Proceedings of the 12th Rodin User and Developer Workshop, 2025

June 10th, 2025
Düsseldorf, Germany

Editors:

Asieh Salehi Fathabadi	University of Southampton
Laurent Voisin	Systerel
Neeraj Kumar Singh	University of Toulouse
Michael Leuschel	Heinrich-Heine-Universität
Son Hoang	University of Southampton

List of logos

Contents

Part I

Summary

Executive Summary

Event-B is a formal method for system-level modelling and analysis. The Rodin platform is an Eclipse-based toolset for Event-B that provides effective support for modelling and automated proof. The platform is open source and is further extendable with plug-ins. A range of plug-ins has already been developed including ones that support animation, model checking, UML-B and text editor. While much of the development and usage of Rodin takes place within past and present EU/UK-funded projects, there is a growing group of users and plug-in developers outside these projects.

The purpose of the 12th Rodin User and Developer Workshop was to bring together existing and potential users and developers of the Rodin toolset and to foster a broader community of Rodin users and developers. For Rodin, the workshop provided an opportunity to share tool experiences and to gain an understanding of ongoing tool developments. For plugin developers, the workshop provided an opportunity to showcase their tools and to achieve better coordination of tool development effort.

The one-day programme consisted of presentations on tool development and tool usage. The presentations are delivered by participants from academia and industry. This volume contains the abstracts of the presentations at the Rodin workshop on June 10th, 2025. The presentations are also available online at https://wiki.event-b.org/index.php/Rodin_Workshop_2025.

The workshop was co-located with the ABZ 2025 conference, Düsseldorf, Germany. The Rodin Workshop was supported by the University of Southampton.

Finally, we would like to thank the contributors and participants, the most important part of our successful workshop.

Organisers

Asieh Salehi Fathabad, University of Southampton
Laurent Voisin, Systerel
Neeraj Kumar Singh, University of Toulouse
Michael Leuschel, Heinrich-Heine-Universität
Son Hoang, University of Southampton

Programme of the Rodin Workshop 2025

09:00–11:05

- Rodin 3.10 and its plug-ins: ***Guillaume Verdier****, Laurent Voisin and Idir Ait-Sadoune*
- Constructing an Event-B Model using Promise-Driven Modeling: ***Felix Schaber***
- Verification of Event-B proofs throught their translation to Lambdapi: ***Anne Grieu*** *and Jean-Paul Bodeveix*
- Interactive Trace Replay for Event-B Models: ***Jan Gruteser*** *and Michael Leuschel*
- Interactive Proving with ProB: *Katharina Engels,* ***Jan Gruteser*** *and Michael Leuschel*

11:05–11:20 Break

11:20–13:00

- Minimal Bad Sequence on Quasi-Orders: ***Dominique Cansell***
- Project Allocation with Event-B and ProB: ***Thai Son Hoang****, Abdolbaghi Rezazadeh and Michael Butler*
- EB[ASTD]: Meta-modelling framework for ASTD: ***Christophe Chen****, Peter Rivière, Neeraj Kumar Singh, Guillaume Dupont, Yamine Ait Ameur and Marc Frappier*
- Extending EB4EB for Parameterised Events: ***Peter Rivière****, Neeraj Kumar Singh, Guillaume Dupont, Yamine Ait Ameur*

Part II

Contributions

Rodin 3.10 and its plug-ins

Guillaume Verdier[1], Laurent Voisin[2], Idir Ait-Sadoune[3]

[1] Toulouse INP, IRIT
guillaume.verdier@irit.fr
[2] Systerel
laurent.voisin@systerel.fr
[3] Paris-Saclay University, CentraleSupélec, LMF
idir.aitsadoune@centralesupelec.fr

1 Introduction

The Rodin platform [1] is an integrated development environment for designing software with Event-B [2]. Based on Eclipse, it is designed to be extensible with plug-ins. Thanks to support from the French ANR project Event-B Rodin Plus (EBRP, ANR-19-CE25-0010), Rodin and many of its plug-ins are actively updated. We present the evolutions of the platform since ABZ 2024.

2 Rodin 3.10

As usual, some new proof rules have been added:

- a rewrite rule for direct product: $(f \otimes g)(x) \equiv f(x) \mapsto g(x)$
- a rewrite rule for parallel product: $(f \parallel g)(x) \equiv f(prj1(x)) \mapsto g(prj2(x))$
- an inference rule on bounds of upto that deduces $a = c \wedge b = d$ from a hypothesis $a..b = c..d$, provided that $a \leq b$

The *abstract expression* tactic has been extended again. Now, in addition to a single name, users can also do simple pattern matching:

- with maplets such as $a \mapsto b = e$; the patterns can be arbitrarily complex, for example $(a \mapsto (b \mapsto c)) \mapsto (d \mapsto e) = x$;
- with a datatype constructor if its datatype only has one constructor; this is particularly useful for record-like datatypes: given a record like $R = C(a \oftype \mathbb{Z}, b \oftype \mathbb{Z}, c \oftype BOOL)$, one can provide as input for *ae* expressions like $C(x, y, z) = r$.

The *oftype* operator could only be applied to atomic expressions. It can now also be applied to extensions that cannot be typed by themselves (typically datatype constructors and operators from the Theory plug-in). For instance, $\text{Either}(A, B) = \text{Left}(a \oftype A) \mid \text{Right}(b \oftype B)$ was unusable: given $\text{Left}(x)$, type parameter A could be deduced from the type of x, but B could not be inferred and conversely for Right. Now, one can use *oftype*, e.g., $\text{Left}(0) \oftype \text{Either}(\mathbb{Z}, BOOL)$.

Peter Riviere found a breaking bug in the translation of datatypes to set theory for SMT provers. Luckily, this bug seems to have a rather limited impact: it appears that the only SMT prover that could use that erroneous translation to prove false is Alt-Ergo, which is not installed by default. Only those who manually added Alt-Ergo and used the Theory plug-in were potentially affected by this issue.

Finally, miscellaneous issues have fixed, particularly to make Rodin a bit more user-friendly:

- a warning is now displayed for expressions matching $\exists X \cdot P \Rightarrow Q$: an implication in an existentially quantified predicate is typically a mistake;
- external provers are now checked several times at startup if they seem to not be working; this will prevent error messages about provers failing due to a timeout: it was caused by a slow startup, but provers actually worked well after the first (slow) execution;
- the text area input in Proof Control is now always saved in the history and cleared when a tactic is executed (the behaviour used to depend on the type of tactic applied);
- the tactic profile dialog has been fixed: one of the parts was too narrow and could only be seen after manually resizing the window.

3 Plug-ins

3.1 Atelier B plug-in

In collaboration with Clearsy, we released two new versions of the Atelier B plug-in for Rodin. Version 2.4.0 updated the provers to include those from the latest release (24.04.2) of Atelier B. Version 2.4.1 fixed a breaking bug with some Chinese editions of Windows.

3.2 SMT provers plug-in

The SMT provers plug-in is currently being updated. The release will feature:

- updated provers (CVC4 from version 1.5 to 1.8 and Z3 from version 4.4.1 to 4.14.1);
- a new prover, CVC5 (version 1.2.1);
- Apple Silicon builds of the provers, in addition to the 64-bit Intel ones.

3.3 Compatibility updates

Although Eclipse and Rodin offer a very stable platform for plug-in development, some very old plug-ins ended up not working with the latest Rodin releases. The following plug-ins have been updated to work with recent Rodin releases:

- B2Latex (release 0.8)
- Renaming Refactory (release 1.4.0)
- Generic Instantiation (Soton) (release 1.1.0)

4 Conclusion

Rodin is under active development and new versions are released yearly. The development team is also updating many plug-ins to ensure that they keep working with new versions of Rodin.

References

[1] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. *Rodin: an open toolset for modelling and reasoning in Event-B.* International Journal on Software Tools for Technology Transfer, 12(6):447–466, Nov 2010.

[2] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, New York, NY, USA, 1st edition, 2010.

Constructing an Event-B Model using Promise-Driven Modeling *

Felix Schaber[1,2]

[1]Hitachi Rail, Austria
[2]Heinrich Heine University Düsseldorf, Germany

Abstract

This workshop presentation describes the construction of an Event-B model for a novel type of train protection system, called the moving block system, which is currently being investigated as part of Europe's Rail Joint Undertaking. The moving block system allows for more dynamicity than the fixed block approach traditionally used in railways. The Event-B model is constructed in Rodin using promise-driven modeling. The core idea behind promise-driven modeling is to prioritize modeling those parts of the system requirements that are least likely to change in later stages, hence supporting early validation through animation. Experiences and lessons learned when constructing the Event-B model of the moving block system using promise-driven modeling are described.

1 Introduction to Promise-Driven Modeling

It is well known that errors discovered in the early stages of system development are significantly less costly to fix than those found later [1]. Modeling is often promoted as a way to catch errors early, thereby reducing overall costs. However, as system complexity grows, so does the complexity of the model.

Consequently, errors introduced in early modeling stages can also become increasingly expensive. Getting this right on the first attempt is a known challenge, particularly for cyber-physical systems, where it is often unclear which part of the system should be modeled first and at what level of detail.

When the system is safety-critical, the model must also support reasoning about safety properties. To address this, we use System Theoretic Process Analysis (STPA) to decompose the system into individual controllers and identify the safety constraints associated with each controller.

This presentation introduces promise-driven modeling as a solution to the challenges mentioned above.

Promise-driven modeling is based on the principle that behaviors least likely to change during model evolution are modeled first. Prioritizing design decisions by their likelihood of change is a generally considered best practice [2].

*Partly funded by the European Union Grant Agreement # 101102001.

This reduces the risk of discovering the need for high-level changes late in the modeling process. High-level changes often ripple through the refinement chain, making them resource-intensive.

The promise-driven modeling approach is used to construct an Event-B [3] model in Rodin [4]. The model is visualized in ProB2-UI using VisB [5]. This allows experts to validate model behavior, even with no prior experience with or knowledge of Event-B.

2 Moving Block System

The moving block system (MBS) controls the movement authorities (MAs) sent to trains [6]. An MA limits how fast and how far a train may run safely. These MAs are enforced by an on-board-unit (OBU) on the train. The OBU calculates the braking curve and enforces the onset of braking if the train driver brakes too late to keep the train within the limits of the MA.

The MAs are proposed to MBS from an external system. MBS can decide to grant or reject the proposal for an MA. Only granted MAs are sent to the train. MAs contain mode profiles describing the responsibility split between MBS, OBU, and the train driver for avoiding collisions. For this workshop, we'll focus on the full-supervision European Train Control System (ETCS) mode, where the MBS is solely responsible for ensuring that the track is and stays clear of trains and obstacles (known at the time of the request). The OBU can request a new MA from the MBS (MA request).

The OBU sends train data (TrainData) to MBS, estimates the physical train position, and periodically sends train position reports (TPRs). Trackside train detection systems (TTDs), installed at fixed sections along the tracks, also detect physical train presence, and information about train presence is sent to MBS.

The Event-B Model model for a selected part of this system is constructed using promise-driven modeling.

3 Conclusion

Using promise-driven modeling, an Event-B model for the moving block system was constructed and the associated Proof Obligations were discharged. During the modeling process, a number of open points and potential gaps were discovered. Animation in ProB2-UI using VisB allowed to discuses associated behaviors with domain experts who had no previous exposure to Event-B or formal modeling.

The promise log contained all promises from which the Event-B model was constructed, closely linking Event-B model with the expected behavior described by the promises. For unexpected model behavior, this promise log was helpful for fostering a discussion between domain experts, who are typically familiar with descriptions of behavior than formal modeling details.

References

[1] N. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety.* 2012.

[2] Nancy G. Leveson. "Design and Assurance of Control Software". In: *IEEE Transactions on Software Engineering* 51.3 (Mar. 2025), pp. 666–672. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2025.3539975. URL: https://ieeexplore.ieee.org/document/10877915/ (visited on 03/30/2025).

[3] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering.* Cambridge University Press, 2010. DOI: 10.1017/CBO9781139195881.

[4] Jean-Raymond Abrial et al. "Rodin: An Open Toolset for Modelling and Reasoning in Event-B". In: *International journal on software tools for technology transfer* 12 (2010), pp. 447–466.

[5] Michelle Werth and Michael Leuschel. "VisB: A Lightweight Tool to Visualize Formal Models with SVG Graphics". In: *Rigorous State-Based Methods.* Ed. by Alexander Raschke, Dominique Méry, and Frank Houdek. Vol. 12071. Cham: Springer International Publishing, 2020, pp. 260–265. DOI: 10.1007/978-3-030-48077-6_21.

[6] Nina D. Versluis et al. "Real-Time Railway Traffic Management under Moving-Block Signalling: A Literature Review and Research Agenda". In: *Transportation Research Part C: Emerging Technologies* 158 (Jan. 2024), p. 104438. DOI: 10.1016/j.trc.2023.104438.

Verification of Event-B proofs through their translation to Lambdapi

Anne Grieu[1] and Jean-Paul Bodeveix[1]

[1]IRIT, INP, Université de Toulouse, CNRS, Toulouse, France , prenom.nom@irit.fr

Rodin Workshop - June 2025 - Düsseldorf

In order to verify formally the Event-B[1, 2] proofs made by Rodin[3], we are working on an embedding of its mathematical language and proof system in Dedukti/Lambdapi [7, 12]. This logical framework, based on the ΛΠ-calculus modulo, allows implementing various logics and is already used to enable interoperability between proof systems[5, 14]. We use Dedukti as a target-framework to work, in addition, on interoperability of proof systems from various other set-theory, like B and its framework Atelier B[8, 13], TLA+ and its framework TLAPS[4, 10][1].

For this purpose, instead of using a specific library to express the mathematical logic of Event-B[6], we use a library, called `lambdapi-stdlib`[2], of which development is still in progress. It provides basic components to support various logics (propositional and first order logic, HOL, CoC, ...). Using parts of the `lambdapi-stdlib`, we can express the mathematical language of Event-B. On top of this library, we define in Lambdapi the type language of Event-B. Each specific operator of Event-B is defined with its signature and rewriting rules to define their behaviour. Then, we state and prove theorems that will be useful to translate the proofs[11]. To structure the proofs, we introduce theorems, meta-theorems and proof terms generators to encode actions in the proofs made by Rodin (deduction rules, rewriting rules, tactics..).

We present a Rodin plug-in, still in development. It takes as input a Rodin bps file, that gives access to obligation proofs, thanks to Rodin API, and generates a `lambdapi` file to be checked by Lambdapi. The plug-in translates the context, the sets, its formulas, using type information given by Rodin (e.g. when you write $\forall x.x > 0$, the formula that will be translated in Lambdapi is $\forall x \circ_{\circ} Z$). It translates also the proofs generated by Rodin, following recursively the proof tree and generating a proof term for Lambdapi with the same structure of the Rodin proof tree. Each node matches with one or more rule applied to one or more hypothesis, that discharge a goal or create new goals to be discharged. Some translations are straightforward, but some others require more complex treatment. In the following we illustrate some of these translations. We use these notations:

- x is a variable, $P, P1, P2, P3, G$ are propositions,
- H_n is an identifier created by the plug-in to name the n^{th} hypothesis added in the context,
- `assume`, `apply`, `refine` are Lambdapi tactics[3],
- $\wedge_i$ is a term/theorem from lambdapi-stdlib

```
constant symbol ∧ᵢ [p q] :
    π p → π q → π (p ∧ q);
```

- `Or2Imp` is a term/theorem of our library:

```
symbol Or2ImpGoal [P Q: Prop] :
    π (((¬ P) ⇒ Q) ⇒ (P ∨ Q));.
```

[1]https://anr.fr/Projet-ANR-21-CE25-0015
[2]https://github.com/Deducteam/lambdapi-stdlib/blob/master/README.md
[3]https://lambdapi.readthedocs.io/en/latest/tactics.html

- Straightforward translation, using tactics:

$\forall$ goal (free x) : $\forall x \cdot P(x)$	$\rightsquigarrow$	`assume x;`	adds x in the context
$\Rightarrow$ goal : P $\Rightarrow G$	$\rightsquigarrow$	`assume` H_n`;`	adds a proof of P in the context
$\top$ goal : $\top$	$\rightsquigarrow$	`refine` $\top_i$`;`	discharges True goal with a proof of true

- Simple translation, applying theorems:

$\vee$ to $\Rightarrow$ in goal : P1 $\vee$ P2	$\rightsquigarrow$	`apply Or2ImpGoal;`	replaces the goal $P1 \vee P2$ by $\neg P1 \Rightarrow P2$
$\wedge$ goal : P1 $\wedge P2$	$\rightsquigarrow$	`refine` $\wedge_i$ `_ _{}{};`	creates 2 new sub-goals P1 and P2

➜ Some behaviours of Rodin needs a special treatment to be expressed in Lambdapi, with the help of the plug-in.

- N-ary operators:

  Some operators are n-ary in Rodin, we can't apply in a straightforward way a Lambdapi-theorem to get the same behaviour with the binary operators defined. For the conjunction of n propositions, for instance, the plug-in generates a composition of introduction rules that will create n new goals.

  $\wedge$ goal : P1 $\wedge P2 \wedge P3$ $\rightsquigarrow$ `refine (`$\wedge_i$ `[P1` $\wedge$ `(P2` $\wedge$ `P3)] _ (` $\wedge_i$ `[P2] [P3] _ _) ){}{}{};`

- Automated actions of Rodin, like elimination of duplicated propositions, are proved once and for all at a meta level in Lambdapi, in a proof-by-reflection approach and the plug-in will instantiate the generic theorem with a mapping function between the position of the propositions and the propositions, in the case of the elimination of duplicated terms and the operator considered by the application of the theorem.

- Call an external prover of Rodin:

  When some sub-goals are proved with the help of a call to an SMT or some other internal prover of Rodin, we handle them by using Zenon modulo[9]. The plug-in will send to Zenon modulo, thanks to a TPTP translation, the lemma to prove, then the obtained lambdapi proof is integrated to the whole proof.

The plug-in we are building is in fact a bridge between Rodin and Lambdapi. We have presented some features of the plug-in, that will be extended to take in account the new needs to include all characteristics of Event-B and Rodin, like a hundred of rules and include the WD. To experiment our embedding of Event-B in Lambdapi, we work on various examples, for instance, several versions of Cantor's theorem.

References

[1] Jean-Raymond Abrial. The B-book - assigning programs to meanings. Cambridge University Press, 1996.

[2] Jean-Raymond Abrial. Modeling in Event-B - System and Software Engineering. Cambridge University Press, 2010.

[3] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. STTT, 12(6):447–466, 2010.

[4] Coltellacci Alessio. Reconstruction of TLAPS proofs solved by verit in lambdapi. In Uwe Glässer, José Creissac Campos, Dominique Méry, and Philippe A. Palanque, editors, Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings, volume 14010 of Lecture Notes in Computer Science, pages 375–377. Springer, 2023.

[5] Ali Assaf, Guillaume Burel, Raphal Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Expressing theories in the λΠ-calculus modulo theory and in the Dedukti system. In TYPES: Types for Proofs and Programs, Novi SAd, Serbia, May 2016.

[6] Laurent Voisin Christophe Métayer. The Event-B mathematical language, 2009. `https://webarchive.southampton.ac.uk/deployeprints.ecs.soton.ac.uk/11/4/kernel_lang.pdf`.

[7] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the Lambda-Pi-Calculus Modulo. In Simona Ronchi Della Rocca, editor, Typed Lambda Calculi and Applications, pages 102–117, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[8] David Delahaye, Catherine Dubois, Claude Marché, and David Mentré. The BWare project: Building a proof platform for the automated verification of B proof obligations. In Proceedings of the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z - Volume 8477, ABZ 2014, page 290–293, Berlin, Heidelberg, 2014. Springer-Verlag.

[9] David Déharbe, Pascal Fontaine, Yoann Guyot, and Laurent Voisin. Integrating SMT solvers in Rodin. Science of Computer Programming, 94:130–143, 2014. Abstract State Machines, Alloy, B, VDM, and Z.

[10] Anne Grieu. From Event-B to lambdapi. In Silvia Bonfanti, Angelo Gargantini, Michael Leuschel, Elvinia Riccobene, and Patrizia Scandurra, editors, Rigorous State-Based Methods, pages 387–391, Cham, 2024. Springer Nature Switzerland.

[11] Anne Grieu and Jean-Paul Bodeveix. Encodage du langage mathématique d'Event-B dans lambdapi. In 36es Journées Francophones des Langages Applicatifs (JFLA 2025), Jan 2025, Roiffé, France, 2025.

[12] G. Hondet and F. Blanqui. The New Rewriting Engine of Dedukti. In Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction, Leibniz International Proceedings in Informatics 167, 2020.

[13] Claude Stolze, Olivier Hermant, and Romain Guillaumé. Towards Formalization and Sharing of Atelier B Proofs with Dedukti. working paper or preprint, January 2024.

[14] François Thiré. Sharing a library between proof assistants: Reaching out to the hol family. Electronic Proceedings in Theoretical Computer Science, 274:57–71, July 2018.

Interactive Trace Replay for Event-B Models

Jan Gruteser and Michael Leuschel

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

ProB [6] is an animator and model checker that can be used in particular for animation of Event-B models created with Rodin. Along with animation, ProB allows to store and replay traces, i.e. sequences of transitions, to validate the correctness of the model's behaviour at different stages of its development. In many cases, traces created for a previous or abstract version of a model can only be partially replayed (or not at all) due to breaking changes, e.g. newly introduced or replaced/renamed events and variables. Refactoring such traces after changes to complex formal models can be tedious.

To address this, we present an integration of interactive trace replay for the ProB tooling, namely for ProB2-UI [2] and the ProB Tcl/Tk interface.
This could be particularly beneficial in the following three scenarios:

Refinement of Traces. Insert new trace steps between two abstract steps by manual animation of new or refined events (cf. Fig. 1a).
Abstraction of Traces. Skip steps with unavailable concrete events (cf. Fig. 1b).
Trace Refactoring. Repair/refactor traces of complex models after major changes.

So far, ProB in its core itself features an "intelligent" trace replay that tries to resolve possible failure scenarios automatically during the replay. It first tries to replay a trace perfectly, matching operation names, parameters and variable values after each trace step. If this is not possible, the replay tries to soften the replay constraints, e.g., allowing to use other parameter values or different operation names (e.g., when an operation has been renamed). The replay can also skip trace steps where the original operation is unavailable. However, this automated replay is not always possible, and more insights are necessary to adapt the old trace for the new model.

This is why we combine the existing logic with the interactive replay so that conflicts that cannot be handled automatically can be resolved by the modeller.

This means that we have two states during an interactive replay: the current trace step and the state of the animator. Based on this, our implementation currently includes the following control options:

- *Replay* the next trace step using a matching transition, if possible
- *"Fast Forward"*: automatic replay until the next step cannot be replayed
- *Skip* the current trace step (this is always possible)
- *Add manual animation* steps anywhere in the trace using the animator
- *Undo* the last replayed transition (from manual animation or replayed)

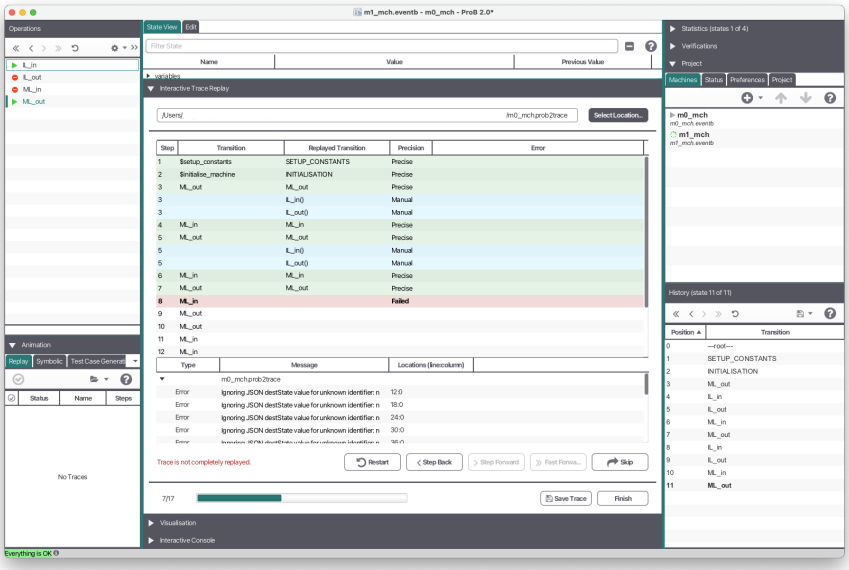
In the future, we would like to provide further options, such as restricting the precision or allowing the user to explicitly select the next replayed transition.

There has been research addressing the problem of trace refinement, like an algorithmic approach by Stock et al. [7]. There is also related work on refinement checking [4,5] and there are some similarities to the interface of the CODA simulator [3]. Also, the interactive manual animation is somehow related to interactive real-time simulation covered by SimB [8].

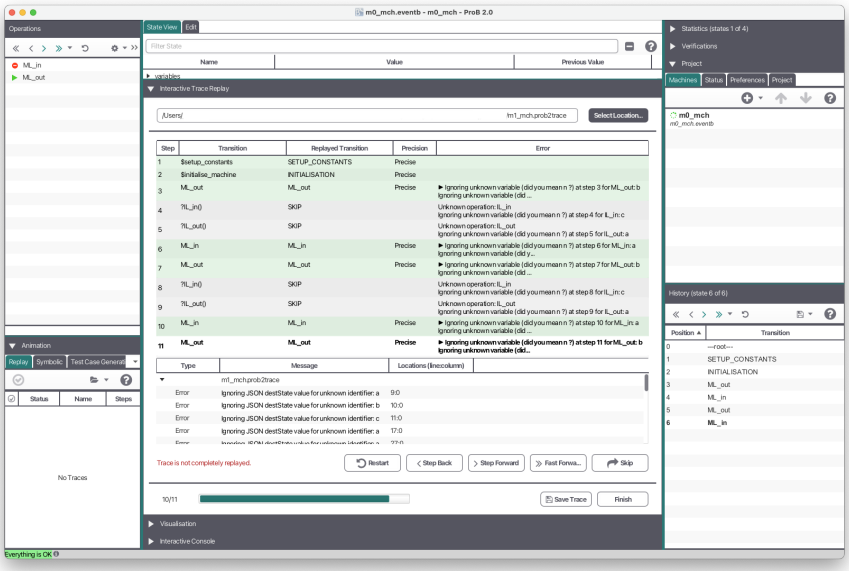
In summary, we believe that the interactive trace replay can be a useful extension to ProB, especially in the context of abstraction and refinement.

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. M. J. Butler, J. Colley, A. Edmunds, C. F. Snook, N. Evans, N. Grant, and H. Marshall. Modelling and refinement in CODA. In *Proceedings Refine@IFM 2013, Turku, Finland, 11th June 2013*, pages 36–51, 2013.
4. T. Gibson-Robinson, P. J. Armstrong, A. Boulgakov, and A. W. Roscoe. FDR3: a parallel refinement checker for CSP. *Int. J. Softw. Tools Technol. Transf.*, 18(2):149–167, 2016.
5. M. Leuschel and M. Butler. Automatic Refinement Checking for B. In K.-K. Lau and R. Banach, editors, *Proceedings ICFEM'05*, LNCS 3785, pages 345–359. Springer-Verlag, 2005.
6. M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
7. S. Stock, A. Mashkoor, M. Leuschel, and A. Egyed. Trace preservation in B and Event-B refinements. *Journal of Logical and Algebraic Methods in Programming*, 137:100943, 2024.
8. F. Vu and M. Leuschel. Validation of Formal Models by Interactive Simulation. In *Proceedings ABZ 2023*, volume 14010 of *LNCS*, pages 59–69. Springer, 2023.



(a) Refinement of a Trace



(b) Abstraction of a Trace

Fig. 1: Interactive Replay in ProB2-UI for "Controlling Cars on a Bridge" [1]

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.

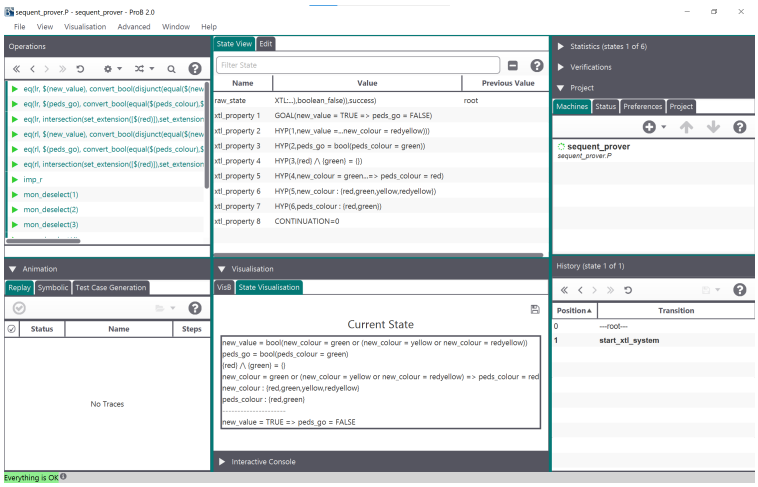


Fig. 1: General Overview of ProB2-UI with Current Hypotheses and the Target Goal, Applicable Proof Rules and State Visualisation

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.

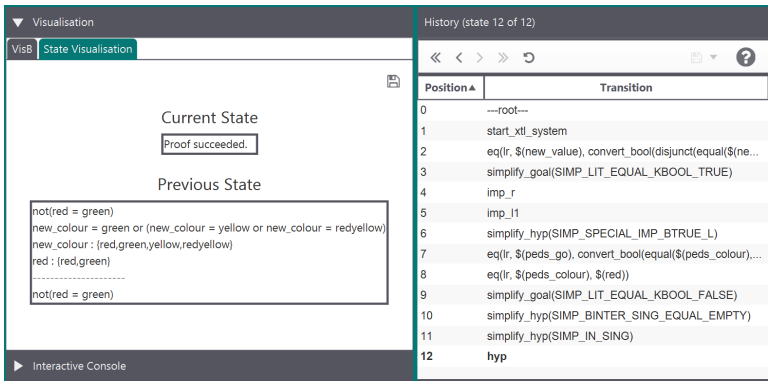


Fig. 2: Visualisation of Current and Previous State and Overview of the Applied Proof Rules in ProB2-UI

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

Minimal Bad Sequence on Quasi-Orders

Dominique Cansell (Lessy, EBRP)

1 Description

In [2] we presented the new JRA's instantiation context to define closure, fixpoint (Tarski), well-founded (Noether) and recursion. A new instantiation plugin [6] was developed in the EBRP project [7]. In this paper we present quasi-orders and well-quasi order and two important theorems on quasi-orders: the existence of a minimal bad sequences when the quasi-order is well-founded but not a well-quasi-order and the second theorem: the set of all values strictly less then value of the minimal bad sequence is well-quasi oder. Rodin [8] is used to develop and prove all theorems describe in this paper.

2 Definitions and some theorems

Let S_type a carrier set A quasi-order is a reflexive and transitive relation.

$qo = \{S \mapsto g | S \subseteq S\_type \wedge g \in S \leftrightarrow S \wedge g; g \subseteq g \wedge S \lhd id \subseteq g\}$

$wqo = \{S \mapsto g | S \mapsto g \in qo \wedge (\forall f \cdot f \in \mathbb{N} \to S \Rightarrow (\exists i, j \cdot i \geq 0 \wedge j > i \wedge f(i) \mapsto f(j) \in g))\}$

$sdc = (\lambda S \mapsto g . S \mapsto g \in qo | \{f | f \in \mathbb{N} \to S \wedge (\forall i, j \cdot i \geq 0 \wedge j > i \Rightarrow f(j) \mapsto f(i) \in g \setminus g^{-1})\})$

$wf = \{S \mapsto g | S \mapsto g \in qo \wedge sdc(S \mapsto g) = \varnothing\}$

$antichain = (\lambda S \mapsto g . S \mapsto g \in qo | \{A | A \subseteq S \wedge (A \times A) \cap g \subseteq id\})$

We have proved much theorems till Kruskal's one given in [3] like the Lemma1.3.2

$\forall S, g, A \cdot S \mapsto g \in wf \wedge A \subseteq S \Rightarrow$

$(\exists A_0 \cdot A_0 \in antichain(S \mapsto g) \wedge A_0 \subseteq A \wedge (\forall x \cdot x \in A \Rightarrow (\exists a \cdot a \in A_0 \wedge a \mapsto x \in g)))$.

We have used it to prove the following one (reformulation of the Lemma1.3.1 (existence of a minimum).

$\forall S, g, A \cdot S \mapsto g \in wf \wedge A \in \mathbb{P}\,1(S) \Rightarrow (\exists m \cdot m \in A \wedge (\forall z \cdot z \in A \wedge z \mapsto m \in g \Rightarrow m \mapsto z \in g))$

We have used our $FrSB$ operator [2] to define general recursive function from $\mathbb{N}$ to S_type First we instantiate $FrSB$ with $S, B := \mathbb{N}, S\_type$. $\{i \mapsto j | i \geq 0 \wedge i < j\}$ is a well-founded relation on $\mathbb{N}$. Let g be a a function such that: $g \in (\mathbb{N} \times (\mathbb{N} \nrightarrow S\_type)) \to S\_type$ .There is a unique total function fr: $fr \in \mathbb{N} \to S\_type$ such that we have: $\forall n \cdot n \in \mathbb{N} \Rightarrow fr(n) = g(n \mapsto 0..n-1 \lhd fr)$ The value of fr at n depends on its value on the set $0..n-1$, $FrSB$ is a function (an operator) which gives the recursive fonction fr: $fr = FrSB(\{i \mapsto j | i \geq 0 \wedge i < j\} \mapsto g)$

3 Bad sequence

Let $S \mapsto g$ be in qo a bad sequence bs is a function from $\mathbb{N}$ to S where $\forall i, j \cdot i \geq 0 \wedge j > i \Rightarrow bs(i) \mapsto bs(j) \notin g$. Remark: if $S \mapsto g$ not in wqo the set of bad sequence is not empty. Let BS be the set of bad sequence on S.

bs is minimal if

$\forall n, f \cdot n \geq 0 \wedge f \in \mathbb{N} \to S \Rightarrow \wedge 0..n-1 \lhd f = 0..n-1 \lhd bs \wedge f(n) \mapsto bs(n) \in g \setminus g^{-1}$

$\Rightarrow$

$(\exists i, j \cdot i \geq 0 \wedge j > i \wedge f(i) \mapsto f(j) \in g))$

3.1 Existence of a minimal bad sequence

When a qo is wf but not wqo a minimal bad sequence exists [4]. Let ch be the choice function on S_type: $ch \in \mathbb{P}\,1(S\_type) \to S\_type$ and $\forall s \cdot s \in \mathbb{P}\,1(S\_type) \Rightarrow ch(s) \in s$. Let $chmin$ be the function which give a minimum in a non empty set when the quasi-order is in wf we have:

$chmin = (\lambda A \cdot A \in \mathbb{P}\,1(S) | ch(\{m | m \in A \wedge (\forall z \cdot z \in A \wedge z \mapsto m \in g \Rightarrow m \mapsto z \in g)\})$

We instantiate g in $FrSB$ with

$\{n, k, b \cdot n \geq 0 \wedge k \in \mathbb{N} \nrightarrow S\_type \wedge 0..n-1 \subseteq \mathrm{dom}(k) \wedge$
$(\{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = k(i)) | f(n)\} \neq \varnothing$
$\Rightarrow b = chmin(\{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = k(i)) | f(n)\})) \wedge$
$(\{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = k(i)) | f(n)\} = \varnothing \Rightarrow b = ch(S\_type))$
$\mid n \mapsto k \mapsto b\}$.

let bs be the sequence $FrSB(\{i \mapsto j | i \geq 0 \wedge i < j\} \mapsto g)$ with our new g we got for free
$bs \in \mathbb{N} \to S\_type$ and $\forall n \cdot n \in \mathbb{N} \;\Rightarrow\; bs(n) = g(n \mapsto 0..n-1 \lhd bs)$ then we have
$\forall n \cdot n \in \mathbb{N} \;\wedge\; \{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = bs(i)) | f(n)\} \neq \varnothing$
$\Rightarrow\; bs(n) = chmin(\{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = bs(i)) | f(n)\})$ and
$\forall n \cdot n \in \mathbb{N} \;\wedge\; \{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = bs(i)) | f(n)\} = \varnothing$
$\Rightarrow\; bs(n) = ch(S\_type)$

Now we can prove by recurrence on n that $\{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = bs(i)) | f(n)\} \neq \varnothing$ and then we can prove that $\forall n \cdot n \in \mathbb{N} \Rightarrow bs(n) = chmin(\{f \cdot f \in BS \wedge (\forall i \cdot i \in 0..n-1 \Rightarrow f(i) = bs(i)) | f(n)\})$ and $\forall n \cdot n \in \mathbb{N} \;\Rightarrow\; bs(n) \in S$.

We can conclude that bs is a minimal bad sequence.

3.2 A well-quasi-order under value of a minimal bad sequence

When a qo is wf but not wqo and bs a bad sequence then $(g^{-1} \setminus g)[ran(bs)] \mapsto ((g^{-1} \setminus g)[ran(bs)] \lhd g \rhd (g^{-1} \setminus g)[ran(bs)] \in wqo$. To prove this theorem we have follow the proof of the lemma 22 in [5].

4 Conclusion

This two lemmas was not well defined but used in [3]. With both we have proved more easily but in the same way the Higman's lemma and the Kruskal's theorem.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010
2. D. Cansell, J.-R. Abrial: *Examples of using the Instantiation Plug-in"*, Rodin Workshop 2021
3. S. Demri, A. Finkel, J. Goubault-Larrecq, S. Schmitz and PH. Schnoebelen. *Well-Quasi-Orders For Algorithms MPRI Course 2.9.1 -2017/2018*. http://wikimpri.dptinfo.ens-cachan.fr/lib/exe/fetch.php?media=cours:upload:poly-2-9-1v02oct2017.pdf
4. C.S.J.A Nash-Williams. *On better-quasi-ordering transfinite sequences*. Proc. Camb. Phil. Soc., 64:273-290. 1968
5. L. Székely and É. Czabarka. *Well-Quasi-Ordering*. http://people.math.sc.edu/laszlo/WQO-Ramsey.pdf
6. G. Verdier and L. Voisin *Context instantiation plug-in: a new approach to genericity in Rodin.*, Rodin Workshop 2021
7. EBRP *Enhancing EventB and Rodin.* https://irit.fr/EBRP
8. *Rodin Platform.* http://www.event-b.org

Project Allocation with Event-B and ProB

Thai Son Hoang[0000−0003−4095−0732], Abdolbaghi Rezazadeh[0000−0002−0029−469X], and Michael Butler[0000−0003−4642−5373]

School of Electronics and Computer Science (ECS), University of Southampton, U.K.
{t.s.hoang,m.j.butler}@soton.ac.uk, ra3@ecs.soton.ac.uk

Abstract. This short paper presents a formal development in Event-B for allocating student projects. Using Event-B as the modelling language, we precisely specify the requirements of the allocation problem and develop an algorithm satisfying the requirements. We then use ProB to "execute" the Event-B specification to perform the allocation. The formal model and the ability of ProB to execute the formal model help us to ensure the fairness of the allocation process with the possibility of extending the algorithms to consider further requirements.

Keywords: Event-B · ProB · Project Allocation

1 Introduction

Annually, we must allocate our project students (i.e., Year 3 or MSc students) to potential supervisors. This is a challenging and cumbersome task for several reasons: (1) the continual growing of the number of students, (2) each staff member can supervise multiple students, (3) student belong to a specific programme must work on a project relevant to the programme, (4) the allocation has to consider staff's loading constraints, (5) the allocation will try to balance the staff's load as much as possible (e.g., it is undesirable to have a staff with 3 students while another staff has no students).

To facilitate the allocation process, a separate system (called "Choice") was developed for the students to input their project preferences. Each staff will add a list of sample project topics to the *Choice* system (together with the programmes that are relevant for those projects). The students can then select (up to 12) options from the systems as their preferences. These preferences are then used as the input for the allocation process, which is the discussion of this paper.

Previously, an existing software was used to automate this project allocation process. However, in recent years, the software performance does not match the expectation as the new programs are added and the cohort's size increases. This often results in some manual allocation process, which is (unsurprisingly) time-consuming and unreliable. As this software is no longer maintained, we develop a *formal specification that can be used for allocation directly, but also adaptable to future changes in the allocation process.*

The remainder of the paper is structured as follows. Section 2 discusses the background on matching algorithms and the Event-B formal method. Section 3 then presents an allocation algorithm and its formalisation. Section 4 gives some evaluation of the result of the allocation. Section 5 summarises and discusses future work.

2 Background

2.1 Matching Algorithms

Matching problems and algorithms are well-studied topics in computer science. One of the most well-known matching algorithms is the Gale–Shapley algorithm for the stable marriage problem [3]. A generalised version of the stable marriage problem is the college admissions problem [3]. Here, the matching is done between applicants and colleges, where each college can accept multiple applicants up to a certain limit. The algorithm involves several rounds, each round contains a *proposal* phase and an *acceptance* phase.

proposal : Each unallocated applicant applied to the most-preferred college to which they have not yet applied.

acceptance : Each college with quota q will (tentatively) accept the top-q applicants (or all applicants if the number is less than q), amongst the new applicants and the applicants that they have (tentatively) accepted, and reject the rest. Notice that potentially, some already accepted students might be rejected if the college prefers some new applicants.

The process is repeated until every student is on the acceptance list or has been rejected by all colleges to which they applied.

There are similarities and differences between the college admissions problem and our project allocation problem.

- In the college admissions problem, both applicants and colleges have a preference list of each other. In our project allocation problem, the students has the preference list of the staff (by ranking the topics proposed by the staff), but the staff do not rank the students.
- In the college admissions algorithm [3], there may be some applicants who will not be allocated due to their low ranking from the colleges. For our project allocation problem, the goal is to allocate all students to a supervisor.
- In the college admissions algorithm, applicant has to give their preferences. We have to handle a small number of students who have not entered their preferences.
- In both problems, there are limited capacities for colleges or staff. However, in the college admissions algorithm, there are no considerations for load balancing amongst the colleges. We have to balance the staff loading to ensure that the allocation of students is fair. We consider this an extended goal for future work.

2.2 Event-B Modelling Method and Tools

Event-B [1] is a state-based formal modelling method based on first-order logic and set theory. An Event-B model contains *contexts* and *machines*. Contexts specify the static part of the model, including carrier sets (types), constants, and axioms that constrain them. Machines specify the dynamic part of the model and include variables, invariants, and events. An Event-B machine corresponds to a discrete transition system, where the states and transitions are represented by variables and guarded events, accordingly.

Rodin [2] is an Eclipse-based tool that supports the Event-B modelling language. Verification of the consistency of the Event-B models can be done by proving the generated obligations using theorem provers or by model checking. We also utilise here an extension of Rodin called CamilleX [4] to support textual input for Rodin.

ProB [5] is a model checker for Rodin. ProB uses constraint solving to analyse the model. Furthermore, we can also validate the Event-B models by animating the models with ProB. This paper also utilises this feature of ProB for "executing" the project allocation algorithm.

3 Formal Development for Project Allocation

3.1 Requirements for Project Allocation

We assume that we have a set of programmes (ASM 1). Furthermore, each student is associated with exactly one programme that they are studying, however each staff can supervise projects in different programmes.

ASM 1 There is a finite set of programmes

ASM 2 There is a finite set of students

ASM 3 There is a finite set of staff

ASM 4 Each student is associated with a programme

ASM 5 Each staff is associated with a set of programmes

We assume that prior to our allocation, we have information about the student preferences of the staff (we omit the information about the topics here).

ASM 6 Students have a preference ranking (without duplication) of the supervisors

Each staff has a certain capacity for supervision, indicating the maximum number of students that they can supervise.

ASM 7 Each staff has a maximum number of students that they can supervise

Evidently, the project allocation cannot be guaranteed to be successful, e.g., when there are insufficient staff. However, we have the following requirements for allocations.

REQ 8 A successful allocation must ensure that every student is allocated to a supervisor.

REQ 9 A student's programme must match one of the supervisor's indicated programme

REQ 10 If a student has some preferences, then the allocated supervisor must be on their preference list

Notice that we omit the requirement about load-balancing here because it conflicts with the above requirements. Nevertheless, our algorithm will try to allocate as many students as possible.

3.2 An Algorithm for Project Allocation

As discussed in Section 2.1, we need to adapt the existing algorithm for our project allocation problem. The algorithm contains three stages.

Greedy Allocation Stage : In this stage, we use a modified version of the college admissions algorithm to allocate the students to their preferred supervisor. The process also contains two phases, i.e. proposal and acceptance, but there will be no rejection of already allocated students here.

proposal phase : Each unallocated student applied to the most-preferred supervisor to whom they have not yet applied.

acceptance phase : Each supervisor with capacity c will accept the $c - n$ new applicants, where n is the number of the current accepted students for that supervisor (or all new applicants if the number of them is less than $c - n$), and reject the rest. Notice that, once allocated, no students will be removed from their allocation in this stage (this is different from the college admissions algorithm).

The process repeats until either all students are allocated (successful allocation) or all the unallocated students' preferences have been taken into account. In the case of an unsuccessful allocation, we move to the Swapping Allocation Stage.

Swapping Allocation Stage : In this stage, we change the student allocation so that we can maximise the number of allocated students. This process involves an allocated student as and unallocated student us, satisfying:

- The allocated supervisor aS for as is on the preference list of us.
- as has another preferred supervisor pS that they have not yet applied to and pS still has a capacity for supervision.

In this case, as swaps aS for pS as the supervisor and aS will become the supervisor of us. We repeat this swapping until either all students are allocated (successful allocation) or we cannot find an allocated student to perform a swap for an unallocated student. Notice that at this point, any unallocated student with staff preferences will require manual allocation.

No-preference Allocation Stage : This stage tries to allocate the students without preferences to a supervisor. Here, the chosen staff will be a staff for the programme that the student studies and have the most capacity for supervision. The process finishes when all students are allocated (successful allocation) or if there are some unallocated students.

3.3 Formal Development

We give a brief overview of the formalisation of the problem and the algorithm using Event-B here.

The assumptions correspond to various sets and constants in different contexts, with appropriate axioms.

```
set PROGRAMME
axiom @axm1: finite(PROGRAMME) // ASM1
set STAFF
axiom @axm2: finite(STAFF) // ASM3
constant staff_programmes : STAFF ↔ PROGRAMME
axiom @axm3: dom(staff_programmes) = STAFF // ASM5
set STUDENT
axiom @axm5: finite(STUDENT) // ASM2
constant student_programme : STUDENT → PROGRAMME // ASM4
theorem @axm9: finite(student_programme)
constant student_preferences : STUDENT → (STAFF ⇸ ℕ) // ASM6
constant staff_limit : STAFF → ℕ // ASM7
```

Greedy Allocation Stage: This stage is modelled by events such as propose, accept, decline and phase/stage-transition events change_to_acceptance, change_to_propose, and move_to_swapping_stage. For example, event propose is specified as follows.

```
event propose
any student staff where
 @grd0: phase = Proposing
 @grd1: student ∉ dom(allocated) // student is unalocated
 @grd2: student ∉ dom(proposes) // student has not yet make a proposal
 // The staff is the top remaining choice for the student
 @grd3: staff ∈ dom(student_remained_choice(student))
 @grd4: ∀other_staff · other_staff ∈ dom(student_remained_choice(student))
    ⇒ student_remained_choice(student)(other_staff) ≤ student_remained_choice(student)(staff)
then
 @act2: proposes(student) := staff
end
```

Other events are omitted here due to space constraints.

Swapping Allocation Stage : This phase is modelled by a single event, namely swap corresponds to the algorithm in Section 3.2, with some stage-transition event move_to_no_preference_allocation.

```
event swap
any student staff allocated_student free_staff
where
 @grd0: process = SwappingAllocation
 @grd1: student ∉ dom(allocated)
 @grd2: staff ∈ dom(student_preferences(student)) ∧ staff ↦ student ∈ dom(staff_preferences)
 @grd3: allocated_student ∈ dom(allocated) ∧ allocated(allocated_student) = staff
 @grd4: free_staff ∈ dom(student_remained_choice(allocated_student))
 @grd5: staff_capacity(free_staff) ≠ 0
then
 @act1: allocated := allocated ⩤ {student ↦ staff, allocated_student ↦ free_staff}
 @act2: staff_capacity(free_staff) := staff_capacity(free_staff) − 1
end
```

No Preference Allocation Stage : This phase is modelled by a single event, namely no_preference_allocation corresponds to the algorithm in Section 3.2. We omit the details of the event here.

Finally, at any stage, if all students are allocated, then we consider the process finished successfully (REQ 8). Furthermore, the following invariants ensure the consistency of the immediate allocation, i.e., (REQ 9) and (REQ 10).

@inv1: $\forall$ student · student $\in$ dom(allocated) $\Rightarrow$ student_programme(student) $\in$ staff_programmes[{allocated(student)}]
@inv2: $\forall$ student · student $\in$ dom(allocated) $\wedge$ dom(student_preferences) $\neq \varnothing \Rightarrow$ allocated(student) $\in$ dom(student_preferences(student))

4 Evaluation

After developing the formal models specifying the algorithm, we utilise the capability of ProB to "run" the model on the actual data.

- We wrote a Python script to convert students' preferences and staff's programmes in CSV format to extended contexts to provide the actual values for sets and constants, e.g., STUDENT, STAFF, student_programme, etc. The output of the Python script is in CamilleX textual format for Event-B model.
- The algorithm will terminate (either successfully or unsuccessfully) when the formal model is deadlock. As a result, we use ProB to check for deadlock-freeness and the trace that ProB provided as a counter-example will essentially be the trace for the project allocation. In particular, we use ProB in depth-first search to avoid exploring the different traces of the model. This helps ProB to speed up considerably compared to the default search strategy.
- We then extract that last state of the ProB check to get the allocation data.

We used this approach for the allocation for 244 MSc students and 134 staff, each staff having a capacity of 3 students.

- After the Greedy Allocation Stage, 236 students were allocated.
- After the Swapping Allocation Stage, 5 more students were allocated
- After the No Preference Allocation Stage, 3 more students were allocated (all 3 students did not input their preferences).

5 Conclusion

This paper presents an approach for formally specifying an algorithm for project allocation and executing the formal model on actual data using ProB. The approach successfully allocates a large cohort of MSc students to one of their preferences. Compared to the previous year, when the allocation was done manually, we managed to release the project allocation two weeks earlier.

In the future, we want to use this approach as the core for the new project allocation software and extend the feature further (e.g., taking into account the topics). Taking into account the load-balancing constraint for the staff will require the relaxation of some constraints. At the moment, this process is done manually. Furthermore, we want to prove the properties of the algorithm, in particular, to discover the various invariants for proving the consistency of the formal model. Finally, on the tooling side, a more integrated approach with the Python pre/post-processing, Rodin static analyser, and ProB model checker is required.

References

1. Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.
3. David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, January 1962.
4. Thai Son Hoang, Colin F. Snook, Dana Dghaym, Asieh Salehi Fathabadi, and Michael J. Butler. Building an extensible textual framework for the rodin platform. In Paolo Masci, Cinzia Bernardeschi, Pierluigi Graziani, Mario Koddenbrock, and Maurizio Palmieri, editors, *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops - AI4EA, F-IDE, CoSim-CPS, CIFMA, Berlin, Germany, September 26-30, 2022, Revised Selected Papers*, volume 13765 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2022.
5. Michael Leuschel and Michael Butler. Prob: an automated analysis toolset for the b method. *International Journal on Software Tools for Technology Transfer*, 10(2):185–203, 2008.

EB[ASTD]: Meta-modelling framework for ASTD

Christophe Chen[1,2], Peter Rivière[1,3], Neeraj Kumar Singh[1],
Guillaume Dupont[1], Yamine Ait Ameur[1], Marc Frappier[2]

[1] *INPT-ENSEEIHT/IRIT, University of Toulouse, France*
[2] *Université de Sherbrooke, Sherbrooke, QC, Canada*
[3] *JAIST - Japan Advanced Institute of Science and Technology, Ishikawa, Japan*
{christophe.chen, peter.riviere, nsingh, guillaume.dupont,
yamine}@enseeiht.fr
marc.frappier@usherbrooke.ca

Algebraic State Transition Diagram (ASTD) [8] is a formal, graphical, state-based modeling language designed for the development of complex critical systems [10,1]. It provides a set of process algebra operators to compose hierarchical state machines, streamlining modularity in system design. Furthermore, its operational semantics defines transition rules for each ASTD operator.

Despite advances in incorporating features such as local state variables [13] and real-time [2], ASTD tool support (cASTD [13,14], pASTD [4], ASTD2EB [7,9]) is based on *ad hoc* model transformation that do not preserve the original structure of ASTDs. Each technology offers custom tools for ASTD, which can introduce new errors and complicate the integration of multiple tools on the same model. Moreover, it is essential to identify conditions characterizing well-defined ASTD, and to establish new properties (e.g., liveness) taking into account their operational semantics. In fact, our main objective is to seek a generic approach for *formally* **reasoning** on ASTD models.

The meta-model. In this context, we introduce EB[ASTD], an algebraic meta-model of ASTD formalizing its operational semantics [6]. This ground model, inspired from the EB4EB methodology [11,12], relies on a deep modelling strategy that integrates the ASTD syntax and semantics as Event-B algebraic theories, as described below. The whole model can be found at `https://www.irit.fr/EBRP/software/`

a) *Syntax and static semantics.* Listing 1 presents the `ASTDStruct` Event-B theory, which describes ASTD in a denotational style. Each compositional operator of ASTD is included in the datatype as a constructor, while component access is handled through destructors. Several predicate operators define the static semantics for ASTD to verify well-instantiation. These operators can generate proof obligations (POs) automatically by setting them as theorem in the Event-B context.

```
THEORY ASTDStruct
TYPE PARAMETERS St, Ev, Var
DATA TYPES
ASTD(St, Ev, Var)
 constructors
   Elementary(...)
   Automaton(...)
   Sequence(...)
   Closure(...)
   Guard(...)
OPERATORS
Invariant_WellCons predicate
Scope_WellCons predicate
 ...
ASTD_WellCons predicate
(a : ASTD(St, Ev, Var), accVar : ℙ(Var))
```

Listing 1. ASTD syntax

b) *Operational semantics.* Listing 2 defines the operational semantics of ASTD.

Several transitions rules are defined for each ASTD operator, and these rules are incorporated in the operator *NextState*. Given an ASTD *astd*, its current state *curr* and a triggered event σ, a set of all possible next states is returned according to the operational semantics. For example, in the case of Automaton, three enumerated set aut_i defined to represent the transition rules for aut_i.

```
NextState expression (
    astd : ASTD(St, Ev, Var) ,σ : Ev ,
    curr : ASTDState(St, Var))
well−definedness condition ...
recursive definition
case astd :
    Elementary(inv) => ...
    Automaton(i, f, ..., inv, mapping) =>
        aut1 ∪ aut2 ∪ aut3
    Sequence(fst, snd, attr, initAttr, inv)
        => ....
    Closure(...) => ...
    Guard(...) => ...
```

Listing 2. ASTD Transitions rules

c) *Two instantiation mechanisms.* We use both the deep and shallow instantiation mechanisms. The deep approach models ASTD as an instance of the meta-model, encoding it as a first-class object. In contrast, the shallow approach leverages the operational semantics of ASTD to generate the initial state and manage state changes. This enables using the ProB model checker and the visual animator VisB for validating ASTDs.

Proof-based reasoning. The framework enables the definition of proof obligations, checking their soundness and generating them. We illustrate this approach using POs for state invariants defined in pASTD [5]. Their POs lacks a formal justification, i.e. that they adequately represent their associated property. To achieve this, we encode the specification of POs in the form of properties on traces, allowing us to demonstrate that $ASTD \vdash POs \implies Spec[PO]\_On\_Traces$. For instance, invariant preservation corresponds to $ASTD \vdash PO_{pASTD} \implies \forall tr \in Traces(ASTD), INV(tr(i))$, i.e., that every execution of the ASTD satisfies the invariant when PO_{pASTD} hold. Three theories are defined for this.

a) *Proof obligations.* The definition and the generation process is straightforward. It requires the direct definition of the PO within the $ASTDPO$ theory operator. Then the operator is used as theorem in Event-B context, entailing PO generation. b) *Trace-based semantics.* The definition of traces exploits the operational semantics of ASTD in the theory $ASTDTraces$.

$$
\begin{array}{l}
thm\_of\_PO\_Correctness: \\
\forall astd, tr\cdot \\
astd \in ASTD(St, Ev, Var) \\
\land ASTD\_WellCons(astd, \emptyset) \\
\land tr \in \mathbb{N} \pfun ASTDState(St, Var) \\
\land IsATrace(astd, tr) \\
\land PO_{pASTD}(astd, ...) \\
\Rightarrow \\
(\forall i \cdot i \in dom(tr) \Rightarrow INV(astd, tr(i)))
\end{array}
$$

Listing 3. Proof of soundness

c) *Soundness.* The last theory $ASTDCorrectness$ leverages the specification of invariant satisfaction for a given state ($INV(tr(i))$). Finally, the soundness theorem is defined and proved in Listing 3. This proof highlighted some bugs in the manual specification [5,3]. Interested reader can consult [6] for more details.

Conclusion. This framework offers an explicit manipulation of ASTD concepts as first-class citizens. It features a proof-based mechanism that enables reasoning on specific ASTDs defined as instances of this meta-model. The tool is built upon the Rodin platform, which provides automated proof obligation generation, automatic and interactive verification, graphical animation, and model checking of ASTDs. Overall, the EB[ASTD] framework provides a sound foundation for proving properties about ASTDs and operates effectively within the Rodin platform, which is specifically designed for managing ASTD models and proofs.

References

1. de Azevedo Oliveira, D., Frappier, M.: Modelling an automotive software system with TASTD. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 124–141. Springer (2023), https://doi.org/10.1007/978-3-031-33163-3_10
2. de Azevedo Oliveira, D., Frappier, M.: TASTD: A real-time extension for ASTD. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 142–159. Springer (2023), https://doi.org/10.1007/978-3-031-33163-3_11
3. de Azevedo Oliveira, D., Frappier, M.: TASTD: A real-time extension for ASTD. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 142–159. Springer (2023), https://doi.org/10.1007/978-3-031-33163-3_11
4. Cartellier, Q., Frappier, M., Mammar, A.: Proving local invariants in ASTDs. In: Li, Y., Tahar, S. (eds.) Formal Methods and Software Engineering - 24th International Conference on Formal Engineering Methods, ICFEM 2023, Brisbane, QLD, Australia, November 21-24, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14308, pp. 228–246. Springer (2023), https://doi.org/10.1007/978-981-99-7584-6_14
5. Cartellier, Q., Frappier, M., Mammar, A.: Proving local invariants in ASTDs. In: Li, Y., Tahar, S. (eds.) Formal Methods and Software Engineering - 24th International Conference on Formal Engineering Methods, ICFEM 2023, Brisbane, QLD, Australia, November 21-24, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14308, pp. 228–246. Springer (2023), https://doi.org/10.1007/978-981-99-7584-6_14
6. Chen, C., Rivière, P., Singh, N.K., Dupont, G., Ait Ameur, Y., Frappier, M.: A proof-based ground algebraic meta-model for reasoning on ASTD in Event-B. In: 13th International Conference on Formal Methods in Software Engineering (FormaliSE) (2025)
7. Fayolle, T., Frappier, M., Laleau, R., Gervais, F.: Formal refinement of extended state machines. In: Derrick, J., Boiten, E.A., Reeves, S. (eds.) Proceedings 17th International Workshop on Refinement, Refine@FM 2015, Oslo, Norway, 22nd June 2015. EPTCS, vol. 209, pp. 1–16 (2015), https://doi.org/10.4204/EPTCS.209.1
8. Frappier, M., Gervais, F., Laleau, R., Milhau, J.: Refinement patterns for ASTDs. Formal Aspects Comput. 26(5), 919–941 (2014), https://doi.org/10.1007/s00165-013-0286-3
9. Milhau, J., Frappier, M., Gervais, F., Laleau, R.: Systematic translation rules from ASTD to Event-B. In: Méry, D., Merz, S. (eds.) Integrated Formal Methods - 8th International Conference, IFM 2010, Nancy, France, October 11-14, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6396, pp. 245–259. Springer (2010), https://doi.org/10.1007/978-3-642-16265-7_18
10. Ndouna, A.R., Frappier, M.: Modelling a mechanical lung ventilation system using TASTD. In: Bonfanti, S., Gargantini, A., Leuschel, M., Riccobene, E., Scandurra, P. (eds.) Rigorous State-Based Methods - 10th International Conference, ABZ 2024, Bergamo, Italy, June 25-28, 2024, Proceedings. Lecture Notes in Computer Science, vol. 14759, pp. 324–340. Springer (2024), https://doi.org/10.1007/978-3-031-63790-2_26

11. Rivière, P., Singh, N.K., Aït-Ameur, Y.: Reflexive Event-B: Semantics and correctness the EB4EB framework. IEEE Trans. Reliab. 73(2), 835–850 (2024), https://doi.org/10.1109/TR.2022.3219649
12. Riviere, P., Singh, N.K., Aït-Ameur, Y., Dupont, G.: Formalising Liveness Properties in Event-B with the Reflexive EB4EB Framework (2023)
13. Tidjon, L.N., Frappier, M., Leuschel, M., Mammar, A.: Extended algebraic state-transition diagrams. In: 23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018. pp. 146–155. IEEE Computer Society (2018), https://doi.org/10.1109/ICECCS2018.2018.00023
14. Tidjon, L.N., Frappier, M., Mammar, A.: Intrusion detection using ASTDs. In: Barolli, L., Amato, F., Moscato, F., Enokido, T., Takizawa, M. (eds.) Advanced Information Networking and Applications - Proceedings of the 34th International Conference on Advanced Information Networking and Applications, AINA-2020, Caserta, Italy, 15-17 April. Advances in Intelligent Systems and Computing, vol. 1151, pp. 1397–1411. Springer (2020), https://doi.org/10.1007/978-3-030-44041-1_118

Extending EB4EB for Parameterised Events

Peter Rivière[1], Neeraj Kumar Singh[2],
Guillaume Dupont[2], Yamine Aït Ameur[2]

[1] *JAIST - Japan Advanced Institute of Science and Technology, Ishikawa, Japan*
[2] *INPT-ENSEEIHT/IRIT, University of Toulouse, France*
priviere@jaist.ac.jp
{nsingh, guillaume.dupont, yamine}@enseeiht.fr

EB4EB [5,6], standing for *Event-B for Event-B*, is a framework that supports the formalisation of Event-B [1] models using first-order logic (FOL) and set-theory. This framework can handle machine elements as formulas, thus the EB4EB framework enables the definition of new specific proof obligations and *analyses* [8,4,7]. In the earlier formalisation of the EB4EB framework only *states* and *events* were handled, limiting the expressive reasoning power of the framework. In this paper, we present an overview of an extension of the EB4EB framework to support parameterised events [9], an important feature of Event-B. This extension is not straightforward in EB4EB. Indeed, the typing system supported by Event-B theories [2,3] is not rich enough to describe such extension in a constructive manner as for the other Event-B features formalised in EB4EB. The proposed solution, described in this paper, consists in defining an axiomatic formalisation of event parameters definitions.

EB4EB Extension. Listing 1 presents the extended version of the EB4EB meta-theory. The `EvtBTheoPar` includes a new type parameter, `PARAM`, for abstracting the type of event parameters. The main difference between the former EB4EB meta-theory and this one is the definition of the *destructors*. The $Machine$ data-type still has the same signature as well as a single constructor ($Cons\_machine$); however, this constructor only has two arguments (and thus two destructors): $Event$ identifying the events of the machine and $State$ identifying its state.

The usual destructors of the machine data-type (Inv, $Progress$, etc...) are defined *as axiomatic operators*, so that they can be free from the limitation of data-types regarding type parameters.

The $PARAM$ is defined as a type bound in the operators but not in the $Machine$ data-type, serving as a means to universally quantifying it independently from the data-type definition. Specifically, when the same operator (e.g., BAP_par) is used in two different contexts within the same machine $m : Machine(STATE, EVENT)$, both instances must reference the same sets for $STATE$ and $EVENT$, as these are fixed by the machine's type. However, the sets for $PARAM$ do not need to be identical, since they are not constrained by the machine's type. This allows each occurrence of $Param$, BAP_par, and Grd_par to have different types, even when used within the same machine.

THEORY EvtBTheoPar
TYPE PARAMETERS $STATE, EVENT, PARAM$
DATATYPES

$Machine(STATE, EVENT)$
CONSTRUCTORS
$Cons\_machine($
$Event : \mathbb{P}(EVENT),$
$State : \mathbb{P}(STATE))$
AXIOMATIC DEFINITIONS
OPERATORS
Param <***expression***> $(m : Machine(STATE, EVENT), e : EVENT) : \mathbb{P}(PARAM)$
Grd_par <***expression***>
$(m : Machine(STATE, EVENT), e : EVENT) : \mathbb{P}(PARAM \times STATE)$
BAP_par <***expression***>
$(m : Machine(STATE, EVENT), e : EVENT) : \mathbb{P}((PARAM \times STATE) \times STATE)$
...
AXIOMS ...

Listing 1. Machine Data-type with parameter

Instantiation principle for parameters. The instantiation of the *EvtB-TheoPar* theory from Listing 1, which introduces parameters, requires the definition of a set of axioms that encode an Event-B machine. The approach involves specifying the different components of the machine through definition axioms– predicates of the form $Op(m, \ldots) = Expr$.

CONTEXT EvtInstantiationSchema
AXIOMS
...
AxmParEv.1 : $Param(m, ev1) = T^{ev1}_{Par}$ ...
AxmParEv.n: $Param(m, evn) = T^{evn}_{Par}$...
AxmParGrd.1 : $Grd\_par(m, ev1) = \{par \mapsto s \mid par \in T^{ev1}_{Par} \land \ldots\}$ ...
AxmParGrd.n: $Grd\_par(m, evn) = \{par \mapsto s \mid par \in T^{evn}_{Par} \land \ldots\}$ ...
AxmParBAP.1 : $BAP\_par(m, ev1) = \{(par \mapsto s) \mapsto sp \mid par \in T^{ev1}_{Par} \land \ldots\}$ ...
AxmParBAP.n: $BAP\_par(m, evn) = \{(par \mapsto s) \mapsto sp \mid par \in T^{evn}_{Par} \land \ldots\}$ ...

Listing 2. Event instantiation schema

Machine POs. In addition to the introduction of event parameters and axiomatic definition, we must also update the defined PO operators. The proof obligations have been updated (defined axiomatically); the operators defining the invariant preservation PO are shown in Listing 3. The PO is divided into two parts: the base case and the induction case with the event. Note that the base and induction cases take into account type homogeneity in their axiomatic definitions and associated POs, respectively. The axiomatic definition of the data-type machine allows for including the parameter `PARAM` type in the proof obligation definition for specific events. Other POs operators are also updated in the same way.

OPERATORS
...
Mch_INV_One_Ev_Par_Def ***predicate*** $(m : Machine(STATE, EVENT),$
$e : EVENT)$
well−definedness $e \in Progress(m)$
direct definition
$BAP\_par(m, e)[(Param(m, e) \times Inv(m)) \cap Grd\_par(m, e)] \subseteq Inv(m)$
Mch_INV_Init ***predicate*** $(m : Machine(STATE, EVENT))$
direct definition
$AP(m) \subseteq Inv(m)$
Mch_INV ***predicate*** $(m : Machine(STATE, EVENT))$
direct definition
$Mch\_INV\_Init(m) \land (\forall e \cdot e \in Progress(m) \Rightarrow Mch\_INV\_One\_Ev\_Def(m, e))$
...

Listing 3. Well defined Data-type operators with parameter (behavioural semantics)

Conclusion. Unlike the original EB4EB framework, which employs constructive definitions for all types and operators within the associated Event-B theory, our approach utilises axiomatic definitions for event parameters. This allows for the instantiation of the theory to define various parameters with differing sets as their types, providing greater flexibility. Our approach has been applied to several examples to demonstrate the flexibility, reliability, and scalability of the extended EB4EB framework in terms of modelling, expressive power, and simplification of the proof process. More details can be found in [9].

References

1. Abrial, J.R.: Modeling in Event-B: System and software engineering. Cambridge University Press (2010)
2. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Tech. rep. (2009), `http://deploy-eprints.ecs.soton.ac.uk/216/`
3. Butler, M.J., Maamria, I.: Practical theory extension in Event-B. In: Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. pp. 67–81 (2013)
4. Mendil, I., Riviere, P., Aït Ameur, Y., Singh, N.K., Méry, D., Palanque, P.A.: Non-intrusive annotation-based domain-specific analysis to certify event-b models behaviours. In: 29th Asia-Pacific Software Engineering Conference, APSEC. pp. 129–138. IEEE (2022)
5. Riviere, P., Singh, N.K., Aït Ameur, Y.: EB4EB: A Framework for Reflexive Event-B. In: International Conference on Engineering of Complex Computer Systems, ICECCS 2022. pp. 71–80. IEEE (2022)
6. Riviere, P., Singh, N.K., Aït Ameur, Y.: Reflexive Event-B: Semantics and Correctness the EB4EB Framework. IEEE Transactions on Reliability pp. 1–16 (2022)
7. Riviere, P., Singh, N.K., Aït Ameur, Y., Dupont, G.: Formalising liveness properties in event-b with the reflexive EB4EB framework. In: NFM. Lecture Notes in Computer Science, vol. 13903, pp. 312–331. Springer (2023)
8. Riviere, P., Singh, N.K., Aït-Ameur, Y., Dupont, G.: Standalone Event-B models analysis relying on the EB4EB meta-theory. In: ABZ. Lecture Notes in Computer Science, vol. 14010, pp. 193–211. Springer (2023)
9. Rivière, P., Singh, N.K., Aït-Ameur, Y., Dupont, G.: Extending the EB4EB framework with parameterised events. Sci. Comput. Program. **243**, 103279 (2025). https://doi.org/10.1016/J.SCICO.2025.103279, `https://doi.org/10.1016/j.scico.2025.103279`