On an Extensible Rule-based Prover for Event-B

Issam Maamria, Michael Butler, Andrew Edmunds, and Abdolbaghi Rezazadeh

ECS, University of Southampton, Southampton SO17 1BJ, UK
{im06r, mjb, ae2, ra3}@ecs.soton.ac.uk

1 Motivation

The Rodin platform [3] provides the practical setting to carry out modelling in Event-B. It seamlessly integrates modelling and proving, and provides an extensible and configurable mechanism that can be adapted to different application domains and development methods [1]. The Rodin platform provides a proving infrastructure that has certain limitations. Extending the prover with proof rules (rewrite and inference rules) requires a certain level of competence using the Java programming language as well as good knowledge of the toolset's internal architecture. A further complication of this approach is that it became non-trivial to verify the soundness of the prover after adding new rules. This paper presents our approach when dealing with prover extensibility and soundness in the context of Event-B.

2 The Theory Construct

Theories will provide a mechanism by which the user can extend the proof capabilities of the Rodin platform by specifying *rewrite* rules. Proof obligations will be generated to verify the soundness of the prover augmented with the new rules. In essence, the theory construct will allow a degree of *meta-reasoning* to be carried out within the same platform in a similar fashion to Event-B reasoning. Figure 1 outlines the structure of the theory construct. In what follows, we briefly describe each of the elements of the theory construct:

1. *Sets*. A theory can define a number of given sets on which it is parametrised.
2. *Metavariables*. A theory can define a number of metavariables each of which has a type.
3. *Rewrite Rules*. Rewrite rules are one-directional equations that can be used to rewrite formulas to equivalent forms. As part of specifying a rewrite rule, the *theory developer* decides whether the rule can be applied automatically without user intervention or interactively following a user request.

3 Rewrite Rules

A rewrite rule defines how a formula lhs may be rewritten to one of several formulae rhs_i provided condition C_i holds. The following two definitions formalise the concept of rewrite rules within theories. Note the use of the well-definedness operator $\mathcal{D}$ [2].

Theory theory_name

Sets $s_1, s_2, ...$

Metavariables $v_1, v_2, ...$

Rewrite Rules $r_1, r_2, ...$

End

Fig. 1. The Theory Construct

Definition 1 (Rewrite Rule). *A rewrite rule is of the form*

$$\begin{aligned} lhs \rightarrow\ & C_1 : rhs_i \\ & ... \\ & C_n : rhs_n \end{aligned}$$

where:

1. $n \geq 1$,
2. lhs *is not a meta-variable but may contain metavariables,*
3. lhs *and* rhs_i *(for all* i *such that* $1 \leq i \leq n$*) are formulas of the same syntactic class i.e., both expressions or both predicates,*
4. C_i *(for all* i *such that* $1 \leq i \leq n$*) are predicates,*
5. C_i *and* rhs_i *(for all* i *such that* $1 \leq i \leq n$*) only contain free variables from* lhs*,*
6. lhs *and* rhs_i *(for all* i *such that* $1 \leq i \leq n$*) have the same type if* lhs *is an expression.*

Note. In this paper, we only consider rewrite rules whose left hand side is a basic predicate (e.g., $\subseteq$) or is an expression not involving binding. More generally, we do not consider rules that require side conditions (i.e., non-freeness conditions).

Definition 2 (Sound Rewrite Rule). *A rewrite rule*

$$\begin{aligned} lhs \rightarrow\ & C_1 : rhs_i \\ & ... \\ & C_n : rhs_n \end{aligned}$$

is said to be sound if the following sequents are valid:

1. $H, \mathcal{D}(lhs) \vdash \mathcal{D}(C_i)$ *for all* i *such that* $1 \leq i \leq n$,
2. $H, \mathcal{D}(lhs), C_i \vdash \mathcal{D}(rhs_i)$ *for all* i *such that* $1 \leq i \leq n$,

3. (a) $H, \mathcal{D}(lhs), C_i \vdash lhs = rhs_i$ *for all* i *such that* $1 \leq i \leq n$ *if* lhs *is an expression, or;*
   (b) $H, \mathcal{D}(lhs), C_i \vdash lhs \Leftrightarrow rhs_i$ *for all* i *such that* $1 \leq i \leq n$ *if* lhs *is a predicate,*

where H *is a predicate providing typing information for all free variables occurring in* lhs*.*

The previous definition ensures that rewrite rules are both *validity-preserving* and *WD-preserving*.

4 The Theory Prototype Plug-in

A theory prototype plug-in has been developed as an extension to the Rodin platform. The plug-in offers the following capabilities:

1. Users can develop and validate theories in the same way as contexts and machines.
2. Users can deploy theories to a specific directory where they become available to the interactive and automatic provers of Rodin.
3. Users can use rewrite rules defined within the deployed theories as a part of the proving activity. A pattern matching mechanism is implemented to calculate applicable rewrite rules to any given sequent.

5 Further Work

This work can be extended to enable the specification and validation of inference rules within theories. It can also be extended by providing a clear set of guidelines to help the theory developer with deciding whether a rule can be applied automatically. Finally, the verification of the pattern matching mechanism will give more confidence in this approach.

References

1. J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An open extensible tool environment for Event-B. In *International Conference on Formal Engineering Methods (ICFEM)*, 2006.
2. Jean-Raymond Abrial and Louis Mussat. On using conditional definitions in formal theories. In *ZB '02: Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B*, pages 242–269, London, UK, 2002. Springer-Verlag.
3. Michael Butler and Stefan Hallerstede. The Rodin Formal Modelling Tool. *BCS-FACS Christmas 2007 Meeting - Formal Methods In Industry, London.*, December 2007.