Addressing Extensibility Issues for Event-B

Issam Maamria and Michael Butler

Electronics and Computer Science
University of Southampton
Southampton, UK

April 28, 2011
Outline

1. Introduction
2. Motivation
3. Language Issues
4. Prover Issues
5. Our Requirements
6. Theory and Rule-based Prover
7. Future Work
Introduction

Tools include but not limited to:

- parser and type checker,
- editors and viewers,
- static checkers and proof obligation generators,
- a proof manager and a set of provers.
Provers include **ML** (rule-based), **PP** (semi-decision procedure) as well as the internal proving infrastructure which has a set of reasoners (schematic proof rules).

Proofs are important to modelling as they enhance the understanding of the model.

Simply inspecting failed automatic proof attempts can provide a sizeable insight into the corresponding model.

The proving infrastructure is extensible. The Rule-based Prover (RbP) is a rewriting-based prover contributed to the framework.

The usual issues of soundness arise when you allow such extensibility.
Extending the proving infrastructure of Rodin can be achieved through external provers as well as Java-based reasoners and tactics.

Rule-based Prover facilitates the specification of rewrite rules and their validation.

It is desirable to relieve the user from having to use Java for developing proof rules.

Soundness is a major concern when giving power to the user to specify proof rules.

An approach that achieves an acceptable trade-off between usability and maintaining soundness is certainly appealing.
CONTEXT

C₀

SETS

alpha

AXIOMS

axml₁ : ∃a · a ∈ alpha

END
We distinguish between Event-B modelling ‘outer’ syntax and ‘inner’ syntax.

Outer syntax is based on the modelling elements of the Rodin Database.

Many plug-ins take advantage of the database to add specialist modelling constructs e.g., Records and Modularisation plug-ins.

Extending the inner syntax (otherwise known as the mathematical language) is a different issue altogether.

The Event-B mathematical language is fixed as an AST.

As an example, the $\texttt{Seq}$ operator which is present in classical B is missing in Event-B. Many modelling patterns may benefit from having such an operator.
Records are recent addition to the Event-B language. The implementation is achieved by placing a syntactic layer whereby records are defined using a familiar syntactic sugar. The definitions are then translated to a set of Event-B functions.

```
tuple
closed
FIELDS
  head  type  alpha
  tail  type  P(alpha)
END
```
Records are important as they are natural choices for certain modelling patterns.

Users will appreciate more power to extend the Event-B mathematical language with operators that might correspond directly to certain elements of their model.

Examples of such operators include the transitive closure \textit{tcl} and the sequence operator \textit{Seq}.

A workaround that is not scalable is to define such operators using contexts. However, they are not operators as they cannot be attributed to be polymorphic. They can only be used with the types (i.e., carrier sets) with which they are defined.
Proofs are mightily important in the reactive approach used for Event-B in Rodin.

Reasoners are Java objects that encapsulate the notion of a schematic proof rule.

They are written in Java which must be scary to some users.

External prover (e.g., ML and PP) can be added by implementing certain interfaces.

The Rule-based Prover can be used for a certain class of rewrite rules.

What is desirable is to have a framework to specify inference rules, rewrite rules as well as polymorphic theorems.
It is an important usability issue that simple proof obligations (POs) get discharged automatically. The idea is that trivial POs do not offer much insight into the model.

However, users will appreciate the ability to define their proof rules. The Isabelle family of theorem provers offers such capabilities.

Furthermore, extensions to the Event-B inner syntax must be accompanied by prover extensions. Users must be able to reason about their extensions.

The Event-B approach of PO generation can be emulated to ensure soundness issues are brought to the user’s attention as is done in the Rule-based Prover.
Our Requirements

- **Language Extensions:**
  - Add support for user-defined (polymorphic) operators.
  - Add support for user-defined datatypes (e.g., Tree).

- **Prover Extensions:**
  - Add support for user-defined inference rules.
  - Add support for user-defined polymorphic theorems.
The Theory component:

```
theory  name

  type parameters  $T_1, \ldots, T_n$

  \{ ⟨ Datatype Definitions ⟩ 
  | ⟨ Operator Definitions ⟩ 
  | ⟨ Rewrite Rules ⟩ 
  | ⟨ Inference Rules ⟩ 
  | ⟨ Theorems ⟩ \}
```
Operator Definitions:

operator symbol (predicate | expression)
(prefix | infix) [assoc] [commut]
arguments $x_1, \ldots, x_m$
condition $P(x_1, \ldots, x_n)$
definition $E(x_1, \ldots, x_n)$
Operator Proof Obligations:

**Well-definedness** \( P(x_1, ..., x_n) \Rightarrow D(E(x_1, ..., x_n)) \)

- Commutativity
- Associativity

Operator Example:

\[
\begin{align*}
T_0.seq & \triangleq \text{Seq EXPRESSION PREFIX} \\
\text{Arguments} & \quad a \in \mathcal{P}(A) \\
\text{Direct Definition} & \quad \{n,f \cdot f \in 1 \cdots n \rightarrow a|f\}
\end{align*}
\]

\[
\begin{align*}
T_0.empty & \triangleq \text{empty PREDICATE PREFIX} \\
\text{Arguments} & \quad a \in \mathcal{Z} \mapsto A \\
\text{Condition} & \quad a \in \text{Seq}(A) \\
\text{Direct Definition} & \quad a = \emptyset
\end{align*}
\]
Datatype Definitions:

```plaintext
datatype  name 
  type arguments  T_1, ..., T_p 
  constructors  
    c_1(d_1^1 : E_1^1, ..., d_j^j : E_j^j) 
    . 
    . 
    c_q(d_q^1 : E_q^1, ..., d_k^k : E_k^k) 
```

Theory component and Rule-based Prover(4)
Datatype Example:

\[
\text{Tree}(A) \doteq \\
\begin{cases}
\text{empty} \\
\text{node(left:Tree(A), leaf:A, right:Tree(A))}
\end{cases}
\]

Research is being carried out on the foundations of datatype extensions for Event-B.
Language extensions require prover extensions.
Theories can contain rewrite rules, inference rules and polymorphic theorems.
All rules are polymorphic on their corresponding theory type parameters.
Inference rule example:

\[ x = 0 \lor y = 0 \quad \vdash \quad x \ast y = 0 \]

Theorem example:

\[ \forall x, y \cdot (x = 0 \lor y = 0) \Rightarrow x \ast y = 0 \]
Future Work

- Add support for operators with mixed children (predicates and expressions).
- Optimise prover extensions.
- Enhance support for datatypes.
- Create a library of operators and datatypes.
- Add support for user-defined binders.
Questions