# Interactive Proving with ProB

Katharina Engels, Jan Gruteser, and Michael Leuschel

Faculty of Mathematics and Natural Science, Institute of Computer Science,
Heinrich Heine University Düsseldorf, Universitätsstr. 1, D-40225 Düsseldorf
{katharina.engels,jan.gruteser,michael.leuschel}@hhu.de

ProB [5] is a tool for validating formal specifications and supports model checking and animation for B and Event-B models. ProB can also be used to prove Event-B proof obligations (PO) using its disprover [3]. Currently, an interactive proof feature for POs exported from Rodin is being developed, using the ProB animator and a Prolog specification of proof rules. It enables users to construct proofs for formal models step by step. The goal is to improve the traceability and understanding of proofs, which can be useful in educational settings where students are learning formal methods and how to mathematically verify the correctness of a system.

For our implementation, we make use of Rodin's inference and rewrite rules[1], but also of rules from [1], and specify them in Prolog. This turns the mathematical definitions into executable Prolog rules, that can be used in a prover. Moreover, by targeting ProB's XTL interface[2] we can turn the Rodin proof rules into a labelled transition system, with sequents as states and applications of proof rules as transitions between sequents. The core is a definition of a ternary transition predicate `trans(Label,StateBefore,StateAfter)`:

```
trans(simplify_goal(Rule),sequent(Hyps,Goal,Cont),
   sequent(Hyps,NewGoal,Cont)) :- simp_rule(Goal,NewGoal,Rule).
trans(imp_r,sequent(Hyps,implication(G1,G2),Cont),
   sequent(Hyps1,G2,Cont)) :- add_hyp(G1,Hyps,Hyps1). [...]
simp_rule(member(X,SetA),equal(X,A),'SIMP_IN_SING') :-
      singleton_set(SetA,A).
```

Using ProB's XTL mode allows us to *animate* the proof of a PO, with each animation step corresponding to the application of a proof rule. We can also use ProB's model checker to search for proofs of a PO, and use ProB's visualisation features to display proofs. Initially, all POs of a machine exported from Rodin are available as start transitions. For this, we use the export of POs in Prolog format created by the ProB disprover. The Prolog representation is then normalised according to ProB's WD prover [4]. By using the same format as the WD prover, we can later integrate the proof rules into the ProB core.

Figure 1 shows ProB2-UI [2] with an example PO of a traffic light system coordinating pedestrians and cars that needs to be proven. The state view lists the hypotheses and the goal that can be proven with a set of already implemented Rodin proof rules, displayed as transitions to the left of the state

[1] https://wiki.event-b.org/index.php/Inference_Rules,
https://wiki.event-b.org/index.php/All_Rewrite_Rules

[2] See https://prob.hhu.de/w/index.php?title=Other_languages.

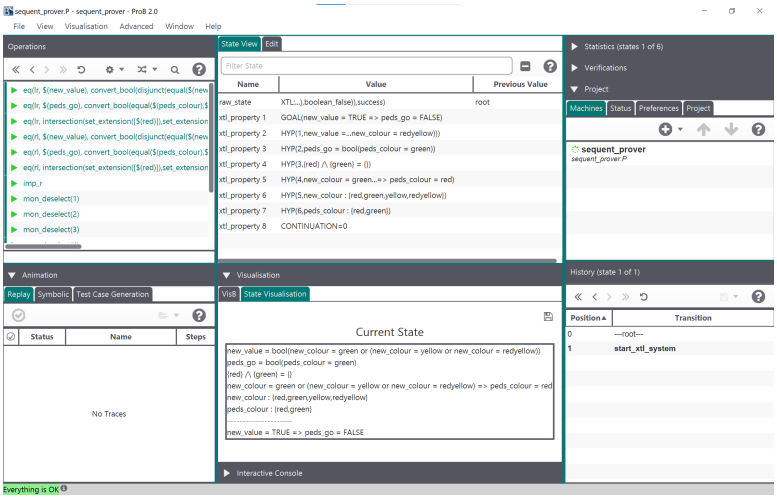


Fig. 1: General Overview of ProB2-UI with Current Hypotheses and the Target Goal, Applicable Proof Rules and State Visualisation

view. A visualisation of the current and previous proof sequent is provided as well. In addition, it is possible to export the proof steps into a stand-alone HTML file for later review (cf. Fig. 3).

Future work will include the ability to export the replayed trace (i.e. the applied rules as on the right in Fig. 2) in a format that can be re-imported later, allowing users to resume their work. Currently, it is not yet possible to incorporate user input for rules, which is important for existential or universal quantifiers. Taking user input into account can also be used to add hypotheses or custom proof rules, making the proof process more dynamic and increasing the level of interactivity. It might also be interesting to use the ProB model checker to automatically find a sequence of proof rules that prove the goal. Further ideas for future development include improving the visualisation to make the proof clearer and improve the user experience by linking the proof rules to clickable elements, as in the Rodin proof view.

## References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. J. Bendisposto, D. Geleßus, Y. Jansing, M. Leuschel, A. Pütz, F. Vu, and M. Werth. ProB2-UI: A Java-based User Interface for ProB. In *Proceedings FMICS*, LNCS 12863, pages 193–201, 2021.
3. S. Krings, J. Bendisposto, and M. Leuschel. From Failure to Proof: The ProB Disprover for B and Event-B. In *Proceedings SEFM 2015*, volume 9276 of *LNCS*, pages 199–214. Springer, 2015.
4. M. Leuschel. Fast and Effective Well-Definedness Checking. In *Proceedings iFM 2020*, volume 12546 of *LNCS*, pages 63–81. Springer, 2020.
5. M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.

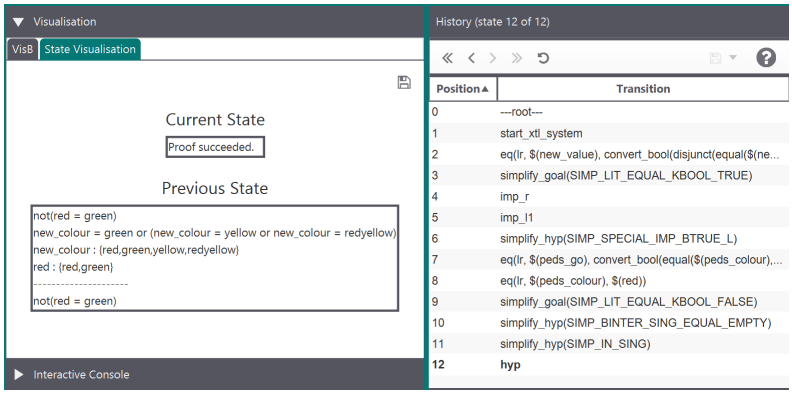


Fig. 2: Visualisation of Current and Previous State and Overview of the Applied Proof Rules in ProB2-UI

**9. simplify_goal(SIMP_LIT_EQUAL_KBOOL_FALSE)**

```
new_colour = green or (new_colour = yellow or new_colour = redyellow)
                {red} /\ {green} = {}
        new_colour : {red,green,yellow,redyellow}
                red : {red,green}
              ----------------------
                not(red = green)
```

**10. simplify_hyp(SIMP_BINTER_SING_EQUAL_EMPTY)**

```
                not(red : {green})
new_colour = green or (new_colour = yellow or new_colour = redyellow)
        new_colour : {red,green,yellow,redyellow}
                red : {red,green}
              ----------------------
                not(red = green)
```

**11. simplify_hyp(SIMP_IN_SING)**

```
                not(red = green)
new_colour = green or (new_colour = yellow or new_colour = redyellow)
        new_colour : {red,green,yellow,redyellow}
                red : {red,green}
              ----------------------
                not(red = green)
```

**12. hyp**

```
Proof succeeded.
```

Fig. 3: An Excerpt of the HTML Export of the State Visualisations and Applied Proof Rules