ProB: Harnessing the Power of Prolog to Bring Formal Models and Mathematics to Life

Michael Leuschel and STUPS Group

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf
michael.leuschel@hhu.de

ProB is an animator, model checker and constraint solver for high-level formal models. On the practical side, it can be used to ensure the safety of critical systems. On the theoretical side, it strives to bring formal models and mathematics to life. Formal modelling languages like B, Z or TLA+ provide a mathematical approach to software and systems development. These languages build on logic and set theory to enable convenient modelling of many safety critical systems and algorithms. Their use is not confined to the academic world, e.g., Alstom's U400 product alone based on B is running on over 100 metro lines and has 25% of the worldwide market share [1]. These high-level models are transformed into executable software using a process based on proof, manual refinement and low-level automatic code generators. Early validation of such models used to be a challenge though; some researchers used to argue that high-level models should not be executable [7].

This was the initial motivation for the development of ProB [8,9], a tool to make high-level formal models executable. The challenge was overcome using the power of Prolog,[1] in particular its co-routining and constraint solving capabilities. ProB thereby enables users to bring their formal models to life, making it possible to detect issues very early in the development process. In particular, ProB enables domain experts, with no education in formal methods, to provide feedback based on their unique expertise. Over the years, the performance of ProB has continually increased and was recently used [6] for the first time to execute a high-level formal model in real-time to control trains (cf., Fig. 1). ProB also provides many visualisation and verification techniques built on top of its animation capabilities. Some of these features have been certified for use in safety-critical applications according to the European norm EN 50128. In particular, ProB has been used by companies like Siemens, Alstom, ClearSy and Thales, to validate the safe configuration of train systems all over the world (e.g., Paris metro line 1, São Paulo, Alger, Barcelona, Mexico, Toronto) [1].

ProB has also been used by many academics in teaching and research. It has been cited over 1000 times[2] and is itself being used in several other tools, ranging

[1] The letters "Pro" in the name ProB also makes allusion to its reliance on Prolog.

[2] According to Google Scholar as of August 31st, 2022 there are 801 citations for [8], and 455 citations for its journal version [9]. In addition, there are considerable number of articles which use ProB but are not taken into account by Google Scholar, as they cite only the web page or mention the tool's name.

from domain specific modeling tools (Meeduse, Coda), university course planning (Plues), railway validation (SafeCap, DTVT, Olaf, ClearSy Data Solver, Dave, Ovado), security analyses (B4MSecure, VTG), UML modeling (iUML, UseCasePro), test case generation (BTestBox, Cucumber-Event-B).

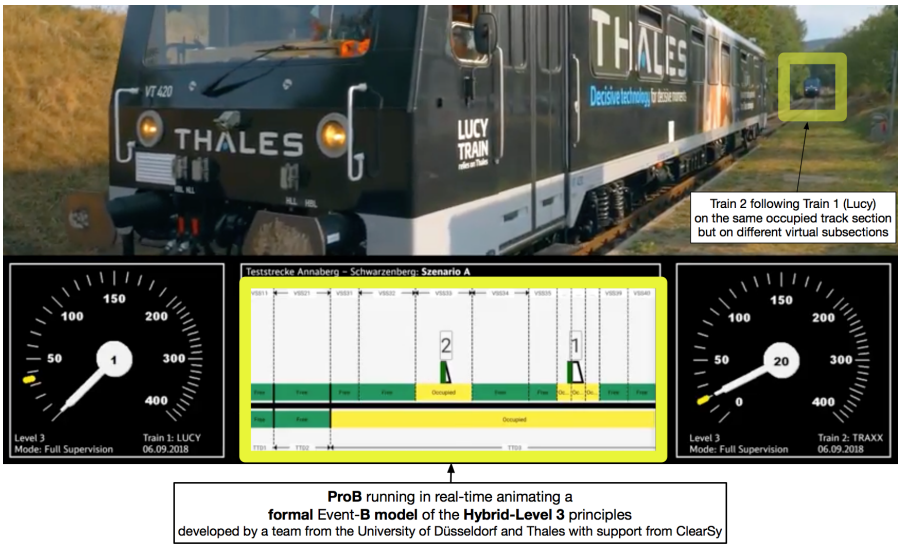

Fig. 1. A video of DB `https://www.youtube.com/watch?v=FjKnugbmrP4` with ProB running a formal B model in real-time to control two trains and demonstrate the ETCS Hybrid Level 3 concepts

Implementation ProB has been developed over around 20 years and was initially developed in sicstus Prolog. The core of ProB consists of about 400 Prolog files containing over 150,000 lines of code along with a large test suite of almost 7 000 unit tests and more than 2 000 integration tests. This strict testing regime is important for certification. For end users, ProB provides multiple user interfaces: a command-line interface (probcli) for batch verification and data validation, a Jupyter kernel [3] for notebook usage, a Java API, and a set of GUIs (Tcl/Tk, JavaFX) for interactive animation, visualisation, and verification. All of these interfaces share the same Prolog core.

Although development began over 20 years ago, ProB is still actively developed. It is certainly a challenge to keep an academic tool in development for that amount of time; but the task was eased by the robustness of Prolog and the excellent support provided by sicstus.

ProB has recently [4] been made compatible with SWI Prolog and We are working to make it compatible with Logtalk and other systems like Ciao Prolog.

Challenge Almost all of ProB's features, from animation to verification, require constraint solving at the level of the underlying formal language, i.e, for an undecidable mathematical language with existential and universal quantification, higher-order sets, functions and relations and unbounded variables.

Moreover, ProB generally needs not just to find one solution for a predicate, but all solutions, e.g., when evaluating a set comprehension. Similarly, for model checking it needs to find all possible transitions to ensure that the complete state space is verified.

Below we illustrate this with a simple predicate evaluated using ProB's Jupyter notebook interface. Observe, how the variable f is automatically recognised as a (higher-order) infinite function and kept symbolic, while the set comprehension *res* of all square roots of 100 is computed explicitly. Also note that $f(2)$ denotes the (also infinite) squaring function and that all solutions for *res* are computed, even though no finite bounds were provided for x.

```
In [26]: f = λe.(e∈ℕ | λbase.(base∈ℤ | base**e)) ∧ res={x|f(2)(x) = 100}
Out[26]: TRUE
```

Solution:

- $res = \{-10, 10\}$
- $f = /* @symbolic * /\lambda e \cdot (e \in \mathbb{N} \mid /* @symbolic * /\lambda base \cdot (base \in \mathbb{Z} \mid base\hat{}e))$

The core of ProB solves these challenges by using co-routining (i.e., building on the independence of the computation rule of logic programming) to build a constraint solver on top of CLP(FD) [2]. In particular, the kernel of ProB contains various specific solvers, e.g., for booleans, integers, sets, relations, functions, sequences, which communicate via reification and co-routines (see [5]).

Formal languages like B, especially when using Unicode syntax, are very close to the way we write mathematics in text books and scientific articles. For example, to colour a graph g with nodes N using four colours we simply stipulate the existence of a colouring function in B: $col \in N \to 1..4 \wedge \forall (x, y).(x \mapsto y \in g \Rightarrow col(x) \neq col(y))$. This makes it possible to copy parts of, e.g., theoretical computer science books into Jupyter enabling students to bring the mathematical definitions to life with ProB [3].

Summary ProB is an animator, model checker and constraint solver for high-level formal models. It has been developed over around 20 years and has harnessed the power of Prolog to help users develop safe systems. ProB takes away tedious choices, automatically detects subtle bugs, but still leave users in control to interactively validate their models. A friendly user experience was always more relevant for ProB than raw benchmark figures. For example, ProB will catch overflows, deal with divisions by zero, and keep track of source level information to visualise and explain errors to end users. We hope that we can keep on improving ProB and that we have not yet reached the end of what Prolog and formal methods have to offer. Indeed, we want to drive the idea of formal models as runtime artefacts further (see Fig. 1). The capabilities of the constraint solver of course still be improved; maybe we can one day reach a state of a human friendly "executable mathematics" language which can be used by novice and expert alike. Safety will also play a crucial role in the future, in particular with the increased use of artificial intelligence in autonomous systems. Here, within the ongoing KI-LOK research project we are striving to certify safety critical train systems using AI components with the aid of ProB.

A Appendix: Team

The first version of ProB was written by Michael Leuschel while in Southampton. During that time Michael Butler, Edd Turner and Laksono Adhianto provided valuable contributions. The first article on ProB was published with Michael Butler at FM'2003 [8]. In 2005 the development moved to Düsseldorf and the STUPS group. Over the years many people from the STUPS have contributed to ProB. In alphabetical order these persons are: Jens Bendisposto, Carl Friedrich Bolz, Joy Clark, Ivo Dobrikov, Jannik Dunkelau, Nadine Elbeshausen, Fabian Fritz, Marc Fontaine, David Gelessus, Stefan Hallerstede, Dominik Hansen, Christoph Heinzen, Michael Jastram, Philipp Körner, Sebastian Krings, Lukas Ladenberger, Li Luo, Thierry Massart, Daniel Plagge, Antonia Pütz, Kristin Ruthenkolk, Mireille Samia, Joshua Schmidt, David Schneider, Corinna Spermann, Yumiko Takahashi, Fabian Vu, Michelle Werth, Dennis Winter.

B Appendix: Proposed Citation

The first conference article about ProB is [8]; its journal version is [9].

References

1. M. J. Butler, P. Körner, S. Krings, T. Lecomte, M. Leuschel, L. Mejia, and L. Voisin. The first twenty-five years of industrial use of the B-method. In M. H. ter Beek and D. Nickovic, editors, *Proceedings FMICS 2020*, LNCS 12327, pages 189–209. Springer, 2020.
2. M. Carlsson, G. Ottosson, and B. Carlson. An Open-Ended Finite Domain Constraint Solver. In H. G. Glaser, P. H. Hartel, and H. Kuchen, editors, *Proceedings PLILP'97*, LNCS 1292, pages 191–206. Springer-Verlag, 1997.
3. D. Gelessus and M. Leuschel. ProB and Jupyter for logic, set theory, theoretical computer science and formal methods. In A. Raschke, D. Méry, and F. Houdek, editors, *Proceedings ABZ 2020*, LNCS 12071, pages 248–254, 2020.
4. D. Geleßus and M. Leuschel. Making ProB compatible with SWI-Prolog. *Theory Pract. Log. Program.*, 22(5):755–769, 2022.
5. S. Hallerstede and M. Leuschel. Constraint-based deadlock checking of high-level specifications. *Theory Pract. Log. Program.*, 11(4–5):767–782, 2011.
6. D. Hansen, M. Leuschel, P. Körner, S. Krings, T. Naulin, N. Nayeri, D. Schneider, and F. Skowron. Validation and real-life demonstration of ETCS hybrid level 3 principles using a formal B model. *STTT*, 22(3):315–332, 2020.
7. I. Hayes and C. B. Jones. Specifications are not (necessarily) executable. *Softw. Eng. J.*, 4(6):330–338, Nov. 1989.
8. M. Leuschel and M. Butler. ProB: A model checker for B. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: Formal Methods*, LNCS 2805, pages 855–874. Springer-Verlag, 2003.
9. M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.