Constructing, Selecting and Repairing Representations of Knowledge

Constructing, Selecting and Repairing Representations of Knowledge [*]

Alan Bundy
A.Bundy@ed.ac.uk
School of Informatics, University of Edinburgh,
EH8 9LE, Scotland

Abstract

Using the achievements of my research group over the last 30+ years, I provide evidence to support two hypotheses:

1. *The representation of a problem is often the key to its solution;*
2. *Problem representations can be automatically both formed and repaired.*

I focus mainly on mathematical problem solving based on logical representations, drawing on work in solving mechanics problems in English, assisting ecologists to construct simulation models, reasoning with diagrams, providing an analogy-based functional program editor and repairing faulty mathematical theories and ontologies. I look especially at the interaction between reasoning and representation, illustrating how failures in reasoning can suggest representational improvements and repairs. Methodologically, interactive systems can provide a vehicle for focusing on some aspects of representation construction, while relying on user interaction to complement automation.

Introduction

This paper recounts a 30+ year personal exploration of the issues surrounding the automated construction, selection and repair of representations of knowledge. I see the automation of these representational processes as a central problem in automated problem solving and as a much neglected one. In particular, there has been insufficient exploration of the interaction of reasoning and representation, especially into how failures of reasoning can suggest the repair of existing representations or the formation of new ones. We briefly mention some of the subfields of AI that touch on these issues.

Automated Theorem Proving is nearly always based on a fixed representation: a formal theory drawn from mathematical practice or hand-crafted by the developer. There has been little exploration of how mathematicians formalise informal problem statements or how formal theories are constructed.

Belief Revision has developed mechanisms for the selective withdrawal of beliefs in order, for instance, to effect a minimal repair on an inconsistent knowledge base. **Truth Maintenance Systems** provide mechanisms for storing alternate justifications for inferred knowledge in order to propagate the effects of withdrawn or reasserted knowledge in a knowledge base. None of these mechanisms affect what we will call the *signature* of the knowledge base, i.e., the functions and predicates with which the knowledge is formulated, together with their arities, types, etc. Nor do they offer mechanisms for the initial *formation* of representations. There has been little investigation of inference failures *not* based on inconsistency resolution, e.g., failures of proof search.

Machine Learning has developed many mechanisms for forming concepts and rules. Symbolic learning techniques usually use an existing representation to define new concepts in terms of old or hypothesise new rules. As with belief revision and truth maintenance, these mechanisms do not affect the underlying signature of the representation. Non-symbolic mechanisms, such as neural nets, build concept recognisers, but there has been little work driving such concept formation by reasoning requirements or failures.

Ontology Matching has developed mechanisms for matching the signature of one representation to that of another. However, most of these mechanisms are restricted to aligning isa hierarchies (or classifications) based on unary predicates or concepts.

Natural Language Processing has developed mechanisms for translating utterances in natural languages, such as English, into formal representations. In the process, ambiguities are resolved, requiring the selection of the most likely representation of the intended meaning. However,

[*] I would like to thank the many colleagues with whom I have worked over the last 30+ years on the research reported in this paper: MECHO (George Luger, Chris Mellish, Rob Milne, Martha Palmer (née Stone) and Bob Welham), ECO (Robert Muetzelfeldt, Mandy Haggith, Dave Robertson and Mike Uschold), DIAMOND (Mateja Jamnik and Ian Green), CYNTHIA (Jon Whittle, Helen Lowe and Richard Boulton), TM/HR (Alison Pease, Simon Colton, Alan Smaill, John Lee and Toby Walsh) and ORS (Fiona McNeill, Marco Schorlemmer and Chris Walton). I would also like to thank Simon Colton, Andrew Ireland, Roy McCasland, Raúl Monroy and Mike Uschold for feedback on an earlier draft. The research reported in this paper was supported by numerous grants and studentships, the most recent of which is EPSRC GR/S01771.

all the alternate representations are usually based on the same, fixed signature and logic.

As you can see from this wide-ranging but superficial survey, AI has played insufficient attention to where representations and especially their underlying signatures and logics come from, or how they are modified when they are found to be wanting. This may have been sufficient for constructing AI systems to work in circumscribed domains. It will not be sufficient for building AI systems in open-ended domains, where it is necessary to react to new and unexpected situations.

In this paper we will explore two hypotheses:

1. *The representation of a problem is often the key to its solution;*
2. *Problem representations can be automatically both formed and repaired.*

I will provide evidence to support these hypotheses, mostly drawn from the work of my research group since 1975. However, I start, not with my group's work, but with the observations on the art of mathematical problem solving provided by one of the worlds foremost mathematicians and most respected mathematical teachers, George Pólya.

How to Solve It

Pólya's beautiful little book "How to Solve It" (Pólya 1945) ought to be a gold mine for automated reasoners. Written by a world famous mathematician, it provides practical advice on mathematical problem solving. It has been widely and successfully used by students of mathematics and related subjects. It has often been observed that this practical advice ought to provide insight for those building automated reasoning and problem-solving computer programs. However, despite the frequency of these observations, there is remarkably little AI or automated reasoning research that one can point to as having been directly influenced by Pólya[1]

Pólya's advice is divided into four parts: understanding the problem, devising a plan, carrying out the plan and looking back. Each part consists mainly of a series of Socratic questions that a problem solver should ask him/herself, together with examples of this technique applied to many particular problems. Typical questions from each part are: "What is the unknown?", "Do you know a related problem?", "Can you see clearly that each step is correct?" , "Can you derive the result differently?". From these examples we can see some of the difficulty in applying Pólya's advice. Consider, for instance, each of these questions in the context of a standard, logic-based, proof system.

- *What is the unknown?* The problem formulation for a logic-based prover usually comes with a well-labelled goal or conjecture, so it is not clear what addressing this question would add.
- *Do you know a related problem?* This question suggests using the proof of an analogous theorem as a guide to construct the proof of the current theorem. There has been periodic work using this technique. (Owen 1990) provides a good survey up to 1990. However, such work has seldom been sustained and has not been very influential.
- *Can you see clearly that each step is correct?* The steps of logic-based problem solvers are typically correct by construction, so this advice has little purchase.
- *Can you derive the result differently?* Problem solvers usually work in a search space of possible solutions. It is technically simple to arrange for this space to be searched for more than one solution, although it is unclear what this gains unless you need a solution with some special property, e.g., optimality, efficient execution, reliability.

These examples questions were selected pretty much at random. Similar remarks apply to most of the others.

Why is Pólya's advice so helpful to human students and so unhelpful for automated problem solvers? The resolution of this paradox, I suggest, is that his advice is mostly aimed at a *different* task than that faced by automated problem-solver designers. Automated problem solvers are typically designed to work with a fixed representation of the world: a description of the blocks world or the axioms of group theory. This representation defines a search space. The main issue is how to search this space efficiently, e.g., by guiding search, by removing redundancy from the space, by searching several branches in parallel, by clever indexing to improve look-up times, by space saving techniques, etc. Pólya has very little to say about any of this.

Much of Pólya's advice is about *representing* the problem in such a way that it can be readily solved, e.g., because some standard technique applies to it, or it reduces to some previously solved problem. A formal representation is required because he assumes that the problem is initially presented informally. For instance, most of the example problems in (Pólya 1945) are stated in English with or without an accompanying diagram.

If, in the light of these observations, we revisit Pólya's previous four questions, we can now interpret them in a very different way.

- *What is the unknown?* In understanding an informally stated problem it is vital to identify what is given and what is sought. Pólya continues by asking us to draw a figure, introduce notation and write down any conditions — all vital steps in *representing* the problem, rather than *solving* it.
- *Do you know a related problem?* If you listen carefully to uses of analogy in everyday conversation, you will see that it is rarely using an old argument to guide the search for a new one. Rather it is suggesting a way of *representing* the problem in a way analogous to an existing representation. The solution is thereby immediately suggested, i.e., no search is required.
- *Can you see clearly that each step is correct?* The solution steps may not have been fully formalised, so may not be correct by construction. Moreover, we have some

[1] One notable exception is Funt's MSc project on ruler and compass constructions in geometry (Funt 1973), but this was based on the very practical hints on this topic in chapter 1 of another of Pólya's books (Pólya 1965).

additional reasoning to check: that the problem representation we have chosen is faithful to the original problem statement.

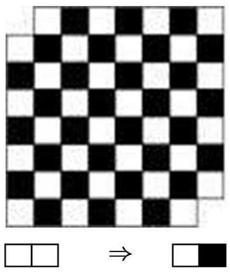
- *Can you derive the result differently?* If a different representation of the problem yields the same result, then this may give us faith in the robustness of the solution to representational variations.

Again, similar reasoning applies to most of Pólya's Socratic questions. So Pólya reminds us that *forming* a representation is a key part of problem solving.

For a good example of the way in which choosing the right representation can make the solution almost trivial, consider the mutilated checkerboard problem made popular by John McCarthy (McCarthy 1964) and illustrated in Figure 1 (a).

> *An* 8×8 *checkerboard has two opposite corner squares removed. Can you cover the resulting mutilated checkerboard completely with* 1×2 *dominoes?*

The key to the solution is to colour the two squares of each domino black and white (see Figure 1). It is then clear that each domino will cover exactly one black and one white square of the grid. So, for the dominoes to completely cover the grid, there must be exactly the same number of black and white squares in the grid, but the numbers are unequal, since both the removed corners squares had the same colour.



The mutilated checkerboard consists of an 8×8 *with alternately coloured squares, from which the top left and bottom right squares have been removed. Note that there are two more white squares than black. Initially, the* 1×2 *domino pieces are uncoloured. By colouring one square white and one black, we can see that it is impossible to cover the mutilated checkerboard completely with dominoes.*

Figure 1: The Mutilated Checkerboard Problem

Though it offers a nice example of the power of representation, examples such as the mutilated checkerboard problem are not a good vehicle for exploring the automation of representation formation. They are one-offs. It would be possible to develop a computer program that could solve this problem by colouring the grid, recognising that each domino placement maintained an invariant, then realising that this invariant was violated in the case of the mutilated grid. However, it would be too tempting to build this heuristic into the heart of the program, so that it tried to solve *every* problem in this way. What is interesting about the problem is *discovering* the heuristic, not *applying* it. We are too far away from a theory of heuristic discovery for this to be a timely project. What we need to start this exploration of representation formation is a domain in which it is routine rather than exceptional: some foothills in which we can hone our representation formation techniques, before we attempt Everest. In the late 1970s, my group found such a domain.

The Idealization of Real-World Objects in Mechanics Problem Solving

The six year MECHO Project (1975-81) (Bundy *et al.* 1979), investigated mechanisms for constructing mathematical models of a real-world situation. In particular, the MECHO program solved mechanics problems stated in English, with examples drawn from the English GCE A-Level, applied-mathematics papers, intended for 16-18 year old pre-university entrants. MECHO took mechanics problems stated in English, parsed them, constructed a semantic representation in the form of first-order logic formulae, extracted equations from this logical representation and then solved them.

For instance, consider the pulley problem illustrated in figure 2. The English sentences from a mechanics text book were first represented as first-order formulae, such as $isa(particle, p_b)$, $mass(p_b, mass_b, period_1)$, $measure(mass_b, b)$, $accel(p_b, a_b, 270, period_1)$, $given(mass_b)$ and $sought(a_b)$[2]. Standard physical laws, such as $F = M.A$, were then instantiated with expressions from this first-order representation to form equations, e.g., $mass_b.g + tension_b = mass_b.a_b$. The choice of physical laws and their instantiation into equations was controlled by an algorithm we christened *The Marples Algorithm*. These equations were then solved, e.g., for a_b in this example, to give the required solution to the problem. MECHO was applied to a wide range of mechanics problems: pulleys, motion under constant acceleration, levers, motion on a smooth path, statics, Hooke's Law, moment of inertia, etc.

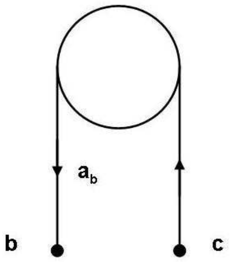
Constructing mathematical representations of real-world situations always involves *idealization*, which Wikipedia[3] defines as follows:

> *"Idealization is the process by which scientific models assume facts about the phenomenon being modeled that are certainly false. Often these assumptions are used to make models easier to understand or solve. Many times idealizations do not harm the predictive accuracy of the model for one reason or another. Most debates surrounding the usefulness of a particular model often are about the appropriateness of different idealizations."*

We can see examples of idealization in the pulley problem in figure 2. For instance, the weights at the end of the pulleys are called "particles", i.e., objects with mass but no extent;

[2] Note how, following Pólya's advice, MECHO has identified the given and the unknown of the problem.

[3] `http://en.wikipedia.org/wiki/Idealization`



"Two particles of mass b and c are connected by a light string passing over a smooth pulley. Find the acceleration of the particle of mass b." [Taken from (Humphrey 1957)].

Figure 2: A Simple Mechanics Problem

the string is described as "light", i.e., its mass can be ignored; the pulley is described as "smooth", i.e., any friction in its axle can be ignored.

A key part of solving mechanics problems is choosing the right idealization. For instance, a ship in a relative velocity problem is usually idealized as a particle on a horizontal plane, whereas in a specific gravity problem it will be idealized as a 3D shell floating on a liquid. In forming its initial first-order representation of an English problem statement, MECHO had to decide how to idealize each of the real-world objects mentioned in the statement. For MECHO, this problem was made relatively simple because most textbook mechanics problems come in fairly readily recognisable types and the problem type largely determines the appropriate idealizations. Cues in the problem statement, e.g., phrases, such as "pulley system", "relative velocity", "Archimedes Principle", etc., help identify the problem type. Further cues, "light string", "smooth pulley", "inelastic plane", etc. , help resolve any remaining ambiguity.

To illustrate how much students come to depend on these cues, consider the following experiment carried out on freshman physics undergraduates by Andrea diSessa (diSessa 1983). They were posed the following problem.

> *"If a ball is dropped, it picks up speed and hence kinetic energy. When the ball hits the floor, however, it stops (before bouncing upward again). At that instant, there is no kinetic energy, since there is no motion. Where did the energy go?" [Taken from (diSessa 1983)]*

The answer given by diSessa is:

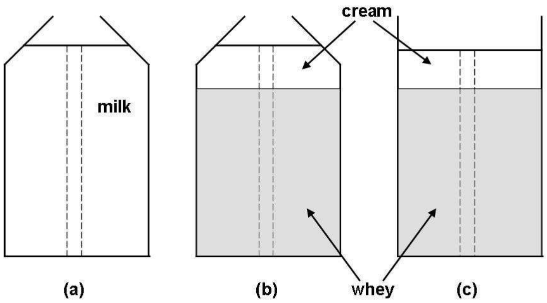
> *"The ball and floor are mutually compressed on impact. That compression, just like the compression of a spring, stores energy in mechanical distortion. That is where the energy goes when the ball comes to a stop."*

The students had a great deal of difficulty discovering this explanation — or even accepting it when it was explained to them. Their problem was that conservation-of-energy problems, such as this, invariably cue an idealization of the ball as a particle with no extent. With no extent, no compression is possible. The resolution of the paradox can only come when this fossilized idealization is rejected and a new one formed, in which the ball *has* extent.

In real physics and engineering tasks, as opposed to textbook, applied-maths problems, idealization is not so fossilized, but is a key part of the task. For instance, in (Cohen 1974), Harvey Cohen describes *Milko* a, so called, "dragon problem" that crucially depends on appropriate idealization.

> *"A milk bottle is allowed to stand so that the cream rises to the top: this occurs without any change in total volume. Does the pressure near the base of the bottle change." [Taken from (Cohen 1974)]*

Many people, posed this question, initially reply that the pressure is unchanged, which Cohen describes as "the canonical wrong answer". However, this answer would be *right* if the milk bottle were idealized as having a uniform cross-section, as rectangular milk cartons do, for instance. Getting what Cohen describes as the right answer depends on idealizing the milk bottle as having a conical (or similarly shaped) top part (See Figure 3 for a discussion). More such dragon problems can be found in (Walker 1975). Unfortunately, there is little uniformity in the idealization process in dragon problems. Each one is a little *tour-de-force*. Just as in McCarthy's mutilated checkerboard problem, this makes dragon problems an unsuitable domain for an investigation of idealization.



Diagrams (a) and (b) show the milk bottle before and after the cream has settled out. Consider a column of milk which does not touch the sides (so pressure from the bottle walls need not be taken into account). The column in (b) contains more cream than the corresponding column in (a), but they both have the same volume. Since cream is lighter than whey, the base pressure in (b) is less than in (a). In diagram (c), the columns have equal proportions whether the cream is separated out or not, so the pressure is unaffected by cream separation.

Figure 3: Cohen's Milko Problem

MECHO's idealization task was made easier by its fossilization within problem classes. However, this made textbook mechanics problems an unpromising vehicle for exploring the idealization process. Since idealization is at the heart of the construction of mathematical models of real-world situation, this caused us to abandon this domain in favour of one in which idealization could be properly addressed.

User-Assisted Idealization in Ecological Modelling

The six year ECO Project (1983-89) (Robertson *et al.* 1991), developed an intelligent front-end for ecological modelling. The ECO program built an ecological simulation program in response to a dialogue with a user. From the dialogue it first constructed a representation of an ecological situation using a first-order sorted logic, then it extracted a Fortran or Prolog program using similar methods to the MECHO program. Ecological simulation models are used to make predictions about different kinds of intervention in the natural or farming world, e.g., to avoid undesirable outcomes, optimise interventions, etc. More people could benefit from such simulation models than have the skill to build one themselves or the resources to have others build one for them. ECO addresses this issue by allowing unskilled users to build their own simulation model at low cost. Its user dialogue is conducted in terms of the ecological situation, rather than in programming or mathematical terms.

Idealization arises in simulation models primarily in the way that objects are divided up into sub-objects, or not. For instance, a field may be divided into different zones, a flock of sheep into age classes or a tree into roots, trunk, branches, leaves, etc. This domain provides an ideal next step up from mechanics textbook problems as a vehicle for investigating idealization. On the one hand, the types of idealization are fairly standardized, enabling a pre-defined set of choices. On the other hand, there is no fossilization of the choices; they will depend on the user's needs. For instance, a field might be divided into: point locations, with or without proximity relations; locations with area; a grid square; zones; or left undivided. The user will choose one of these idealizations depending on the physical characteristics of the field or the detail required in the model. The interactive environment also enabled ECO to provide automated assistance where we had developed theories about the idealization process, but to leave the decision to the user where we had no such theories or where the user wanted to override our theories.

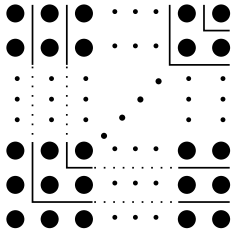
One idealization theory developed in the ECO project was that objects should be sub-divided when they represented an independent variable on which some dependent variable depended, and where the user wanted to predict that dependent variable. For instance, suppose the user wanted to track the biomass of a pack of wolves. Suppose, further, that wolf biomass was known, from a previous user interaction, to depend on both age and location. Then ECO would suggest sub-dividing the wolves into age and location classes. It had some standard classification methods for both age and location, which it would suggest to the user via a menu.

ECO also had standard idealizations for processes, e.g., logistic growth, predator-prey equation, competition equation, interacting objects, etc., which were presented to the user to choose from. Gaps in the knowledge required to automate these processes could also serve as a trigger for further idealization. For instance, the idealization needed for the interacting objects process requires a measure of proximity between the objects that are interacting. In the absence of such a proximity measure, an appropriate location class will be triggered.

The interactive context, which enabled us to make partial progress in our study of idealization, also ultimately inhibited it. Many of the idealization choices must depend on the user's goal and knowledge of the world, so could not be automated except via user interaction. To make further progress in the automation of idealization, it was necessary to look for a domain in which the automated agent had both internal goals and some knowledge about how different representations would promote or inhibit the realisation of those goals. As Pólya has shown, pure mathematics can provide such a domain. The goal is to prove a theorem and representations can be objectively tested against this goal.

Diagrammatic Reasoning

For her PhD (1995-1999), Mateja Jamnik developed the DIAMOND program: an interactive system which constructs "proofs without words" (Jamnik, Bundy, & Green 1999; Jamnik 2001). The term "proofs without words" is drawn from Nelsen's book (Nelsen 1993), in which theorems of natural numbers are proved solely using a diagram (see Figure 4 for an example).



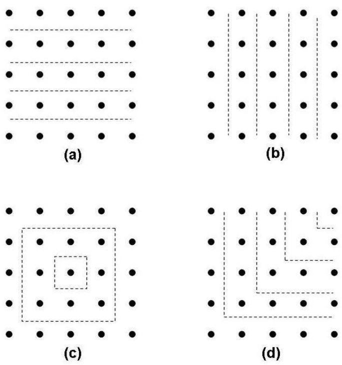
The diagram gives a proof of the theorem $n^2 = 1 + 3 + \cdots + (2n-1)$. The diagram can be viewed as describing both the left and right hand sides of the equation. The whole square represents n^2. Each of the L-shapes represents one of the odd numbers summed on the right-hand side.

Figure 4: A Proof without Words

DIAMOND conducts idealization in its choice of diagrammatic representations of algebraic formulae. For instance, a square can be represented in at least the following ways:

1. A sequence of rows, each of which is a list of dots.
2. A sequence of columns, each of which is list of dots.
3. A concentric sequence of circumferences, each of which is a square ring.
4. A nested sequence of ell shapes.

Figure 5 illustrates these options.



The same square is decomposed into (a) rows, (b) columns, (c) rings and (d) ells.

Figure 5: Alternative Representations of a Square

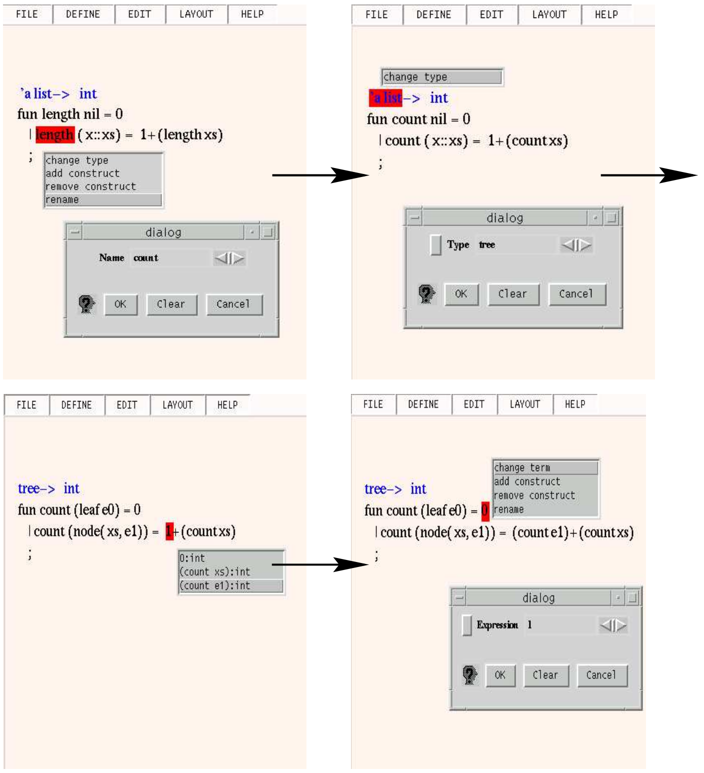
A successful proof depends crucially on finding appropriate idealizations. Most proofs without words work by mapping each side of the equation to be proved to the same shape, but idealized in two different ways. For instance, in the proof illustrated in Figure 4, both sides of the equation are represented by a square. However, whereas the left-hand-side of the equation, n^2, is idealized using either (1) or (2), the right-hand-side, $1 + 2 + \ldots + (2n - 1)$ is idealized using (4).

Currently, DIAMOND requires user interaction to provide these idealizations implicitly by deconstructing a couple of example diagrams, e.g., for $n = 3$ and $n = 4$. It remains an unrealised aim to automate this idealization process. Fortunately, there is more hope of complete idealization in this domain than there was in the ecological modelling one. Appropriate idealization does not depends on some implicit user requirements for a simulation program, but only on success in rapid proof of the theorem, for which we have an objective test. Moreover, we can map certain algebraic operations onto specific idealizations: multiplication onto idealizations (1) or (2) of a rectangle; odd numbers onto idealization (4); $8n$ to idealization (3); etc. These idealizations correspond to recursive decompositions of the general n-ary diagram. They are equivalent to constructing an induction rule when carrying out an inductive proof. A lot is known about how induction rules can be constructed (Bundy 2001). There is some hope of adapting that knowledge to this situation.

All the mechanisms described above build representations of a problem from a fixed signature of predicates, functions, diagrams, etc. But I have argued that we also need to consider how to form and evolve the signatures themselves. We first addressed this problem in the context of constructing new computer programs.

An Analogy Driven Program Editor

For his PhD (1995-1999), Jon Whittle developed the CYNTHIA program: an analogy-based editor for ML functional programs, (Whittle *et al.* 1999; Whittle 1999). CYNTHIA enables ML programmers to efficiently convert an old ML function into a new one, while being protected from various syntactic, semantic and termination errors. Powerful commands are associated with each part of the old function and provided in drop-down menus. For instance, editing the type signature of a function also updates all the corresponding patterns in the heads of its definition. An extra argument added to a function is inserted in all occurrences of that function. Changing the form of a function's recursive call is effected with a single command, but updates all the step and base cases of that recursive function. Figure 6 gives an example session with CYNTHIA.



```
'a list-> int
fun length nil = 0
  | length (x::xs) = 1+(length xs)
  ;
```

Name count

```
'a list-> int
fun count nil = 0
  | count (x::xs) = 1+(count xs)
  ;
```

Type tree

```
tree-> int
fun count (leaf e0) = 0
  | count (node( xs, e1)) = 1+(count xs)
  ;
```

0:int
(count xs):int
(count e1):int

```
tree-> int
fun count (leaf e0) = 0
  | count (node( xs, e1)) = (count e1)+(count xs)
  ;
```

Expression 1

A function to calculate the length of a list is edited into a function to count the nodes of a tree. Firstly, the name of the function is changed from $length$ to $count$. Note how all three occurrences are changed. Next, the type declaration is changed from $'alist \rightarrow int$ to $tree \rightarrow int$. Note how the patterns in the heads of the definitions are automatically updated to the new datatype, and that a type error is highlighted in red in the body. Finally, this type error is corrected and the edit is complete.

Figure 6: A Session with the CYNTHIA Analogy-Based Program Editor

Behind the scenes, the function is stored as a proof that it meets a very weak specification, namely that it has the correct type. The function is extracted from this verification proof. Editing commands actually change this underlying proof, rather than affect the function itself. An editing command may render the proof invalid. CYNTHIA tries to automatically repair any invalid proof. If it succeeds, then it will ensure that: the resulting function is syntactically correct; is neither over nor under-defined; and that it terminates. If it fails, it will flag up parts of the function that need to be edited to ensure these properties are true, e.g., that it type checks properly.

The editing commands can be regarded as effecting a kind of representational repair or adaptation. Representational repairs range from changing the name of a function, via adding or subtracting a function's argument, or changing the type of a function, through to changing its recursive structure. CYNTHIA's role is: to present legal representation change options to a user; implement those changes that the user selects; and ensure that various desirable properties are maintained or any violations are brought to the user's attention to be fixed.

Belief revision and truth maintenance mechanisms change a representation by adding and deleting formulae from a knowledge base. The underlying signature from which these formulae are constructed remains unchanged. CYNTHIA shows that one can go beyond this to change the underlying signature, i.e., effect changes of name, arity, semantics, type, etc., while maintaining well-formedness properties such as syntax, well-definedness, termination, etc.

However, CYNTHIA required user assistance to determine *which* signature changes to make. There is an open question about whether such signature changes can be driven automatically in pursuit of some higher-goal. One possibility is to react to unexpected failures (or successes) of reasoning to repair a faulty ontology to make it a better representation of the world. We explore this possibility in the next two sections.

Repairing Faulty Theories

For her PhD (2000-06), Alison Pease developed the TM program: an implementation of ideas of Lakatos (Lakatos 1976) for repairing faulty mathematical theories (Pease *et al.* 2004). TM uses Colton's HR system (Colton, Bundy, & Walsh 1999; Colton 2002), which formulates concepts and conjectures by generalising from examples. TM uses HR to discover conjectures that are nearly true, i.e., have only a few counter-examples. HR, the Otter theorem prover and the Mace counter-example finder then interact to first analyse the counter-examples and failed proofs, and then to repair the mathematical theory.

Pease's work shows that Lakato's methods of proof and refutation suggest computationally effective procedures for automated reasoning, in particular for repairing faulty mathematical theories. Lakatos methods, such as piecemeal exclusion, strategic withdrawal and counter-example-barring, are used automatically to reformulate the original conjecture, redefine some of the concepts or repair a failed proof. This contrasts with our current failure to exploit the advice of Pólya. But, if as I suggest, we revisit Pólya's advice as applying to the formation and repair of representations, then we will find that it too suggests effective computational mechanisms.

TM redefines concepts and reformulates conjectures in terms of a fixed underlying primitives, whereas CYNTHIA changes the underlying signature. In the next section we see how signature changes can also be automated.

Dynamic Ontology Repair

For her PhD (2001-06), Fiona McNeill developed the ORS program: an automated system for repairing faulty ontologies in response to unexpected failures in plan execution (Bundy, McNeill, & Walton 2006; McNeill 2005). ORS forms plans to achieve its goals using the services provided by other agents. In forming these plans, ORS draws upon its knowledge base, which represents its world, including its beliefs about the abilities of other agents and under what circumstances they will perform various services. To request actions or ask questions of the other agents, ORS uses a simple performative language implemented in KIF[4], an ontology language based on first-order logic.

The representation of the world used by ORS may be faulty, not just in containing false beliefs, but also in using a signature that does not match that used by some of its collaborating agents. This mismatch will inhibit inter-agent communication, leading to faulty plans that will fail during execution. ORS analyses its failed plans, communicates with any agents that unexpectedly refused to perform a service, and proposes repairs to its ontology, including the signature of that ontology. Repairs can include: adding, removing or permuting arguments to predicates or functions, merging or splitting of predicates or functions and changing their types, as well as some belief revisions, such as adding or removing the precondition of an action. The signature repairs are based on a survey of types of abstraction in AI problem solving (Giunchiglia & Walsh 1992). It uses not only all these forms of abstractions, which remove detail from a signature, but also their inverses, refinements that add detail to a signature. Sometimes a refinement is indicated by the analysis, but cannot be executed due to insufficient knowledge about the detail to be added.

The ORS project has demonstrated that, in many cases, it is possible to automate the repair of a faulty signature of a representation, as well as faulty beliefs represented in that signature. This ability is vital if we are to build autonomous agents that can evolve their internal representations to cope with a complex and changing world. It is unrealistic, for instance, to assume that innumerable agents comprising the Semantic Web[5] can be organised to share a common ontology. They will, for instance, download different versions and make local customisations. Nor is it realistic to assume that agents will be able to upload their ontologies to other agents for analysis and matching. They may not have been built with this functionality and they may not be willing to share what might be commercially confidential information

[4] `http://logic.stanford.edu/kif/kif.html`

[5] `http://www.w3.org/2001/sw/`

with a customer or a rival. So only limited channels of communication can be realistically assumed. Currently, we are adapting and extending this work to a Semantic-Web-like context in the Open Knowledge project[6].

Conclusion

In this paper we set out to establish two hypotheses. We now survey the evidence we have presented for each of them.

The representation of a problem is often the key to its solution. In his book "How to solve it", George Pólya drew on a lifetime's experience of mathematical problem-solving, at the highest level, to pass on the skills he had acquired. As we have seen, a large part of his problem-solving advice centres on the formation of representations that will simplify the problem-solving task. In the mutilated checkerboard and Milko problems, we have seen the central role that representation plays. In diSessa's experiments on freshman physics students, we can see how the choice of an inappropriate representation can render a problem insoluble. In our MECHO project, we saw that the appropriate idealization of a problem was crucial to its solution, but that the idealization task had been made unrealistically easy by the fossilization of the link between problem type and idealization. Such fossilization can cause us to underestimate the central role that representation plays outside the artificial world of the applied-maths textbook. In our DIAMOND project, we saw how the choice of diagrammatic sub-structure was the key step in making the truth of a mathematical identity immediate from the diagram. In our ORS project, we saw that a mismatch between a representation and the world in which it operates led to failures of plan execution.

Problem representations can be automatically both formed and repaired. In the Mecho project we saw how two successive representations of an informally stated problem could be automatically formed using a combination of natural language processing technology, appropriate idealization and the Marples Algorithm. The success of the subsequent problem-solving process demonstrated the success of this automatic representation formation. However, the key idealization process was made unrealistically easy due to its fossilization in this domain. The ecological modelling domain provided a more realistic idealization challenge. Idealization of ecological objects was more typical of regular, real-world problem solving, varying significantly within sub-domains, but being drawn from standard kinds. The task of selecting from among these idealization options could be partially automated, but ultimately required human intervention in order that the simulation model would meet human defined specifications. The DIAMOND project also used human intervention to select appropriate diagrammatic idealizations of algebraic expressions. However, in this domain, there is some hope that this process could be totally automated since the goal this idealization has to solve, proving a theorem, is more circumscribed. This is unfinished business.

So far, these mechanisms have constructed representations from a fixed signature. But to provide the representational autonomy required of an intelligent agent, it is also necessary to include signature formation and repair. In the CYNTHIA project, we explored the space of signature manipulation: changing function names, changing their types and arities, changing their recursive structure, etc. These changes were under human direction, but with the machine maintaining some properties of syntactic correctness, well-definedness and termination. In the HR project we showed how new concepts and conjectures could be automatically formed by generalising from examples. In the TM project we used HR in a more directed way, to repair faulty theories in order to make a nearly true conjecture be totally true. In the ORS project we showed how even the signature underlying a representation could be automatically repaired to improve its fit to the situation it was modelling.

A complete proof of this second hypothesis is unfinished business. The work described above provides partial and encouraging evidence, but a lot more remains to be done. Representational manipulation has been a neglected area of AI. It is a hard problem, but a critical one if we are to create truly autonomous agents able to react to new and unexpected situations. Such agents will need to form new representations of new worlds to solve new problems. They will have to modify and repair their existing representations to cope with a changing world with changing demands. These modifications will need to go beyond the addition and removal of facts and rules from a knowledge base. It will be necessary also to modify the *signature* of the formalism, maybe also its semantics and its logic. Finding mechanisms to automate these formation and modification processes will be one of the major goals of the *next* 50 years of AI.

[6] http://www.openk.org/

References

Bundy, A.; Byrd, L.; Luger, G.; Mellish, C.; Milne, R.; and Palmer, M. 1979. Solving mechanics problems using meta-level inference. In Buchanan, B. G., ed., *Proceedings of IJCAI-79*, 1017–1027. International Joint Conference on Artificial Intelligence. Reprinted in 'Expert Systems in the microelectronic age' ed. Michie, D., pp. 50-64, Edinburgh University Press, 1979. Also available from Edinburgh as DAI Research Paper No. 112.

Bundy, A.; McNeill, F.; and Walton, C. 2006. On repairing reasoning reversals via representational refinements. In *Proceedings of the 19th International FLAIRS Conference*.

Bundy, A. 2001. The automation of proof by mathematical induction. In Robinson, A., and Voronkov, A., eds., *Handbook of Automated Reasoning, Volume 1*. Elsevier.

Cohen, H. A. 1974. The art of snaring dragons. Technical report, La Trobe University. A revised version of LOGO Working Paper No. 28 AI Lab MIT.

Colton, S.; Bundy, A.; and Walsh, T. 1999. HR: Automatic concept formation in pure mathematics. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence, Stockholm, Sweden*, 786–791.

Colton, S. 2002. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag.

diSessa, A. 1983. Phenomenology and the evolution of intuition. In Stevens, A., and Gentner, D., eds., *Mental Models*. Erlbaum. 15–33.

Funt, B. V. 1973. A procedural approach to constructions in Euclidean geometry. Master's thesis, University of British Columbia.

Giunchiglia, F., and Walsh, T. 1992. A theory of abstraction. *Artificial Intelligence* 56(2–3):323–390. Also available as DAI Research Paper No 516, Dept. of Artificial Intelligence, Edinburgh.

Humphrey, D. 1957. *Intermediate Mechanics, Dynamics*. Longman, Green & Co., London.

Jamnik, M.; Bundy, A.; and Green, I. 1999. On automating diagrammatic proofs of arithmetic arguments. *Journal of Logic, Language and Information* 8(3):297–321.

Jamnik, M. 2001. *Mathematical Reasoning with Diagrams: From Intuition to Automation*. Stanford, CA: CSLI Press.

Lakatos, I. 1976. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press.

McCarthy, J. 1964. A tough nut for proof procedures. Stanford Artificial Intelligence Project Memo 16, Stanford University.

McNeill, F. 2005. *Dynamic Ontology Refinement*. Ph.D. Dissertation, School of Informatics, University of Edinburgh.

Nelsen, R. B. 1993. *Proofs without Words: Exercises in Visual Thinking*. The Mathematical Association of America.

Owen, S. 1990. *Analogy for Automated Reasoning*. Academic Press Ltd.

Pease, A.; Colton, S.; Smaill, A.; and Lee, J. 2004. Modelling Lakatos's philosophy of mathematics. In *Proceedings of the Second European Computing and Philosophy Conference, E-CAP2004*.

Pólya, G. 1945. *How to Solve It*. Princeton University Press.

Pólya, G. 1965. *Mathematical Discovery*. John Wiley & Sons, Inc. Two volumes.

Robertson, D.; Bundy, A.; Muetzelfeldt, R.; Haggith, M.; and Uschold, M. 1991. *Eco-Logic: Logic-Based Approaches to Ecological Modelling*. MIT Press.

Walker, J. 1975. *The Flying Circus of Physics*. John Wiley & Sons, Inc.

Whittle, J.; Bundy, A.; Boulton, R.; and Lowe, H. 1999. An ML editor based on proofs-as-programs. In *Proceedings of the 14th IEEE International Conference on Automated Software Engineering (ASE'99)*, 166–173. Earlier version available as Research Paper no. 939, DAI, University of Edinburgh.

Whittle, J. 1999. *The Use of Proofs-as-Programs to Build an Analogy-Based Functional Program Editor*. Ph.D. Dissertation, Division of Informatics, University of Edinburgh.