University of Southampton

UNIVERSITY OF Southampton
School of Electronics and Computer Science

An extensible rule-based prover for Event-B

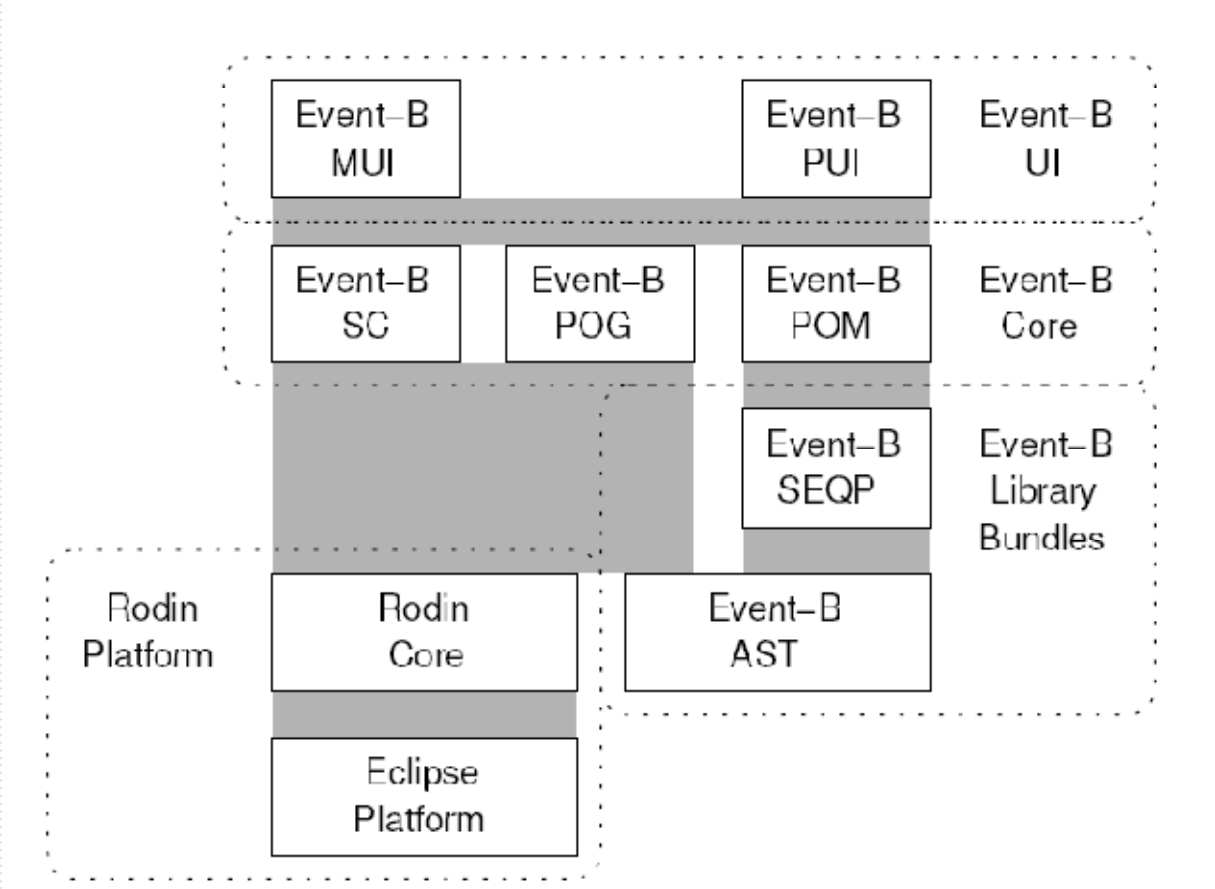
Issam Maamria
Michael Butler
Andrew Edmunds
Abdolbaghi Rezazadeh

July 16th, 2009

Outline

1. Current Architecture
2. Limitations
3. Proposal
4. Q&A

Event-B Tool


Event–B
MUI
Event–B
PUI
Event–B
UI
Event–B
SC
Event–B
POG
Event–B
POM
Event–B
Core
Event–B
SEQP
Event–B
Library
Bundles
Rodin
Platform
Rodin
Core
Event–B
AST
Eclipse
Platform

Event-B Tool (The Prover)

- What to prove is represented as a sequent **Hyp |- Goal**.
- A proof is represented as a tree with its root as the sequent to prove.
- A proof rule is generated using a rule schema (a reasoner).

Event-B Tool (The Prover)

- Basic tactics are wrappers around proof rules to act on proof trees.
- There are tactical tactics.
- Tactical tactics .e.g., apply a tactic on all pending sub-goals, compose a number of tactics.

Event-B Tool (The Prover)

- If a new rule is to be added:
  - *org.eventb.core.seqprover.reasoners*
  - *org.eventb.core.seqprover.autoTactics*
  - *org.eventb.ui.proofTactics*
  - *org.eventb.core.postTactics*
  - *org.eventb.core.pomTactics*

The point is you have to write Java code.

The Prover: limitations

- The need for writing Java code.
- Maintain the soundness of the prover after adding rules:
  - Testing
  - Verification of Java code

Proposal

- Specify proof rules in a similar manner to developing models.
- A new construct *Theory* distinct from contexts and machines.
- Generate proof obligations to validate rules.

Proposal

- Reasoners define how they are applied inside their Java classes.
- Instead, we use a generalised pattern matching mechanism to check for applicability.

Proposal

- Theory Development Lifecycle
  - Development
    - Specify proof rules.
    - Validate proof rules.
  - Deployment
    - Deploy sound theories.
    - Use proof rules with the generalised pattern matching mechanism.

Proposal

- Initially, we will specify rewrite rules.

- Conditional rewrite rules of the form:
  - $lhs \quad \Xi \quad C_1 : rhs_1$

    $\ldots$

    $C_n : rhs_n$

$n \geq 1$

Syntactic constraints are handled by the Static Checker.

Proposal

- Unconditional rewrite rules are a special case of conditional rules.
- Proof obligations are to ensure *soundness* as well as *well-definedness preservation*.

Proposal

- Conditional rewrite rule PO's:
  - WD(lhs) |- WD(C_i) *WD-P*
  - WD(lhs), C_i |- WD(rhs_i) WD-P
  - WD(lhs), C_i |- lhs = rhs_i or S
  - WD(lhs), C_i |- lhs $\Leftrightarrow$ rhs_i S

Proposal

- After deploying a theory, it can be used.
- Annotations are used for rules to specify how they should be handled by the prover (.e.g., automatic rules)
- The pattern matching mechanism will work out what rule are applicable for a given sequent and at what positions.

Proposal

- So far:
  - The Theory construct.
    - Caters for rewrite rules initially.
    - PO's are generated.
- TODO:
  - Specify and implement the pattern matching

Q&A

- Any questions?