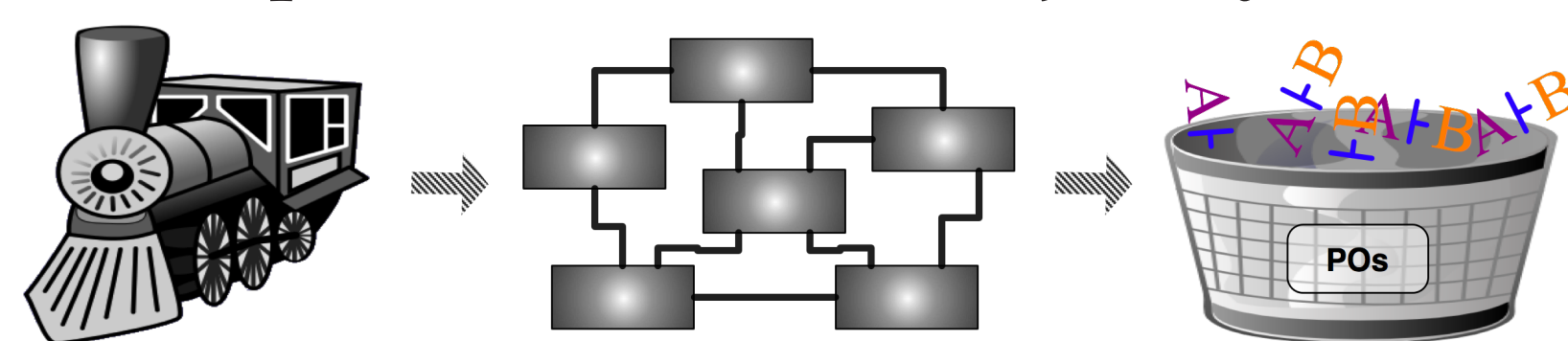


Background & motivation

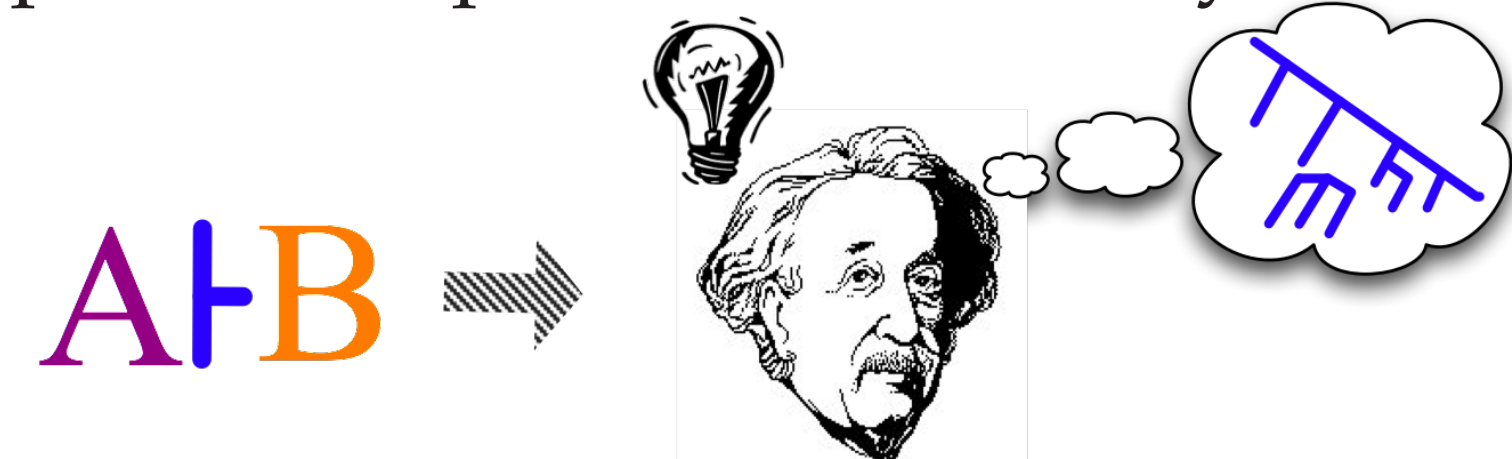
As a mathematical approach, formal methods (FMs) can lead to reliable and dependable systems. The use of FMs has increased not only in safety critical area, but also in other sectors. e.g. Microsoft has used FMs to verify device drivers. FMs can be applied in either bottom-up approach or top-down approach:

- bottom-up approach: is usually applied to verify or debug existing products or legacy systems.
- top-down approach: is applied in development stage, and normally results in source code. It tends to be posit-and-prove, where a developer posits a step of development, then tries to justify it.



The step of justifications in top-down can generate POs, which require proofs. Most of the POs are discharged automatically by the theorem provers, however some of them (typically 5 – 20%) still require human interactions. There are two strategies to discharge them, the modelling strategy and the proof strategy.

- the modelling strategy: change the model to simplify the POs that can be discharged by theorem prover automatically.
- the proof strategy: accept the challenge of proof, and prove them directly.



The proof strategy is what we're interested in this project. Discharging the remaining POs can be an expensive process, because it requires experiences from expert users. Thus it is becoming a bottleneck in industrial applications of formal methods.

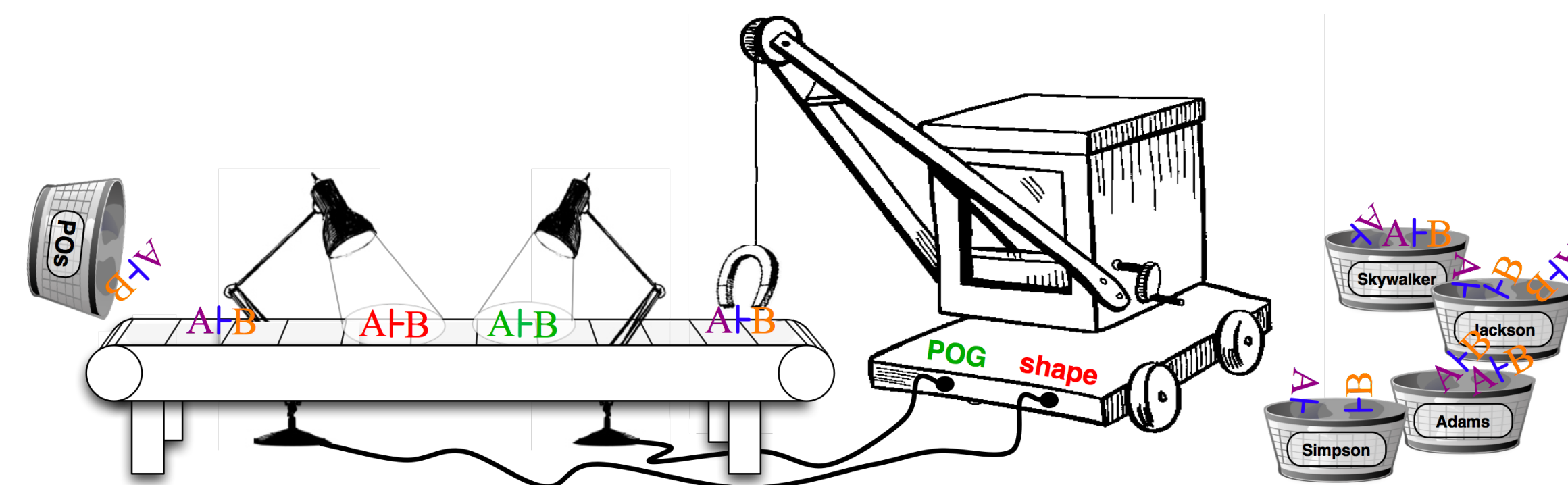
However, we observe that many other POs can be discharged by following the same proof pattern. The goal of this project is to automatically discharge POs of "similar form" by exploring the information from failures and patches.

Our approach

To illustrate the "similarity" in FM PO proof: an industrial case-study using Event-B generated 300 POs, 100 of these required user-interaction, however they could be handled by 5 strategies.

Therefore, we hope to explore the notion of proof families. We believe that the notion of failures and heuristics can be explored to guide the proof further.

Research hypothesis: *We believe that it is possible to classify FM PO proofs into families by analysing the context of failures. In each family there exist patterns of proof strategies which can be used to guide the proof search of the blocked proofs.*



An example of similarity in POs

This example comes from a telephone exchange specification in Z by Jim Woodcock. The (main) state involved in this example are:

Exchange
 $Free, Unavailable, Callers : \mathbb{P} Subs$
 $call : Subs \rightarrow SubRec$
 $connected : Subs \leftrightarrow Subs$
 $\langle Free, Unavailable, dom\ call \cup ran\ connected \rangle$
 $partition\ Subs$
 $Callers = dom((call \ ;\ st) \triangleright Connected)$
 $connected = Callers \triangleleft (call \ ;\ num)$

An operation "LiftFree" are described as below:

$s? \in Free$
 $call' = call \cup \{(s? \mapsto (seize, \langle \rangle))\}$

We want to illustrate two POs in this operation. They are generated to guarantee that the invariants in the states can still hold, after the change of operation in "LiftFree" has been made, e.g. $connected = Callers \triangleleft (call \ ;\ num)$

PO1: $Callers = dom((call \cup \{(s? \mapsto (seize, \langle \rangle))\} \ ;\ st)) \triangleright Connected$
 PO2: $connected = Callers \triangleleft ((call \cup \{(s? \mapsto (seize, \langle \rangle))\} \ ;\ num)$

Proofs of PO1: From $s? \in Free$ have $s? \notin dom(call)$
 Hence: $Callers = dom((call \ ;\ st) \cup (\{(s? \mapsto (seize, \langle \rangle))\} \ ;\ st)) \triangleright Connected$
 Hence: $Callers = dom(((call \ ;\ st) \triangleright Connected) \cup (\{(s? \mapsto (seize, \langle \rangle))\} \ ;\ st) \triangleright Connected)$
 Hence: $Callers = dom(((call \ ;\ st) \triangleright Connected) \cup \emptyset)$ Done!

Proofs of PO2: From $s? \in Free$ have $s? \notin dom(call)$
 Hence: $connected = Callers \triangleleft ((call \cup \{(s? \mapsto (seize, \langle \rangle))\} \ ;\ num)$
 Hence: $connected = Callers \triangleleft ((call \ ;\ num) \cup (\{(s? \mapsto (seize, \langle \rangle))\} \ ;\ num))$
 Hence: $connected = (Callers \triangleleft (call \ ;\ num)) \cup (Callers \triangleleft (\{(s? \mapsto (seize, \langle \rangle))\} \ ;\ num))$
 Simplify $(Callers \triangleleft (\{(s? \mapsto (seize, \langle \rangle))\} \ ;\ num))$ to $Callers \triangleleft \{s? \mapsto \langle \rangle\}$. Then with $s? \notin dom(call)$, it can be further simplified to \emptyset Done!

These two POs can be discharged by following one strategy, which is to "push" changes to the outer level, until it can be showed that the changes are empty.

Rippling and proof critics

Our work is inspired by previous work on *rippling* [?], which is a rewriting technique that works when the goal is embedded in one of the givens. A technique which uses failures productively in rippling is *Proof critics*. It can suggest patches to unblock proofs for which fails in rippling. Rippling is applicable to certain types of POS, those where a system invariant has to be preserved over the operations. In [?] we report on a small experiment where rippling is used for such POs arising from an Event-B model. A few observations can be made there to illustrate our hypothesis:

- each rippling step has to reduce some measures (to ensure termination) and to preserve what is known as the skeleton. Some steps were not measure reducing while preserving the skeleton. One patch which would have solved this was to introduce a secondary measure on the symbols.
- most rewrite rules were conditional, hence when rippling terminated; the conditions (which were not rippling goals) had to be discharged. Separate patches had to be developed for these.

Resources

This work has been supported by EPSRC grants EP/H024204/1

AI4FM: using AI to aid automation of proof search in Formal Methods. For more information on AI4FM. See www.ai4fm.org

References

[1] 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.

[2] G. Grov, A. Bundy and L. Dixon. A small experiment in Event-B rippling. In proc. of AV-oCS 2010 and Rodin User and Developer Workshop 2010.