Dynamic Rippling, Middle-Out Reasoning and Lemma Discovery

Moa Johansson[1], Lucas Dixon[2], and Alan Bundy[2]

[1] Dipartimento di Informatica, Università degli Studi di Verona [**]
[2] School of Informatics, University of Edinburgh
`moakristin.johansson@univr.it, {l.dixon, a.bundy}@ed.ac.uk`

Abstract. We present a succinct account of *dynamic rippling*, a technique used to guide the automation of inductive proofs. This simplifies termination proofs for rippling and hence facilitates extending the technique in ways that preserve termination. We illustrate this by extending rippling with a terminating version of *middle-out reasoning* for lemma speculation. This supports automatic speculation of schematic lemmas which are incrementally instantiated by unification as the rippling proof progresses. Middle-out reasoning and lemma speculation have been implemented in higher-order logic and evaluated on typical libraries of formalised mathematics. This reveals that, when applied, the technique often finds the needed lemmas to complete the proof, but it is not as frequently applicable as initially expected. In comparison, we show that theory formation methods, combined with simpler proof methods, offer an effective alternative.

1 Introduction

Inductive proof techniques are required for reasoning about repetition. Examples include recursively defined data structures, such as natural numbers, lists and trees. A significant strand of research in automated inductive theorem proving revolves around a technique called *rippling*, which is primarily used to guide rewriting of the step-case [23, 11, 6]. The guidance provided by rippling is based on observing and annotating syntactic differences between the step-case subgoal and the induction hypothesis. Rippling ensures termination without requiring rewrite rules to be oriented in advance, which makes it easy to configure, and allows it to solve some problems that are otherwise difficult with traditional approaches to rewriting. Recent work has focused on *dynamic rippling* which computes the differences between the goal and the induction hypothesis after each step [23, 11]. In the rest of this paper, we refer to dynamic rippling simply as rippling.

Automating inductive proofs gives rise to several challenging problems, including the discovery of lemmas [5]. It has generally been assumed that lemma

[**] This research was funded by EPSRC grants EPE/005713/1 and EP/P501407/1, and by grant 2007-9E5KM8 of the Ministero dell'Istruzione Università e Ricerca. We would also like to thank Andrew Ireland for many helpful discussions.

discovery requires user intervention. Interactive theorem provers, such as Isabelle [21], have large libraries of previously proved theorems and carefully configured proof tools that use them. However, for new theory developments, users often spend considerable time identifying and proving the required background lemmas.

Lemma speculation is a heuristic technique for automatically constructing a missing lemma using information from a failed inductive proof attempt. It has long been considered a promising technique to improve rippling based proofs [6, 5, 15], and a similar approach has also been proposed in [18]. Lemma speculation constructs a schematic lemma which preserves parts of the goal that are similar to the inductive hypothesis and introduces meta-variables to stand for unknown term-structure. The meta-variables are then incrementally instantiated by rewriting steps. This approach, of gradually instantiating the meta-variables in each step, is called *middle-out reasoning*, and has been proposed to tackle a wide variety of problems [8, 15, 19, 7].

The main novel contributions in this paper are as follows:

– A logical account of *dynamic-rippling*. This separates the issues of annotation, guidance, and termination. In particular, it describes how these issues fit together. The result is a succinct formal description which is easy to extend and experiment with. Furthermore, it results in much simpler termination proofs than that of Basin and Walsh [3].
– An illustration of how our formalism for rippling can be extended by describing lemma speculation with middle-out reasoning. This improves on earlier work as it applies to higher-order domains, ensures the termination of middle-out reasoning and enjoys the property that every middle-out rippling proof corresponds to a traditional rippling proof with the speculated lemma.
– An implementation and evaluation of rippling with lemma speculation in IsaPlanner [11]. The implementation builds on the LCF-design methodology of the Isabelle system [21]; all proofs are compositions of a small set of basic trusted inference rules in Isabelle. Our evaluation considers a variety of common theories, including lists, natural numbers, trees and inequalities.

Although our evaluation is positive for our middle-out reasoning, and generally positive for the effectiveness of lemma speculation, it highlights an important and interesting negative result for the applicability of lemma speculation. We remark that this is necessarily an empirical result: there are infinitely many problems which lemma speculation works for, and infinitely many for which it fails. Our empirical evaluation is based on a study of typical theories in interactive proof assistants.

Another approach to lemma discovery is to attempt to automate the synthesis of a richer background theory, given the initial definitions of datatypes and functions, as implemented in the IsaCoSy system [17]. IsaCoSy synthesises progressively larger conjectures using the available theory and avoids the generation of any reducible terms. We compare IsaPlanner's performance using lemma speculation against using a background theory generated by IsaCoSy along with a simpler technique called *lemma calculation*. This shows that more theorems are proved in the latter case. Theory formation offers an effective alternative to the more complex lemma speculation technique.

Notation. We use the symbol @ for the list append function, # for cons, [] for nil, as well as the usual notation where $[1, 2, 3]$ is the list $1\#2\#3\#[]$. Variables which are allowed to be instantiated by unification are written $?F$, following the Isabelle convention. These are referred to as *meta-variables*. A *schematic* goal (or lemma) is a goal containing meta-variables. We use the equals symbol $(=)$ for object level equality and the symbol $(\equiv)$ to denote that two terms are syntactically identical. We use $:=$ for definitions. Finally, we write $t\sigma$, for a term, t, under substitution σ.

2 Proof-Planning and Rippling

Proof-planning is a technique used to guide search in automated theorem proving by exploiting the fact that there are families of proofs with a similar structure [9]. One such family is proof by induction. The development of proof-planning was motivated by the observation that human mathematicians often have a high level plan for how to go about solving a proof and then fill in the exact details. Rippling is a proof plan commonly used to guide rewriting of the step-case in inductive proofs [6]. It works by identifying and annotating differences and similarities between two terms, typically referred to as the *skeleton* and the *goal*. In an inductive proof, the skeleton is the inductive hypothesis of the step-case, and the inductive conclusion of the step case is the goal. Rippling guides rewriting to reduce the differences between the goal and the skeleton, with the aim of arriving at a situation where the goal can be justified by the skeleton. This justification is called *strong fertilisation*.

A term can be annotated, with respect to a given skeleton, by identifying the *wave-fronts*, *wave-holes*, and *sinks*. The wave-fronts are the parts of a term that differ between the goal and the skeleton. In constructor-based inductive proofs, these are initially the constructors introduced in the step-case goal. Wave-holes are sub-terms inside a wave-front that are part of the skeleton. Sinks are the positions corresponding to universally quantified variables in the skeleton, which can be instantiated during fertilisation. Wave-fronts have directions: *outward* wave-fronts intend to move the differences to the top-of the term tree, while *inward* wave-fronts intend to move them to a sink.

A typical example of an annotated goal is the step-case in the inductive proof of the commutativity of addition:

$$\begin{array}{ll} \textbf{Skeleton} \text{ (inductive hypothesis):} & \forall b'.\ a + b' = b' + a \\ \textbf{Goal} \text{ (step-case goal):} & \boxed{\mathit{Suc}\ \boxed{a}}^{\uparrow} + \lfloor b \rfloor = \lfloor b \rfloor + \boxed{\mathit{Suc}\ \boxed{a}}^{\uparrow} \end{array} \tag{2.1}$$

The position of the universally quantified variable b' becomes a sink in the goal, annotated as $\lfloor b \rfloor$. Wave-fronts are visualised by shaded boxes with an arrow indicating outward or inward direction. For our purposes, annotations can be viewed as function symbols on the term. Wave-fronts are annotated by the function $wf : (\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, where the first argument is the term inside the wave-front, and the second argument is the wave-hole. Sinks are annotated by the function $sink : \alpha \Rightarrow \alpha$. For the term in the example above this becomes: $(wf\ \ Suc\ a) + (sink\ \ b) = (sink\ \ b) + (wf\ \ Suc\ a)$. For a more detailed account of annotated terms see [12, 11].

During proof search, each application of a rule moves the wave-fronts towards the top of the term tree. As we shall see in Example 1, this decreases the *ripple-measure* (see Def. 4) which guides the proof search. In the final step, the goal is an instance of the skeleton and no wave-fronts remain. The inductive hypothesis is then applied to conclude the proof by strong fertilisation.

Example 1: Rippling and Strong Fertilisation. As an example of a rippling proof we return to the step-case of the inductive proof of the commutativity of addition (from equation 2.1). The rippling proof of the step case is given below. Note that this should be read bottom up to see how the proof was discovered:

$$\begin{array}{c|l|r}
\hline & \text{Strong Fertilisation} & \\
a + \lfloor b \rfloor = \lfloor b \rfloor + a & \text{Measure: 0} & \\
\hline & \text{ripple-step, using:} & \\
& ((Suc\ x) = (Suc\ y)) = (x = y) & (2.2) \\
\boxed{Suc(\boxed{a + \lfloor b \rfloor})}^{\uparrow} = \boxed{Suc(\boxed{\lfloor b \rfloor + a})}^{\uparrow} & \text{Measure: 2} & \\
\hline & \text{ripple-step, using:} & \\
& x\ +\ (Suc\ y) = Suc(x\ +\ y) & (2.3) \\
\boxed{Suc(\boxed{a + \lfloor b \rfloor})}^{\uparrow} = \lfloor b \rfloor + \boxed{Suc(\boxed{a})}^{\uparrow} & \text{Measure: 3} & \\
\hline & \text{ripple-step, using:} & \\
& (Suc\ x)\ +\ y = Suc(x\ +\ y) & (2.4) \\
\boxed{Suc(\boxed{a})}^{\uparrow} + \lfloor b \rfloor = \lfloor b \rfloor + \boxed{Suc(\boxed{a})}^{\uparrow} & \text{Measure: 4} &
\end{array}$$

There are two functions of interest for annotated terms, *erasure* which returns the unannotated term, and *skel* which returns the instance of the skeleton that is within the goal.

Definition 1 (Erasure). *The function erasure(t) replaces every wave front symbol (wf), in the term t, by the lambda term $\lambda f\ t.\ f\ t$, and every sink symbol (sink) by the identity function.*

Definition 2 (Skel). *The function skel(t) replaces every wave front symbol (wf), in the term t, by the projection lambda term $\lambda f\ t.\ t$, and every sink symbol by the identity function.*

Definition 3 (Annotion set). *The set of annotations for a given term, t, with respect to a skeleton, s, is defined by:*

$$annot(s,t) := \{a \mid erasure(a) \equiv t \wedge \exists \sigma.\ skel(a) \equiv s\sigma\}$$

Annotation algorithms are described in [23, 11]. After the initial goal has been annotated, rippling proceeds by applying rules derived from function definitions, axioms and existing theorems and lemmas. These rules are referred to as *wave-rules*. To guide the application of wave-rules so that the differences between the skeleton and the goal are decreased, rippling uses a well-founded measure on annotated terms, called a *ripple-measure*.

Definition 4 (Ripple-Measure). *A ripple measure is a well-founded (strict) partial-order on annotated terms, parametrised by a common skeleton, such that when the skeleton unifies with the annotated term, the measure is minimal. We write $Mes_s(t_1) < Mes_s(t_2)$ when t_1 is less than t_2 with respect to a ripple-measure parametrised by skeleton s.*

We use the *sum of distances* measure, from [11], which sums the distance from outward wave-fronts to the top of the term tree, and from inward wave-fronts to the nearest sink (see Example 1). This results in a natural number. The ripple-measure is the usual less-than ordering on natural numbers.

Theorem 1 (Sum of distances is a ripple-measure). *The sum of distances measure is well-founded and when the goal unifies with the skeleton, it is minimal.*

Proof. The well-foundedness follows directly from the less-than ordering on natural numbers. When a goal term t unifies with a skeleton s, then t contains no unsunk wave-fronts (otherwise t is not an instance). Hence the sum of distances measure for all annotations is 0, the minimal value.

Definition 5 (Dynamic Rippling, Ripple-Step). *Let W be a context containing a set of wave-rules, s be a skeleton, and a_i be an annotated term. A* rippling step *is an inference of the form:*

$$\frac{W,\ s,\ a_2 \vdash g[t_2\sigma]}{W,\ s,\ a_1 \vdash g[t_1\sigma]} \quad \begin{array}{l} ((t_1 = t_2) \in W \vee (t_2 = t_1) \in W) \\ a_1 \in annot(s, g[t_1\sigma]) \\ a_2 \in annot(s, g[t_2\sigma]) \\ Mes_s(a_2) < Mes_s(a_1) \end{array}$$

The first condition identifies an equational wave rule in the context W. The next two conditions ensure that the goals have rippling annotations, and the last condition ensures that the ripple measure decreases. Rippling *is the repeated application of ripple-steps.*

Using the above definition of rippling, the proof of termination follows easily:

Theorem 2 (Termination of Rippling). *Rippling always terminates.*

Our formalism renders this theorem as a trivial consequence of the well-foundedness of the ripple measure; there is no infinite sequence of ripple-steps.

Definition 6 (Blocked). *Rippling is blocked at a goal when the set of ripple-steps is empty. This happens when no measure decreasing wave-rules can be applied.*

Example 2. Returning to example 1, if rule 2.2 was not available, then rippling would be blocked at the subgoal: $\boxed{Suc(\boxed{a + \lfloor b \rfloor})}^{\uparrow} = \boxed{Suc(\boxed{\lfloor b \rfloor + a})}^{\uparrow}$. When rippling is blocked, in many cases, including this one, it is still possible to complete the proof by rewriting the goal with the inductive hypothesis. This is called *weak fertilisation.*

Definition 7 (Weak Fertilisation).

$$\frac{W,\ s \vdash g[t_2\sigma]}{W,\ s,\ a \vdash g[t_1\sigma]} \quad \begin{array}{l}(s \equiv (t_1 = t_2) \vee s \equiv (t_2 = t_1)) \\ a \in annot(s, g[t_1\sigma])\end{array}$$

In example 2, the blocked goal can be weak-fertilised to produce the new subgoal $Suc(b + a) = Suc(b + a)$, which is true by reflexivity.

Lemma calculation is a lemma discovery technique that is simpler than lemma speculation. While the latter applies when rippling becomes blocked before fertilisation, lemma calculation proves residual goals as lemmas after fertilisation. Logically, it is justified by the cut-rule. Although simple, variants of this technique are widely used by automatic inductive provers.

Definition 8 (Lemma Calculation).

$$\frac{W,\ s \vdash l \qquad W \cup l, s \vdash g}{W,\ s \vdash g} \quad l \equiv generalise(g)$$

where the generalise function produces a lemma such that $\exists \sigma.\ l\sigma \equiv g$

Example 3: Lemma Calculation. In example 1, note that only rule 2.4 comes from the standard definition of addition in Peano arithmetic. If rippling is only given the definition of '+' with no extra lemmas, then rippling gets blocked even earlier, at the subgoal: $\boxed{Suc(\boxed{a + \lfloor b \rfloor})}^{\uparrow} = \lfloor b \rfloor + \boxed{Suc(\boxed{a})}^{\uparrow}$. Weak-fertilisation is then applied to the left-hand side of the blocked goal, resulting in the new subgoal: $Suc(b + a) = b + (Suc\ a)$. However, this goal cannot be solved by simplification. This is when the lemma calculation is applied. In this case, the lemma is simply the remaining subgoal, which is the missing rule 2.3.

In our implementation in IsaPlanner, calculated lemmas are generalised using common sub-term generalisations (replacing equal sub-terms occurring on both sides of an equation with a new variable). The lemma created by calculation is then subjected to an inductive proof attempt. Other generalisation techniques used in lemma calculation have been studied in [1]. The proof-plan for the step-case of our inductive prover is summarised by the functional pseudo-tactic shown in Fig. 1. The interested reader may examine the implementation at:
`http://dream.inf.ed.ac.uk/projects/isaplanner`

```
rippling = blocked ORELSE
  (ripple_step THEN rippling);

fertilisation =
  strong_fertilisation ORELSE
    (weak_fertilisation THEN
    (simplification THEN
    (solved ORELSE
      (lemma_calculation THEN
      (inductive_proof AND apply_lemma)))
    ));
```

```
stepcase_lemma_calc =
  rippling THEN fertilisation;

stepcase_lemma_spec =
  rippling THEN
  (fertilisation ORELSE
    (speculate_lemma THEN
     middle_out_rippling THEN
     ((solved ORELSE
         stepcase_lemma_spec)
      AND inductive_proof)));
```

Fig. 1. A tactic-style presentation of the proof-plans for the step-case of an inductive proof. Left: shows rippling and fertilisation with lemma calculation. Right: shows the basic step-case proof plan as well as its extension with lemma speculation. The `solved` tactic succeeds only if the goal was solved; the `THEN`, `ORELSE` have the standard behaviour; the `AND` tactical attempts to solve the first subgoal by the first tactic and if that is successful then attempts the second subgoal with the second tactic; the `inductive_proof` tactic is the top-level inductive prover.

3 Middle-Out Rippling and Lemma Speculation

If rippling becomes blocked before the inductive hypothesis can be applied, the basic proof-plan using lemma calculation after weak fertilisation fails (Fig. 1, top-right). Lemma speculation solves this difficult class of problems by creating a schematic equational lemma to unblock rippling. Like lemma calculation, the lemma is cut into the proof. However, unlike lemma calculation, the lemma is applied to the goal as a rewrite before the lemma has been proved. This introduces the schematic variables into the subgoal. Subsequent specialised ripple-steps, called middle-out ripples, are then used to rewrite the goal and instantiate the meta-variables (see §3.2). Finally, once the goal has been proved and the lemma is fully instantiated, a proof of the lemma is attempted. The tactic for rippling with lemma speculation is shown at the right of Fig 1, with the corresponding formal definition of a lemma speculation step being:

Definition 9 (Lemma Speculation).

$$\frac{W \cup (l = r),\ s,\ a,\ c \vdash g[r] \qquad W \vdash (l = r)}{W,\ s,\ a[l'] \vdash g[l]} \quad \begin{array}{l} (l = r) \in \mathit{LemmaSpecSet}(a) \\ c \equiv \mathit{Trace}(s, [g[r], g[l]]) \\ \lambda x.\ \mathit{erasure}(a[x]) \equiv \lambda x.\ g[x] \\ a[l'] \in \mathit{annot}(s, g[l]) \end{array}$$

The schematic lemma $(l = r)$ is said to unblock the subterm l. The last two conditions ensure that $l \equiv \mathit{erasure}(l')$ and l is the subterm corresponding to l'.

The lemma speculation rule extends the context with a trace (c) which is used to maintain a list of the goals following a lemma speculation step. The trace is used to ensure the termination of middle-out reasoning (see §3.1). The set $\mathit{LemmaSpecSet}(a)$ is all equations of the form $l = r$ where l is the erasure of a subterm, l', of the annotated goal, and l' contains at least one wave front. The right hand, r, is the skeleton of l' with meta-variables inserted into all positions where a wave-front may occur, i.e. above each function symbol and in each sink position.

For example, consider the goal $g[l] \equiv p\ (f\ (h\ x)\ y)$ with the annotated subterm l' as: $f\ (\boxed{h\ \boxed{x}}^{\uparrow})\ \lfloor y \rfloor$. A schematic lemma for this term is $f\ (h\ x)\ y = ?F_1\ (f\ x\ (?F_2\ y))$. The new subgoal produced by lemma speculation is then:

$$p\ (\boxed{?F_1\ (f\ x\ (\lfloor ?F_2\ y \rfloor))}) \tag{3.1}$$

The dashed box marks the location of an unsunk meta-variable which is called a *potential wave-front* in [15]; instantiation of the meta-variable may introduce wave-fronts into the goal.

3.1 Measures for Schematic Goals

The key invariant initiated by lemma speculation is that there exists a substitution that will result in the lemma's application being a valid ripple-step. This invariant is then extended to the application of ripple-steps as they instantiate the meta-variables. This is captured by the predicate *MesDecr* which, for the list of goals in the trace $([g_n, \ldots, g_1])$, is true if there is a substitution that makes $[g_1, \ldots, g_n]$ sequentially measure decreasing with respect to a given skeleton s:

$$\begin{aligned} &\mathit{MesDecr}(\sigma, \mathit{Trace}(s, [g_n, \ldots, g_1])) \equiv \exists a_n \ldots a_1.\ \mathit{Projection}(\sigma, \mathit{Vars}([g_n, \ldots, g_1])) \\ &\quad \wedge a_n \in \mathit{annot}(s, g_n\sigma) \wedge \cdots \wedge a_1 \in \mathit{annot}(s, g_1\sigma) \\ &\quad \wedge \mathit{Mes}_s(a_n) < \ldots < \mathit{Mes}_s(a_1) \end{aligned} \tag{3.2}$$

where $\mathit{Projection}(\sigma, \mathit{Vars}([g_n, \ldots, g_1]))$ ensures that σ consists of projections for all meta-variables in the goals.

Theorem 3 (Lemma Speculation is Measure Decreasing). *By construction, all lemmas produced by lemma speculation will initially result in a list of measure decreasing subgoals:* $\exists\sigma.\ \mathit{MesDecr}(\sigma, \mathit{Trace}(s, [g[r], g[l]]))$.

Proof. The projection of the meta-variables onto their arguments will result in $g[r]$ removing all wave fronts. This is always measure decreasing for the sum-of-distances measure.

3.2 Middle-Out Rippling

After speculation, the lemma subgoal, $W \vdash (l = r)$, and the main subgoal, $W \cup (l = r),\ s,\ c \vdash g[r]$, contain shared meta-variables. If ordinary rippling were applied to the main subgoal, the presence of these meta-variables would let higher-order unification find unifiers with any rewrite rule, causing the search space to become intractably large. In order to manage the rippling search space and ensure termination of rippling, we use a variant of the ripple-step rule, called *middle-out ripple-step*.

Definition 10 (Middle-Out Ripple-Step).

$$\frac{(\Delta \quad W,\ s,\ c_2 \vdash g[t_2])\sigma_1}{\Delta \quad W,\ s,\ c_1 \vdash g[t_1]} \quad \begin{array}{l} (t_1' = t_2) \in W \vee (t_2 = t_1') \in W \\ \mathit{RestrUnifiers}(t_1, t_1', \sigma_1) \\ c_1 \equiv \mathit{Trace}(s, l) \quad c_2 \equiv \mathit{Trace}(s, (g[t_2]\#l)\sigma_1) \\ \exists \sigma_2.\ \mathit{MesDecr}(\sigma_2, c_2) \end{array}$$

This rule differs from a regular rewrite step in that the substitutions can involve variables in both the rule and the goal. Meta-variables shared with the lemma are also instantiated in all other subgoals (Δ), notably in the lemma subgoal. The relation *RestrUnifiers*, embodies a heuristic that restricts the unifiers (σ_1) so as to avoid trivial ones which can occur with any wave-rule (see §3.3).

The *MesDecr* relation ensures that the new subgoal forms part of a measure decreasing chain. This holds if there exists a grounding substitution σ_2 for the remaining meta-variables, which makes the trace measure decreasing. Note that the definition of *MesDecr* requires σ_2 to be a projection of the uninstantiated variables of the trace (see equation 3.2). This is the key condition that ensures termination of middle-out rippling (see §3.4).

Each middle-out step is followed by an attempt to find projection-instantiations for the remaining meta-variables using either of the following *eager-fertilisation* rules:

Definition 11 (Eager Strong Fertilisation).

$$\frac{\Delta\sigma}{\Delta \quad W,\ s,\ c \vdash g} \quad \begin{array}{l} s\sigma \equiv g\sigma \\ \mathit{MesDecr}(\sigma, c) \end{array}$$

Definition 12 (Eager Weak Fertilisation).

$$\frac{(\Delta \quad W \vdash g[t_2])\sigma}{\Delta \quad W,\ s,\ c \vdash g[t_1]} \quad \begin{array}{l} s \equiv (t_1' = t_2) \vee s \equiv (t_2 = t_1') \\ t_1\sigma \equiv t_1'\sigma \\ \mathit{MesDecr}(\sigma, c) \end{array}$$

As the *MesDecr* condition requires σ to be a projection of all remaining meta-variables, eager fertilisation results in a ground goal. When the goal contains all variables from the speculated lemma, this also results in the speculated lemma having no further meta-variables. Like middle-out steps, the *MesDecr* condition ensures that instantiations result in c being a valid measure-decreasing ripple-trace. When eager fertilisation produces a fully instantiated lemma, a counter-example check is applied before attempting to prove the lemma in order to prune out obviously false lemmas. Our proof machinery does not address subgoals which fail to fully instantiate the lemma.

3.3 Restricted Higher-Order Unification

Rewriting using an equation involves finding a unifier between the equation's left hand side and a sub-term of the goal. However, when the goal contains a meta-variable, there is a trivial higher-order unifier between the subterm with the meta-variable at head-position and every rule's left-hand side[3]. The general characteristic of these trivial unifiers is that all arguments of the meta-variable are ignored. Our restricted form of higher-order unification deliberately sacrifices completeness to avoid such unifiers by ensuring some non-variable argument of the meta-variable is unified with a non-variable subterm of the wave rule's left hand side.

Definition 13 (Restricted Higher-order unification).

$$
\begin{aligned}
&\mathit{RestrUnifiers}(t_1[t_1'], t_2, \sigma_1) \equiv \\
&\quad (\ t_1' \equiv f\ \ldots && (3.3)\\
&\quad \wedge\ t_2 \equiv ?X\ t'_{(2,1)} \ldots t'_{(2,i)} \ldots t'_{(2,n)}) && (3.4)\\
&\quad \wedge\ t_1'\sigma \equiv t'_{(2,i)}\sigma && (3.5)\\
&\quad \wedge\ \sigma_1 \equiv \sigma \cup \{?X \mapsto \lambda x_1 \ldots x_i \ldots x_n.\ t_1[x_i]\}\ \ ) && (3.6)
\end{aligned}
$$

Condition 3.3 ensures that there is a subterm of the first argument which does not contain a head-positioned meta-variable, while condition 3.4 requires that the second argument does contain a meta-variable at head position. To illustrate restricted unification, consider a middle-out ripple on the example from goal 3.1: $p\ (?F_1\ (f\ x\ (?F_2\ y)))$, using the equation $g\ (f\ ?u\ ?v) = f\ ?u\ (g\ ?v)$. Restricted unification will have the arguments:

$$
\begin{aligned}
t_1[t_1'] &:= g[(f\ ?u\ ?v)]\\
t_2 &:= ?F_1\ (f\ x\ (?F_2\ y))\\
\sigma_1 &:= \{?v \mapsto (?F_2\ y), ?u \mapsto x\} \cup \{?F_1 \mapsto (\lambda z.\ g\ z)
\end{aligned}
$$

The result is that the goal is rewritten to $p\ (f\ x\ (g\ (?F_2\ y)))$ and the speculated lemma is instantiated from $f\ (h\ x)\ y = ?F_1\ (f\ x\ (?F_2\ y))$ to $f\ (h\ x)\ y = g\ (f\ x\ (?F_2\ y))$.

3.4 Termination of Middle-Out Rippling

The previous implementation of lemma speculation in the CLAM 3 system did not guarantee termination of middle-out steps [15]. A major improvements of our version is the retention of the termination of rippling, even when extended to schematic goals. As discussed in §3.1, this is achieved by recomputing the ripple-measures for the whole trace of schematic goals.

Theorem 4. *The rule for a middle-out ripple-step may only be applied finitely many times to a schematic goal arising from lemma speculation.*

[3] The trivial unifier between a pair of terms t, and $?X a_1 \ldots a_n$ is the substitution $?X \mapsto \lambda x_1 \ldots x_n.\ t$

Proof. Assume we have a trace of schematic goals $[g_n, \ldots g_1]$. By construction, the last goal in the list (g_1) does not contain any meta-variables, and has a fixed ripple measure (by definition 9 and theorem 3). The measure condition (equation 3.2) ensures that there is a substitution such that each step in the trace must decrease the measure with respect to the previous step. The fixed initial measure and well-foundedness of the rippling-measure (theorem 1) thus ensures there is no infinite chain of measure decreasing steps from g_1, and hence that middle-out rippling terminates.

4 Evaluation and Limitations of Lemma Speculation

To evaluate our technique, we considered inductive theorems from: the list-library of Isabelle-2009[4] (223 theorems); theories of Peano arithmetic, including Isabelle's natural number library (46 theorems) as well as a large variety of alternative formalisations from [11] (524 theorems); and all previous theorems known to need lemma speculation (the first 7 theorems in Table 1) [15]. We also examined, in a more ad-hoc fashion, theories concerned with inequalities over natural numbers, ordinal arithmetic, additional properties of fold, append, map, and reverse, and problems from the domain of higher-order function synthesis [10].

The important negative result for lemma speculation is that it is rarely applicable. We found only 18 theorems where it can be applied; 7 examples were in the literature, 5 new examples were found by hand, 1 new example was found in Isabelle's list library, and 5 new examples were found in alternative formalisations of Peano arithmetic. The reason for the limited number of examples is that when rippling is blocked, it is rarely the case that the inductive hypothesis cannot be applied. Tables 1 and 2 highlights a representative set of the results[5].

Another limitation of lemma speculation is that when it is applicable and forms the last step required before fertilisation, no middle-out rewriting steps are applied. This results in the speculated lemma being underspecified, as happens for theorems 7-9 in Table 1. Overall, lemma speculation provides fully instantiated lemmas for 11 of the 18 problems.

Our domains consist largely of equational theorems. While lemma speculation was rarely applicable, lemma calculation was frequently so. In non-equational domains, such as inequality proofs, where lemma calculation after weak fertilisation was not applicable, neither was lemma speculation. Proofs in this domain often require more sophisticated reasoning about assumptions, notably something like the extensions to fertilisation described in [2]. This kind of reasoning may allow proofs to get to the point where lemma speculation is applicable. Domains where there is a higher degree of nesting in term-structures and longer chains of rippling may also have more opportunity for lemma speculation. In these problems, the chances for middle-out reasoning to fully instantiate a lemma is also greater.

[4] http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/library/HOL/List.html

[5] Due to space restrictions, the full results for lemma speculation, including run-times, can be found online: `http://dream.dai.ed.ac.uk/projects/lemmadiscovery/lemspec_results.php`.

	Theorem	**Lemma Speculation**		**IsaCoSy**
		Found Lem	Proved Lem	Proved Thm
1	$x + (Suc\ x) = Suc(x + x)$	yes	yes	yes
2	$even(x + x)$	yes	no	yes
3	$even(len(l\ @\ l))$	yes	no	no
4	$rev((rev\ l)\ @\ m) = (rev\ m)\ @\ l$	yes	yes	yes
5	$rev(rev(l\ @\ m)) = rev(rev\ l)\ @\ rev(rev\ m)$	yes	yes	yes
6	$rev(rev\ l)\ @\ m = rev(rev(l\ @\ m))$	yes	yes	yes
7	$rotate\ (len\ l)\ (l\ @\ m) = m\ @\ l$	fail	-	yes
8	$rev(concat\ l) = concat(map\ rev\ (rev\ l))$	fail	-	yes
9	$len(concat\ (map\ f\ l)) = len(maps\ f\ l)$	fail	-	no
10	$foldl\ (\lambda\ x\ y.\ y + x)\ n\ ((rev\ l)\ @\ m) = foldl\ (\lambda\ x\ y.\ y + x)\ n\ (m\ @\ l)$	yes	no	weak fert.
11	$foldl\ (\lambda\ x.\ y.\ x + (len\ y))\ n\ ((rev\ l)\ @\ m) = foldl\ (\lambda\ x.\ y.\ x + (len\ y))\ n\ (m\ @\ l)$	yes	no	weak fert.
12	$x \leq (y\ +\ x)$	yes	no	yes

Table 1. The sub-columns for lemma speculation indicate when a lemma is found, and when this lemma can be proved automatically. The 'IsaCoSy' column indicates if IsaPlanner can prove the theorem automatically given the background theory produced by IsaCoSy. For Theorems 7-9, lemma speculation fails to fully instantiate the lemma. For theorems 10 - 11, IsaPlanner makes some progress using IsaCoSy's background theory, allowing application of weak fertilisation. The same lemmas as lemma speculation finds can then be produced by the simpler lemma calculation technique.

Finding such problem domains and evaluating lemma speculation on them is left as further work.

A positive result for our version of middle-out rippling is that, as well as working in higher-order domains, it supports speculating the same lemmas as the earlier implementation in CLAM 3. This indicates that the power of middle-out rippling has not been adversely affected by ensuring termination.

4.1 Comparison of Lemma Speculation and Theory Formation

Another approach to automated lemma discovery is to attempt to generate a richer background theory in advance, as done by the IsaCoSy theory formation system [17]. We compared two versions of IsaPlanner's inductive prover: one using lemma speculation and one using just lemma calculation but provided with the lemmas from IsaCoSy's automatically generated background theory. In general, lemma calculation is preferable over speculation, as it has a considerably smaller search space and thus faster run-times. On a 2Ghz, 2GB-RAM desktop PC lemma calculation takes on average less than 1 second while lemma speculation takes up to 137. Automatically generating the background theory uniformly adds all generated theorems as wave rules and may take several hours,

Thm	Speculated Lemma(s)	Synthesised Lemma(s) Used
1	$Suc\ x + y = Suc(y + x)$	$x + Suc\ y = Suc(x + y)$
2*	$x + (Suc\ x) = Suc(x + x)$	$x + Suc\ y = Suc(x + y)$
3*	$len(l\ @\ (h\#l)) = Suc(len(l\ @\ l))$ **or** $len(l\ @\ (h\#l)) = len(h\#(l\ @\ l))$	-
4	$rev\ (h\#l)\ @\ m = rev\ l\ @\ (h\#m)$	$rev(xs\ @\ ys) = (rev\ ys)\ @\ (rev\ xs)$
5	$rev\ (l\ @\ [h]) = h\#(rev\ l)$	$rev\ (l\ @[h]) = h\#(rev\ l)$ **or** $rev(xs\ @\ ys) = (rev\ ys)\ @\ (rev\ xs)$ **and** $(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
6	$rev\ (l\ @\ [h]) = h\#(rev\ l)$	$rev(l\ @\ [h]) = h\#(rev\ l)$
7	-	$(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
8	-	$rev(xs\ @\ ys) = (rev\ ys)\ @\ (rev\ xs)$ **and** $map\ f\ (rev\ l) = rev(map\ f\ l)$
9	-	-
10	$foldl\ (\lambda\ x\ y.\ y + x)\ n\ (l\ @\ (h\#m)) = foldl\ (\lambda\ x\ y.\ y + x)\ h\ (n\#(l\ @\ m))$	$(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
11	$foldl\ (\lambda\ x\ y.\ x + (len\ y))\ n\ (l\ @\ (h\#m)) = foldl\ (\lambda\ x\ y.\ x + (len\ y))\ n\ (h\#(l\ @\ m))$	$(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
12	$Suc\ z \leq a + (Suc\ z) = z \leq Suc(a + z)$	$x + Suc\ y = Suc(x + y)$ **and** $(x \leq (x\ +\ y)) = True$

Table 2. Lemmas discovered by lemma speculation compared to those by syntheses, numbered according to Table 1. Recall that the lemmas speculated for theorems 2 and 3 had to be proved interactively. Using the synthesised background theory results in two alternative proofs for theorem 5, either using the first lemma, or both the others.

but only needs to be done once. Both experiments have the same initial configuration, consisting of only the function definitions; all lemmas had to be discovered automatically. The evaluation set and results are shown in Table 1, with the lemmas shown in Table 2. When given the background theory generated by IsaCoSy, IsaPlanner proves two theorems (7 and 8) that it cannot prove using lemma speculation. Conversely, using the speculated lemmas, theorem 3 can be proved[6]. However, it seems likely that improvements to IsaCoSy will allow generation of lemmas needed to allow lemma calculation to complete the proofs of the theorems 3 and 9, which currently fail. For two theorems (10 and 11) the background theory allows lemma calculation to conjecture the same lemmas as lemma speculation.

It is frequently the case that the lemmas constructed by lemma speculation cannot be proved. This happens for 5 out of 9 speculated lemmas. Speculated lemmas tend to be more specialised compared to lemmas produced and proved by IsaCoSy. This makes the latter generally easier to apply and prove (the proofs are easier because the induction hypothesis is stronger). The lemmas produced by IsaCoSy for theorems 2 and 12, and arguably also 4 and 5, are more general

[6] The lemma was automatically discovered, but had to be proved interactively.

than those found by lemma speculation. The wider variety of general lemmas also result in an alternative proof for theorem 5. Finally, we remark that the background theories generated by IsaCoSy are useful for a wider variety of proofs than those to which lemma speculation is applicable. See [17] for a more detailed evaluation of IsaCoSy.

5 Related Work

Middle-out rippling is an instance of narrowing, i.e. rewriting where both the goal and rule may contain variables instantiated by unification. However, it differs from the account of higher-order narrowing in [22], as we allow meta-variables to occur as arguments to each other in schematic lemmas.

By separating the concepts of annotations, measures and rippling-steps, our formalisation becomes significantly more succinct than that of Basin and Walsh [3]. In particular, it makes the termination proof simpler and supports extensions more easily. The main improvement over the previous versions of middle-out reasoning, e.g. [13, 15], is that ours ensures termination. Our restricted unification heuristic also efficiently cuts out undesirable unifiers, while previous work had to apply filtering of unwanted results after unification [13], or use heuristics specific to rippling [15].

Lemma speculation sometimes fails to fully instantiate schematic lemmas. Previous work attempted to instantiate such a lemma by during the lemma's proof [15]. However, this heuristic works very few examples, and it was thus not implemented in IsaPlanner. Another approach to finding lemmas is *interactive lemma speculation* [16], which can provide the user with some useful feedback, even when the lemma cannot be fully instantiated. A lemma discovery algorithm similar to speculation has been proposed in [18]. Like lemma speculation, subterms in the goal are equated with the hypothesis, and meta-variables used for unknown term-structure. However, we are not aware of any experimental evaluation of this algorithm. Finally, the MATHsAiD [20] and Theorema [14, 4] systems have also been applied to inductive theory formation, although only to natural numbers. The similarity of their results in this domain suggests that if extended to other domains, they would provide similar results to IsaCoSy.

6 Conclusions and Further Work

We have extended techniques for middle-out reasoning and lemma speculation to dynamic rippling and higher-order domains. We also provided a novel, concise and formal account of dynamic rippling, which extends to the middle-out case. The main improvement over previous approaches is the ability to prove termination easily, and extend it naturally to middle-out rippling.

We have implemented middle-out rippling in the IsaPlanner system and performed a practical evaluation in the context of lemma speculation. The lemma speculation technique attempts to construct missing lemmas from failed proof attempts. Contrary to expectations, our results suggest that lemma speculation is not widely applicable. We also showed that, for the otherwise hard problems to which lemma speculation is applicable, theory formation techniques in combination with simpler proof methods offer an effective alternative. Further work includes extending rippling-based proof methods so that other domains can be examined, where lemma speculation may be more applicable, and exploring other applications of middle-out reasoning and theory formation.

References

1. M. Aderhold. Improvements in formula generalization. In *CADE-21*, volume 4603 of *LNAI*, pages 231–246. Springer, 2007.
2. A. Armando, A. Smaill, and I. Green. Automatic synthesis of recursive programs: the proof-planning paradigm. In *ASE-97*, pages 2–9. IEEE CS, 1997.
3. D. Basin and T. Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(1-2):147–180, 1996.
4. B. Buchberger, A. Craciun, T Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkrantz, and W. Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, 4(4):470–504, 2006.
5. A. Bundy. The automation of proof by mathematical induction. In *Handbook of Automated Reasoning*, chapter 13. MIT Press, 2001.
6. A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*. Cambridge University Press, 2005.
7. A. Bundy, L. Dixon, J. Gow, and J. Fleuriot. Constructing induction rules for deductive synthesis proofs. In *CLASE'05*, volume 153, pages 3–21. ENTCS, 2006.
8. A. Bundy, A. Smaill, and J. Hesketh. Turning Eureka steps into calculations in automatic program synthesis. In *UK IT 90*, pages 221–226, 1990.
9. A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7(3):303–324, 1992.
10. A. Cook, A. Ireland, and G. Michaelson. Higher order function synthesis through proof planning. In *ASE-16*, pages 307–310, 2001.
11. L. Dixon. *A Proof Planning Framework for Isabelle.* PhD thesis, University of Edinburgh, 2005.
12. L. Dixon and J. Fleuriot. Higher-order rippling in IsaPlanner. In *TPHOLs-17*, LNCS, pages 83–98. Springer, 2004.
13. J. Hesketh, A. Bundy, and A. Smaill. Using middle-out reasoning to control the synthesis of tail-recursive programs. In *CADE-11*, pages 310–324. Springer, 1992.
14. M. Hodorog and A. Craciun. Scheme-based systematic exploration of natural numbers. In *Synasc-8*, pages 26–34, 2006.
15. A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.
16. A. Ireland, M. Jackson, and G. Reid. Interactive proof critics. *Formal Aspects of Computing*, 11(3):302–325, 1999.
17. M. Johansson, L. Dixon, and A. Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 2010. (To appear).
18. D. Kapur and M. Subramaniam. Lemma discovery in automating induction. In *CADE-13*, LNCS, pages 538–552. Springer, 1996.
19. I. Kraan, D. Basin, and A. Bundy. Middle-out reasoning for synthesis and induction. *Journal of Automated Reasoning*, 16(1–2):113–145, 1996.

20. R. McCasland, A. Bundy, and S. Autexier. Automated discovery of inductive theorems. *Special Issue of Studies in Logic, Grammar and Rhetoric: Festschrift in Honor of A. Trybulec*, 10(23):135–149, 2007.
21. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL - A proof assistant for higher-order logic*, volume 2283 of *LNCS*. Springer, 2002.
22. C. Prehofer. Higher-order narrowing. In *LICS-9*, pages 507–516. IEEE Computer Society Press, 1994.
23. A. Smaill and I. Green. Higher-order annotated terms for proof search. In *TPHOLs-9*, pages 399–413, London, UK, 1996. Springer.