

Ideas for a high-level proof strategy language: *starting* the AI4FM project

Cliff B. Jones (University of Newcastle)
Gudmund Grov (University of Edinburgh)
Alan Bundy (University of Edinburgh)

AFM-10
Edinburgh
2010-07-14



AI4FM

Jones, Grov & Bundy [1]

Contents

Background

Strategy language

Conclusion



AI4FM

Jones, Grov & Bundy [2]

Motivation

- (most) formal methods generate POs
 - coherence of a layer
 - consistency between layers
 - “method” as very high-level strategy!
- on industrial examples
 - some 5-20% require user interaction to achieve proof (3-40%?)
 - significant user expertise is required to discharge POs manually
 - proof effort becomes a bottleneck in such developments
- groups of proofs turn out to be of **similar form**
 - empirical evidence suggests “families” of 20+ POs (Event-B)
- during development: models **evolve!**
- **can we identify/learn/recycle proof “strategies”?**



Model-based formal development methods

- formal development methods are remarkably similar
 - Mondex case study in Formal Aspect of Computing (20:1)
- one example: a “data reification” step requires:
 - VDM: retrieve function
 - (classical) B: linking invariant
 - Event-B: gluing invariant
 - Morgan: coupling invariant
- all give rise to (similar type of) POs
- target:
 - increased automation for such methods
 - **real industrial applications**
 - **... and use!** — cf. J Moore yesterday (40 year view)



Research goal

- when heuristics don't work! (which happens somewhere)
- **suppose:** user manually proves one exemplar proof
- **then:** proof of POs in the same "family" can be discharged by "reusing" the strategy
- implies more robustness to model changes
- **Research hypothesis:** By learning from an exemplar proof we can automatically discharge POs of "similar form"
- so, not just looking for smarter heuristics
- ... benefit from user in the loop
- I confess, implications of this still evolving in our minds



Example of (VDM) data *reification*

Recording equivalence relations

- abstract specification

$Part = (X\text{-set})\text{-set}$

$\mathbf{inv} (p) \triangleq \bigcup p = X \wedge is\text{-prdisj}(p) \wedge \{\} \notin p$

no problem with *operations* preserving invariant

- intermediate data reification step

$Keyed = X \xrightarrow{m} Key$

- Fischer/Galler representation

$Forest = X \xrightarrow{m} X$

$\mathbf{inv} (m) \triangleq \forall s \subseteq \mathbf{dom} m \cdot s \neq \{\} \Rightarrow \neg(\mathbf{rng}(s \triangleleft m) \subseteq s)$



(Two) VDM reification rules

- homomorphic mapping from representation to abstraction (cf. Milner's "rejected" JACM paper)
- relation (cf. Lucas' "twin machine") – [Nip86]
- these are: strategies for proving "observational equivalence"!
- **VDM(1) requires that "adequacy" of representation**

$$\forall a \in A \cdot \exists r \in R \cdot a = retr(r)$$

- **different strategies to cope with existential quantifiers**
- we have used several such reifications as small tests
- ... aware that industrial structures are much bigger



Nature of such proofs

- rarely deep
 - we are *not* trying to prove real math theorems!
 - ... yet?
- complexity reduced by layering abstractions
 - cf. next speaker! (and talk at VERIFY workshop)
 - return to question of whether this should be key strategy!?
- lots of detail (on larger examples)
 - data structures play a key role
 - return to under "Toy Versions"



GUI matters — cf. mural [JJLM91]

viewing *hypotheses/goals* (nested)

Proof for $\wedge\vee$ -dist		attempts
rule:	$\wedge\vee$ -dist	main proof
theory:	Propositional LPF	
marked items:	<input type="checkbox"/> clear	
consistent incomplete not assumed		
tactic tool	justif tool	
main proof		
$\text{h1 (A } \wedge \text{ (B } \vee \text{ C))}$ $\text{1 (B } \vee \text{ C)} \quad \text{by } \wedge\text{-E-left on [h1]; []}$ 2 [] 2.h1 B $\text{2.c ((A } \wedge \text{ B) } \vee \text{ (A } \wedge \text{ C))} \quad \text{<Justif>}$ 3 [] 3.h1 C $\text{3.c ((A } \wedge \text{ B) } \vee \text{ (A } \wedge \text{ C))} \quad \text{<Justif>}$ $\text{c ((A } \wedge \text{ B) } \vee \text{ (A } \wedge \text{ C))} \quad \text{by } \vee\text{-E on [1]; [2, 3]}$		



AI4FM

Jones, Grov & Bundy [9]

GUI matters

user controls what viewed

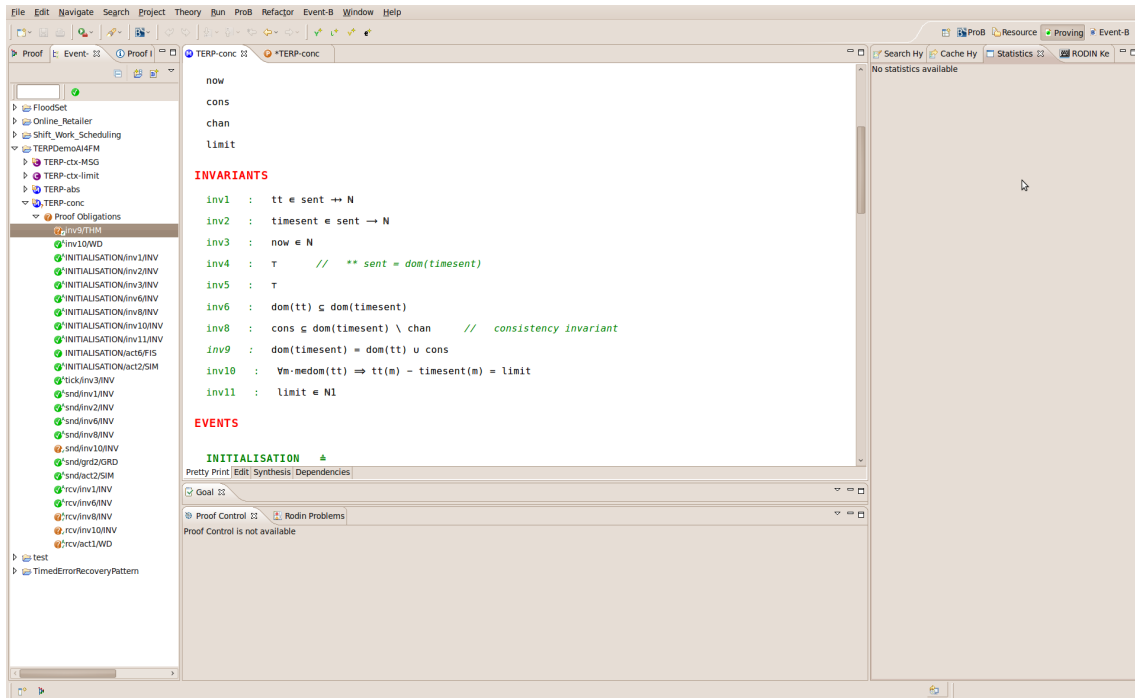
Propositional LPF		parents
signature	$\wedge : \text{Constant}$ $\frac{(\neg(\neg[[e1]]) \vee (\neg[[e2]]))}{[[e1]] \wedge [[e2]]}$ <p>accept</p>	$\vee : \text{Constant}$ $[2,0]$ $[[e1]] \vee [[e2]]$ <p>accept</p>
rules	$\wedge\text{-I : Rule}$ $\frac{\{ \} \quad \{ e1, e2 \}}{(e1 \wedge e2)}$ <p>accept</p>	$\vee\text{-E : Axiom}$ $\frac{\{ []\{e1\} \vdash e, []\{e2\} \vdash e \}}{\{ (e1 \vee e2) \}}$ <p>e</p> <p>accept</p>
rule groupings		prop. elim. rules
tactics		ApplyMixedStep ApplyRuleBackwa ApplyRuleForward ApplyRulesBackwa ApplyRulesForward ForwardStep MultiApplyRulesBa oracles PropositionalOrac



AI4FM

Jones, Grov & Bundy [10]

GUI matters — cf. Rodin Platform www.event-b.org user aware of background progress



Contents

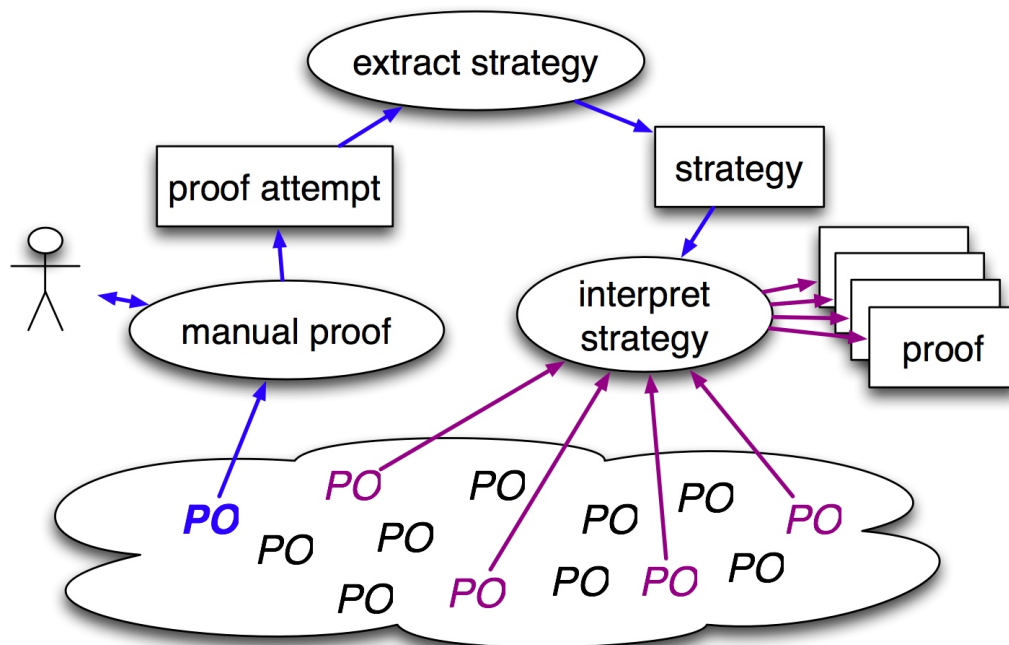
Background

Strategy language

Conclusion



Role of strategy language



AI4FM

Jones, Grov & Bundy [13]

What *is* a proof “strategy”?

- not (I think) tactics/tacticals
- don't show me the finished proof; but ...
- ... proof creation order
 - forwards??
 - backwards?
 - **splits!**
- heuristics for splitting big proof task into smaller ones
 - sadly, no metric for “simpler”
 - analogy: chess heuristics
 - ... in this situation, spawn lemmas/sub-goals like ...
 - come back to parallelism
 - cont ...



AI4FM

Jones, Grov & Bundy [14]

Big questions

- devise a language for strategies
- how to extract strategies
- recognise which strategy most applicable (in a particular situation)
 - I now suspect this is hardest/where AI will pay off



A few heuristics for splitting

- disjunctions in hypotheses suggest case splits lower level
- conjunctions in goals often best tackled separately
- tackling equalities
 - direction change?
 - split to inequalities?
- all operators (functions) in the same algebra
 - use rewrite rules
 - expand out (e.g. ...)
- decide on where to fight battle:
 - within datatype/dicourse vs.
 - map down
- goal uses operators in different algebra (from hyps)
 - use definitions
- recursive data structures/functions
 - (move towards) induction



A sequence of lemmas

- a strategy may describe a sequence of intermediate lemmas that could be spawned:
prove lemmas L_1, L_2, \dots, L_n s.t. GOAL follows
- sub-goals may have other strategies
- similar to the “ACL2 approach”
 - experiences from the Piton project [Moo96] confirms
- at a “reasonable” level, discharge using
 - e.g. a (correct) decision procedure
 - with a time-out feature
- requires a notion of a **gap**
 - an unproved sub-goal/lemma
 - “hiproofs” — see paper



We (also) want “nice” proofs

- ... for re-use
- so, might create multiple proofs
 - choose “simplest”
- abstract out lemmas
- a failed proof can tell us more than one that forced through!
 - suppose rewriting (has obvious \cap/\cup distribution) doesn't see
 $A \cap (B \cup C \cup D) = A \cap B \cup A \cap C \cup A \cap D$
 - ... so maps down to FoPC
 - we don't want to copy this
 - better to go back to the set expression



Spotting the clues to *select* a strategy

- classification of proof tasks
 - data structures
 - shape of PO (cf. earlier example of data reification)
 - information from PO Generator
 - e.g. refinement proof, proof of invariant preservation, ...
 - ...
- combination of a high-level proof strategy
 - ... with a “vocabulary” of terms
 - instantiated using data structure information
- maybe even problem domain
 - e.g. “rail systems” make heavy use of relational compositions
- machine learning?



Evidence for strategy language: cf. [BBHI05]

Rippling

- provides evidence for such high-level strategy language
- Rippling is a **proof plan**
 - AI technique for mechanising formal reasoning based upon high level proof patterns
- Rippling is a rewriting technique
 - based on difference reduction
 - achieved by annotations
- great flexibility, e.g. productive use of failure
- wide range of applications
 - software/hardware verification, synthesis, correction of faulty specifications, ...
 - promotes **reuse** across domains
- provides starting point for the strategy language



Pointers

- proof critics (as used in rippling)
- extraction of “Toy problems” – J. Moore with ACL2
 - simpler data structure
 - but user expertise achieves “strategy carry over”
 - fits our research hypothesis
- generalisation
- general forms of induction axiom (“complete”, ordering)
- learn from an expert user!
 - cut down search space
 - “machine learn” selection of hypotheses/derived rules
 - mirror explanation to another specialist
- see paper for:
 - hiproofs
 - gazing
 - anti-unification



Known unknowns

- can we derive anything from “failed” proofs
- we need complete proof transcripts (not just final proofs)
 - “instrument” the Rodin Tools!
- ... and ... remember most POs start out being wrong!
- failure (to prove) can mark non-theorem!
 - in fact, many POs are initially not theorems!
 - easy-to-use tools required to help detect silly mistakes
- “similar proofs” will
 - deviate more than patterns currently captured in proof planning & rippling
 - need higher level of abstraction
- we expect to use additional kinds of abstraction
 - data structure, form, POG, ...
 - cf. (unfunded) Phase I — but see Teresa’s talk
- impact of SAT/SMT tools on proof style



Contents

Background

Strategy language

Conclusion



AI4FM

Jones, Grov & Bundy [23]

Our plan

- analyse a lot of
 - POs (instrumented Rodin-Tools?)
 - ... their (expert-provided) proofs attempts
 - and at identifying “families”
- **based on analysis, develop a strategy language**
- provide tool support (Isabelle?)
 - to extract a strategy out of an exemplar proof
 - to interpret strategies to discharge “similar POs”
- a lot of challenges
 - classification of POs
 - creating a strategy language
 - deriving strategies from proofs
 - interpreting strategies for the same “family”



AI4FM

Jones, Grov & Bundy [24]

Parallelism

at least as good as (more scope than) Rodin tools

```

now
cons
chan
limit

INVARIANTS
inv1 : tt e sent ↔ N
inv2 : timesent e sent → N
inv3 : now ∈ N
inv4 : T // ** sent = dom(timesent)
inv5 : T
inv6 : dom(tt) ⊆ dom(timesent)
inv8 : cons ⊆ dom(timesent) \ chan // consistency invariant
inv9 : dom(timesent) = dom(tt) ∪ cons
inv10 : ∀m. medom(tt) ⇒ tt(m) - timesent(m) = limit
inv11 : limit ∈ N1

EVENTS

INITIALISATION

```



AI4FM

Jones, Grov & Bundy [25]

Summary

- useful automation can be achieved by learning from one proof
 - to discharge “similar” POs
 - strategies hopefully robust to (many) model changes
- believe this is applicable to:
 - most model-based top-down formal development methods
 - ... possibly to bottom-up approaches as well
- strong industrial & academic need for such tool
 - e.g. Systerel, SAP, ACL2 (Piton), ...
- not looking to find generic proof structures:
 - the user must still provide the exemplar proof
- remember, this project just starting
 - track via www.ai4fm.org
 - see there for mailing list



AI4FM

Jones, Grov & Bundy [26]

References



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.



C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore.

mural: A Formal Development Support System.

Springer-Verlag, 1991.



J. Strother Moore.

Piton: A Mechanically Verified Assembly-Level Language.

Springer, 1996.



T. Nipkow.

Non-deterministic data types: Models and implementations.

Acta Informatica, 22:629–661, 1986.

