

AI4FM

Gudmund Grov (University of Edinburgh)
Cliff B. Jones (University of Newcastle)
Alan Bundy (University of Edinburgh)
Andrew Ireland (Heriot-Watt University)

Refinement Based Methods
for the Construction of Dependable Systems
Schloss Dagstuhl
13.09.09 - 18.09.09

Motivation

- ▶ Most formal methods give rise to POs.
- ▶ 5 – 20% of these will (still) require interactive proofs.
- ▶ In industrial scale developments a large amount of user expertise is required to manually discharge POs.
- ▶ Proof automation is a bottleneck in such developments.
- ▶ Many of these proofs will be of **similar form**.
- ▶ Models **change** during development
 - ▶ existing proofs may change as a result of model changes.

Research Hypothesis

- ▶ The user manually proves one exemplar proof.
- ▶ **Consequently:** once the hurdle of this exemplar proof is passed then proof of POs in the same “family” can be discharged by “manipulating” the exemplar proof.
- ▶ **Research hypothesis:** By learning from the exemplar proof we can automatically discharge the POs of “similar form”.
 - ▶ This also implies more robustness to model changes.

Research Hypothesis

- ▶ The user manually proves one exemplar proof.
- ▶ **Consequently:** once the hurdle of this exemplar proof is passed then proof of POs in the same “family” can be discharged by “manipulating” the exemplar proof.
- ▶ **Research hypothesis:** By learning from the exemplar proof we can automatically discharge the POs of “similar form”.
 - ▶ This also implies more robustness to model changes.
- ▶ Targets **real industrial needs** as exemplified yesterday:
 - ▶ **Mathieu Clabaut (Systemel):** Proof engineering
 - ▶ design proof for **management & reuse**
 - ▶ **Andreas Roth (SAP)**
 - ▶ guided tool usage: “... *verification tools may fail because the problem is not typical...*”
 - ▶ the need for robust tools.

Model-based formal development methods

- ▶ Model-based formal development methods are remarkably similar¹.
- ▶ For example, a reification/refinement step requires
 - ▶ VDM retrieve function
 - ▶ (classical) B linking invariant
 - ▶ Event-B gluing invariant
 - ▶ Morgan's coupling invariant.
- ▶ All of them give rise to (similar type of) POs.
- ▶ Our proposal targets increased automation for most model-based formal development methods.

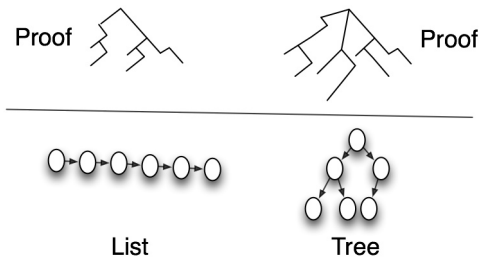
¹see Formal Aspects of Computing Vol. 20, No. 1, 2008.

Approaches to automation

- ▶ Model checking/SMT solvers
 - ▶ automatic “push-button” techniques
 - ▶ limited expressiveness.
- ▶ Theorem provers
 - ▶ search space too large for complete search
 - ▶ most contain heuristics to guide search
 - ▶ user-interaction still required for 5 – 20% of POs.
- ▶ We aim to complement these techniques
 - ▶ by targeting those 5 – 20% of POs
 - ▶ achieved by exploring new information from an exemplar proof
 - ▶ and reuse this to automatically discharge “similar” POs.

“Re-use” of proof

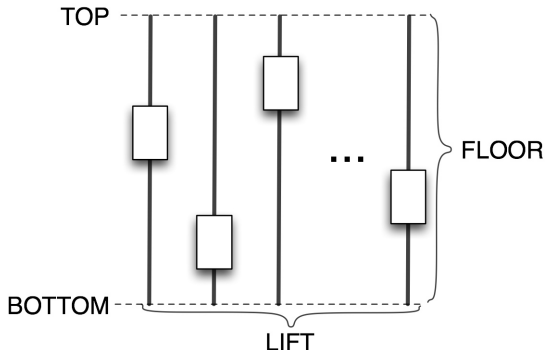
- ▶ “Similar form” does not mean same proof
 - ▶ different lemmas required
 - ▶ different instantiations
 - ▶ different data structures
 - ▶ ...



A lift example in Event-B: specification

axiom1 : $\forall f \cdot (BOTTOM \leq f \wedge f \leq TOP) \Leftrightarrow (f \in FLOOR)$

inv1 : $lift \in LIFT \rightarrow FLOOR$



A lift example in Event-B: specification

axiom1 : $\forall f \cdot (BOTTOM \leq f \wedge f \leq TOP) \Leftrightarrow (f \in FLOOR)$

inv1: $lift \in LIFT \longrightarrow FLOOR$

EVENT *move_up* $\hat{=}$

ANY *l*

WHERE

grd1: $l \in LIFT$

grd2: $lift(l) < TOP$

THEN

act1: $lift(l) := lift(l) + 1$

EVENT *move_down* $\hat{=}$

ANY *l*

WHERE

grd1: $l \in LIFT$

grd2: $lift(l) > BOTTOM$

THEN

act1: $lift(l) := lift(l) - 1$

A lift example in Event-B: POs

axiom1 : $\forall f \cdot (BOTTOM \leq f \wedge f \leq TOP) \Leftrightarrow (f \in FLOOR)$

inv1: $lift \in LIFT \longrightarrow FLOOR$

$lift \in LIFT \longrightarrow FLOOR$

$l \in LIFT$

$lift(l) < TOP$

\vdash

$lift \triangleleft \{l \mapsto lift(l) + 1\}$
 $\in LIFT \longrightarrow FLOOR$

$lift \in LIFT \longrightarrow FLOOR$

$l \in LIFT$

$lift(l) > BOTTOM$

\vdash

$lift \triangleleft \{l \mapsto lift(l) - 1\}$
 $\in LIFT \longrightarrow FLOOR$

A lift example in Event-B: POs

axiom1 : $\forall f \cdot (BOTTOM \leq f \wedge f \leq TOP) \Leftrightarrow (f \in FLOOR)$

inv1: $lift \in LIFT \longrightarrow FLOOR$

$lift \in LIFT \longrightarrow FLOOR$

$l \in LIFT$

$lift(l) < TOP$

\vdash

$lift \Leftarrow \{l \mapsto lift(l) + 1\}$
 $\in LIFT \longrightarrow FLOOR$

$lift \in LIFT \longrightarrow FLOOR$

$l \in LIFT$

$lift(l) > BOTTOM$

\vdash

$lift \Leftarrow \{l \mapsto lift(l) - 1\}$
 $\in LIFT \longrightarrow FLOOR$

Proof outline:

- ▶ lemma $lift(l)+1 \in FLOOR$
- ▶ inst $[lift(l)+1/f]$ in *axiom1*

Proof outline:

- ▶ lemma $lift(l)-1 \in FLOOR$
- ▶ inst $[lift(l)-1/f]$ in *axiom1*

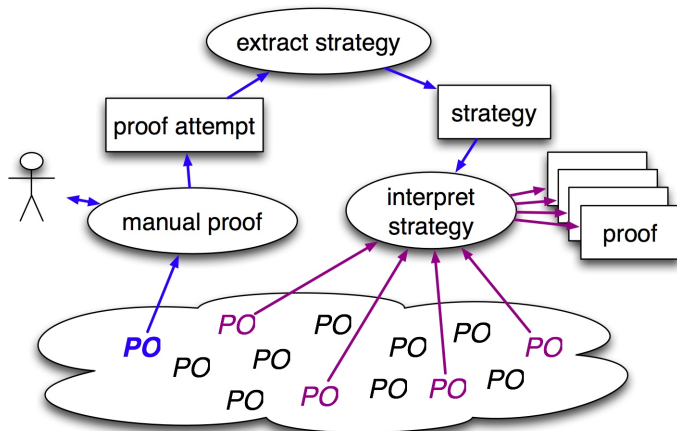
“Re-use” of proof: properties

- ▶ Not striving for automatic detection of similar “claims”
 - ▶ differs from analogy-based reasoning.
- ▶ Not expecting a fixed set of “patterns”
 - ▶ the user must still provide an exemplar proof.
- ▶ Low-level LCF tactics are too brittle
 - ▶ behaves differently for similar POs
 - ▶ not sufficiently robust to cope with changes
 - ▶ something higher-level is required...
 - ▶ ...as in proof planning...
 - ▶ however, we need to go beyond that...

Towards a strategy language

- ▶ We will explore additional information from the exemplar proof.
- ▶ Classification is a key challenge
 - ▶ as in exemplar-based reasoning
 - ▶ however, user-supplied exemplar proof required.
- ▶ Classification of proofs
 - ▶ data structures
 - ▶ shape of PO
 - ▶ information from PO Generator
 - ▶ e.g. refinement proof, invariance proof, ...
- ▶ A combination of a high-level proof strategy
- ▶ ... with a “vocabulary” of terms
 - ▶ instantiated using data structure information.

Illustration of strategy language



Evidence for strategy language: rippling

- ▶ **Rippling** provides evidence for such a high-level strategy language.
- ▶ Rippling is a **proof plan**
 - ▶ AI technique for mechanising formal reasoning based upon high level proof patterns
- ▶ 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 a starting point for the strategy language.

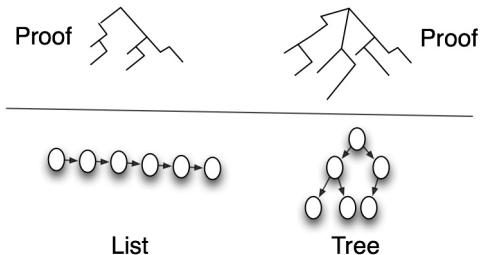
Strategy language properties (1)

- ▶ “Standard” proof plans and patches
 - ▶ exemplar proof will typically exhibit a new plan or a patch to an existing plan
 - ▶ we also aim to capture “dead ends” in proof attempts and expert recovery.
- ▶ Choices of unusual induction rules and variables
 - ▶ choice of an alternative induction rule/variable as a patch.
- ▶ Intermediate lemmas
 - ▶ construct and prove key intermediate lemmas.
- ▶ Generalisation
 - ▶ generalise the original PO as a patch.
- ▶ Insert case-analysis.

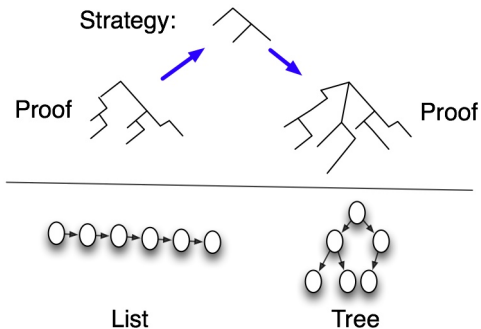
Strategy language properties (2)

- ▶ “Similar proofs” will deviate more than patterns currently captured in proof planning & rippling
 - ▶ higher level of abstraction.
- ▶ We expect to use additional kinds of abstraction
 - ▶ e.g. explore various dimensions of information
 - ▶ data structure, form, POG, ...
- ▶ The strategy language must be sufficiently **expressive, robust** and **general-purpose**
 - ▶ it must be able to deal with unanticipated proof plans and patches.
- ▶ We will use generic taxonomies
 - ▶ type of induction rule
 - ▶ type of generalisation
 - ▶ ...

“Re-use” of proof revisited



“Re-use” of proof revisited



A sequence of lemmas

- ▶ A strategy may describe the sequence of intermediate lemmas required to be proved:

prove lemma L_1

then prove lemma L_2

⋮

then prove lemma L_n

then GOAL follows from $L_1 \cdots L_n$

- ▶ Similar to the “ACL2 approach”
 - ▶ experiences from the Piton project² confirms to our hypothesis
- ▶ At a “reasonable” level, discharge using
 - ▶ e.g. a (correct) decision procedure
 - ▶ with e.g. a time-out feature.
- ▶ Requires a notion of a **gap**
 - ▶ an unproved sub-goal/lemma.

²see J S. Moore. *Piton: A Mechanically Verified Assembly-Level Language*.

Our plan

- ▶ We plan to analyse a lot of
 - ▶ POs,
 - ▶ their expert-provided proofs attempts,
 - ▶ and their “families”.
- ▶ Based on analysis develop a strategy language.
- ▶ Provide tool support
 - ▶ to extract a strategy out of an exemplar proof
 - ▶ to interpret strategies to discharge “similar POs”.
- ▶ There are a lot of challenges
 - ▶ classification of POs
 - ▶ creating a strategy language
 - ▶ deriving strategies from proofs
 - ▶ interpreting strategies for the same “family”.

Conclusion

- ▶ Significant proof automation can be achieved by learning from one proof to discharge “similar” POs.
- ▶ The use of such proof strategy approach is robust to model changes.
- ▶ We believe this is applicable to most model-based top-down formal development methods
 - ▶ ... and possibly to bottom-up approaches as well.
- ▶ There is a strong industrial & academic need for such tool
 - ▶ e.g. Systemel, SAP, ACL2 (Piton), ...