*Technical Report*

UCAM-CL-TR-345
ISSN 1476-2986

Number 345

UNIVERSITY OF CAMBRIDGE

**Computer Laboratory**

# A proof environment for arithmetic with the Omega rule

Siani L. Baker

August 1994

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

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

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

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

ISSN 1476-2986

# A Proof Environment for Arithmetic with the Omega Rule*

*Siani L. Baker*
Computer Laboratory
University of Cambridge
Pembroke Street
Cambridge CB2 3QG
Siani.Baker@cl.cam.ac.uk

**Abstract**

An important technique for investigating derivability in formal systems of arithmetic has been to embed such systems into semi-formal systems with the $\omega$-rule. This paper exploits this notion within the domain of automated theorem-proving and discusses the implementation of such a proof environment, namely the CORE system which implements a version of the primitive recursive $\omega$-rule. This involves providing an appropriate representation for infinite proofs, and a means of verifying properties of such objects. By means of the CORE system, from a finite number of instances a conjecture for a proof of the universally quantified formula is automatically derived by an inductive inference algorithm, and checked for correctness. In addition, candidates for cut formulae may be generated by an explanation-based learning algorithm. This is an alternative approach to reasoning about inductively defined domains from traditional structural induction, which may sometimes be more intuitive.

**Key words.** Automated theorem proving, $\omega$-rule, infinite proofs.

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

# Contents

# 1 Introduction

Normally, proofs considered in theorem-proving are finite; however, there is a reasonable notion of infinite proof involving the $\omega$-rule, which infers a proposition from an infinite number of individual cases of that proposition. The $\omega$-rule involves the use of infinite proofs, and therefore poses a problem as far as implementation is concerned.

With the goal of automatic derivation of proofs within some formalisation of arithmetic in mind, an (implementable) representation for an arithmetical system including the $\omega$-rule is proposed. The implemented system is useful as a proof environment (and incidentally also as a guide to generalisation in the more usual formalisation of arithmetic [Baker 94]).

The following sections present a formalisation of arithmetic with the $\omega$-rule ($PA_{c\omega}$), discuss how this was correctly implemented to produce the framework of the CORE proof environment, and establish soundness of the implementational system with respect to $PA_{c\omega}$. We also give an indication of how the system can be used.

# 2 The Constructive Omega Rule

In this section a constructive version of the infinitary $\omega$-rule is introduced as an interesting alternative to induction proofs in arithmetic. A standard form of the $\omega$-rule is

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

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 $(PA)$[1], 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 $PA \vdash G(x)$. 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 $\omega$-rule so that the infinite proofs considered possess some important properties of finite proofs. One suitable option is to use a **constructive $\omega$-rule**. The $\omega$-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 $A(\underline{n})$ [Takeuti 87]. 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) [Yoccoz 89]. There is a primitive recursive counterpart[2]

[1] See for example [Schwichtenberg 77] for a formalisation.

[2] In other words, such that there is a primitive recursive function $f$ for which, for every $n$, $f(n)$ is a Gödel number of the proof of $A(\underline{n})$, the $n$th numerator of the $\omega$-rule.

which is also a candidate for implementation. Note that in particular these rules differ from the form of the $\omega$-rule (involving the notion of provability) considered by Rosser [Rosser 37] and subsequently Feferman [Feferman 62].

Various theoretical results are known for these systems. Shoenfield has shown that '$PA$ + $\omega$-rule' ($PA_\omega$)[3] is equivalent to '$PA$ + recursively restricted $\omega$-rule' [Shoenfield 59]. The sequent calculus enriched with the recursively restricted $\omega$-rule in place of the rule of induction (let us call it $PA_{r\omega}$ [4]) has cut elimination, and is complete [Shoenfield 59].

The primitive recursive variant has also been shown to be complete by Nelson [Nelson 71]. If one has the rule of repetition $\frac{\Gamma\vdash\Delta}{\Gamma\vdash\Delta}$ in $PA_\omega$, any recursive derivation can be "stretched out" to a primitive recursive derivation using the same rules of inference, plus this rule [López-Escobar 76, P169]. Since our implementation is developed using effective operations over representations of object-level syntax (where effectiveness is an analogous concept to primitive recursion), and $PA$ with the unrestricted $\omega$-rule forms a conservative extension of this system, the (classical) system $PA$ with a primitive recursive restriction on the proof-trees was chosen as a basis for implementation.

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 we do not claim completeness, some proofs that normally require generalisation can be generated more easily in $PA_{c\omega}$ than $PA$.

# 3 $PA_{C\omega}$: Arithmetic with the Constructive Omega Rule

The system $PA_\omega$ is essentially $PA$ enriched with the $\omega$-rule in place of the rule of induction. The derivations are then infinite trees of formulae; a formula is demonstrated in $PA_\omega$ by "exhibiting" a proof-tree labelled at the root with the given formula. Syntactical details about this system $PA_\omega$ are given in [López-Escobar 76, P162] (see [Prawitz 71, P266–267] for a natural deduction representation). $PA_\omega$ has been described by Schütte as a semi-formal system to stress the difference between this and usual formal systems which use finitary rules [Schütte 77, P174].

For implementational purposes, infinite proofs must be thought of in the constructive sense of being generated, rather than absolute. It is necessary to place a restriction on the proof-trees of $PA_\omega$ such that only those which have been constructively generated are allowed, in order to capture the notion of infinite labelled trees in a finite way. The normal approach when dealing with a system with infinitary proofs such as $PA_\omega$ is to work with numeric codes for the derivations rather than using the derivations themselves. See [Schwichtenberg 77, P886] for further details, including the case of the $\omega$-rule. By adding the provability relation and numeric encoding, a reflection system which necessarily extends the original one may

[3] See Section 3 below for description.

[4] For a more formal description see [Baker 92a].

be formed [Kreisel 65, P163]. However, the necessity of using this Gödel numbering approach may be avoided by following Tucker in defining primitive recursion ("effectiveness") over various data-types that are better adapted to computational purposes [Tucker *et al* 90].

If an arithmetical encoding method were to be used, the primitive recursive constraint could be attached directly to the $\omega$-rule. However, without using such an approach the restriction must be placed on the shape of the proof tree in which the $\omega$-rule appears: only derivations which are "effective" will be accepted.

Hence we define

$$\vdash_{PA_{c\omega}} \Phi \quad \textit{iff} \quad \exists f.\ f \text{ is an 'effective' proof-tree of } PA_{c\omega} \text{ with } \Phi \text{ as initial sequent.}$$

$PA_{c\omega}$ may be defined as a (semi-)formal system by further specifying axioms and rules of inference (in this case, corresponding to those of $PA$: "this wholesale carry over of derived rules from predicate logic is one of the special virtues of cut free infinite proofs" [Kreisel 65, P166]).

The alternative, standard approach is to use numeric encoding (using notation $\ulcorner . \urcorner$) and strengthen $PA$ by adding an arithmetic schema of the form:

$$(\exists \Pi\ (\text{proof-tree}(\Pi)\ \wedge\ \text{conc}(\Pi) = \ulcorner \Phi \urcorner)) \rightarrow \Phi.$$

We must now provide some means for reasoning about primitive recursive infinite proof trees. The objects of interest are recursive (possibly infinite) proof-trees (in the sense of Löpez-Escobar [Löpez-Escobar 76]), labelled with formulae (namely, the sequents to be proved at each point) and rules. The notion of effectiveness of a tree, which corresponds to primitive recursion, is defined in [Baker 92a]. In addition, a (proof) tree must be well-founded, in the sense that it does not have an infinitely deep branch. The rules that relate the formulae between node and subnode are the standard rules for the logical connectives, the extra $\omega$-rule with subgoals $\Phi(\underline{0}), \Phi(\underline{1}), \ldots$, and substitution. A formula in $PA$ is demonstrated in the extended theory by exhibiting a proof-tree labelled at the root with the given formula. Properties of such primitively recursively defined trees can be proved using induction principles associated with the datatypes, as we see in section 5. These are the sorts of proofs that have been automated by [Bundy *et al* 93], and we are able to automate the simpler proofs that arise here. This involves, for example, giving a proof that a given rewrite applied a given number of times to a formula schema yields a particular formula schema.

## 3.1 Definition of effective prooftrees for $PA_{c\omega}$

These notions are now formalised. We define prooftrees as functions

$$f : \text{Position of Node} \mapsto (\text{Sequent at Node, Rule used at Node}),$$

where the range specifies labels, or symbols, in the tree associated with each node, and the position is represented by lists of natural numbers.

**Definition 3.1 (Effective prooftree for $PA_{c\omega}$)** *$f$ is an effective prooftree for $PA_{c\omega}$ if and only if $f$ is an effective function $f : nat\ list \to seq \times rule$ such that $f$ is well-founded and correct.*

**Definition 3.2 (Order in tree)** *Define the relation $\leq$ on nat list $\times$ nat list by:*

$$Pos1 \leq Pos2 \leftrightarrow \exists l\ Pos2 = Pos1 <> l,\ l \in nat\ list$$

**Definition 3.3 (Empty node)** *$\breve{\in}$, the empty label, is shorthand for* (dummy seq, dummy rule)*, and indicates that there is nothing at a particular node.*

**Definition 3.4 (Derivation in $PA_{c\omega}$)** *$f : nat\ list \to seq{*}rule$ describes a derivation in $PA_{c\omega}$ for the sequent $\Phi$, where $\Phi$ is the sequent at the top of the tree (viz. at the node [ ]) if:*

1. *$\{p : nat\ list | f(p) \neq \breve{\in}\}$ is a well-founded tree according to $\leq$.*
2. *If $f(p) \neq \breve{\in}$, then $q_2^1(f(p))$ is a sentence associated with the node $p$ (namely the sequent to be proved), and $q_2^2(f(p))$ is the name of a rule of $PA_{c\omega}$ used to produce its immediate successors, where $q_i$ are projection functions such that $q_2^1(A, B) = A$ and $q_2^2(A, B) = B$.*
3. *If $p$ is a bottommost node in the tree, ie. $f(q) = \breve{\in}$ for all $q$ such that $p \leq q$, and $f(p) \neq \breve{\in}$, then either $q_2^1(f(p))$ is an axiom of $PA_{c\omega}$ and $q_2^2(f(p))$ is axiom, or else $f(p)$ is set to* incomplete *to indicate that the tree is incomplete.*
4. *If $Pos <> [K]$ ($K \in \mathbb{N}$) is not a bottommost node, then $q_2^1(f(Pos <> [K]))$ is the $K$th subgoal of $q_2^2(f(Pos <> [K]))$ applied to $q_2^1(f(Pos))$.*

**Definition 3.5 (Incomplete tree)** *The derivation is incomplete if not all the leaves are closed ie. if* incomplete *is associated with any node in the tree.*

**Definition 3.6 (Prooftree)** *The derivation will be a prooftree if it is a complete derivation, in other words if all its leaves are axioms (and the others marked as dummy nodes, if appropriate, since there is infinite branching at each node).*

**Definition 3.7 (Subgoals)** *The* subgoals *of $p$ may be defined as $subgoals(p) = \{q_2^1(f(p <> [n]))|n \in \mathbb{N}, f(p <> [n]) \neq \breve{\in}\}$.*

### 3.1.1 Consequences of this definition

Properties of the tree will be:

1. Defining $br(Rule)$, where $Rule$ is a rule of $PA_{c\omega}$, as the number of subgoals of $Rule$ (ie. $br(\forall r_\omega) = \omega$, $br(\to r = 1$, $br(\to l) = 2$ etc.), if $L \neq \omega$, where $br(q_2^2(f(Pos))) = L, L \in \mathbb{N}$, then $f(Pos <> [M]) = \breve{\in}\ \forall M \geq L, M \in \mathbb{N}$.[5] Since $br(axiom) = 0$, if $p \leq q$ and $p \neq q$ for some position representation $p$,$q$, where $p$ is a bottommost node in the tree, then $f(q) = \breve{\in}$.

[5] $\geq$, since the natural numbers, and the subnodes, are taken to start at 0.

2. If $f(Pos <> [I]) = \breve{\in}$, then $f(Pos <> [J]) = \breve{\in}\ \forall J \geq I \in \mathbb{N}$.

The approach described above is suitable for automation, since it generates the subgoals of the $\omega$-rule, rather than having to check their presence.

## 3.2 Correctness of prooftrees

There are two main notions of correctness of well-founded trees, namely 'local' correctness, which checks that an appropriate rule is applied at each node of the tree, and 'global' correctness, which is concerned with whether the tree is well-founded.

To check for local correctness, structural induction is used over the prooftrees. In this case, the corresponding meta-induction (for *nat list*) is used for the tree-defining function $f$ : *nat list* $\rightarrow$ *string* $\times$ *string*. Thus:

$$\frac{f([\,]) \quad f(Pos) \Rightarrow f(Pos <> [k])}{\forall x\ f(x)}$$

where $Pos, x \in$ *nat list* and $k \in$ *nat*. That is to say that given an initial sequent (and initial rule used), plus a way of obtaining from a sequent at some position the sequent at a node directly below that position, then the sequent at each node of the tree is defined. This process of obtaining sequents at subnodes, given a sequent at a node, is carried out by applying the rule associated with the node to the sequent at the node, and is described above. The result is uniquely determined, given a rule and sequent, and hence the prooftrees are locally correct. Transfinite induction over the partial ordering $\leq$ of the tree representation is allowable if numeric encoding is used at each node [Kreisel 65, P163].

At this stage it could be objected that there might be circularity, for although the $\omega$-rule is used instead of induction, meta-induction is being introduced here, which might result in there being no advance. However, the generalisation problem does not in practice occur in the theory of trees, and so there is a gain after all.

# 4 $\omega$-Proofs

This section deals with the issues involved in making the $\omega$-rule into a rule for machine proof. One use of the constructive $\omega$-rule is to enable automated proof of formulae, such as $(x + x) + x = x + (x + x)$, which cannot be proved in the normal axiomatisation of arithmetic without recourse to the cut rule, which is the logical justification of any generalisation step. In these cases the correct proof could be extremely difficult to find automatically. However, it is possible to prove this equation using the $\omega$-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. The algorithm used to automatically recognise the general pattern generalises an initial set of rewrite rules describing an individual proof, and then updates this generalisation according to other individual proof examples until the general proof representation ($\omega$-proof) satisfies all of the (large number of) cases considered; any appropriate inductive inference algorithm, such as Plotkin's least general

generalisation [Plotkin 69], or that of Rouveirol, who has tackled the problem of controlling the hypothesis generation process to get only the most relevant candidates [Rouveirol 90], could be used to guess the $\omega$-proof from the individual proof instances. In general, the complexity of the algorithm needed to guess an $\omega$-proof from non-uniformly generated examples is exponential, whereas the stages of checking the $\omega$-proof and suggesting a cut formula are less complex, and this is reflected in the time taken to produce the result. As an alternative, the user may bypass this whole stage by specifying the $\omega$-proof directly. Meta-induction is used 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. Note that such inductive inference algorithms for generating generalisation produce a proof for an arbitrary instance: the penultimate section suggests how this can relate to finding the proper induction formulae for inductive theorem provers.

Such $\omega$-proofs are not simply disguised $PA$-proofs, since the system $PA_{c\omega}$ is a logically stronger system than that of $PA$ [Shoenfield 59]. Moreover, these $\omega$-proofs may be considered to be more intuitive than standard inductive proofs of the same theorems, in the sense of corresponding more closely to the way in which people convince themselves of the correctness of the proof. Philosophical induction (from "trivial" test cases) may sometimes be the means of construction by humans of a generic proof for case $n$. In addition, the generic proof itself might have some psychological validity. For instance, in order to show that $\forall l\ rotate(length(l), l) = l$ where

$$rotate(0, l) \Rightarrow l \text{ and } rotate(s(n), h :: t) \Rightarrow rotate(n, append(T, h :: nil))$$

a human might provide the following explanation

> "Imagine applying the definition of rotate $n$ times. The elements pop off the front of the list in order and stack up at the end in the same order. Eventually you get back to the original expression."

rather than " Suppose it were true for $n$. Now consider the case $n + 1 \ldots$".

For the implementation it is necessary to provide (for the $n$th case) a description for the $\omega$-proof in a constructive 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 $A(x)$ 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 primitive recursive function sought is described by the sequence of rule applications, parameterised 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 processes of generation of a (recursive) $\omega$-proof from individual proof instances, and the (metalevel) checking that this is indeed the correct proof have been automated (see [Baker 92b]). 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 [Baker 92a].

**Axioms**

$$0 + y = y \quad (1)$$
$$s(x) + y = s(x + y) \quad (2)$$

**Proof**

| | |
|---|---|
| | $(\underline{n} + \underline{n}) + \underline{n} = \underline{n} + (\underline{n} + \underline{n})$ |
| $\underline{n} \equiv s^n(0)$ | $(s^n(0) + s^n(0)) + s^n(0) = s^n(0) + (s^n(0) + s^n(0))$ |
| (2) $n$ TIMES ON LEFT | $s^n(0 + s^n(0)) + s^n(0) = s^n(0) + (s^n(0) + s^n(0))$ |
| (1) ON LEFT | $s^n(s^n(0)) + s^n(0) = s^n(0) + (s^n(0) + s^n(0))$ |
| (2) $n$ TIMES ON RIGHT | $s^n(s^n(0)) + s^n(0) = s^n(0 + (s^n(0) + s^n(0)))$ |
| (1) ON RIGHT | $s^n(s^n(0)) + s^n(0) = s^n(s^n(0) + s^n(0))$ |
| (2) $n$ TIMES ON LEFT | $s^n(s^n(0) + s^n(0)) = s^n(s^n(0) + s^n(0))$ |
| | EQUALITY |

Figure 1: An $\omega$-Proof of $(x + x) + x = x + (x + x)$

Thus, the $\omega$-proof representation represents $P(n)$, the proof of the $n$th numerator of the constructive $\omega$-rule, in terms of rewrite rules applied $f(n)$ or a constant number of times to formulae (dependent upon the parameter $n$). As an example, the implementational representation of the $\omega$-proof for $(x + x) + x = x + (x + x)$ takes the form given in Figure 1 (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 1.

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 $\omega$-proof, under the appropriate instantiation of variables, with the aim of reducing both sides of the equation to the same formula. The $\omega$-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. We now describe in more detail how this is done.

# 5 Showing Correctness of $\omega$-Proofs.

By an effective proof-tree, as discussed in Section 3, we understand a function which returns for each potential position in the tree either a pair representing the sequent and rule associated with that position, or a token indicating that the position is outside the tree. Positions are given by a list of positive integers referring to the path through the tree from the root. We thus have a partial function

$$tree : list(int) \rightarrow sequent \times rule$$

where we must define *rule*, *sequent*.

## 5.1 Schematic syntax

The representation of sequents as a function of position is achieved as follows. For simplicity, we will concentrate on the single goal formula of a sequent, though the discussion extends to the full sequent.

We consider formulae simply as strings[6]. A function

$$form : int \to string$$

can be considered as representing formulae schematically, provided that the function always takes values among the formulae of the language.

For example, using ML syntax we can define the following functions (where ^ is infix string concatenation; `nth_suc` implements $s^n(t)$ above):

```
val zero = "0"
fun plus x y = "plus(" ^ x ^ "," ^ y ^ ")"
fun nth_suc 0 x = x
  | nth_suc n x = "s(" ^ (nth_suc (n-1) x) ^ ")"
fun eq x y = x ^ " = " ^ y
fun goal n =
    let val m = nth_suc n zero
    in
        eq (plus (plus m m) m) (plus m (plus m m))
    end;
```

The function `goal` here returns a string representing the $n$th subgoal to the $\omega$-rule application in our example above:

```
- goal 0;
"plus(plus(0,0),0) = plus(0,plus(0,0))" : string
- goal 1;
"plus(plus(s(0),s(0)),s(0)) = plus(s(0),plus(s(0),s(0)))" : string
```

In this way a proof-tree can be built up by assigning such formulae to positions. There are constraints on which positions have attached formulae, to ensure that the tree is indeed a tree, and is well-founded. The *rule* specification is likewise given by a string, corresponding to the name of the rule applied at this position, and any parameters used. The function gives an $\omega$-proof if, at every node of the tree, the formulae at the sub-nodes and the formula at the node are correctly related according to the rule of inference associated with the node.

It would be extremely cumbersome to build up such tree functions explicitly, and we do not do this in practice. However, the system is intended to ensure that such a tree is constructible whenever an $\omega$-rule application is shown correct.

[6] Alternatively, and more elegantly, we could have made use of abstract syntax here.

## 5.2 Rewrite rules as derived rules of inference

It is customary to use recursion equations as rewrite rules in order to evaluate expressions. Since we have a substitution rule, we can show that such applications to quantifier-free formulae are sound. For example, the axiom (2) gives us the rewrite

$$s(X) + Y \Rightarrow s(X + Y). \tag{3}$$

The rewrite rule of inference then, given a goal and a specified sub-term that matches the left hand side of the rewrite, yields as the single subgoal the rewritten goal. Note that any use of this derived rule could be expanded into a small proof tree of fixed shape in our original theory.

In reasoning about $\omega$-proofs, we will chiefly use this derived rule.

## 5.3 Multiple rule applications

In the simpler examples of the use of the $\omega$-rule, there is only one application of that rule, and by using the rewrite inference rule we can regard each subsequent branch of the proof as linear, with no further branching of the tree.

In this case, what we want to know is that the sequence of rewrites along the branch is correct, and leads to an axiom. To capture the generality of the shape of the tree, we need to be able to reason about such things as the application of a given rewrite $n$ times in a particular position of the $n$th formula. (More generally, the position and the number of applications can both be functions of $n$.)

Supposing that we have defined what it is for one-step rewriting to apply using a given rule at a given position, we can define $n$-fold rewriting by:

```
fun nfoldRewrite 0 rule pos wff = wff
  | nfoldRewrite n rule pos wff =
      rewrite rule pos (nfoldRewrite (n-1) rule pos)
```

Notice that our definitions of schematic syntax and multiple rewriting are primitive recursive. In order to prove properties of them, we can use standard techniques for proof of primitive recursively defined functions. For example, as part of the correctness proof for our example, we want to show that $n$ applications of (3) on the left hand side of

$$(s^n(0) + s^n(0)) + s^n(0) = s^n(0) + (s^n(0) + s^n(0))$$

yields

$$s^n(0 + s^n(0)) + s^n(0) = s^n(0) + (s^n(0) + s^n(0)).$$

Writing the second expression as a function of $n$ as `goal2(n)`, calling the rewrite rule `plus2`, and noting that the left hand side is picked out by position `[2]`, what has to be shown is that

```
nfoldRewrite n plus2 [2] (goal n) = goal2 n.
```

The proof of this proceeds by induction; show that

```
nfoldRewrite 0 plus2 [2] (goal 0) = goal2 0
```

and that

```
nfoldRewrite n plus2 [2] (goal n) = goal2 n
   ⊢  nfoldRewrite (n+1) plus2 [2] (goal (n+1)) = goal2 (n+1)
```

This is the proof that is carried out automatically. When each of the rewrite steps has been verified in this way, and the leaf checked as an axiom, we conclude that there is a correct $\omega$-proof.

Next we move on to consider further the relationship between the $\omega$-proof representation and the system $PA_{c\omega}$.

## 5.4 Relationship between $\omega$-proof representation and $PA_{c\omega}$

In the representation of $PA_{c\omega}$, a derived form of primitive recursive function-defining equations as inference rules is required to allow rewriting of formulae (which takes place in $\omega$-proof examples, and hence an analogy is needed). These are of the type $\mathcal{I}$ below — it is just convenient to define these as single steps, as shorthand in order to avoid going through all the equivalent tedious steps of $PA$.

$$\frac{\Gamma \vdash \forall xy R(x,y) = S(x,y) \quad \Delta \vdash T(R(p,q))}{\Gamma,\ \Delta \vdash T(S(p,q))}\mathcal{I}$$

An example of such a derived step would be the justification of $s(0+0) = s(0)$ from $s(0) + 0 = s(0)$, given rewrite rule (3) above. $\mathcal{I}$ is a derived rule of $PA$, as it is equivalent to the following:

$$\dfrac{\dfrac{\dfrac{}{R(p,q) = S(p,q) \vdash R(p,q) = S(p,q)}\,axiom}{\forall xy\ R(x,y) = S(x,y) \vdash R(p,q) = S(p,q)}\,\forall l(\times 2) \qquad \Gamma \vdash \forall xy\ R(x,y) = S(x,y)}{\Gamma \vdash R(p,q) = S(p,q)}\,cut \qquad \Delta \vdash T(R(p,q))}{\Gamma,\ \Delta \vdash T(S(p,q))}\,subst$$

One may deduce a more general rule $\mathcal{I}'$, in a similar manner, where $\mathcal{I}'$ is the following:

$$\frac{\Gamma \vdash \forall \vec{x} R(\vec{x}) = S(\vec{x}) \quad \Delta \vdash T(R(\vec{p}))}{\Gamma,\ \Delta \vdash T(S(\vec{p}))}\mathcal{I}'$$

where if the arity of $\vec{x}$ is $k$, then $\vec{p} = \{p_1, \ldots p_k\}$ for terms $p_i$ $(1 \leq i \leq k)$ in the language of $PA_{c\omega}$.

The derived rule $\mathcal{I}$ above corresponds to "apply $\mathcal{R}$ once", where $\mathcal{R}$ is the rewrite rule $R(x,y) \Rightarrow S(x,y)$. "Apply $\mathcal{R}$ a constant number of times, $k$" just repeats $\mathcal{I}$ $k$ times. $\mathcal{I}$ corresponds to the basic formulation of $PA_{c\omega}$, so if the given procedure about tree construction from rules is followed, a recursive prooftree will be obtained. Note that in the tree there is infinite branching, so dummy subtrees will have to be inserted if necessary.

In the $\omega$-proof, one starts with something to be proved, and ends with equality, corresponding to an axiom. In practice, only rewrite rules are usually used, but it is possible to use logical rules which might cause the $\omega$-proof to split. However, such case splits of conditionals do not pose any problem, because they correspond exactly to the logical rules of $PA_{c\omega}$.

Note that if such derived rules were to be used in the prooftree representation, the whole equivalent subtree should be substituted instead, since the new node position otherwise would not be a direct subnode (in terms of its position representation) of the original node at which such a rule was applied.

The other case to consider is when a rewrite rule of the form "apply $\mathcal{R}$ some function of $n$ times" is applied in the $\omega$-proof. This corresponds to the (derived) rule $\mathcal{J}$ below, which may be proven in $PA_{c\omega}$, with the use of induction.

$$\frac{\Gamma \vdash \forall x \forall y R(x,y) = S(x,y) \quad \Delta \vdash T(R^{f(n)}(p,q))}{\Gamma, \Delta \vdash T(S^{f(n)}(p,q))}\,\mathcal{J}$$

where $S^{f(n)}(p,q)$ is the result of applying the rule $\mathcal{R}$ $f(n)$ times to $R^{f(n)}(p,q)$ (assuming that such a rule application may be carried out). One such example is that $s^n(x) + y \Rightarrow s^n(x+y)$ corresponds to $n$ applications of the rewrite rule $s(x) + y \Rightarrow s(x+y)$. It is also possible to derive a more general version $\mathcal{J}'$, which is analogous to $\mathcal{I}'$.

The proof in $PA_{c\omega}$ of $\mathcal{J}$ is as follows:

$$\dfrac{\Delta \vdash T(R^n(p,q)) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{R(R^r(p,q)) = S(R^r(p,q)) \vdash R(R^r(p,q)) = S(R^r(p,q))}\,ax}{\vdash R(R^r(p,q)) = S(R^r(p,q))}\,\forall l \qquad \Gamma \vdash \forall x y R(x,y) = S(x,y)}{\Gamma \vdash R(R^r(p,q)) = S(R^r(p,q))}\,cut \qquad \dfrac{}{T(S(R^r(p,q))) \vdash T(S(R^r(p,q)))}\,ax}{\vdash T(S(R^r(p,q))) \to T(S(R^r(p,q)))}\,\mathcal{I}}{T(R^r(p,q)) \to T(S^r(p,q)) \vdash T(R^{s(r)}(p,q)) \to T(S^{s(r)}(p,q))}\,\mathcal{I} \quad \star}{\Gamma \vdash T(R^n(p,q)) \to T(S^n(p,q))}\,ind}{\Gamma, T(R^n(p,q)) \vdash T(S^n(p,q))}\,\to r}{\Gamma, \Delta \vdash T(S^n(p,q))}\,cut$$

At $\star$ there is the prooftree:

$$\dfrac{\dfrac{}{R(p,q) = S(p,q), T(R(p,q)) \vdash T(S(p,q))}\,ax \qquad \dfrac{\dfrac{}{\Gamma \vdash \forall x y R(x,y) = S(x,y)}\,ax \qquad \dfrac{\dfrac{}{R(p,q) = S(p,q) \vdash R(p,q) = S(p,q)}\,ax}{\forall x y R(x,y) = S(x,y) \vdash R(p,q) = S(p,q)}\,\forall l}{\Gamma \vdash R(p,q) = S(p,q)}\,cut}{\Gamma \vdash T(R(p,q)) \to T(S(p,q))}\,cut$$

A summary of the above consideration of the relationship of the $\omega$-proof representation with $PA_{c\omega}$ is that the proof of $A(\underline{n})$ in the $\omega$-rule $\frac{A(0),\ldots A(\underline{n})\ldots}{\forall x A(x)}$ is represented

in the $\omega$-proof using notation which is not that of $PA_{c\omega}$, but which is more suitable for implementation. However, the notation (represented by derived rules $\mathcal{I}$ and $\mathcal{J}$), may be converted to that of $PA_{c\omega}$, as may be seen by the fact that the rules $\mathcal{I}$ and $\mathcal{J}$ correspond to proofs in $PA_{c\omega}$. Hence, the $\omega$-proof can be accounted for in terms of $PA_{c\omega}$.

## 5.5 Correctness of the $\omega$-proof representation

Global correctness necessitates showing that for each $\omega$-proof construct there is a primitive recursive function which indicates what is at any particular node in the tree representation described above. This is possible by inspecting the tree diagrams for the derived inference rules $\mathcal{I}$ and $\mathcal{J}$ above. The primitive recursive function would be analogous to the definition of $f$ above. When "apply rule $\mathcal{R}$ $k$ times" appears in the $\omega$-proof, the function would generate the tree as previously described, but using the rules from $\mathcal{I}$ (repeated as appropriate), or $\mathcal{J}$, and generating the rest of each infinite layer as dummy variables. Of course, the layer is not literally filled in, as this will be a non-terminating process. It is merely necessary to note that, for example, $f(Pos <> [l]) = \breve{\in} \forall l \in nat\ list$, and that any particular individual position could be checked as yielding $\breve{\in}$. This is enough to give correctness of the tree. A tree with infinite branching points may be generated using the constructive $\omega$-rule (ie. from the $\omega$-proof). This should be using a depth-first generation, because the tree is required to be well-founded; in this way the tree would in essence be completed — after a certain point with generating the subgoals of the $\omega$-rule, the case for the $k$th subtree could be given if termination was required.

Therefore, in order to convert from $\omega$-proofs to a recursive prooftree form, it is necessary to substitute $k$ for $\underline{n}$, where $k \in \mathbb{N}$, to get the $k$th subtree. Rewrite rules should be converted to the appropriate rules of $PA_{c\omega}$ (from the trees given above for $\mathcal{I}$ and $\mathcal{J}$) and applied as appropriate; this also covers the case of the application of rules $f(n)$ times.

In this section a justification for the implemented representation of $\omega$-proofs has been given. The following section describes the overall structure of the implemented system.

# 6 A Proof Environment for the Constructive Omega Rule

This section describes the Constructive Omega Rule Environment (CORE), which is a proof development environment in which a (constructive) version of the $\omega$-rule may be used as a rule of inference, and a system in which $\omega$-proofs may be displayed and investigated. The implementation allows both the automatic or incremental construction of $\omega$-proofs, and the validations of descriptions of $\omega$-proofs. It is carried out within the framework of an interactive theorem-prover with Prolog as the tactic language, namely Oyster, which is a reimplementation of NuPRL [Bundy *et al* 90]. This embodies a higher-order, typed constructive logic in sequent-calculus form.

| Theorem | Cut Formula |
|---|---|
| $\forall x\ (x+x)+x = x+(x+x)$ | $\forall x \forall y \forall z\ (x+y)+z = x+(y+z)$ |
| $\forall x\ x+s(x) = s(x+x)$ | $\forall x \forall y\ x+s(y) = s(x+y)$ |
| $\forall x\ x+s(x) = s(x)+x$ | $\forall x \forall y\ x+s(y) = s(x)+y$ |
| $\forall x\ x.(x+x) = x.x+x.x$ | $\forall x \forall y \forall z\ x.(y+z) = x.y+x.z$ |
| $\forall x\ (x+x).x = x.x+x.x$ | $\forall x \forall y \forall z\ (x+y).z = y.z+x.z$ |
| $\forall x\ (2+x)+x = 2+(x+x)$ | $\forall x \forall y\ (2+x)+y = 2+(x+y)$ |
| $\forall x\ \forall y\ (x+y)+x = x+(y+x)$ | $\forall x \forall y \forall z\ (x+y)+z = x+(y+z)$ |
| $\forall x\ x \neq 0 \rightarrow p(x)+s(s(x)) = s(x)+x$ | $\forall x \forall y\ x \neq 0 \rightarrow p(x)+s(s(y)) = s(x)+y$ |
| $\forall x\ even(x+x)$ | $\forall x\ even(2.x)$ |
| $\forall l\ len(rev(l)) = len(l)$ | $\forall l\ len(rev(l) <> a) = len(rev(a) <> l)$ |
| $\forall l\ rotate(len(l), l) = l$ | $\forall l\ rotate(len(l), l <> a) = a <> l$ |
| $\forall l\ rev(rev(l) <> y :: nil) = y :: rev(rev(l))$ | $\forall l\ rev(a <> y :: nil) = y :: rev(a)$ |
| $\forall l\ rev2(l, nil) = rev(l)$ | $\forall l\ rev2(l, a) = rev(l) <> a$ |
| $\forall l\ (l <> l) <> l = l <> (l <> l)$ | $\forall l \forall p \forall q\ (l <> p) <> q = l <> (p <> q)$ |

Table 1: Cut Formulae Suggested by Guiding Method for Various Examples

Within the Oyster framework, the object-level logic is replaced with Peano and Heyting arithmetic and the rules that can be applied are those of the sequent calculus axiomatisation of first order logic given in [Dummett 77, P133], together with mathematical induction. The search for a proof must be guided either by a human user or by a proof tactic. Each proof is built up in the form of a tree, and every stage of the tree may be displayed on the screen with information as to the hypotheses, goals, position in the tree and whether the subtree is proved below it.

Within CORE, any finitely large number of individual instances of proofs of a proposition may be generated automatically by the use of various tactics. The general representation of the proofs is provided by an inductive inference algorithm, which starts with an initial generalisation and then works by updating this $\omega$-proof using the other individual proofs, until the $\omega$-proof seems to have reached a stable form. This $\omega$-proof is then automatically checked to see if it is indeed the correct one, as described above. There are two options which are allowable from a goal $\Gamma \vdash \forall x P(x)$. One is to ask to use the constructive $\omega$-rule, whereby the system will check to see whether it can find a correct $\omega$-proof, and then return to the former system and close the branch, or else report failure. The user may then continue to investigate other positions in the proof-tree. The other option, which shall be discussed more fully in the following section, is to ask for an appropriate cut to be carried out in $PA$ (the cut being worked out by the system from the $\omega$-proof), with a further option to complete the tree as far as possible (using standard theorem-proving techniques). The $\omega$-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 $\omega$-proof. The user may specify the proof incrementally, in terms of 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 $\omega$-proof, and is able to display individual proofs in addition to the general case.

[Baker 92b] provides details of the proof development systems upon which the implementation is based; representation of the $\omega$-rule and its subgoals; generation of individual proofs; the application of rewrite rules; provision of a $\omega$-proof; correctness checking of the $\omega$-proof (using meta-induction); generalisation, and finally, the interactive system, and how to use it.

# 7 An Application

As mentioned above, the CORE system provides implementation of a new generalisation method (described in [Baker 94]). A cut formula is automatically suggested from $\omega$-proofs using an implementation based on the method of explanation-based generalisation, which is a technique for formulating general concepts on the basis of specific training examples, first described in [Mitchell 82]. In general terms 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. The method is applied in this instance to a new domain, namely that of $\omega$-proofs. It is of course possible to carry out explanation-based generalisation upon proofs in $PA$ in order to produce a more general expression. However, if inductive proof is not possible (because induction is blocked) and therefore use of the cut rule is required, such a method will not work, and another proof (such as the $\omega$-proof) must be used in order to obtain a generalisation. Further details regarding such a generalisation method, including details of how $\omega$-proofs may be "linearised" to suggest inductive proofs, are given in [Baker 94].

The resulting system has been tested on a variety of arithmetical examples: cut formulae are automatically suggested for examples including all the arithmetical examples of Table 1. Although the examples listed in the table are of a similar simple form, this method may also be applied to complicated examples containing nested quantifiers, etc., for the $\omega$-rule applies to arbitrary sequents. The seventh example provides an instance of nested use of the $\omega$-rule, which carries through directly. If an $\omega$-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. For example, the cut formula of $even(2.x)$ could possibly be extracted by a user from the form of the $\omega$-proof for $even(x + x)$, which is an improvement over other generalisation methods. Thus the generalisation method may still be useful within a co-operative environment if it breaks down. 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.

Hence a new method for generalisation has been proposed which 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 (cf. Table 1, many of the examples in which pose a problem for other theorem provers). This same generalisation method can be used in a more general context for lemma generation, regardless of the way an $\omega$-proof was obtained, so long as one can represent in the particular system of interest the notions of $n$th-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 $\omega$-rule, in practice suitable proof environments such as HOL [Gordon 88] would require extension, and moreover the user would have to input the proof directly unless some other method of generation could be provided.

# 8 Conclusions

Implementation of a system of arithmetic with the $\omega$-rule has been carried out within the framework of an interactive theorem-prover with Prolog as the tactic language. This can provide a useful aid to automated deduction. The approach suggested in this paper is of general relevance, both regarding lemma generation in systems for reasoning about inductive domains, and also the representation of infinite and schematic proofs.

**Acknowledgements** I would like to acknowledge the help of Alan Smaill from the Mathematical Reasoning Group in Edinburgh University.

# References

[Baker 92a] S. Baker. *Aspects of the Constructive Omega Rule within Automated Deduction.* PhD thesis, University of Edinburgh, 1992.

[Baker 92b] S. Baker. CORE manual. Technical Paper 10, Dept. of Artificial Intelligence, Edinburgh, 1992.

[Baker 94] S. Baker. A new application for explanation-based generalisation within automated deduction. In A. Bundy, editor, *12th International Conference on Automated Deduction*, pages 177–191, Springer-Verlag, 1994. Also available from Cambridge as Computer Laboratory Technical Report 327.

[Bundy *et al* 90] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag, 1990. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[Bundy *et al* 93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*,

62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[Dummett 77] M. Dummett. *Elements of Intuitionism.* Oxford Logic Guides. Oxford Univ. Press, Oxford, 1977.

[Feferman 62] S. Feferman. Transfinite recursive progressions of axiomatic theories. *Journal of Symbolic Logic*, 27:259–316, 1962.

[Gordon 88] M. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P.A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis.* Kluwer, 1988.

[Kreisel 65] G. Kreisel. Mathematical logic. In T.L. Saaty, editor, *Lectures on Modern Mathematics*, volume III, pages 95–195. John Wiley and Sons, 1965.

[Löpez-Escobar 76] E.G.K. Löpez-Escobar. On an extremely restricted $\omega$-rule. *Fundamenta Mathematicae*, 90:159–72, 1976.

[Mitchell 82] T.M. Mitchell. Toward combining empirical and analytical methods for inferring heuristics. Technical Report LCSR-TR-27, Laboratory for Computer Science Research, Rutgers University, 1982.

[Nelson 71] G.C. Nelson. A further restricted $\omega$-rule. *Colloquium Mathematicum*, 23, 1971.

[Plotkin 69] G. Plotkin. A note on inductive generalization. In D Michie and B Meltzer, editors, *Machine Intelligence 5*, pages 153–164. Edinburgh University Press, 1969.

[Prawitz 71] D. Prawitz. Ideas and results in proof theory. In J.E. Fenstad, editor, *Studies in Logic and the Foundations of Mathematics: Proceedings of the Second Scandinavian Logic Symposium*, volume 63, pages 235–307. North Holland, 1971.

[Rosser 37] B. Rosser. Gödel-theorems for non-constructive logics. *JSL*, 2(3):129–137, September 1937.

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

[Schütte 77] K. Schütte. *Proof Theory.* Springer-Verlag, 1977.

[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 $\omega$-rule. *Bull. Acad. Sc. Polon. Sci., Ser. des sc. math., astr. et phys.*, 7:405–7, 1959.

[Takeuti 87] G. Takeuti. *Proof theory.* North-Holland, 2 edition, 1987.

[Tucker *et al* 90] J.V. Tucker, S.S. Wainer, and J.I. Zucker. Provable computable functions on abstract-data-types. In M.S. Paterson, editor, *Automata, Languages and Programming*, pages 660–673. Springer-Verlag, 1990. Lecture Notes in Computer Science, vol 443.

[Yoccoz 89] S. Yoccoz. Constructive aspects of the omega-rule: Application to proof systems in computer science and algorithmic logic. *Lecture Notes in Computer Science*, 379:553–565, 1989.