

On an Extensible Rule-based Prover for Event-B

Issam Maamria¹, Michael Butler¹, Andrew Edmunds¹, and Abdolbaghi Rezazadeh¹

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

Abstract. Event-B is a formalism for discrete system modelling. Key features of Event-B include the use of set theory as a modelling notation, the use of refinement to model systems at different levels of abstraction, and the use of mathematical proof to verify consistency between refinement levels. The Rodin platform provides a toolset to carry out specification, refinement and proof in Event-B. The importance of the proving activity as a part of modelling cannot be emphasised enough, and as such, it is imperative to provide effective tool support for it. An important aspect of this support is the extensibility of the prover, and more pressingly, how its soundness is preserved while allowing extensibility. In this paper, we outline our approach when dealing with extensibility and soundness, in the process of designing and implementing a rule-based prover for Event-B.

1 Introduction

The complexity of systems can be tackled by means of abstraction and modelling. Reasoning provides a formal basis to understanding models, and as such, it should be regarded as an integral step in the modelling process. Consequently, any tool support for modelling should include an effective framework to facilitate the proving activity.

Event-B [1] is an evolution of the B-method [2]. It can be used for specifying and reasoning about complex systems such as concurrent and reactive systems. The semantics of the model is given by means of its proof obligations; these are used to show its consistency. In this paper, we show how an extensible rule-based prover for Event-B is designed where proof obligations play a pivotal role in ensuring that its soundness does not get compromised when adding new proof rules.

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 [4]. This work is focused on the proving infrastructure provided by Rodin. This is discussed in some detail in Section 2.

The main contribution of this paper is to address a certain limitation within the Event-B toolset regarding prover extensibility. At the time of writing this paper, 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. To support prover extensibility, we extend Event-B with a theory construct by which users may define proof rules. These rules become available in the automatic and interactive provers of Rodin. We define proof obligations for user-defined proof rules that ensure their soundness. A prototype plug-in has been developed.

Outline. Section 2 presents an overview of Event-B and its toolset. Emphasis is placed on the proving architecture of the Rodin platform and its limitations. Section 3 provides an overview of the theory construct. Next, Section 4 presents a treatment of term rewriting systems [5] within a partial setting where well-definedness of terms is a concern. Our aim in Section 4 is to provide justifications for the different definitions presented in Section 3. Section 5 provides an overview of the prototype plug-in that uses the different ideas presented in this paper. We conclude by briefly surveying work related to our research and discussing potential future work.

2 Event-B and the Rodin Platform

An Event-B model is described using contexts and machines. Contexts define the static part of a model. They may include carrier sets (which are assumed to be non-empty) and constants. They also include axioms that describe the properties of those sets and constants. Moreover, contexts may contain theorems for which proof obligations arise to prove that they follow from preceding axioms and theorems. Finally, contexts can be extended by other contexts and seen by machines.

Machines describe the behavioural aspects of a model. Each machine has a state defined by means of variables. Variables correspond to simple mathematical objects such as numbers, functions, etc. They are constrained by invariants $I(v)$ where v are the variables of the machine. Invariants hold in all reachable states of the machine. Theorems can be specified, and proof obligations arise to prove that they follow from the axioms of the seen contexts as well as the machine invariants.

A machine may specify a number of atomic events which define its possible state transitions. Each event is guarded and has an action. The guard describes the enabling condition under which the event may occur. The action determines how the state (i.e., the variables) of the machine evolves after the occurrence of the event. An event may be allowed to occur only if its guard holds. Consequently, when the guards of several events hold at the same time, one event is non-deterministically chosen to be performed. Proof obligations of a machine arise to verify its consistency.

Machine refinement provides a means to introduce more complexity to the behavioural properties of a model [1]. A machine CM can refine at most one

other machine AM , and the state of CM is linked to the state of AM by means of a gluing invariant $J(v, w)$ where v are the variables of AM and w the variables of CM . Proof obligations arise to verify that the refinement relationship between the two machine indeed holds.

Proof obligations generated for Event-B models are specified in typed set theory [2]. Since Event-B models may contain partial functions (and operators), well-definedness of terms is a concern. As a result, the sequent calculus used to carry out proofs is *well-definedness preserving*, and is the one appearing in [6] and is similar to the one developed in [7].

Our intention was to provide a brief introduction to Event-B. For a more detailed treatment, we refer to [1] and [8].

The Rodin platform [4] is an open extensible tool for Event-B based on Eclipse¹. It offers support for specification and proof, and it can be easily extended with other useful tools e.g., there is a plug-in for model checking called ProB². In what follows, we limit our discussion to the proving infrastructure of Rodin. For a detailed account of the overall architecture, we refer to [4].

The Proof Manager, as its name suggests, is in charge of maintaining proofs associated with proof obligations. For each proof obligation, it constructs a proof tree whose root is the sequent of the obligation itself. The proof manager works both automatically (without user intervention) and interactively (with user intervention and possibly with input).

Reasoners are proof rule schemas that can be used to generate concrete rules. An example rule schema is the following well-known \wedge intro rule:

$$\frac{\mathbf{H} \vdash P \quad \mathbf{H} \vdash Q}{\mathbf{H} \vdash P \wedge Q} \wedge \text{intro}$$

Concrete rules can be generated by appropriately instantiating the meta-variables \mathbf{H} , P and Q . Note that \mathbf{H} stands for a set of predicates (the set of hypotheses).

Proof Trees are recursive structures based on proof tree nodes. A proof tree node represents a single node as well as the proof tree (or sub-tree) rooted at that node. Each proof tree node has a sequent, a concrete proof rule and a list of child nodes. A proof tree node can be either:

1. *pending*, if its concrete rule is `null`. Consequently, the list of child nodes is `null`.
2. *non-pending*, if it has a `non-null` concrete rule, and the child nodes correspond to the result of applying the proof rule to its sequent.

Tactics provide a uniform mechanism to manipulate proof trees. A tactic can be a wrapper around a proof rule, in which case it is called a basic tactic. Tactical tactics, on the other hand, are more structured and can be used to specify a proof strategy [9]. An example is a tactic that repeats another tactic until it fails.

¹ <http://www.eclipse.org/>

² <http://www.stups.uni-duesseldorf.de/ProB/overview.php>

The proof manager can be extended with new reasoners (schema proof rules) and tactics. There is a well-defined protocol for both extensions. The reasoner contract is also used to integrate external provers. The idea is to encapsulate a call to the external prover as a reasoner application. The call is successful if the external prover discharges the sequent. One limitation is that information about how the external prover went about the proof (e.g., used hypotheses) is not always available to the proof manager.

Two external provers that have been successfully integrated are:

1. *The Predicate Prover (PP)*: this prover is built around a hierarchy of provers. It contains a decision procedure for propositional logic and a semi-decision procedure for first order logic [10]. Another major component is the translator from set theory to first order logic. It is built in accordance with the set-theoretic construction outlined in the B-Book [2].
2. *The ML Prover (ML)*: is a rule-based prover used in the Logic Solver which is the compiler-interpreter used for B. PP was originally developed to validate the many proof rules of ML. ML and PP are part of Atelier-B [11] which provides the proving infrastructure for B.

Despite being optimised for proof reuse [9], the current architecture has the following limitations:

- in order to add a new proof rule, it is required to implement a rule schema (i.e., a reasoner) and a wrapper tactic. Therefore, a certain level of competence with the Java programming language as well as knowledge of Rodin architecture are necessary;
- after a new rule is added, soundness of the prover augmented with the new rule has to be established. It is not clear how this can be achieved at the level of Java code.

3 The Theory Construct

Modelling in Event-B is carried out using two constructs: contexts and machines. As discussed in the last section, contexts can be used to describe static properties of a model (e.g., a constant), whereas machines are used to define dynamic behaviour. Following a similar approach, and in order to facilitate extending the prover with proof rules, we propose a third construct which we refer to as *theory*.

Theories will provide a mechanism by which the user can extend the proof capabilities of the Rodin platform by specifying *rewrite* and *inference* 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. In this paper, we only discuss the implications of adding new rewrite rules. Figure 1 outlines the structure of the theory construct. In what follows, we briefly describe each of the elements of the theory construct:

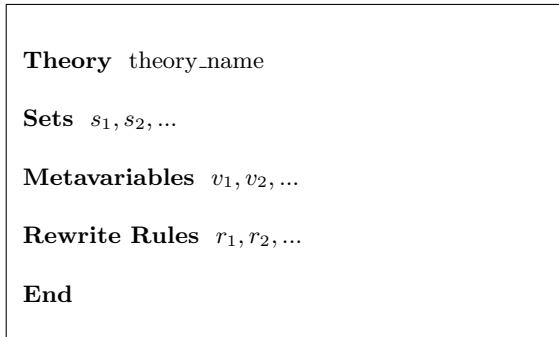


Fig. 1. The Theory Construct

1. *Sets*. A theory can define a number of given sets which have a similar purpose to carrier sets in contexts. These sets define the types on which the theory is parametrised.
2. *Metavariables*. A theory can define a number of metavariables that can be used to specify rewrite rules. Each metavariable is associated with a type; this can be constructed using the given sets of the theory as well as the built-in types (e.g., \mathbb{Z}) using type constructors. For example, if a given set S is defined within a theory, then $\mathbb{P}(\mathbb{Z}) \times S$ can be used as a type for a metavariable.
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. Rewrite rules are discussed further in the next subsection.

The theory construct can be extended to enable the specification of inference rules. In brief, it facilitates the following:

- specification of proof rules within the same platform providing a degree of meta-reasoning within Rodin,
- validation of specified proof rules to ensure that the soundness of the prover is not compromised.

3.1 Specifying Rewrite Rules

In the Event-B mathematical language (quite similar to the one in [2]), predicates and expressions are distinguished as two separate syntactic categories. Furthermore, each expression must have a type. Note that we use the word ‘formula’ to refer to both expressions and predicates, therefore, we say that two formulas are of the same *syntactic class* if they are both expressions or both predicates.

A rewrite rule defines how a formula *lhs* may be rewritten to one of several formulae *rhs_i* provided condition C_i holds. This translates to the following definition.

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

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

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,
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). For instance, we do not consider the following rule

$$\{x : S \mid x \in s\} \rightarrow s \text{ if } x \setminus s .$$

Verifying the validity of the previous rule demands sophisticated meta-reasoning not available within the Rodin platform at the time of writing this paper. Research, however, is being carried out to address this limitation.

Definition 1 describes the syntactic properties of a rewrite rule; these can be statically checked. Next, we describe two important properties of rewrite rules that require mathematical proof. In the following definition, we make use of the well-definedness operator \mathcal{D} which is discussed further in Section 4. We write $\mathcal{D}(f)$ for the condition under which formula f is well-defined.

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

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

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*. In Section 4, we formally justify the adequacy of the previous definition.

Definition 3 (Coverage-complete Rewrite Rule). A sound rewrite rule

$$\begin{aligned} lhs &\rightarrow C_1 : rhs_1 \\ &\dots \\ &C_n : rhs_n \end{aligned}$$

is said to be *coverage-complete* if the following sequent is valid:

$$H, \mathcal{D}(lhs) \vdash \bigvee_{i=1}^n C_i,$$

where H is a predicate providing typing information for all free variables of lhs .

A rewrite rule is said to be *unconditional* if it has one right hand side whose condition is \top . If a rewrite rule is not unconditional, it is *conditional*.

Example 1. Assuming a given set S and two metavariables E and F of the same type S , then the following is a rewrite rule

$$E \in \{F\} \rightarrow \top : E = F$$

which is sound and coverage-complete. This rule is unconditional.

Example 2. Assuming two metavariables x and y of the same type \mathbb{Z} , then the following is a rewrite rule

$$\begin{aligned} (x - 1)(y - 1) &\rightarrow x = 1 : 0 \\ &y = 1 : 0 \end{aligned}$$

which is sound but not coverage-complete. This rule is conditional.

Example 3. Assuming two metavariables a and b of type \mathbb{Z} , then the following is a rewrite rule (cardinality of a range of integers)

$$\begin{aligned} card(a..b) &\rightarrow a \leq b : b - a + 1 \\ &a > b : 0 \end{aligned}$$

which is sound and coverage-complete. This rule is conditional.

The theory construct described thus far offers a uniform mechanism to specify rewrite rules. Rewrite rules are specified in accordance with Definition 1. Theory developers can tag rules as automatic, manual or both. They can also tag rules as coverage-complete. Finally, the proof obligations associated with theories correspond to the sequents in Definition 2 and Definition 3.

4 Well-Definedness and Rewriting

In this section, we outline the treatment of well-definedness and how rewrite rules preserve well-definedness. We use the language signature Σ defined by a set V of variable symbols, a set F of operator symbols and a set P of total predicate symbols.

Definition 4 (Expression). T_Σ , the set of Σ -expressions is inductively defined by:

- each variable of V is an expression;
- if $f \in F$, $\text{arity}(f) = n$ and each of e_1, \dots, e_n is an expression, then $f(e_1, \dots, e_n)$ is an expression.

Definition 5 (Predicate). P_Σ , the set of Σ -predicates is inductively defined by:

- $p(e_1, \dots, e_n)$ is a predicate provided $p \in P$, $\text{arity}(p) = n$ and each of e_1, \dots, e_n is an expression;
- $e_1 = e_2$ is a predicate provided e_1 and e_2 are expressions;
- $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\varphi \Rightarrow \psi$ are predicates if φ and ψ are predicates;
- $\neg\varphi$ is a predicate if φ is a predicate;
- $\forall x.\varphi$ and $\exists x.\varphi$ are predicates if $x \in V$ and φ is a predicate;

We shall use the term ‘basic predicates’ to refer to predicates of the shape $p(e_1, \dots, e_n)$ and $e_1 = e_2$. Examples of basic predicates include $e \in S$ and $s_1 \subseteq s_2$.

Definition 6 (Formula). F_Σ , the set of Σ -formulas is defined as follows:

$$F_\Sigma = P_\Sigma \cup T_\Sigma .$$

We say that two formulas are of the same *syntactic class* if they are both expressions or both predicates.

4.1 The Well-Definedness Operator

The well-definedness operator ‘ \mathcal{D} ’ formally encodes what is meant by well-definedness. $\mathcal{D} : F_\Sigma \rightarrow P_\Sigma$ is a syntactic operator that maps formulas (both expressions and predicates) to their well-definedness predicates. We interpret the predicate $\mathcal{D}(F)$ as being valid if and only if F is well-defined. For a detailed treatment of the \mathcal{D} operator, we refer to [12].

The well-definedness (WD) of expressions is defined recursively as follows:

$$\begin{aligned} \mathcal{D}(x) &\hat{=} \top \quad \text{if } x \in V , \\ \mathcal{D}(f(t_1, \dots, t_n)) &\hat{=} \bigwedge_{i=1}^n \mathcal{D}(t_i) \wedge C_{t_1, \dots, t_n}^f . \end{aligned}$$

where C_{t_1, \dots, t_n}^f effectively defines the domain of the operator f . Since we ensure that predicates $p(t_1, \dots, t_n)$ (for $p \in P$ and expressions t_1, \dots, t_n) are total, ill-definedness can only be introduced by expressions. For the well-definedness of predicates, we use the various expansions present in [12].

4.2 WD-Preserving Conditional Term Rewriting

In what follows, we assume that the signature Σ is equipped with a proof theory in the shape of a WD-preserving sequent calculus similar to the one appearing in [9]. A judgement in such a calculus is of the shape $\Gamma \vdash_{\mathcal{D}} \Delta$ defined as follows:

$$\Gamma \vdash_{\mathcal{D}} \Delta \quad \hat{=} \quad \mathcal{D}(\Gamma), \mathcal{D}(\Delta), \Gamma \vdash \Delta .$$

That is, the well-definedness of Γ and Δ is assumed when proving $\Gamma \vdash \Delta$. In this subsection, we assume a syntactic operator $\mathcal{V}ar : T_{\Sigma} \rightarrow \mathbb{P}(V)$ such that $\mathcal{V}ar(t)$ is the set of free variables occurring in t . We restrict our study to expression rewrite rules, we will later deal with basic predicate rewrite rules.

Definition 7 (Conditional Identity). *A Σ -conditional identity (or simply conditional identity) is a triplet $(l, c, r) \in T_{\Sigma} \times P_{\Sigma} \times T_{\Sigma}$. In this case, l is called the left hand side, r the right hand side, and c the condition of the identity.*

Definition 8. *A conditional identity (l, c, r) is valid iff*

$$\vdash_{\mathcal{D}} \forall x_1, \dots, x_n \cdot c \Rightarrow l = r .$$

A conditional identity can be turned into a rewrite rule if it satisfies the syntactic restrictions presented in the following definition.

Definition 9 (Conditional Term Rewrite Rule). *A conditional term rewrite rule is a conditional identity (l, c, r) such that:*

1. l is not a variable,
2. $\mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$,
3. $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$.

In this case, we use the notation $l \xrightarrow{c} r$ instead of (l, c, r) .

Definition 10. *A conditional rewrite rule $l \xrightarrow{c} r$ is said to be WD-preserving if the following conditions hold:*

1. $\forall x_1, \dots, x_n \cdot \mathcal{D}(l) \Rightarrow \mathcal{D}(c)$,
2. $\forall x_1, \dots, x_n \cdot \mathcal{D}(l) \wedge c \Rightarrow \mathcal{D}(r)$.

In what follows, we assume that the reader is familiar with the basic notions of rewriting, for instance, in [5]. The usual notions such as a position (where ϵ denotes the root position), a subterm and subterm replacement are generalised to formulas (i.e., to include predicates). We use $[x := E]P$ to denote the syntactic replacement of free occurrences of x by E in P . We also use $x \setminus F$ to denote the non-freeness condition of x in F . We define the *domain* and the *range* of a substitution σ (both finite), denoted $\mathcal{D}om(\sigma)$ and $\mathcal{R}an(\sigma)$ respectively, as follows

$$\begin{aligned} \mathcal{D}om(\sigma) &= \{x \in V \mid \sigma(x) \neq x\} , \\ \mathcal{R}an(\sigma) &= \{\sigma(x) \mid x \in \mathcal{D}om(\sigma)\} . \end{aligned}$$

When p is a position in a formula F , we write $F|_p$ for the subformula (expression or predicate) of F at p . Finally, we write $F[s]_p$ for the formula obtained by replacing $F|_p$ by s in F .

We turn our attention to rewrite rule application. Consider applying rule $l \xrightarrow{c} r$ to $P[s]_p$. The left hand side l is matched against s by finding a substitution σ such that $\sigma(l) = s$ (one-way matching). Provided $\sigma(c)$ holds, $P[s]_p$ can be rewritten to $P[\sigma(r)]_p$. Following this approach, rewriting can be added as a proof step. The following two theorems ensure that Definition 10 is indeed sufficient to maintain well-definedness when rewriting. The proofs of these two results can be found in the appendix.

Theorem 1. *Let $l \xrightarrow{c} r$ be a conditional term rewrite rule, P be a predicate, p be a position within P such that $P|_p$ is an expression, and σ be a substitution. If $l \xrightarrow{c} r$ is valid, then the following holds*

$$\sigma(c) \Rightarrow (P[\sigma(l)]_p \Leftrightarrow P[\sigma(r)]_p) .$$

Theorem 2. *Let $l \xrightarrow{c} r$ be a conditional term rewrite rule, P be a predicate, p be a position within P such that $P|_p$ is an expression, and σ be a substitution. If $l \xrightarrow{c} r$ is valid and WD-preserving, then the following holds*

$$(\mathcal{D}(P[\sigma(l)]_p) \wedge \sigma(c)) \Rightarrow \mathcal{D}(P[\sigma(r)]_p) .$$

4.3 Rewriting as a Proof Step

Rewriting can be used in proofs alongside the WD-preserving sequent calculus. Conditional rewrite rules which have the same left hand side are grouped together. For this purpose, we use a notation similar to the one used in Section 2. Given a valid and WD-preserving (grouped) conditional rewrite rule

$$\begin{array}{c} l \rightarrow c_1 : r_1 \\ \dots \\ c_n : r_n \end{array}$$

we can add the following proof step to our calculus

$$\frac{\left\{ \begin{array}{l} H, P[\sigma(l)]_p \vdash_{\mathcal{D}} \mathcal{D}(\sigma(c_1 \vee \dots \vee c_n)) \\ H, P[\sigma(l)]_p \vdash_{\mathcal{D}} \sigma(c_1 \vee \dots \vee c_n) \\ H, \sigma(c_1), P[\sigma(r_1)]_p \vdash_{\mathcal{D}} R \dots H, \sigma(c_n), P[\sigma(r_n)]_p \vdash_{\mathcal{D}} R \end{array} \right.}{H, P[\sigma(l)]_p \vdash_{\mathcal{D}} R} \rightarrow hyp_{\mathcal{D}} \quad (1)$$

under the proviso that all free variables of $\sigma(c_i)$ (for all i such that $1 \leq i \leq n$) occur free in $P[\sigma(l)]_p$. This proof step allows the hypothesis $P[\sigma(r_1)]_p$ to be rewritten to several cases according to the rewrite rule. Under the same proviso, the following proof step can be added for goal rewriting

$$\frac{\left\{ \begin{array}{l} H, P \vdash_{\mathcal{D}} \mathcal{D}(\sigma(c_1 \vee \dots \vee c_n)) \\ H, P \vdash_{\mathcal{D}} \sigma(c_1 \vee \dots \vee c_n) \\ H, \sigma(c_1), P \vdash_{\mathcal{D}} R[\sigma(r_1)]_p \dots H, \sigma(c_n), P \vdash_{\mathcal{D}} R[\sigma(r_n)]_p \end{array} \right.}{H, P \vdash_{\mathcal{D}} R[\sigma(l)]_p} \rightarrow goal_{\mathcal{D}} . \quad (2)$$

Proof steps (1) and (2) can be derived using the cut rule, followed by a disjunction elimination (i.e., case split) after which rewriting can be applied (see appendix for formal derivation). We, now, examine a special case that can be used to facilitate proofs.

Definition 11 (Top-Level Occurrence). *Let t be an expression, P be a predicate, p be a position within Q . We say that t has a top-level occurrence p in P if P is either of the form*

1. $q(t_1, \dots, t_n)[t]_p$ where q is a predicate symbol and t_1, \dots, t_n are expressions, or;
2. $(t_1 = t_2)[t]_p$ where t_1 and t_2 are expressions.

If t has a top-level occurrence in P , then it also has a top-level occurrence in $\neg P$.

Proposition 1. *If the expression t has a top-level occurrence in predicate P , then the following holds*

$$\mathcal{D}(P) \Rightarrow \mathcal{D}(t) .$$

Proposition 1 can be used to simplify proofs. Let $P[\sigma(l)]_p$ be a predicate such that $\sigma(l)$ occurs at the top-level. Since the rewrite rule is valid and WD-preserving, and using the previous proposition, we have the following

$$\begin{aligned} \mathcal{D}(P[\sigma(l)]_p) &\Rightarrow \mathcal{D}(\sigma(l)) \\ &\Rightarrow \bigwedge_{i=1}^n \mathcal{D}(\sigma(c_i)) \end{aligned}$$

under the proviso that all free variables of $\sigma(C_i)$ (for all i such that $1 \leq i \leq n$) occur free in $P[\sigma(l)]_p$. In this particular case, the sequents

$$\begin{aligned} H, P[\sigma(l)]_p &\vdash_{\mathcal{D}} \mathcal{D}(\sigma(c_1 \vee \dots \vee c_n)) , \\ H, P &\vdash_{\mathcal{D}} \mathcal{D}(\sigma(c_1 \vee \dots \vee c_n)) \end{aligned}$$

in (1) and (2) respectively, are guaranteed to be discharged. As such, they could be removed from the list of sub-goals that the modeller sees.

Important Note. The results of this section can be generalised to cover rewrite rules whose left hand sides are basic predicates (i.e., $p(t_1, \dots, t_n)$ or $t_1 = t_2$). This can be achieved by syntactically defining basic predicate rewrite rules and following a similar approach to that of expression rewrite rules.

5 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 theories in the same way as contexts and machines. At the moment, theory development includes specification of rewrite rules including definition of sets and metavariables. Metavariables must be defined with their types which can be constructed from the theory sets and any built-in types (e.g., \mathbb{Z}) using type constructors (e.g., \mathbb{P}).
2. Users can validate rewrite rules through generated proof obligations. For the type of rules covered at the moment, the existing Atelier-B provers are sufficient to discharge all proof obligations. The proof obligations generated for rules are to establish soundness, well-definedness preservation as well as coverage-completeness.
3. Users can deploy theories to a specific directory where they become available to the interactive and automatic provers of Rodin. Theory deployment adds soundness information to all deployed rules.
4. 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.

6 Related Work

The architecture of proof tools continues to stir up much heated debate. One of the main talking points is how to strike a reasonable balance between three important attributes of the prover: efficiency, extensibility and soundness. In [13], Harrison outlines three options to achieve prover extensibility:

1. If a new rule is considered to be useful, simply extend the basic primitives of the prover to include it.
2. Use a full programming language to specify new rules using the basic primitives. The new rules ultimately decompose to these primitives.
3. Incorporate the *reflection* principle, so that the user can add and verify new rules within the existing infrastructure.

Many theorem provers including Isabelle [14] and HOL [15] employ the LCF approach. The functional language ML [16] is used to implement these systems, and acts as their meta-language. The approach taken by such systems is to use ML to define data types corresponding to logical entities such as terms and theorems. A number of ML functions are provided that can generate theorems; these functions implement the basic inference rules of the logic. The ML type system ensures that theorems are only constructed by the aforementioned functions. Therefore, the LCF approach offers both “reliability” and “controllability” of a low level proof checker combined with the power and flexibility of a sophisticated prover [13]. On the flip side, however, a major drawback for this approach is that each newly developed proof procedure must decompose into the basic inference rules. There are cases where this may not be a possible or indeed efficient solution e.g., truth table method for propositional logic [17].

The PVS [18] system follows a similar approach to LCF with more liberal support for adding external provers. This liberality comes at a risk of encountering soundness problems. It, however, presents the user with several choices

of automated provers which may ease the proving experience. A comparison between Isabelle/HOL and PVS from a user’s point of view is presented in [19]. Interestingly, it mentions that “soundness bugs are hardly ever unintentionally explored” during proof, and that “most mistakes in a system to be verified are detected in the process of making a formal specification”. A similar experience is reported when using the Rodin platform [9].

The Mural formal development system [20] consists of a VDM support tool and a proof assistant. In essence, it provides support for many-sorted predicate calculi which are expressible in natural deduction style. The Mural system allows adding internally proved rules i.e., rules that follow directly from existing rules. This results in the exclusion of a large class of rules that could be proved by employing a ‘more sophisticated meta-reasoning’. Adding new rules in Mural can be achieved through extending existing theories providing a verifiably “open system”.

Our approach does not necessarily subdue the old mechanism of extending the prover. As such, the new prover architecture resembles that of PVS. It still allows the liberality of integrating external decision procedures (e.g., for arithmetic) while providing a collection of sound rules. On the other hand, verifying the soundness of added rules using proof obligations enables meta-reasoning within the same platform. This can be viewed as a limited incorporation of the reflection principle within Rodin. The limitations of our approach, however, are similar to the limitations of the Mural architecture, since sophisticated meta-reasoning is not possible at the moment.

7 Future Work & Conclusive Remarks

In what follows, we outline the areas in which research can be carried out as an extension of this work:

- Extending the theory construct to enable specifying rewrite rules with side conditions as well as inference rules. This will require considering the different options for validating these types of rules.
- Verifying the new architecture using a Java verification tool. In particular, the verification of the pattern matching algorithms implemented will give more confidence in the new architecture.
- Provide guidelines that can be used to help the theory developer with deciding whether a specific rule should be applied automatically or manually. This requires termination and confluence analysis of theories since they effectively define rewriting systems.

We have presented an extension to the Event-B toolset that enables specification and verification of the soundness of rewrite rules. We have also shown how rewriting can be used as a proof step without deviating from the WD-preserving sequent calculus. We envisage the theory construct evolving to facilitate specification of other types of rule. It could also provide a foundation for extending the mathematical language used by Event-B.

References

1. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundam. Inf.* **77**(1-2) (2007) 1–28
2. Abrial, J.R.: *The B-book: assigning programs to meanings*. Cambridge University Press, New York, NY, USA (1996)
3. Butler, M., Hallerstede, S.: *The Rodin Formal Modelling Tool*. BCS-FACS Christmas 2007 Meeting - Formal Methods In Industry, London. (December 2007)
4. Abrial, J.R., Butler, M., Hallerstede, S., Voisin, L.: An open extensible tool environment for Event-B. In: *International Conference on Formal Engineering Methods (ICFEM)*. (2006)
5. Baader, F., Nipkow, T.: *Term rewriting and all that*. Cambridge University Press, New York, NY, USA (1998)
6. Mehta, F.: A practical approach to partiality - a proof based approach. In: *ICFEM*. (2008) 238–257
7. Behm, P., Burdy, L., Meynadier, J.M.: Well Defined B. In: *B '98: Proceedings of the Second International B Conference on Recent Advances in the Development and Use of the B Method*, London, UK, Springer-Verlag (1998) 29–45
8. Hallerstede, S.: On the Purpose of Event-B Proof Obligations. In: *ABZ '08: Proceedings of the 1st international conference on Abstract State Machines, B and Z*, Berlin, Heidelberg, Springer-Verlag (2008) 125–138
9. Mehta, F.: *Proofs for the Working Engineer*. PhD Thesis, ETH Zurich (2008)
10. Abrial, J.R., Cansell, D.: *Click'n Prove: Interactive Proofs within Set Theory*. (2003)
11. Steria: *Atelier B, User and Reference Manuals*, Aix-en-Provence, France. (1996) Available at http://www.atelierb.societe.com/index_uk.html.
12. Abrial, J.R., Mussat, L.: 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*, London, UK, Springer-Verlag (2002) 242–269
13. Harrison, J.: *Metatheory and reflection in theorem proving: A survey and critique*. Technical Report CRC-053, SRI Cambridge, Millers Yard, Cambridge, UK (1995) Available on the Web as <http://www.cl.cam.ac.uk/~jrh13/papers/reflect.dvi.gz>.
14. Nipkow, T., Paulson, L.C., Wenzel, M.: *Isabelles logics: Hol*
15. Gordon, M.: *HOL: A Machine Oriented Formulation of Higher Order Logic* (1985)
16. Milner, R., Tofte, M., Harper, R.: *The definition of Standard ML*. MIT Press, Cambridge, MA, USA (1990)
17. Cultura, I.T.D., Armando, A., Armando, A., Cimatti, A., Cimatti, A.: Building and executing proof strategies in a formal metatheory. In: *Advances in Artificial Intelligence: Proceedings of the Third Congress of the Italian Association for Artificial Intelligence, IA*AI'93*, Volume 728 of *Lecture Notes in Computer Science*, Springer-Verlag (1993) 11–22
18. Owre, S., Shankar, N., Rushby, J.M., Stringer-calvert, D.W.J.: *PVS Language Reference* (2001)
19. Griffioen, D., Huisman, M.: A comparison of PVS and Isabelle/HOL. In: *Theorem Proving in Higher Order Logics*, number 1479 in *Lect. Notes Comp. Sci*, Springer (1998) 123–142
20. Lindsay, P.A., Jones, C.B., Jones, K.D., Moore, R.D.: *Mural: A Formal Development Support System*. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1991)

A Proofs

First of all, we need the following propositions.

Proposition 2. *Let E be an expression, $l \xrightarrow{c} r$ be a WD-preserving conditional rewrite rule. We have the following properties:*

$$\begin{aligned} [x := E]\mathcal{D}(l) &\Rightarrow [x := E]\mathcal{D}(c) , \\ [x := E]\mathcal{D}(l) \wedge [x := E]c &\Rightarrow [x := E]\mathcal{D}(r) . \end{aligned}$$

Proposition 3. *Let E be an expression. If $l \xrightarrow{c} r$ is WD-preserving conditional rewrite rule, then the following holds:*

$$\begin{aligned} \mathcal{D}([x := E]l) &\Rightarrow \mathcal{D}([x := E]c) . \\ \mathcal{D}([x := E]l) \wedge [x := E]c &\Rightarrow \mathcal{D}([x := E]r) . \end{aligned}$$

The proof of the previous proposition is straightforward and relies on the following property concerning the well-definedness of expressions:

$$\mathcal{D}([x := E]l) \Leftrightarrow [x := E]\mathcal{D}(l) \wedge \mathcal{D}(E) \quad \text{if } x \text{ occurs free in } l . \quad (3)$$

Property 3 can be proved by induction on the structure of Σ -expressions, and can be generalised for arbitrary substitutions as follows

$$\mathcal{D}(\sigma(l)) \Leftrightarrow \bigwedge_{E \in \mathcal{R}an(\sigma)} \mathcal{D}(E) \wedge \sigma(\mathcal{D}(l))$$

if x occurs free in l for each $x \in \mathcal{D}om(\sigma)$.

Proposition 4. *Let σ be a substitution. If $l \xrightarrow{c} r$ is WD-preserving rule, then*

$$\begin{aligned} \mathcal{D}(\sigma(l)) &\Rightarrow \mathcal{D}(\sigma(c)) . \\ \mathcal{D}(\sigma(l)) \wedge \sigma(c) &\Rightarrow \mathcal{D}(\sigma(r)) . \end{aligned}$$

Proposition 5. *Let $l \xrightarrow{c} r$ be a conditional rewrite rule, and σ be a substitution. If $l \xrightarrow{c} r$ is valid, then*

$$\sigma(c) \Rightarrow \sigma(l) = \sigma(r) .$$

Proposition 6. *Let $l \xrightarrow{c} r$ be a conditional rewrite rule, t be an expression, p be a position within t and σ be a substitution. If $l \xrightarrow{c} r$ is valid, then*

$$\sigma(c) \Rightarrow t[\sigma(l)]_p = t[\sigma(r)]_p .$$

If $l \xrightarrow{c} r$ is WD-preserving, then

$$(\mathcal{D}(t[\sigma(l)]_p) \wedge \sigma(c)) \Rightarrow \mathcal{D}(t[\sigma(r)]_p) .$$

Proofs of Theorem 2 and Theorem 1 are similar. We just show the proof of Theorem 2.

Proof of Theorem 2. We proceed by induction on the structure of predicate P .

1. *Base Case:* $q(t_1, \dots, t_n)$. We have to prove

$$(\mathcal{D}(q(t_1, \dots, t_n)[\sigma(l)]_p) \wedge \sigma(c)) \Rightarrow \mathcal{D}(q(t_1, \dots, t_n)[\sigma(r)]_p) .$$

We have the following

$$q(t_1, \dots, t_n)[\sigma(l)]_p \hat{=} q(t_1, \dots, t_i[\sigma(l)]_{p'}, \dots, t_n)$$

for some position p' such that $p = ip'$. Since predicate symbols are total, we have the following

$$\mathcal{D}(q(t_1, \dots, t_i[\sigma(l)]_{p'}, \dots, t_n)) \Leftrightarrow \mathcal{D}(t_1) \wedge \dots \wedge \mathcal{D}(t_i[\sigma(l)]_{p'}) \wedge \dots \wedge \mathcal{D}(t_n)$$

By Proposition 6, we have

$$\begin{aligned} & \mathcal{D}(t_1) \wedge \dots \wedge \mathcal{D}(t_i[\sigma(l)]_{p'}) \wedge \dots \wedge \mathcal{D}(t_n) \wedge \sigma(c) \\ \Rightarrow & \\ & \mathcal{D}(t_1) \wedge \dots \wedge \mathcal{D}(t_i[\sigma(r)]_{p'}) \wedge \dots \wedge \mathcal{D}(t_n) \end{aligned}$$

We deduce that

$$\mathcal{D}(q(t_1, \dots, t_i[\sigma(l)]_{p'}, \dots, t_n)) \wedge \sigma(c) \Rightarrow \mathcal{D}(q(t_1, \dots, t_i[\sigma(r)]_{p'}, \dots, t_n)) .$$

Finally, we conclude

$$(\mathcal{D}(q(t_1, \dots, t_n)[\sigma(l)]_p) \wedge \sigma(c)) \Rightarrow \mathcal{D}(q(t_1, \dots, t_n)[\sigma(r)]_p) .$$

2. *Base Case:* $t_1 = t_2$. Similar to the previous case.
3. *Inductive Case:* $\neg P$. Proof omitted.
4. *Inductive Case:* $P \wedge Q$. We assume the inductive hypothesis

$$(\mathcal{D}(P[\sigma(l)]_p) \wedge \sigma(c)) \Rightarrow \mathcal{D}(P[\sigma(r)]_p) .$$

We have to prove

$$(\mathcal{D}((P \wedge Q)[\sigma(l)]_{1p}) \wedge \sigma(c)) \Rightarrow \mathcal{D}((P \wedge Q)[\sigma(r)]_{1p}) .$$

We have the following

$$\mathcal{D}((P \wedge Q)[\sigma(l)]_{1p}) \hat{=} \mathcal{D}(P[\sigma(l)]_p \wedge Q) .$$

By expanding $\mathcal{D}(P[\sigma(l)]_p \wedge Q)$, we have the following

$$\begin{aligned} & \mathcal{D}(P[\sigma(l)]_p \wedge Q) \wedge \sigma(c) \\ \Rightarrow & \\ & [(\mathcal{D}(P[\sigma(l)]_p) \wedge \mathcal{D}(Q)) \vee (\mathcal{D}(P[\sigma(l)]_p) \wedge \neg P[\sigma(l)]_p) \vee (\mathcal{D}(Q) \wedge \neg Q)] \wedge \sigma(c) . \end{aligned}$$

By the distributivity of \wedge over \vee , the inductive hypothesis and Theorem 1, we deduce

$$\begin{aligned} & \mathcal{D} (P[\sigma(l)]_p \wedge Q) \wedge \sigma(c) \\ \Rightarrow & (\mathcal{D} (P[\sigma(r)]_p) \wedge \mathcal{D}(Q)) \vee (\mathcal{D}(P[\sigma(r)]_p) \wedge \neg P[\sigma(r)]_p) \vee (\mathcal{D}(Q) \wedge \neg Q) . \end{aligned}$$

Finally, we conclude

$$(\mathcal{D}((P \wedge Q)[\sigma(l)]_{1p}) \wedge \sigma(c)) \Rightarrow \mathcal{D}((P \wedge Q)[\sigma(r)]_{1p}) .$$

5. *Inductive Case: $P \vee Q$.* Proof omitted.
6. *Inductive Case: $P \Rightarrow Q$.* Proof omitted.
7. *Inductive Case: $\forall x \cdot P$.* We assume the inductive hypothesis

$$\forall x \cdot ((\mathcal{D}(P[\sigma(l)]_p) \wedge \sigma(c)) \Rightarrow \mathcal{D}(P[\sigma(r)]_p)) .$$

such that x is not free in $\sigma(c)$. We have to prove

$$(\mathcal{D}((\forall x \cdot P)[\sigma(l)]_{1p}) \wedge \sigma(c)) \Rightarrow \mathcal{D}((\forall x \cdot P)[\sigma(r)]_{1p})$$

We have the following

$$(\forall x \cdot P)[\sigma(l)]_{1p} \hat{=} \forall x \cdot P[\sigma(l)]_p .$$

By expanding $\mathcal{D}(\forall x \cdot P[\sigma(l)]_p)$, we obtain

$$\begin{aligned} & \mathcal{D} (\forall x \cdot P[\sigma(l)]_p) \wedge \sigma(c) \\ \Rightarrow & [\forall x \cdot \mathcal{D}(P[\sigma(l)]_p) \vee (\exists x \cdot (\mathcal{D}(P[\sigma(l)]_p) \wedge \neg P[\sigma(l)]_p))] \wedge \sigma(c) \end{aligned}$$

By the distributivity of \wedge over \vee , the inductive hypothesis and Theorem 1, we deduce

$$\begin{aligned} & \mathcal{D} (\forall x \cdot P[\sigma(l)]_p) \wedge \sigma(c) \\ \Rightarrow & \forall x \cdot \mathcal{D}(P[\sigma(r)]_p) \vee (\exists x \cdot (\mathcal{D}(P[\sigma(r)]_p) \wedge \neg P[\sigma(r)]_p)) . \end{aligned}$$

Finally, we conclude

$$(\mathcal{D}((\forall x \cdot P)[\sigma(l)]_{1p}) \wedge \sigma(c)) \Rightarrow \mathcal{D}((\forall x \cdot P)[\sigma(r)]_{1p}) .$$

8. *Inductive Case: $\exists x \cdot P$.* Proof omitted.

□

B Derivations

In this section, we show the derivations resulting in proof steps (1) and (2). Since they are quite similar, we only show the formal derivation for (1). We start by applying the cut rule as follows

$$\frac{\begin{cases} H, P[\sigma(l)]_p \vdash_{\mathcal{D}} \mathcal{D}(\sigma(c_1 \vee \dots \vee c_n)) \\ H, P[\sigma(l)]_p \vdash_{\mathcal{D}} \sigma(c_1 \vee \dots \vee c_n) \\ H, \sigma(c_1) \vee \dots \vee \sigma(c_n), P[\sigma(l)]_p \vdash_{\mathcal{D}} R \end{cases}}{H, P[\sigma(l)]_p \vdash_{\mathcal{D}} R} \text{ cut}_{\mathcal{D}}$$

under the proviso that all free variables in $\sigma(c_i)$ (for all i such that $1 \leq i \leq n$) occur free in $P[\sigma(l)]_p$. Applying the disjunction elimination rule ($\vee hyp_{\mathcal{D}}$) to the sequent

$$H, \sigma(c_1) \vee \dots \vee \sigma(c_n), P[\sigma(l)]_p \vdash_{\mathcal{D}} R ,$$

results in the following

$$\frac{H, \sigma(c_1), P[\sigma(l)]_p \vdash_{\mathcal{D}} R \dots H, \sigma(c_n), P[\sigma(l)]_p \vdash_{\mathcal{D}} R}{H, \sigma(c_1) \vee \dots \vee \sigma(c_n), P[\sigma(l)]_p \vdash_{\mathcal{D}} R} \vee hyp_{\mathcal{D}} .$$

We can now rewrite the hypothesis $P[\sigma(l)]_p$ according to the rewrite rule as follows

$$\frac{H, \sigma(c_i), P[\sigma(r_i)]_p \vdash_{\mathcal{D}} R}{H, \sigma(c_i), P[\sigma(l)]_p \vdash_{\mathcal{D}} R} ,$$

where $1 \leq i \leq n$. Therefore, we have the proof step

$$\frac{\begin{cases} H, P[\sigma(l)]_p \vdash_{\mathcal{D}} \mathcal{D}(\sigma(c_1 \vee \dots \vee c_n)) \\ H, P[\sigma(l)]_p \vdash_{\mathcal{D}} \sigma(c_1 \vee \dots \vee c_n) \\ H, \sigma(c_1), P[\sigma(r_1)]_p \vdash_{\mathcal{D}} R \dots H, \sigma(c_n), P[\sigma(r_n)]_p \vdash_{\mathcal{D}} R \end{cases}}{H, P[\sigma(l)]_p \vdash_{\mathcal{D}} R} \rightarrow hyp_{\mathcal{D}} .$$