UNIVERSITY OF Southampton
School of Electronics and Computer Science

Addressing Extensibility Issues in Rodin

Language and Prover Extensibility

Issam Maamria and Michael Butler

Electronics and Computer Science
University of Southampton
Southampton, UK

September 22, 2010

Outline

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*.

Introduction(2)

- 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.

Motivation

- 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.

Motivation(2)

```
CONTEXT
  C0

SETS
  alpha

AXIOMS
  axm1    :    ∃a· a ∈ alpha

END
```

Motivation(3)

- 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 *Seq* operator which is present in classical B is missing in Event-B. Many modelling patterns may benefit from having such an operator.

Rule-based Prover

- The Rule-based Prover is a rewriting-based prover that can work in two modes: automatic and interactive.
- The user can specify rewrite rules, and proof obligations are generated to validate them.
- The concept of theory deployment ensures that the user is aware of the soundness of the specified rules.
- The prover uses the existing reasoner and tactics framework. It, however, uses a generic pattern matching mechanism to handle rules applicability and application.

Rule-based Prover(2)

```
THEORY
  T0

METAVARIABLES
  a  ∈  ℤ
  b  ∈  ℤ

REWRITE RULES

  rule1   :   card(a‥b)        (coverage-complete, non-automatic, interactive)
  ≙
   rhs1   :   a ≤ b   ⟶   b−a+1
   rhs2   :   a > b   ⟶   0

END
```

Language Issues

- 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
```

Language Issues(2)

- 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 *tcl* and the sequence operator *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.

Prover Issues

- 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.

Prover Issues(2)

- 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.

Theory component and Rule-based Prover

- The Theory component:

theory *name*

type parameters $T_1, ..., T_n$

$\{\langle$ *Datatype Definitions* $\rangle$

$|\langle$ *Operator Definitions* $\rangle$

$|\langle$ *Rewrite Rules* $\rangle$

$|\langle$ *Inference Rules* $\rangle$

$|\langle$ *Theorems* $\rangle\}$

Theory component and Rule-based Prover(2)

- Operator Definitions:

operator *symbol* **(predicate** | **expression)**
(prefix | **infix) [assoc] [commut]**
arguments $x_1, ..., x_m$
condition $P(x_1, ..., x_n)$
definition $E(x_1, ..., x_n)$

Theory component and Rule-based Prover(3)

- Operator Proof Obligations:

  Well-definedness $P(x_1, ..., x_n) \Rightarrow \mathcal{D}(E(x_1, ..., x_n))$

  Commutativity

  Associativity

- Operator Example:

T0.Seq ≙ Seq EXPRESSION PREFIX

Arguments

a ∈ ℙ(A)

Direct Definition

{n,f·f∈1‥n⟶a|f}

T0.empty ≙ empty PREDICATE PREFIX

Arguments

a ∈ ℤ⤔A

Condition

a ∈ Seq(A)

Direct Definition

a = ∅

Theory component and Rule-based Prover(4)

- Datatype Definitions:

datatype *name*

type arguments $T_1, ..., T_p$

constructors

$c_1(d_1^1 : E_1^1, ..., d_1^j : E_1^j)$

.

.

$c_q(d_q^1 : E_q^1, ..., d_q^k : E_q^k)$

Theory component and Rule-based Prover(5)

- Datatype Example:

```
Tree(A)  ≜
  ▶ empty
  ▶ node(left:Tree(A), leaf:A, right:Tree(A))
```

- Research is being carried out on the foundations of datatype extensions for Event-B.

Theory component and Rule-based Prover(6)

- 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:

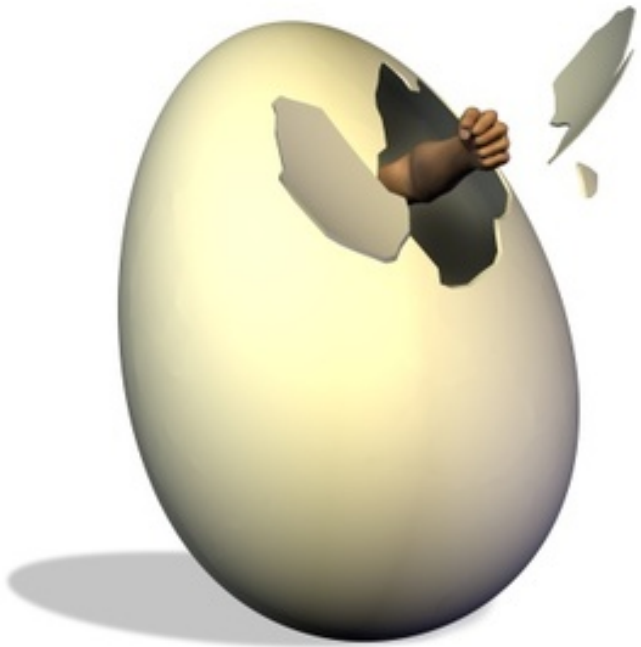
$$x = 0 \lor y = 0$$
$$\vdash$$
$$x * y = 0$$

- Theorem example:

$$\forall x, y \cdot (x = 0 \lor y = 0) \Rightarrow x * y = 0$$

Timeline

- Release of Theory Plug-in v0.5: October 8, 2010.



- Need user feedback to improve the plug-in before it gets shipped as part of the Core.

Conclusion

- We have shown an approach that addresses extensibility issues in Rodin using a familiar technique.
- Proof obligations are generated to ensure soundness is not compromised.
- Operator definitions do not extend the language in the true sense of the word.
- Datatype definitions do extend the language. More work is being carried out on the foundations of such additions.
- Language extensions go hand in hand with prover extensions. As such, both are defined using the same construct.

Acknowledgement

This work has been possible with the valuable contributions from various people:

- Laurent Voisin
- Nicolas Beauger
- Thomas Muller
- Carine Pascal
- Matthias Shmaltz
- Stefan Hallerstede
- Michael Leuschel
- The Southampton Team

Questions