Applying Lakatos-style reasoning to AI problems

Alison Pease[1], Alan Smaill[1], Simon Colton[2], Andrew Ireland[3], Maria Teresa Llano[3], Ramin Ramezani[2], Gudmund Grov[1], Markus Guhe[1]

Abstract

One current direction in AI research is to focus on combining different reasoning styles such as deduction, induction, abduction, analogical reasoning, non-monotonic reasoning, vague and uncertain reasoning. The philosopher Imre Lakatos produced one such theory of how people with different reasoning styles collaborate to develop mathematical ideas. Lakatos argued that mathematics is a quasi-empirical, flexible, fallible, human endeavour, involving negotiations, mistakes, vague concept definitions and disagreements, and he outlined a heuristic approach towards the subject. In this chapter we apply these heuristics to the AI domains of evolving requirement specifications, planning and constraint satisfaction problems. In drawing analogies between Lakatos's theory and these three domains we identify areas of work which correspond to each heuristic, and suggest extensions and further ways in which Lakatos's philosophy can inform AI problem solving. Thus, we show how we might begin to produce a philosophically-inspired AI theory of combined reasoning.

1 Introduction

The philosophy of mathematics has relatively recently added a new direction, a focus on the history and philosophy of informal mathematical practice, advocated by Lakatos (1976, 1978), Davis and Hersh (1980), Kitcher (1983), Tymoczko (1998) and Corfield (1997). This focus challenges the view that Euclidean methodology, in which mathematics is seen as a series of unfolding truths, is the bastion of mathematics. While Euclidean methodology has its place in mathematics, other methods, including abduction, scientific induction, analogical reasoning, embodiment (Lakoff and Núñez, 2001), and natural language with its associated concepts, metaphors and images (Barton, 2009) play just as important a role. Mathematics is a flexible, fallible, human endeavour, involving negotiations, vague concept definitions, mistakes, disagreements, and so on, and some hold that the philosophy of mathematics should reflect this. This situation is mirrored in current approaches to AI domains, in which simplifying assumptions are gradually rejected and AI researchers are moving towards a more flexible approach to reasoning, in which concept definitions change, information is dynamic, reasoning is non-monotonic, and different approaches to reasoning are combined.

Lakatos characterised ways in which quasi-empirical mathematical theories undergo conceptual change and various incarnations of proof attempts and mathematical statements appear. We hold that his heuristic approach applies to non-mathematical domains and can be used to explain how other areas evolve: in this chapter we show how Lakatos-style reasoning applies to the AI domains of software requirements specifications, planning and constraint satisfaction problems. The sort of reasoning we discuss includes, for instance, the situation where an architect is given a specification for a house and produces a blueprint, where the client realises that the specification had not captured all of her requirements, or she thinks of new requirements partway through the process, or uses vague concepts like "living area" which the architect interprets

[1]School of Informatics, University of Edinburgh, Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom.
[2]Department of Computing, Imperial College, 180 Queens Gate, London, SW7 2RH, United Kingdom.
[3]School of Mathematical and Computer Sciences, Heriot-Watt University, Riccarton Campus, Edinburgh, EH14 4AS, United Kingdom.

differently to the client's intended meaning. This is similar to the sort of reasoning in planning, in which we might plan to get from Edinburgh to London but discover that the airline interprets "London" differently to us and lands in Luton or Oxford, or there may be a strike on and the plan needs to be adapted, or our reason for going to London may disappear and the plan abandoned. Similarly, we might have a constraint satisfaction problem of timetabling exams for a set of students, but find that there is no solution for everyone and want to discover more about the students who are excluded by a suggested solution, or new constraints may be introduced partway through solving the problem. Our argument is that Lakatos's theory of mathematical change is relevant to all of these situations and thus, by drawing analogies between mathematics and these problem-solving domains, we can elaborate on exactly how his heuristic approach may be usefully exploited by AI researchers.

In this chapter we have three objectives:

- to show how existing tools in requirement specifications software can be augmented with automatic guidance in Lakatosian style: in particular to show how this style of approaching problems can provide the community with a way of organising heuristics and thinking systematically about the interplay between reasoning and modelling (section 3);
- to show that Lakatos's theory can extend AI planning systems by suggesting ways in which preconditions, actions, effects and plans may be altered in the face of failure, thus incorporating a more human-like flexibility into planning systems (section 4);
- to show how Lakatos's theory can be used in constraint satisfaction problems to aid theory exploration of sets of partial solutions, and counterexamples to those solutions, after failed attempts to find a complete solution (section 5).

In each field we outline current problems and approaches and discuss how Lakatos's theory can be profitably applied.

2 Background

2.1 Lakatos's theory

Lakatos analysed two historical examples of mathematical discovery in order to identify various heuristics by which discovery can occur: the proof by (Cauchy, 1813) of the Descartes–Euler conjecture and the defence by (Cauchy, 1821) of the principle of continuity. He has been criticised for overly generalising, since he claimed that his method of proofs and refutations (the key method that he identifies) is "a very general heuristic pattern of mathematical discovery" (Lakatos, 1976, p. 127). For instance Feferman (1978) argues that these case studies are not sufficient to claim that these methods have a general application. We consider that our arguments in this paper support Lakatos's claim. However, while portraying existing AI work in Lakatosian terms is an interesting intellectual exercise (and a comment on the generality of his theory), we are more concerned with showing how Lakatos's heuristics can extend current AI research. In this section we describe his heuristics, so that we can show how they can be applied to AI problems in the following sections. We abbreviate the "Lakatos-style reasoning" described here to LSR.

Lakatos's main case study was the development of the Descartes–Euler conjecture and proof. This started with an *initial problem*, to find out whether there is a relationship between the number of edges, vertices and faces on a polyhedron, which is analogous to the relation which holds for polygons: that the number of vertices is equal to the number of edges. The *naïve conjecture* is that for any polyhedron, the number of vertices (V) minus the number of edges (E) plus the number of faces (F) is equal to two. Cauchy's 'proof' of this conjecture (Cauchy, 1813) was a thought experiment in

which an arbitrary polyhedron is imagined to be made from rubber, one face removed and the polyhedron then stretched flat upon a blackboard, and then various operations are performed upon the resulting object, leading to the conclusion that on this object $V - E + F = 1$ and hence prior to removing the face, the equation was $V - E + F = 2$. Most of the developments of proof, conjecture and concepts are triggered by counterexamples. Suppose, for instance, that the hollow cube (a cube with a cube shaped hole in it) is proposed as a counterexample to the conjecture that for all polyhedra, $V - E + F = 2$, since in this case $V - E + F = 16 - 24 + 12 = 4$. One reaction is to surrender the conjecture and return to the initial problem. Alternatively we might modify the conjecture to "for all polyhedra *except those with cavities*, $V - E + F = 2$" by considering the counterexample, or to "for all *convex* polyhedra, $V - E + F = 2$" by considering supporting examples (such as regular polyhedra). Another reaction might be to argue that the hollow cube is not a polyhedron and thus does not threaten the conjecture, or to argue that there are different ways of seeing the hollow cube and that one interpretation satisfies the conjecture. Lastly, we might examine the proof to see which step the hollow cube fails, and then modify the proof and conjecture to exclude the problem object.

We outline Lakatos's heuristics below, presented as differing reactions (by different parties in a discussion) to a counterexample to a conjecture, where the outcome is a modification to a particular aspect of a theory. We represent this formally for a conjecture of the form $\forall x(P(x) \rightarrow Q(x))$, supporting examples S such that $\forall s \in S, (P(s) \wedge Q(s))$ and counterexamples C such that $\forall c \in C, (P(c) \wedge \neg Q(c))$.

1. *Surrender* the conjecture, and return to the initial problem to find a new naïve conjecture. More formally, abandon the conjecture when the first $c \in C$ is found. The outcome here is a change in focus.
2. Look for general properties which make the counterexample fail the conjecture, and then modify the conjecture by excluding that type of counterexample – *piecemeal exclusion*, or if there are few counterexamples and no appropriate properties can be found, then exclude the counterexamples individually – *counterexample barring*. These are types of *exception-barring*. More formally, determine the extension of C, generating an intensional definition $C(x)$ of a concept which covers exactly those examples in C and then modifying the conjecture to $\forall x((P(x) \wedge \neg C(x)) \rightarrow Q(x))$. The outcome here is to modify the conjecture.
3. Generalise from the positives and then limit the conjecture to examples of that type – *strategic withdrawal* (this is the only method for which supporting rather than counterexamples are needed). This is the other type of exception-barring. More formally, determine the extension of S, generating an intensional definition $S(x)$ of a concept which covers exactly those examples in S, and then modifying the conjecture to $\forall x((P(x) \wedge S(x)) \rightarrow Q(x))$. The outcome here is to modify the conjecture.
4. Perform *monster-barring* by excluding the kind of problematic object from the concept definitions within the conjecture: that is, argue that the counterexample is irrelevant since the conjecture does not refer to that type of object. More formally, argue that $\forall c \in C$, $\neg P(c)$, either by narrowing an already explicit definition, or by formulating a first explicit definition of P. Each party in the discussion must then accept the new definition of P, and revise their theory accordingly. The outcome here is to modify one or more of the (sub)concepts in the conjecture.
5. Perform *monster-adjusting* by re-interpreting the counterexample as a supporting example. More formally, argue that $\forall c \in C$, $Q(c)$, again formulating and negotiating the definition as for monster-barring. The outcome here is modify one or more of the (sub)concepts in the conjecture.
6. Perform *lemma-incorporation* by using the counterexample to highlight areas of weakness in the proof. A counterexample may be global (violate a conjecture) and/or local (violate a step in the proof). If it is both global and local then modify the conjecture by incorporating the problematic proof step as a condition. If it

is local but not global then modify the problematic proof step but leave the conjecture unchanged. If it is global but not local then look for a hidden assumption in the proof which the counterexample breaks, and make this assumption explicit. The counterexample will then be global and local. More formally, use each $c \in C$ to identify flaws in the proof which can then be rectified. The outcome here is to modify either the proof or the conjecture which it purports to prove. This method evolves into *proofs and refutations*, which is used to find counterexamples by considering how areas of the proof may be violated.

The *problem of content* concerns the situation where a conjecture has been specialised to such an extent that its domain of application is severely reduced. (Lakatos, 1976, p 57) argues that a proof and theorem should explain *all* of the supporting examples, rather than just exclude the counterexamples. *Concept stretching* provides one solution, where a concept definition is widened to include a certain class of object: this is the opposite of monster-barring.

2.2 Computational accounts of Lakatos's theory

Our argument that Lakatos's theory applies to particular AI domains will be stronger if we can demonstrate the following. Firstly, we should show that it is possible to provide a computational reading of Lakatos's theory, by interpreting it as a series of algorithms and implementing these algorithms as a computer program. Secondly, we should demonstrate that the theory has already been usefully applied to other AI domains. Lastly, we should draw convincing analogies between mathematics, the domain in which the theory was developed, and the AI domains. In particular we need to identify parts of the AI domain which correspond to the key notions of mathematical conjecture, proof, concept, supporting example and counterexample. We describe our attempts to support the first two claims below, and draw appropriate analogies between mathematics and requirement specifications, planning and constraint satisfaction problems at the start of each discussion on applying LSR to these domains (sections 3, 4 and 5 respectively).

A computational model of Lakatos's theory

We have developed a computational model of Lakatos's theory, HRL[4], in order to test our hypotheses that *(i)* it is possible to computationally represent Lakatos's theory, and *(ii)* it is useful to do so (Pease et al., 2004; Pease, 2007). HRL is a multiagent dialogue system in which each agent has a copy of the theory formation system HR (Colton, 2002), which can form concepts and make conjectures which empirically hold for the objects of interest supplied. Distributing the objects of interest between agents means that they form different theories, which they communicate to each other. Agents then find counterexamples and use the methods identified by Lakatos to suggest modifications to conjectures, concept definitions and proofs. This system operated in the mathematical domains of number theory and group theory, thus demonstrating that LSR applies to domains other than topology and real analysis, and also with a machine learning data-set from inductive logic programming on animal taxonomy (Pease, 2007, chap. 10).

Applications of LSR to AI domains

We have previously built the TM system (Colton and Pease, 2005) which was inspired directly by Lakatos's techniques. TM was built to handle non-theorems in the field of

[4]The HRL system incorporates HR (Colton, 2002), which is named after mathematicians Godfrey Harold Hardy (1877 - 1947) and Srinivasa Aiyangar Ramanujan (1887 - 1920), and extends it by modelling the ideas of the philosopher Imre Lakatos (1922-1974).

automated theorem proving. Given an open conjecture or non-theorem, TM effectively performed strategic withdrawal and piecemeal exclusion in order to find a specialisation of the problem which could be proved. To do this, it used the MACE model generator (McCune, 2001) to find supporting examples and counterexamples to the conjecture, then employed the HR automated theory formation system (Colton, 2002) to learn concepts which characterised subsets of the supporting examples. The concepts HR produced were used to specialise the conjecture in such a way that the Otter theorem prover (McCune, 1994) could find a proof of the specialised conjecture. We demonstrate the effectiveness of this approach by modifying conjectures and non-theorems taken from the TPTP library of first order theorems. (While it may not be surprising that we can apply LSR to theorem proving in AI, since both operate on mathematical domains, the fact that we have both automated and usefully applied LSR supports our argument.)

Colton and Miguel (2001) have already used an indirect form of Lakatosian reasoning to reformulate CSPs. They built a system which takes as input a CSP and uses the Choco constraint programming language (Laburthe and the OCRE project team, 2000) to find simple models which satisfy the constraints. These were input to HR, which found implied and induced constraints for the CSPs. A human user interpreted these results and used them to reformulate the CSP to include these additional constraints. As an example, Colton and Miguel ran their system in the domain of quasi groups, *i.e.*, finite algebras where every element appears in every row and column. Given the quasi group axioms and the additional axiom of $(a * b) * (b * a) = a$, which defines QG3, the task was to find example quasi groups of different sizes. Their system found examples up to size 6 and these examples were passed to HR, which found the concept "anti-Abelian", *i.e.*, the constraint that no pair of distinct elements commute. It then used Otter to prove that *all* examples of QG3 are anti-Abelian, thus the extension of the examples is the same, although the intension is different. This implied constraint was then added to the CSP, which sufficiently narrowed the search space to enable the system to find examples of size 7 and 8. HR also found the concept "quasi groups with symmetry of left identities", *i.e.*, $\forall a, b(a * b = b \rightarrow b * a = a)$. Since these form a strict subset of QG3, in this case both the intent and extent are different from the original CSP. When this induced constraint was added the system found an example of size 9. This can be seen as strategic withdrawal, where the new CSP is a specialisation of the original one. While the CSPs in this example are from a mathematical domain, Colton and Miguel (2001) argue that their system could be applied to other problem classes such as tournament scheduling. The ICARUS system (Charnley et al., 2006) extended the project by Colton and Miguel (2001) by fully automating the process (omitting the human interaction).

We have also developed ideas on applying LSR to work in the AI argumentation field (Pease et al., 2009). We discuss the meta-level argumentation framework described in (Haggith, 1996), in which both arguments and counter-arguments can be represented, and a catalogue of argument structures which give a very fine-grained representation of arguments is described. Using Lakatos's case studies, we showed that Haggith's argumentation structures, which were inspired by the need to represent different perspectives in natural resource management, can be usefully applied to mathematical examples. We also showed that combining Lakatos's conjecture-based and Haggith's proposition-based representations can be used to highlight weak areas in a proof, which may be in the relationships between sets of conjectures or in the claims asserted by the conjectures. Applying Lakatos's ideas to Haggith's argumentation structures showed a way of avoiding her black box propositions, thus enabling new areas for flaws to be found and repaired. Lakatos's methods suggested new structures for Haggith (although she made no claim to have identified all structures, adding new examples to the catalogue was a valuable contribution to Haggith's work). Aberdein (2005) also discusses argumentation theory in the context of mathematics.

Hayes-Roth (1983) describes five heuristics, which are actually based on Lakatos's methods, for repairing flawed beliefs in the planning domain. He demonstrates these in terms of revising a flawed strategy in a simple card game, Hearts. In Hearts a pack of cards is divided amongst players, one player plays a card and the others must all put down a card in the same suit as the first if they have one, and otherwise play any card. The person who played the highest card in the specified suit wins that trick and starts the next. One point is awarded for each heart won in a trick, and 13 for the queen of spades (QS). The aim of the game is to get either as few points as possible ("go low") or all the points ("shoot the moon"). An example of a justification of a plan (corresponding to a mathematical proof) is "(a) the QS will win the trick, therefore (b) the player holding the QS will get the 13 points, therefore (c) this plan will minimise the number of my points"; an example of an action which is executed according to a plan (corresponding to an entity) is to "play the 9 of spades"; and an example of a concept is "a spade lower than the Queen". Counterexamples correspond to moves which follow a strategy but which do not have the desired outcome. For instance, a strategy which beginners sometimes employ is to win a trick to take the lead, and then play a spade in order to flush out the QS and avoid the 13 points. Hayes-Roth represents this as shown below (Hayes-Roth, 1983, p.230):

Plan:	Flush the QS
Effects:	(1) I will force the player who has the QS to play that card
	(2) I will avoid taking 13 points
Conditions:	(1) I do not hold the QS
	(2) The QS has not yet been played
Actions:	First I win a trick to take the lead, and whenever I lead I play a spade

The plan (analogous to a faulty conjecture) may backfire if the beginner starts with the king of spades (KS) and then wins the trick and hence the unwanted points (this situation is a counterexample to the plan). Heuristics then provide various ways of revising the plan: we show these in terms of Lakatos's methods below.

Surrender is called *retraction*, where the part of the plan which fails is retracted, in this case effect (2). *Piecemeal exclusion* is known as *avoidance*, where situations which can be predicted to fail the plan are ruled out, by adding conditions to exclude them. For example the condition "I do not win the trick in which the queen of spades is played" might be added, by assessing why the plan failed. A system can further improve its plan by negating the new condition "I win the trick in which the queen of spades is played", using this and its knowledge of the game to infer that it must play the highest card in the specified suit, and then negating the inference to get "I must not play the highest card in the specified suit". This is then incorporated into the action which becomes "First I win a trick to take the lead and whenever I lead, I play a spade which is not the highest spade". *Strategic withdrawal* is known as *assurance*, where the plan is changed so that it only applies to situations which it reliably predicts. In this case the faulty prediction is effect (2) above, and so the system would look for conditions which guarantee it. It does this by negating it, inferring consequents and then negating one of these and incorporating it into the action. For example negating effect (2) gives "I do take 13 points", the game rules state that "the winner of the trick takes the points in the trick" so we can infer that "I win the trick", then use this and the rule that "the person who plays the highest card in the suit led wins the trick" to infer that "I play the highest card in the suit led". Given that "player X plays the QS" we can now infer that "I play a spade higher than the QS" and negate it to get "I play a spade lower than the QS". An alternative heuristic, which also relates to strategic withdrawal is *inclusion*. This differs from assurance in that the situations for which the plan is known to hold are listed rather than a new concept being devised. Therefore, instead of adding "I play a spade lower than the QS" to the action, we add "I play a spade in the set {*2 of spades, 3 of spades, 4 of spades ..., 10 of spades, Jack of spades*}". *Monster-barring:* is called *exclusion*, where the theory is barred from applying to the

current situation, by excluding the situation. The condition "I do not play KS" is then added.

We can extend Hayes-Roth's example to include monster-adjusting and lemma-incorporation, where *monster-adjusting* is a type of *re-evaluation*, in which the counterexample is reinterpreted as a positive example by changing the overall strategy into shooting the moon rather than going low. In this case, getting the QS has a positive effect on the goal of winning. *Lemma-incorporation* can be seen as *consider the plan*, where the proof is considered and counterexamples to the following lemmas are sought: *(a)* the QS will win the trick; *(b)* the player holding the QS will get the 13 points; and *(c)* this plan will minimise the number of my points. This plan might suggest the counterexample of the KS which violates *(a)* (and *(b)*). Analysis of the counterexample would show that it is both local and global, and so the first lemma would be incorporated into the conjecture as a further condition. This then becomes: if (1) I do not hold the QS, and (2) The QS has not yet been played, and (3) The QS wins the trick (is the highest spade in the trick), then (1) I will force the player who has the QS to play that card, and (2) I will avoid taking 13 points.

3 Applying Lakatos-style reasoning to evolving requirement specifications

3.1 Lakatos's methods in Event-B

The process of turning informal customer requirements into precise and unambiguous system specifications is notoriously hard. Customers typically are unclear about their requirements. Clarity comes through an iterative process of development analogous to that of Lakatos's characterisation of mathematical discovery. However, conventional approaches to representing specifications lack the rigour that is required in order to truly support LSR. As a consequence, defects, omissions and inconsistencies may go undetected until late on in the development of a system with obvious economic consequences. In order to embrace Lakatos's ideas fully within software engineering requires the use of formal notations and reasoning. Adopting the rigour of formal argument, coupled with the Lakatos's methods, holds the potential for real productivity and quality gains in terms of systems development. Below we explore this idea, using Event-B (Abrial, 2009), a formal method that supports the specification and refinement of discrete models of systems. Within the context of Event-B, the methods of Lakatos can be used to reason about the internal coherence of a specification, as well as the correctness of refinements. The formal reasoning is underpinned by the generation of proof obligations (POs); mathematical conjectures that are discharged by proof.

An Event-B specification is structured into contexts and models. A context describes the static part of a system, *e.g.*, constants and their axioms, while a model describes the dynamic part. Models are themselves composed of three parts: variables, events and invariants. Variables represent the state of the system, events are guarded actions that update the variables and invariants are constraints on the behaviour described by the events. As described in the example below, in a traffic controller system, a traffic-light can be represented as a variable, an event may be responsible for switching a traffic-light to green when another traffic-light is displaying red, and an invariant may constrain the direction of the cars when a particular traffic-light is green. Events can be refined into more concrete events by adding more detailed information about the system. For example, a more concrete version of the events that change the traffic-lights to green could be achieved by adding information about pedestrian crossing signals.

In this domain, Lakatos's terminology can be interpreted in different ways. For instance, an event may be refined to a more concrete event and the refinement verified

through the use of invariants. In this case, the abstract event and the invariants can be seen as the concepts while the concrete event can be seen as the conjecture. Furthermore, in order to prove the internal coherence of a model, each invariant must be preserved over all events. In such proofs an invariant can be seen as a concept and an event as the conjecture; or vice versus. A third view is to always see the POs as the conjectures and both the invariants and events as concepts. However, in this scenario a change in the conjecture (PO) is necessarily a change in the concepts, *i.e.*, the invariants and/or events. Constants, variables and axioms can always be seen as concepts. Animating, or simulating the specification can lead to supporting examples (valid values) and counterexamples (invalid values) being obtained.

If too much detail is introduced within a single step then it may be necessary to backtrack to a more abstract level where a smaller refinement step is introduced. Additionally, within a single step, an invariant, event or variable may be abandoned if it is discovered that it is being represented at the wrong level of abstraction. For example, this might involve backtracking in order to change an abstraction, or delay the introduction of an event until later within a development. This can be seen as a type of *surrender* in which the naïve conjecture is abandoned, and the initial problem (the overall design) is revisited. However it differs in that it may not be triggered by a counterexample. Another interpretation is *strategic withdrawal*, where withdrawal is to the "safer" domain of the more abstract level.

Piecemeal exclusion involves generalising across a range for which a conjecture is false, then modifying the conjecture to exclude the generalisation. Such exclusion may be achieved by adding guards to the events associated with failed conjectures, or by making invariants conditional. If the generalisation step is omitted then this would be an instance of counterexample barring. *Strategic withdrawal* has a similar effect in the sense that a guard is added, or the invariant is made conditional. However, the process of discovery is different in that it focuses on the supporting examples.

In *monster-barring* we argue that the values leading to a counterexample are not valid. Such values may for example be the input of an event. This type of argument is introduced in a model by adding an additional invariant. Regarding *monster-adjusting*, the counterexamples may be used to modify invariants or events (but without restricting them). An illustration of this case is the introduction of an additional action to an event. Finally, a failure in a proof can be the result of a missing axiom, and lemma-incorporation involves adding an axiom as a result of the counterexample.

3.2 An example in Event-B

We illustrate Lakatos's discovery and justification methods for evolving Event-B specification using Abrial's "Cars on a Bridge" example Abrial (2009). Figure 1 presents the essential details of the example, where the events are identified in bold. We will focus our discussion on a small part of Abrial's model, which is explained next. The example consists of an island connected to the mainland by a bridge. The bridge has one lane, and the direction of the traffic is controlled by traffic-lights on each side. A maximum number of cars are allowed on the bridge/island, and is denoted by d. Variables a and c denote the numbers of cars travelling towards the island and towards the mainland respectively, while b denotes the number of cars on the island. These variables should be seen as part of the specification of the *environment*, since they are not directly controlled by the system. *ml_tl* and *il_tl* describe the colour of the traffic-lights on the mainland and island respectively, and can be seen as the *system variables*. Events *ML_tl_green* and *IL_tl_green* change the traffic-lights to green, and events *ML_out* and *IL_out* model cars leaving the mainland and the island respectively. We delay to later discussion of how traffic-lights are switched to red. The following invariants state the

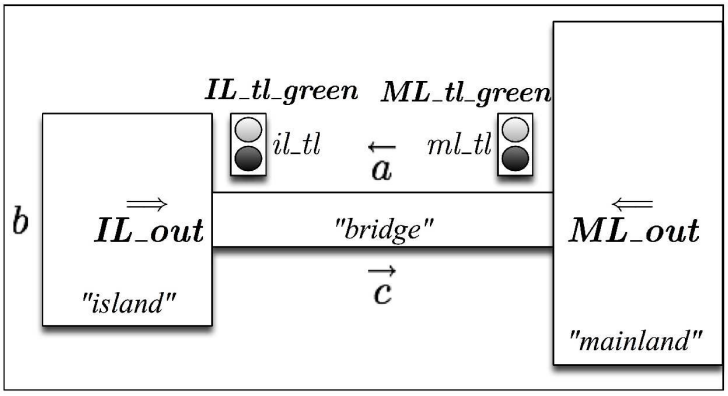

Figure 1: An Event-B example

conditions when the traffic-lights are green.

$$ml\_tl = green \Rightarrow a + b < d \wedge c = 0 \quad (inv1)$$
$$il\_tl = green \Rightarrow 0 < b \wedge a = 0 \quad (inv2)$$

In Event-B each model must contain an unguarded *Initialisation* event that defines the valid initial state(s). In the example, we require no cars on the bridge/island and the lights set to red, *i.e.*,

$$Initialisation \mathrel{\widehat{=}} \textbf{Begin } a, b, c := 0, 0, 0 \parallel ml\_tl, il\_tl := red, red \textbf{ End}$$

This initialisation produces a counterexample with respect to the invariant *inv2*, *i.e.*,

$$red = green \Rightarrow \underline{0 < 0} \wedge 0 = 0$$

Note that the false part is underlined. The counterexample highlights a weakness in the specification, which can be fixed with the *lemma incorporation* method, leading to the introduction of an additional axiom of the form $red \neq green$.

Now consider the following definition of the *ML_out* event, which models a car leaving the mainland:

$$ML\_out \mathrel{\widehat{=}} \textbf{When } ml\_tl = green \textbf{ Then } a := a + 1 \textbf{ End}$$

Here, *ml_tl=green* is the guard of the action which increments a by 1. That is, a car can only leave the mainland when the traffic-light is green. Again, a counterexample with respect to invariant *inv2* is found, *i.e.*,

$$green = green \Rightarrow 0 < 2 \wedge \underline{1 = 0}$$

Using the *piecemeal exclusion* method, the conjecture can be fixed via the counterexample, by restricting either *il_tl* or a. We prefer not to restrict the environment whenever possible, therefore we use *il_tl*. The only way to make $il\_tl = green$ false, is by assigning *il_tl* the value *red*. *ML_out* is then restricted by this additional guard as follows:

$$ML\_out \mathrel{\widehat{=}} \textbf{When } ml\_tl = green \underline{\wedge\ il\_tl = red} \textbf{ Then } a := a + 1 \textbf{ End}$$

Note that the counterexample is used directly; therefore, this is an instance of the *counterexample barring* method. *IL_out* has a similar failure and patch for invariant *inv1*, and becomes:

$$IL\_out \mathrel{\widehat{=}} \textbf{When } il\_tl = green \underline{\wedge\ ml\_tl = red} \textbf{ Then } b, c := b - 1, c + 1 \textbf{ End}$$

where the underlined part is added as a result of the counterexample barring method. However, instead of restricting the applicability of events using *piecemeal exclusion*, we can *monster-bar* the counterexample via an invariant. For instance, if we step back and analyse both failures, it becomes clear that the newly introduced guards can be weakened by the existing guards, *e.g.*, within *ML_out*, $il\_tl = red$ then becomes $ml\_tl = green \Rightarrow il\_tl = red$. In fact, both these failures can be generalised to the same conjecture, which we monster-bar by adding the following invariant:

$$il\_tl = red \lor ml\_tl = red \quad (inv3)$$

Informally, this invariant is an obvious requirement since it formalises that cars are only allowed on to the bridge in one direction at a given time. Nevertheless, invariant *inv3* is not preserved by the IL_tl_green and ML_tl_green events. We will only discuss the latter:

$$\begin{array}{lll} ML\_tl\_green & \widehat{=} & \textbf{When } ml\_tl = red \ \land \ a + b < d \ \land \ c = 0 \\ & & \textbf{Then } ml\_tl := green \ \textbf{End} \end{array}$$

Here the counterexample arises if *il_tl* is *green* when the ML_tl_green event is executed. Here, we apply the *monster-adjusting* method and use the counterexample as a supporting example. This results in the introduction of an additional action which eliminates the counterexample. The action sets *il_tl* to red:

$$\begin{array}{lll} ML\_tl\_green & \widehat{=} & \textbf{When } ml\_tl = red \ \land \ a + b < d \ \land \ c = 0 \\ & & \textbf{Then } ml\_tl := green \ || \ \underline{il\_tl := red} \ \textbf{End} \end{array}$$

IL_tl_green is monster-adjusted in the same way.

3.3 Discussion

The example developed above was supported by the Rodin tool-set (Abrial et al., 2009) and ProB (Leuschel and Butler, 2008). That is, the management of specifications, and the generation of POs, proofs and counterexamples were all automated via the tool-set. In contrast, the high-level Lakatos style analysis was undertaken manually. Our current programme of research is concerned with augmenting the existing tools with automatic guidance in the style of Lakatos. Our approach involves combining heuristic knowledge of proof and modelling, to achieve what we call *reasoned modelling*. Lakatos's methods provides us with a way of organising our heuristics and thinking systematically about the interplay between reasoning and modelling. Moreover, we would like to raise the level of interaction by building upon graphical notations such as UML-B (Snook and Butler, 2008).

4 Applying Lakatos-style reasoning to planning

4.1 Lakatos's methods and planning

The ability to formulate and achieve a goal is a crucial part of intelligence, and planning is one way to tackle this (another way might be situated reflex action, for instance to achieve an implicit goal like survival). The traditional approach to planning in AI involves designing algorithms which take three types of input, all in some formal language: a description of the world and current state, a goal, and a set of possible actions that can be performed. The algorithms then output a plan consisting of a set of actions for getting from the initial state to the goal (this is known as batch planning). These work in various ways, for instance by refinement (gradually adding actions and constraints to a plan), retraction (eliminating components from a plan), or a combination of both (transformational planners). Plans can be constructed from scratch (generative planners) or found via some similarity metric from a library of cases (case-based planners). Traditional approaches to planning employ simplifying assumptions such as

atomic time (the execution of an action cannot be interrupted or divided), deterministic effects (the same action on the same state of the world will always result in the same effect), an omniscient planning agent, and the assumption that the rest of the world is static (it only changes via the agent's actions). Weld (1994) describes these characteristics of classical planning in further detail. There are now many variations to the traditional approach which reject some of these simplifying assumptions to get a more sophisticated model, for instance, Donaldson and Cohen (1998). We describe two such approaches in this section and discuss how Lakatos-style reasoning might be used to interpret or extend them.

Different interpretations of the analogy

There are strong similarities between the planning domain and a procedural notion of mathematics (as opposed to declarative mathematics). In planning, given certain preconditions, background information about the world and a goal, the aim is to construct a plan which starts from the preconditions and ends with achieving the goal. In mathematics, given an arbitrary object of a certain type and mathematical background knowledge such as axioms and theorems, and the goal of showing that certain properties hold true of the object, the aim is to construct a proof in which mathematical operations are performed on the object and it is demonstrated that the required properties must hold true. (Since it was arbitrary, such a proof would demonstrate that these properties hold for all objects of that type.) Note that the proof may include recursion and case splits, which does not affect our argument. The analogy is particularly clear in Lakatos's Descartes–Euler case study as Cauchy's proof is procedural: it is represented as a series of actions to be performed on an object which starts as a polyhedron and is transformed via the actions to a two-dimensional object, a triangulated graph, etc. That is, given the input of an arbitrary polyhedron, background mathematical knowledge, and the goal of showing that $V - E + F = 2$ for this polyhedron, Cauchy's proof consists of a set of actions which achieve the goal: *i.e.*, it is a plan. This analogy is strengthened by the "Proofs as processes" perspective presented by Abramsky (1994) in which proofs are seen as concurrent processes (or processes as morphisms) and by Constable's work connecting programs and mathematical proofs, such as Constable and Moczydłowski (2006).

If we accept the analogy between the planning domain and mathematics then we would expect there to be a productive relationship between LSR and planning methods in AI. For instance, LSR should suggest ways in which a rudimentary plan might evolve by being patched or refined in the face of failure; how agents may communicate in social and collaborative planning; how plans can be formed and revised without an omniscient planning agent; when and how beliefs may be revised, or inconsistencies in a plan handled; how a dynamic environment can be used to develop a plan, etc. More specifically, Lakatos's theory and the extended theory in (Pease, 2007) can suggest when a plan should be abandoned (surrendered) and another one formed; how a plan might be modified to exclude cases which are known to fail (piecemeal exclusion), or limited to cases for which the plan is known to work (strategic withdrawal); how cases which fail a plan can be reconstrued such that the plan was not intended to cover them (monster-barring), or examples thought to cause failure reconstrued as supporting example, perhaps by a different interpretation of what it means to achieve a goal (monster-adjusting); how failure can be used to highlight areas of weakness in a plan and then strengthen them (lemma-incorporation), and how examination of steps in a plan could suggest sorts of cases which might fail them (proofs and refutations).

In order to apply LSR to planning we need to have analogical concepts of mathematical conjecture, proof, supporting example and counterexample. There are at least two rival interpretations of mathematical conjecture in the planning domain. Firstly, given a situation s which satisfies certain preconditions, there exists a set of actions such that performing them on s will result in another situation which satisfies certain

effects. The second interpretation is that given a set of preconditions, a set of actions to perform on a situation, and a set of effects, if a situation satisfies the preconditions then the result of performing the actions will be another situation which satisfies the effects. In the first interpretation we conjecture that *there exists* a set of actions which will turn one specific situation into another, and in the second interpretation we conjecture that a certain *given* set of actions will turn one specific situation into another. Put formally using the "Result" operator from situation theory, this is:

First interpretation: $\exists A_T$ such that $\forall s \in S_T$, $(P_T(s) \rightarrow E_T(\text{Result}(A_T, s)))$, where A_T is a set of actions in the theory, S_T is the set of possible situations in the theory, $C_T(s)$ means that s satisfies a set of criteria C in the theory (which may be preconditions P_T or effects E_T).

Second interpretation: A_T is a set of actions in the theory such that $\forall s \in S_T$, $(P_T(s) \rightarrow E_T(\text{Result}(A_T, s)))$, where S_T is the set of possible situations in the theory, $C_T(s)$ means that s satisfies a set of criteria C in the theory (which may be preconditions P_T or effects E_T).

In the first interpretation, mathematical proof would correspond to the plan. This fits our notion of a mathematical conjecture in that we can discover and understand it without knowing the proof: for example, Goldbach's conjecture that "every even integer greater than 2 can be expressed as the sum of two primes", which is one of the oldest open conjectures in number theory. (Polya (1962) suggests how conjectures might arise, without considering proof.) However, the corresponding notions of supporting examples and counterexamples are problematic. There is no notion of a supporting example which is independent of a proof. Similarly, although it may be possible to prove that a situation satisfying certain preconditions cannot be transformed into one which satisfies certain effects, it is difficult to falsify an existential claim. Under this interpretation then, only one of Lakatos's methods, local-only lemma-incorporation (which only involves counterexamples to a step in the proof, in this case to an action), has an obvious analogue.

In the second interpretation, there *is* a notion of supporting and counterexamples: a situation s_1 such that $P_T(s_1) \wedge E_T(\text{Result}(A_T, s_1))$, and a situation s_2 such that $P_T(s_2) \wedge -E_T(\text{Result}(A_T, s_2))$, respectively[5]. Thus, there are analogues to Lakatos's methods. However, the corresponding notion of proof is a justification of a plan, *i.e.*, *why* it would work. Thus, Lakatos's methods would focus on refining the justification rather than the plan: this may be contrary to the desired focus. It may be that the connection between the two interpretations is that of synthesis (the first interpretation) and verification (second interpretation): we can also see the distinction in Lakatosian terms as the initial problem (first interpretation) and naïve conjecture (second interpretation). We may be able to rectify the situation somewhat if we restrict ourselves to a finite domain. Consider, for instance, planning in the context of a game such as chess. A conjecture would take the form "there exists a path from the current state to the goal state" (where the goal state could be a winning state or any other desirable state). Under this analogy, mathematical *axioms* and *inference rules* would map respectively to the start state and the legal moves which each piece can perform. *Theorems* and *lemmas* would correspond to states towards which a path can be shown to exist from the start state[6].

[5] We do not consider here whether a situation s which does not satisfy the preconditions, $-P_T(s)$, would form a supporting example of the conjecture, as dictated by material implication, or merely be considered irrelevant.

[6] Note that this process may appear to be the opposite of the traditional way in which mathematics is thought to be done, since games start in the start state, whereby a conjecture is (presumably) first suggested and then a mathematician tries to show that there is a path from the conjecture to the axioms. In this case, our games analogy seems closer to work by (McCasland and Bundy, 2006; McCasland et al., 2006), where every new statement follows on the from axioms or theorems and is necessarily either a lemma or theorem itself (depending on how interesting it's judged to be). However, games are not normally planned one move

Since we reserve the term "theorem" in mathematics for *interesting* proved statements, we map this to interesting board states, and use the lower status term "lemma" for less interesting board states; intermediate states between the interesting ones. *Entities* correspond to each individual piece, for instance the pawn in square *b2* in the start state is an entity, and *concepts* to types of piece (for instance the concept pawn, which has an extensional definition of all sixteen pawns and an intensional definition of an entity such that it starts in the second and seventh row, advances a single square or two squares (the first time it is moved from its initial position), capture other entities diagonally (one square forward and to the left or right) and may not move backwards). Concepts might be split further into sub-concepts, for instance "pawns" into "white pawn" and "black pawn", just as the concept "number" might be split into "even number" and "odd number". Under this interpretation the notion of supporting and counter examples now makes sense: a *supporting example* for a conjecture would be an entity for which a known path exists from its current state to the goal state. A *counterexample* would be an entity for which it is known that no path exists between its current state and the goal state (for example, if the goal state involves both black bishops on a square of the same colour). This approach more accurately captures the sort of mathematics that Lakatos describes, since it is possible to formulate a conjecture without any support or counterexamples, and to find supporting or counterexamples without having a proof.

4.2 An example: structural and semantic misalignment in the context of planning

Developments such as the semantic web and the grid, in which large numbers of agents with different, evolving ontologies interact in a highly dynamic domains without a centralised agent, have raised the need for automated structural and semantic re-alignment. That is, if two interacting agents find that they have different representations or semantics in a given context, then there is a need to able to automatically resolve this on the fly. McNeill and Bundy (2007) have developed an ontology refinement system, ORS, which automatically re-aligns some part of its ontology with that of some part of the world, in the face of a mismatch. This works in the context of planning, and contrasts classical planners. ORS is able to recursively create and execute plans, detect and diagnose failures, repair its ontologies and re-form an executable plan, to avoid a known failure. In this section we discuss this work in the context of LSR.

The main contribution of ORS is the ability to diagnose and repair ontological mismatches discovered during agent communication. The system repairs its ontologies in the face of a mismatch by making changes to its predicates, action rules and individual objects so that the particular problematic representation becomes identical to that of the agent with whom it is communicating.

ORS can change its action rules by adding or removing preconditions or effects. Adding a precondition corresponds to piecemeal exclusion, and removing one is related to Lakatos's problem of content. With regard to mismatches in the effects of an action, ORS is able to explicitly add or remove effects. There is an interesting link to Lakatos's theory here: in his (only) example of local-only lemma-incorporation, in which the preconditions (a triangulated network) are satisfied and the action (removing a triangle) can be performed but the effects (the value of the equation $V - E + F$ is unchanged) are not as predicted (removing an inner triangle *does* change the value of $V - E + F$ by reducing it by 1). Given the counterexample, or mismatch, one possibility is to add more effects, for instance "either $V - E + F$ is unchanged or it is reduced by 1", or more

at a time, and the typical situation is where a player has a goal/subgoal state in mind, can see how some pieces would get there and forms the hypothesis that it is possible to get all pieces to their required position. The player then works top down and bottom up to form a planned path from current to desired board state, a similar way to that in which mathematicians are thought to work.

specifically, "there are now three possibilities: either remove an edge, in which case one face and one edge disappear; or remove two edges and a vertex, in which case one face, two edges and a vertex disappear; *or* we remove one face, in which case only one face disappears", where the latter effect is the new one to be added. However, this would break the proof. Therefore we want to preserve the effect and make changes elsewhere to compensate. In this example the patch is to change the action to "removing a *boundary* triangle". ORS cannot currently change actions themselves: this idea, in which the original action is replaced by one which is a subtype of it, might be a useful extension.

There is no analogue of strategic withdrawal in ORS: repairs are only made if there is a mismatch. A way of incorporating this method would be to observe that a plan which has worked consistently in a number of examples contains a general predicate, for example the "Paper" predicate, which has only ever been invoked by a subtype of that predicate, such as "PdfPaper", and thus change the general case to the specific. This (unprovoked) refinement might be useful if the goal were to form a fool-proof plan which is known to work (as opposed to the current context of McNeill and Bundy's work, in which a plan is formulated in order to achieve a specific desired goal, and deleted once this has been successfully carried out).

ORS is also able to change the names and types of individual objects, where types may change to a sub or a super-type, one which is semantically related in a different way, or one which is not semantically related. Changing a type to a super-type, such as "Paper" to "Item" is an example of the first aspect of monster barring, in which the type of a problematic object might be changed from "polyhedron" to "three-dimensional object" (note that in monster-barring however, there might not be a replacement type, just the observation that object x is not of type T). The second aspect of monster-barring, in which the focus then turns from an individual object to a concept, or predicate, is represented by the ability of ORS to changing the name, arity, argument type, order of argument and predicate relationships for a predicate mismatch. In particular, when detail is added to a predicate, *i.e.* a refinement is performed, this can be seen as a form of monster-barring. For instance, ORS is able to replace a predicate name by one which is a subtype (*e.g.*, change "Paper" to "PdfPaper") in order to match that of the communicating agent, to avoid failure. This is analogous to changing the predicate "solid whose surface consists of polygonal faces" (which includes the hollow cube) to "surface consisting of a system of polygons" (which excludes the the hollow cube). Conversely, ORS can able to replace a predicate name by one which is a super-predicate (*e.g.*, change "PdfPaper" to "Paper"); analogous to Lakatos's concept stretching.

There is no analogue of monster-adjusting in ORS. An example of this might be to change the *value* that an argument takes, rather than its type. (It is possible to do this in ORS by taking away an argument and then adding one, but this requires extra work as there is nothing to link the two types, so the latter type would need to be determined independently.) In Lakatos's example of the star-polyhedron, suppose that a polyhedron is represented as a predicate including arguments of type "natural number" corresponding to the number of faces, edges and vertices: *i.e.* as:

$$\text{polyhedron}(PolyhedronName, NumberFaces, NumberEdges, NumberVertices, \vec{x}).$$

Then the original interpretation of a star-polyhedron (Lakatos, 1976, p. 16), in which it is raised as a counterexample, would be represented thus:

$$\text{polyhedron}(\textit{star-polyhedron}, 12, 30, 12, \vec{x}).$$

The later interpretation in which it is a supporting example (Lakatos, 1976, p. 31), would be:

$$\text{polyhedron}(\textit{star-polyhedron}, 60, 90, 32, \vec{x}).$$

One can imagine this being useful in the context of McNeill and Bundy's paper example if, for instance, two researchers are collaborating on a paper and the first has made changes to the value (but not type) of any of the arguments "PaperTitle",

"WordCount", or "Format" which the second has not recognised: *paper(PaperTitle, WordCount,Format)*. In this case, the second researcher would need to update his or her ontology.

ORS also uses the notion of surprising questions. These are questions asked by a service provider agent to a planning agent, which do not pertain to the planning agent's preconditions of the action to be performed. If a surprising question has been asked directly before a failure, then these are used to locate the source of the problem.

A further example: the slot machine

McNeill and Bundy illustrate some of their ideas with a hypothetical example of an agent buying something which costs £5, from a slot machine. We suggest a set of actions in order to see the example as the following conjecture: "If I have £5 (preconditions) and I perform the plan (set of actions) then I can obtain the item (effect)", where the plan (which roughly corresponds to a proof idea, with the reservations discussed above) is:

```
(1) insert money into slot,
(2) select and press button,
(3) empty the tray.
```

Suppose that the agent has a £5 note and can perform the actions in the plan. McNeill and Bundy suggest modifications that might take place:

- It is discovered that the machine accepts only coins, not notes. While McNeill and Bundy do not elaborate on how this might be discovered, we can imagine that this is a case of *lemma-incorporation* where the counterexample is both global (given the preconditions the goal has not been achieved) and local (the agent cannot carry out step *(1)* since the note will not fit into the slot). The concept "items which satisfy the problem lemma" is then formed, in this case "money in a form which will fit into the slot", *i.e.*, coins and this concept incorporated into the conjecture, in this case into the preconditions. Thus the conjecture becomes "If I have £5 in coins then I can obtain the item". Alternatively we could insert an extra action into the the plan *(1a)* convert money into coins, and then change what was previously *(1)* to *(1b)* insert *coins* into slot. This is the same case as the paper format example below.
- The agent then finds that the machine does not take the new 50p coin. We can see this as an example of *hidden lemma-incorporation* since the counterexample is global (given the preconditions the goal has not been achieved) but not local (we seem to be able to perform each step). According to Lakatos's *retransmission of falsity* principle (Lakatos, 1976, p. 47), if there is a problem with the conjecture then there must be a problem with the proof. In this case we examine each of the steps for a hidden assumption, which is marked by a feeling of surprise when it is violated. We might find that when carrying out step *(1)*, while we could insert the coin into the slot it simply dropped down into the tray. To someone who had used slot machine previously this might result in the first notion of surprise that we developed in Pease et al. (2009), when an entity does not behave in the expected way, where the "expected way" has been learned from previous examples. In all other cases the inserted money did not fall into the tray (analogous to the Cauchy example where we expect that having removed a face from a polyhedron and stretched it flat on a blackboard, we are left with a connected network). Therefore this hidden assumption should now be used to form a new concept which then becomes an explicit condition which is incorporated into the plan and the conjecture. This might result in the new concept "coins which are accepted by the machine", the modified conjecture "If I have £5 in coins which are accepted by the machine which then I can obtain the item", and a modified plan, with

first step now: "*(1)* insert money into slot so that it does not fall into the tray". Alternatively, we could see this example as exception-barring, where the concept "new fifty pence piece" is found and the conjecture becomes "If I have £5 in coins except for the new fifty pence piece, then I can obtain the item".

- The agent finds that some (perhaps old or worn) coins are unexpectedly rejected, and has to further modify the preconditions to exclude these particular coins. This also could be modified in the same way as the *hidden lemma-incorporation* above (and if being carried out chronologically then the concept "coins which are accepted by the machine" would be expanded to exclude the old coin). Alternatively, we could see this as a case of counterexample-barring, where no generalised concept covering the the counterexample is found, and so this specific coin is barred. In that case the conjecture would be modified to: "If I have £5 in coins except for this problematic one, then I can obtain the item".
- McNeill and Bundy then discuss the situation when an agent finds that the machine accepts coins which it not designed to accept, such as foreign or toy coins (again, they do not discuss how this may be found). This is a case of concept stretching, in which the problem of content is addressed by widening the domain of application: this has a valuable application since usually the weakest, or most general preconditions are more desirable.

The formulation of new concepts such as "£5 in coins", "coins except the new 50 pence piece" "coins except this particular coin", "Sterling coins and these similar foreign coins" and the subsequent modifications to the conjecture are easily describable in Lakatosian terms.

4.3 Discussion

McNeill and Bundy's approach has several commonalities with Lakatos's work. They both start from the same point, when a rudimentary proof or plan has been suggested (Lakatos claims that his discussion "starts where Polya stops" (Lakatos, 1976, p. 7), referring to Polya's work on finding a naïve conjecture (Polya, 1945, 1954), and the main thrust of McNeill and Bundy's system starts once a plan has been generated using a classical planner). Both are triggered by counterexamples or failures, and in both cases the aim is not to match the whole mathematical belief system or ontology, but to find local agreement on a particular problem. In both, the notion of surprise is used to guide repair and in particular to suggests *where* two different ontologies may differ. Both approaches are also highly recursive, with the methods being applied as many times as necessary. In Lakatos's case, the methods are repeated until agreement between mathematicians has been reached (which may later be reneged), or until the domain of application has become too narrow – the "problem of content". In McNeill and Bundy's case ontology refinement is carried out until either the goal has been achieved or it becomes impossible, given the updated ontology, to form a plan to achieve the goal.

Perhaps the most important difference between McNeill and Bundy's approach and Lakatos's work is motivation: Lakatos describes situations in which people want to *understand* something, McNeill and Bundy describe situations in which people want to *achieve* something. McNeill and Bundy's case studies describe a pragmatic approach in which a plan which works well enough to achieve a goal in a specific (possibly one-off) situation is sought: they are not looking for a general, fool-proof plan (we want a slot machine to work, we do not want to understand it). A closer analogy to Lakatos in the planning domain would be someone who wants to write a *generally* usable plan, such as a set of instructions for assembling a piece of flat-pack furniture. Connected to this difference in motivation is a different attitude to counterexamples: Lakatos views them as useful triggers for evolving a theory (proceed by trying to falsify), and McNeill and Bundy view them as obstacles to be overcome (proceed by trying to satisfy a goal).

In developing ORS, McNeill and Bundy made several simplifying assumptions. Further versions of the system could use LSR in order to suggest ways of dealing with more complex situations. Another example is that if it is possible in ORS, then the planning agent will always change its own ontology in the face of a miscommunication. This bypasses issues of trust, status, entrenchment of a belief of representation, and so on. Lakatos indirectly discusses willingness to change one's ontology in order to better fit with that of collaborators.

LSR has a useful application in the planning domain. Consider, for example, the conjecture in the domain of flat packed furniture "given this flat pack kit (preconditions), the item of furniture (goal) can be constructed", where the notion of proof corresponds to the set of instructions (plan). One can imagine using LSR to improve upon a poorly written set of instructions, to find hidden assumptions and make them explicit. Developments in structural and semantic misalignment, in the context of planning as well as other areas, and in particular flexible and dynamic thinking, are of key importance to the semantic web, the grid and other areas. Thus, approaches that may contribute to their development are worth exploring: we hold that LSR is one such approach.

5 Applying Lakatos-style reasoning to constraint satisfaction problems

5.1 Lakatos's methods and constraint satisfaction problems

A constraint satisfaction problem (CSP) consists of a set of problem variables, a domain of potential values for each variable, and a set of constraints specifying which combinations of values are acceptable (Tsang, 1993). A solution specifies an assignment of a value to each variable in such a way as to not violate any of the constraints. CSPs are usually represented in a tree-like structure, where a current search path represents a current set of choices of values for variables. CSPs appear in many areas, such as scheduling, combinatorial problems and vision. One example is the classic N-queens problem: given N, the solver is supposed to place N-queens on an $N * N$ chess board in such a way that none of the queens can threaten the others.

A conjecture corresponds to a current search path, or modular solution, which is hypothesised to satisfy all the constraints. Supporting examples correspond to constraints which are satisfied by the model, and counterexamples to constraints which are violated by the model. We show some correspondences to Lakatos's methods below.

Surrender would entail abandoning a current search path (model) as soon as a single inconsistency is encountered (*i.e.*, a constraint is violated). This is most commonly used for CSPs, and triggers backtracking techniques. Freuder and Wallace (1992) develop techniques for partial constraint satisfaction, which are analogous to retrospective, prospective and ordering techniques for CSPs (a comparable search tree in mathematics might have an initial branching of the different equations under consideration, which of course might be dynamic, *i.e.*, new equations are created in the light of previous ones and added as new branches). These are necessary if there is no complete solution at all (the problem is over-constrained), or we cannot find the complete solution with the resources given (some algorithms are able to report a partial solution while working on improving this solution in the background if and when resources allow), and can be seen as *piecemeal exclusion* and *strategic withdrawal*. Constraints may be weakened by enlarging a variable domain (introduce a new value that a variable might take), enlarging a constraint domain (deciding that two previously constrained values are acceptable), removing a variable (one aspect of the problem is dropped), or removing

a constraint (deciding that any combination of two previously constrained variables is acceptable). Of particular interest to us is Freuder *et al.*'s position on alternative problems: "We suggest viewing partial satisfaction of a problem, P, as a search through a space of alternative problems for a solvable problem 'close enough' to P." (Freuder and Wallace, 1992, p. 3). This has a very clear analogue in Lakatosian terms, where 'conjecture' is substituted for 'problem', and 'provable' for 'solvable'. They go on to argue that a full theory of partial satisfaction should consider how the entire solution set of the problem with altered constraints differs from the solution set of the original problem, as opposed to merely considering how a partial solution requires us to violate or vitiate constraints: that is, they compare problems rather than violated constraints.

Monster-barring and *monster-adjusting* would correspond to a claim that the proposed counterexample constraint is not a valid constraint, and formulate properties that a valid constraint must have, or a claim that the model *does* satisfy the problem constraint. Flexible (or soft), as opposed to conventional, CSPs relax the assumption that solutions (or models) must satisfy every constraint (imperative) and that constraints are either completely satisfied or else violated (inflexible). In particular, fuzzy CSP represent constraints as fuzzy relations, where their satisfaction or violation is a continuous function. Ruttkay (1994) discusses the issue of soft constraint satisfaction from a fuzzy set theoretical point of view.

5.2 An example: a constraint satisfaction problem in scheduling

To explore the possibility of using Lakatos's ideas in constraint solving, we performed a small, hand-crafted experiment. We wrote a simple constraint satisfaction problem which models a scheduling problem (a common problem type for which CSP solvers perform very well). In the model, there are five people who need to be scheduled for an appointment at a particular time and a particular place. The CSP was designed so that there was in fact no solution. However, if we reduce the number of variables in the CSP, there are indeed solutions. This models the situation with Lakatos, if we consider the variables which we don't solve for as being the counterexamples to the existence proof of a full schedule for the five people. In addition to the CSP, we also randomly generated some data which describes the five people in the scheduling problem. We defined ten predicates of arity one: nurse, pilot, busy, teacher, parent, professional, doctor, live_north_london, and live_south_london. For each person to be scheduled, we randomly chose between 1 and 10 predicates to describe them, for instance, person four was described as a busy parent who is a pilot.

We wrote a wrapper to find all the partial solutions to the CSP, and to determine which variables (people) the solution did not cover. We found that there were 10 schedules which worked for four of the five people, 110 schedules for three people, 170 schedules for two people and 40 schedules for one person. In addition, for each of the partial solutions, we took the list of omitted people and used them as the positive examples in a machine learning classification problem (with the non-omitted people becoming the negatives). In particular, we used the background information about the people (*i.e.*, being a nurse, pilot, etc.), in a session with the Progol inductive logic programming machine learning system (Muggleton, 1995). In each case, we asked Progol to determine a general property of the omitted people. We removed the duplicate cases, *i.e.*, different partial solutions of the CSP which managed to schedule the same subset of people. In total, after this removal, there were 5 cases where four people were scheduled, 10 cases where three people were scheduled, 11 cases where two people were scheduled, and 5 cases where one person was scheduled.

When Progol was run with the machine learning problems, we checked its output for a general solution. That is, if Progol stated that the unscheduled people had a particular set of properties that the scheduled people did not share, we counted this as a success. If, however, Progol had to resort to using the name of one or more people

in its solution, we counted this as a failure. We found that Progol was only able to find solutions to 5 of the 31 cases. This is largely due to the very limited amount of data available: in many cases, the compression of the answer was not sufficiently high, so Progol chose not to supply an answer. As an illustrative example, the CSP solver found a schedule for 3 of the 5 people, and Progol highlighted the fact that the two unscheduled people were both pilots (and none of the scheduled people were pilots). Note that we ran the experiment again with different random data for the background of the people to be scheduled, and Progol solved 4 of the 31 problems.

5.3 Discussion

Along with our work on TM, HR with CSPs and ICARUS, described in section 2.2, this simple experiment hints at the potential for applying Lakatos-inspired methods in constraint solving. The approach contrasts with existing CSP reformulation approaches which tend to change the CSP constraints, rather than the variables. For instance, the CGRASS system applies common patterns in hand-transformed CSPs, in order to improve the CSP model gradually (Frisch et al., 2002). To the best of our knowledge, no CSP reformulation approach appeals to a machine learning system to offer further advice when a solving attempt fails. Note that the TAILOR solver learns parameters for implied constraints (Gent et al., 2009), and the CONACQ system uses version space learning to define a CSP model from scratch given only positive and negative examples of solutions for it. However, these uses of learning in constraint solving are different to our approach.

6 Conclusions and future work

We have given some simple examples of how LSR might be applied to AI problems, and argued that an automation of the type of reasoning that Lakatos describes would be profitable in these domains. Clearly, the examples in this paper are not the only examples of LSR in AI domains, since programs may have implicit aspects of LSR which, while not directly based on LSR, we can link to one of his methods. For example, Skalak and Rissland (1991) indirectly show how LSR might be applied to AI and legal reasoning in their theory of heuristics for making arguments in domains where "A rule may use terms that are not clearly defined, or not defined at all, or the rule may have unspoken exceptions or prerequisites" (Skalak and Rissland, 1991, p 1). In this case, their term *rule* corresponds to the mathematical term *conjecture*, *term* to *concept*, *case* to *entity*, and *argument* to *proof*. In particular, Skalak and Rissland (1991) are interested in cases where terms within a rule are open to interpretation, and different parties define the term differently according to their point of view: this corresponds very closely to Lakatos's method of monster-barring. Skalak and Rissland (1991) discuss argument moves which use cases to determine which interpretation of an ambiguous term in a rule is to be adopted. These moves are implemented within CABARET (Rissland and Skalak, 1991). Winterstein (2004) provides another example. He devised methods for representing and reasoning with diagrams, and argued that his *generalisation method* can be seen as a simple form of Lakatos's method of strategic withdrawal (Winterstein, 2004, p. 69). This method analyses positive examples of a proof, abstracts the key features from these examples, and then restricts the domain of application of the theorem and proof accordingly.

We have described analogies between Lakatos's theory of mathematical evolution and the fields of evolving requirement specifications, planning and constraint satisfaction problems. Showing the relevance of Lakatos's theory to these diverse domains highlights connections between them and suggests ways in which philosophy can inform AI domains. This is a good starting point for a more complete interpretation, and we intend to investigate further the implementation of LSR in each of our three main case study domains. In general, we propose a programme of research in which

AI domains are investigated in order to determine: (a) whether there is a useful analogy between them and mathematics, (b) whether we can implement (some of) LSR, (c) how LSR performs: (i) how the methods compare to each other (in mathematics, Lakatos presented them in increasing order of sophistication, but that may not hold in other domains, (ii) whether (and how) LSR enhances the field: how models with LSR compare to models without LSR, according to criteria set by each field.

In order to build the sort of AI which might one day pass the Turing test, whether one views that as strong or weak AI, it will be necessary to combine a plethora of reasoning and learning paradigms, including deduction, induction, abduction, analogical reasoning, non-monotonic reasoning, vague and uncertain reasoning, and so on. This combination of systems and reasoning techniques into something which is "bigger than the sum of its parts" has been identified as a key area of AI research by Bundy (2007) at his Research Excellence Award acceptance speech at IJCAI-07. The philosopher Imre Lakatos produced one such theory of how people with different reasoning styles collaborate to develop mathematical ideas. This theory and suggestions of ways in which people deal with noisy data, revise their beliefs, adapt to falsifications, and exploit vague concept definitions, has much to recommend it to AI researchers. In this chapter we have shown how we might begin to produce a philosophically-inspired AI theory of reasoning.

Acknowledgements

We are grateful to the DReaM group in Edinburgh for discussion of some of these ideas. This work was supported by EPSRC grants EP/F035594/1, EP/F036647 and EP/F037058.

References

Aberdein, A. (2005). The uses of argument in mathematics. *Argumentation*, 19:287–301.

Abramsky, S. (1994). Proofs as processes. *Theoretical Computer Science*, 135:5–9.

Abrial, J.-R. (2009). *Modelling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, UK. To be published.

Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T. S., Metha, F., and Voisin, L. (2009). Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *Journal of Software Tools for Technology Transfer*.

Barton, B. (2009). *The Language of Mathematics: Telling Mathematical Tales*. Mathematics Education Library, Vol. 46. Springer.

Bundy, A. (2007). Cooperating reasoning processes: More than just the sum of their parts. *Research Excellence Award Acceptance Speech at IJCAI-07*.

Cauchy, A. L. (1813). Recherches sur les polyèdres. *Journal de l'École Polytechnique*, 9:68–86.

Cauchy, A. L. (1821). *Cours d'Analyse de l'Ecole Polyechnique*. de Bure, Paris.

Charnley, J., Colton, S., and Miguel, I. (2006). Automatic generation of implied constraints. In *Proceedings of the 17th European Conference on AI*.

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

Colton, S. and Miguel, I. (2001). Constraint generation via automated theory formation. In *Proceedings of the Seventh International Conference on the Principles and Practice of Constraint Programming*, Cyprus.

Colton, S. and Pease, A. (2005). The TM system for repairing non-theorems. In *Selected papers from the IJCAR'04 disproving workshop, Electronic Notes in Theoretical Computer Science*, volume 125(3). Elsevier.

Constable, R. and Moczydłowski, W. (2006). Extracting programs from constructive HOL proofs via IZF set-theoretic semantics. *Lecture Notes in Computer Science*, 4130/2006:162–176.

Corfield, D. (1997). Assaying Lakatos's philosophy of mathematics. *Studies in History and Philosophy of Science*, 28(1):99–121.

Davis, P. and Hersh, R. (1980). *The Mathematical Experience*. Penguin, Harmondsworth.

Donaldson, T. and Cohen, R. (1998). Selecting the next action with constraints. In *Lecture Notes in Computer Science*, volume 1418, pages 220–227. Springer, Berlin / Heidelberg.

Feferman, S. (1978). The logic of mathematical discovery vs. the logical structure of mathematics. In Asquith, P. D. and Hacking, I., editors, *Proceedings of the 1978 Biennial Meeting of the Philosophy of Science Association*, volume 2, pages 309–327. Philosophy of Science Association, East Lansing, Michigan.

Freuder, E. and Wallace, R. (1992). Partial constraint satisfaction. *Artificial Intelligence*, (58):21–70.

Frisch, A., Miguel, I., and Walsh, T. (2002). CGRASS: A system for transforming constraint satisfaction problems. In *Proceedings of the Joint Workshop of the ERCIM Working Group on Constraints and the CologNet area on Constraint and Logic Programming on Constraint Solving and Constraint Logic Programming (LNAI 2627)*, pages 15–30.

Gent, I., Rendl, A., Miguel, I., and Jefferson, C. (2009). Enhancing constraint model instances during tailoring. In *Proceedings of SARA*.

Haggith, M. (1996). *A meta-level argumentation framework for representing and reasoning about disagreement*. PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh.

Hayes-Roth, F. (1983). Using proofs and refutations to learn from experience. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 221–240. Tioga Publishing Company, Palo Alto, CA.

Kitcher, P. (1983). *The Nature of Mathematical Knowledge*. Oxford University Press, Oxford, UK.

Laburthe, F. and the OCRE project team (2000). Choco: implementing a CP kernel. In *Proceedings of the CP'00 Post Conference Workshop on Techniques for Implementing Constraint Programming Systems (TRICS)*, Singapore.

Lakatos, I. (1976). *Proofs and Refutations*. Cambridge University Press, UK.

Lakatos, I. (1978). Cauchy and the continuum: the significance of non-standard analysis for the history and philosophy of mathematics. In Worral, J. and Currie, C., editors, *Mathematics, science and epistemology*, chapter 3, pages 43–60. CUP, UK.

Lakoff, G. and Núñez, R. (2001). *Where Mathematics Comes From: How the Embodied Mind Brings Mathematics into Being*. Basic Books Inc., U.S.A.

Leuschel, M. and Butler, M. (2008). ProB: an Automated Analysis Toolset for the B Method. *Journal Software Tools for Technology Transfer*, 10(2):185–203.

McCasland, R. and Bundy, A. (2006). MATHsAiD: a mathematical theorem discovery tool. In *SYNASC'06*, pages 17–22. EEE Computer Society Press.

McCasland, R., Bundy, A., and Smith, P. (2006). Ascertaining mathematical theorems. In *Electronic Notes in Theoretical Computer Science (ENTCS)*, volume 151, number 1, pages 21–38. Elsevier.

McCune, W. (1994). Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA.

McCune, W. (2001). MACE 2.0 Reference Manual and Guide. Technical Report ANL/MCS-TM-249, Argonne National Laboratory, Argonne, USA.

McNeill, F. and Bundy, A. R. (2007). Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution. *IJSWIS (International Journal on Semantic Web and Information Systems) special issue on Ontology Matching*, 3(3):1–35.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245–286.

Pease, A. (2007). *A Computational Model of Lakatos-style Reasoning*. PhD thesis, School of Informatics, University of Edinburgh. Online at http://hdl.handle.net/1842/2113.

Pease, A., Colton, S., Smaill, A., and Lee, J. (2004). A model of Lakatos's philosophy of mathematics. *Proceedings of Computing and Philosophy (ECAP)*.

Pease, A., Smaill, A., and Colton, S.and Lee, J. (2009). Bridging the gap between argumentation theory and the philosophy of mathematics. *Special Issue: Mathematics and Argumentation, Foundations of Science*, 14(1-2):pp. 111–135.

Pease, A., S. A. C. S. I. A. L. M. R. R. G. G. G. M. (2010 (forthcoming)). Applying lakatos-style reasoning to ai problems. In Vallverdü, J., editor, *Thinking Machines and the philosophy of computer science: Concepts and principles*. IGI Global, PA, USA.

Polya, G. (1945). *How to solve it*. Princeton University Press.

Polya, G. (1954). *Mathematics and plausible reasoning (Vol. 1): Induction and analogy in mathematics*. Princeton University Press, Princeton, USA.

Polya, G. (1962). *Mathematical Discovery*. John Wiley and Sons, New York.

Rissland, E. L. and Skalak, D. B. (1991). Cabaret: Statutory interpretation in a hybrid architecture. *International Journal of Man-Machine Studies*, 34:839–887.

Ruttkay, Z. (1994). Fuzzy constraint satisfaction. In *3rd IEEE Int. Conf. on Fuzzy Systems*, pages 1263–1268.

Skalak, D. B. and Rissland, E. L. (1991). Argument moves in a rule-guided domain. In *Proceedings of the 3rd International Conference on Artificial Intelligence and Law*, pages 1–11, New York, USA. ACM Press.

Snook, C. F. and Butler, M. (2008). UML-B: A plug-in for the Event-B Tool Set. In Börger, E., Butler, M., Bowen, J. P., and Boca, P., editors, *ABZ 2008*, volume 5238 of *LNCS*, page 344. Springer.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego.

Tymoczko, T., editor (1998). *New directions in the philosophy of mathematics*. Princeton University Press, Princeton, New Jersey.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4):27–61.

Winterstein, D. (2004). *Using Diagrammatic Reasoning for Theorem Proving in a Continuous Domain*. PhD thesis, University of Edinburgh.