Ours *is* to reason why

Cliff B. Jones, Leo Freitas, Andrius Velykis

School of Computing Science, Newcastle University
{cliff.jones, leo.freitas, andrius.velykis}@ncl.ac.uk

Abstract. It is now widely understood how to write formal specifications so as to be able to justify designs (and thus implementations) against such specifications. In many formal approaches, a "posit and prove" approach allows a designer to record an engineering design decision from which a collection of "proof obligations" are generated; their discharge justifies the design step. Modern theorem proving tools greatly simplify the discharge of such proof obligations. In typical industrial applications, however, there remain sufficiently many proof obligations that require manual intervention that an engineer finds them a hurdle to the deployment of formal proofs. This problem is exacerbated by the need to repeat proofs when changes are made to specifications or designs. This paper outlines how a key additional resource can be brought to bear on the discharge of proof obligations: the central idea is to "learn" new ways of discharging families of proof obligations by tracking one interactive proof performed by an expert. Since what blocks any fixed set of heuristics from automatically discharging proof obligations is issues around data structures and/or functions, it is expected that what the system can learn from one interactive proof will facilitate the discharge of significant "families" of recalcitrant proof tasks.

1 The challenge

The stimulus for this research has been observing engineers in industry using "formal methods". For example, in the DEPLOY project [24, 19], engineers from six companies attempted to use theorem proving tools. It became clear that the number of proof obligations that were not discharged automatically presented a disincentive for deploying formal specifications and design verification.

In a UK project known as AI4FM [23, 4, 6, 5, 13] we have set ourselves the challenge of improving the effectiveness of formal methods by using Artificial Intelligence (AI) approaches to remove some of the bottlenecks in the construction of formal proofs.[1] More specifically, we see scope for learning proof strategies from experts.

[1] The title of this paper plays on the jingoistic poem *The Charge of the Light Brigade* by Tennyson which includes:

> Theirs not to reason why,
> Theirs but to do and die

The target of the project is the sort of proofs needed in typical justifications of the design and implementation of software systems; it does not aspire to learn how mathematicians prove deep theorems. This section explains the desirability of meeting this challenge and our approach thereto; the body of the paper (Sections 2 and 3) presents an architecture for a system under construction and the rationale for that model. Section 4 discusses the status of this ongoing research.

1.1 Setting out the challenge

Increasingly, organisations are recognising that formal descriptions of systems are a useful intermediate step between informal requirements and detailed design.[2] A crucial advantage of a system description in a tractable formal notation is that it provides a basis for the construction of a correctness argument for design. A stepwise process can therefore provide a chain of argument which shows that an implementation (under assumptions about adherence to the semantics of the implementation language) satisfies the initial specification.

"Posit and prove" development methods such as VDM [9] or Event-B [1] allow engineers to make intuitive design steps from which are generated "proof obligations" (POs) whose discharge justifies the posited design choices. Tailored automatic theorem proving tools such as those available in the "Rodin Tools" [18], or general purpose theorem provers (TPs) such as Isabelle [16, 17], can automatically discharge a high percentage of proof obligations for industrial scale problems; but a small percentage of a large number still leaves an unwelcome interactive theorem proving load for industrial engineers who are neither specifically trained as logicians nor do they obtain the same enjoyment that an academic might from polishing off proofs. In an example from the EU-funded DEPLOY project [24], around 500 POs were generated just to show that a model was consistent; 80% of these were discharged automatically by the Rodin Tools but this left the engineers facing about a hundred interactive proofs. There is little point in arguing whether some other TP tool would discharge a larger proportion of such POs — with a fixed set of heuristics, some POs will always remain undischarged. This issue is delaying –and will continue to limit– wider use of "formal methods".

Further investigation of the hundred remaining POs does, however, offer some more encouraging news: on examination, there were no more than five ideas which made it easy to discharge all of the residual POs. It is this observation –which is echoed in many other examples– that leads us to the approach being followed in the AI4FM project. It is clear that major research progress has been made in discovering general purpose TP heuristics; on the other hand, undecidability results limit hubris and experience suggests that it is properties that are specific to the data structures and functions of an application that make

[2] This paper does not address the transition from requirements to formal specifications: the issues around understanding requirements are addressed in Jackson's "Problem Frame Approach" [8]; ways of determining specifications of control systems from requirements on overall system behaviour are considered in [14].

proofs go through. The resource that our project hopes to tap is the ability of an expert to spot these specific properties. Having once identified them, the system should then absorb them and use them to discharge further similar proofs. This has two major payoffs: not only is the expert's time not wasted performing encores with proofs that are somehow "in the same family"; but there is also a higher probability that proofs will be found automatically after (inevitable) minor changes to specifications or designs.

The hypothesis of the AI4FM project is:

> *Enough information can be automatically extracted from an interactive proof that examples of the same class can be proved automatically*

As discussed below, this information might either be high level strategies or be captured as lemmas.

Before moving on to our approach and a model of the proposed system, there are a few issues worth putting to rest lest they are of concern to readers. First and foremost, we are fully aware of the considerable power of modern TP systems and their tactic languages: as indicated below and in a companion technical report [22],[3] the initial action with any PO is to pass it to at least one TP system. A particularly encouraging experiment is described in Matthias Schmalz's ETH PhD thesis [20] where he shows that a tailored version of Isabelle manages to discharge a higher percentage of the POs than the built-in TP tools of the Rodin Toolset. (The case study is of significant size and is taken from a different DEPLOY partner than the example discussed above.)

It is also worth noting some factors that must qualify reported figures about "automatically" discharging POs. One such factor is alluded to in reporting Schmalz's experiment: he was very careful to split the collection of POs into training and evaluation sets but the fact remains that Isabelle was hand tailored to the training set. A useful view of the AI4FM hypothesis is that we are aiming to automate that tailoring based on monitoring the activities of an expert in discharging a small number of intransigent POs.

Another significant caveat to any claimed figures on "percentages of automatically discharged POs" concerns reformulations of the models. To take one source, in [1] there are some beautifully staged developments that are split into many steps with the effect that the POs are relatively easy to discharge. Even were it the case that the published developments actually represent the author's first attempts, it must be remembered that the author is both an expert and understood thoroughly the strengths/weaknesses of the TPs in the Rodin Tools. An engineer hoping to deploy the same tools is neither likely to be so expert nor wish to reformulate rather larger (than in *any* textbook) models to make the task of the TP system easier. Abrial's book is chosen for comparison because his proofs have been, laudably, discharged using tools. One of the current authors also extols the advantages of stepwise development (see for example, [9, 10, 12]) so the

[3] Although frequent references are made to this technical report, the current paper should be self-contained. The longer report contains details of an example that is large enough that it cannot be covered in a paper of this length.

question of how much reformulation is in order is clearly one of degree. Without being able to provide precise metrics, the position taken in the AI4FM project is that "posit and prove" developments should split a design so that each step reflects a clear design decision. If –as often happens– that leaves undischarged POs, the problem should be tackled by introducing concepts during theorem proving rather than by interposing extra steps of development. (This issue is discussed further –and illustrated– in the companion technical report [22].)

One final comment is in order: the current authors are (painfully) aware that many POs actually represent unprovable conjectures. The role of model-checking approaches such as "ProB" [26] in detecting mistakes is invaluable.

1.2 Tackling the challenge

The objectives of the AI4FM project were recognised by its proposers and reviewers alike as being "ambitious". Of course, the objectives might not just be ambitious — they might be unachievable. At a minimum, the project has to design an unusual way of describing high-level strategies. Here, our experience suggests that the design of such a "language" is more likely to succeed if it is driven from the "state" of the language (rather than its syntax).[4]

A relevant experience for the current project is that which created *mural* [11]. A prime objective of the earlier project was to devise a style of interacting with a TP system that kept the user fully aware of the status of a proof and able to make any sort of forward, backward or intermediate ("cut") step that he or she wished. For its time, this was also considered to be ambitious. In the project that built the *mural* system, we spent a long time iterating versions of its formal description; in fact, project members role-played many versions before any thought was given to actual implementation. The ratio of design time to (initial) implementation was more than four to one.[5] The Newcastle AI4FM team is taking a similar approach. What follows is the n'th iteration of a model of the architecture of a system that we are only now beginning to implement. The following sections (2 and 3) represent an attempt to provide a readable introduction to the model that is summarised in Appendix A — Section 4 includes a discussion of some alternatives.

2 Organising theories

The project will only succeed if a way is found of expressing high-level strategies; moreover, such strategies need to be generalised from instances of lower-level steps. We expect to use a strongly declarative "language" rather than strings of instructions to a TP system. It is argued in Section 3.1 that this will only be possible if a "top-down" hierarchical view of proofs is taken. We anticipate that

[4] Christopher Strachey argued for working out what you want to say before worrying about how to say it.

[5] An additional bonus was that the model was kept up-to-date during the evolution of the system — it is published as [11, Appendix C].

the system will be able to track the overall process by which a user constructs an interactive proof and that "parsing" the detail against the user's "intent" will be possible.

Thus we intend that our proposed system notes the intent of an expert user so as to match this against other tasks. These ideas are described in Section 3. The current section builds up an understanding of the architecture of the AI4FM system in which information about proofs themselves is stored.

The ideas have been tested on a number of examples and the companion technical report [22] contains a non-trivial development (the discussion there and the fact that we made mistakes that were uncovered whilst discharging the POs support the view that, although the example is smaller than the industrial examples that are our final target, it is significantly more challenging than any that would fit in a paper of this length).

2.1 Bodies of knowledge

The accumulated knowledge in AI4FM is stored in a collection of named bodies (in the sense of "body of knowledge").[6]

$$\Sigma :: bdm : BdId \xrightarrow{m} Body$$
$$\ldots$$

These bodies can be related to each other in various ways — this topic is discussed in Section 3.2. There will be bodies of knowledge about general mathematical theories such as set theory (cf. Section 2.2); there will also be bodies that relate to a specific application (cf. Section 2.3). Thus far, this is a conventional structure but it is one into which more novel concepts are embedded.

2.2 Base theories (as *Body* objects)

Consider, say, the *Body* for sequences of "locations"[7] as in the model in [22, Appendix B]. The *BdId* will be some memorable name such as LocSeq. It will "use" both the theory for *Loc* and that for $\mathbb{N}$ (for indexing and the result of **len** s). Within the theory, there will be a series of functions such as $s(i)$, $s_1 \frown s_2$, **hd** s, **tl** s (operators are viewed as functions written in an infix –or even mixfix– notation). A *FnDefn* will contain the signature of the function and, optionally, an explicit definition in terms of more basic operators. So, "append" might be an operator characterised by axioms; whereas *rev* might be defined by a recursive definition. Thus far:

$$Body :: uses : BdId\text{-}\mathbf{set}$$
$$\ldots$$
$$functions : FnId \xrightarrow{m} FnDefn$$
$$\ldots$$

[6] Records, mappings, sets etc. are defined in VDM notation — this should present no real hurdle but readers who want to check details are referred to [10].

[7] Obviously, we intend to handle polymorphism — but this is not covered in the current paper. The approach will almost certainly follow that worked out in [11].

$$
\begin{array}{lll}
\mathit{FnDefn} :: & \mathit{type} & : \mathit{Signature} \\
& \mathit{defn} & : [\mathit{Definition}]
\end{array}
$$

2.3 Specifications give rise to bodies

As well as general theories, we also expect each user specification to be linked to a *Body* corresponding to its "state". Something like the Overture tool [25] will generate a *Body* for each specification (cf. Appendices of [22] that include a top-level specification and two refinement steps). It is often useful to know the problem domain to which a specification relates — for example, in the RAIL domain, frequent use is made of relations to record track layouts.

$$
\begin{array}{lll}
\mathit{Body} :: & \cdots & \\
& \mathit{domain} & : \{\textsc{Rail}, \textsc{Auto}, \ldots\} \\
& \cdots &
\end{array}
$$

In most industrial cases, the state will be defined as a record. In examples such as those from the industrial partners in the DEPLOY project, states of 20 fields were not unusual — and these states also tended to have lengthy invariants. It might be worth generating theories for any separable sub-states in the sense that data type invariants and/or operations force some fields to be grouped together — other than these constraints, models should be split as far as is possible — each distinct record type will be translated into a body.

Within a body for a specification, a proof obligation generator (POG) will place a *Conjecture* for each PO about the consistency (e.g. invariant preservation) of that single specification. Proof obligations will also be generated corresponding to the claim that one model reifies another (obviously this has to be triggered by the claimed reification link).

2.4 Conjectures

The information in a *Body* that is of use in proofs is the collection of formal results that are built up over the lifetime of that body.

$$
\begin{array}{lll}
\mathit{Body} :: & \cdots & \\
& \mathit{theory} & : \mathit{ConjId} \xrightarrow{m} \mathit{Conjecture} \\
& \cdots &
\end{array}
$$

When first generated, a *Conjecture* is actually a proof task. Each such conjecture has hypotheses and a goal both containing judgements. A *Judgement* can be a sequent or an (in-)equation. In addition there can be any number of (attempts at) justifications. Thus:

$$
\begin{array}{lll}
\mathit{Conjecture} :: & \cdots & \\
& \mathit{hyps} & : \mathit{Judgement}^* \\
& \mathit{goal} & : \mathit{Judgement} \\
& \mathit{justifs} & : \mathit{JusId} \xrightarrow{m} \mathit{Justification} \\
& \cdots &
\end{array}
$$

$$\mathit{Judgement} = \mathit{Sequent} \mid \mathit{Equation} \mid \mathit{Ordering} \mid \cdots$$

So, for example, $s \in \mathit{LocSeq} \vdash \mathit{rev}(\mathit{rev}(s)) = s$ is likely to be a judgement accompanied by a proof; other judgements will be axioms.

2.5 Justifications

Turning to *Justification*, notice that it is explicitly envisaged that there can be multiple attempts to justify a proof task. When a conjecture is first generated, it will have no justifications. A user might start one proof *Attempt*, leave it aside and try another, then come back and complete the first proof.

Many conjectures will not contain proofs as such. There might for example be an axiom that $\mathbf{hd}\,([a] \frown s) = a$ and a proof of $\mathit{rev}(\mathit{rev}(s)) = s$. Unsurprisingly, a flag AXIOM will be used to mark axioms. Another way in which a justification need not be (a graph of) a logical proof is that it might be copied from some separate TRUSTED source.

In practice, TP tools such as Isabelle and Z/EVES are powerful enough that a user will hardly ever interact at the level of the (natural deduction) laws of the logic itself. So, in fact, the most prevalent examples of *Justification* ought come from the underlying theorem prover. Automatic use of a TP system will be recorded as an instance of *Tool* (and might include the configuration used). Other obvious examples of *Tool* might record the use of a SAT/SMT tool (which could also be used to look for counter examples if the first attempt at proof fails).

$$\mathit{Justification} = \text{AXIOM} \mid \text{TRUSTED} \mid \mathit{Tool} \mid \mathit{Attempt}$$

The idea of *Attempt* is to be able to accommodate (manual) proof steps.

$$\begin{array}{lll} \mathit{Attempt} :: & \mathit{rule} & : \mathit{ConjId} \\ & \mathit{hyps} & : \mathit{ConjId}^* \\ & \mathit{subst} & : \mathit{Term} \xrightarrow{m} \mathit{Term} \\ & \mathit{sub\text{-}probs} & : \mathit{ConjId}\text{-}\mathbf{set} \end{array}$$

Notice an attempt corresponds to one step in a proof: collecting a whole proof requires tracing the attempts at the sub-conjectures. Thus the notion of whether a proof is complete (in the sense of (transitively) relying only on axioms) is a complex recursive predicate. A low-level instance of *Attempt* might record that the (*rule*) on which it is based is "or elimination". More interesting would be the use lemmas.

3 Strategies

As indicated, the aim of the AI4FM project is to support users with the discharge of industrial scale POs. The way in which we expect to extract strategic insight from proofs –possibly undertaken by experts– is described in Section 3.1; selection and replay (with modifications) is covered in Section 3.2. First, the data structures are described.

Strategies reside in the relevant *Body*:

$$\begin{array}{lll} \mathit{Body} :: & \cdots & \\ & \mathit{strats} & : \mathit{StrId} \xrightarrow{m} \mathit{Strategy} \end{array}$$

A low level strategy might split a problem into sub-cases; another could reduce an expression to a normal form; an important collection of strategies will be for induction; an interesting strategy might shift the representation of an object of interest to a different body of knowledge.

We have for a long time within the AI4FM project referred to the "why" of strategies and conjectures (and this is the reason for the use of this word in the title of the paper). The point is that it is easier to achieve a high level of strategy re-use if the intent is captured rather than if only a transcript is recorded. The initial conjectures come from POG and their *source* will contain the name of the kind of proof obligation. Conjectures that are generated as sub-problems by a strategy will be marked with the *Why* value of the strategy.

$$\begin{array}{lll} \mathit{Conjecture} & :: & \mathit{source} : \mathit{Origin} \mid \mathit{Why} \\ & & \ldots \end{array}$$

Examples of values for *Why* are given in Section 3.1 (and more are listed in [22]). The set will never be closed so that a user can always add a new concept.

Strategic information needs to represent both "and" and "or" situations. The "or" function is represented by having alternative strategies. For example, we do not explicitly say that three strategies whose *StrId*s are StructuralIndn, ℕPeanoIndn and ℕCompleteIndn are options — it's just that their *intent* fields are all likely to be marked as something like HandleUniversal. The selection between alternatives is, in a sense, underneath the covers for the user (it might give rise to limited parallelism).

An "and" split in a *Strategy* records that, in order to justify a conjecture, certain other conjectures must be discharged (although in some cases it will just be a reformulation and generate only one sub-task — e.g. contrapositives of implications, use of an isomorphic model — at the leaves of a strategy there are no sub-tasks). Just as in LCF-like systems, the flip side of *split* is the *justif* which proves that the sub-goals (when discharged) justify the original goal.

$$\begin{array}{lll} \mathit{Strategy} & :: & \mathit{intent} : [\mathit{Why}] \\ & & \mathit{split} : \mathit{Conjecture} \to \mathit{Conjecture}\text{-}\mathbf{set} \\ & & \mathit{justif} : \mathit{JusId} \\ & & \ldots \end{array}$$

Notice that *split* is a general function, not a mapping. One possibility for the *split* field is that it could contain a text in a language that is interpreted. In contrast, it fits our top-down view better to have high-level derived rules — see the discussion of lemmas in the next sub-section.

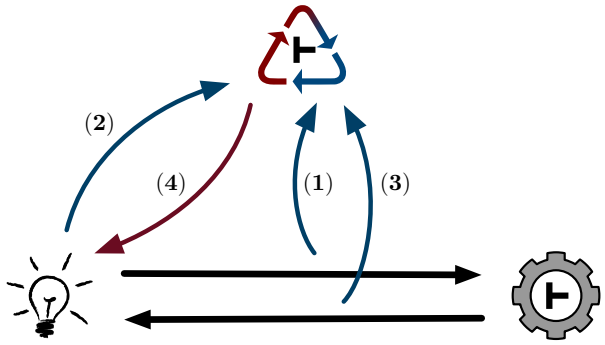
After a split is made, the theorem prover of choice will be triggered on each of the generated sub-goals. If they are all discharged, this reinforces the strategy used; if the user hits a dead-end, the likelihood of trying that strategy in similar circumstances is reduced. Furthermore, the expert has to backtrack to some other decision in the proof process.

3.1 Extracting strategies

As indicated in Section 1.1, over and above general purpose heuristics, the key resource that AI4FM hopes to exploit is to garner information from interactive

proofs. It is convenient to talk about this in terms of the interactive proof being undertaken by an expert but it might also be the case that an engineer who is working on PO discharge behaves as the expert — perhaps after some reflection.

The following diagram shows how AI4FM is intended to snoop on the interaction of the expert with the theorem proving system. The symbols stand for the expert (depicted by a lightbulb for inspiration), the TP system (a cogwheel around the turnstile symbol) and at the top of the diagram AI4FM (marked by our trademark for recycling deductions). The basic two way interaction between the expert and the TP system is marked by the horizontal arrows.



The numbered arcs showing interactions with AI4FM are explained as follows:

1. Having a record of why a conjecture is being tackled, the system can attempt to "parse" any interactions initiated by the expert against existing strategies.
2. The expert will be asked to name any new strategies and be invited to mark identifying features.
3. The system can note undischarged goals, record success/failure of strategies; and record the lemmas that are used.
4. The system can suggest strategies to the expert.

This illustrates the primary way in which AI4FM will accrete information. Notice that the aim is to mine the proof *process* which we feel has far more information than just finished –and possibly polished– proofs.

One point that we feel is crucial is the importance of starting the analysis of what the user (expert) is doing from the initial goal (the "top" of the proof): knowing why a conjecture arose is the key to getting an appropriate "parse" of the steps made; looking at the steps alone is a much harder way of determining an expert's intent.[8]

[8] Hearing a seminar on programs that "understand" music prompted the analogy of trying to guess the form of a piece of music a bar at a time versus trying to "parse" it against some expected structure(s).

A failure to discharge a PO will be apparent when one or more conjectures cannot be proved either by the underlying theorem proving systems or any of the available strategies. When such an impasse is reached, an expert might introduce a lemma. Alternatively, the expert might respond by making a different choice in some step of the proof. For now, we assume that this step is captured without immediate generalisation (cf. Section 3.2). The expert is prompted to provide a name (*Why*) for the new idea. In some cases, it will be possible for AI4FM to track that a new strategy specialises an existing one but in the worst case it is certainly worth having the option to add this relationship by hand.

It is also useful to organise a "taxonomy" of strategies. The idea is perhaps best illustrated by an example:

ℕPeanoIndn specialises ℕIndn
ℕCompleteIndn specialises ℕIndn
ℕIndn specialises Indn
StructuralIndn specialises Indn

So the final field of *Strategy* is:

Strategy :: ⋯
specialises : [*StrId*]

There are also what might be thought of as "meta-strategies". One of these is referred to by the second author as "zooming". Given, for example (in technical report [22, B.1]), a proof obligation that involves expressions such as $pre\text{-}NEW0(s, \sigma)$ there are three levels at which the PO can be passed to a theorem prover: as is (with nothing expanded); expansion of the specific predicate $pre\text{-}NEW0$ (about which there are likely to be no lemmas) to terms/operators of set theory; or even an expansion of everything down to predicate calculus. In general, proofs are clearest to a user if they can be conducted with least expansion. In fact, a genuine expert will often "anti-zoom" and prove results at a more general level than their original expression.

A frequent contribution from an expert is to spot that a lemma will provide the clue that makes automatic theorem proving succeed. This approach is seen most clearly in the "waterfall" of ACL2 [15] but it also applies to LCF-style theorem provers such as Isabelle. Lemmas essentially bundle up steps in an argument so that one application of a strategy moves a proof many steps towards its goal. In this section, we assume that lemmas are captured in exactly the form in which they are used; Section 3.2 indicates one source of generalisation. We are also investigating other ways of spotting generalisations at the point of lemma capture. A way of relating bodies of knowledge is described in the next section and this is one technique by which patterns between lemmas can be utilised.

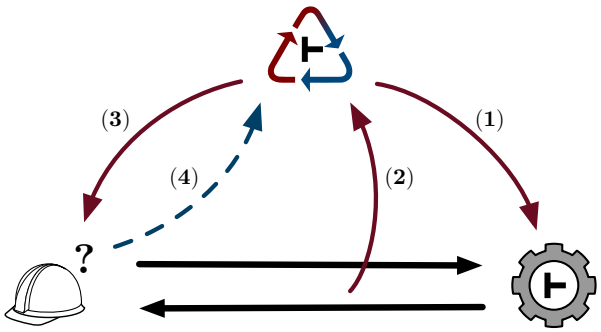
3.2 Replaying strategies

For some specific *Conjecture*, a user might wish to provide a justification. There is, in fact, rather more behind this comment than someone schooled in say Isabelle might expect. First of all, AI4FM will provide many ways of viewing the outstanding proof tasks so that, for example, a user can see all of the unjustified

leaves of the tree from some specific PO. It is also envisaged that a user can opt to provide new proofs for already justified results — this is why *Conjecture* contains a map (*justifs*) to any number of justifications.

The description that follows, however, assumes a single point of focus: a *Conjecture*. If the theorem prover of choice can discharge the conjecture automatically, the justification is recorded as a *Tool* transcript. The more interesting case for AI4FM is where automatic proof fails.

The following diagram shows how AI4FM can assist the interaction of an engineer (marked by a hardhat) with a TP system.



Here again, the extra indexed arcs are explained:

1. The system can replay (possibly modified versions of) strategies that fit the context and have been previously generated in expert mode. As explained below, an attempt is made to order the use of options based on previous success/failure.
2. Success/failure of strategies is noted both to trigger a move to the next option and to adjust weights that will affect future choices. If necessary, failure of the final option will cause the system to backtrack to an earlier point in the proof tree.
3. The system must keep the user informed (especially about backtracks); it might also ask about lemmas.
4. The engineer might be able to assist if automatic attempts (just) fail; alternatively, there might be a need to bring an expert on line.

Given a collection of strategies, we need a way of selecting the one that is most likely to succeed. There are two sources of information. The provenance information in the *source* field of *Conjecture* is discussed above; in addition to the fields listed at the beginning of Section 2.4, a *Conjecture* will contain information about its features:

$$\begin{aligned} \mathit{Conjecture} &:: \cdots \\ &\quad \mathit{match} : \mathit{Features} \end{aligned}$$

A putative shape of *Features* is given in Appendix A — the final list will be chosen after experimentation. The order in which strategies are tried is governed by its *Score* and this is an area where we hope to use some form of "machine learning". It is therefore crucial that the success or failure of strategies is recorded to adjust the weights in the *rank* function

$$\begin{aligned} \mathit{Strategy} :: {} & \cdots \\ & \mathit{rank} : \mathit{Conjecture} \to \mathit{Score} \\ & \cdots \end{aligned}$$

Another way of ordering strategies is based on the assumption that the most specific strategy should be tried first and a *specialises* field is added to *Strategy* to locate the next more general strategy:

$$\begin{aligned} \mathit{Strategy} :: {} & \cdots \\ & \mathit{specialises} : [\mathit{StrId}] \end{aligned}$$

In this case, failure prompts trying the next more general strategy.

A stored strategy might well rely on a lemma and the exact form of the lemma used in the situation from which the strategy was extracted might not fit the context where the strategy is being replayed. So far, we have considered two ways of evolving ("educing") lemmas: one involves looking in related bodies of knowledge; the other attempts to infer a modified lemma from contextual information.

With regard to relationships between bodies of knowledge, Σ contains:

$$\begin{aligned} \Sigma :: {} & \cdots \\ & \mathit{bdrels} : (\mathit{BdId} \times \mathit{Relationship} \times \mathit{BdId})\text{-}\mathbf{set} \end{aligned}$$

which stores relationships between bodies of knowledge. Like *Why* itself, this will have to be expandable by the user. Some examples include:

$$\begin{aligned} \mathit{Relationship} = {} & \mathit{Specialisation} \mid \mathit{Morphism} \mid \mathit{Isomorphism} \mid \\ & \mathit{Inherits} \mid \mathit{Sub} \mid \mathit{Similarity} \mid \mathit{Difference} \mid \cdots \end{aligned}$$

AI4FM might, for example, have some abstract items in *Body* such as Larch's "collector" [7]; sets, sequences and maps would all then be specialisations of collector. Another abstract item might be "inductable" where the more general knowledge about setting up inductive proofs would reside. *Morphism* and *Isomorphism* will be used for precise mathematical relationships — the latter where results can be used in either direction. *Similarity* will be for less precise connections (fuzzy matches).

To begin with a low-level example of how the relationships between bodies can be used, suppose a strategy for rearranging operators was applied on set operators and the associativity of union was used, then application of the same strategy on sequence operators might generate the need for a lemma for the associativity of concatenation. In this case, one would expect that such a lemma would already be in the appropriate *Body*. A more interesting example might be ways of creating witnesses for existential quantifiers.

The approach of evolving a lemma from the originating proof to match a new context looks to be feasible. Comparing the hypothesis and goals of the original proof task with the lemma that was generated in expert mode ought to provide

enough information to tailor a new lemma that fits the context in which the containing strategy is replayed.

If all options for a strategy have proved fruitless, AI4FM will be able to locate a higher point in the "proof tree" and try alternatives from there. The balance of exploring breadth versus resorting to backtracking will have to await experiments.

4 Status and way forward

As indicated in Section 1.2, we have already spent a lot of time discussing the architecture of our proposed system in terms of a model whose current version is in Appendix A. Some of the issues already resolved are outlined in Section 4.2; Section 4.3 sketches our immediate priorities; the first sub-section outlines our experiments with case studies.

4.1 Outline of case studies

As has been made clear above, the model of the architecture described in the preceding sections and summarised in Appendix A has evolved over experiments with case studies. Experience has shown that relatively little can be learnt from small examples and that many issues only become clear when non-trivial case studies are considered. The specification and development in [22] is not as large as those met –for example– in the DEPLOY project (see [19]) and it has been important that such industrial applications are kept in mind. Apart from the experience of working closely with industrial teams, the authors have direct writing experience of specifications and developments for applications like cash cards, file stores and systems that control access to secure sites.

The management of a free storage "heap" is a well understood computing problem and the development in [12, §7] provided a good starting point for a useful case study.[9] The heap example is too large to cover in detail in a paper of this length which is why a companion technical report [22] is being made available along with all of the formal material in machine readable form.[10] Deviations from the original development of [12, §7] are discussed in [22, Appendix B.3.6]. That report also contains additional observations that come from other case studies.

One thing that has come as a surprise is the degree of difficulty in handling partial terms efficiently in Isabelle. Our surprise might puzzle a reader who knows that the first author has long argued for the use of a "Logic of Partial Functions" [2]; this is clearly an area for more investigation and Schmalz's [20] might offer the approach that fits most closely with Isabelle.

[9] In fact, most of the chapters in [12] are non-trivial and usable as case study material.

[10] We have actually undertaken the proofs in both Z/EVES and Isabelle/HOL and the differences are discussed in [22].

4.2 Alternatives already considered

There are some ideas that we have considered but have yet to build into the model. Two such issues are mentioned here: "analysing proof failures" and "recording negative results". Our project colleagues in Edinburgh have pioneered general ideas about analysing proof failures and more specifically about "rippling" [3]. Superficially, it would be easy to include one or more indicators such as StuckInduction among the values of *Why* but further investigation and experimentation is required to check that this provides a convenient bridge to established –and new– ways of analysing failure.

Prompted by an interesting discussion with Aaron Sloman, we experimented with the idea of recording in the model what might be termed "negative results". The point being that knowing –for example– that, while set union and list concatenation both enjoy properties such as associativity, the latter is not commutative (in general) might provide important clues when trying to replay a strategy from one context in a different body. This idea remains under consideration but for now it is assumed that such negative information is stored as a *Difference* relationship in *bdrels*.

Probably the biggest issue in our discussion on the architecture has been the ways in which one can view the design of a "strategy language". To oversimplify, one can contrast "bottom up" approaches that try to extend the vocabulary of existing tactic languages with the "top down" approach followed in this paper. Of course, the ideal is that these approaches converge and it is clear that the *split* field of *Strategy* in Appendix A could contain texts of an extended tactic language. As indicated in Section 3.1, a strong argument for the top-down approach is that making sense of ("parsing") interactions in expert mode is only possible if the system can track the user's overall objective. It must however be conceded that it would be simpler to capture the tactic-level steps that an expert makes than it will be to parse these against high-level goals. The idea of taking the lower-level approach and adding annotations to such scripts is viewed as a fall-back position; our immediate plan is to tackle the high-level objective.

The model in Appendix A does not order sub-goals in the *split* field of *Strategy*. This assumes that only graph shape matters but we accept that there are cases where order might be important. In fact, we have considered the idea of time stamping each *Conjecture*. We have yet to build this into the model (one can always write a function that drops this information where not needed).

Another option in the model is to be able to locate instances of the use of strategies — but, for the time being at least, the pointers are in the other direction.

As was found in the *mural* project, records (in the VDM sense) can be difficult in that there is really a different theory of selectors and constructors for each record shape. Records are so ubiquitous that we have to do something for them and we do not favour expanding out "axioms" for all of the constructors/selectors. One reason for preferring some built-in handling of records is that theorem provers can actually suffer from an excess of lemmas: the irrelevant clutter makes searching inefficient or even useless.

4.3 Next steps

Our immediate activity is to tension the model in Appendix A against more non-trivial examples. A trade-off then has to be made as to the point in time when greater productivity can be achieved by building the model into our on-going tool support activities. The balance here is that it takes almost no time to revise the model on paper but somewhat longer to revise the Eclipse-based tools that are the current work of (mainly) the third author. The first version of an Eclipse interface to Isabelle has been released [21] and it provides an integration platform for our tools and experiments which gather information from interactive proofs.

Even once we switch to slightly slower revision iterations involving the tools, it is our intention to follow the good practice in the *mural* project and to keep the formal model up to date.

Acknowledgements

It is a pleasure to acknowledge the fruitful collaboration with our Scottish colleagues in the AI4FM project. The first author is grateful to Aaron Sloman for a useful discussion at the Birmingham BCTCS meeting. We also derived stimulus from discussions at the Schloß Dagstuhl event (12271) on "AI Meets Formal Software Development". Last, but by no means least, the authors are grateful for the EPSRC funding of the AI4FM project.

References

1. J.-R. Abrial. *The Event-B Book*. Cambridge University Press, Cambridge, UK, 2010.
2. H. Barringer, J.H. Cheng, and C. B. Jones. A logic covering undefinedness in program proofs. *Acta Informatica*, 21:251–269, 1984.
3. A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*, volume 56 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2005.
4. Alan Bundy, Gudmund Grov, and Cliff B. Jones. Learning from experts to aid the automation of proof search. In Liam O'Reilly and Markus Roggenbach, editors, *AVoCS'09 – PreProceedings of the Ninth International Workshop on Automated Verification of Critical Systems*, CSR-2-2009, pages 229–232. Swansea University, 2009.
5. Alan Bundy, Gudmund Grov, and Cliff B. Jones. An outline of a proposed system that learns from experts how to discharge proof obligations automatically. In Jean-Raymond Abrial, Michael Butler, Rajeev Joshi, Elena Troubitsyna, and Jim C. P. Woodcock, editors, *Dagstuhl 09381: Refinement Based Methods for the Construction of Dependable Systems*, pages 38–42, 2009.
6. Leo Freitas and Cliff B. Jones. Learning from an expert's proof: AI4FM. In Tom Ball, Lenore Zuck, and Natarajan Shankar, editors, *UV10 (Usable Verification)*, November 2010.
7. John V Guttag, James J Horning, Withs J Garl, Kevin D Jones, Andres Modet, and Jeannette M Wing. *Larch: languages and tools for formal specification*. Springer-Verlag, 1993.

8. Michael Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley, 2000.
9. C. B. Jones. *Software Development: A Rigorous Approach*. Prentice Hall International, 1980.
10. C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990.
11. C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. *mural: A Formal Development Support System*. Springer-Verlag, 1991.
12. C. B. Jones and R. C. F. Shaw, editors. *Case Studies in Systematic Software Development*. Prentice Hall International, 1990.
13. Cliff B. Jones, Gudmund Grov, and Alan Bundy. Ideas for a high-level proof strategy language. In Bruno Dutertre and Hassen Saidi, editors, *AFM'10 (Automated Formal Methods)*, July 2010.
14. Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson. Deriving specifications for systems that are connected to the physical world. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems*, volume 4700 of *LNCS*, pages 364–390. Springer Verlag, 2007.
15. Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *ACL2 Computer-Aided Reasoning: An Approach*. University of Austin Texas, 2009.
16. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A proof assistant for higher-order logic*, volume 2283 of *LNCS*. Springer, 2002.
17. Lawrence C. Paulson. Isabelle: A Generic Theorem Prover. *Lecture Notes in Computer Science*, 828, 1994.
18. Rodin. The Rodin Tools can be downloaded from SourceForge, 2008. `http://www.sourceforge.net/projects/rodin-b-sharp`.
19. Alexander Romanovsky and Martyn Thomas, editors. *Industrial Deployment of System Engineering Methods*. Springer, 2013.
20. Matthias Schmalz. *Formalising the Logic of Event-B*. PhD thesis, ETH, Zuerich, 2012.
21. Andrius Velykis. Isabelle/Eclipse, 2013. `http://andriusvelykis.github.io/isabelle-eclipse`.
22. Andrius Velykis, Leo Freitas, Cliff B. Jones, and Iain Whiteside. How to say why (in AI4FM). Technical Report CS-TR-1376, Newcastle University, March 2013.
23. WWW. AI4FM, as at February 2013. `http://www.ai4fm.org`.
24. WWW. Deploy project web site, as at February 2013. `http://www.ncl.ac.uk/computing/research/project/3625`.
25. WWW. Overture, as at March 2013. `http://www.overturetool.org`.
26. WWW. The ProB animator and model checker, as at February 2013. `http://www.stups.uni-duesseldorf.de/ProB`.

A Summary of "model"

$$
\begin{array}{lll}
\Sigma :: & bdm & : BdId \xrightarrow{m} Body \\
& bdrels & : (BdId \times Relationship \times BdId)\text{-}\mathbf{set}
\end{array}
$$

$$
\begin{array}{lll}
Body :: & uses & : BdId\text{-}\mathbf{set} \\
& domain & : \{\textsc{Rail}, \textsc{Auto}, \ldots\} \\
& functions & : FnId \xrightarrow{m} FnDefn \\
& theory & : ConjId \xrightarrow{m} Conjecture \\
& strats & : StrId \xrightarrow{m} Strategy
\end{array}
$$

$$
\begin{array}{lll}
FnDefn :: & type & : Signature \\
& defn & : [Definition]
\end{array}
$$

$$
\begin{array}{lll}
Conjecture :: & source & : Origin \mid Why \\
& hyps & : Judgement^* \\
& goal & : Judgement \\
& justifs & : JusId \xrightarrow{m} Justification \\
& match & : Features
\end{array}
$$

$$Judgement = Sequent \mid Equation \mid Ordering \mid \cdots$$

$$Justification = \textsc{Axiom} \mid \textsc{Trusted} \mid Attempt \mid Tool$$

$$
\begin{array}{lll}
Attempt :: & rule & : ConjId \\
& hyps & : ConjId^* \\
& subst & : Term \xrightarrow{m} Term \\
& sub\text{-}probs & : ConjId\text{-}\mathbf{set}
\end{array}
$$

$$Tool = \cdots$$

Could include info about *blocks*: *ConjId*-**set**

$$
\begin{array}{lll}
Strategy :: & intent & : [Why] \\
& split & : Conjecture \to Conjecture\text{-}\mathbf{set} \\
& justif & : ConjId \\
& rank & : Conjecture \to Score \\
& specialises & : [StrId]
\end{array}
$$

$$
\begin{array}{lll}
Features :: & mainTps & : BdId\text{-}\mathbf{set} \\
& mainFns & : FnId\text{-}\mathbf{set} \\
& other & : \cdots
\end{array}
$$

$$Origin = Token$$

$$Why = Token$$

$$
\begin{aligned}
Relationship = {} & Specialisation \mid Morphism \mid Isomorphism \mid \\
& Inherits \mid Sub \mid Similarity \mid Difference \mid \cdots
\end{aligned}
$$