The History of the DReaM Group

The History of the DReaM Group

Alan Bundy

Abstract I describe the history of the DReaM Group (Discovery and Reasoning in Mathematics), which I created after my arrival at the University of Edinburgh in 1971. The group has been characterised by its diversity of approaches to the representation of and reasoning with knowledge, including: deduction; meta-level reasoning; learning, especially of new reasoning methods; representation creation and change; as well as applications to problems as diverse as formal verification, analogical blending and computational creativity. From 1982, we have been supported first by a series of EPSRC rolling grants and then, when this funding mechanism ceased, platform grants. Now that the latter mechanism has also ceased, we felt it was time to take stock, celebrate our achievements, assess our strengths and plan our future research. This history lays the bedrock for that self-analysis. Inevitably, space restrictions have forced me to be highly selective in what research I cover. I apologise to those whose excellent research I've had to omit or only hint at. My selection has been mainly influenced by my desire to illustrate our methodological and application diversity. I hope that the other chapters in this book will fill some of those gaps.

1 Why DReaM?

I have been engaged in DReaM Group research for well over four decades. I once turned down a five-fold salary increase in order to continue doing so. I have worked well past my official retirement date. Over 100 fellow researchers have enthusiastically contributed to the Group in those four decades. Why this enthusiasm?

I love mathematics, especially logic. I'm also fascinated with cognition — not especially in the cognition of humans or of other animals, but how cognition is *even possible*. Emulating cognition using logical reasoning, then, ticks all my boxes.

Alan Bundy
School of Informatics, University of Edinburgh, e-mail: A.Bundy@ed.ac.uk

Such emulation can't just be limited to deduction though. It's clear that cognition goes well beyond this and involves a multitude of interacting reasoning processes: abduction; analogy; planning; uncertainty; meta-level reasoning; learning; the use of diagrams; conjecture formation; fault diagnosis; representation formation and evolution; etc.

There is a long-standing argument as to whether computing is a branch of engineering or of science. Of course, it's both. It can also be mathematics. What I have been most drawn to, however, is *computing as art*. Art values beauty, of course, and many computing solutions are beautiful, including ingenious algorithms and elegant representations. For me, though, the main attraction of art is *surprise*: especially pointing to new ways of thinking. As Sir William Lawrence Bragg said:

> "The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them"

Artificial intelligence is well suited to this. It often demonstrates that an aspect of cognition that most people thought was beyond automation can, in fact, be automated. The DReaM Group has had more than its fair share of such demonstrations. We have shown how: intermediate lemmas can be constructed; search control can be reasoned about; proofs can consist of diagrams; interesting conjectures can be made; concepts can be analogically blended; models of the environment can be invented and faulty ones repaired; paintings, music and poems can be created. Even more surprisingly, mathematics, often in the form of logic, can play a key role in the automation of aspects of cognition that seem inherently informal.

It's been a delight to write this chapter and remind myself of these many achievements of the DReaM Group.

2 Arrival in Edinburgh

I first came to Edinburgh in June 1971, having just finished a PhD in Mathematical Logic under Reuben Louis Goodstein at Leicester. I joined Bernard Meltzer's Metamathematics Unit (MMU), which was a major centre for research on Automated Theorem Proving, and was loosely attached to the Department of Machine Intelligence and Perception (DMIP), with which it shared accommodation in Hope Park Square. Bernard had a knack for recruiting excellent people.

Bob Kowalski: had just completed the development of Selected Literal Resolution [42] with Donald Kuehner, and was co-founding the field of logic programming [41].

Pat Hayes: was pioneering John McCarthy's programme of representing common sense reasoning using logic [53].

Bob Boyer & J Moore: J's original PhD project was automating the understanding children's stories supervised by Donald Michie, but switched to automated theorem proving. BobB joined MMU at the same as me and he and J formed

the closest working partnership I've ever known, which led to the Boyer-Moore series of inductive theorem provers [6].

Anecdote 1 (The dangers of being over-rehearsed) *I first met Bernard when he came to give a seminar at Leicester. I wanted to find a job where I could use my logical knowledge but apply it to something practical. So I rehearsed a little speech, which I imagined would go like this?*

Me: "Do you have any logicians in your group?"
Bernard: "No."
Me: "That's what I thought. Are you looking for one?"

Unfortunately, how it actually went was:

Me: "Do you have any logicians in your group?".
Bernard: "No. I suppose that was obvious from my talk."
Me: "That's what I thought. Are you looking for one?"

Fortunately, Bernard immediately knew what had happened and thought it very funny.

I came to Edinburgh at a turbulent time for DMIP. It was composed of three research groups in three separate sites, the heads of which did not get on. MMU became embroiled by the ensuing spate of heated argument and serial reorganisation. This turmoil was fuelled by the Lighthill Report [48], which passed a damning verdict on AI, leading to an AI Winter, during which funding was hard to get. Many researchers left for new pastures, including BobK, Pat, BobB and J. In 1974, out of this ferment, a new, but very small, Department of Artificial Intelligence (DAI) crystallised. To make us viable, the University created two new lectureships. I was lucky to get one and Gordon Plotkin got the other. My first task was to organise a new undergraduate course, called Artificial Intelligence 2[1].

3 The Mecho Project

MMU was funded by a fore-runner of the rolling grant scheme. The funding agency, which was then called SRC[2], sent a panel to inspect us every few years.

Anecdote 2 (Clarity considered harmful) *The attitude of these panels was that they did not fully understand what Bernard's group was doing but thought*

[1] '2' because it was a second year course.

[2] Later to become SERC and then EPSRC.

it was good work that deserved support. For the meeting before I arrived, BobK and Pat had decided that they would make a big effort to get the panel to really understand the work. This succeeded, but the panel said "Ah! Now we understand and we've decided not to fund you".

We got another chance but the pressure was on to convince the panel that our research was worth funding.

Bernard managed to reinvigorate his research group with new people and there was talk of a group project to consolidate us. So at the next SERC panel meeting, I described such a project in which we would solve mechanics problems stated in English. The panel suggested that I submit my own grant proposal to do this. I did, and I got two tranches of SERC funding for a total of six years. We called our program *Mecho* (for Mechanics Oracle). We took A Level applied mathematics problems about particles, inelastic strings, friction-less pulleys, inclined planes, etc., translated the English into a first-order logic representation, then extracted and solved equations [12]. Martha Stone[3], Chris Mellish and Rob Milne did PhDs with me to translate English into FOL. Lawrence Byrd, George Luger and I automated the extraction of simultaneous equations from the FOL, and Bob Welham and I automated solving the equations. In retrospect, this was the beginning of the DReaM Group[4], although we did not call it that then. We were the *Mecho Group*. Martha and Chris both developed international reputations for NLP. George is best known for his successful AI textbook. Laurence joined Quintus, a Silicon Valley, logic program-based start-up founded by David Warren. BobW joined Hewlett Packard in Bristol. Rob became an expert-system entrepreneur in Scotland, but sadly died at 48 climbing Mount Everest — the last of his 'Seven Summits' challenge[5].

BobK had brought Prolog to Edinburgh and, with the work of David Warren at Edinburgh, it became a practical programming language. The Mecho Group adopted it for all the phases of the Mecho program, and it became the biggest Prolog program in the world. Members of the group, including Chris Mellish, Lawrence Byrd, Richard O'Keefe and Leon Sterling, contributed some of the key Prolog textbooks [19, 71, 63].

Meta-level reasoning was a major focus of the Mecho project. It was used to control search in natural language understanding, common sense inference, representation formation and algebraic manipulation. Meta-level reasoning had been much discussed in Bernard's group as a way to impose methodological hygiene by separating object-level reasoning from the heuristics used to control search. It was in sharp contrast to the expert systems then being built, which incorporated heuristics as additional conditions into rules, thus making it hard to separate soundness and completeness from pragmatism. In particular, proof methods, such as isolation, collection and attraction, were used to control the algebraic rewriting used to solve

[3] Now Palmer.

[4] AKA - the Mathematical Reasoning Group.

[5] Climbing the highest peaks in each of seven continents.

equations in our Prolog Equation Solving System, PRESS, which can be seen as an early version of proof planning.

The limiting factor in the Mecho project was deciding how to represent as Physics abstractions the real-world objects described in English. For instance, was a ship to be represented as a particle on a horizontal plane, as used in relative velocity problems, or as a container floating on a fluid, as in Archimedes Principle problems? In school mechanics problems, we found that these choices were hinted at using key words in the English text. This is unsatisfactory. The key skill in engineering modelling is to choose the best representation. Mecho was not the best vehicle for addressing this idealisation issue.

Anecdote 3 (The Importance of Dreaming) *In 1979 I was head hunted by Schlumberger to work on the application of expert systems to the exploration of oil. My wife, Josie, and I were flown out to Ridgefield, Connecticut, wined and dined and offered 5 times my then salary. I turned them down.*

The head of their research lab had a great saying about research methodology. "To do research you need two things:

1. *You have to have a dream.*
2. *And you need to know what to do tomorrow."*

This was one reason for calling ourselves the DReaM Group*, which stands for Discovery and Reasoning in Mathematics.*

4 The Eco Project

As a new lecturer I had to attend a week-long course on teaching methods, which focused on practical skills. One of the exercises was to describe your research project. I chose to describe Mecho. After my talk, I was approached by another new lecturer on the course: Bob Muetzelfeldt. BobM was an ecological modeller in the Department of Forestry and Natural Resources. Ecological models represented the environment using differential equations to describes flows of energy between animals, plants, etc. Writing differential equations was not a common skill among biologists, so BobM's vision was to build a front end which allowed users to describe the ecological environment in biological terms, from which equations could then be automatically extracted. In Mecho, he saw an exemplar of his vision. So, BobM and I joined forces to construct what we decided to call the *Eco System*.

Eco proved to be a good vehicle for addressing the idealisation issue. Idealisation clues could be found, for instance, in the questions the ecologist wanted the ecological model to answer. If the question was about comparing the milk productivity of cattle of different ages, then classifying them into age classes was required. If it was, instead, about the productivity of different breeds, then classifying them by breed

was required. Similarly, the environment might be divided into meadows *vs* pasture *vs* woods — or just represented as a grid.

We got two tranches of SERC funding for two RAs for six years in total [67]. One of the RAs was Mike Uschold, who became a founder of the ontology community and had a successful industrial career. Our initial choice of the second RA proved to be unsuccessful, so we had to replace him. BobM proposed a very bright undergraduate, who proved to be a brilliant choice. Dave Robertson was a key member of our group. At the end of the project, he became a lecturer in DAI, founding his own group on lightweight software engineering. He later went on to become first the Director of our Research Institute, CISA, then Head of the School of Informatics, and is currently the head of the College of Science and Engineering at the University of Edinburgh.

BobM's favoured programming language was Fortran, and Eco's differential equations were represented as difference equations in Fortran and solved numerically. Working with us, however, he became enamoured of Prolog, and switched to that as his main programming language.

The Alvey Programme[6] was a UK response to the Japanese Fifth Generation Project[7]. Alvey initiated an 'Architecture Study', and I was asked to lead a section of this entitled 'Intelligent Front Ends' (IFE). These were user-friendly interfaces to software packages, whose aim was to make these packages accessible to non-computer scientists. Both Mecho and Eco were reinterpreted as IFEs. Another IFE developed in our group was Richard O'Keefe's ASA [62], which aimed to provide an IFE to statistics packages. The idea was for the user to describe an experiment requiring statistical analysis and ASA to suggest appropriate commands in a statistics package.

5 Building a Wider Community

5.1 Rolling Funding and Platform Grants

From 1980, our work on automated reasoning focused more on the PRESS system [70]. We were now calling ourselves the *Press Gang*. Bernard Silver's PhD was on automating the learning of equation solving methods from example solutions. Leon Sterling was developing proof methods for inductive proof. The work was funded by two separate SERC grants. SERC persuaded me to merge these two grants into one rolling funding grant. Rolling grants are reviewed every two or three years by a visiting panel then, if successful, extended for another four years. Entitled "Computational Modelling of Mathematical Reasoning", this first rolling grant started in 1982 and has rolled since then, albeit becoming a platform grant in 2002. This gave

[6] `https://en.wikipedia.org/wiki/Alvey`. Accessed 26.8.19

[7] `https://en.wikipedia.org/wiki/Fifth_generation_computer`. Accessed 26.8.19

our group a great deal of security and the ability to rapidly explore new directions of research.

Anecdote 4 (Virtual Money) *The rolling grants funded not just RAs, but also equipment. We had our own network of computers and a computing officer to support us. That meant we had to decide what to buy. Unfortunately, the group members were frozen by the large sums involved and were unable to make any decisions. I, therefore, proposed that we divide all prices by 100, so that a £2,000 computer appeared to be priced at only £20. That did the trick and we rapidly reached an agreement.*

The rolling and platform grants enabled our group to grow not only in diversity but also geographically. In 1982, the rolling grant had one principal investigator and one site. In 2019, our final platform grant had eleven investigators and included three sites: Edinburgh, Heriot Watt and Queen Mary London. We try to recruit the best postdocs and postgraduate students. I learnt early in my career that the best researchers will soon want to leave and start their own research groups. Rather than be disappointed at losing a key team member, I'm delighted at the opportunity to establish a new collaboration. In practice, this was achieved by adding these pathfinders as new co-investigators to the grant and exploring with them the opportunities for new collaborations. As they took new research paths, this added to the diversity of our research, but the coherence of the group was maintained because many common themes continued to underpin our research.

The range of our research also grew to encompass new areas.

Rippling: In particular, we developed the *rippling* proof method for controlling the step case of inductive proofs. It used *wave rules* that moved and removed differences between the induction conclusion and the induction hypothesis [11] (see §6.2).

The Productive Use of Failure: Automatic analysis of a failed proof attempt can be used to suggest missing intermediate lemmas that enable the proof attempt to continue [36] (see §6.4.1).

Repair of Faulty Representations: Failures of reasoning, e.g., proving false theorems or failing to prove true ones, can be automatically analysed to suggest changes to the language of a theory [57, 46, 47] (see §6.4.3).

Proof Planning: Our development of proof methods and meta-level reasoning crystallised into *proof planning*: specifying proof tactics so that plan formation could be used to construct a plan for a whole proof [8] (see §6.5.1).

Applications of Proof Planning to Formal Methods: Since inductive reasoning is required for reasoning about repetition in both software and hardware, our automation of it could be applied to software verification [37, 51, 49], hardware verification [17] and program synthesis [43, 27, 38] (see §6.6.1).

Applications to Cyber Security: Inductive reasoning was also applied to discover attacks on security protocols [69]. This work later led to work on reverse engi-

neering the implementations of the RSA PKCS♯11 API standard for smart-card protocols [35]. These projects revealed serious security flaws in these protocols (see §6.6.2).

Tactic Learning: By detecting patterns in successful proofs, novel and useful tactics were automatically constructed [68, 31] (see §6.6.3).

Theory Exploration: We developed various heuristic methods for conjecturing and proving interesting theorems [20, 58, 40, 15, 50] (see §6.6.4).

Diagrammatic Reasoning: We pioneered the automation of reasoning by manipulating diagrams rather than logical expressions [39, 72]. This work drew on earlier work on the constructive omega rule [2] (see §6.6.5).

Analogical Reasoning: We have automated analogical reasoning using both ideas of Lakatos and by applying colimits in Category Theory to achieve conceptual blending [66, 52]. These have been evaluated on both mathematics and music (see §6.6.6).

Computational Creativity: Members of our group pioneered computational creativity research [22, 24, 4, 32], with applications to mathematics, computer games, art and music (see §6.6.6).

Representing Uncertainty: We have explored mechanisms for uncertain reasoning in the FRANK System [16], which not only combines deductive with statistical reasoning, but also associates error bars with numerical data and inherits them through the inference process to the final answer. We also developed the Incidence Calculus [7], a kind of probabilistic logic in which sets of weighted possible worlds were associated with formulae rather than associating numeric probabilities directly (see §6.6.7).

To reflect this increasing diversity, when, in 2002, the platform grant series replaced the rolling grants, we changed the title to "The Integration and Interaction of Multiple Mathematical Reasoning Processes". This title change reflected not just the broadening of our scope from deductive reasoning to planning, analogy, learning, diagrams and failure analysis, but also to the ways these processes cooperated to become more than the sum of their parts [9]. We retained our mathematical methodology by requiring rigour in the theory of these reasoning processes, but the domain of application widened to physics, multi-agents, the arts, etc.

5.2 Blue Book Notes and Trip Reports

Early in the history of the DReaM Group, I initiated a series of informal notes for internal communication between group members. The first one was issued in November 1976. They were initially kept in a blue folder and entitled *blue book notes*, abbreviated BBN. Each one has a footnote on the first page which says "Notes in this series are for ε-baked ideas, for $1 \geq \varepsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the Mathematical Reasoning Group". The idea was to encourage Group members to record progress made or problems encountered, however minor. They are typically a few pages long. The purpose of the footnote

was to encourage people to keep a record, but with no quality threshold. Both the upper and lower bound on the value of ε were intended to be taken seriously. The circulation restriction to Group members is intended to encourage people to be frank and open.

I frequently find that when I record progress, the discipline required to explain this to other people helps reveal problems that had not previously occurred to me. Contrariwise, when I record problems, the same discipline helps reveal potential solutions that also had not previously occurred to me. The notes can also sometimes serve as 0^{th} drafts of research papers. BBN 1000 has a more extended discussion of their value. This is a note I *am* prepared to distribute outwith the Group. One measure of their success is that, at time of writing, we have reached BBN 1855 and our 44^{th} year.

At the same time, I initiated a series of trip reports. Group members attending conferences, workshops or bilateral lab visits are strongly encouraged to write a report for the benefit of those Group members who were unable to attend. These are best when they are issued quickly after the trip and focus on aspects of the trip that can't be found in published proceedings, e.g., informal discussions, pointers to related work, suggestions for future work, the future of the conference/workshop. They are not expected to be polished. I've taken to writing them during the talks. I don't find this a distraction. In fact, it helps me identify the key message of the talk so I can summarise it succinctly. Then, by the wonders of the internet, I can even publish them on the trip home. At time of writing, we've reached TR 485.

The trip reports were initially kept in a brown folder. Now, of course, we've foregone hard-copy and both series are mounted on our Group webpages, but still only accessible to Group members. The choice of blue and brown folders was not accidental. You have to know that I am a grand-student of Wittgenstein.

Anecdote 5 (Wittgenstein's Vision) *My PhD supervisor, Reuben Louis Goodstein, was himself supervised by Ludwig Wittgenstein. Goodstein related that Wittgenstein's research group would all go to the cinema. Wittgenstein was short sighted, but too vain to wear glasses. Consequently, he insisted that they all sat in the front row, which was a bit of a struggle for those with regular sight.*

5.3 International Collaboration

We have always been open to collaboration beyond the DReaM Group, especially with European partners. One of the first opportunities to do this was by setting up a network of fellow researchers in inductive theorem proving. Inductive proof is needed to reason about repetition, especially in recursive or iterative programs, but also about recursive data-structure and parameterised hardware. Induction, however,

requires the solution of especially difficult search problems, compared, for instance, to first-order deduction. For instance, even for simple inductive theorems, it may be necessary to discover and prove intermediate lemmas that don't arise as a side-effect of backwards reasoning from the goal theorem. Discovering such lemmas was usually thought to require human intervention, but our productive use of failure work, outlined in §5.1, showed how it could be automated.

In 1992, we received two-year EU ESPRIT funding for the Mechanising Induction (MInd) Network, which funded a series of workshops for 10 EU sites, which we held with a similar sized group of USA researchers. This Network stimulated deeper collaborations with the community of ATM researchers interested in induction. We followed up especially with some of the European sites, using British Council lab-twinning money: first, in 1993, with German sites, especially Saarbücken and Darmstadt then in 1995 with Italian sites in Trento and Genoa. This funding led to the series of CIAO workshops[8], which ran from 1992to 2013. They were initially focused on inductive reasoning, but gradually widened their remit to include most of the areas listed in §5.1. It also funded bilateral lab visits; we benefited greatly both from visitors from the Germany and Italian sites and from having DReaM Group members visiting them. These collaborations led to our involvement in the EU Calculemus Training Network, which brought together ATP researchers with Symbolic Computation ones, and spawned the Calculemus, Mathematical Knowledge Management and Intelligent Computer Mathematics conference series. I've written about this history of European ATP collaboration in more detail in [10].

Anecdote 6 (Cornered at Speed) *I first met Fausto Giunchiglia at IJCAI-89 in Milan, where I was the Conference Chair. I had advertised a vacant RA position and Fausto was very interested in it. He tried to meet with me, but I was so busy with committee meetings, for which IJCAIs are notorious, that we couldn't find a time. He offered to drive me to the airport, which seem like a good solution, but I had reckoned without his outrageous driving. He drove at enormous speed with his head turned to talk to me. To stay alive, I had to keep my eyes focused on the road ahead — and to offer him the job!*

6 DReaM Motifs

The breadth and diversity of the research in our growing and spreading group is outlined in §5.1. What has, nevertheless, held us together are several common motifs.These enable us to continue to relate each branch of our research to the others.

[8] `http://dream.inf.ed.ac.uk/events/CIAO/`

6.1 Rigour and Heuristics

Many of these motifs have their origin in Meltzer's group. For instance, one of my first lessons was the importance of separating rigorous inference and heuristic search. Logical reasoning describes a search space consisting of axioms, the theorems that can be derived from them and the rules that link them together. This search space is typically too large to search exhaustively, so heuristics are used to decide which parts of it to search and in what order. Their role is to optimise the chance that a proof is found of the target conjecture. It is not possible for such heuristics to threaten the soundness of the reasoning.

This organisation facilitates proofs of the soundness of the reasoning and the completeness of the search space. It also enables a distinction between the completeness of both the search space and of its sub-space that is actually searched.

When it is represented as a logical theory, knowledge is given a semantics. Using this semantics, the truth of each axiom in the theory can then be independently established. This gives an assurance that the representation is a faithful account of what it represents.

This methodology now seems to me to be obvious and apt. One can, unfortunately, find many *ad hoc* reasoning methodologies that break it, for instance, by including heuristic conditions in rules.

An exemplar of our concern for rigour is our automation of conceptual blending. 'Houseboat' is an example of a conceptual blend between 'house' and 'boat', in which the boat is regarded as a house. 'Boathouse' is another example, but this time the boat is the occupant of the house. Conceptual blending is a form of analogy and it's tempting to take an *ad hoc* approach to its automation. Our approach to automation, however, is via the concept of *colimit* from Category Theory [52]. Suppose 'house' and 'boat' are each represented as parent logical theories, and there is a third, general, theory of which they are both instances. The general theory can be used to determine which parts of the two parent theories are aligned and which distinct, e.g., is the boat aligned with the house or its occupants? The colimit operation then merges these aligned parts to construct the conceptual blend: houseboat or boathouse.

6.2 Meta-Level Reasoning

We have also extended our desire for rigour to search heuristics. *Meta-level reasoning* is the encoding of such heuristic knowledge as a logical theory in its own right. The theory whose reasoning it is then guiding is called the *object-theory*. The domain that the meta-theory is describing consists of the logical expressions of the object-theory, object-level derivations and the common patterns of search for constructing them.

An illustration of meta-level reasoning is our work on rippling. This proof tactic is used to rewrite a goal formula so that a given formula can be used in its proof.

Rippling's precondition is that the given can be embedded in the goal, i.e., that each symbol in the given can be mapped to one in the goal, so that the nesting of the given is preserved. Table 1 shows a very simple example of such an embedding. The result of rippling is that a subexpression of the transformed goal matches the given.

Given	**Goal**	**Rules**
$a+b=42$	$((c+d)+a)+b=(c+d)+42$	$(x+y)+z=x+(y+z)$
		$u=v \implies w+u=w+v$

Table 1 A Simple Example that Benefits from Rippling

As is usual in ATP, we work backwards from the goal. The associativity equation $(x+y)+z=x+(y+z)$ can be applied to the goal, left to right, in three different ways, only one of which makes progress towards proving it. The three alternative subgoals are:

$$((c+d)+a)+b = c+(d+42) \quad (1)$$
$$(c+(d+a))+b = (c+d)+42 \quad (2)$$
$$(c+d)+(a+b) = (c+d)+42 \quad (3)$$

Subgoals (1) and (2) are unwanted, as they not only make no progress, but one makes the situation worse. Subgoal (3) is the only one that makes progress. The next step will be to apply monotonicity rule right to left[9]. The final subgoal, $a+b=42$, which we will call the *target*, now matches the given. In general, the given will just be a subexpression of the target, not the whole of it.

Using rippling we can prevent the two unwanted subgoals being inferred and only allow the wanted one. The bits of the original goal and rewritten subgoals that don't correspond to the given are annotated with orange boxes, which we call *wave-fronts*. Those bits inside the wave-fronts that *do* correspond to the given are called *wave-holes*. The embedding of the given in the (sub-)goals is highlighted in red. This includes both the contents of the wave-holes and all the other parts of the (sub-)goals that are not in wave-fronts.

A *wave-measure* can then be calculated on these annotations, which strictly reduces on the wanted rewriting (3), but does not reduce on the other two: (1) and (2). The annotated (sub-)goals are shown in Table 2. David Basin and Toby Walsh showed how formulae can be automatically annotated with wave-fronts via a process they called *difference unification* [3].

The wave-measure is calculated as follows. Note that each wave-front can be associated with a unique sub-expression of the given, e.g., $(c+d)+\ldots^{\uparrow}$ is associated with 42. If we write the given as a tree, then it has three levels: $=$ is at the top level, $+$ and 42 are at the middle level and a and b at the bottom level. We sum up how many orange boxes occur at each of the three levels. For instance, in the

[9] Recall that we are reasoning backwards from the goal.

Label	(Sub)Goals	Measure
Goal	$(\overset{\uparrow}{(c+d)+a})+b = \overset{\uparrow}{(c+d)+42}$	$[1,1,0]$
(1)	$(\overset{\uparrow}{(c+d)+a})+b = \overset{\uparrow}{c+(d+42)}$	$[1,1,0]$
(2)	$\overset{\uparrow}{(c+(d+a))+b} = \overset{\uparrow}{(c+d)+42}$	$[1,2,0]$
(3)	$\overset{\uparrow}{(c+d)+(a+b)} = \overset{\uparrow}{(c+d)+42}$	$[0,2,0]$
Target	$a+b = 42$	$[0,0,0]$

Table 2 An Example of Rippling Annotation

original goal, one box is associated with a and one with 42. So one is at the bottom level, one in the middle level and none at the top level. We order these scores bottom to top and write these scores in a triple $[1,1,0]$, which is the measure of the original goal. The scores of the three subgoals are given in Table 2.

Anecdote 7 (Orange is a Strange Colour) *Why were the wave-fronts orange? The research started before the advent of data-projectors. We used felt tip pens on acetate sheets on overhead projectors. By trial and error we discovered that orange was a transparent colour, so that the formulae were visible through the orange ink.*

Unfortunately, my PhD student, Pete Madden, did not get the message and used red ink instead. The red obscured the formulae, making his talk difficult to follow.

The aim is to move the wave-fronts up through the levels and, ideally, to remove them altogether, so that the given appears as an unannotated sub-expression of the final sub-goal[10]. The given can then be used to complete the proof. We, therefore, order the measure tuples lexicographically and we require rippling to strictly decrease the measure. So, $[1,1,0]$ is larger than $[0,2,0]$ but smaller than $[1,2,0]$. We now see that subgoal (1) leaves the measure unchanged, subgoal (2) increases it and only subgoal (3) decreases it. Applying the monotonicity rule now removes all wave-fronts and the given can be used to prove this final subgoal and complete the proof.

The rewrite rules can also be annotated with wave-fronts. We then call them *wave-rules*.

[10] The metaphor is of ripples on a pond. Initially, they obscure the reflection of the surrounding countryside, but as the ripples move out, the reflection is restored.

Wave – Rules	LHS	RHS
$(x+y^{\uparrow})+z \rightarrow x+(y+z)^{\uparrow}$	$[1,0]$	$[0,1]$
$w+u^{\uparrow} = w+v^{\uparrow} \rightarrow u=v$	$[2,0]$	$[0,0]$

The **LHS** and **RHS** columns show the wave-measures of the left-hand and right-hand sides of the rules. Note that the wave-measures strictly decrease from left to right. Rewrite rules can also be automatically annotated as wave-rules so that the wave-measures strictly decrease. During application of the wave-rules, not only must the object-level expressions match, but so must the meta-level wave-annotations. In this way, rippling is completely automated and is guaranteed to terminate.

In general, rewrite rules can be annotated as wave-rules in multiple ways. For instance, $(x+y)+z = x+(y+z)$ can also be annotated as $x+(y+z^{\uparrow}) \rightarrow (x+y)+z^{\uparrow}$. Even though this wave-rule is directed in the opposite direction from the previous annotation, rippling will not loop. This is because the wave-annotation must match. So, for instance, this reverse-directed wave-rule is not applicable to the goal in Table 2 — nor to any of the sub-goals.

6.3 Why Prolog?

As mentioned in §3, Prolog was our implementation programming language of choice for the Mecho project, and for many subsequent projects until relatively recently. Partly, this can be explained by the strong Edinburgh involvement in (a) the vision of computational logic and (b) the development of Prolog as a practical programming language. There is, however, more to it than that. Prolog is a conducive vehicle for developing meta-theories.

6.3.1 Meta-Level Axioms for Rippling.

For instance, rippling as the successive application of wave-rules to annotated goals, can be recursively defined by the meta-theory axioms:

$$Subterm(given, goal) \implies Ripple(given, goal, goal)$$
$$Embed(given, goal) \wedge Wave(wrule, goal, subgoal) \wedge Ripple(given, subgoal, target) \implies Ripple(given, goal, target)$$

where;

- $Ripple(g, g_1, g_2)$ means that subgoal g_2 is the result of applying rippling wrt given g to goal g_1;
- $Subterm(given, goal)$ means that $given$ is a subterm of $goal$;

- $Embed(given, goal)$ means that $given$ is embedded in $goal$; and
- $Wave(wrule, g_1, g_2)$ means that subgoal g_2 is the result of applying wave-rule $wrule$ to goal g_1.

These axioms can be readily expressed as the Prolog program:

```
ripple(Given,Goal,Goal) :- subterm(Given,Goal), !.
ripple(Given,Goal,Target) :- embed(Given,Goal),
                             wave(Wrule,Goal,Subgoal),
                             ripple(Given,Subgoal,Target).
```

Functional languages, such as ML and Haskell, are widely used to implement tactics in ATP. We have used functional and other languages in recent work. Logic programs, however, seem especially well suited to meta-level reasoning. Compare for instance, this functional definition of *Ripple*.

$$Subterm(given, goal) \implies Ripple(given, goal) = goal$$
$$Embed(given, goal) \implies Ripple(given, goal) = Ripple(given, Wave(wrule, goal))$$

Unfortunately, *Ripple* isn't a total function. It may return several results or none. Representing it instead as a relation addresses the absence of both the existence and uniqueness properties that define functionality.

6.3.2 Meta-Level Axioms Attraction.

Similarly, the meta-level axioms that define the isolation, collection and attraction equation-solving methods of PRESS (see §3) can be presented relationally. Consider, for instance, *Attraction*, which moves occurrences of the unknown closer together so that they can be collected into one occurrence and then this occurrence can be isolated on the LHS of the equation. Some example attraction rewrite rules are given in Table 3. When applying these rules, the meta-level condition is that u

Attraction Rules	LHS	RHS
$Log_w u + Log_w v \rightarrow Log_w u.v$	4	2
$w.u + w.v \rightarrow w.(u+v)$	4	2
$(w^u)^v \rightarrow w^{u.v}$	3	2
$u^{v.w} \rightarrow (u^v)^w$	3	2

Table 3 Examples of Attraction Rules

and v be instantiated to terms containing the variable, say x, which is to be attracted, but w must *not* be instantiated to a term containing x. These applications will have the effect of moving the occurrences of x closer together. The distances between u and v are defined to be the size of the smallest sub-expression that contains both of them. These least sub-expressions are highlighted in red in Table 3.

The attraction proof method can be defined by the following meta-level axioms:

$$Collect(x, goal, target) \implies Attract(x, goal, target)$$
$$AttractRule(x, arule, goal, subgoal) \wedge Attract(x, subgoal, target) \implies Attract(x, goal, target)$$

where:

- $Attract(x, g_1, g_2)$ means that subgoal g_2 is the result of attracting x in g_1;
- $AttractRule(x, arule, g_1, g_2)$ means that g_2 is the result of applying attraction rule $arule$ to attract occurrences of x in g_1.
- $Collect(x, g_1, g_2)$ means that subgoal g_2 is the result of collecting x in g_1;

We can interpret these rules as saying that we keep rewriting the goal with attraction rules until either collection can be applied (exit with success) or we can no longer find any applicable attraction rules (exit with failure).

6.4 The Productive Use of Failure

Failure to find a proof at the first attempt is not necessarily a reason to give up. An analysis of the cause of failure can suggest a proof repair. For instance, it might suggest trying to prove an intermediate lemma that would help move the initial proof on. It might also suggest a change of representation that would enable a wanted proof to succeed (or an unwanted one to fail).

The productive use of failure has been a recurring theme in the *DReAM* group, not just in repairing broken proofs, but also in repairing broken representations, plans and programs. In particular, we have an especial interest in the methods suggested by Lakatos to cope with counter-examples to conjectures [44, 25].

6.4.1 Suggesting Intermediate Lemmas.

Rippling can fail if, at some point, there is no wave-rule available to move the wave-fronts up to the next level. When this happens, we know a lot about the wave-rule we would like to have. This can often be enough to identify the wave-rule we would like and to prove it as a new lemma to complete the proof. We call this *the productive use of failure*. That is, we analyse the failure to work out how to unblock the failed proof attempt.

Here is a very simple example. In Peano Arithmetic [65], addition is defined recursively by the following axioms[11]:

[11] These axioms follow Peano's spirit, but have been modified according to modern practice.

$$x+0 = x$$
$$x+succ(y) = succ(x+y) \tag{4}$$

where *succ* is a function used to construct the natural numbers, i.e., $succ(0)$ represents 1, $succ(succ(0))$ represents 2 and so on.

Suppose we now try to prove the commutativity of $+$, i.e., $m+n = n+m$. The proof will be by structural induction. Since the goal is symmetric in m and n, it doesn't matter which we choose as the induction variable, say m. The step case of the induction proof is:

$$m+n = n+m \implies succ(m)+n = n+succ(m)$$

Rippling is an ideal proof tactic for the step cases of inductive proofs. The given is the induction hypothesis $m+n = n+m$ and the goal is the induction conclusion $succ(m)+n = n+succ(m)$. Annotating the goal with wave-fronts gives:

$$\boxed{succ(\boxed{m})}^{\uparrow} + n = n + \boxed{succ(\boxed{m})}^{\uparrow} \tag{5}$$

and annotating (4), the step case of the recursive definition of $+$, gives:

$$x + \boxed{succ(\boxed{y})}^{\uparrow} \rightarrow \boxed{succ(\boxed{x+y})}^{\uparrow} \tag{6}$$

Unfortunately, wave-rule (6) applies only to the RHS of (5) to give:

$$\boxed{succ(\boxed{m})}^{\uparrow} + n = \boxed{succ(\boxed{n+m})}^{\uparrow} \tag{7}$$

The rippling process is now blocked as we lack a wave-rule to ripple the LHS.

We know a lot about the missing wave-rule. We want its LHS to match the LHS of (7) and we want it to ripple the LHS wave-front out so that it surrounds the wave-hole $m+n$. That is, it should have the shape:

$$\boxed{succ(\boxed{x})}^{\uparrow} + y \rightarrow \boxed{\mathscr{F}(\boxed{x+y})}^{\uparrow} \tag{8}$$

where $\mathscr{F}$ stands for the, as yet, unknown contents of the RHS wave-front. We can represent $\mathscr{F}$ as a meta-level variable standing for an unknown object-level term. It remains to instantiate $\mathscr{F}$ with a ground term that will enable the remaining proof to succeed.

To complete this blocked proof, two subgoals remain: (i) prove the missing wave-rule (8) as a lemma; and (ii) complete the rippling of the step-case. Second-order unification can be used to instantiate $\mathscr{F}$ as a side effect of either of these remaining proof subgoals. In either case, $\mathscr{F}$ will be instantiated to *succ*, so that the missing wave-rule is:

$$\boxed{succ(\boxed{x})}^{\uparrow} + y \rightarrow \boxed{succ(\boxed{x+y})}^{\uparrow}$$

and the next rippling step is:

$$succ(m+n)^{\uparrow} = succ(n+m)^{\uparrow} \tag{9}$$

The ripple can then be completed with the monotonicity axiom of $=$, considered as a wave-rule[12]:

$$succ(x)^{\uparrow} = succ(y)^{\uparrow} \rightarrow x = y \tag{10}$$

which reduces the rewritten induction conclusion to the given, which is the induction hypothesis.

6.4.2 Lakatos Methods and Counter-Examples.

In [44], Lakatos describes a rational reconstruction of the history of Euler's Theorem $V-E+F=2$ about polyhedra, where V is the number of vertexes, E the number of edges and F the number of faces. Lakatos's aim was to describe the evolution of mathematical methodology via the evolution of techniques to deal with counter-examples to false conjectures. An initial 'proof' is given due to Cauchy. Counter-examples are then discovered and a succession of repair techniques are described. They include, for instance, *strategic withdrawal* and *counter-example barring*. In strategic withdrawal a key concept, e.g., the definition of polyhedra, is specialised to exclude some counter-examples. In counter-example barring, the negation of a concept describing some counter-examples is made into a new precondition of the conjecture.

Simon Colton and Alison Pease's Theorem Modifier (TM) [25] repaired false conjectures using some of Lakatos's methods. TM combines: the Mace counter-example finder [56] to find counter-examples to false conjectures; Colton's HR system [21] to learn, from examples and non-examples, the new concepts needed for strategic withdrawal and counter-example barring; and the OTTER theorem prover [55] to confirm that the repaired conjecture is now a theorem. An example is given in Fig. 1.

6.4.3 Suggesting Changes of Representation.

Suppose a representation of the environment is faulty. We will consider two cases:

Incompatibility: The faulty representation enables us to prove (i.e., predict) something that is false (i.e., not consistent with our observations).

Insufficiency: The faulty representation fails to prove something that we observe to be true.

[12] If you are concerned with the direction of the rewrite arrow, recall that we are reasoning backwards, so the direction of implication is right to left.

TM was given the following faulty conjecture in Ring Theory:

$$\forall x, y.\ x^2 * y * x^2 = e$$

where e is the multiplicative identity element.

Mace found 7 supporting examples and 6 counter-examples to this conjecture. Given these two sets of examples, HR invented a concept that can be simplified to:

$$\forall z.\ z^2 = z + z$$

and used it for strategic withdrawal. Otter was then able to prove the original conjecture for just those rings with this new property.

Fig. 1 Correcting a Faulty Conjecture in Ring Theory

These two cases are dual and can be addressed in a symmetric way. Most research into representation repair works by either deleting axioms (belief revision [33]) or adding axioms (abduction [26]). These are sometimes the most appropriate repair, but sometimes it is preferable to change the *language* of the representation, i.e., the signature of the theory. In [47], Xue Li, Alan Smaill and I applied the *Reformation algorithm* to suggest such signature changes, for instance: splitting or merging of predicates or constants; adding or removing arguments to predicates. The ABC system described here combines abduction, belief revision and Reformation. For technical reasons, our implementation is currently limited to Datalog-like logical theories [18]. The axioms are Horn clauses with no functions except constants. Any variables in the head of a clause must also appear in the body.

Consider, for instance, the following Datalog theory, $\mathbb{T}$, which is adapted from [33]):

$$\begin{aligned} German(x) &\implies European(x) \\ European(x) \wedge Swan(x) &\implies White(x) \\ \implies German(Bruce) \qquad & \implies Swan(Bruce) \end{aligned}$$

From $\mathbb{T}$ we can infer that $White(Bruce)$:

$$\dfrac{\dfrac{\dfrac{White(Bruce) \implies}{European(Bruce) \wedge Swan(Bruce) \implies}\ European(x) \wedge Swan(x) \implies White(x)}{German(Bruce) \wedge Swan(Bruce) \implies}\ German(x) \implies European(x)}{\dfrac{Swan(Bruce) \implies}{\implies}\ \implies German(Bruce)}\ \implies Swan(Bruce)$$

This proof is by Selected Literal Resolution (SL) [42], which, for Horn clauses, has the convenient property that each resolution step is between an axiom and a goal. This has the advantage that we can apply any repair directly to the axiom involved in either the current or an earlier SL Resolution step in the current branch, so we do not need to inherit the repair back up through derived clauses to an axiom. Each

axiom in the proof is given on the right-hand side of the inference step. Each proof step unifies a literal in a goal with a literal in a fact or rule head. We have highlighted these literals in brown or red.

We use Kowalski form to present clauses:

$$P_1 \wedge \ldots \wedge P_m \implies Q_1 \vee \ldots \vee Q_n$$

For Horn clauses, $n = 0$ or $n = 1$. When $n = 0$, the Horn clause is a goal. When m is also 0, it is the empty clause. The clauses on the left-hand side of the proof are all goal clauses.

Suppose, however, that we observe that the swan *Bruce* is not white but black. The proof above proves something that is false, i.e., $\mathbb{T}$ is incompatible. This proof is unwanted and must be broken. In [33], Gärdenfors suggests repairs to the axioms, e.g., adding an exception to one of the rules:

$$x \neq Bruce \wedge European(x) \wedge Swan(x) \implies White(x)$$

But this is a unsatisfying hack. A better repair is to split *European* into a European type of object or a European resident. *Bruce* is *not* a European type of swan, but *is* resident in Europe. The proof can then be broken at the red unification step. Reformation is based on the unification algorithm. In the case of an incompatibility it can break a successful unification and in the case of insufficiency it can enable an unsuccessful unification.

In our example, the red unification is broken by adding an extra argument to *European* and ensuring that this argument will be instantiated to different constants when the two literals are to be unified, thus preventing it from succeeding. The two constants are named generically *Normal* and *Abnormal*. Reformation is a purely syntactic algorithm and we don't currently have the semantic knowledge to call them, say, *Type* and *Resident*.

The repaired theory $\nu(\mathbb{T})$[13] is:

$$\begin{aligned} German(x,y) &\implies European(x,y) \\ European(x,Normal) \wedge Swan(x) &\implies White(x) \\ \implies German(Bruce,Abnormal) \qquad &\implies Swan(Bruce) \end{aligned}$$

where the new arguments are high-lighted in red. Note that the constraints of Datalog force *German* to also be given a new argument, because any variable in the head of a clause must also appear in the body.

[13] Pronounced 'new $\mathbb{T}$'.

6.5 The Interaction of Multiple Reasoning Processes

As outlined in §5.1, the DReaM Group has always had an interest in a wide range of reasoning processes: deduction, learning, abduction, analogy, statistics, diagrams, failure analysis, representation formation, change of representation, meta-level reasoning, creativity, etc. We are especially interested in how different reasoning processes interact [9]. Some of these interactions have already been discussed. For instance, §3 and §6.3.2 explain how meta-level reasoning was used to control search in natural language understanding, common sense inference, representation formation and algebraic manipulation. §6.2 and §6.3 explain how meta-level reasoning can guide object-level deduction. §6.4 describes how failure analysis can suggest new lemmas and, thereby, aid deduction, and how it can suggest changes of representation. §6.4.2 describes the combination of counter-example finding, concept learning and theorem proving to repair faulty conjectures.

In this section, we briefly summarise some of the other interactions we have pioneered.

6.5.1 Proof Planning: Abstraction of Deduction

Human mathematicians often report having a plan of a proof before tackling the detailed steps. This plan helps them guide the search for a proof. It is not bound to succeed, so may require re-planning to deal with unexpected obstacles. *Proof planning* is an attempt to automate this process [8]. Proof planning was originally implemented in Prolog, by Christian Horn and Frank van Harmelen, as the *CLAM/Oyster* system [14]. This was later upgraded to an implementation in λProlog, called *λCLAM* [29]. The current implementation is Lucas Dixon's IsaPlanner [30], which is built on top of Isabelle.

Anecdote 8 (A Language too Far) *On paper, λProlog looked like the ideal implementation language for our proof planner. For instance, its built-in, higher-order unification was just what we needed for instantiating meta-variables when speculating missing intermediate lemmas (see §6.4.1). It was, however, an experimental prototype complete with bugs, but without adequate maintenance support. So, when we inevitably stumbled on bugs, there was no one to fix them. At one stage, we discovered that inclusion of some redundant λProlog code could inexplicably cause a broken λCLAM to start working again. If it subsequently stopped working, then commenting out the redundant code would effect another temporary fix.*

Proof tactics are specified, in a meta-logic, with preconditions to describe when they are applicable and postconditions to describe their effect. For example, the specifications of the wave tactic from §6.3.1 are:

Preconditions:

1. The given embeds into the goal.
2. There is a wave-rule that matches the goal and produces a new subgoal.
3. Any preconditions of that wave-rule are provable.

Postconditions: These postconditions are guaranteed by the successful application of a wave-rule.

1. The given also embeds into the new subgoal.
2. The wave-measure of the subgoal is strictly smaller than that of the goal.

Such specifications enable a plan to be automatically constructed by matching the preconditions of a later tactic to the postconditions of an earlier one.

Proof plans are hierarchical, i.e., a tactic can be defined in terms of subtactics, which may include recursive calls to itself. This aids understanding, as a proof can be inspected at the top level and optionally unpacked along one or more branches of the plan. We can express this hierarchical structure as a *hiproof* [28]: a graph in which the nodes are hiproofs. An example hiproof is displayed graphically in Fig. 2.

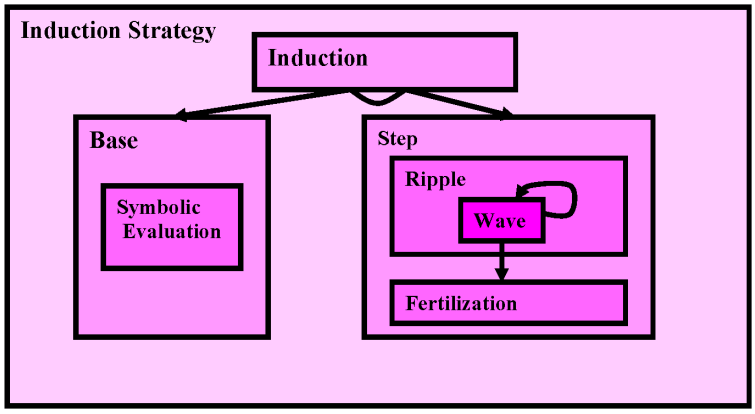


Fig. 2 A Hiproof of an Inductive Proof Plan

The specifications of tactics facilitate recovery from failure [36]. Failure occurs if the preconditions of a plan's tactic are not satisfied. Re-planning can then take place to bridge the gap between the effects of successful tactic applications and those needed by the failed tactic or by an alternative tactic. Andrew Ireland and I showed that different patterns of precondition failure suggest different kinds of plan repair. In inductive proofs, for instance, the analysis of failure can suggest: intermediate lemmas (see §6.4.1); generalisations of the conjecture, changes of the

induction rule; and case analyses. Failure of precondition 2, for instance, suggests a missing intermediate lemma that could serve as the required wave-rule. On the other hand, if there is a wave-rule that partially matches, in the sense that only the wave-front fails to match, then this suggests a change of induction rule in order to ensure that the wave-front also matches.

Proof planning is just one method for abstracting proofs. In [34], Fausto Giunchiglia and Toby Walsh developed a theory of the many different forms that abstraction has taken in automated reasoning. Typically, a theory and conjecture are abstracted by ignoring some details. A proof is then found, we hope more easily, in this abstract theory. For instance, a first-order theory might be abstracted to a propositional one. This abstract proof is then used as a plan to construct a proof in the original theory.

6.6 Diverse Applications

The many techniques we have developed within the DReaM Group have a wide diversity of applications. They range from formal methods to cognitive science, via cyber security, mathematics, uncertainty, diagrammatic reasoning, analogy and interestingness — always combining rigour with heuristics. We highlight a few of these.

6.6.1 Formal Methods

Proof planning has been applied to formal verification and synthesis of both software and hardware. In particular, inductive proof is usually required whenever the software uses recursion or loops, and when the hardware is parameterised, e.g., an n-bit multiplier. One of the hardest problems in software verification is finding a loop invariant. There is a strong relationship between loop invariants and induction rules, so Andrew Ireland adapted our techniques for suggesting induction rules to suggest loop invariants [37]. Yuhui Lin used ripple failures to suggest intermediate lemmas in Event-B verification proofs [49]. As mentioned in §5.1, we have also explored a variety of techniques for program synthesis [43, 27, 38].

6.6.2 Cyber Security

Graham Steel's Coral system [69] found attacks on security protocols for group key agreement by refuting incorrect inductive conjectures. This approach avoided the need for abstraction to a group of fixed size, which can miss attacks on larger groups. By posing inductive conjectures about the trace of messages exchanged, we could investigate novel properties of the protocol, such as tolerance to disruption, and whether it results in agreement on a single key. This has allowed us to find

three distinct novel attacks on groups of size two and three. This work led to the successful spin-out Cryptosense[14].

Anecdote 9 (Just in Time) *At 1am on 8th December 2002, Graham was just finishing his PhD. Walking home from a night out when he passed our offices at 80 South Bridge and saw that they were on fire. He ran to his flat further along South Bridge and, sitting in his bay window facing onto the fire, he logged onto the servers in the basement of number 80 and downloaded his nearly complete PhD thesis. Minutes later the servers melted in the heat. The fire engulfed 11 buildings and burnt for 52 hours before being completely extinguished. All the occupants of 80 South Bridge were relocated to The Appleton Tower.*

Cryptosense proposed a project to discover security flaws in smart cards [35]. Prior work had searched for flaws in the commonly used RSA PKCS♯11 API standard for smart-card protocols. In contrast, the REPROVE system automatically reverse engineered the low-level implementations of PKCS♯11. Proprietary implementations are often used in an attempt to obtain security through obscurity. We not only showed that such implementations could be automatically decoded, but our analysis revealed extremely serious, previously unknown flaws in these implementations that severely compromised security.

A DReaM Group member, David Aspinall, is the founder of The Edinburgh Cyber Security, Privacy and Trust Institute[15]. This is a multidisciplinary research and teaching network at The University of Edinburgh.

Anecdote 10 (Men *vs* Machines) *The Appleton Tower had won many ugliest architecture prizes. To house us, it was refurbished floor by floor — turning what had been lecture rooms into offices. We were then shuffled from floor to floor. Andrew Ireland visited me when I was on the 2nd floor and the 3rd was being refurbished. We used the lift to go to the basement, but Appleton Tower lifts have a life of their own, so it took us to the 3rd floor instead. The doors opened on wooden boarding which prevented access to what was a building site. The doors then stuck open, which prevented the lift from moving. We were imprisoned. Fortunately, I knew exactly where to stamp to persuade the doors to close. Men 1; machines 0.*

[14] `https://cryptosense.com/`

[15] `https://www.ed.ac.uk/cyber-security-privacy`

6.6.3 Machine Learning of Proof Tactics.

The proof methods used in meta-level reasoning were manually designed. This raises the question as to whether they could be learnt from examples. Bernard Silver described the application of precondition analysis successfully to learn the equation-solving methods of isolation, collection, attraction, homogenisation, etc. from example solutions [68]. The implementation was called Learning PRESS (LP). Unlike statistical machine learning techniques, precondition analysis can learn a new proof method from a single example. Like explanation-based generalisation, it generalises an example by abstracting away from the specific details of the example to construct a method that will work on similar problems. To aid it in abstraction it has available the language of the PRESS meta-level theory.

Hazel Duncan used a statistical approach to learn Isabelle [64] tactics from proofs in Isabelle's librariesIn [31]. Proofs were abstracted into sequences of proof steps. Variable-length Markov models were then used to identify patterns that occurred more frequently than chance. These patterns were then combined by genetic programming, using a grammar of loops and splits, to form tactics. This enabled a pool of simple tactics to be formed.

6.6.4 When is a Theorem Interesting?

Most applications of automated reasoning start with a conjecture and reason backwards to the axioms. If you know what theorem you want to prove, this works fine. But suppose you just want to explore a theory to discover interesting theorems? This might be a useful aid for mathematicians to explore the foothills of a new theory, while they have a coffee, to see whether it is worth taking further. Also, as an alternative to discovering new lemmas on an as-needed basis (see §6.4.1), you might build up a library of useful lemmas in advance.

Discovering new theorems is trivial — just reason forward from the axioms. The trick is to discover ones that are *interesting*. But what do we mean by interesting? We have conducted several experiments to address this question.

- Simon Colton compared and contrasted the measures of interestingness used in 5 machine discovery systems, extracting general principles of truth, novelty, surprisingness, non-triviality and understandability [23]. He then applied one of these 5 systems, his HR system, to the automatic invention of interesting integer sequences, seventeen of which were deemed interesting enough to have been accepted into Sloane's "Encyclopedia of Integer Sequences" [22].
- Moa Johansson's IsaCoSy [40] and Omar Montano-Rivas's IsaScheme [58] both built systems for finding interesting theorems in recursive, equational theories. They used different interestingness measures, but still got surprisingly similar empirical results.

  IsaCoSy generated only equational conjectures between irreducible terms, i.e., those that a set of rewrite rules cannot simplify any further. Obvious non-

theorems were rejected using a counter-example checker. Newly discovered theorems were turned into rewrite rules and added to the simplifier. This ensured that new theorems could not be proved just by simplification, but required a more powerful proof method, such as induction.

IsaScheme used schemas representing common patterns of interesting theorems, such as associative, distributive and idempotency laws. New theorems were also normalised with Knuth-Bendix completion.

- MacCasland's MATHsAiD system [54] explored algebraic theories. It tried to find a balance between simplicity and non-triviality, that is, theorems were interesting iff they could be proved in a few steps but using at least one non-trivial method, e.g., simplification is trivial but induction is not. It was able to prove theorems connecting two more more theories, including one about Zariski spaces that McCasland had proved in a previous career as an Algebraist.

Evaluation of exploration system is challenging because there is no agreed criteria for what counts as interesting — indeed, that's what the research is aiming to investigate. The researcher's above used comparisons with previously explored theories. For instance, for IsaCoSy and IsaScheme, comparison between their outputs and Isabelle's libraries showed high values for precision and recall. For MATHsAiD, a comparison of its outputs and the theorems selected for discussion in several standard algebra textbooks showed a greater agreement between the MATHsAiD and each of the textbooks than between the textbooks themselves.

IsaCoSy is one component of the TheoryMine[16] spin-out [13]. Given an automatically-generated, novel, recursive theory, it generates interesting, novel theorems, that customers can name.

6.6.5 The Constructive Omega Rule, Induction and Diagrams.

The ω-*rule* is a complete, but infinitary, alternative to mathematical induction.

$$\frac{\phi(1) \wedge \phi(2) \wedge \phi(3) \wedge \ldots}{\forall x.\ \phi(x)}$$

Clearly this is infeasible for practical theorem proving. Used backwards to prove $\forall x.\ \phi(x)$ it creates infinite branching. There is, however, a feasible version: the *constructive* ω-rule. It has the additional requirement that the $\phi(n)$ premises be proved in a *uniform* way, that is, that there exists an effective procedure, $proof_\phi$, which takes a natural number n as input and returns a proof of $\phi(n)$ as output. We will write this as $proof_\phi(n) \vdash \phi(n)$.

Cauchy's proof of Euler's formula, quoted in [44], effectively uses a polyhedral version of the *constructive* ω-rule. He describes, by example, the effective procedure $proof_{V-E+F=2}$ by applying it to a cube. He then invites us to recognise its generality by claiming that it could be successfully applied to *any* polyhedron. It turns out

[16] `theorymine.co.uk`

that this claim is false — as ably demonstrated by Lakatos' many counter-examples throughout the rest of the book. Cauchy's proof omits an important step: to verify that $proof_{V-E+F=2}$ will always prove Euler's theorem for any polyhedron.

Returning to the natural number version of the constructive ω-rule, we need to prove that:

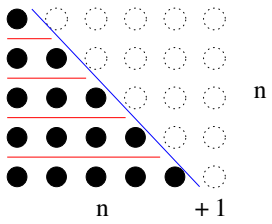
$$\forall n.\ proof_{\phi}(n) \vdash \phi(n) \tag{11}$$

We could, for instance, prove this by induction on n:

$$\begin{array}{l} proof_{\phi}(0) \vdash \phi(0) \\ proof_{\phi}(n) \vdash \phi(n) \implies proof_{\phi}(n+1) \vdash \phi(n+1) \end{array} \tag{12}$$

We seem to have gone full circle: replacing induction by the constructive ω-rule only to reintroduce it in the verification of (11). As Siani Baker showed in [1], however, the induction required to prove (11) is often simpler than the induction it replaces. For instance, whereas the original induction may require an intermediate lemma, the new verification one may not.

The constructive ω-rule was also used as the basis for Mateja Jamnik's work on diagrammatic reasoning [39]. In [59], Nelsen showed how arithmetic theorems can be 'proved' by displaying diagrams made of dots arranged into geometric shapes. See Fig. 3 for an example. Note that it exhibits only a concrete case of the theorem e.g., $\phi(5)$. The reader is expected to see that the implied special case procedure, $proof_{\phi}(5)$, can be generalised to the general one, $proof_{\phi}(n)$. Jamnik's Diamond system automated these proofs by extracting these special case proofs from the diagram, generalising them and then verifying (11) by induction.



The diagram can be viewed as a proof of the equation $1+2+\ldots+n = \frac{n.(n+1)}{2}$ *in the case that* $n = 5$. *The black circles represent the LHS of the equation. The whole rectangle represents the numerator of the RHS.*

Fig. 3 A Proof Without Words.

6.6.6 Category Theory and Analogy.

As outlined in §6.1, an *analogical blend* occurs when two old concepts are merged to form a new one. For instance, house and boat can be analogically blended in two ways: houseboat and boathouse.

A *colimit* is a construction from Category Theory that generalises constructions such as disjoint unions, direct sums, co-products, pushouts and direct limits[17]. A morphisms is defined between two parent logical theories. These match entities in one parent theory to those in the other. The colimit operation is a bit like unification in that it produces a minimal super-theory of the two parent theories that respects the morphism.

The colimit construction can be implemented as a procedure that takes two parent theories and outputs this minimal super-theory. This procedure can model analogical blending. Ewen Maclean and Alan Smaill were part of the team that applied this procedure to construct new mathematical concepts from old [5]. A simple example is the construction of the integers from the natural numbers and a theory of function inversion. The same team than applied it to construct new musical concepts [32]. An example is:

> "This simple blending mechanism 'invents' a chord progression that embodies some important characteristics of the Phrygian cadence (bass downward motion by semitone to the tonic) and the perfect cadence (resolution of tritone); the blending algorithm creates a new harmonic 'concept' that was actually introduced in jazz centuries later than the original input cadences." [5, p1]

Our approach to modelling analogy is typical of our Groups' combination of Mathematical theory and Cognitive Science.

6.6.7 Representing Uncertainty

The Web is a vast and rapidly growing source of information. To make best use of this information we want, not just to retrieve known facts from it, but to combine known information from diverse sources to infer new information. To realise this ambition requires us to address several challenges. For instance, information is stored in diverse formats: databases, RDF triples, description logics, natural language. These need to be curated into a common format so that inference can combine them. Due to its size, rapid growth and constant revision, however, it is infeasible to curate it globally. We must curate it locally as we gather the information we need for each inference task. The information is also noisy and inaccurate. Inferred knowledge needs to be assigned some indication of our uncertainty in asserting it.

Query answering on the Web is an ideal vehicle for exploring the interaction of multiple reasoning processes. Deduction is not sufficient. We also want to make predictions, for instance, estimating future populations from census data. This calls

[17] `https://en.wikipedia.org/wiki/Limit_(category_theory)` accessed 15.7.19.

for statistical reasoning techniques, such as regression or machine learning. Having formed functions by regression, we need to reason directly with them using higher-order methods, such as calculus, e.g., to estimate rates of change, points of intersection, maxima and minima, etc. We need reasoning calculi that assign uncertainty to assertions and inherit these through the reasoning methods, including methods, such as regression, that are themselves inherently uncertain. We need representational change to curate information into a common format. We need to diagnose and repair faults in the information we retrieve. Kwabena Nuamah implemented some of these techniques in the FRANK system (Functional Reasoner for Acquiring New Knowledge) [16, 60, 61].

An earlier attempt to merge deduction and probabilistic reasoning was my *Incidence Calculus* [7]. This was proposed, as an alternative to probabilistic logics, in order to solve the problem that numeric probabilities cannot be inherited through a derivation unless they are conditionally independent. Instead, sets of weighted possible worlds were associated with formulae rather than numeric probabilities. The degree of dependence between probabilities can then be represented by the amount of overlap between these sets.

7 Conclusion

For more than four decades, the DReaM Group has conducted a diverse range of innovative research projects, characterised by the motifs of: rigour combined with heuristics; meta-level reasoning, the productive use of failure; the interaction of diverse reasoning techniques; the learning of new reasoning methods; representational creation and evolution; and applications of rigorous mathematics to cognitive science problems, such as analogy, music, diagrammatic reasoning, etc.

These four decades have seen a radical change in system building. In the 1970s, AI researchers built systems from scratch, typically in Lisp, Prolog or a similar declarative language. Mecho [12], for instance, was built in this way. The result was often brittle systems that were slow to develop. Nowadays, large parts of systems are constructed using third-party packages from libraries, github, etc. Since these packages have often been thoroughly tested in prior applications, they tend to be robust. Development time is also faster, since you can build on the shoulders of giants rather than from first principles. Our FRANK system has been built like this [16].

There is a danger, however, as illustrated in Anecdote 8. If you rely on a package that is not well maintained, then bugs may not be fixed, and your system can become unusable. You may not discover this until you have invested a lot of energy and time into a failing system for which a replacement package is not available. So, although package use is definitely the way to go, it will repay you to carry out due diligence on each package before you commit to it. And, of course, there is not a package for every functionality you need, so you will still need to roll up your sleeves to write code to fill in the gaps.

It's instructive to contrast the kind of research programme described in J Moore's chapter in this volume, with the kind I have described above. His career has been devoted to constructing and applying a sequence of closely related theorem provers, culminating in the ACL2 prover. This can be applied to verify huge industrial-strength systems, such as microprocessors. As such, it is used routinely by several multinational hardware developers. In contrast, we have automated a diverse range of cognitive tasks building proof-of-concept prototypes that were not intended to be generic packages and were not widely used outside our group. We have been pioneers reconnoitring unexplored jungle. The six-lane highways come later. Or, you may describe our approach as 'AI as art': surprising people that it is even possible to automate aspects of cognition that seemed inherently to require human interaction.

J's chapter contrasts the 3-4 year UK PhDs with the 5-7 year US ones. The longer PhD provides the time for robust package building, and many US PhDs follow that pattern, with the developers offering maintenance support and community building to facilitate the continued use of their package. This is 'AI as Engineering'. Even so, few packages attain widespread uptake, especially when the packages' developers move on from their PhDs, perhaps to work for someone with a different agenda. Thus, our group's approach to research may have been inevitable.

AI has become almost synonymous with statistical machine learning (SML). In contrast, our group's research has been mainly symbolic: the epitome of 'Good, Old-Fashioned AI' (GOFAI). If we can't beat them, should we join them: mug up on statistics and obtain some massive data? I think not. SML has some fundamental limitations:

- It *requires* huge amounts of data. Humans, in contrast, can often learn from a single example combined with background knowledge.
- It is typically used to classify objects: cats *vs* dogs; good move *vs* bad; mortgage or not. Sometimes, however, it is necessary to learn compound structures, e.g., computer programs, that can't be viewed as classification tasks.
- SML systems work only with the features provided by the user. Sometimes, it is necessary to learn new concepts.
- It is inherently hard to explain the workings of a SML system. There is no logical derivation, only a huge and complex statistical calculation. But some applications *require* explanations — it may even be a legal requirement.

SML applications are beginning to confront these limitations. So, there is a future for us in developing hybrid approaches, where symbolic and statistical techniques complement each other. Launchbury describes a 'Third Wave of AI' that advocates this hybrid approach [45]. Members of our Group have already shown considerable enthusiasm for this third wave. Several of us attended a recent workshop at Imperial College organised by the Human-Like Computing Network+ (HLC), which has also embraced the third wave. The HLC Network+ has also created a community of AI researchers and Cognitive Scientists to learn from each other, which will be a crucial ingredient in the third wave, since humans can often do what is beyond SML and vice versa.

Of the work described here, that on learning new tactics from example proofs comes closest to realising that third-wave vision (see §6.6.3). The work of my PhD students Bernard Silver [68] and Simon Colton [21] were inherently logic-based learning techniques, but Hazel Duncan's [31] used the statistical technique of variable-length Markov models. The time might be ripe to re-investigate this approach.

We are also combining deductive and statistical reasoning in the FRANK system (see §6.6.7). So far, our use of statistics has been confined to regression, but we have plans to explore a variety of SML techniques. Currently, FRANK explains its reasoning by abstracting its deductive reasoning and associating English text augmented with graphical displays of regression. It will be interesting to explore the potential for explanations when more complex forms of SML are interleaved with deduction.

Acknowledgements The research reported in this paper was mainly supported by a succession of EPSRC rolling and platform grants. Thanks to Greg Michaelson, Kwabena Nuamah, Predrag Janičić and two anonymous referees for feedback on earlier versions.

References

1. Baker, S.: A new application for explanation-based generalisation within automated reasoning. In: A. Bundy (ed.) 12th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence, Vol. 814, pp. 177–191. Springer-Verlag, Nancy, France (1994)
2. Baker, S., Ireland, A., Smaill, A.: On the use of the constructive omega rule within automated deduction. In: A. Voronkov (ed.) International Conference on Logic Programming and Automated Reasoning — LPAR 92, St. Petersburg, Lecture Notes in Artificial Intelligence No. 624, pp. 214–225. Springer-Verlag (1992)
3. Basin, D.A., Walsh, T.: Difference unification. In: R. Bajcsy (ed.) Proc. 13th Intern. Joint Conference on Artificial Intelligence (IJCAI '93), vol. 1, pp. 116–122. Morgan Kaufmann, San Mateo, CA (1993). Also available as Technical Report MPI-I-92-247, Max-Planck-Institut für Informatik
4. Besold, T.R., Schorlemmer, M., Smaill, A. (eds.): Computational Creativity Research: Towards Creative Machines, *Atlantis Thinking Machines*, vol. 7. Atlantis Press (2015)
5. Bou, F., Schorlemmer, M., Corneli, J., Gomez-Ramirez, D., Maclean, E., Smaill, A., Pease, A.: The role of blending in mathematical invention. In: Proceedings of the sixth international conference of computational creativity (2015)
6. Boyer, R.S., Moore, J.S.: Proving theorems about LISP functions. In: N. Nilsson (ed.) Proceedings of the third IJCAI, pp. 486–493. International Joint Conference on Artificial Intelligence (1973). Also available from Edinburgh as DCL memo No. 60
7. Bundy, A.: Incidence calculus: A mechanism for probabilistic reasoning. Journal of Automated Reasoning **1**(3), 263–284 (1985)
8. Bundy, A.: A Science of Reasoning, pp. 178–198. MIT Press (1991)
9. Bundy, A.: Cooperating reasoning processes: more than just the sum of their parts. In: M. Veloso (ed.) Proceedings of IJCAI 2007, pp. 2–11. IJCAI Inc (2007). Acceptance speech for Research Excellence Award.
10. Bundy, A.: European collaboration on automated reasoning. AI Communications **27**(1), 25–35 (2013). DOI 10.3233/AIC-130584

11. Bundy, A., Basin, D., Hutter, D., Ireland, A.: Rippling: Meta-level Guidance for Mathematical Reasoning, *Cambridge Tracts in Theoretical Computer Science*, vol. 56. Cambridge University Press (2005)
12. Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R., Palmer, M.: Solving mechanics problems using meta-level inference. In: B.G. Buchanan (ed.) Proceedings of IJCAI-79, pp. 1017–1027. International Joint Conference on Artificial Intelligence (1979)
13. Bundy, A., Cavallo, F., Dixon, L., Johansson, M., McCasland, R.L.: The theory behind TheoryMine. In: Automatheo. FLoC (2010)
14. Bundy, A., van Harmelen, F., Horn, C., Smaill., A.: The Oyster-Clam system. 10th International Conference on Automated Deduction (1990)
15. Bundy, A., McCasland, R., Smith, P.: Mathsaid: Automated mathematical theory exploration. Applied Intelligence **47**(3), 585–606 (2017). DOI 10.1007/s10489-017-0954-8
16. Bundy, A., Nuamah, K., Lucas, C.: Automated reasoning in the age of the internet. In: 13th International Conference on Artificial Intelligence and Symbolic Computation, vol. LNAI 11110, pp. 3–18. Springer, Cham (2018). DOI 10.1007/978-3-319-99957-9_1. Invited Talk
17. Cantu, F., Bundy, A., Smaill, A., Basin, D.: Experiments in automating hardware verification using inductive proof planning. In: M. Srivas, A. Camilleri (eds.) Proceedings of the Formal Methods for Computer-Aided Design Conference, no. 1166 in Lecture Notes in Computer Science, pp. 94–108. Springer-Verlag (1996)
18. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Surveys in Computer Science. Springer-Verlag, Berlin (1990)
19. Clocksin, W.F., Mellish, C.S.: Programming in Prolog. Springer Verlag (1984)
20. Colton, S.: Automated conjecture making in number theory using HR, Otter and Maple. Journal of Symbolic Computation **39**, 593–615 (2005)
21. Colton, S., Bundy, A., Walsh, T.: HR: Automatic concept formation in pure mathematics. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, pp. 786–791 (1999)
22. Colton, S., Bundy, A., Walsh, T.: Automatic invention of integer sequences. In: Proceedings of the 17th National Conference on Artificial Intelligence, Austin, Texas, USA, pp. 558–563 (2000)
23. Colton, S., Bundy, A., Walsh, T.: On the notion of interestingness in automated mathematical discovery. International Journal of Human Computer Studies **53(3)**, 351–375 (2000)
24. Colton, S., Gow, J., Torres, P., Cairns, P.: Experiments in objet trouve browsing. In: Proceedings of the 1st International Conference on Computational Creativity (2010)
25. Colton, S., Pease, A.: The TM system for repairing non-theorems. In: W. Ahrendt, P. Baumgartner, H. de Nivelle, S. Ranise, C. Tinelli (eds.) Selected papers from the IJCAR'04 disproving workshop, *Electronic Notes in Theoretical Computer Science*, vol. 125 (3), pp. 87–101 (2004)
26. Cox, P.T., Pietrzykowski, T.: Causes for events: Their computation and applications. In: J. Siekmann (ed.) Lecture Notes in Computer Science: Proceedings of the 8th International Conference on Automated Deduction, pp. 608–621. Springer-Verlag (1986)
27. Cresswell, S., Smaill, A., Richardson, J.D.C.: Deductive synthesis of recursive plans in linear logic. In: Proceedings of the 5th European Conference on Planning, Durham, UK, *LNAI*, vol. 1809. Springer Verlag (1999)
28. Denney, E., Power, J., Tourlas, K.: Hiproofs: A hierarchical notion of proof tree. Electronic Notes in Theoretical Computer Science **155**, 341–359 (2006)
29. Dennis, L.A., Jamnik, M., Pollet, M.: On the comparison of proof planning systems: Lambda-Clam, Omega and IsaPlanner. In: Proceedings of 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus 2005), Electronic Notes in Computer Science (ENTCS) (2005). Available from http://www.cs.nott.ac.uk/ lad/work/publications.html
30. Dixon, L., Fleuriot, J.D.: IsaPlanner: A prototype proof planner in Isabelle. In: Proceedings of CADE'03, *LNCS*, vol. 2741, pp. 279–283 (2003)

31. Duncan, H., Bundy, A., Levine, J., Storkey, A., Pollet, M.: The use of data-mining for the automatic formation of tactics. In: Workshop on Computer-Supported Mathematical Theory Development. IJCAR-04 (2004)
32. Eppe, M., Confalonieri, R., Maclean, E., Kaliakatsos, M., Cambouropoulos, E., Codescu, M., Schorlemmer, M., Kühnberger, K.: Computational invention of cadences and chord progressions by conceptual chord-blending. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (2015)
33. Gärdenfors, P.: Belief Revision. No. 29 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (1992)
34. Giunchiglia, F., Walsh, T.: A theory of abstraction. Artificial Intelligence **56**(2–3), 323–390 (1992)
35. Gkaniatsou, A., McNeill, F., Bundy, A., Steel, G., Focardi, R., Bozzato, C.: Getting to know your card: Reverse-engineering the smart-card application protocol data unit. In: ACSAC 2015 Proceedings of the 31st Annual Computer Security Applications Conference, pp. 441–450. ACM (2015). DOI 10.1145/2818000.2818020
36. Ireland, A., Bundy, A.: Productive use of failure in inductive proof. Journal of Automated Reasoning **16**(1–2), 79–111 (1996)
37. Ireland, A., Ellis, B.J., Cook, A., Chapman, R., Barnes, J.: An integrated approach to high integrity software verification. Journal of Automated Reasoning: Special Issue on Empirically Successful Automated Reasoning **36**(4), 379–410 (2006)
38. Ireland, A., Stark, J.: Combining proof plans with partial order planning for imperative program synthesis. Journal of Automated Software Engineering **13**(1), 65–105 (2005)
39. Jamnik, M., Bundy, A., Green, I.: On automating diagrammatic proofs of arithmetic arguments. Journal of Logic, Language and Information **8**(3), 297–321 (1999)
40. Johansson, M., Dixon, L., Bundy, A.: Conjecture synthesis for inductive theories. Journal of Automated Reasoning **47**, 251–289 (2011)
41. Kowalski, R.: Logic for Problem Solving. Artificial Intelligence Series. North Holland (1979)
42. Kowalski, R.A., Kuehner, D.: Linear resolution with selection function. Artificial Intelligence **2**, 227–60 (1971)
43. Kraan, I., Basin, D., Bundy, A.: Middle-out reasoning for logic program synthesis. In: Proc. 10th Intern. Conference on Logic Programing (ICLP '93) (Budapest, Hungary), pp. 441–455. MIT Press, Cambridge, MA (1993)
44. Lakatos, I.: Proofs and Refutations: The Logic of Mathematical Discovery. Cambridge University Press (1976)
45. Launchbury, J.: A DARPA perspective on artificial intelligence (2018). Talk with slides on YouTube
46. Lehmann, J., Chan, M., Bundy, A.: A higher-order approach to ontology evolution in Physics. Journal on Data Semantics pp. 1–25 (2013). DOI 10.1007/s13740-012-0016-7
47. Li, X., Bundy, A., Smaill, A.: ABC repair system for Datalog-like theories. In: 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, vol. 2, pp. 335–342. SCITEPRESS, Seville, Spain (2018). DOI 10.5220/0006959703350342
48. Lighthill, S.J.: Artificial Intelligence: A General Survey, pp. 1–21. Science Research Council (1973)
49. Lin, Y., Bundy, A., Grov, G., Maclean, E.: Automating Event-B invariant proofs by rippling and proof patching. Formal Aspects of Computing pp. 1–35 (2019). DOI 10.1007/s00165-018-00476-7
50. Llano, M.T., Ireland, A., Pease, A.: Discovery of invariants through automated theory formation. Formal Aspects Computing **26**(2), 203–249 (2014)
51. Maclean, E., Ireland, A., Grov, G.: Proof automation for functional correctness in separation logic. Journal of Logic and Computation (2014). DOI: 10.1093/logcom/exu032
52. Martinez, M., Abdel-Fattah, A., Krumnack, U., Gomez-Ramirez, D., Smaill, A., Besold, T.R., Pease, A., Schmidt, M., Guhe, M., Kühnberger, K.U.: Theory blending: Extended algorithmic aspects and examples. Annals of Mathematics and Artificial Intelligence **80**(1), 65–89 (2017). DOI 10.1007/s10472-016-9505-y. URL `http://homepages.inf.ed.ac.uk/smaill/martinezEtAl16.pdf`

53. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: B. Meltzer, D. Michie (eds.) Machine Intelligence 4. Edinburgh University Press (1969)
54. McCasland, R.L., Bundy, A., Smith, P.F.: Ascertaining mathematical theorems. In: J. Carette, W.M. Farmer (eds.) Proceedings of Calculemus 2005. Newcastle, UK (2005)
55. McCune, W.: The Otter user's guide. Tech. Rep. ANL/90/9, Argonne National Laboratory (1990)
56. McCune, W.: A Davis-Putnam program and its application to finite first-order model search. Tech. Rep. ANL/MCS-TM-194, Argonne National Laboratoriess (1994)
57. McNeill, F., Bundy, A.: Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. International Journal On Semantic Web and Information Systems **3**(3), 1–35 (2007). Special issue on ontology matching.
58. Montano-Rivas, O., McCasland, R., Dixon, L., Bundy, A.: Scheme-based theorem discovery and concept invention. Expert Systems with Applications **39**(2), 1637–1646 (2012)
59. Nelsen, R.B.: Proofs without Words: Exercises in Visual Thinking. The Mathematical Association of America (1993)
60. Nuamah, K.: Functional inferences over heterogeneous data. Ph.D. thesis, University of Edinburgh (2018)
61. Nuamah, K., Bundy, A.: Calculating error bars on inferences from web data. In: SAI Intelligent Systems Conference (IntelliSys), pp. 618–640. Springer, Cham (2018). DOI 10.1007/978-3-030-01057-7_48
62. O'Keefe, R.: Logic and lattices for a statistics advisor. Ph.D. thesis, University of Edinburgh (1987)
63. O'Keefe, R.A.: The Craft of Prolog. MIT Press, Cambridge, Mass (1990)
64. Paulson, L.: Isabelle: the next 700 theorem provers. In: P. Odifreddi (ed.) Logic and Computer Science, pp. 77–90. Academic Press (1990)
65. Peano, G.: Arithmetices Principia Novo Methodo Exposita. Bocca (1889). URL `http://eudml.org/doc/203509`
66. Pease, A., Colton, S., Ramezani, R., Smaill, A., Guhe, M.: Using analogical representations for mathematical concept formation. In: L. Magnani, W. Carnielli, C. Pizzi (eds.) Model-based Reasoning in Science and Technology: Abduction, Logic, And Computational Discovery, no. 341 in Studies in Computational Intelligence, pp. 301–314. Springer (2010). URL `http://springerlink.com/content/y1t348758g46q462/fulltext.pdf`
67. Robertson, D., Bundy, A., Muetzelfeldt, R., Haggith, M., Uschold, M.: Eco-Logic: Logic-Based Approaches to Ecological Modelling. MIT Press (1991)
68. Silver, B.: Meta-level inference: Representing and Learning Control Information in Artificial Intelligence. North Holland (1985). Revised version of the author's PhD thesis, Department of Artificial Intelligence, U. of Edinburgh, 1984
69. Steel, G., Bundy, A.: Attacking group protocols by refuting incorrect inductive conjectures. Journal of Automated Reasoning **First Online Edition**, 1–28 (2005). Special Issue on Automated Reasoning for Security Protocol Analysis
70. Sterling, L., Bundy, A., Byrd, L., O'Keefe, R., Silver, B.: Solving symbolic equations with PRESS. J. Symbolic Computation **7**, 71–84 (1982). Also available from Edinburgh as DAI Research Paper 171.
71. Sterling, L., Shapiro, E.: The Art of Prolog. MIT Press, Cambridge, MA (1986)
72. Winterstein, D., Bundy, A., Gurr, C.: Dr. Doodle: A diagrammatic theorem prover. In: D. Basin, M. Rusinowitch (eds.) Automated Reasoning, Lecture Notes in Computer Science, pp. 331–335. Springer Berlin Heidelberg (2004). DOI 10.1007/978-3-540-25984-8_24