A Model for Capturing and Replaying Proof Strategies

Leo Freitas, Cliff B. Jones, Andrius Velykis and Iain Whiteside

School of Computing Science, Newcastle University, NE1 7RU, UK
`first.last@newcastle.ac.uk`

Abstract. Modern theorem provers can discharge a significant proportion of Proof Obligations (POs) that arise in the use of Formal Methods (FMs). Unfortunately, the residual POs require tedious manual guidance. On the positive side, these "difficult" POs tend to fall into families each of which requires only a few key ideas to unlock. This paper outlines a system that will identify and characterise ways of discharging POs of a family by tracking an interactive proof of one member of the family. This opens the possibility of capturing ideas from an expert and/or maximising reuse of ideas after changes to definitions. The proposed system has to store a wealth of meta-information about conjectures, which can be matched against previously learned strategies, or can be used to construct new strategies based on expert guidance. This paper describes this meta-information and how it is used to lessen the burden of FM proofs.

1 Introduction

Formal methods based on one or another chosen specification language are now used to document different levels of abstraction for many systems. In those methods that adopt a "posit and prove" style of development, engineering decisions are recorded in concrete models and Proof Obligations (POs) are generated whose discharge establishes that the reified model has a behaviour compatible with a more abstract model. (There are also POs that establish internal consistency of one level of model — e.g. respecting invariants.)

Both clever engineering and AI techniques have led to Automated Theorem Provers (ATPs) that can discharge an impressively large proportion of POs but the manual discharge of the remaining POs is an impediment to wider use of formal methods. The research hypothesis of the AI4FM project is that a system can be built that learns, from interactive proof, ideas that will facilitate automatic discharge of other recalcitrant POs *in the same family.* The emphasised qualification in the previous sentence indicates that the system is not intended to discover general heuristics; the aim is to extract intuition about functions and data structures used in the specific family of POs.

The design of the AI4FM system is itself being conducted formally. Moreover, exploration of the design space is being undertaken by recording and modifying formal models of the proposed system. A similar process was used to considerable effect in the creation of the *mural* theorem proving assistant [JJLM91].

To underpin the explanation of the AI4FM system, Section 2 introduces a formal specification of a heap memory manager; POs and proof strategies arising from this example are used to illustrate the AI4FM system features. Sections 3–4 minimally present the specification of the core of the proposed system, particularly the representation of proof strategies and the important high-level information about POs needed to reuse the strategies. A summary of the specification is given in Appendix A, whereas a complete description can be found in [FJVW13, Chapter 3] and in [JFV13]. The model of how proof strategies are (re)played and captured is described in Sections 4 and 5. Proof strategies illustrating the model have been informally—but more in-depth—presented in [FW14]. Section 6 reviews some related work and Section 7 summarises the conclusions.

2 Background

This section introduces the relevant functions, features and POs of a heap memory model that is used below to illustrate AI4FM. The example is adapted from [JS90, Ch.7], where it was specified using the VDM notation [Jon90]. The *Heap* specification has been formally mechanised (i.e. all the POs have been proved) using both Isabelle [Pau94] and Z/EVES [Saa97] theorem provers. The mechanisation of the proofs is recorded in [FJVW13]. This work has also generated an account of practical proof patterns [FW14], which descriptions underpin the proof-process modelling presented in this paper.

The heap memory manager is modelled using two datatypes and two operations at two levels of abstraction (here specified using VDM notation):

Loc: the type of a single memory location, represented as $\mathbb{N}$.

Free: the type of the heap as a collection of all free locations. At *level 0*, it is represented as a set of locations $\mathit{Free}0 \triangleq \mathit{Loc}\text{-}\mathbf{set}$. At *level 1*, it is represented as a map from start locations to sizes that is constrained by a datatype invariant to be *disj*oint and *sep*arate:

$$\mathit{Free}1 = \mathit{Loc} \xrightarrow{m} \mathbb{N}_1$$

$$\mathbf{inv}\ (f) \triangleq \mathit{disj}(f) \wedge \mathit{sep}(f)$$

$$\mathit{disj}(f) \quad \triangleq \quad \forall l, l' \in \mathbf{dom}\, f \cdot l \neq l' \ \Rightarrow\ \mathit{locs\text{-}of}(l, f(l)) \cap \mathit{locs\text{-}of}(l', f(l')) = \{\,\}$$

$$\mathit{sep}(f) \quad \triangleq \quad \forall l \in \mathbf{dom}\, f \cdot (l + f(l)) \notin \mathbf{dom}\, f$$

The invariant conditions ensure that the range of locations identified by any two map elements (defined as $\{l \ldots l + f(l) - 1\}$ by *locs-of*) do not intersect (*disj*) and that contiguous memory regions are as large as possible (*sep*).

NEW: takes a size as input and, if available, returns a starting location for a contiguous chunk of memory of the appropriate size after updating the state.

DISPOSE: returns a contiguous chunk of memory to the heap. This operation takes a start location and size as parameters and updates the state.

At level 0, these operations are defined using set operations, whereas at level 1 they are refined to use mapping from locations to their corresponding sizes. The main PO discussed in this paper is feasibility of the *NEW* operation and the associated lemmas needed for the proof At level 1, the *NEW* operation is:

$$
\begin{array}{l}
\mathit{NEW1}\ (s\colon \mathbb{N}_1)\ r\colon \mathit{Loc} \\
\textbf{ext wr}\ f_1\ :\ \mathit{Free1} \\
\textbf{pre}\ \exists l \in \textbf{dom}\, f_1 \cdot f_1(l) \geq s \\
\textbf{post}\ r \in \textbf{dom}\, \overleftarrow{f_1} \wedge (\overleftarrow{f_1}(r) = s \wedge f_1 = \{r\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \vee \\
\qquad\qquad\qquad\qquad (\overleftarrow{f_1}(r) > s \wedge f_1 = (\{r\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \cup_m \{r + s \mapsto \overleftarrow{f_1}(r) - s\})
\end{array}
$$

*NEW*1 has two behaviours depending on whether a location of exactly the required or strictly larger size has been located. If the size matches, then that element is removed from the map; if the map element refers to a larger region, then the remaining locations in the region must be added back to the heap (hence the map union). The precondition embraces both cases by using $\geq$.

The feasibility PO for *NEW* requires that, for every starting state ($\overleftarrow{f_1}$) and input (s) that satisfy the precondition and invariant, there exists an updated state (f_1) and result (r) that satisfy the postcondition and invariant.

$$
\begin{array}{l}
\forall s \in \mathbb{N}_1, \overleftarrow{f_1} \in \mathit{Free1} \cdot \exists l \in \textbf{dom}\, \overleftarrow{f_1} \cdot \overleftarrow{f_1}(l) \geq s \wedge \mathit{sep}(\overleftarrow{f_1}) \wedge \mathit{disj}(\overleftarrow{f_1}) \ \Rightarrow \\
\exists f_1 \in \mathit{Free1}, r \in \mathit{Loc} \cdot [r \in \textbf{dom}\, \overleftarrow{f_1} \wedge (\overleftarrow{f_1}(r) = s \wedge f_1 = \{r\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \vee \\
(\overleftarrow{f_1}(r) > s \wedge f_1 = (\{r\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \cup_m \{r + s \mapsto \overleftarrow{f_1}(r) - s\})] \wedge \mathit{sep}(f_1) \wedge \mathit{disj}(f_1)
\end{array}
$$

The first step proof is to expose the postcondition by application of introduction rules, then supply an appropriate witness for the result (l, from the precondition), resulting in a conjecture of the form:

$$
\frac{\overleftarrow{f_1} \in \mathit{Free1},\ s \in \mathbb{N}_1,\ l \in \textbf{dom}\, \overleftarrow{f_1},\ l \geq s,\ \mathit{sep}(\overleftarrow{f_1}),\ \mathit{disj}(\overleftarrow{f_1})}{\begin{array}{l}\exists f_1 \in \mathit{Free1} \cdot [l \in \textbf{dom}\, \overleftarrow{f_1} \wedge (\overleftarrow{f_1}(l) = s \wedge f_1 = \{l\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \vee \\ (\overleftarrow{f_1}(l) > s \wedge f_1 = (\{l\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \cup_m \{l + s \mapsto \overleftarrow{f_1}(l) - s\})] \wedge \mathit{sep}(f_1) \wedge \mathit{disj}(f_1)\end{array}}
$$

To progress this proof, a witness for the updated state is required; however, a *hidden* case analysis on the $\geq$ in the hypothesis is required because of the alternative behaviours of *NEW*. The two cases, $l > s$ and $l = s$, can then be solved independently. In both cases, the witness for f_1 is explicit in the postcondition. The invariant is challenging. For the $l > s$ case, the goal is:

$$
\frac{\overleftarrow{f_1} \in \mathit{Free1},\ s \in \mathbb{N}_1,\ l \in \textbf{dom}\, \overleftarrow{f_1},\ l \geq s,\ \mathit{sep}(\overleftarrow{f_1}),\ \mathit{disj}(\overleftarrow{f_1}}{\begin{array}{l}\mathit{sep}((\{l\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \cup_m \{l + s \mapsto \overleftarrow{f_1}(l) - s\}) \wedge \\ \mathit{disj}((\{l\} \mathbin{\lhd\!\!\!-} \overleftarrow{f_1}) \cup_m \{l + s \mapsto \overleftarrow{f_1}(l) - s\})\end{array}}
$$

which can be solved by lemmas that distribute the invariant over map operators.

Even in this relatively small case study, the notion of PO families plays an important role. The two main ideas in this proof are: discovering a *hidden* case analysis (i.e. a missing hypothesis) and using lemmas that distribute the map operators over the invariant — and these are exactly what is needed to solve the (more complicated) *DISPOSE* feasibility proof obligation.

3 The AI4FM System

This section describes the abstract model of the AI4FM system that realises the functionality described in the introduction. Specifically, we describe the meta-information content of *conjectures* that enable strategies to be learned and matched against proof situations, as described in Sections 4–5.

The accumulated information in an AI4FM instantiation can be thought of as a collection of bodies (in the sense of "body of knowledge" as meta-information associated to user theories). Each *body* contains proof tasks (conjectures and their justifications), user-defined functions and types, and strategies. This paper focuses on conjectures, justifications and strategies and we refer the reader to [FJVW13, Chapter 3] for a full description of the model.

3.1 Conjectures

A proof task is a *Judgment*, which contains hypotheses and a conclusion, and a *role*. A *Judgement* can be typing information, a sequent or an equation. A *role* describes the purpose of this task. In addition there can be any number of (attempts at) justifications. Thus:

$$\begin{array}{lll} \mathit{Conjecture} :: & \mathit{what} & : \mathit{Judgement} \\ & \mathit{role} & : \{\textsc{Axiom}, \textsc{Trusted}, \textsc{Lemma}, \textsc{Subgoal}, \cdots\} \\ & \mathit{justifs} & : \mathit{JusId} \xrightarrow{m} \mathit{Justification} \\ & \mathit{specialises} & : [\mathit{ConjId}] \\ & \ldots & \end{array}$$

An example of a low level conjecture would be a natural deduction proof rule for "or elimination" which might be marked as an axiom (Axiom). Within a body for a VDM specification like the heap, a proof obligation generator will create a *Conjecture* for each proof obligation (PO) about the consistency of that single specification. Thus, another conjecture may be the Lemma representing the *NEW* feasibility PO of Section 2.

3.2 Justifications

Turning to *Justification*, it is explicitly envisaged that there can be multiple attempts to justify a proof task (*i.e. Conjecture*s can be mapped to different *Justification*s). When a conjecture is first generated, it will have no justifications. A user might start one proof justification, leave it aside and try another, then come back and complete the first proof. But notice that the notion of whether a proof is complete (in the sense of (transitively) relying only on axioms) is a complex recursive predicate. Overall,

$$\begin{array}{lll} \mathit{Justification} :: & \mathit{by} & : (\mathit{ConjId} \mid \mathit{ToolOP}) \\ & \mathit{with} & : \mathit{ConjId}^* \end{array}$$

A justification which uses an established inference rule will point to its *ConjId*. The *with* field points to any sub-problems that need to be discharged to complete the proof. Notice that such a justification corresponds to one step in a proof: collecting a whole proof requires tracing the attempts at the sub-conjectures. 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 an attached Theorem Prover. Use of an ATP will be recorded as an instance of *ToolOP* — such output will be specific enough to the ATP that it is not further specified here (e.g. a concrete proof script instance).

3.3 Meta-information

Finally, conjectures include meta-information (or *features*):

$$\begin{array}{lll} \mathit{Conjecture} :: \cdots & & \\ \quad \mathit{provenance} & : & (\mathit{Origin} \mid \mathit{Why})^* \\ \quad \mathit{emphTps} & : & \mathit{TyId} \xrightarrow{m} \mathbb{N} \\ \quad \mathit{emphFns} & : & \mathit{FnId} \xrightarrow{m} \mathbb{N} \\ \quad \mathit{other} & : & \cdots \end{array}$$

The additional features of each *Conjecture*, i.e. *provenance*, *emph*asised types or functions, are the key information collected to enable analysis and creation or execution of proof strategies. The *provenance* feature details how a particular conjecture has arisen. For example, for top-level PO conjectures, it would record the type of PO; for sub-goal conjectures, it would record the strategies that lead to that particular goal, e.g. the *provenance* for the *NEW*1 feasibility *Conjecture* nearing its complete justification would be [*EXPOSE-POST*, *WITNESS*, *ONE-POINT*, *INV-BRKDOWN*, *HCA*]. In general, the set of tokens indicating provenance is open ended, yet fairly stable per domain. Registering this information allows simplifying proof strategy search for similar proofs. The AI4FM system would match the provenance with a strategy in an earlier proof and suggest adapting it: e.g. "On a previous similar proof, your next step was this particular strategy". For example, after invariant breakdown for *NEW*1, we get stuck and need a hidden case analysis (HCA) to proceed (see Section 5.1). So, when a similar situation happens for *DISPOSE*1 (i.e. when its proofs nears the invariant breakdown), the AI4FM system might suggest a specialisation of *HCA* as a good candidate—which happens to be the necessary case (see Section 5.2).

Moreover, it would be useful if definitions within bodies of knowledge contained information about their use within the proof obligations/conjectures of the body. For example, this could include properties of operators, such as them being distributive and associative but not commutative — they could influence strategy matching.

4 Playing proof strategies

The most common mode of operation for AI4FM is the intelligent suggestion and application of "intentful" strategies and lemmas to break down and solve POs. In general, the "ethos", described fully in [FW14], is to use strategies to deconstruct a high-level PO to the point where so-called *weakening* lemmas (e.g. goal simplification rules) can be suggested to break down the conjectures until automation can kick in to discharge the remaining goals. AI4FM strategies are *intentful* because they advertise their applicability to conjectures by *MTerms* (the 'M' is for *Matching*) and are optionally annotated with an explicit *intent* as a natural language record of a strategy's purpose and is identifiable by machine.

The suitability of any strategy for progressing the proof of a given conjecture is given by evaluating the propositional *MTerm* over a *conjecture*'s features (as described in Section 3). It is this information that is utilised to select and apply a strategy within the AI4FM system. Such functionality is modelled via two VDM operations: respectively, the *FIND-STRATEGY* operation to suggest a list of appropriate strategies for a given conjecture; and, the *APPLY* operation to execute the strategy found.

The following section describes the structure of a strategy in detail and Section 4.2 instantiates some example strategies. Then, Section 4.3 and Section 4.4 describe the *FIND-STRATEGY* and *APPLY* operations, respectively.

4.1 Anatomy of a strategy

The first component of an AI4FM *strategy* is an *intent*, given as a *Why* token:

$$\begin{array}{ll} \mathit{Strategy} :: & \mathit{intent} : [\mathit{Why}] \\ & \cdots \end{array}$$

The *intent* serves a dual purpose:

1. It serves to explain what the strategy does;
2. It is added to the provenance of the (sub)conjectures generated.

The latter enables subsequent strategies to be suggested *because* this strategy has been applied (see Section 4.2.3, for example). The purpose of a strategy is to progress proofs and the crucial component that instructs AI4FM how that strategy should be applied is the *by* field:

$$\mathit{Strategy} :: \cdots \mathit{by} : (\mathit{conjId} \mid \mathit{ToolIP}) \cdots$$

The *by* field shows that a strategy can be justified by either a call to an internal conjecture (given by a *conjId*) or an external tool (given by a *ToolIP*). Appeals to inference rules, axioms, and previously proved (or even still open) conjectures are given by internal references. Tools can either be part of the theorem prover or can be separately developed "apps" within AI4FM — e.g. [GKL13a]. The *script* component of a *ToolIP* represents the input of different tools can vary.

$$\begin{array}{lll} \mathit{Strategy} :: & \cdots \mathit{weightings} & : \mathit{MTerm} \xrightarrow{m} \mathit{Weight} \\ & \mathit{mvars} & : \mathit{mvar}^* \cdots \end{array}$$

The next component, the *weightings* map, describes *when* a strategy should be applied and contains a strategy's key feature matching objects: *MTerms*. Each *MTerm* is associated with a *Weight*: a natural number that describes the utility of the current strategy for progressing the proof, should that *MTerm* be satisfied. A basic *MTerm* is a proposition built from negation ($\neg$), conjunction ($\wedge$), and an unbounded set of *atomic* (paramaterised) predicates. For example, the atomic $\mathit{prov\text{-}test}(\mathit{VDM\text{-}FEAS}, \mathit{conjId})$ checks if the *VDM-FEAS* token is part of the provenance of the conjecture *conjId*.

Rather than permit choice explicitly in *MTerms*, the weightings *map*, means that a strategy can have multiple *MTerms*, with different associated *weights*. This approach enables multiple (disjunctive) scenarios in which a strategy is applicable and a notion of partial match to be modelled. The weightings are used by the *FIND-STRATEGY* operation to rank strategies by value; matching in a single strategy must therefore return the highest ranking *MTerm* in the *weightings*. Machine learning techniques like PageRank adaptations [KU14] can be used to learn ways to improve the strategies matching rate. For more sophisticated matching, a simple binding mechanism called *matching variables* (*mvars*) for *MTerms* is provided, and is written with the following syntax:

$$\textbf{match}\ ?x\ ?y\ \textbf{with}\ \mathit{atomic}_1(?x, ?y, \mathit{conjId}) \wedge \cdots \wedge \mathit{atomic}_n(?x, ?y, \mathit{conjId})$$

to mean that evaluation of an *MTerm* must instantiate $?x$ and $?y$ as terms from the *judgement* to satisfy the basic *MTerm* part. Any *mvars* in *MTerms* must be declared in the strategy itself. This means that they can also appear in the tool input *script* to customise the behaviour of a strategy based on the match. *MTerms*, *mvars*, and *ToolIP* are exemplified in the next section, but further discussion is postponed until Section 4.3, where they are in action using the *FIND-STRATEGY* operation. Finally, strategies can be organised into a "taxonomy". The idea is perhaps best illustrated by an example:

ℕPeanoInduct specialises InductionProof
ℕCompleteInduct specialises InductionProof

So the final field becomes:

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

Specialisation is a simple example of strategy *capture*, described in Section 5.

4.2 *MWhy* strategies example

To ground the previous section, with a strong focus on *MTerms*, three example strategies, implementing the proof patterns from [FW14], are given.

4.2.1 VDM Structural breakdown. The first illustration of an AI4FM strategy is for *structural breakdown*, which simply decomposes a top-level proof obligation — e.g., the VDM feasibility, narrow-postcondition, or widen-precondition POs [Jon90] — by performing safe introduction rules. For example, it would tackle a feasibility proof obligation of the form:

$$\forall \overleftarrow{\sigma}, \bar{i} \cdot \mathit{pre}\text{-}\mathit{OP}(\overleftarrow{\sigma}, \bar{i}) \ \Rightarrow\ \exists \sigma, \bar{o} \cdot \mathit{post}\text{-}\mathit{OP}(\overleftarrow{\sigma}, \bar{i}, \sigma, \bar{o})$$

where $\overleftarrow{\sigma}$ and $\bar{i}$ are the initial state and inputs and σ and $\bar{o}$ represent the updated state and outputs for a particular operation would be transformed to:

$$\exists \sigma, \bar{o} \cdot \mathit{post}\text{-}\mathit{OP}1_a \wedge \ldots \wedge \mathit{post}\text{-}\mathit{OP}1_n \wedge \mathit{post}\text{-}\mathit{OP}1\text{-}\sigma\text{-}\mathit{inv}_1 \wedge \ldots \wedge \mathit{post}\text{-}\mathit{OP}1\text{-}\sigma\text{-}\mathit{inv}_n$$

That is, quantifiers and assumptions are stripped (and added to the context) and the conclusion is unfolded to a conjunction of postconditions and state invariants. The result of structural breakdown on the feasibility PO for *NEW* from Section 2 would be the following:

$$\frac{\overleftarrow{f_1} \in \mathit{Free}1,\ s \in \mathbb{N}_1,\ l \in \textbf{dom}\ \overleftarrow{f_1},\ l \geq s,\ \mathit{sep}(\overleftarrow{f_1}),\ \mathit{disj}(\overleftarrow{f_1})}{\begin{array}{l}\exists r \in \mathbb{N}_1, f_1 \in \mathit{Free}1 \cdot [r \in \textbf{dom}\ \overleftarrow{f_1} \wedge (\overleftarrow{f_1}(r) = s \wedge f_1 = \{r\} \mathbin{⩤} \overleftarrow{f_1}) \vee \\ (\overleftarrow{f_1}(r) > s \wedge f_1 = (\{r\} \mathbin{⩤} \overleftarrow{f_1}) \cup_m \{r + s \mapsto \overleftarrow{f_1}(r) - s\})] \wedge \mathit{sep}(f_1) \wedge \mathit{disj}(f_1)\end{array}}$$

The *intent* for this strategy is given as *EXPOSE-POST* to describe that it is really just exposing the main postcondition. The *weightings* map consists of

three individual *MTerms* that check whether the conjecture is one of the required top-level VDM POs. For feasibility, the *MTerm* is:
$\quad$ $\textit{prov-test}(\textit{FEAS-PO}, \textit{conjId}) \wedge \neg\, \textit{prov-test}(\textit{EXPOSE-POST}, \textit{conjId})$

Since the *MTerms* for this strategy are straightforward, there are no *mvars* to be recorded and it does not specialise any other strategies.

4.2.2 Hidden case analysis. A common technique for progressing a tricky proof is to introduce new hypotheses to help solve the goal. The price to pay is that these hypotheses need to be discharged at some point. Fortunately, some inference rules allow one to make use of new hypotheses, as in disjunction elimination, where the burden is to prove the same goal under different (disjunctive) hypotheses. This use of disjunction elimination is itself a specialisation of a fundamental concept in theorem proving, the "cut rule":

$$\boxed{\text{cut-disj-elim}}\,\frac{H \vdash C_1 \vee C_2 \quad C_1 \vdash G \quad C_2 \vdash G}{H \vdash G}$$

When there is no explicit disjunction in the goal, yet there is a missing hypothesis to finish the proof, it is called Hidden Case Analysis (HCA). A specific instance of HCA is where the "cut" hypothesis is $\geq$ — its *MTerm* provides a first use of matching variables:
$\quad$ **match** $?hyp$ **with** $\quad \textit{hyp-test}(?hyp, \textit{conjId}) \wedge \textit{top-symb}(?hyp, \geq)$

where a hypothesis with a $\geq$ is matched against the conjecture. This strategy is used in the *NEW* feasibility PO to split $l \geq s$ into $l = s \vee l > s$.

4.2.3 Existential witnessing. The final example strategy, existential witnessing, is a general strategy for progressing with POs with an existential quantifier, by instantiating it with a witness provided by the user. The *MTerm* is:
$\quad$ $\textit{prov-test}(\textit{EXPOSE-POST}, \textit{conjId}) \wedge \textit{top-symb}(\exists, \textit{conjId.what.conc})$

and the intent of this strategy is simply *WITNESS*.

Existential witnessing is an example of a strategy that has specialisations: two important examples of such a specialised existential witnessing strategy are *single-point* and *fully witnessed*.

Single-point existential witnessing. A common situation that arises with POs is the need to witness updated state variables — the PO may be of the form $\exists \sigma \cdot \sigma = t \wedge \ldots$. A strategy provided for this situation, with an *MTerm*:
$\quad$ **match** $?tm\ ?x\ ?y$ **with**
$\quad$ $\textit{prov-test}(\textit{EXPOSE-POST}, \textit{conjId}) \wedge \textit{subtree}(?tm, \textit{conjId.what.conc})$
$\quad$ $\wedge\ \textit{exvar-test}(?x, \textit{conjId.what.conc}) \wedge$
$\quad$ $\textit{top-symb}(=, ?tm) \wedge \textit{left-child}(?x, ?tm) \wedge \textit{right-child}(?y, ?tm)$

which picks out a subterm of the conclusion that has the shape $?x = ?y$ and $?x$ is an existentially bound variable.

The *by* for this strategy could be a call to an Isabelle tactic with the following script: $\textit{apply}(\textit{rule-tac}\ x = ?y\ \textit{in}\ \textit{exI})$, which is the Isabelle command for performing existential introduction with explicit witnesses. The matching of $?x$ and $?y$ as f_1 and $(\{l\} \lessdot \overleftarrow{f_1})$ respectively is the appropriate single-point witness for the $l = s$ case of *NEW* feasibility PO.

Fully witnessed. This specialisation of witnessing has a niche purpose: match conjectures that have been witnessed already, and had all the existentials stripped from them. It does nothing to the goal, but adds *WITNESSED* to the *Conjecture*'s provenance, enabling further proof strategies to attack the witnessed postconditions and invariant: the meat of the PO. The *MTerm* for this strategy is:

$\mathit{prov\text{-}test}(\mathit{WITNESS}, \mathit{conjId}) \wedge \neg\, \mathit{top\text{-}symb}(\mathit{conjId.what.conc}, \exists\cdot) \wedge \neg$
$\mathit{prov\text{-}test}(\mathit{WITNESSED}, \mathit{conjId})$

Specialisations of strategies must have a higher *weight* than their general versions, to ensure that if it matches, the specialised version will be triggered.

4.3 Finding a strategy

Given a conjecture that cannot be solved by proof automation[1], the *FIND-STRATEGY* operation can be used to search through the set of available strategies and check their applicability to the current conjecture, ranked in order of their applicability (using the *Weight*). As described above, this requires evaluating the *MTerms* of a strategy. The function for evaluating an *MTerm* has type:

$\mathit{match} : \mathit{MTerm} \times \mathit{ConjId} \to \mathit{Binding}\text{-}\mathbf{set}$

$\mathit{match}(\mathit{mt}, c) \quad \triangleq \quad \ldots$

where a *Binding* is a map between *mvars* and (user PO) *terms*:

$\mathit{Binding} = \mathit{mvar} \xrightarrow{m} \mathit{term}$

We return a set of bindings because there can be many and each may be interesting for an engineer. Consider, for example, an *MTerm* as follows:

$\mathit{hyp\text{-}test}(?\mathit{hyp}, \mathit{conjId}) \wedge \mathit{top\text{-}symb}(?\mathit{hyp}, \leq)$

which matches any hypothesis of the conjecture that has a $\leq$ as its top symbol. The strategy associated with this *MTerm* could, for example, apply case analysis on $\leq$ (introducing the $=$ and $<$ cases). In a goal $x \leq y, x \leq z \vdash \ldots$, there would be two possible matches. A failure to match the *MTerm* against the conjecture is represented by the empty set. This is different from the case where an *MTerm* is successfully matched, but has no matching variables, which returns a singleton set containing the empty map. Thus, since each strategy has multiple *MTerms*, it must return the highest weight from all those *MTerms* that match, which is:

$$\mathit{Max}(\mathit{ran}(\{\mathit{mt} \mid \mathit{mt} \in \mathbf{dom}\, \mathit{weightings} \wedge \mathit{match}(c, \mathit{mt}) \neq \{\}\} \lhd \mathit{weightings}))$$

and if that *MTerm* contains multiple matches, then each binding must also be returned. This weighted filtering of available strategies is a crucial part of

[1] It is assumed that ATP is always used before strategic intervention is required.

matching evaluated *MTerms*, so that the most suitable strategies with actual instantiations to the user's term are found. This means that the operation is:

$FIND\text{-}STRATEGY\ (f\colon BdId, n\colon ConjId)\ r\colon (StratId \times Weight \times Binding)^*$
$\textbf{pre}\ n \in \textbf{dom}\, guts(f)$
$\textbf{post}\ \ldots \wedge (st, w, bd) \in r$
$\quad \Rightarrow\ \textbf{dom}\, bd \subseteq st.mvars \wedge \exists mt \in \textbf{dom}\, weightings \cdot bd \in match(mt, n)$
$\quad \wedge \ldots$

We use a sequence since we want to order strategies as most applicable, but leave the *Weight* to show the distribution of applicability. We give the precondition and the part of the postcondition that ensures that any returned strategies and bindings are indeed a match. We elide the rest of the postcondition for brevity, but the additional details can be seen in [FJVW13].

4.4 Applying a strategy

Once an engineer has chosen the strategy to use, based on those found to be applicable, the *APPLY* operation can be used. *APPLY* takes the conjecture and a strategy with associated binding and applies it, updating the state by:

1. Adding new conjectures generated by the strategy, including the *Why* of the applied strategy as part of the *provenance* of the new conjectures.
2. Adding a justification to the conjecture that the strategy was applied to, including any tool output, *ToolOp*.

If the strategy uses an external tool, then the *Binding* must be used to instantiate any potential *mvars* in the *script* of the tool input, *ToolIP*. If the strategy fails, then the operation will return *false* and not update the state.

The signature of *APPLY* and part of the postconditions relevant to the provenance of generated conjectures:

$APPLY\ (f\colon BdId, n\colon ConjId, s\colon StratId, b\colon Binding)\ r\colon bool \times conjId\textbf{-set}$
$\textbf{ext wr}\ \Sigma$
$\textbf{pre}\ \ldots$
$\textbf{post}\ (r = (true, cs)$
$\quad \Rightarrow\ \ldots \wedge \forall c \in cs \cdot c \in \textbf{dom}\, \Sigma.guts \wedge s.intent \in c.provenance \wedge \ldots)$
$\quad \wedge (r = (false, cs)\ \Rightarrow\ \ldots)$

The postcondition ensures that all of the generated conjectures (the set *cs*) have been added to the *guts* of the *body*, and that the provenance has been updated to include the *intent* of the applied strategy.

5 Capturing proof strategies

When an expert has to intervene to help progress a proof, this is identified within AI4FM as an opportunity to learn both new proof techniques and new *why*s of a proof technique. The key aim of AI4FM is to *learn* from expert intervention in a single proof to improve automation in subsequent proof attempts that are similar in some way. We have devised several ways in which AI4FM can learn new strategies and in this section we describe another example of the most commonly used technique: strategy capture by *specialisation*.

After showing strategy specialisation for existential variable witnessing, we motivate this technique with another example with hidden-case analysis. Section 5.1 describes how this can be performed in AI4FM. Finally, in Section 5.2, we briefly summarise other ways in which AI4FM can learn from an expert.

5.1 Hidden Case Analysis

As described in Section 4.2.2, exposing a hidden case analysis is an important technique for progressing proofs, based on the specialised *cut-disj-elim* rule from above. The strategy behind this is the act of introducing case analysis by this specialised version of the "cut rule". It introduces the two cases and a lemma that requires showing that the disjunction holds in the current context.

For example, applying *cut-disj-elim* with $C_1 = x > y$ and $C_2 = x = y$ is the correct case analysis for the $NEW1$ feasibility proof obligation described in Section 4.2.2. The goal $x \geq y \vdash x > y \vee x = y$ can then be proved as a lemma. This example strategy was first created for the $NEW1$ feasibility proof, which requires the hidden case split on the $\geq$ present in its precondition.

To learn a new strategy from the basic inference rule strategy *cut-disj-elim*, we need to provide an *MTerm* and reference the lemma generated as proved. This done, the system can automatically suggest the specialised $\geq$ hidden-case analysis strategy in similar situations: namely, whenever $\geq$ appears in the hypothesis and the user is stuck (i.e. no known strategy is applicable and the user explicitly asks help to AI4FM), alternative hidden-case analysis or indeed instantiation of known ones could be suggested. This is detailed below.

5.2 Strategy specialisation

The process of capturing new strategies works in two ways: a priori, where the expert user interacts and informs AI4FM about novel ideas from previous known successful ones; and post-facto, where searching/clustering procedures can try and learn new strategies from successfully applied ones. So far, we have investigated the former, and are working on the latter.

Thus, at the point where the (expert-)user suggests the hidden-case analysis for $\geq$ as a new strategy, the AI4FM system could request the user for alternative hypothesis generating instances of the "cut rule" that would make sense in the context. For example, spotting that $\geq$ is a reflexive pre-order maximises the chance of a strategy generalising this to $\leq, \subseteq, \cdots$. Moreover, assuming the *provenance* information from the $DISPOSE1$ feasibility proof back to *cut-disj-elim*, but without the knowledge what to plug in for the disjuncts, the user could be asked to suggest something. In this case, the required disjunctions are related to set/map emptiness. These strategies are likely to contain *mvars* so that different bindings when searching/applying the strategies are possible.

Hidden case analysis in DISPOSE1. The feasibility proof for *DISPOSE*1 (see details in [FJVW13, App. E.8, F]), needs a specialised version of hidden case analysis, where the new hypotheses are about whether the adjoining sets of memory below and above the memory being disposed are empty or not. This is important because it will determine, because of the *sep* invariant, the largest contiguous memory as the correct state update. So, if neither below nor above is empty, the returned value is their unions, whereas if either is empty, the formulae in *DISPOSE*1 postcondition is greatly simplified (e.g. the summation of sizes and discovery of minimal location are affected).

All these cases rely on the fact of the application of the hidden cases analysis strategy, where the hidden disjunction is about emptiness of both sets (leading to 4 cases). A detailed technical explanation for this is in [FJVW13, Appendix E.8, F] and [FJVW13, Section 3.2.3]. It discusses how the hidden case analysis strategy applied in *NEW*1 (at the right time) is specialised for *DISPOSE*1, providing the (expert-)user informing the system about the hidden disjunct on the empty sets. The capturing was played independently in two proof modes (i.e. tactical and Isar) in Isabelle to the same result.

The **interesting observation** here is that despite great difference in details, the overall proof strategy for *NEW*1 is quite similar to *DISPOSE*1, as we predicted (in the AI4FM hypothesis above). That means, **with enough strategies available, the level of successful application to different problems is likely higher in our experience with the idea so far**.

6 Related work and status

As discussed above, the system described in this paper should be seen in the wider context of the AI4FM project, where the project partners and collaborators have been working on tools that can be used by the system described here. Some of these related tools and how they fit are:

PSGraph. *Proof Strategy Graph* [GKL13b] is a tool for encoding proof strategies. PSGraphs can be composed staticaly based on a notion of *types* on incoming and outgoing goals. The goal types simultaneously allow for additional control of a strategy's behaviour and understanding of its purpose and result. The goal types on PSGraphs correspond closely to our *MTerms*, enabling the learning of new AI4FM strategies from instances of PSGraphs.

Lemology. Lemology [HKJM13] suggests lemmas by analogy with a similar theorem; furthermore, it can be used to suggest analogous conditions to speculate new lemmas. The AI4FM should capture enough details of proof and conjecture features to detect analogy and suggest applicable lemmas.

IsaCoSy. IsaCoSy [JDB11] can generate conjectures based on functions and types of interest, run them through a counterexample checker to eliminate obvious false conjectures, then use an automated theorem prover to attempt to prove the remainder. We plan to use the *emphFns* and *emphTps* from unproven conjectures to feed into IsaCoSy to help generate lemmas that can progress a proof automatically.

Other related work. In [HK13], machine learning is used to identify clusters of lemma usefulness. They apply the technique to simple inductive theorems in both CoQ and ACL2 provers. The idea is to try and identify lemmas with high "quality", in the sense of helping solve more goals. A key difference to our approach is that we anticipate the use of meta-information for learning, instead of the actual raw proof data, given it is not easy to gather enough of it (i.e. one would not provide 100 samples of the same proof).

In [KU14], authors describe a way to mine proof tracing information in order to detect lemma relevance, duplication, and ranks them in order of importance. The idea uses machine learning (clustering and PageRank) techniques by sifting through the large amount of data found in the proof object (inference graph). It takes into account intermediate lemmas (or sub-goals), as well as user defined lemmas. Their approach observe the problem at this lowest-level of proof object, as well as at the theorem prover's tactic-application level, where the difference helps determine what is a useful lemma (in terms of its applicability during proof), as opposed to what makes the theorem prover "happy" (in terms of the interaction between the lemma shape and the way it relates to the various proof tactics applied). We are trying to understand their experimental setup data in order to make use of similar techniques in AI4FM. Their work, however, has no notion of user-supplied input, where proof intent (e.g. meta-proof information) is provided in order to search for similar proof strategies on different goals.

Status. Proof engineering is essential for scalability: it takes a good amount of unrelated proof effort to enable one to tackle the actual proof obligations of interest. Lemmas are useful whenever one needs to either: decompose a complex problem; fine-tune the theorem prover's rewriting abilities to given goals; generalise a solution of some related (usually more abstract) problem; and to provide alternative solutions of the same data structure being modelled; *etc.*

In our experiment we have tested our hypothesis by having the same proof task performed independently by three different people with three different backgrounds (formal methods proof expert, Isabelle proof expert, MSc student), in two different provers (Isabelle/HOL and Z/EVES), and encoded in two different methods (VDM and Z) on medium size refinement problem (i.e. the Heap). Analysing the proof traces and scripts of the Isabelle development (using Perl) looking for commonalities and differences. On the expert proof engineer development, our new lemmas on VDM maps in Isabelle were the ones with highest reuse rate (at 22%), with other available Isabelle library lemmas reuse being quite high too (at 38%). On the Isabelle expert, the ratio was slightly different at 16% and 65% respectively. The effort on PO-specific weakening lemmas and type bridges was comparable at 23% and 17% for each expert. This indicates that a considerable amount of effort (around 20% for both experts) was related to setting up VDM map operators and lemmas in Isabelle, whereas around the same effort was needed on the actual POs. Arguably, the VDM lemmas are reusable across problems, hence the patterns described for the Heap problem do transfer across problems (in VDM at least). The encoding in Z/EVES was relatively straightforward, as there were no issues with undefinedness and the

Z mathematical toolkit is quite similar to VDM's. This part of the experiment was useful, however, in early detection of possible proof-difficulty in the model, which only appeared much latter in the Isabelle development.

Prototype implementation. For a comprehensive capture of proof strategies, a prototype AI4FM system is being developed. It allows us to track proof histories and provides a convenient user interface and scalable data persistence to record the necessary meta-information about the expert's proof process and strategies [Vel12,FJV14]. The current system implementation[2] supports integration with and proof capture from both Isabelle and Z/EVES proof assistants. While it currently employs an older version of *MWhy* (meta-)information model and requires significant engineering effort to keep up with developments of underlying provers, having a tool support for proof capture, strategy extraction and replay expands the AI4FM approach beyond pencil-and-paper exercises.

7 Conclusions

From our experiments, the use of *MWhy* (meta-)information about the proof process has helped reduce the burden of proof within three separate proof exercises (see `http://www.ai4fm.org`). This confirms our hypothesis that it is possible to learn (or capture) (re-)playable proof strategies across the same Formal Method (FM) problem, such that from a few POs and key ideas, remaining (recalcitrant and tedious) POs can be discharged.

This paper describes a summary of our meta-information capturing, playing, and replaying AI4FM system, where *MWhy* represents the state, and several operations over this state represent finding suitable proof strategies (play), applying them to a different goal (replay), as well as suggesting specialisations of available strategies as a means to improve strategy application (capture).

This abstract description of the AI4FM system can be implemented in different ways. The prototype *ProofProcess* framework mentioned above acts as an add-on to different theorem provers by capturing and storing the meta-information externally. Alternatively, the AI4FM system could be closely integrated with a theorem prover, allowing –for example– an expert to specify the meta-information within the formal specification, etc.

Further work. When discovering different strategies, we need to create specific *MTerms*. So far, we have created *MTerms* common to a category of FM POs. We are working to expand that by performing proof exercises over a variety of examples, as well as tapping into previous proofs by authors from the Grand Challenge experiments [BFW09,FW08,FW09].

Acknowledgements

Other AI4FM members helped us understand important problems in automated reasoning. We are grateful for discussions with Moa Johansson on lemma generation. EPSRC grants EP/H024204/1 and EP/J008133/1 support our research.

[2] *ProofProcess* framework, `http://github.com/andriusvelykis/proofprocess`.

References

BFW09. Andrew Butterfield, Leo Freitas, and Jim Woodcock. Mechanising a formal model of flash memory. *Science of Comp. Prog.*, 74(4):219–237, 2009.

FJV14. Leo Freitas, Cliff B. Jones, and Andrius Velykis. Can a system learn from interactive proofs? In Andrei Voronkov and Margarita Korovina, editors, *HOWARD-60. A Festschrift on the Occasion of Howard Barringer's 60th Birthday*, pages 124–139. EasyChair, 2014.

FJVW13. Leo Freitas, Cliff B. Jones, Andrius Velykis, and Iain Whiteside. How to say why. Technical Report CS-TR-1398, Newcastle University, www.ai4fm.org/tr, November 2013.

FW08. Leo Freitas and Jim Woodcock. Mechanising Mondex with Z/Eves. *Formal Aspects of Computing*, 20(1):117–139, 2008.

FW09. Leo Freitas and Jim Woodcock. A chain datatype in Z. *International Journal of Software and Informatics*, 3(2-3):357–374, 2009.

FW14. Leo Freitas and Iain Whiteside. Proof patterns for formal methods. In *Formal Methods*. Formal Methods Europe, Springer, May 2014.

GKL13a. Gudmund Grov, Aleks Kissinger, and Yuhui Lin. A graphical language for proof strategies. To appear in LPAR'13. Available at arXiv:1302.6890, 2013.

GKL13b. Gudmund Grov, Aleks Kissinger, and Yuhui Lin. A graphical language for proof strategies. In McMillan et al. [MMV13], pages 324–339.

HK13. Jonathan Heras and Ekaterina Komendantskaya. ML4PG in computer algebra verification. In *Conf. on Intelligent Computer Mathematics*, 2013.

HKJM13. Jónathan Heras, Ekaterina Komendantskaya, Moa Johansson, and Ewen Maclean. Proof-pattern recognition and lemma discovery in acl2. In McMillan et al. [MMV13], pages 389–406.

JDB11. Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.

JFV13. Cliff B. Jones, Leo Freitas, and Andrius Velykis. Ours is to reason why. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods*, volume 8051 of *LNCS*, pages 227–243, 2013.

JJLM91. C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. *mural: A Formal Development Support System*. Springer-Verlag, 1991.

Jon90. C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall, 1990.

JS90. C. B. Jones and R. C. F. Shaw, editors. *Case Studies in Systematic Software Development*. Prentice Hall International, 1990.

KU14. Cezary Kaliszyk and Josef Urban. Learning-assisted theorem proving with millions of lemmas. *CoRR*, abs/1402.3578, 2014.

MMV13. Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors. *LPAR-19. Proceedings*, volume 8312 of *LNCS*. Springer, 2013.

Pau94. L. Paulson. Isabelle: A Generic Theorem Prover. *LNCS*, 828, 1994.

Saa97. Mark Saaltink. The Z/EVES system. In Jonathan Bowen et al, editor, *ZUM '97: The Z Formal Specification Notation*, volume 1212 of *Lecture Notes in Computer Science*, pages 72–85. Springer Berlin / Heidelberg, 1997.

Vel12. Andrius Velykis. Inferring the proof process. In Christine Choppy et al., editors, *FM2012 Doctoral Symposium*, Paris, France, August 2012.

A Model

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

$$\begin{array}{llll} Body & :: & domain & : Domain \\ & & functions & : FnId \xrightarrow{m} FnDefn \\ & & types & : TyId \xrightarrow{m} TyDefn \\ & & guts & : ConjId \xrightarrow{m} Conjecture \\ & & strats & : StratId \xrightarrow{m} Strategy \end{array}$$

$$\begin{array}{llll} FnDefn & :: & type & : Signature \\ & & tags & : FnTag\textbf{-set} \\ & & defn & : \left[Definition\right] \end{array}$$

$$\begin{array}{llll} Conjecture & :: & what & : Judgement \\ & & role & : \{\textsc{Axiom}, \textsc{Trusted}, \textsc{Lemma}, \textsc{Subgoal}, \cdots\} \\ & & justifs & : JusId \xrightarrow{m} Justification \\ & & specialises & : [ConjId] \\ & & provenance & : (Origin \mid Why)^* \\ & & emphTps & : TyId \xrightarrow{m} \mathbb{N} \\ & & emphFns & : FnId \xrightarrow{m} \mathbb{N} \\ & & other & : \cdots \end{array}$$

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

$$\begin{array}{llll} Sequent & :: & hyps & : Judgement^* \\ & & goal & : Judgement \end{array}$$

$$\begin{array}{llll} Justification & :: & by & : (ConjId \mid ToolOP) \\ & & with & : ConjId^* \end{array}$$

$$ToolOP = \cdots$$

$$Origin = Token$$

$$Why = Token$$

$$FnTag = \{\textsc{Inv}, \textsc{Pre}, \textsc{Post}, \ldots\}$$

$$\begin{array}{llll} Strategy & :: & intent & : [Why] \\ & & by & : (ConjId \mid ToolIP) \\ & & weightings & : MTerm \xrightarrow{m} \mathbb{N} \\ & & mvars & : mvar^* \\ & & specialises & : [StratId] \end{array}$$

$$\begin{array}{llll} ToolIP & :: & name & : \{\textsc{Sledgehammer}, \mathrm{SMT}, \textsc{Simplify}, \mathrm{PS}\textsc{Graph}, \cdots\} \\ & & script & : Token \end{array}$$

$$Atomic = prov\text{-}test \mid tag\text{-}test \mid hyp\text{-}test \mid \cdots$$

$$\begin{array}{llll} MTerm & :: & mvars & : mvar^* \\ & & mterm & : \text{propositional terms over } Atomic \end{array}$$

$$\begin{array}{l} Relationship = \textsc{Uses} \mid Specialisation \mid Morphism \mid Isomorphism \mid \\ \qquad\qquad\qquad\qquad Inherits \mid Sub \mid \textsc{Similarity} \mid \cdots \end{array}$$