The Use of Rippling to Automate Event-B Invariant Preservation Proofs

Yuhui Lin
University of Edinburgh, UK
Y.H.Lin-2@sms.ed.ac.uk

Alan Bundy
University of Edinburgh, UK
a.bundy@ed.ac.uk

Gudmund Grov
University of Edinburgh, UK
ggrov@inf.ed.ac.uk

Abstract

Proof automation is a common bottleneck for industrial adoption of formal methods. In Event-B, a significant proportion of proof obligations which require human interaction fall into a family called *invariant preservation*. In this paper we show that a rewriting technique, called rippling, does increase the automation of proofs in this family. We then extend this technique by combining two existing approaches. An earlier version of this paper has previously been published in [8]

1 Introduction

Event-B [2] is a “top-down” formal modelling method, which captures requirements in an abstract formal specification, and then stepwise refines the specification into the final product. Proof obligations (POs) are raised to ensure correctness of each development step. Most of these POs can be discharged automatically by the Rodin platform [3], yet still 3% to 10% of them require human interaction [6]. In an industrial sized project, this proportion of interactive proofs can be thousands, e.g. 43,610 POs with 3.3% interactive proof in the Roissy Airport Shuttle project [1]. One type of PO, called *invariant preservation (INV) POs*, can account for a significant proportion of the POs needing interaction. To illustrate, 188 out of 317 (59%) the undischarged POs in the BepiColombo case study[1] are INV POs. Moreover, specifications often change frequently, which requires users to reprove previously proven POs. Here we argue that part of the problem is a lack of meta-level reasoning, since Rodin provers only work at the object level logic. Here, we propose to use a meta-level reasoning technique, called *rippling* [5], for INV POs. Our hypothesis is

> *By utilising rippling we can increase the automation of Event-B invariant preservation proof obligations and make proofs more robust to changes.*

2 INV POs and Rippling

Machines are key components of Event-B specifications. A machine contains variables, invariants and events. Variables represent the states of the machine, and invariants describe

```
any
  s
where
  $s \in \mathit{Subs} \land s \notin \mathrm{dom}\,\mathit{call}$
then
  $\mathit{call} := \mathit{call} \cup \{(s \mapsto (\mathit{seize} \mapsto \varnothing))\}$
end
```

Notation	Definition
$x \mapsto y$	denotes the pair (x, y)
$\mathrm{dom}(r)$	$\{x \mid \exists y. x \mapsto y \in r\}$
$r \rhd s$	$\{x \mapsto y \mid x \mapsto y \in r \land y \in s\}$
$r\,;s$	$\{x \mapsto y \mid \exists z. x \mapsto z \in r \land z \mapsto y \in s\}$

Figure 1: An example of events & mathematical notations

[1]The case study can be found in `http://deploy-eprints.ecs.soton.ac.uk/136/`

constraints on the states. INV POs are generated to guarantee that the invariants hold in all states: they hold initially; and each event preserves them. To illustrate, consider the invariant

$$Callers = \mathrm{dom}((call \,;\, st) \rhd Connected) \tag{1}$$

in which *Callers*, *call*, *st* and *Connected* are variables. Figure 1 defines an event in which s is an argument followed by a *guarded action*, describing how the state changes. An INV PO (2) is generated to ensure (1) holds under changes made in the event

$$\begin{aligned} s \in Subs, s \notin \mathrm{dom}\, call, Callers = \mathrm{dom}((call \,;\, st) \rhd Connected) \vdash \\ Callers = \mathrm{dom}((call \cup \{(s \mapsto (seize \mapsto \varnothing))\} \,;\, st) \rhd Connected). \end{aligned} \tag{2}$$

We have observed that INV POs follow a pattern of $f(x) \vdash f(g(x))$, that is, the invariant $f(x)$ is embedded in the INV PO $f(g(x))$. This feature makes rippling applicable. Rippling is a rewriting technique which is applicable in any scenario where one of the assumptions can be embedded in the goal. In the case of INV POs, the invariant is embedded in the goal. The pattern showed above can be annotated as $f(x) \vdash f(\boxed{g(\underline{x})})$. $f(x)$ is the hypothesis. The embedding in the goal is called the *skeleton* (i.e. the non-boxed part), while the differences are the *wave-front*, and the *underline* annotates the skeleton insides (but not belongs to) the wave-front. When applying a rewrite rule, the skeleton in a goal should be preserved, and a *ripple measure* must decrease, e.g. skeletons which are separated by wave-front are moving together. To illustrate, by applying a rewrite rule $f(g(x)) = h(f(x))$, the annotation of the pattern would be $f(x) \vdash \boxed{h(\underline{f(x)})}$, where the hypothesis $f(x)$ is now a subterm of the goal.

3 IsaScheme and lemma conjecture

The key advantage of rippling is that the strong expectation of how the proof should succeed can help us to build a proof patching mechanism when a proof is blocked, e.g. due to a missing lemma. It can contribute to proof automation and makes proofs more robust to changes. This mechanism is known as *proof critics* [4]. One useful critic is *lemma speculation* [4] which is applicable when proofs are blocked due to a missing lemma. Meta-level annotations are used to guide and construct the lemma being conjectured. Consider the following blocked rippling proof:

$$Callers = \mathrm{dom}((\boxed{\underline{call} \cup \{(s \mapsto (seize \mapsto \varnothing))\}} \,;\, st) \rhd Connected)$$

we construct the left hand side of the missing lemma with one of our wave-fronts and parts of skeletons, which is $\boxed{\underline{call} \cup \{(s \mapsto (seize \mapsto \varnothing))\}} \,;\, st$. Because the skeleton needs to be preserved, we can construct the right hand side of the lemma by introducing a meta-variable $?F_1$ to represent the unknown part. Then we have

$$\boxed{\underline{call} \cup \{(s \mapsto (seize \mapsto \varnothing))\}} \,;\, st = \boxed{?F_1 \ \underline{call \,;\, st} \ \{(s \mapsto (seize \mapsto \varnothing))\} \ st}$$

In middle-out reasoning [7], this meta-variable is stepwise instantiated by unification during the proof. However, higher-order unification brings a challenge for this approach. Therefore, we propose a new approach by using IsaScheme[9] which is a scheme-based approach to instantiate these meta-variables, to generate the missing lemmas. Given a scheme and candidate terms and operations, IsaScheme can instantiate the meta-variables and construct and prove lemmas. The scheme will help constrain the lemmas generated, and we can further filter out those that will not provide valid ripple steps. The following algorithm shows more details about how to construct a scheme and get the potential lemmas with the example showed in this section (To make it more readable, we represent $\{(s \mapsto (seize \mapsto \varnothing))\}$ as X).

When no rewriting rules are applicable, the following process can be triggered.

1. Construct the lhs of the scheme with a wave-front and part of skeletons, i.e. $\boxed{(\underline{call} \cup X)}\ ; st$
2. Utilising rippling properties, we can partially predict how the term evolves on the rhs. i.e. evolve from $\boxed{(\underline{call} \cup X)}\ ; st$ to $\boxed{\underline{call\ ; st}...}$. Also we need to construct a term with meta-variables to specify the new shape of combination of the constants and variables in the wave-fronts, i.e. X, and those next to the wave-fronts in the skeleton, i.e. st. In our example this term would be $(?F_2\ X\ st)$. Now we introduce meta-variables to combine these terms to compose the rhs of the missing lemma, i.e. $... = \boxed{?F_1 \underline{(call\ ; st)} (?F_2\ X\ st)}$.
3. Now we have a scheme to instantiate. i.e.
   $Myscheme\ ?F_1\ ?F_2 \equiv (call \cup X)\ ;\ st = ?F_1\ (call\ ;\ st)\ (?F_2\ X\ st)$
   in which *Myscheme* is the name of our scheme; $?F_1$ and $?F_2$ are meta-variables to be instantiated from a set of given terms. Then we try this scheme in IsaScheme with relevant proof context, including assumptions.
4. IsaScheme returns potential lemmas which we can apply to unblock the current proof. In our example, we get the following lemma, i.e. $(call \cup X)\ ; st = (call\ ; st) \cup (X\ ; st)$, which helps to proceed with the proof.

4 Conclusion & Further Work

We have showed that rippling is applicable to prove INV POs. We have combined lemma speculation and scheme-based theory exploration to the discovery of missing lemmas when proofs are blocked. This has to be done manually in Rodin. Moreover, with meta-level reasoning and its patching mechanism, the robustness of proofs can be improved, as the proof strategy remains the same even if the POs are required to be re-proven when specifications change. Currently these schemes are deduced manually, and next we plan to automate this process.

Acknowledgement: This work is supported by EPSRC grant EP/H024204/1 (AI4FM). Thanks to Omar Montano Rivas, Andrew Ireland, Moa Johansson and the AI4FM partners for useful discussions.

References

[1] J.R. Abrial. Formal methods in industry: achievements, problems, future. In *Proceedings of the 28th international conference on Software engineering*, pages 761–768. ACM, 2006.

[2] J.R. Abrial. *Modeling in Event-B System and Software Engineering.* Cambridge Univ Pr, 2010.

[3] J.R. Abrial, M. J. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in event-B. *STTT*, 12(6):447–466, 2010.

[4] A.Ireland. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, March 1996.

[5] A. Bundy. *Rippling: meta-level guidance for mathematical reasoning*, volume 56. Cambridge Univ Pr, 2005.

[6] C. B. Jones, G. Grov, and A. Bundy. Ideas for a high-level proof strategy language. Technical Report CS-TR-1210, School of Computing Science, Newcastle Univ, 2010.

[7] I. Kraan, D. Basin, and A. Bundy. Middle-out reasoning for synthesis and induction. *Journal of Automated Reasoning*, 16(1):113–145, 1996.

[8] Y. Lin, A. Bundy, and G. Grov. The use of rippling to automate event-b invariant preservation proofs. *NASA Formal Methods*, pages 231–236, 2012.

[9] O. Montano-Rivas, R. McCasland, L. Dixon, and A. Bundy. Scheme-based synthesis of inductive theories. *Advances in Artificial Intelligence*, pages 348–361, 2010.