# The Use of Proof Plans to Sum Series *

Toby Walsh Alex Nunes Alan Bundy

Department of AI, Edinburgh University

**Abstract**

We describe a program for finding closed form solutions to finite sums. The program was built to test the applicability of the *proof planning* search control technique in a domain of mathematics outwith induction. This experiment was successful. The series summing program extends previous work in this area and was built in a short time just by providing new series summing methods to our existing inductive theorem proving system CLAM.

One surprising discovery was the usefulness of the *ripple* tactic in summing series. Rippling is the key tactic for controlling inductive proofs, and was previously thought to be specialised to such proofs. However, it turns out to be the key sub-tactic used by all the main tactics for summing series. The only change required was that it had to be supplemented by a *difference matching* algorithm to set up some initial meta-level annotations to guide the rippling process. In inductive proofs these annotations are provided by the application of mathematical induction. This evidence suggests that rippling, supplemented by difference matching, will find wide application in controlling mathematical proofs.

## 1 Introduction

In [2] we introduced *proof planning*, a new technique for controlling the search for a proof by using the common structure of a family of similar proofs as a guide. The application of proof planning to the control of inductive proofs is described in [3] whilst *rippling*, a key tactic in inductive proofs, is described in [4]. Proof planning has been implemented in the Oyster/CLAM system [3].

This paper explores the usefulness of proof planning, in general, and rippling, in particular, in a non-inductive domain: the discovery of closed form solutions to finite series. We describe several methods for summing series and show how they can be represented in the proof plans formalism. They have all been implemented in the Oyster/CLAM system and tested on a wide range of series problems. Most of these methods make use of rippling as a key submethod. In order to use rippling in non-inductive domains it is necessary to supplement it with a special matching algorithm called *difference matching* (see [5] in this volume for details).

The mathematical problem we address is to derive closed form solutions for finite

*The research reported in this paper was supported by SERC grant GR/F/71799, a SERC PostDoctoral Fellowship to the first author and a SERC Senior Fellowship to the third author. We would like to thank the other members of the mathematical reasoning group for their feedback on this project.

series like:

$$\sum_{i=0}^{n} s(i).a^i$$

where $a$ is a constant and $s$ is the successor function, *i.e.* $s(i) = i + 1$. By 'derive a closed form solution' we mean find an expression, equal to the finite sum, that is free of the summation operator. In this example, for $a \neq 1$, such a closed form solution would be:

$$\frac{s(n).a^{s(n)}}{a-1} - \frac{a^{s(n)} - 1}{(a-1)^2}$$

There has been limited research into this domain. Some researchers have tackled the topic as a verification problem, [9, 7]; both these teams use mathematical induction to prove that a series is equal to a user supplied closed form. In the work reported below, the closed form solution to a series is simultaneously synthesised and verified. None of our solution methods uses induction for either the synthesis or the verification task.

Another approach is to use a decision procedure, like Gosper's algorithm, [8], to compute closed form solutions. Such decision procedures have the drawback of being "black-boxes" of only being applicable to a narrow class of series. The work reported here is applicable to a much wider class of series and the solutions produced can be understood by mathematicians. Indeed our solution methods are modelled on those used by mathematicians.

# 2 Proof Planning

A brief description of the ideas and concepts involved in proof planning follows in order to set the background for what is to come. A more detailed account is given in [2, 3].

The notion of explicit *proof plans* as a technique for guiding an automatic theorem prover in its search for a proof by mathematical induction originated in a project to develop automated search in the Oyster program synthesis system, a re-implementation in Prolog of the Cornell Nuprl system, [6]. Oyster is an interactive proof editor for a logic based on Martin-Löf's Intuitionistic Type Theory. Following LCF, the search for a proof in Oyster can be guided by programs called *tactics*. Oyster's tactics for inductive proofs are written in Prolog; they are based on and extend ideas in the Boyer-Moore theorem prover, Nqthm, [1].

To control the application of tactics, the CLAM plan formation program analyses the current theorem and constructs a special-purpose super-tactic to prove it. To enable CLAM to do this, every tactic is (partially) specified by giving preconditions for its attempted application and some of the effects of its successful application. This partial specification is called a *method*. Methods are described in a *meta-logic*, whose domain of discourse is mathematical expressions and which describes syntactic properties of these expressions, *e.g.* the number and location of particular subexpressions.

# 3 The Ripple Tactic

Since the *ripple* tactic plays a key role both in inductive proofs and in summing series, we will illustrate the proof planning technique by describing it. This description will be necessarily brief and superficial. For more details see [4].

Rippling is used during the step case of an inductive proof. Its job is to rewrite the induction conclusion into a form in which it contains one or more subexpressions which match the induction hypothesis. This enables the next tactic, *fertilize*, to use the induction hypothesis to prove the induction conclusion.

To visualise how rippling works consider the following analogy. Some mountains are reflected in a loch[1] in the valley below them. Someone throws a stone into the loch, disturbing the reflection. The waves from the impact ripple outwards to the shore of the loch leaving the reflection undisturbed again. The mountains are the induction hypothesis, the reflection is the induction conclusion and the wave-fronts are those parts of the induction conclusion which differ from the induction hypothesis. The rippling is the selective application of rewrite rules of a suitable form to move the wave-fronts out of the way.

Induction conclusions are necessarily similar to their induction hypotheses except for the addition of some subexpressions, called *wave-fronts*, which are provided by the form of induction rule used. For instance, in the standard inductive proof of the associativity of +, the induction hypothesis is:

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

and the induction conclusion is:

$$\boxed{s(\underline{x})} + (y + z) \quad = \quad (\boxed{s(\underline{x})} + y) + z \tag{1}$$

The wave-fronts are those subexpressions enclosed in boxes less the subexpressions that are underlined. In general, wave-fronts are terms with one or more *wave-holes* in them. The part of the induction conclusion that is similar to the induction hypothesis is called the *skeleton*.

To move these wave fronts outwards the ripple tactic applies *wave-rules*. These are rewrite rules with the property that the left and right hand sides are identical except for the addition of different wave fronts on each side. Furthermore, more of the skeleton is in the wave-hole(s) on the right hand side than it is on the left hand side. The wave-front on the right hand side can be empty. For instance, the recursive definition of + and the substitution law for $s$ provide two wave-rules.

$$\boxed{s(\underline{U})} + V \quad \Rightarrow \quad \boxed{s(\underline{U + V})} \tag{2}$$

$$\boxed{s(\underline{U})} = \boxed{s(\underline{V})} \quad \Rightarrow \quad U = V \tag{3}$$

Wave-rule (2) can be applied to each side of the induction conclusion (1). This causes both the wave-fronts to ripple outwards.

$$\boxed{s(\underline{x + (y + z)})} = \boxed{s(\underline{(x + y)})} + z,$$

Wave-rule (2) can be applied again to the right hand side of the equality to produce:

$$\boxed{s(\underline{x + (y + z)})} = \boxed{s(\underline{(x + y) + z})}.$$

At this point, the two wave-fronts can be eliminated by applying wave-rule (3) to give:

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

[1] The Scottish word for lake.

which is identical to the induction hypothesis. The *fertilize* tactic is then used to prove (trivially) the induction conclusion from the induction hypothesis. The proof of the step case is then complete.

The reasons for the success of rippling are:

- It involves little or no search since the wave-fronts in the goal must correspond to wave-fronts in the wave-rule. The consequence of this is a very controlled application of rewrite rules which in practice means very low branching rates, typically one choice or none at all.
- It terminates. Rippling always makes progress moving wave-fronts in some direction; hence termination is guaranteed, even when applying rewrite rules that would normally, without wave-front annotation, lead to loops.
- It applies only "good" rewrites. As wave-rules are skeleton preserving, if rippling terminates successfully, the hypothesis can be used to prove or simplify the conclusion.

# 4 Difference Matching

A precondition of rippling is that wave-front annotations have been placed in the formula to be rippled. In inductive proofs these are provided by the induction rule in a natural way. In this paper we observe that rippling can be used by a wide variety of theorem proving tactics provided wave-front annotations can be provided. In particular, rippling is useful for rewriting a goal formula so that it contains a subexpression that matches a hypothesis formula. Many proofs have hypotheses and goals with shared structure. We conjecture that rippling will prove useful for putting these goals into a form in which the hypotheses can be used to prove them. This paper provides supportive evidence for our conjecture.

To annotate the goal formula with wave-fronts we use a *difference matcher*. The difference matcher takes the goal, $G$, and hypothesis, $H$, as inputs. It returns $G'$, a copy of $G$ annotated with wave fronts, and substitutions, $\sigma$, such that the skeleton of $G'$ equals $H$ under substitution $\sigma$. Although difference matching generalises first-order matching, it is not just matching. It is an attempt to make two expressions identical by both variable instantiation and structure hiding; the hidden structure is the wave-front. Further details of an algorithm for difference matching can be found in this volume [5].

# 5 Methods for Summing Series

We now explain our methods for summing series. They are called: *standard form*, *perturbate*, *conjugate*, *telescope* and *closed form*. The first four are substantive methods whilst the last is just a simplifier and a checker that the solution is in closed form. Each of the first four method makes significant use of rippling augmented with difference matching.

In the rest of the paper, we will adopt the following conventions: The letters $i$, $j$ and $k$ will be used for indices of summation, *i.e.* bound variables of type natural number. The letters $l$, $m$ and $n$ will stand for constants of type natural number, *i.e.* they will not depend on any indices of summation. These will typically be used for the bounds of summation. The letters $a$, $b$, $c$ and $d$ will stand for constants and variables of type real, *i.e.* they will not depend on any indices of summation. The letters $u$, $v$, $w$, $x$, $y$ and $z$ will stand for terms of type real, *i.e.* they may depend on indices of summation. In $\sum_{i=0}^{n} a.x$, for instance, $x$ may depend on $i$, but $a$ and $n$ do not.

## 5.1 Standard Form

The *standard form* method is the backbone of our methods. It does not find closed form solutions from first principles but tries to reduce the current problem to one which has already been solved. We illustrate this with an example.

Consider the finite sum $\sum_{i=0}^{n} b.i + c$. We can use the *standard form* method to break this into two sub-problems which match previously solved ones, namely the following standard forms:

$$\sum_{i=0}^{n} i = \frac{n.(n-1)}{2} \tag{4}$$

$$\sum_{i=0}^{n} a = s(n).a \tag{5}$$

This will be done by using difference matching to annotate the sum with wave-fronts and then rippling to reduce it to the sub-problems. For the rippling, *standard form* will use the following wave-rules:

$$\sum_{i=m}^{n} \boxed{\underline{x} + \underline{y}} \Rightarrow \boxed{\underline{\sum_{i=m}^{n} x} + \underline{\sum_{i=m}^{n} y}} \tag{6}$$

$$\sum_{i=m}^{n} \boxed{a.\underline{x}} \Rightarrow \boxed{a.\underline{\sum_{i=m}^{n} x}} \tag{7}$$

Note that wave-rule (6) contains two wave-holes, one on the $x$ and one on the $y$. CLAM automatically creates the two weakened versions of this wave-rule which just contain one wave-hole, *eg* :

$$\sum_{i=m}^{n} \boxed{\underline{x} + y} \Rightarrow \boxed{\underline{\sum_{i=m}^{n} x} + \sum_{i=m}^{n} y} \tag{8}$$

Note that wave-rule (7) requires a meta-level condition that $a$ does not contain $i$. Such meta-level conditions are readily handled by the proof planning mechanism.

First $\sum_{i=0}^{n} b.i + c$ is annotated with wave-fronts by difference matching it with the standard form (4). This gives the annotated sum:

$$\sum_{i=0}^{n} \boxed{b.\underline{i} + c}$$

Rippling with wave-rule (8) gives:

$$\boxed{\underline{\sum_{i=0}^{n} \boxed{b.\underline{i}}} + \sum_{i=0}^{n} c}$$

and then with wave-rule (7) gives:

$$\boxed{b.\underline{\sum_{i=0}^{n} i} + \sum_{i=0}^{n} c}$$

This is then be fertilized with standard form (4), to give:

$$b.\frac{n.(n-1)}{2} + \sum_{i=0}^{n} c$$

Since a summation sign is still present, the current problem is difference matched with standard form (5) to give the new annotated sum:

$$\boxed{b.\frac{n.(n-1)}{2} + \underline{\sum_{i=0}^{n} c}}$$

Since this is already fully rippled it is immediately fertilized with (5) to give:

$$b.\frac{n.(n-1)}{2} + s(n).c$$

which is in closed form, as required.

The *standard form* method can be summarised as follows:

- Find a standard form which difference matches with the current problem and add this as a hypothesis.
- Use difference matching to annotate the current problem with wave-fronts.
- Ripple these wave-fronts outwards.
- Fertilize with the hypothesis.

## 5.2 Perturbate

The *perturbate* method's proof strategy has many similarities to induction. From the usual recursive definition of a sum we have the following equation:

$$\sum_{i=m}^{s(n)} u_i = \sum_{i=m}^{n} u_i + u_{s(n)}$$

Alternatively, we can strip off terms from the other end to derive the following equation:

$$\sum_{i=m}^{s(n)} u_i = u_m + \sum_{i=m}^{n} u_{s(i)}$$

Combining these two equations gives:

$$\sum_{i=m}^{n} u_i + u_{s(n)} = u_m + \sum_{i=m}^{n} u_{s(i)} \tag{9}$$

The idea of *perturbate* is to rewrite $\sum_{i=m}^{n} u_{s(i)}$ into a function of $\sum_{i=m}^{n} u_i$, say $f(\sum_{i=m}^{n} u_i)$ using rippling. Therefore all occurrences of $s(i)$ in $\sum_{i=m}^{n} u_{s(i)}$ are annotated with wave-fronts which are then rippled outwards, *i.e.* equation (9) is annotated to:

$$\sum_{i=m}^{n} u_i + u_{s(n)} = u_m + \sum_{i=m}^{n} u_{\boxed{s(\underline{i})}} \tag{10}$$

We will call this equation the *perturbation equation*. The wave-fronts in the perturbation equation are rippled outwards until it is in the form:

$$\sum_{i=m}^{n} u_i + u_{s(n)} \quad = \quad u_m + \boxed{f(\underline{\textstyle\sum_{i=m}^{n} u_i})}$$

This equation is then solved for $\sum_{i=m}^{n} u_i$ using the equation solving tactics of PRESS, [10]. There is a possibility of failure since the unknown, $\sum_{i=m}^{n} u_i$, sometimes cancels out.

To illustrate *perturbate* consider the example sum:

$$\sum_{i=0}^{n} i.a^i$$

Now, by the perturbation equation, (10), we have:

$$\sum_{i=0}^{n} i.a^i + s(n).a^{s(n)} \quad = \quad 0.a^0 + \sum_{i=0}^{n} \boxed{s(\underline{i})}.a^{\boxed{s(\underline{i})}}$$

To ripple this we need the wave-rules (8), (7) and:

$$\boxed{s(\underline{x})}.y \quad \Rightarrow \quad \boxed{\underline{x.y} + y} \tag{11}$$

$$x^{\boxed{s(\underline{y})}} \quad \Rightarrow \quad \boxed{x.\underline{x^y}} \tag{12}$$

Rippling first with wave-rule (11) gives:

$$\sum_{i=0}^{n} i.a^i + s(n).a^{s(n)} \quad = \quad 0.a^0 + \sum_{i=0}^{n} \boxed{\underline{i.a^{\boxed{s(\underline{i})}}} + a^{s(i)}}$$

then with wave-rule (8) gives:

$$\sum_{i=0}^{n} i.a^i + s(n).a^{s(n)} \quad = \quad 0.a^0 + \boxed{\underline{\textstyle\sum_{i=0}^{n} i.a^{\boxed{s(\underline{i})}}} + \textstyle\sum_{i=0}^{n} a^{s(i)}}$$

then with wave-rule (12) gives:

$$\sum_{i=0}^{n} i.a^i + s(n).a^{s(n)} \quad = \quad 0.a^0 + \boxed{\underline{\textstyle\sum_{i=0}^{n} i.\boxed{a.\underline{a^i}}} + \textstyle\sum_{i=0}^{n} a^{s(i)}}$$

and finally with wave-rule (7) gives:

$$\sum_{i=0}^{n} i.a^i + s(n).a^{s(n)} \quad = \quad 0.a^0 + \boxed{a.\underline{\textstyle\sum_{i=0}^{n} i.a^i} + \textstyle\sum_{i=0}^{n} a^{s(i)}}$$

This equation can be solved for $\sum_{i=0}^{n} i.a^i$ using PRESS's methods (provided $a \neq 1$) giving an equation for $\sum_{i=0}^{n} i.a^i$ in terms of $\sum_{i=0}^{n} a^{i+1}$. The standard from method is then called to replace $\sum_{i=0}^{n} a^{i+1}$ by a closed form expression. This gives:

$$\sum_{i=0}^{n} i.a^i \quad = \quad \frac{a^{s(n)} - a.\frac{a^{s(n)}-1}{a-1}}{a-1}$$

The *perturbate* method can be summarised as follows:

- Instantiate the perturbation equation to the current series.
- Ripple the wave fronts on the right hand side of the equation outwards.
- Solve the resulting equation, using PRESS tactics, treating the current series as the unknown.

The *perturbate* method can be generalised so that it uses more complex forms of perturbation equation based on more complex forms of the recursive definition of summation. For instance, it could use the following two step perturbation equation:

$$\sum_{i=m}^{n} u_i + u_{s(n)} + u_{s(s(n))} \quad = \quad u_m + u_{s(m)} + \sum_{i=m}^{n} u_{s(s(i))}$$

This is useful for series like:

$$\sum_{i=0}^{n} (-1)^i . \frac{1}{2^i}$$

Analogously to mutual recursion, we can also perform mutual perturbations. This is useful for series like $\sum_{i=0}^{n} \sin(i.\theta)$. These generalisations have yet to be implemented. However, we do not envisage any significant difficulties in extending *perturbate* in these ways.

## 5.3 Conjugate

The *conjugate* method transforms the finite sum of a term into the finite sum of its conjugate, in the hope that it will be easier to find a closed form solution to the conjugate sum than to the original sum. The conjugate can be one of several second order operations, *e.g.* the differential or integral of the original term, or the mapping of a trigonometric series onto the real or imaginary part of a complex series. Thus the *conjugate* method is a generic one covering a wide range of transformations.

The general idea can be understood as follows. Suppose we want to find a closed form for $\sum u$. Let $F$ be a second-order function with an inverse, $F^{-1}$. That is, there is an equation of the form:

$$F(F^{-1}(v)) \quad = \quad v \tag{13}$$

Let us also assume that there exists a wave-rule which will ripple the function $F$ through the summation operator.

$$\sum \boxed{F(\underline{v})} \quad \Rightarrow \quad \boxed{F(\underline{\sum v})} \tag{14}$$

Thus, combining these two equations we have

$$\begin{aligned}\sum u &= \sum \boxed{F(\underline{F^{-1}(u)})} \\ &= \boxed{F(\underline{\sum F^{-1}(u)})}\end{aligned}$$

This new expression looks syntactically more complicated than the original but often $F^{-1}(u)$ simplifies to some expression $u'$, whereby $\sum u'$ is easier to sum than $\sum u$.

In order to prevent *conjugate* being universally applicable or looping, it is necessary to impose a constraint on it. We have adopted the constraint of a heuristic postcondition that $u'$ must have a lower complexity than $u$. Complexity is measured using a simple Knuth-Bendix term order.

For example, consider the sum mentioned in the introduction,

$$\sum_{i=0}^{n} s(i).a^i$$

where $a \neq 1$. Let $F$ be the differentiation operator and $F^{-1}$ be integration. Now differentiation ripples through summation:

$$\sum \boxed{\frac{d\underline{u}}{dx}} \quad \Rightarrow \quad \boxed{\frac{\underline{d\sum u}}{dx}} \tag{15}$$

And integrating $s(i).a^i$ with respect to the free variable $a$ gives $a^{s(i)}$. Constants of integration can be safely ignored since they will disappear on differentiation. Since $a^{s(i)}$ is simpler than $s(i).a^i$ in our Knuth-Bendix order, *conjugate* can proceed. It rewrites the sum to:

$$\sum_{i=0}^{n} \frac{da^{s(i)}}{da}$$

Wavefront annotations are added by difference matching against the standard result for the sum of a geometric series,

$$\sum_{j=0}^{n} b^j$$

This gives:

$$\sum_{i=0}^{n} \boxed{\frac{d\underline{a^{\boxed{s(\underline{i})}}}}{da}}$$

Rippling first with wave-rule (15) gives:

$$\boxed{\underline{\frac{d}{da}\sum_{i=0}^{n} a^{\boxed{s(\underline{i})}}}}$$

And then with wave-rule (12) gives:

$$\boxed{\frac{d}{da}\underline{\sum_{i=0}^{n} \boxed{a.\underline{a^i}}}}$$

And finally with wave-rule (7) gives:

$$\boxed{\frac{d}{da}(a.\underline{\sum_{i=0}^{n} a^i})}$$

This is then fertilized with the standard form for a geometric series, to give:

$$\frac{d}{da}(a.\frac{a^{s(n)}-1}{a-1})$$

The *closed form* method (described in §5.5) then differentiates this expression giving the final answer:

$$\sum_{i=0}^{n} s(i).a^i \quad = \quad \frac{s(n).a^{s(n)}}{a-1} - \frac{a^{s(n)}-1}{(a-1)^2}$$

The *conjugate* method can be summarised as follows:

- Find a second order operator, $F$, that ripples through summation.
- Apply $F^{-1}$ to the series term, $u$, and simplify the result to $u'$.
- If $u'$ is simpler in the Knuth-Bendix ordering than $u$ then sum the series $\sum u'$ giving an answer $v'$.
- Simplify $F(v')$ and return this as the final result.

A major use of the *conjugate* method is for summing trigonometric series by transforming them into the real or imaginary parts of exponential series. Consider, for example:

$$\sum_{i=0}^{n} \sin(i.\theta)$$

This is solved by *conjugate* by rewriting it into:

$$\sum_{i=0}^{n} Im(e^{\sqrt{-1}.i.\theta})$$

This series can then be summed by difference matching against the standard form for a geometric series and rippling. Other series which can be solved in a similar way include $\sum \cos(i.\theta)$, $\sum \sin(i.\theta).\cos(i.\theta)$, $\sum \sin^2(i.\theta)$ and $\sum \cos^2(i.\theta)$

## 5.4 Telescope

The *telescope* method is based on the idea that if one part of the term in the series can be cancelled against part of the next term then the sum can be collapsed like a folding "telescope" into a hopefully simpler problem. The version of *telescope* described here concentrates on a restriction of this strategy in which consecutive terms of the series cancel each other out totally.

To give a more rigorous description of this technique we introduce the upper difference operator:

$$\triangle\ u_i \quad = \quad u_{i+1} - u_i$$

This operator has the useful property for summing series that:

$$\begin{aligned}
\sum_{i=m}^{n} \triangle\, u_i &= (u_{s(n)} - u_n) + (u_n - u_{n-1}) + \dots \\
& \quad \dots + (u_{m+2} - u_{s(m)}) + (u_{s(m)} - u_m) \\
&= u_{s(n)} - u_m \qquad (16)
\end{aligned}$$

We call this the *telescope equation*. It can be used by *telescope* provided that the terms, $v_i$, of the series being summed, $\sum v_i$, can be rewritten into the form of an upper difference, $\triangle\, u_i$. The telescope equation, (16), can then be used to reduce the series to $u_{s(n)} - u_m$. At the moment, upper differences are supplied by the user. Recently, however, we have proposed a higher order procedure for discovering upper differences automatically.

To illustrate the *telescope* method consider:

$$\sum_{i=0}^{n} \binom{i}{m}$$

Where:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Using the identity:

$$\binom{s(n)}{s(m)} = \binom{n}{m} + \binom{n}{s(m)}$$

We get:

$$\begin{aligned}
\binom{i}{m} &= \binom{i+1}{s(m)} - \binom{i}{s(m)} \\
&= \triangle \binom{i}{s(m)}
\end{aligned}$$

The series is therefore rewritten by *telescope* into the sum of an upper difference:

$$\sum_{i=0}^{n} \triangle \binom{i}{s(m)}$$

Using (16) as a standard form this is rewritten by the *standard form* method into:

$$\binom{s(n)}{s(m)} - \binom{0}{s(m)}$$

The *telescope* method can be summarised as follows:

- Express the series term as an upper difference.
- Instantiate the telescope equation with this upper difference version of the series.

The *telescope* method can sum a wide variety of series including any series like $\sum i^3$ which is polynomial in the index of summation.

### 5.5 Closed Form

The *closed form* method terminates all our proof plans by checking that any solutions derived are in closed form. It uses the following definition of closed formedness:

> **Definition 1 (Closed formedness)** : *An expression, exp is a closed form iff it is of the general form:*

$$
\begin{aligned}
exp &:= constant \mid var \mid s(exp) \mid exp + exp \mid exp - exp \mid -exp \mid \\
& \quad exp.exp \mid \frac{exp}{exp} \mid exp^{exp} \mid \ln(exp) \mid \log_{exp}(exp) \mid \\
& \quad \sin(exp) \mid \cos(exp) \mid if(test, exp, exp) \\
constant &:= 0 \mid e \\
var &:= universally\ quantified\ variables \\
test &:= exp > exp \mid exp < exp \mid exp = exp \mid exp \geq exp \mid exp \leq exp
\end{aligned}
$$

This definition could be easily extended to include a larger set of constants, functions and tests (one obvious extension would be the factorial if one were to reason with products). Its most significant feature is what it leaves out, *i.e.* summation operators, but also the differential, integral, real and imaginary operators.

Before checking that solutions are in closed form, *closed form* simplifies the solution. This has the effect of eliminating any functions that lie outside the closed form grammar and which can be simply eliminated by evaluation. Note that the check for closed formedness is essentially a meta-level operation, *i.e.* it is couched in terms of the syntax of the expression rather than its semantics. This is easily handled by the meta-logical language of the proof plan methods. It shows that some kind of meta-level reasoning is essential in this domain.

## 6 Implementation and Results

The five series summing methods described above have been implemented as methods in the CLAM system and tested successfully on a range of examples.

The problem of summing a series is represented as a logical theorem, *i.e.* to find a closed form for the series $\sum_{i=m}^{n} u_i$ we get CLAM to plan the proof of the theorem:

$$\forall m{:}nat\,.\,\forall n{:}nat\,.\,\exists S.\, S = \sum_{i=m}^{n} u_i \tag{17}$$

As yet, we have not written the tactics necessary to execute the plans in Oyster; that is, we only build plans and not their corresponding object-level proofs. However, since the preconditions to our methods are *complete* specifications of the methods' applicability, the successful execution of any plan is guaranteed. Indeed, the mapping from plans to their corresponding proofs is purely mechanical.

For reasons of simplicity, the summation operator is represented in the meta-level using a *pseudo* first order term, $sum(i, 0, n, u_i)$. All manipulations of such terms are checked to see that they are valid (*eg* that a bound variable is not being instantiated in an unsound way). This guarantees soundness. We perform simple first order matching on this representation; this looses us completeness since we can only perform imitation (and not arbitrary higher order unification). So far, however, this incompleteness has not proved a significant problem since our manipulations have only required this very restricted form of matching; our methods have failed to find closed form solutions but not because of this incompleteness. We eventually intend to move to a full higher order representation.

In planning a proof, CLAM uses the methods in the following order: *closed form* is considered first as it is the only terminating method; *standard form* is considered second as it finishes many of the proofs begun by the other methods; *conjugate* and *telescope* are considered next, in that order; and *perturbate* is considered last as (like the induction method in inductive proof planning) it is nearly always applicable and thus a strategy of last resort. Note that theorem (17) admits a trivial solution in which the witness to $S$ is $\sum_{i=m}^{n} u_i$. However, this trivial solution is not found by CLAM because its plans for summing series can only terminate with the *closed form* method, which insists that $S$ is in closed form.

These methods are successful at summing a large number of series with little search. Rippling does need to perform more search than in inductive theorem proving. This is mostly a consequence of the greater number of difference matches possible compared with the (usually) sole induction hypothesis in inductive theorem proving. Rippling is still, however, very controlled as the absence of suitable wave-rules usually terminates unsuccessful branches of the search quickly. Additionally, we are currently developing heuristics for selecting between difference matches which should help to eliminate some of this search.

Among the series that have been summed in this way are those shown in table 1. Problems 8 to 10 are of particular interest as they fall outside the range of Gosper's algorithm, and have not, as far as we are aware, been automatically synthesised before.

| No | Problem | Closed Form | Main Method Used |
|---|---|---|---|
| 1 | $\sum i$ | $\frac{n.s(n)}{2}$ | *telescope* |
| 2 | $\sum i^2$ | $\frac{2.n^3+3.n^2+n}{6}$ | *telescope* |
| 3 | $\sum i+i^2$ | 1)+2) | *standard form* |
| 4 | $\sum a^i$ | $\frac{a^{s(n)}-1}{a-1}$ | *perturbate* |
| 5 | $\sum i.a^i$ | $\frac{s(n).a^{s(n)}-a.\frac{a^{s(n)}-1}{a-1}}{a-1}$ | *perturbate* |
| 6 | $\sum (i+1).a^i$ | $\frac{s(n).a^{s(n)}}{a-1} - \frac{a^{s(n)}-1}{(a-1)^2}$ | *conjugate* |
| 7 | $\sum \frac{1}{i.(i+1)}$ | $\frac{n}{s(n)}$ | *telescope* |
| 8 | $\sum F_i$ | $F_{n+2}-1$ | *telescope* |
| 9 | $\sum \sin(i.\theta)$ | $\frac{(\cos\theta-1).\sin(s(n).\theta)-\sin\theta(\cos(s(n).\theta)-1)}{(\cos\theta-1)^2+\sin^2\theta}$ | *conjugate* |
| 10 | $\sum \cos(i.\theta)$ | $\frac{(\cos\theta-1).(\cos(s(n).\theta)-1)+\sin\theta.\sin(s(n).\theta)}{(\cos\theta-1)^2+\sin^2\theta}$ | *conjugate* |
| 11 | $\sum \binom{m+i}{i}$ | $\binom{m+s(n)}{n}$ | *telescope* |
| 12 | $\sum \binom{s(i)}{s(m)}$ | $\binom{s(n)}{s(s(m))} + \binom{s(n)}{s(m)}$ | *standard form* |

*All sums are from 0 to $n$, $a \neq 1$, $F_i$ is the ith Fibonacci number, and* $\cos(\theta) \neq 1$. *As well as the main method listed, each problem required the use of difference matching, followed by rippling and fertilization. Each plan was terminated by the* closed form *method.*

Table 1: Some Series Summed by Our System

# 7 Related Work

Previous work on summing series falls into two camps: verification and decision procedures.

## Verification

Sometimes we are given a closed form solution and a series, and we verify that they are equal, usually by mathematical induction. This approach has been adopted by Hutter, [9], and by Clarke and Zhao, [7]. Hutter has used the INKA inductive theorem prover system to verify sums and to prove properties of them, *e.g.*

$$\sum_{i=1}^{n} i + \sum_{i=1}^{n} i = n.s(n) \qquad\qquad \sum_{i=1}^{n} i^3 = \sum_{i=1}^{n} i \,.\, \sum_{i=1}^{n} i$$

Clarke and Zhao have used their Analytica theorem prover, built on top of Mathematica, to prove[2]:

$$\sum_{i=0}^{n} \frac{2^i}{1+a^{2^i}} = \frac{1}{a-1} + \frac{2^{s(n)}}{1-a^{2^{s(n)}}}$$

They also sum series using Gosper's algorithm and the built-in Mathematica simplifier.

In principle, it should be possible to adapt these verification methods to the synthesis of closed form solutions by proving theorems of the form (17). However, existing inductive theorem provers are weak at proving theorems containing existential quantifiers like (17). Moreover, like the methods described in this paper, the methods used by humans and reported in mathematics textbooks often do not use induction.

## Decision Procedures

Decision procedures for summation are implemented in general-purpose computer algebra systems like MACSYMA, MAPLE, Mathematica and REDUCE. Such decision procedures are restricted to certain narrow classes of series. For instance, Gosper's algorithm, probably the best decision procedure for summation [8], is restricted to series where the ratio of consecutive partial sums is a rational function. Our technique is not restricted in this way and several of the series listed in table 1 fall outside this class. Another advantage is that these method can be extended to return answers which are not, strictly speaking, closed form (*eg* they can transform certain sums into functions of the Harmonic numbers, $H_n = \sum_{i=1}^{n} \frac{1}{i}$). Additionally, theses methods could equally well be used to reason about infinite absolutely convergent series. Although some such series can sometimes be summed by a decision procedure by considering the limit of a finite series (*eg* $\lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{i^2}$), many cannot as they have no finite closed form (*eg* $\sum_{i=0}^{n} \frac{1}{i!}$).

Another disadvantage of the decision procedure approach is that they are 'black-boxes', providing no rational explanation of the answers they come up with. Our technique produces proofs which are similar in structure to the methods used by humans and reported in mathematics textbooks. They are, therefore, intelligible to mathematicians.

[2]Note that this result is incorrect in the case $a = 1$. This error appears to be due to unsoundness in the Mathematica simplifier.

## 8 Conclusions

Our research in the domain of summing series has shown that the proof planning search control technique is applicable not just to inductive proofs but also to a non-inductive domain. Indeed, some of the tactics developed specifically for inductive proofs are applicable to summing series. In particular, rippling, already shown to be the key tactic for inductive proofs, turns out, remarkably, to be the key tactic in this new domain. It is used as the main sub-tactic by all the major series summing tactics. Outside inductive proofs, rippling must be supplemented by difference matching [5] to set up the initial wave-fronts. With this addition, we predict that rippling will be widely applicable in automated theorem proving. There is room for further extensions to the tactics described to attack a greater class of series. New series summing tactics could be constructed in the same vein.

The proof planning based technique we have described for summing series extends previous techniques in this area. It can be used to synthesise solutions rather than just verify them. It is not restricted to a small class of series. It was designed and built within the space of a few months as an MSc project. It was a simple matter to adapt our existing programs for inductive proofs to this new domain. Most of the methods we have developed for summing series can be readily adapted to closely related tasks *e.g.* finding closed form solutions to products and integrals. The above observations provide evidence for the general applicability of the proof planning formalism in controlling mathematical proofs.

## References


[1] R.S. Boyer and J.S. Moore. *A Computational Logic.* Academic Press, 1979.

[2] A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120, Springer-Verlag, 1988.

[3] 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.

[3] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[4] A. Bundy, F. van Harmelen, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 132–146, Springer-Verlag, 1990.

[5] D. Basin and T. Walsh. *Difference Matching.* In D. Kapur, editor, *11th International Conference on Automated Deduction*, Springer-Verlag, 1992.

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

[7] E. Clarke and X. Zhao. *Analytica - A Theorem Prover for Mathematica.* Technical Report, Carnegie Mellon University, 1991.

[8] R.W. Gosper. Indefinite hypergeometric sums in MACSYMA. In *Proc. MACSYMA Users Conference*, pages 237–252, 1977.

[9] D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 147–161, Springer-Verlag, 1990.

[10] L. Sterling, A. Bundy, L. Byrd, R. O'Keefe, and B. Silver. Solving symbolic equations with PRESS. *J. Symbolic Computation*, 7:71–84, 1989.