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

Yuhui Lin, Alan Bundy, and Gudmund Grov

School of Informatics, University of Edinburgh, 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 can increase the automation of proofs in this family, and extend this technique by combining two existing approaches.

Keywords: Event-B, automated reasoning, rippling, lemma conjecture

1 Introduction

Event-B [2] is a "top-down" formal modelling method, which captures requirements into an abstract formal specification and then stepwise refines the specification into the final product. In each step of development, designers justify the correctness of the specification by proving proof obligations (POs). The generation of the POs is automated by the Rodin platform [3] which also contains support for both automatic and interactive theorem proving. Most of these POs can be discharged automatically, yet still 3% to 10% of them require human interaction [8]. In an industrial sized project, this proportion of interactive proofs can be thousands. For example, 43,610 POs with 3.3% interactive proof in the Roissy Airport Shuttle project, and 27, 800 POs with 8.1% interactive proof in the Paris Metro line 14 project [1]. Moreover, specifications often change frequently, which require users to reprove previous proven POs.

Invariant preservation (INV) POs is a family of Event-B POs which can account for a significant part of all of the POs that require human interaction. To illustrate, 188 out of 317 (59%) undischarged POs in the BepiColombo case study[1] belong to the INV family.

Here we argue that part of the problem is a lack of meta-level reasoning since Rodin provers only work on the object level logic. In [5] it is argued that, to achieve a better understanding of reasoning, both logic and a meta-level understanding should be put into consideration. In this paper we propose to use a meta-level reasoning technique, called *rippling* [6] for Event-B POs. It is a rewriting technique which can be applied when the goal embeds in one of the hypothesis, and details are given in Sect. 2.2. Our hypothesis is

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

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

The contributions of this paper are two-fold: (1) In Sect. 2 we illustrate the use of rippling for INV POs; (2) The key advantage of rippling is that, due to meta-level reasoning, we can often patch a broken proof, thus making proofs more robust to changes. In Sect. 3 we describe a novel combination of two existing techniques, lemma speculation and scheme-based theory exploration, to conjecture lemmas when proofs are blocked by the absence of lemmas. Finally, we describe further work and conclude in Sect. 4 and Sect. 5, respectively.

2 INV POs and Rippling

2.1 INV POs

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

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

Fig. 1. An example of events & mathematical notions

Machines are key components of Event-B specifications. A machine contains variables, invariant and events. Variables represent the states of the machine, and invariants describe constraints on the states. INV POs are generated to guarantee that the invariants are still preserved under changes made by the events.

To illustrate, let us consider the invariant (1)

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

in which $Callers$, $call$, st and $Connected$ are variables. Figure 1 defines an event in which s is an argument followed by a *guard* and an *action*, describing how the state changes.

An INV PO (2) is generated to ensure (1) holds under changes made in the event.

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

2.2 Rippling

Rippling[2] is a rewriting technique which was developed originally for inductive proofs. Although it was designed to guide the step cases of inductive proofs, it 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, thus making rippling applicable. The embedding in the goal is called the *skeleton*, while the differences are the *wave-front*.

To illustrate, let us recall the INV PO in Sect. 2.1. The annotated version of the goal (2) with the embedding (1), becomes:

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

in which the wave-fronts are shaded by a box. When applying a rewrite rule, the skeleton (i.e. the non-shaded part) in a goal should be preserved, and a *ripple measure* must decrease, e.g. skeletons which are separated by wave-front are moving together. Therefore, by applying

$$(f \cup g) \; ; S = (f \; ; S) \cup (g \; ; S)$$

we have

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

With the following two rules

$$(f \cup g) \rhd S = (f \rhd S) \cup (g \rhd S)$$
$$\mathrm{dom}(f \cup g) = \mathrm{dom}\, f \cup \mathrm{dom}\, g$$

we have

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

and finally get the INV PO to

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

Then the hypothesis (1) can be applied to simplify the proof by substituting $\mathrm{dom}((call \; ; st) \rhd Connected)$ with $Callers$, which results in

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

[2] Rippling has been implemented in *IsaPlanner* which we use to start our experiments. For more details about IsaPlanner, please refer to `http://dream.inf.ed.ac.uk/projects/isaplanner/`

This step of using the hypothesis is called *fertilisation*. Now as st projects the second first argument (e.g. $(x \mapsto (y \mapsto z))$ into $(x \mapsto y)$) and $Connected = \{ringing, speech, suspended\}$, $\mathrm{dom}((\{(s \mapsto (seize, \varnothing))\} ; st) \rhd Connected)$ can be trivially simplified into $\varnothing$ by using the assumptions in (2). This completes the proof.

The advantage of rippling is the meta-level guidance which can be used to guide the application of rewrite rules. Due to its lack of meta-level guidance, many rules[3] can only be applied manually in Rodin. Let us consider the following distribution rule:

$$(p \cup q) \lhd r = (p \lhd r) \cup (q \lhd r)$$

We can generate two rewrite rules from the left hand side (lhs) to the right hand side (rhs) and the other way round. Rodin does not add both of them into its rewriting system, because the rewriting may not terminate. However, due to the meta-level guidance of rippling, both directions can be added in rippling whilst guaranteeing termination. For example:

$$From\_lhs\_to\_rhs : (p \cup q) \lhd r \Rightarrow (p \lhd r) \cup (q \lhd r)$$

$$From\_rhs\_to\_lhs : (p \lhd r) \cup (q \lhd r) \Rightarrow (p \cup q) \lhd r$$

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 change. This mechanism is known as proof critics [7]. One useful critic in our case is lemma speculation [7]. It's 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:

$$Callers = dom((call \lhd x) ; st \rhd Connected)$$

we construct the left hand side of the missing lemma with one of our wave-fronts and parts of skeletons, which is $(call \lhd x) ; st$. With the skeleton preservation rule, 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

$$(call \lhd x) ; st = ?F_1\, call ; st\ x\ st$$

[3] For more rules, please refer to
`http://wiki.event-b.org/index.php/All_Rewrite_Rules`

This meta-variable is stepwise instantiated by unification during the proof. This approach is called middle-out reasoning [9]. However, higher-order unification brings a challenge for this approach.

Therefore, instead of using middle-out reasoning, we propose a new approach by using IsaScheme[10] 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 return lemmas that pass a counter-example checker. 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 the beginning of this section.

Precondition: When no rewriting rules are applicable and fertilisation can not be applied, the following process can be triggered.

1. Construct the lhs of the scheme with a wave-front and part of skeletons, i.e. $(call \lhd x)\ ; st = ...$
2. Since the skeleton has to be preserved and a ripple measure must decrease, we can partially predict how the term evolves on the rhs.
   i.e. evolve from $(call \lhd x)\ ; st$ to $call\,;st \lhd ...$
   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.
   $... = ?F_1\,(call\,;st)\,(?F_2\ x\ st)$.
3. Now we have a scheme to instantiate. i.e.

   $$Myscheme\ ?F_1\ ?F_2 \equiv (call \lhd r)\ ;\ st = ?F_1\ (call\ ;\ st)\ (?F_2\ r\ st)$$

   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 which can help to proceed the proof[4]

   $$(call \lhd r)\,;st = (call\,;st) \lhd (r\,;st)$$

4 Further work

These schemes are currently deduced manually, and next we plan to automate this process. Moreover, we observe that many POs contain quantifiers and are

[4] This lemma relies on some properties of the specification

conditional. Rippling is not particularly suited for such POs, and to handle this, we are currently exploring integrating a technique called *piecewise fertilisation* [4] with rippling. Longer term we will develop a rippling plug-in for Rodin, which will be based on the existing Rodin to Isabelle translator by Schmalz[5].

5 Conclusion

We have showed that the use of rippling can improve the automation of proofs in Event-B INV POs and make it more robust to changes. We have combined lemma speculation and scheme-based theory exploration to discovery the missing lemma 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.

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 the useful discussions.

References

1. Abrial, J.R.: Formal methods in industry: achievements, problems, future. In: Proceedings of the 28th international conference on Software engineering. pp. 761–768. ACM (2006)
2. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
3. Abrial, J.R., Butler, M.J., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in event-B. STTT 12(6), 447–466 (2010)
4. Armando, A., Smaill, A., Green, I.: Automatic synthesis of recursive programs: The proof-planning paradigm. Autom. Softw. Eng 6(4), 329–356 (1999)
5. Bundy, A.: A science of reasoning. In: 10th International Conference on Automated Deduction. pp. 633–640. Springer (1990)
6. Bundy, A.: Rippling: meta-level guidance for mathematical reasoning, vol. 56. Cambridge Univ Pr (2005)
7. Ireland, A.: Productive use of failure in inductive proof. Journal of Automated Reasoning 16(1–2), 79–111 (Mar 1996)
8. Jones, C.B., Grov, G., Bundy, A.: Ideas for a high-level proof strategy language. Tech. Rep. CS-TR-1210, School of Computing Science, Newcastle University (2010)
9. Kraan, I., Basin, D., Bundy, A.: Middle-out reasoning for synthesis and induction. Journal of Automated Reasoning 16(1), 113–145 (1996)
10. Montano-Rivas, O., McCasland, R.L., Dixon, L., Bundy, A.: Scheme-based synthesis of inductive theories. In MICAI, LNCS vol. 6437, pp. 348–361. Springer (2010)

[5] See `http://wiki.event-b.org/index.php/Export_to_Isabelle` for details.