Interactive Proving with ProB

Katharina Engels, **Jan Gruteser**, Michael Leuschel

12th Rodin Workshop
Düsseldorf, 10.06.2025

Heinrich Heine
University
Düsseldorf

prob.hhu.de
stups.hhu.de

ProB

hhu
Heinrich Heine
University Düsseldorf

- Model checker, animator, and constraint solver
- High-level formal specification languages (like Event-B and TLA+)

- Tooling:
  - CLI
  - ProB Java API
  - **ProB Rodin Plugin**
  - **ProB Disprover**
  - ProB2-UI: VisB, SimB

- Developed by STUPS group at HHU Düsseldorf (open source)

https://prob.hhu.de

Motivation

Heinrich Heine University Düsseldorf

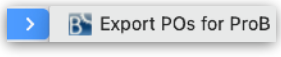
1. Teaching

- In our course on Event-B modelling, we teach students how to prove proof obligations (POs)
- Rodin may not be a good choice for beginners
  - complex proof view
  - can be difficult to explore (and understand) single proof steps
- Idea: interactive proving with ProB together with visualisation + HTML export
  - Improve traceability and understanding of proofs
  - Comprehensible and explorable proof (tree)

2. Integration of Proof Rules in ProB (later)

- ProB already has a WD prover, which tries to prove as much as possible in a “big” step
- Idea: use the Prolog implementation of proof rules for (automatic?) proving of arbitrary POs in ProB

Overview



Representation of POs in Prolog

Core of this work

Proof Rules in Prolog

ProB XTL Mode

Animation + Visualisation

arr ∈ 1 .. n → ℕ₁	arr ∈ 1 .. n → ℕ₁
mxi ∈ 1 .. n	mxi ∈ 1 .. n
n ∈ ℕ₁	n ∈ ℕ₁
∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j))	∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j))
---	---
⊤	1 .. n ∈ FIN(1 .. n)

Export

- HTML Trace *(inspection)*
- JSON Trace *(replay)*

POs in Prolog

1. Use Rodin to generate proof obligations (or create by hand)
2. Export them via `ProB Standalone > Export POs for ProB`
   - Prolog representation of all generated POs (originally for ProB Disprover)

3. Translate and normalise AST according to WD prover representation
   - Ignore position information
   - Allow later integration into ProB core

```
[member($'d',natural_set),…,less_equal($'n',$'d'),…]

 member(add($'a',add($'b',$'c')),natural_set)
```

thm1/THM

$$\frac{d \in \mathbb{N} \quad n \leq d \quad \ldots}{c + b + a \in \mathbb{N}}$$

Encoding Proof Rules in Prolog

Heinrich Heine University Düsseldorf

- We use the Rodin proof rules as a starting point
  - Rewrite rules
  - Inference rules

$$E \in \{F\} \;\widehat{=}\; E = F \qquad \textbf{SIMP\_IN\_SING}$$

$$\frac{\mathbf{H}, \mathbf{P} \vdash \mathbf{Q}}{\mathbf{H} \vdash \mathbf{P} \Rightarrow \mathbf{Q}} \qquad \textbf{IMP\_R}$$

- Turn mathematical definitions into executable Prolog rules that can be used in a prover:

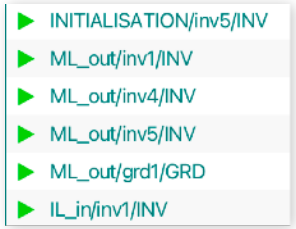
```
trans(simplify_goal(Rule), sequent(Hyps,Goal,Cont), sequent(Hyps,NewGoal,Cont)) :-
    simp_rule(Goal,NewGoal,Rule).

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

[…]

simp_rule(member(E,SetF), equal(E,F), 'SIMP_IN_SING') :-
    singleton_set(SetF,F).
```

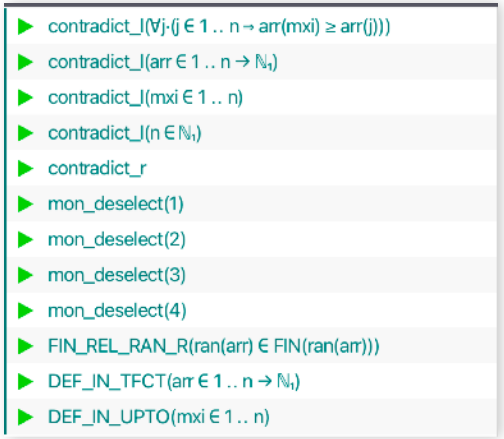
https://wiki.event-b.org/index.php/Inference_Rules
https://wiki.event-b.org/index.php/All_Rewrite_Rules

Animate the Proof

- ProB's XTL mode allows to *animate* the proof rules specified in Prolog
  - `start(InitialState)`
  - `trans(Label, StateBefore, StateAfter)`
- Labelled transition system:
  - Each PO is a start state, proof rules are transitions
  - The state is the current proof sequent:

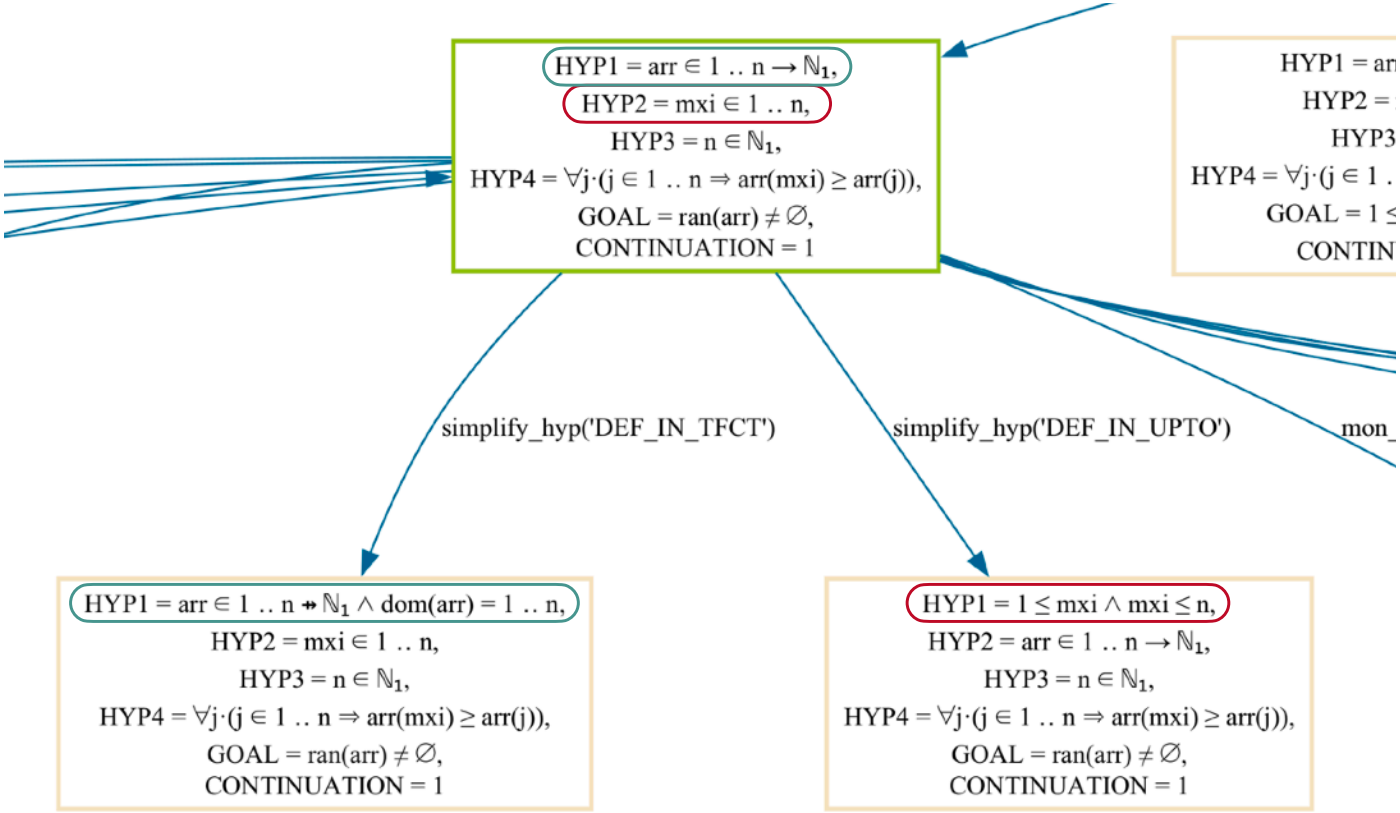
    `sequent([Hyps], Goal, Continuations)`
  - Continuations: branches of the proof tree that are not yet proven
- Based on the current sequent and the transition predicates, the ProB animator gives the applicable proof rules
  - we can provide more readable descriptions for transitions/rules





https://prob.hhu.de/w/index.php?title=Other_languages

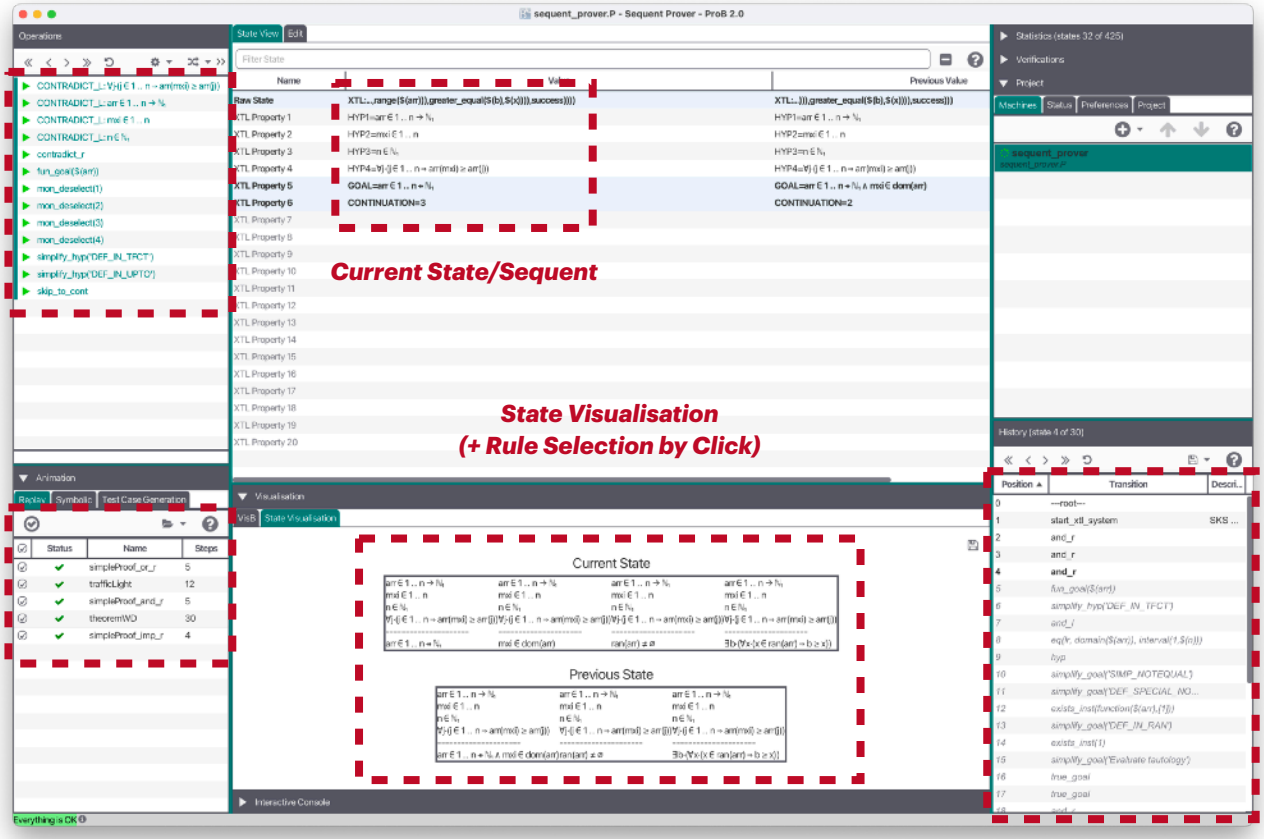
State Space


HYP1 = arr ∈ 1 .. n → ℕ1,
HYP2 = mxi ∈ 1 .. n,
HYP3 = n ∈ ℕ1,
HYP4 = ∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j)),
GOAL = ran(arr) ≠ ∅,
CONTINUATION = 1
HYP1 = ar
HYP2 =
HYP3
HYP4 = ∀j·(j ∈ 1 .
GOAL = 1 ≤
CONTIN
simplify_hyp('DEF_IN_TFCT')
simplify_hyp('DEF_IN_UPTO')
mon_
HYP1 = arr ∈ 1 .. n ⇸ ℕ1 ∧ dom(arr) = 1 .. n,
HYP2 = mxi ∈ 1 .. n,
HYP3 = n ∈ ℕ1,
HYP4 = ∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j)),
GOAL = ran(arr) ≠ ∅,
CONTINUATION = 1
HYP1 = 1 ≤ mxi ∧ mxi ≤ n,
HYP2 = arr ∈ 1 .. n → ℕ1,
HYP3 = n ∈ ℕ1,
HYP4 = ∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j)),
GOAL = ran(arr) ≠ ∅,
CONTINUATION = 1

ProB2-UI as an Interactive Prover

Heinrich Heine University Düsseldorf

Applicable Proof Rules



Replay Traces of Proofs

Applied Proof Steps

Demo

- Simple example: `thm1/THM`
  - can be proven easily with `EQL_LR` for n, then `HYP`

thm1/THM

$$n = c + b + a$$
$$n \in \mathbb{N}$$
$$\dots$$
$$\text{———————}$$
$$c + b + a \in \mathbb{N}$$

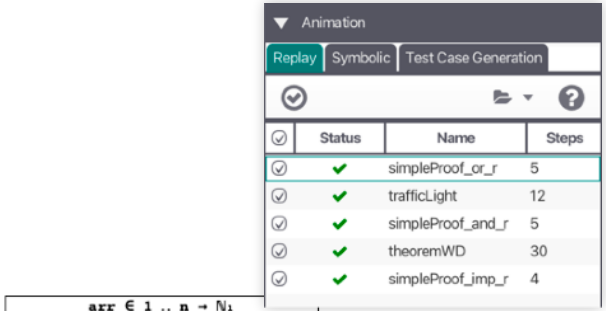
- More advanced example:
  - 30 proof steps

$$arr \in 1 \mathinner{..} n \rightarrow \mathbb{N}_1$$
$$mxi \in 1 \mathinner{..} n$$
$$n \in \mathbb{N}_1$$
$$\forall j \cdot (j \in 1 \mathinner{..} n \Rightarrow arr(mxi) \geq arr(j))$$
$$\text{———————————————————}$$
$$arr \in 1 \mathinner{..} n \twoheadrightarrow \mathbb{N}_1 \land mxi \in dom(arr) \land ran(arr) \neq \varnothing \land \exists b \cdot (\forall x \cdot (x \in ran(arr) \Rightarrow b \geq x))$$

Proof Export: Traces and HTML

- Enabled JSON Trace Export for ProB's XTL mode
- Save sequence of applied proof rules for
  - later replay of the proof
  - continuation of the work on a partial proof

- Human-readable HTML export of a trace together with visualisation
  - standalone document to inspect the proof



arr ∈ 1 .. n → ℕ₁
mxi ∈ 1 .. n
n ∈ ℕ₁
∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j))

∃b·(∀x·(x ∈ ran(arr) ⇒ b ≥ x))

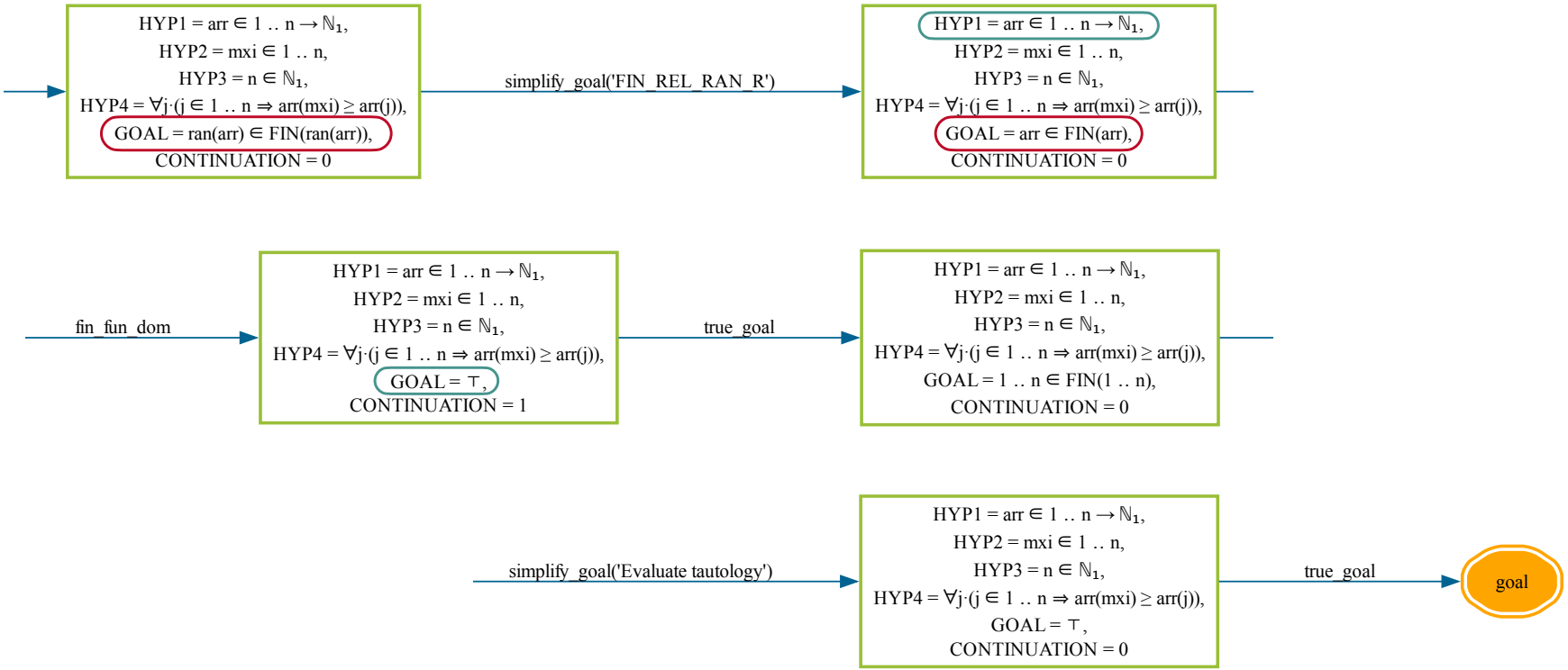
25. upper_bound_1

arr ∈ 1 .. n → ℕ₁
mxi ∈ 1 .. n
n ∈ ℕ₁
∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j))
---
ran(arr) ∈ FIN(ran(arr))

26. simplify_goal('FIN_REL_RAN_R')

arr ∈ 1 .. n → ℕ₁
mxi ∈ 1 .. n
n ∈ ℕ₁
∀j·(j ∈ 1 .. n ⇒ arr(mxi) ≥ arr(j))
---
arr ∈ FIN(arr)

Trace Visualisation



Again: Interactive Trace Replay

- Can be used to refactor a sequence of proof steps
  - Replace and skip rules / parts of the proof

- Example: replace `FUN_GOAL` by manual proof steps:

$$\frac{arr \in 1 \mathinner{..} n \rightarrow \mathbb{N}_1 \quad \ldots}{arr \in 1 \mathinner{..} n \nrightarrow \mathbb{N}_1}$$

`DEF_IN_TFCT`

$$\frac{arr \in 1 \mathinner{..} n \nrightarrow \mathbb{N}_1 \land dom(arr) = 1 \mathinner{..} n \quad \ldots}{arr \in 1 \mathinner{..} n \nrightarrow \mathbb{N}_1}$$

Step	Transition	Replayed Transition	Precision
1	start_xtl_system	start_xtl_system	Precise
2	and_r	and_r	Precise
3	and_r	and_r	Precise
4	and_r	and_r	Precise
4		simplify_hyp('DEF_IN_TFCT')	Manual
4		and_l()	Manual
4		hyp()	Manual
5	fun_goal	SKIP	
6	simplify_hyp	simplify_hyp	Precise
7	and_l	and_l	Precise
8	**eq**	**eq**	**Precise**
9	hyp		

Conclusion and Outlook

Heinrich Heine University Düsseldorf

- Implemented a subset of Rodin proof rules in Prolog
  - Simple POs generated by Rodin can already be proven
  - (Graph) visualisation and HTML export for traceability
  - *still in development*

- There are a lot of ideas for future work:
  - Implement more rules
  - Allow to recognise user input (add hypotheses, instantiations of $\exists, \forall$)
  - Allow to add custom proof rules
  - More advanced visualisation using VisB, improving user interaction
  - Apply automatic provers as proof step (as in Rodin), or even try model checking

- More ideas, inspirations, questions?