$ (essentially unaltered by the rules given) must be distinct from the other terms in the formula, in the sense that they must have a distinct variable. This is because we do not wish to carry out induction on these "unaltered terms" as we wish them to remain the same. In this case this is not so, and hence standard induction does not work and the formula is only provable using the cut rule, at the object level. (See above, noting the correspondence between $\mathcal{C}$ and λ.)

However, we may convert rules for proving $\mathcal{A}$ to those for proving $\mathcal{B}$ by the "meta-rule" $rule_i(n) \Rightarrow rule_i(s(n))$. Any uniform manipulation on the nth case proof (ie. proof of $\mathcal{A}$), resulting in the n+1th case proof (ie. proof of $\mathcal{B}$) will achieve the desired effect. As a result we are able to prove $\mathcal{B}$, without substituting $\mathcal{A}$ directly.

In conclusion, in cases of this type, we can avoid doing a cut at the object-level by doing △-induction in $PA+\omega \setminus cut$. Thus, the existence of the logic $PA+\omega \setminus cut$ (and of △-induction) is justified, for by this means a proof may be given of those propositions not provable without using the cut rule, the latter being considered undesirable as regards the automation of proofs. So we have shown that certain proofs not provable without cut in PA are provable in the stronger system. Next we move on to consider how this proof might suggest the form of the proof in PA (using cut).

5 How Proofs in Omega-Arithmetic Might Suggest Generalisation in Peano-Arithmetic

For the class of proofs, such as $(x + x) + x = x + (x + x)$, which are provable only using the cut rule, we consider whether the proof of the nth case, derived when doing △-induction within a variant of omega-arithmetic suggests an object-level proof, ie. in particular, what the **CUT FORMULA** would be in a proof in Peano Arithmetic?

For example, what would the A be below?

$$\frac{A \vdash (x + x) + x = x + (x + x) \qquad \vdash A}{\vdash (x + x) + x = x + (x + x)^*} CUT$$

Ordinary induction does not work on $*$. That is why we have to use the cut rule. We would wish A to be a more general version of $*$, so that we could prove $A \vdash *$, but on the other hand to be suitable to give an inductive proof, so that we could prove $\vdash A$ by induction. Hence we would be tackling the problem of generalisation by using an alternative (stronger) representation of arithmetic as a guide.

One answer would be to see what remains unaltered in the nth case proof, (viz. the "general proof") and then write out the original formula, but with that variable re-named. In this case, this would give

$$A \equiv (x + y) + y = x + (y + y).$$

A could then be proved by induction on x. Note that what is meant by 'unaltered' is defined by what is unaffected by the rewrite rules. Note also that it has not been proved that the cut formula would definitely work, but rather postulated that if there were a cut formula which did work, then it would be of the form where induction was not carried out on these 'unaltered' variables.

The detection of the unaltered variable could also take place by means of a comparison of the original formula $*$ with the axioms given. (The latter idea is to derive rewrite rules of the form $s^n(0) + y \Rightarrow s^n(0 + y)$, say, and compare the argument positions of the original formula which are unaffected by such rules). The (n+1)th proof, if available, would not provide any additional information in this respect.

5.1 Comparison of Different Generalisation Methods

This method should be compared with the approach of Raymond Aubin to generalisation [Aubin 75], which is a development of Boyer and Moore's work [Boyer & Moore 88]. 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 not, then he will "guess" a different generalisation. Obviously, this is a very 'ad hoc' approach, and moreover, his 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.

In general, Aubin is unable to generalise expressions which would require a generalisation of a variable which is present in the equation both individually and as an argument to a function (giving a term), for example when x and s(x) both need to be generalised to y and s(y), in order to do induction on y. So for the example $x + s(x) = s(x) + x$, Aubin's method would fail. The ω-guiding would suggest a cut of $x + s(y) = s(x) + y$, which works. The Boyer and Moore system (and hence CLAM,[5] the theorem-proving system developed at Edinburgh which is an implementation of their strategy) would look for common subexpressions, and hence give the result $x + y = y + x$. This would be a valid approach, but it might well be that insufficient axioms were available to be able to prove the commutativity of plus, or that the method proposed above might be considered in a sense a more direct approach.

5.2 A Worked Example Illustrating the Difference Between Various Methods of Generalisation

In this section we compare and contrast the various methods of generalisation applied to the example $x \neq 0 \rightarrow p(x) + s(s(x)) = s(x) + x$. This is an interesting example due to the fact that its proof must involve at least one cut, that it is a conditional proposition and further, that both constructor and destructor functions are involved.

The axioms used will be as follows:

$$0 + y = y \qquad (1)$$
$$s(x) + y = s(x + y) \qquad (2)$$
$$x = 0 \rightarrow p(x) = x \qquad (3)$$
$$x \neq 0 \rightarrow p(s(x)) = x \qquad (4)$$

The Boyer-Moore Theorem Prover fails when asked to prove this example, using only the recursion equations.[6]

By Aubin's method, using the definition

$$X + Y = cond2(X, s(p(X) + Y), Y)^7$$

[5]See [vanHarmelen 89] for details. It is a system based on OYSTER to support the use of proof plans. OYSTER is a reconstruction in Prolog of Nuprl [Constable 86] by C. Horn, a visitor to the Artificial Intelligence Department at Edinburgh.

[6]For printout of failed attempt, see [Baker 91].

[7]This definition, plus (3) and (4), is equivalent to the axioms (1)-(4) above. (*cond2(X,Y,Z)*

we get

$$s(p(\underline{p(x)}) + s(s(x)))$$

and

$$s(p(s(x)) + x)$$

The ugly expressions are underlined. (An ugly expression is the argument to a destructor function [in this case 'p'] such that the complex structure containing it cannot be broken down by rewrite rules, since there are none applicable). The second expression reduces to s(x+x), and so there is no ugly expression for the right hand side of the equation. Aubin requires that there should be the same ugly expression on each side of the equation before generalisation may take place, and hence his method fails.

Let us next consider the guiding method proposed in this paper. The general nth case proof will be of the following form (we ignore case splits to simplify the presentation):

$$\begin{array}{rcl}
p(s^n(0)) + \underbrace{s(s(s^n(0)))}_{unchanged} & = & s(s^n(0)) + \underbrace{s^n(0)}_{unchanged} \\
s^{n-1}(0) + s^{n+2}(0) & = & s^{n+1}(0) + s^n(0) \\
s^{n-1}(0 + s^{n+2}(0)) & = & s^n(s(0) + s^n(0)) \\
s^{n-1}(s^{n+2}(0)) & = & s^{n+1}(0 + s^n(0)) \\
s^{2n+1}(0) & = & s^{2n+1}(0)
\end{array}$$

According to this method, the unchanged terms are renamed, to avoid induction being applied to them. Hence, the generalisation is:

$$x \neq 0 \rightarrow p(x) + s(s(z)) = s(x) + z$$

and the solution yielded is as follows (base case omitted):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{0 \neq 0 \rightarrow s(s(z)) = s(z) \vdash 0 = s(0)}{0 \neq 0 \rightarrow p(0) + s(s(z)) = s(0) + z \vdash 0 = s(p(0))}\,A1, A2 \qquad \dfrac{\vdash s(m) = s(m)}{m = s(p(m)) \vdash s(m) = s(p(s(m)))}\,A4}{r \neq 0 \rightarrow \ldots \vdash r = s(p(r))}\,ind(r)}{\ldots \vdash r + s(s(z)) = s(p(r)) + s(s(z))}\,structural rule}{\ldots \vdash r + s(s(z)) = s(p(r) + s(s(z)))}\,A1}{\ldots \vdash (s(r) \neq 0 \rightarrow) r + s(s(z)) = s(s(r) + z)}\,hyp(subst)}{\dfrac{r \neq 0 \rightarrow p(r) + s(s(z)) = s(r) + z \vdash s(r) \neq 0 \rightarrow p(s(r)) + s(s(z)) = s(s(r)) + z}{\vdash x \neq 0 \rightarrow p(x) + s(s(z)) = s(x) + z}\,ind(x)}\,A4, A2$$

represents "if X=0, then Z, else Y".)

5.3 Discussion and Conclusions

The Boyer and Moore method for generalisation seems somewhat 'ad hoc'. Identical terms occurring on both the left and right side of the equation are picked for rewriting as a new variable. 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 proof is not 'tailor-made' to the axioms given in the sense that the proof provided by the guiding method will be.

Aubin's method is partially theoretically justified - he will pick out at least some of the 'problem cases', but not necessarily all of them, and easily too many, by his method of finding ugly expressions. Hence he has to have some method to guess the pairs when he has too many (he chooses the left-most ones), and in the case when there are too few, the method fails. There is a danger of overgeneralisation to non-theorems with both he above methods.

Using the guiding method, the proof is best suited to the given axioms (unlike CLAM, which takes no account of the form of the axioms when choosing a generalisation). Hence it might be argued that a better, 'easier' proof is provided. This is because the structure of the proof according to the axioms given is being revealed by the general proof. Hence, it is not necessary to store or prove large numbers of sub-lemmas. The general proof shows what we *do not* wish to do induction upon, rather than what we *do* (as Aubin's method does). The problem with which Aubin is faced, namely candidates for induction appearing on one side only of the equation, will not occur, and so this will not result in a failure of the method. Hence, this method 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. It provides a uniform approach and does not have to check extra criteria, nor work through individual examples. There is less likelihood of overgeneralisation to a non-theorem.

A selection of the theorems proved by this method is listed in Table 1. Note that examples 10-15 involve generalisation apart, or else generalisation of common subexpressions. In these cases the method works fairly straightforwardly. Further work is currently being carried out to extend this method to other forms of generalisation, such as when using accumulators. In some cases, an ω-proof may be provided, but it is not clear what the cut formula might be.

In example 16, the axioms and general proof are as follows.

Axioms

$$\begin{aligned} \forall x\, \forall y\; s(x)+y &= s(x+y) && (5)\\ \forall y\; 0+y &= y && (6)\\ even(0) &= true && (7)\\ even(s(0)) &= false && (8) \end{aligned}$$

$$\forall n\ even(s(s(n))) \quad = \quad even(n) \tag{9}$$

General Proof

	$even(n+n) = true$
$EXPAND\ n \rightarrow s^n(0)$	$even(s^n(0)+s^n(0)) = true$
$USE\ (5)\ n\ times$	$even(s^n(0+s^n(0))) = true$
$USE\ (6)$	$even(s^{2n}(0)) = true$
$USE\ (9)$	$even(0) = true$
$USE\ (7)$	**CORRECT**

The Boyer-Moore strategy does not apply here. Obvious choices for the cut formula here would result in circularity or a non-theorem, but the general proof gives some clues as to what is happening, and does not suggest an incorrect cut formula. Indeed, the cut formula which is suggested is that of $even(2.y)$, because of the way a successor function from each of the original variables pairs off together in the general proof (notice the occurrence of the formula $even(s^{2n}(0))$). This is a useful improvement on the alternative methods.

$$\begin{aligned}
\forall x\ x+s(x) &= s(x+x) && (10)\\
\forall x\ (x+x)+x &= x+(x+x) && (11)\\
\forall x\ x+s(x) &= s(x)+x && (12)\\
\forall x\ x.(x+x) &= x.x+x.x && (13)\\
\forall x\ (2+x)+x &= 2+(x+x) && (14)\\
\forall x\ \neq 0 \rightarrow p(x)+s(s(x)) &= s(x)+x && (15)\\
\forall x\ even(x+x) &= true && (16)
\end{aligned}$$

Note that the above expressions may not be proved without the use of the cut rule in Peano Arithmetic. They may all be proved by the method proposed (namely by using the constructive ω-rule) and a correct cut formula produced as appropriate.

Table 1: Examples of Theorems Proved

6 Implementation

The implementation is 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. Any finitely large number of individual instances of proofs of a proposition may be generated automatically by the use of various tactics. These may then automatically be converted to the form *proof*/2 detailed in Section 2, and then the general representation of the proofs is provided. The latter is done by having an initial generalisation from a 'starting' proof, and then by updating this general proof using the other individual proofs, until the general proof seems to have reached a stable form. This general proof is then checked to see if it is indeed the correct one, as described in Section 3.

There are two options which are allowable from the object-level proof, with goal $\Gamma \vdash \forall x P(x)$. One is to ask to use the constructive ω-rule, whereby the system will check to see whether there is a correct general proof, and then return to the former system and close the branch, or else state that this rule may not be used. The user may then continue to investigate other positions in the proof tree. The other option is to ask for an appropriate cut to be carried out at the object-level (the cut being worked out by the system from the general proof), with a further option to complete the tree as far as possible (using the CLAM proof-planner). The general proof may be provided automatically, but there is an option in each case to switch temporarily to another system which will allow for the description, manipulation and display of the general proof. The user may specify the proof incrementally, in terms of object applications in positions in the tree, plus induction over a distinguished parameter, or all at once — and this is checked. The system builds up a recursive function description of the general proof, and is able to display individual proofs in addition to the general case. The system is currently being extended to incorporate conditional and case-split proofs.

7 Conclusions

With the goal of automatic derivation of proofs within some formalisation of arithmetic in mind, a logic incorporating the ω–rule has been proposed and it has been demonstrated that certain expressions not provable in the normal formalisation of arithmetic without cut are provable in this logic. In addition, a new method of generalisation is proposed by means of the ω-proof "guiding" a proof at the object level, which may succeed in providing a valid proof at the object-level when other methods of generalisation fail. This work has been implemented in a way which allows interaction with the user, and the construction of a recursive general proof. When a general proof is available, or proofs of instances can be found easily, it may be easier to use this approach than to search directly for a proof in PA. Moreover, there is no problem of generalisation in this system.

Note that this general approach would work 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 general procedure for constructing a proof for each individual of a sort is specified, universal statements about objects of the sort could be proved. 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.

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.

[Baker 91] S. Baker. Notes on the constructive omega-rule and a new method of generalisation. Internal working paper, Dept of Artificial Intelligence, Edinburgh University, March 1991.

[Boyer & Moore 88] R.S. Boyer and J.S. Moore. *A Computational Logic Handbook*. Academic Press, Boston, 1988.

[Constable 86] R.L. et al. Constable. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, Inc., New Jersey, 1986.

[Kinber & Brazma 90] E.B. Kinber and A.N. Brazma. Models of inductive synthesis. *Journal of Logic Programming*, 9:221–233, 1990.

[Orey 56] S. Orey. On ω-consistency and related properties. *The Journal of Symbolic Logic*, 21:246–52, 1956.

[Penrose 89] R. Penrose. *The Emperor's New Mind*. Vintage, 1989.

[Schwichtenberg 77] H. Schwichtenberg. Proof theory: Some applications of cut-elimination. In Barwise, editor, *Handbook of Mathematical Logic*, pages 867–896. North-Holland, 1977.

[Shoenfield 59] J.R. Shoenfield. On a restricted ω-rule. *Bull. Acad. Sc. Polon. Sci., Ser. des sc. math., astr. et phys.*, 7:405–7, 1959.

[Sundholm 78] B.G. Sundholm. The omega rule: A survey. University of Oxford, 1978. B. Phil Thesis.

[Thomas & Jantke 89] M. Thomas and K.P. Jantke. Two inductive inference algorithms for solving divergence in Knuth Bendix completion. Departmental research report, Department of Computing Science, University of Glasgow, July 1989.

[vanHarmelen 89] F. van Harmelen. The clam proof planner. Dai technical paper no. 4, Dept Artificial Intelligence, University of Edinburgh, 1989.