Extensions to the Rippling-Out Tactic for Guiding Inductive Proofs

Extensions to the Rippling-Out Tactic for Guiding Inductive Proofs *

Alan Bundy, Frank van Harmelen, Alan Smaill and Andrew Ireland

Department of Artificial Intelligence,
University of Edinburgh,
Edinburgh, EH1 1HN, Scotland.
Email: bundy@edinburgh.ac.uk, Tel: 44-31-225-7774.

Abstract

In earlier papers we described a technique for automatically constructing inductive proofs, using a heuristic search control tactic called rippling-out. Further testing on harder examples has shown that the rippling-out tactic significantly reduces the search for a proof of a wide variety of theorems, with relatively few cases in which all proofs were pruned. However, it also proved necessary to generalise and extend rippling-out in various ways. Each of the various extensions are described with examples to illustrate why they are needed, but it is shown that the spirit of the original rippling-out tactic has been retained.

1 Introduction

In [Bundy 88] we introduced the concept of proof plans and gave a simple example proof plan for guiding inductive proofs. The central idea of this inductive proof plan was a tactic for manipulating the induction conclusion to enable the induction hypothesis to be used in its proof. Following Aubin, [Aubin 75], we called this tactic *rippling-out*. In [Bundy *et al* 88] we described an implementation of this inductive proof plan within the OYSTER-CLAM system for a higher-order, intuitionist, typed logic, and we reported its performance on various standard example inductive theorems. OYSTER is a reimplementation in Prolog of the Nuprl interactive proof editor, [Constable *et al* 86]. Tactics are Prolog programs which drive OYSTER by applying its rules of inference. A proof plan is a tactic

*The research reported in this paper was supported by SERC grant GR/E/44598, and an SERC Senior Fellowship to the first author. We wish to thank our colleagues in the Edinburgh Mathematical Reasoning Group and three anonymous referees for feedback on this paper.

together with its meta-level specification. CLAM is a plan formation program which builds a special-purpose proof plan for each theorem from a set of general-purpose proof plans.

We have tested our implementation extensively on a large number of inductive theorems, most of which are drawn from the Boyer-Moore corpus ([Boyer & Moore 79], appendix A), adapting the original proof plan as necessary. This paper reports the result of that study. The main finding is that the spirit of the proof plan has survived the test. In particular, the rippling-out tactic remains the central idea of the proof plan. Our tests have suggested principled extensions to rippling-out, which preserve the essential intuition behind it while increasing its range of application. Since these extensions include rippling in other directions than 'out', we have named the extended tactic, *rippling*.

Below we report these extensions to rippling-out[1]. They are rippling with multi-wave rules, rippling with conditional wave rules, rippling-sideways and rippling on hypotheses. We illustrate each extension with simple examples of proofs which require them. We argue that the extensions are natural improvements of the original idea. We end with examples of harder inductive theorems whose proof is now within the range of the proof plan.

Before we can do this, however, we briefly recap the main idea of rippling-out and the role it plays within our inductive proof plan.

2 Rippling-Out in Inductive Proofs

To understand rippling-out imagine a loch[2] in which the induction conclusion appears as a reflection of the hypothesis. The reflection is not a perfect image of the original because wherever the induction variable appears in the induction hypothesis, the induction term appears in the induction conclusion. The expressions which appear in the induction conclusion, but not in the induction hypothesis, we call *wave fronts*. They are like ripples on the surface of the loch, which spoil the reflection. Consider, for instance, a simple proof of the associativity of $+$, $\forall X, Y, Z.\ X + (Y + Z) = (X + Y) + Z$ by successor induction on X. The induction hypothesis is

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

where x, y and z represent skolem constants[3]. The induction conclusion is

$$\boxed{s(}\,x\,\boxed{)} + (y + z) \quad = \quad (\boxed{s(}\,x\,\boxed{)} + y) + z \tag{2}$$

The induction term is $s(x)$ and the $s(\ldots)$ constructor function is the wave front.

More generally, a *wave front* is a term from which a proper subterm is deleted. Alternatively, a wave front can be thought of as a collection of nested functions with the innermost argument removed. When this argument is replaced by a term the wave front

[1] To improve readability we have translated the Martin Löf style language used by OYSTER-CLAM into a more conventional notation.

[2] The scottish word for 'lake'.

[3] We adopt the Prolog convention that identifiers starting with upper case letters indicate variables and those starting with lower case letters indicate constants.

is said to *dominate* the term. We adopt the convention that wave fronts are indicated by boxes, as in the example above. Wave fronts are illustrated graphically in figure 1.

Initially, the wave fronts are functions which immediately dominate the induction variable. The role of rippling-out is to move them outwards — just like the ripples on a loch — leaving behind them an unspoilt reflection of the induction hypothesis. Rippling-out works by backwards reasoning from the induction conclusion to the induction hypothesis using *wave rules*. A wave rule is a rewrite rule of the form:

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

where F, S and T are terms with one distinguished argument. T may be empty, but S and F must not be. S and T are called the *old* and *new wave fronts*, respectively. Note that the effect of applying such a rule is to move the old wave front S, in the induction conclusion, outwards past the F and to turn it into a new wave front T.

This general form includes many rewrite rules formed from the step cases of recursive definitions. Restricted to such recursive rules, rippling-out is a constrained version of unfolding or symbolic evaluation. But, as we will see, many wave rules are not formed from recursive definitions, so rippling-out extends unfolding.

A wave rule can be formed from the step case of the recursive definition of $+$, namely:

$$\boxed{s(}u\boxed{)} + v \;\Rightarrow\; \boxed{s(}u + y\boxed{)}$$

where U is u, S and T are s and $F(U)$ is $u + v$. Repeated application of this rule to (2) ripples the two wave fronts to the outside of the left and right hand terms of the induction conclusion.

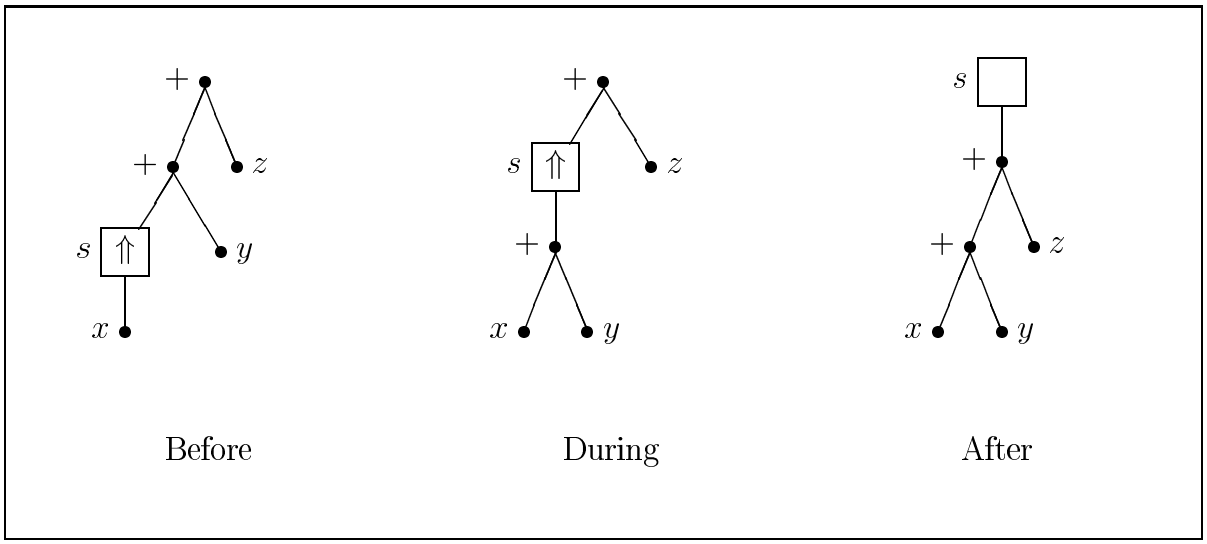
$$\begin{aligned}
\boxed{s(}x\boxed{)} + (y + z) &= (\boxed{s(}x\boxed{)} + y) + z \\
\boxed{s(}x + (y + z)\boxed{)} &= \boxed{s(}x + y\boxed{)} + z \\
\boxed{s(}x + (y + z)\boxed{)} &= \boxed{s(}(x + y) + z\boxed{)}
\end{aligned} \tag{3}$$

A diagrammatic representation of rippling-out is given in figure 1.

Our initial version of rippling-out applied only to induction conclusions that were equations or equivalences. As in this example, wave fronts were rippled-out until they dominated the left- and right-hand sides of these induction conclusions.

The movement of wave fronts can terminate in three ways.

- If a wave front is moved to dominate the left- or right-hand term of the induction conclusion then we say it is *beached*. No further rippling-out of this wave front is then possible or necessary. Two examples can be found in equation (3), above.
- If a wave rule is applied in which T is empty then there is no new wave front. We say it *peters out*. Two examples can be found in equation (4), below.
- If a wave front is not beached, but no wave rule applies to it, then we say the wave front is *stuck*.



The three trees show the term before, during and after rippling-out. Each tree represents the term as an expression tree. The nodes of each tree are labelled by a function or predicate symbol or a constant or variable. A function or predicate of arity n has n descendent subtrees: one for each of its arguments. Wave fronts are indicated by square nodes and all other nodes by dots. Note that the three trees differ from each other only in the position of the square nodes. These square nodes are higher in each successive tree; the 'After' one being beached at the top. The ⇑s in the square nodes indicate the direction of rippling-out.

Figure 1: Rippling-Out $(s(x) + y) + z$

If all wave fronts in a term are beached or peter out then then we say it is *fully rippled*. Otherwise, we say the rippling-out is *stuck*.

When no further rippling-out is possible then the induction hypothesis can often be used as a rewrite rule to produce an equation between two identical terms. Following Boyer and Moore, we call this tactic *fertilization*. The induction hypothesis may be used either way round. In our example we can use it left to right on the left hand side of the induction conclusion, (3), to produce the equation:

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

which is readily proved, so completing the step case of the inductive proof.

Both rippling-out and fertilization are subject to careful control by our overall inductive proof plan.

- Rippling-out is applied to the induction conclusion immediately after the application of induction. The precondition of applying a wave rule to rewrite a sub-expression of the induction conclusion is that any wave front in that sub-expression must match

the old wave front in the rule. This greatly restricts the search space of rippling-out, usually to zero or one wave rule for each wave front in the induction conclusion. It also permits equations which can be interpreted as wave rules in both directions to be used either way round without the risk of looping, since wave fronts must always move outwards and never inwards.

- Fertilization is applied to the induction conclusion when no further rippling-out is possible. It uses the induction hypothesis as a rewrite rule either left to right to the left-hand side of the induction conclusion or right to left to its right-hand side. If the wave fronts on both sides of the induction conclusion are fully rippled then either one of these is sufficient. If only one side is fully rippled then this side is rewritten. If neither side is fully rippled then fertilization is not possible.

3 Multi-Wave Rules and Strong Fertilization

Our first extension is to allow wave rules which ripple out more than one wave front at once. We call these *multi-wave rules*. They are highly desirable if one side of the induction conclusion contains more than one wave front. They allow rippling to merge several wave fronts into one and, hence, continue rippling beyond the point at which the old version of rippling-out would have become stuck. They also allow us to strengthen rippling-out and fertilization and, thereby, apply them to induction conclusions that are not equations or equivalences.

While we are making this extension we will also extend wave rules to cope with wave fronts containing more than one induction variable, and to multiple induction hypotheses. Thus, the new general form of wave rules is:

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

where $1 \leq q_i, r_i \leq m_i$ for $1 \leq i \leq n$. As before, F, the S_is and T are terms with distinguished arguments. T may be empty, but the S_is and F must not be. The S_is are the old wave fronts and T is the new wave front. Note that the concept of wave front has now been generalised to a term from which one *or more* proper subterms have been deleted.

The advantages of multi-wave rules are as follows.

- The replacement of S with the S_is enables the simultaneous rippling-out of multiple wave fronts.
- The replacement of U with the U_is enables the rippling-out of wave fronts containing more than one induction variable.
- The replacement of the single F argument of T by several F arguments allows the simultaneous fertilization of the induction conclusion with more than one induction hypothesis.

These last two extensions go together; multiple Us tend to create multiple Fs, each one making a different selection of Us for its arguments.

A multi-wave rule can be formed from the substitutivity axiom for s, namely:

$$\boxed{s(}\,u\,\boxed{)} = \boxed{s(}\,v\,\boxed{)} \quad \Rightarrow \quad u = v$$

Recall that rippling-out reasons backwards. To apply the implication $p \rightarrow q$ it uses the rewrite rule $q \Rightarrow p$[4]. Thus the direction of implication in the rule above is right to left. Note that this rule is an example of a wave rule which is not formed from the step case of a recursive definition.

Instead of applying fertilization to the induction conclusion (3), we can use this more general form of wave rule to continue the rippling-out and infer:

$$x + (y + z) \quad = \quad (x + y) + z \tag{4}$$

which is identical to the induction hypothesis, (1), *i.e.* the wave fronts have petered out. The proof thus terminates by direct appeal to the induction hypothesis. This direct use of the induction hypothesis constitutes a new form of fertilization which we will call *strong fertilization*. The form of fertilization described above in §2 we will rename *weak fertilization*. Since strong fertilization applies to induction conclusions of any form it is more generally applicable, and its application will normally be the purpose of rippling-out. However, weak fertilization must still be retained to cope with equational or equivalential induction conclusions that get stuck on one side.

To illustrate the use of multi-wave rules to prove non-equational theorems with the aid of multiple induction variables and multiple induction hypotheses, consider the following example.

$$\forall t{:}tree.\ max\_ht(t) \quad \geq \quad min\_ht(t)$$

where $max\_ht(t)$ is the length of the longest path in a binary tree, t, and $min\_ht(t)$ is the length of the shortest. Note that this is an inequality. Trees will be built from the constructor functions $leaf(t)$ and $tree(t_1, t_2)$. For the induction step we will assume the two induction hypotheses:

$$max\_ht(l) \geq min\_ht(l) \qquad max\_ht(r) \geq min\_ht(r)$$

and prove the induction conclusion:

$$max\_ht(\boxed{tree(}\,l, r\,\boxed{)}) \geq min\_ht(\boxed{tree(}\,l, r\,\boxed{)}) \tag{5}$$

in which both l and r are induction variables.

We will need the following multi-wave rules:

$$\begin{aligned}
max_ht(\boxed{tree(}\,u_1^1, u_1^2\,\boxed{)}) \quad &\Rightarrow \quad \boxed{s(max(}\,max_ht(u_1^1), max_ht(u_1^2)\,\boxed{))} \\
min_ht(\boxed{tree(}\,u_1^1, u_1^2\,\boxed{)}) \quad &\Rightarrow \quad \boxed{s(min(}\,min_ht(u_1^1), min_ht(u_1^2)\,\boxed{))} \\
\boxed{s(}\,u_1^1\,\boxed{)} \geq \boxed{s(}\,u_2^1\,\boxed{)} \quad &\Rightarrow \quad u_1^1 \geq u_2^1 \\
\boxed{max(}\,u_1^1, u_1^2\,\boxed{)} \geq \boxed{min(}\,u_2^1, u_2^2\,\boxed{)} \quad &\Rightarrow \quad u_1^1 \geq u_2^1 \boxed{\wedge} u_1^2 \geq u_2^2
\end{aligned}$$

[4]We adopt the convention that $\rightarrow$ stands for implication and $\Rightarrow$ stands for rewriting.

Note that all four of these rules have multiple Us, the first two and last have multiple Fs, and that the last one also has multiple Ss. Recall that the direction of implication in the last two rules is right to left.

Rippling-out (5) then produces:

$$\begin{array}{rcl} \boxed{s(max(}\, max\_ht(l), max\_ht(r) \,\boxed{))} & \geq & \boxed{s(min(}\, min\_ht(l), min\_ht(r) \,\boxed{))} \\ \boxed{max(}\, max\_ht(l), max\_ht(r) \,\boxed{)} & \geq & \boxed{min(}\, min\_ht(l), min\_ht(r) \,\boxed{)} \\ max\_ht(l) \geq min\_ht(l) & \boxed{\wedge} & max\_ht(r) \geq min\_ht(r) \end{array}$$

to which strong fertilization applies, finishing the proof of the step case. Note that the first two wave fronts are compound and must be rippled-out in two stages. Note also that the last wave front is a logical connective. Rippling-out can proceed through predicates and connectives as well as functions.

4 Conditional Wave Rules

Our second extension is to allow conditional wave rules, *i.e.* wave rules that are only true under some condition. They have the form:

$$< condition > \;\rightarrow\; LHS \;\Rightarrow\; RHS$$

where $LHS \Rightarrow RHS$ is a wave rule.

If the condition of a rule is provable from the current hypotheses then, clearly, we can use the rule. But even if it is not currently provable we can still use the rule provided we divide the proof into two cases using the condition and its negation. The condition is then trivially provable in the first case. So a major problem to be solved in the use of conditional rules is when to try to prove the condition within the current case (either immediately or later) and when to use the condition to split the current case into sub-cases.

As a partial solution to this problem, related conditional rules are stored, in CLAM , in complementary sets of the form:

$$\begin{array}{rcl} P_1 & \rightarrow & F(\boxed{S_1^1(}\, U_1^1, \ldots, U_1^{m_1} \,\boxed{)}, \ldots, \boxed{S_n^1(}\, U_n^1, \ldots, U_n^{m_n} \,\boxed{)}) \;\Rightarrow\; RHS_1 \\ \ldots & & \ldots \qquad\qquad\qquad\qquad\qquad\qquad \ldots \\ P_k & \rightarrow & F(\boxed{S_1^k(}\, U_1^1, \ldots, U_1^{m_1} \,\boxed{)}, \ldots, \boxed{S_n^k(}\, U_n^1, \ldots, U_n^{m_n} \,\boxed{)}) \;\Rightarrow\; RHS_k \end{array}$$

where RHS_i is either of the form:

$$\boxed{T_i(}\, F(U_1^{q_1^i}, \ldots, U_n^{q_n^i}), \ldots, F(U_1^{r_1^i}, \ldots, U_n^{r_n^i}) \,\boxed{)}$$

$1 \leq q_j^i, r_j^i \leq m_j$ for $1 \leq j \leq n$ and $1 \leq i \leq k$, or is an expression not containing F and where

$$P_1 \vee \ldots \vee P_k$$

and

$$\forall 1 \leq i \leq n, \exists S_i, \forall 1 \leq j \leq k.\ subsumes(S_i, S_i^j)$$

and the S_is are the wave fronts to be rippled-out in the induction conclusion.

- Note that it is not always possible to form a complementary set from conditional wave rules alone. Some conditional rules in a complementary set will be conditional wave rules and some will be conditional definitions of F.
- A wave front, S, subsumes a wave front S' if S consists of nested copies of S'. For instance, the wave front $s(s(\ldots))$ subsumes the wave front $s(\ldots)$. Note that two consecutive ripples of $s(\ldots)$ will ripple $s(s(\ldots))$ once.

When applying conditional wave rules CLAM proceeds as follows.

- If the condition is one of the existing hypotheses of the current case then the rule can be applied with no further work.
- Else if the rule is a member of a complementary set then the current case is split into sub-cases using the conditions of that set.
- Otherwise, the condition is set up as an additional sub-goal.

In this way we avoid dividing into sub-cases unless each of the sub-cases is likely to succeed. Since, for each sub-cases, there is a conditional rule whose condition is satisfied, then one of the following two situations will obtain.

- If the rule for a sub-case is a conditional wave rule, then rippling-out can continue.
- If the rule for a sub-case does not contain F then the rule provides an explicit definition of F in this case and the proof proceeds by symbolic evaluation (see [Bundy *et al* 88]) rather than induction.

This gives some assurance of success in each sub-case.

This is only a partial solution to the problems of using conditional rules because we might not invest enough effort into proving the condition in the current case before giving up and dividing into cases. Thus we will create an unnecessary case split.

An example of a complementary set of conditional rules can be taken from the definition of the intersection of two sets (represented as lists).

$$\begin{array}{rcrcl} h \in s & \rightarrow & (\boxed{h :: t}) \cap s & \Rightarrow & \boxed{h ::}(t \cap s) \\ \neg h \in s & \rightarrow & (\boxed{h :: t}) \cap s & \Rightarrow & t \cap s \end{array}$$

where $h :: t$ is the set formed by adding element h to the set t. Note that both rules are conditional wave rules, and that T is empty in the second rule.

Consider the theorem:

$$\forall A, B, C.\ A \in B \wedge A \in C \ \rightarrow \ A \in B \cap C$$

The induction conclusion of this theorem is:

$$a \in \boxed{e :: } b \wedge a \in c \ \rightarrow \ a \in (\boxed{e :: } b) \cap c$$

Consider the second wave front. The only matching wave rules are the two conditional rules above. Since these form a complementary set we can use them to suggest a division into two cases, each of which has some assurance of success. After dividing into these two cases and applying the conditional wave rules we get:

$$\begin{array}{rl} e \in c \ \vdash & a \in \boxed{e :: } b \wedge a \in c \ \rightarrow \ a \in \boxed{e :: } (b \cap c) \\ \neg e \in c \ \vdash & a \in \boxed{e :: } b \wedge a \in c \ \rightarrow \ a \in b \cap c \end{array}$$

The remaining wave fronts can be rippled-out once each with the wave rule:

$$x \in \boxed{h :: } t \ \Rightarrow \ \boxed{x = h \vee } x \in t$$

This gives:

$$\begin{array}{rl} e \in c \ \vdash & (\boxed{a = e \vee } a \in b) \wedge a \in c \ \rightarrow \ \boxed{a = e \vee } a \in b \cap c \\ \neg e \in c \ \vdash & (\boxed{a = e \vee } a \in b) \wedge a \in c \ \rightarrow \ a \in b \cap c \end{array}$$

and the wave fronts introduced by this rule can be beached with propositional wave rules. The second case, for example, becomes:

$$\neg e \in c \ \vdash \ \boxed{a = e \wedge a \in c \rightarrow a \in b \cap c} \ \wedge \ a \in b \wedge a \in c \rightarrow a \in b \cap c$$

After strong fertilization this leaves the residue:

$$\neg e \in c \ \vdash \ a = e \wedge a \in c \rightarrow a \in b \cap c$$

which can be readily proved. The first case is similar.

5 Rippling-Sideways and Accumulators

Our third extension is to consider a third way in which rippling can successfully terminate. In our examples so far we have transformed universally quantified variables into skolem constants in both induction hypothesis and induction conclusion. However, universally quantified non-induction variables can be transformed into free variables. Doing this significantly increases the options open to the theorem prover. Wave fronts can be rippled-sideways so that they are absorbed by these free variables during strong fertilization. In principle, this rippling-sideways can always be done using ordinary wave rules: first

applied forwards then applied backwards. In practice, not all the required wave rules are likely to be available, and the central part of the rippling must be done by a new kind of wave rule. We will call these new kinds of wave rule *transverse wave rule*, and we will rename the original wave rules as *longitudinal wave rules*.

A simple transverse wave rule is a rewrite rule of the form:

$$F(\boxed{S(}\,U\,\boxed{)}, V) \;\Rightarrow\; F(U, \boxed{T(}\,V\,\boxed{)})$$

i.e. it moves a wave front from one argument of the function F to another one. F, S and T are each a non-empty term with a distinguished argument. If T were empty then the rule would be a longitudinal wave rule.

Just as for longitudinal rules, the simple transverse wave form can be generalised, to give the multi-wave form:

$$\begin{aligned} &F(\boxed{S_1(}\,U_1\,\boxed{)}, \ldots, \boxed{S_m(}\,U_m\,\boxed{)}, V_1, \ldots, V_n) \\ &\quad\Rightarrow\; F(U_1, \ldots, U_m, \boxed{T_1(}\,V_1\,\boxed{)}, \ldots, \boxed{T_n(}\,V_n\,\boxed{)}) \end{aligned}$$

although, we cannot generalise this to multiple Us, Vs and Fs. Of course, we can also have complementary sets of conditional transverse wave rules, hybrid longitudinal/transverse wave rules, *etc.*

To illustrate rippling-sideways, consider the following example. Let unary rev be the naive list reversal function and binary $qrev$ be the tail recursive list reversal function, so that the following rewrite rules are available.

$$\begin{aligned} rev(nil) &\Rightarrow nil \\ rev(\boxed{hd :: }\,tl) &\Rightarrow rev(tl)\boxed{<> (hd :: nil)} \qquad (6) \\ qrev(nil, l) &\Rightarrow l \\ qrev(\boxed{hd :: }\,tl, l) &\Rightarrow qrev(tl, \boxed{hd :: }\,l) \qquad (7) \end{aligned}$$

where $<>$ is infix list append and the second argument of $qrev$ is an accumulator. Note that 6 is a longitudinal wave rule and 7 is a transverse wave rule.

Consider the following theorem connecting these two list reversal functions.

$$\forall L, M.\; rev(L) <> M \;=\; qrev(L, M)$$

To prove this theorem let L be the induction variable then, by the discussion above, the induction hypothesis is:

$$rev(l) <> M \;=\; qrev(l, M) \qquad (8)$$

where M is a free variable, and the induction conclusion is:

$$rev(\boxed{h :: }\,l) <> m \;=\; qrev(\boxed{h :: }\,l, m)$$

To make the induction conclusion match the induction hypothesis we will ripple the two wave fronts sideways so that each is adjacent to an m. Applying rule (7) to the right hand side does this in one step.

$$rev(\boxed{h :: }l) <> m \quad = \quad qrev(l, \boxed{h :: }m)$$

Rippling the left hand wave front is more difficult. First, (6) is applied to ripple the wave front out.

$$(rev(l)\boxed{<> (h :: nil)}) <> m \quad = \quad qrev(l, \boxed{h :: }m)$$

Second, the associativity of $<>$ is used as a transverse wave rule to ripple the wave front sideways to the left hand m.

$$rev(l) <> (\boxed{(h :: nil) <>}m) \quad = \quad qrev(l, \boxed{h :: }m)$$

Third, symbolic evaluation is applied to tidy up the wave fronts surrounding the sinks. This rewrite $(h :: nil) <> m$ to $h :: m$ on the left hand side. Finally, strong fertilization applies, instantiating M to $h :: m$, and proving the step case.

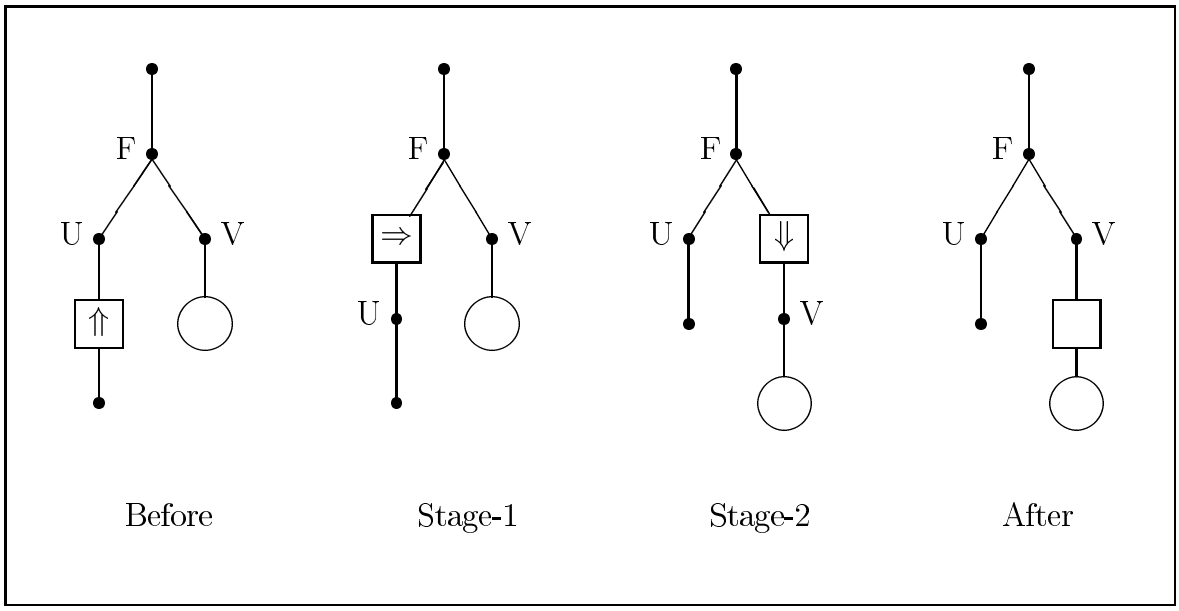
Universally quantified variables in the conjecture which are not chosen as induction variables (like M in $rev/qrev$ proof above) become skolem constants in the induction conclusion (m in the example above). We will call these constants *accumulators*[5] An occurrence of an accumulator is called a *sink*. Rippling-sideways enables us to get rid of a wave front by moving it to directly dominate a sink. This is done in such a way that every sink of each accumulator is dominated either by the same wave front or by none.

Rippling-sideways is controlled as follows. Sets of wave fronts and sets of sinks are paired up so that for each accumulator either none of its sinks are paired or all are. For each pair of wave fronts and sinks the wave fronts are first rippled-out using longitudinal rules, then across with a transverse rule, then down to the sinks using longitudinal rules backwards. A diagrammatic representation of a simple case of this process is given in figure 2.

Note that not all wave rules can be legally used both forwards and backwards, *e.g.* if they are formed from implications then they can only be used forwards on sub-formulae of positive polarity and backwards on sub-formulae of negative polarity. The polarity of a sub-expression is the parity of the number of implicit or explicit negations it is contained within, *e.g.* in $\neg p \wedge q \rightarrow r$, p and r are of positive polarity and q is of negative polarity. Especial attention must be paid to this during the last stage of rippling-sideways (and rippling on hypotheses, see §6).

The strong direction imposed on rippling-sideways by this procedure prevents looping. A wave front must always move towards a sink, and cannot move backwards just because there is a rule available to so move it. This explains why the associativity of $<>$ does not cause a loop in the $rev/qrev$ example above.

[5]We adopt this terminology because of the analogy with accumulators in computer programs. In fact, there is a direct relationship between them, which space does not permit us to explore.



The four expression trees show the expression before, during and after rippling-sideways. Sinks are indicated by circle nodes and wave fronts by square nodes, as in figure 1. F is the least upper bound of the wave front and sink. U and V are the immediate daughters of F in the direction of the wave front and sink, respectively. In Stage-1 the wave front has been rippled up between the U and F nodes. In Stage-2 it has been rippled sideways to between the V and F nodes. In After it has been rippled down to the sink.

Figure 2: Stages of Transverse Rippling

6 Rippling on Hypotheses

Our fourth extension is to adapt rippling to proofs by destructor induction. In all the proofs considered so far the induction rules have been in the constructor style, that is the induction hypotheses have been of the form $P(x)$ and the induction conclusions of the form $P(c(X))$, for some constructor function, c. This neglects proofs in which the induction hypothesis has the form $P(d(x))$ and the induction conclusion has the form $P(x)$, for some destructor function, d.

Rippling is readily adapted to destructor induction. Instead of rippling backwards on the induction conclusion we ripple forwards on the induction hypothesis. Both the forwards direction of the rewriting and the use of wave rules about destructor functions means that we must make much more use of conditional rules than in the constructor case. This tends to make the proofs more messy. On the plus side many inductions are more naturally expressed in destructor form.

To illustrate the technique, here is a destructor version of the associativity of $+$. The induction hypothesis is:

$$x \neq 0 \ \wedge \ \boxed{p(}\boxed{x}\boxed{)} + (y + z) \ = \ (\boxed{p(}\boxed{x}\boxed{)} + y) + z$$

where p denotes the predecessor function on natural numbers. The induction conclusion is:

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

To ripple-out the wave fronts in the induction hypothesis we will need the two wave rules:

$$\begin{array}{rcl} u \neq 0 & \rightarrow & \boxed{p(}u\boxed{)} + v \ \Rightarrow \ \boxed{p(}u + v\boxed{)} \\ u \neq 0 \wedge v \neq 0 & \rightarrow & \boxed{p(}u\boxed{)} = \boxed{p(}v\boxed{)} \ \Rightarrow \ u = v \end{array}$$

This permits rippling similar to that for the constructor proof (see §2), but with additional work to prove the conditions.

7 A More Interesting Example

The extensions to rippling-out described above were all illustrated on examples chosen for their simplicity. However, each of these extensions has significantly increased the power of our OYSTER-CLAM system. As an illustration of this increased power we discuss here a more interesting theorem that our system is now able to prove.

Let binary *count* measure the number of occurrences of an element in a list and unary *sort* be a function for sorting lists. OYSTER-CLAM can prove that the number of occurrences of each element of the list is invariant under sorting, *i.e.*

$$\forall A, L.\ count(A, sort(L)) \ = \ count(A, L)$$

where *sort* is defined as a naive, insert sort algorithm from which we can extract the rewrite rules:

$$\begin{array}{rcl} sort(nil) & \Rightarrow & nil \\ sort(h :: t) & \Rightarrow & insert(h, sort(t)) \end{array}$$

This theorem is from the Boyer-Moore corpus (theorem 118, p342 of [Boyer & Moore 79]).

The proof divides into 7 cases. These case splits are determined by using complementary sets of wave rules. For instance, one split is determined by the following complementary set of conditional wave rules, which is derived from the recursive definition of *count*.

$$\begin{array}{rcl} x = h & \Rightarrow & count(x, \boxed{h :: }t) = \boxed{s(}count(x, t)\boxed{)} \\ x \neq h & \Rightarrow & count(x, \boxed{h :: }t) = count(x, t) \end{array}$$

It also requires some rippling with multi-wave rules.

The proof plan of this theorem is one of the biggest generated by CLAM . It consists of 58 nodes with 3 inductions and 7 case splits and takes 41.5 cpu secs to find, but requires no search. The two nested inductions correspond to proving 'in-line' two complementary conditional wave rules which are required in the proof. OYSTER takes 1,535 cpu secs to unpack this plan into 4,204 object-level inference steps[6].

An 'in-line' proof of a wave rule happens as follows. During the main induction rippling becomes stuck on one side of an equation or equivalence. Weak fertilization is used on one side, but this leaves a subgoal to be proved, usually by a nested induction. This subgoal is an instance of a wave rule that would have unstuck the ripple had it been available in the first place.

8 Conclusion

In this paper we have described four extensions to the rippling-out tactic: multi-wave rippling, conditional rippling, rippling-sideways and rippling on hypotheses. Our extensions have improved the power of rippling while preserving its spirit.

Rippling plays the central role in our proof plan for inductive proofs. This proof plan and the rippling tactic have proved surprisingly successful. It has been tested on 68 of the examples from the Boyer-Moore corpus and proved all but 3[7] of them. No search is required. Our proof plan is much less sensitive than the Boyer-Moore theorem prover to the order in which theorems are presented; if an essential lemma is not available for rippling when required, an appropriate instance of it is proved 'in-line'. It does, however, sometimes rely on a previously proved lemma to suggest the appropriate induction rule.

However, our proof plan does not constitute a decision procedure for induction proofs, which is in any case an undecidable area. Nor does rippling always succeed in bridging the gap between induction hypothesis and induction conclusion. Rippling will stick if the right wave rule is not available. If it sticks on both sides of an equation, even weak fertilization will not apply. One way to proceed in such situations is shake the current goal into a form in which a wave rule *will* apply. For instance, a wave front might be moved sideways to a non-sink with a commutative law or transverse wave rule. We are currently investigating such possibilities.

The OYSTER-CLAM system is written in Quintus Prolog. Multi-wave, conditional and sideways rippling and rippling on hypotheses have all been implemented in the system. Timings quoted above are for the system running on a Sun 3/60 computer.

[6] 3631 of these steps are concerned with proving the well-formedness of logical expressions, *i.e.* that expressions belong to types. The presence of large numbers of such steps is a property of the Martin Löf logic. They are mostly trivial. Thus only the remaining 673 steps are significant.

[7] These recalcitrant 3 fail because of essentially trivial difficulties with the typing, and not because of limitations of the proof plan.

References

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

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

[Bundy 88] 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. Longer version available from Edinburgh as Research Paper No. 349.

[Bundy *et al* 88] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. *Experiments with Proof Plans for Induction*. Research Paper 413, Dept. of Artificial Intelligence, Edinburgh, 1988. To appear in JAR.

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