

An Event-B Sequent Prover in Prolog and ProB



Katharina Engels, Jan Gruteser, Michael Leuschel

13th Rodin Workshop
東京 (Hybrid), 18.05.2026

hhu Heinrich Heine
University
Düsseldorf 

prob.hhu.de
stups.hhu.de

Last Year:

- We started implementing the Rodin proof rules in Prolog
- Initial idea: improve traceability and understanding of proofs in teaching
 - ▶ In Rodin, individual proofs steps can be difficult to explore (and understand)
- Turn mathematical definitions into executable Prolog rules that can be used in a prover

https://wiki.event-b.org/index.php/Inference_Rules
https://wiki.event-b.org/index.php/All_Rewrite_Rules

- ProB's sequent prover allows to *animate* the proof rules
 - ▶ Each PO is a start state
 - ▶ Based on the current sequent and the transition predicates, the ProB animator gives the applicable proof rules
 - ▶ The state is the current proof sequent (+ continuations)
- Prolog transition for the **IMP_R** rule:

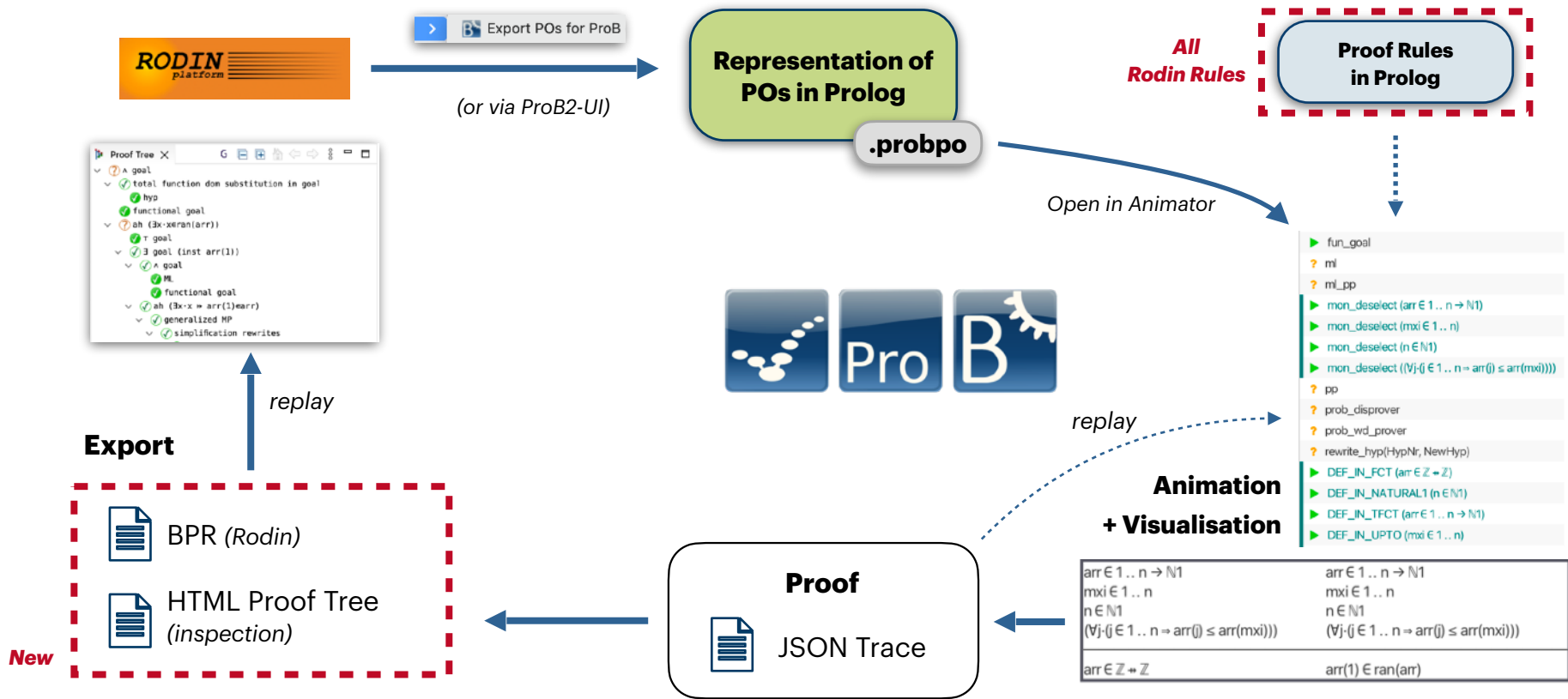
```
trans(imp_r, sequent(Hyps, implication(P,Q), Cont),
      sequent(Hyps1, Q, Cont)) :-
    add_hyp(P, Hyps, Hyps1).
```

$$\frac{H, P \vdash Q}{H \vdash P \Rightarrow Q}$$

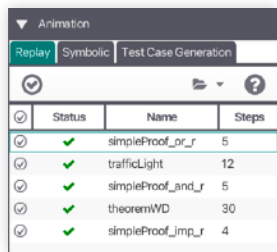
```
▶ imp_r
▶ lasso
? ml
? ml_pp
▶ mon_deselect(ELEMENTS ⊆ Z)
▶ mon_deselect(¬(elements = ∅))
? pp
? prob_disprover
? prob_wd_prover
▶ reselect_hyp(e ≤ 0 ∧ e ∈ elements)
▶ reselect_hyp(elements \ {e} = ∅)
▶ reselect_hyp(finite(ELEMENTS))
▶ reselect_hyp(elements ⊆ ELEMENTS)
▶ reselect_hyp({x:0 < x ∧ x ∈ ELEMENTS \ elements:x} ⊆ ELE
? rewrite_hyp(HypNr, NewHyp)
▶ DERIV_IMP (y ∈ {x:0 < x ∧ x ∈ ELEMENTS:x} ⇒ y ∈ {x:0 < x ∧
▶ SIMP_IN_COMPSET_ONEPOINT (y ∈ {x:0 < x ∧ x ∈ ELEME
▶ SIMP_IN_COMPSET_ONEPOINT (y ∈ {x:0 < x ∧ x ∈ ELEME
```

- Allmost all rules implemented (≈ 600) and integrated into the ProB core
- Interactive proving with ProB together with visualisation
- New exports:
 - ▶ Interactive HTML proof tree
 - ▶ Proof export as Rodin BPR file (experimental)
- First version of an automated prover

Overview

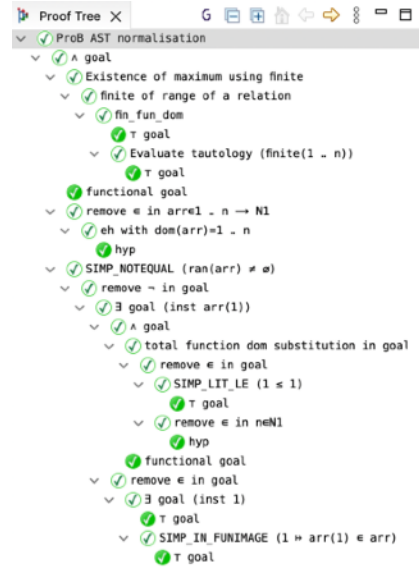
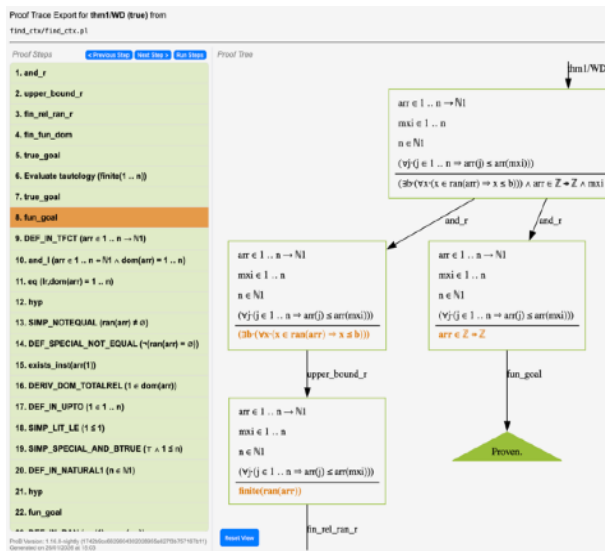


Proof Exports from ProB



History (state 7 of 8)

Position	Transition	Description
0	--root--	
1	start_xtl_system	inv14/WD (true)
2	reselect_hyp(member\$(radiat...)	reselect_hyp (radiation ∈ field ...
3	reselect_hyp(subset(set_exte...	reselect_hyp ((rover) ⊆ field)
4	and_r	
5	fun_goal	
6	simplify_goal(DERIV_DOM_T...	DERIV_DOM_TOTALREL (ro...
7	simplify_hyp(SIMP_SUBSE...	SIMP_SUBSETEQ_SING (r...
8	hyp	



JSON Trace of Proof Steps
replay with ProB

Interactive HTML Document
inspection of the proof tree

BPR Proof File
replay with Rodin
(2nd chain)

Demo: ProB2-UI

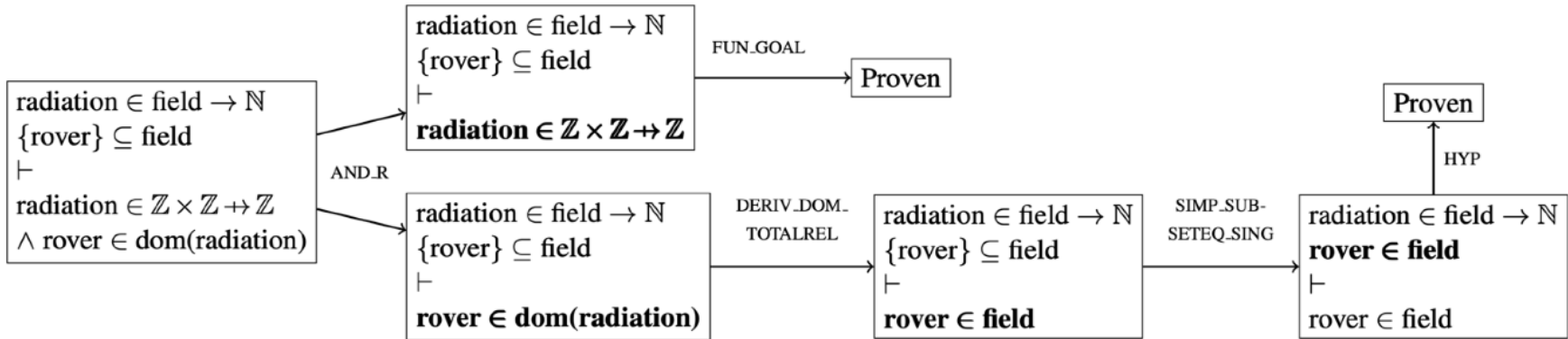
The screenshot shows the ProB2-UI interface for the 'Rover3.probpo' project. The main window is titled 'Rover3.probpo - SequentProverExamples - ProB2-UI*'. The interface is divided into several panels:

- Operations:** A list of operations including `fin_reLj(ReI)`, `fin_subseteq_r(Set)`, `forall_instl(hypNr, Inst)`, `forall_inst_rmp(hypNr, Inst)`, `forall_inst_rst(hypNr, Inst)`, `hyp`, `lasso`, `mt`, `mt_po`, `mon_deselect (radiation E field → %)`, `mon_deselect (rover E field)`, `pp`, `prob_disprover`, `prob_wd_prover`, `reselect_hyp (I0 ≤ B ∧ B ∈ N1)`, and `reselect_hyp (I0 ≤ L A L E N1)`. An 'Export Proof' button is visible.
- Animation:** Includes a 'Replay' button and a table for 'Test Case Generation' with columns for Status, Name, and Steps. The table shows a single entry: `rover3_inv14_wd_with_fungoal` with status '✓' and steps '8'.
- Visualisation:** Shows 'State Visualisation' with 'Current State' and 'Previous State' boxes. The current state contains: `radiation E field → %`, `rover E field`, and `rover E field`. The previous state contains: `radiation E field → %`, `rover ⊆ field`, and `rover E field`.
- History:** A table with columns 'Position', 'Transition', and 'Description'. It shows a sequence of transitions from position 0 to 8, including `start_xl_system`, `reselect_hyp(member$(radiat... reselect_hyp (radiation e field ...`, `reselect_hyp(subset(set_exta... reselect_hyp (rover) ⊆ field)`, `and_r`, `fun_goal`, `simplify_goal(DERIV_DOM_T... DERIV_DOM_TOTALREL (ro...`, `simplify_hyp(SIMP_SUBSE... SIMP_SUBSETEQ_SING (r...`, and `hyp`.

The screenshot shows the 'Proof Obligations' and 'Proof Tree' panels in the ProB2-UI. The main window is titled 'ProB2-UI*'. The 'Proof Obligations' panel shows a table with columns 'Status', 'ID', and 'Proof Obligations'. The table contains several entries, all with a status of '✓'. The 'Proof Tree' panel shows a tree structure for the proof, starting with 'ProB AST normalisation' and 'sL/ds', leading to a goal `reselect_hyp ((rover) ⊆ field)`, which is further broken down into a functional goal, a total function dom substitution in goal, and a hypothesis `SIMP_SUBSETEQ_SING ((rover) ⊆ field)`.

Demo: Mars-Rover

$$\text{FUN_GOAL} \quad \frac{}{H, f \in E \text{ op } F \vdash f \in T_1 \leftrightarrow T_2}$$



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

$$\text{DERIV_DOM_TOTALREL} \quad \text{dom}(r) \hat{=} E$$

$$\text{SIMP_SUBSETEQ_SING} \quad \{E\} \subseteq S \hat{=} E \in S$$

Rules with User Input:

- „execute by predicate“ mechanism
 - ▶ transitions arguments provided by the user

Additional Provers:

- Symbolic transitions
 - ▶ can only be executed manually
 - ▶ call external provers
 - ▶ e.g. AtelierB ML/PP, ProB WD/Disprover

EXISTS_INST

$$\frac{H \vdash WD(E) \quad H \vdash P(E)}{H \vdash \exists x \cdot P(x)}$$

Execute by Predicate

Operation: exists_inst

Parameters:

Name	Value	Type
Inst	{x x ∈ S ∧ x ∈ T}	Parameter

Additional predicate (evaluated in post-state, use \$0 to access variabl...)

STATE_PROPERTY("GOAL") = "T"

Execute

Failed to execute operation with the specified parameters ... Visual...

Prolog vs Java Encoding of Proof Rules

$$\text{FIN_SETMINUS_R} \quad \frac{H \vdash \text{finite}(S)}{H \vdash \text{finite}(S \setminus T)}$$

```
trans_desc_wo_info(
  fin_setminus_r,
  sequent(Hyps, finite(set_subtraction(S, _)), Cont),
  sequent(Hyps, finite(S), Cont),
  'finite of \'
).
```

```
public class FiniteSetMinus extends EmptyInputReasoner {

    public static final String REASONER_ID = SequentProver.PLUGIN_ID + ".finiteSetMinus";

    @Override
    public String getReasonerID() {
        return REASONER_ID;
    }

    @ProverRule("FIN_SETMINUS_R")
    protected IAntecedent[] getAntecedents(IProverSequent seq) {
        Predicate goal = seq.goal();

        // goal should have the form finite(S \ T)
        if (!Lib.isFinite(goal))
            return null;
        SimplePredicate sPred = (SimplePredicate) goal;
        if (!Lib.isSetMinus(sPred.getExpression()))
            return null;

        // There will be 1 antecedents
        IAntecedent[] antecedents = new IAntecedent[1];

        BinaryExpression aExp = (BinaryExpression) sPred.getExpression();

        Expression S = aExp.getLeft();
        final FormulaFactory ff = seq.getFormulaFactory();
        Predicate newGoal = ff.makeSimplePredicate(Predicate.KFINITE, S, null);

        antecedents[0] = ProverFactory.makeAntecedent(newGoal);
        return antecedents;
    }

    protected String getDisplayName() {
        return "finite of \" ;
    }

    @Override
    public IReasonerOutput apply(IProverSequent seq, IReasonerInput input,
        IProofMonitor pm) {
        IAntecedent[] antecedents = getAntecedents(seq);
        if (antecedents == null)
            return ProverFactory.reasonerFailure(this, input,
                "Inference " + getReasonerID() + " is not applicable");

        // Generate the successful reasoner output
        return ProverFactory.makeProofRule(this, input, seq.goal(),
            getDisplayName(), antecedents);
    }
}
```

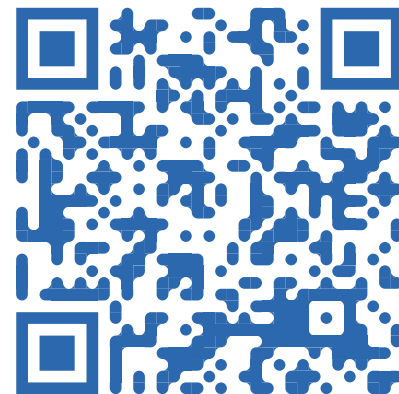
Prolog vs Java Encoding of Proof Rules

```
simp_rule_with_hyps(domain(R), Dom, 'DERIV_DOM_TOTALREL', Hyps) :-  
    member(member(R, RType), Hyps), is_rel(RType, total, Dom, _).  
simp_rule_with_hyps(range(R), Ran, 'DERIV_RAN_SURJREL', Hyps) :-  
    member(member(R, RType), Hyps), is_surj(RType, _, Ran).
```

- Java (Rodin): class with more than 200 lines for first rule, second rule not implemented
- Prolog (ProB): add second rule with 1-2 lines of code

Tool	Language	Code for Sequent Prover				Missing Rules	
		Files	LOC	SLOC	Person Years	Rewrite	Inference
Rodin	Java	320	50182	27580	6.51	55/536	16/123
PROB	Prolog	4	4208	3705	0.79	12/536	8/123

- Implemented almost all of Rodin proof rules in Prolog
 - ▶ new Sequent Prover mode of ProB
 - ▶ HTML: interactive proof tree export — also useful for teaching
 - ▶ BPR: proof replay in Rodin — use of ProB prover as second chain
- Ideas for future work:
 - ▶ continue and improve automated prover, integrate as Rodin Plugin?
 - ▶ PO generation with ProB
 - ▶ AI-guided proof
- More ideas, questions?



https://prob.hhu.de/w/index.php?title=Sequent_Prover
https://stups.hhu-hosting.de/models/sequent_prover