

Adventures in Mathematical Reasoning

Toby Walsh

Abstract *“Mathematics is not a careful march down a well-cleared highway, but a journey into a strange wilderness, where the explorers often get lost. Rigour should be a signal to the historian that the maps have been made, and the real explorers have gone elsewhere.”*

W.S. Anglin, the Mathematical Intelligencer, 4 (4), 1982.

1 Introduction

In 1986¹, I moved to Edinburgh to start a Masters conversion course into Artificial Intelligence after having studied mathematics at the University of Cambridge. I had dreamed about working in AI for many years. So it was my good fortune to fall into the gravitational attraction of Alan Bundy and become a member of the DReaM group. Shortly after, I began a PhD under Alan’s careful supervision² [1, 2, 3].

I would now start to dream about getting computers to do mathematics. It was perhaps not surprising that this was the orbit into which I fell. I had always liked mathematics and now I could combine two of my passions: Artificial Intelligence and mathematics.

I would stay in Edinburgh for most of the next dozen or so years, apart from some enjoyable excursions to work at INRIA in Nancy and with Fausto Giunchiglia at IRST in Trento. There was a lot to like about living and working in Aulde Reekie. However, fresh challenges started to emerge and I began to build up to an escape velocity. I took a research position in Glasgow but stayed living in Edinburgh, visiting the DReaM group frequently. Then I

University of New South Wales, Sydney and Data61

¹ How can it be that long ago?

² I was lucky also to have one of Alan’s postdocs, Fausto Giunchiglia as a second supervisor. We would work together closely for the next decade.

moved to York, and finally a sling shot sent me past Cork to Sydney, Australia where I remain today.

In due course, I would leave behind the work that I had done in Edinburgh and explore other parts of Artificial Intelligence. However, this volume offers me the chance to consider how those ripples might have had a small influence on what followed. More importantly, it lets me thank Alan for his mentoring. This summary is necessarily high level and will avoid going into many of the technical details. I have a lot of ground to cover so it is impossible in this limited space to go deeper.

2 Rippling

Alan was (and is) a neat AI researcher. One strategy he promoted was to take some scruffy research and make it neat. He even gave it a name: undertaking a “rational reconstruction” of some past research. In the late 1980s [4], the DReaM group embarked on a project to rationally reconstruct the rather scruffy inductive theorem proving techniques to be found in Boyer and Moore’s NQTHM theorem prover [5]. Dieter Hutter in Karlsruhe was going about a similar task developing the INKA prover [6] and soon became closely involved with the efforts in Scotland.

Central to the inductive theorem proving heuristics in NQTHM were the ideas of recursion analysis (picking an induction rule and variable), and then rewriting the step case to match the induction hypothesis in a process that became rationally reconstructed as an annotated form of rewriting called “rippling” [7, 8]. Picking an appropriate induction rule and variable depends on how you can simplify the resulting step case so the two ideas are closely connected.

In the early 1990s, rippling ran into some annoying problems. In particular, the annotations used by rippling to guide rewriting could become ill-formed, and there was as yet no principled way to annotate terms in the first place. Alan’s newest postdoc, David Basin and I set about fixing these problems. I shall tell the story backwards as it makes for a more rational reconstruction of the work.

2.1 *A calculus for rippling*

Rippling is a form of rewriting guided by special kinds of (meta-level) annotation. Consider, for example, a proof that appending a list onto the empty list leaves the list unchanged. In the step-case, the induction hypothesis is,

$$(\text{append } x \text{ nil}) = x$$

From this, we need to derive the induction conclusion,

$$(\text{append } (\text{cons } e \ x) \ \text{nil}) = (\text{cons } e \ x)$$

We can annotate the induction conclusion to highlight the differences between it and the induction hypothesis,

$$(\text{append } \boxed{(\text{cons } e \ \underline{x})} \ \text{nil}) = \boxed{(\text{cons } e \ \underline{x})}$$

The square boxes are the “wavefronts”. The underlined parts are the “waveholes”. If we eliminate everything in the wavefronts but not in the waveholes, we get the “skeleton” in which the induction conclusion matches exactly the induction hypothesis.

To simplify the induction hypothesis, we will use a rewrite rule derived from the recursive definition of `append`,

$$(\text{append } (\text{cons } a \ b) \ c) =_{\text{def}} (\text{cons } a \ (\text{append } b \ c))$$

We can turn this into a rewrite rule annotated with wavefronts and waveholes,

$$(\text{append } \boxed{(\text{cons } a \ \underline{b})} \ c) \Rightarrow \boxed{(\text{cons } a \ (\underline{\text{append } b \ c})}$$

This is called a “wave rule”. It preserves the skeleton, $(\text{append } a \ c)$ but moves the wavefront, $(\text{cons } a \ \dots)$ up and hopefully out of the way. Applying this rule to the left-hand side of the induction conclusion gives,

$$\boxed{(\text{cons } e \ (\underline{\text{append } x \ \text{nil}}))} = \boxed{(\text{cons } e \ \underline{x})}$$

We can now simplify the left-hand side using the induction hypothesis as the rewrite rule,

$$(\text{append } x \ \text{nil}) \Rightarrow x$$

This rewriting step is called fertilization, and leaves the following equation,

$$\boxed{(\text{cons } e \ \underline{x})} = \boxed{(\text{cons } e \ \underline{x})}$$

The left-hand side of the rewritten induction conclusion now matches the right-hand side. The step case therefore holds by the definition of equality.

Rewriting annotated terms in this way take us beyond the normal rewriting of terms. David and I therefore formalized a calculus for describing such annotated rewriting [9, 10]. We showed that this calculus had the following four desirable properties.

Well-formedness: rewriting properly annotated terms with wave rules leaves them properly annotated.

Skeleton preservation: rewriting properly annotated terms with wave rules preserves the skeleton.

Correctness: we can perform the corresponding derivation in the underlying un-annotated theory. Annotation is thus merely guiding search.

Termination: given appropriate measures on annotated terms, we can guarantee rippling terminates. There are, for example, no loops.

We showed that different termination orderings can profitably be used within and without induction [11]. Such new orderings let us combine the highly goal directed features of rippling with the flexibility and uniformity of more conventional term rewriting. For instance, we proposed two new orderings which allow unblocking, definition unfolding, and mutual recursion to be added to rippling in a principled (and terminating) fashion.

2.2 *Difference matching and unification*

But where do the annotations come from in the first place? David and I generalized both (1-sided) matching and (2-sided) unification to annotate terms appropriately. Difference matching extended first-order matching to make one term, the pattern match another, the target by instantiating variables in the target, as well as by hiding structure with wavefronts also in the target [12]. Difference unification extended unification to make two terms syntactically equal by variable instantiation and by hiding structure with wavefronts in both terms [13]. Difference unification is needed, for instance, to annotate rewrite rules as wave rules.

A single difference match can be found in time linear in the size of the target. If the pattern contains a variable, then set this to the target and put everything else in the wavefront. If not, the pattern is ground and we can simply descend through the term structure hiding any differences between the pattern and the target in wavefronts. However, there can be exponentially many difference matches in the size of the pattern in general so returning all of them can take exponential time. In practice, though, there are usually only a few successful difference matches and these can be found quickly.

Difference unification is more problematic computationally. Even if we limit wavefronts to one term, deciding if two terms difference unify together is NP-hard (Theorem 8 in [13]). Thus, supposing $P \neq NP$, we cannot in general find even a single difference unifier in polynomial time. Looking again at this result more than 25 years later, I would not leave the analysis there but would look closer at the source of complexity. Difference unification hasn't proved too intractable in practice and we can likely argue why not. The reduction showing NP-hardness reduced a propositional satisfiability problem in m clauses to difference unifying two m -ary functions. The functions being difference unified in this proof therefore can have very great arity. I con-
ject-

ture that difference unification is polynomial, more precisely fixed-parameter tractable, when applied to terms of bounded arity.

An unexpected tale:

To find the difference unification with the least amount of annotation, we proposed a new generic AI search called left-first search (LFS) [13]. Left branches of our search tree introduced annotations, whilst right branches matched terms. Left-first search explored leaf nodes of this search tree in order of the number of left branches taken. I presented the search method at IJCAI 1993.

Two years later, I was listening to an IJCAI 1995 conference talk on a new search method called limited discrepancy search (LDS) [14] when one of the authors put up a slide showing the order of the leaf nodes explored by LDS. This appeared identical to that of LFS, a slide I remember preparing two years before. At the end of the talk, I therefore asked about the difference between the two search methods. A colleague called it the “question from hell” but my intention was just to understand how they differed.

Unbeknown to me, LDS was being patented, and it set off a chain of unfortunate events. Lawyers had to rewrite the patent application at some significant cost. I was asked to be an expert witness in a patent dispute over LDS. And I was considered by the authors of LDS to be a “trouble maker”.

Eventually it would blow over as there is a simple but crucial difference. Our search trees were small and so LFS expanded them in memory. LDS was intended for much larger search trees and so, whilst it expanded leaf nodes in the same order as LFS, did so in a lazy space efficient fashion by returning repeatedly to the root node much like iterative deepening search. This adds just a constant factor to the time asymptotically so is worth paying when space is an issue.

Whilst difference unification was invented to deal with inductive proof, it captures a deeper and more general idea used in mathematics. In [15], J.A. Robinson presented a simple account of unification in terms of difference reduction. He observed,

“Unifiers remove differences ... We repeatedly reduce the difference between the two given expressions by applying to them an arbitrary reduction of the difference and accumulate the product of these reductions. This process eventually halts when the difference is no longer negotiable [reducible via an assignment], at which point the outcome depends on whether the difference is empty or nonempty.”

Difference unification can be seen as a direct extension of Robinson’s notion of difference reduction: we reduce differences not just by variable assignment, but also by term structure annotation. However, what makes this extended notion of unification attractive, is that this annotation is precisely what is required for rippling to remove this difference. And, as we see shortly, rippling has found an useful role to play in a number of other proof areas.

3 Proof planning

An important idea explored within the DReaM group is the separation of logic and control. Proof planning, developed originally for inductive proof [16], brought together ideas of meta-level control explored in the earlier PRESS project [17] with AI planning operators specified by pre- and post-conditions, Theorem proving heuristics are described by general purpose proof planning methods such as rippling and fertilization that are glued together using simple AI search techniques like depth-first or best-first search. Since proof planning was proving useful for inductive proof, I became keen to try to apply it elsewhere.

3.1 *Summing series*

To explore the use of proof planning in general, and rippling in particular outside of inductive proof, I chanced on the domain of summing series [18]. Inductive proofs can be used to verify identities about finite sums. But where do these identities come from in the first place?

I developed a set of proof planning methods to solve such problems. To my surprise, rippling proved to be key to many of these methods. I will illustrate this with the CONJUGATE method. This method transforms a finite sum of terms into the finite sum of some conjugate. 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.

Consider, for example, find a closed form expression for,

$$\sum_{i=0}^n (i+1)x^i$$

The CONJUGATE method transformed this into a simpler looking sum,

$$\sum_{i=0}^n \frac{dx^{i+1}}{dx}$$

This now looks close to a known result, the closed form sum of a geometric series,

$$\sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1}$$

Difference matching our simpler looking sum against the left-hand side of this known result gives some wavefronts we need to remove out of the way by rippling with wave rules,

$$\sum_{i=0}^n \frac{\frac{d}{dx} x^{i+1}}{dx}$$

Since the derivative of a sum is the sum of the derivatives, rippling gives,

$$\frac{d \sum_{i=0}^n x^{i+1}}{dx}$$

Rippling with a wave rule derived from the definition of exponentiation then expands the exponent,

$$\frac{d \sum_{i=0}^n x \cdot x^i}{dx}$$

One final rewriting step uses rippling to move the constant term outside the sum,

$$\frac{d x \cdot \sum_{i=0}^n x^i}{dx}$$

The FERTILIZE method substitutes the closed form sum for the geometric series,

$$\frac{d x \cdot \frac{x^{n+1} - 1}{x - 1}}{dx}$$

Finally a DIFFERENTIATE method then symbolically computes a closed form answer by algebraically differentiating the quotient. The derivation is now complete.

We subsequently looked at some other mathematical domains such as theorems about limits [19]. Rippling and proof planning again proved up to the challenge.

3.2 A divergence critic

Proof planning methods come with high expectations of success. Their failure can therefore be a useful tool in patching proofs. I explored how the failure of rippling can be used to suggest missing lemma needed to complete a proof by means of a “divergence critic” [20, 21]. Other members of the DReaM group have explored similar ideas in closely related settings (e.g. [22, 23]).

My divergence critic identified when a proof attempt is diverging by means of difference matching. The critic then proposed lemmas and generalizations of these lemmas to try to allow the proof to go through without divergence. For example, when the prover failed to show inductively that $(rev (rev x)) = x$, the critic proposed the key lemma, a missing wave rule needed to complete the proof,

$$(rev \boxed{(append \underline{X} (cons Y nil))}) = \boxed{(cons Y (rev X))}$$

In my view, such failure is something we still exploit too little in automating mathematical reasoning. As a mathematician, I spend most of my time failing to prove conjectures. But those failures eventually often lead me to find either a proof when the conjecture is true, or a counter-example when it is false.

4 Mathematical discovery

Mathematics is more than just proving theorems. It’s also defining theories. Inventing definitions. Conjecturing results. Finding counter-examples. Developing proof methods. And more. One of the pleasures of the DReaM group was to witness and contribute to automating some of these other mathematical activities.

In 1996, Alan started to supervise a young and ambitious PhD student, Simon Colton. I was lucky enough to help out. Simon wanted to build a system to invent new theories. Doug Lenat had shown the possibility to do this with AM and the followup Eurisko [24, 25]. Simon set out to rationally reconstruct Lenat’s work in his HR program [26, 27, 28, 29]. This was named appropriately after the famous double act, Hardy and Ramanujan. Actually, HR wasn’t much of a rational reconstruction of AM other than to work on the same problem as AM, and to use a two letter name like AM.

In a wonderful example of why PhD students shouldn’t listen to their supervisors, I suggested to Simon to keep well away from number theory. I reasoned that there had been thousands of years of attention to number theory. New and automated mathematical discoveries were more likely therefore to be found in some newer and little studied theory like that of Moufang

loops. Fortunately, Simon ignored this advice and HR made a number of discoveries in number theory. On the other hand, in a wonderful example of why PhD students should listen to their supervisors, Simon wasn't keen to submit an update on his work to AAAI 2000. I persuaded him to do so, and the paper won the Best Paper award.

5 The meta-level

One of the other rewards of working in the DReaM group was Alan's attention to the meta-level. This wasn't just the meta-mathematical level, but the meta-level of doing research. Alan thought long and hard about how we do research, and how you could do it better. I still recommend the Researcher's Bible that Alan co-wrote to my PhD students whether they're starting out, or writing up their thesis [30]. And when I left Edinburgh, I "borrowed" many of his techniques for doing research on my own: writing half formed ideas down in internal notes, trying to think of questions to ask at every seminar, giving informal talks on any interesting papers I'd seen at summer conferences, etc.

6 Conclusions

Out of interest, I downloaded one of Alan's latest paper [31]. To my surprise and pleasure, it repeats and expands on many of the ideas I've discussed here. It applies rippling to a new domain, invariant preservation proofs. The meta-level guidance rippling provides is used to build proof patches to recover failed attempt and eventually finish the proofs. And the paper ends with an appendix containing a formal definition of rippling, along the lines of the calculus we presented 25 years ago. It feels just like yesterday. Thank you for everything, Alan.

Acknowledgments

Funded by the European Research Council under the Horizon 2020 Programme via the Advanced Research grant AMPLify 670077.

References

1. Walsh, T.: A Theory of Abstraction. PhD thesis, University of Edinburgh (1991)

2. Giunchiglia, F., Walsh, T.: A Theory of Abstraction. *Artificial Intelligence* **56**(2–3) (1992) 323–390 Also available as DAI Research Paper No 516, Dept. of Artificial Intelligence, Edinburgh.
3. Giunchiglia, F., Villafiorita, A., Walsh, T.: Theories of abstraction. *AI Communications* **10**(3,4) (1997) 167176
4. Bundy, A., Van Harmelen, F., Hesketh, J., Smaill, A., Stevens, A.: A rational reconstruction and extension of recursion analysis. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence. IJCAI'89, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1989)* 359–365
5. Boyer, R., Moore, J.: *A Computational Logic*. Academic Press (1979) ACM monograph series.
6. Hutter, D.: Guiding inductive proofs. In Stickel, M., ed.: *10th International Conference on Automated Deduction, Springer-Verlag (1990)* 147–161 *Lecture Notes in Artificial Intelligence* No. 449.
7. Bundy, A., van Harmelen, F., Smaill, A., Ireland, A.: Extensions to the rippling-out tactic for guiding inductive proofs. In Stickel, M., ed.: *10th International Conference on Automated Deduction, Springer-Verlag (1990)* 132–146 *Lecture Notes in Artificial Intelligence* No. 449. Also available from Edinburgh as DAI Research Paper 459.
8. Bundy, A., Stevens, A., van Harmelen, F., Ireland, A., Smaill, A.: Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence* **62** (1993) 185–253 Also available from Edinburgh as DAI Research Paper No. 567.
9. Basin, D., Walsh, T.: A calculus for rippling. In: *Proceedings of CTRS-94. (1994)*
10. Basin, D., Walsh, T.: A calculus for and termination of rippling. *Journal of Automated Reasoning* **16**(1–2) (1996) 147–180
11. Basin, D., Walsh, T.: Termination orderings for rippling. In Bundy, A., ed.: *12th Conference on Automated Deduction, Springer Verlag (1994)* 466–483 *Lecture Notes in Artificial Intelligence* No. 814.
12. Basin, D., Walsh, T.: Difference matching. In Kapur, D., ed.: *11th Conference on Automated Deduction, Springer Verlag (1992)* 295–309 *Lecture Notes in Computer Science* No. 607. Also available from Edinburgh as DAI Research Paper 556.
13. Basin, D., Walsh, T.: Difference unification. In: *Proceedings of the 13th IJCAI, International Joint Conference on Artificial Intelligence (1993)*
14. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: *Proceedings of the 14th IJCAI, International Joint Conference on Artificial Intelligence (1995)* 607–613
15. Robinson, J.: Notes on resolution. In Bauer, F., ed.: *Logic, Algebra, and Computation. Springer Verlag (1989)* 109–151
16. Bundy, A.: The use of explicit plans to guide inductive proofs. In Lusk, E.L., Overbeek, R.A., eds.: *9th International Conference on Automated Deduction. Volume 310 of Lecture Notes in Computer Science., Springer (1988)* 111–120
17. Sterling, L., Bundy, A., Byrd, L., O'Keefe, R.A., Silver, B.: Solving symbolic equations with PRESS. *J. Symb. Comput.* **7**(1) (1989) 71–84
18. Walsh, T., Nunes, A., Bundy, A.: The use of proof plans to sum series. In Kapur, D., ed.: *11th International Conference on Automated Deduction. Volume 607 of Lecture Notes in Computer Science., Springer (1992)* 325–339
19. Yoshida, T., Bundy, A., Green, I., Walsh, T., Basin, D.A.: Coloured rippling: An extension of a theorem proving heuristic. In Cohn, A.G., ed.: *Proceedings of the Eleventh European Conference on Artificial Intelligence, John Wiley and Sons, Chichester (1994)* 85–89
20. Walsh, T.: A divergence critic. In Bundy, A., ed.: *12th Conference on Automated Deduction, Springer Verlag (1994)* 14–25 *Lecture Notes in Artificial Intelligence* No. 814.
21. Walsh, T.: A divergence critic for inductive proof. *Journal of Artificial Intelligence Research* **4** (1996) 209–235

22. Ireland, A., Bundy, A.: Extensions to a generalization critic for inductive proof. In McRobbie, M.A., Slaney, J.K., eds.: 13th International Conference on Automated Deduction. Volume 1104 of Lecture Notes in Computer Science., Springer (1996) 47–61
23. Ireland, A.: Productive use of failure in inductive proof. *J. Autom. Reasoning* **16**(1-2) (1996) 79–111
24. Lenat, D.B., Brown, J.S.: Why AM and Eurisko appear to work. In Genesereth, M.R., ed.: Proceedings of the National Conference on Artificial Intelligence (AAAI83), AAAI Press (1983) 236–240
25. Lenat, D.B., Brown, J.S.: Why AM and EURISKO appear to work. *Artif. Intell.* **23**(3) (1984) 269–294
26. Colton, S., Bundy, A., Walsh, T.: Automatic identification of mathematical concepts. In: Proceedings of 16th IJCAI, International Joint Conference on Artificial Intelligence (1999)
27. Colton, S., Bundy, A., Walsh, T.: Automatic invention of integer sequences. In: Proceedings of the 16th National Conference on AI, Association for Advancement of Artificial Intelligence (2000) 558–563
28. Colton, S., Bundy, A., Walsh, T.: Automatic automatic identification of mathematical concepts. In: Proceedings of the 17th International Conference on Machine Learning. (2000)
29. Colton, S., Bundy, A., Walsh, T.: On the notion of interestingness in automated mathematical discovery. *International Journal of Human-Computer Studies* **53**(3) (2000) 351–375
30. Bundy, A., Du Boulay, B., Howe, J., Plotkin, G.: The researchers' bible. Department of Artificial Intelligence, University of Edinburgh (1985)
31. Lin, Y., Bundy, A., Grov, G., Maclean, E.: Automating Event-B invariant proofs by rippling and proof patching. *Formal Asp. Comput.* **31**(1) (2019) 95–129