Teaching and Training in Formalisation with B

Thierry Lecomte[1]

CLEARSY, 320 avenue Archimède, Aix en Provence, France
thierry.lecomte@clearsy.com

Abstract. Despite significant advancements in the design of formal integrated development environments, applying formal methods in software industry is still perceived as a difficult task. To ease the task, providing tools that help during the development cycle is essential but proper education of computer scientists and software engineers is also an important challenge to take up. This paper summarises our experience of 20 years spent in the education of engineers, either colleagues or customers, and students, together with the parallel design and improvement of supporting modelling tools.

Keywords: B method, training, teaching

1 Introduction

Formal methods were developed to address software crisis by providing mathematically based techniques, including formal specification, refinement, proof, and verification. In theory, we now know how to use formal notations to write specifications and refine them gradually into a correct implementation, and use logic to prove programs correct. However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects. The problem is the complexity of implementing formal methods and the scarcity of skilled labour. However, this difficulty can be alleviated by providing more suitable teaching content[7] [10] [6] and tools that facilitate the implementation of formal methods.

This paper presents the experience collected during the last 20 years, training (future) engineers to use the B method, while developing the IDE supporting the B Method (Atelier B), using them (the tool and the method) for industry strength projects (development, Verification & validation), and boosting their dissemination in academia by providing specific hands-on teaching sessions.

The article is structured in 7 parts. Section 2 introduces the terminology. Section 3 briefly introduces the main principles of the B method. Section 4 presents how our training and teaching activities are structured. Section 5 describes our teaching material. Section 6 presents the return of experience that we have collected over the last 25 years, before concluding.

2 Terminology

This chapter clarifies a number of unusual terms and concepts used in this paper.

Atelier B is an industrial tool that allows for the operational use of the B Method to develop defect-free proven software[1].

B0 is a subset of the B language[2] that must be used at implementation level. It contains deterministic substitutions and concrete types. B0 definition depends on the target hardware associated to a code generator [4]. Most railways product lines use their own own specific code generator.

CSSP abbreviates CLEARSY Safety Platform. The CLEARSY Safety Platform is made up of a hardware execution platform, an IDE enabling the generation of diverse binaries from a single B model, and a certification kit describing its safety features as well as the safety constraints exported to the hosting system.

Safety refers to the control of recognized hazards in order to achieve an acceptable level of risk.

3 Introduction to the B Method

B[1] is a method for specifying, designing, and coding software systems. It covers central aspects of the software life cycle (Fig. 1): the writing of the technical specification, the design by successive refinement steps and model decomposition (layered architecture), and the source code generation.

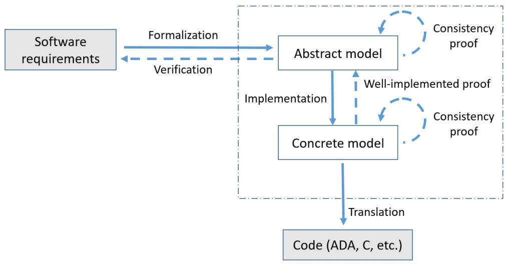


Fig. 1: A typical B development cycle, from requirements to code.

B is also a modelling language that is used for both specification, refinement (Fig. 2), and implementation (Fig. 3). It relies on substitution calculus, first order logic and set theory. All modelling activities are covered by mathematical proofs that finally ensure that the software system is correct.

B is structured with modules and refinements. A module is used to break down a large software into smaller parts. A module has a specification (called a machine) where a static and a dynamic description of the requirements are formalized. It defines a mathematical model of the subsystem concerned with:

- an abstract description of its state space and possible initial states,
- an abstract description of operations to query or modify the state.

[1] https://www.atelierb.eu/en/atelier-b-tools/

This model establishes the external interface for that module: every implementation will conform to this specification. Conformance is assured by proof during the formal development process. A module specification is refined. It is re-expressed with more information: adding some requirements, refining abstract notions with more concrete notions, getting to implementable code level. Data refinement consists in introducing new variables to represent the state variables for the refined component, with their linking invariant. Algorithmic refinement consists in transforming the operations for the refined component. A refinement

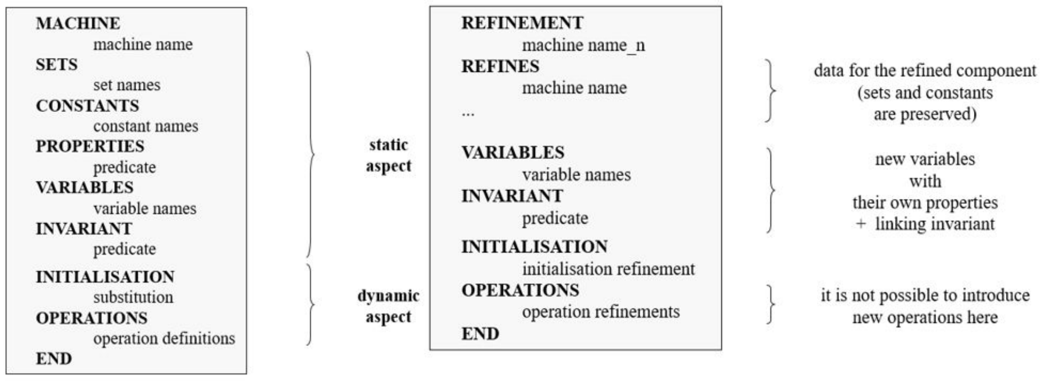


Fig. 2: Structure of MACHINE and REFINEMENT components.

may also be refined. The final refinement of a refinement column is called the implementation, it contains only B0-compliant models. In a component (machine,

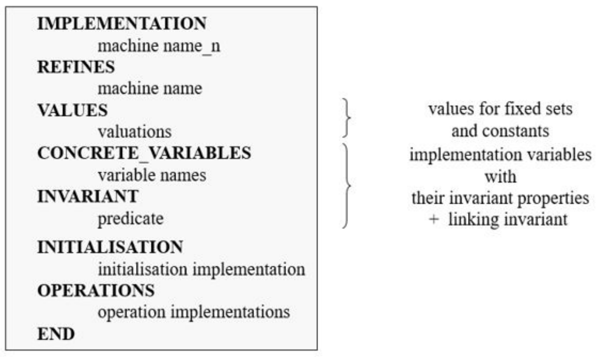

IMPLEMENTATION
machine name_n
REFINES
machine name
VALUES
valuations
CONCRETE_VARIABLES
variable names
INVARIANT
predicate
INITIALISATION
initialisation implementation
OPERATIONS
operation implementations
END
values for fixed sets and constants
implementation variables with their invariant properties + linking invariant

Fig. 3: Structure of IMPLEMENTATION component.

refinement, or implementation), sets, constants, and variables define the state space while the invariants define the static properties for its state variables. The initialisation phase (for the state variables) and the operations (for querying or modifying the state) define the way variables are modified. From these, proof obligations are generated such as: the static properties are consistent, they are established by the initialisation, and they are preserved by all the operations. Atelier B contains a model editor merging model and proof by displaying the number of proof obligations associated to any line of a B model, its current proof status (fully proved or not) and the body of the related proof obligations.

Finally a B project is a set of linked B modules. Each module is formed of components: an abstract machine (its specification), possibly some refinements and an implementation. The principal dependency links between modules are IMPORTS links (forming a modular decomposition tree) and SEES links (read only transversal visibility). Sub-projects may be grouped into libraries. A software developed in B may integrate or may be integrated with traditionally developed code.

4 Training vs Teaching

Training and teaching are both aimed at delivering some pedagogical content to an audience. However objectives and expectations may vary between these two activities. This chapter presents the structure of these activities.

4.1 Training

As the software company responsible for the development of Atelier B, professional training has always been a key activity, be it to train colleagues or engineers from other companies. The objectives of the participants vary:

- [**OBJ1**] it may be to understand and analyse an existing B model when accepting a deliverable provided by a third party. This is a strong regulatory requirement when the deliverable contributes to a critical system. The need is to be able to read the models, to determine which properties are expressed and how they are distributed within the model.
- [**OBJ2**] The need is then to adapt a model without damaging the architecture. It is necessary to be able to write the required specifications and implementations in a correct and efficient way without calling into question the existing technical modelling choices. It is also necessary to preserve as much as possible the mathematical proofs of the model.
- [**OBJ3**] It may be a case of building *ex nihilo* a complete model B which corresponds to a given technical problem and which interfaces with particular software components. It is then necessary to know how to specify efficiently, how to distribute the processing within components and how to optimise the proof work through levels of refinement.
- As the B-method is definitely proof oriented, it is obvious that a model has to be developed in order to facilitate its proof. A model can be expressed in

many ways and some of them are more easily proved by a theorem prover. [**OBJ4**] It is then necessary to have a deeper knowledge of automatic and interactive proof tools, of their capacity to prove such or such mathematical predicate.

Hence three training levels have been elaborated - "Understand B", "Practice B", and "Prove B" - to be practiced in this order and with some delay between each training to let the modeller assimilate the new concepts and get used to the technical environment.

"Understand B", directly aimed at [**OBJ1**], is designed to help understand the fundamental principles of the B Method and discover the B language. B is introduced as a method of formal specification and design with proof, which can go as far as the concrete level (with B0 language), and which offers formal specification and construction of a model by systematic description of its properties. Notions of modularity and hierarchy are presented, as a B model is built in a modular way, and its properties are introduced in a hierarchical way. Finally the proof is briefly presented as a mean to ensure the respect of invariant properties as it ensures in an exhaustive way that the code is in conformity with its specifications. To complete the picture, a description of the main uses of B in the industry is given. In a second part, predicates, mathematical expressions, and substitutions are all studied through their syntax and semantics, and implemented in short examples (often one-liners). The three types of B components (abstract machines, refinements and implementations) are presented. More than half of the training is hands-on session using Atelier B as a platform for experimenting the modelling and the automatic proof in B. The session is made of 4 consecutive full days, with a maximum number of 10 trainees for 2 trainers. Requirements for attending this training are a knowledge of the general principles of the development cycle of a system or software, a basic knowledge of computer science, and a mathematical knowledge at the level of a scientific baccalaureate.

"Practice B" covers both [**OBJ2**] and [**OBJ3**]. It is designed to help understand the principles of developing a B project, to practice building "good" B models, and to understand B language advanced concepts. Developing a B project requires to make clear the path leading from informal specifications to formal specifications. It also requires to know the modular construction of a B project and the various types of links between B modules, as well as the rules governing these links. A first methodological base is proposed on which to build a B project as an assembly of modules. The B model building practice is heavily based on exercises where the formal significance of "complying with specifications" is explained and linked to the proof obligations obtained. The participants are asked to create formal specifications on complete examples based on informal requirements. The principles for drafting models facilitating proof are studied. The session is similar to the "Understand B" one. At least, one month of intensive practice since "Understand B" training is required to let the participants increase their modelling skills.

"Prove B", directly aimed at [**OBJ4**], is designed to help verify models with proof, to understand how the automatic and interactive provers work. The verification activity relies on the use of an automatic prover to demonstrate most of the obligations of correct proof, the examination of remaining proof obligations to detect errors and the finalisation of the proof with the interactive prover. The automatic prover is described as a collection of collaborating proof strategies and mechanisms[2]. The main principles of the interactive prover are presented together with its interactive proof commands. Several methodological recommendations for a proper interactive demonstration allow to improve modelling efficiency. The session is similar to the ones above. At least, several months of intensive practice since "Practice B" training is required to let the participants increase their proof skills.

4.2 Teaching

Teaching at universities or engineering schools has a more pedagogical purpose than in a company. It is about educating students and complementing their ability to learn how to learn.

It uses a single resource base, which is made of:

- a presentation of the field of critical systems, which strongly recommends the use of formal methods for the highest criticality levels. The regulatory standards are introduced at this level.
- a presentation of the technical applications, the functions realised with the formal methods and the safety and security levels achieved.
- a modular presentation of the development cycle, the language and the associated tools.
- a corpus of simple examples for learning the language and more complicated (but simplified) examples from real systems.

The aim is to give students a formal touch, to teach them to model simple properties and to get to grips with the proof tool. In some cases, the code generation aspect is addressed. The teaching material is heavily based on the training resources. However the requirements are much lower than in a company and do not require a technical level to develop an industry-strength product.

5 Education material

Most of education has been completed with traditional means such as slides for the 3 training levels [3] and books (pdf format) [4]. Teaching slides are directly

[2] The collaboration is static and has been designed decades ago to optimize proof benchmarks.

[3] https://b-method.gitbook.io/training-resources-for-atelier-b/b-training-course/slides

[4] https://github.com/CLEARSY/CSSP-Programming-Handbook

offered to the students before the lectures, but are not released publicly. Exercises are completed on the blackboard or through computer manipulations.

Training "Understand B" comes with several exercises:

- **Specifying a resource management system** (model and proof obligations). 5 services have to be formally specified from a natural language description. For example, let the fourth service be named ReleaseResource. This operation takes a resource identifier rr as input, and may only be called when rr belongs to RESOURCES and indeed to the subset in_use as well. Its effect is such as available becomes available with element rr included, and in_use becomes in_use with element rr excluded. The resulting modelling is as follows:

```
ReleaseResource(rr) = PRE
    rr: RESOURCES &
    rr: in_use
  THEN
    available:=available\/{rr} ||
    in_use:=in_use-{rr}
  END
```

- **Simplified greatest common divisor**. The exercise makes use of the integer division to calculate the GCD of 2 positive integers that differ by no more than 2.
- **Batteries switch program**. 3 switches controlling 3 batteries powering a device have to be regularly controlled to avoid the same battery to discharge during a too long period. Properties are defined by learners (no short-circuit, power supply continuity) before their modelling.
- **Detection of the presence of two numbers in a list**. The exercise is aimed at using bool(P) expression.
- **Proving formal properties**: quantified predicates, function structure, simple induction. Several kinds of proof are introduced: contradiction, generalisation, and induction.
- **Block: Building a Complete Software B Project**. This software controls a railroad line, divided into fixed blocks. The purpose of the functionality is to establish safely, from the software point of view, which blocks are occupied by a train and which are free. Five different detectors are used but they are not accurate enough at the borders and they may be faulty. The project is made of 7 pre-existing components that need to be completed. For example, the operation set_block_occupancy should establish that a block having one of its border detector occupied or having its trackside detector occupied has to be occupied. In B, `<:` is the ASCII representation for set inclusion. `A <: B` means that the set A is included in the set B.

```
set_block_occupancy =
    BEGIN
        ob, tdl_alarm
```

```
            :(
            ob        <: t_block &
            tdl_alarm <: t_block &
            d_b2b[obd] \/ otd <: ob
        )
    END
```

The exercise covers formal modelling of non-trivial properties[5], specification and implementation of operations, and provides a first experience of a multi-component B project. Exercises are often selected as they address concrete, well-known devices with a short specification, vague enough to generate discussion and to obtain various models.

Training "Practice B" comes with several other exercises:

- **Refinement principles**. Refinement proof obligations are studied, in particular the ones related to a missing gluing invariant.
- **Traffic light control system**. This is the occasion to find properties for this well-known system, from different points of view (safety, traffic-flow, end-user, maintenance). Several subjects are treated: linkage with an external environment, modular decomposition and maintainability.
- **Implementation concerns**. A collection of small examples related to ensuring the absence of overflow, an explosion of proof obligations, the proof of correctness for a simple loop, the SEES clause and avoidance of aliasing. There is also an introduction to abstract iterators[6] for loop.

```
  cond, bl <-- iterate_t_block =
  PRE
    blocks_to_treat /= {}
  THEN
    ANY chosen_block WHERE
      chosen_block : t_block_i &
      chosen_block : blocks_to_treat
    THEN
      blocks_to_treat := blocks_to_treat - { chosen_block } ||
      treated_blocks := treated_blocks \/ { chosen_block } ||
      bl := chosen_block ||
      cond := bool(blocks_to_treat /= { chosen_block })
    END
  END
END
```

[5] Properties are not limited to typing. They require to use in combination diverse expressions and operators like composition, relational image, reverse, intersection, restriction in the domain, etc.

[6] With abstract iterators, the loop is prepared from the specification level by separating the iteration elements from the main substitution in a systematic way that could be efficiently implemented with automatic refinement. In the example below, t_block_i is the block super type.

– **Formal proof**. Several exercises to discover the proof activity: a proof of associativity (demonstration on paper then with the prover) and the language of proof-rules (introduction to the training "Prove B").
– **Modelling access to an island through a tunnel**. Introduction to the Event-B modelling.

Training "Prove B" comes with a large collection of exercises, too large to be listed individually:

– **Modifying the model**. Adding ASSERTIONS to a model to trigger simplifications or proof mechanisms.
– **Understanding proof commands**. Introduction to most common interactive commands including Proof by cases, Set Solver.
– **Adding user rules**. Extend the mathematical rules database with user rules (that need to be validated by the tool or manually).
– **Ambiguity**. Some operators like - or * have several meaning types (set, integer). This ambiguity may block some simplification mechanisms. Adding hypotheses could solve the problem (command ah - *Add Hypothesis*). In the model below, assertions have to be demonstrated with invariant and properties as hypotheses. Assertions are ordered: assertion in line 232 comes as an hypothesis to assertion in line 233; assertions in lines 232 and 233 come as hypotheses to assertion in line 234.

```
CONSTANTS
  ii,jj
PROPERTIES
  ii: NAT &
  jj: NAT
ASSERTIONS
  ii-ii = 0;     /* ah(ii-ii = -ii+ii) */
  ii+1-1 = ii;   /* ah(ii+1-1 = -1+1+ii) */
  ii*jj = jj*ii  /* ar(CommutativityXY) */
END
```

These resources help understand the behaviour of the proof tool. Often the tool leaves you in the middle of a proof tree and it is up to you to figure out what is missing to continue/complete the proof. Directions are given to browse/discover the rules database, to write mathematical rules and proof tactics.

To complement these online resources, new formats have been made available:

– **videos** [7] : several videos demonstrating how to use Atelier B and the CLEARSY Safety Platform (CSSP).
– **MOOC** [8]: 20 videos covering the basic aspects of B. The examples come from [9]. 5 videos are related to B project management.
– **self-training document** (for colleagues only): a compilation of "Understand B" and "Practice B" with a small number of exercises.

[7] https://www.youtube.com/@atelierbclearsy
[8] https://mooc.imd.ufrn.br/course/the-b-method

– **Collections of models** [9]: a large number of models which allow the study of different styles of modelling in B.

6 Return of Experience

This chapter summarises our activity and our findings accumulated since the beginning of the training and teaching activity.

6.1 Activity

Training has been ensured during more than 25 years, mainly in Europe, for an audience ranging from junior to senior engineers, project managers, safety and security evaluators. Target industries include railways, smart card, automotive, nuclear energy, and telecommunications. All objectives (from [**OBJ1**] to [**OBJ4**]) have been addressed. Some participants followed the whole course, most of them were involved only in the first two training levels. Some sessions were specifically tailored for a particular kind of model or on an existing (difficult to complete or to maintain) B project. Indeed, some models are part of a critical infrastructure and have a life span of several decades. It is therefore necessary to maintain a level of competence that allows the associated software to survive company turnover.

Teaching has been ensured at various occasions during the last 25 years, on most continents: lectures in a university course, contribution to a doctoral school, tutorial or dedicated workshop for scientific conference, invited presentation. The duration varies from a few hours to 3 or 4 days, spread over a month. The audience is quite often composed of students in the last year of their master. The profiles varied greatly: future general engineers receiving an introduction to formal methods, students with training in mathematics, computer science, embedded systems or mechatronics, researchers, and teachers. The teaching has happened either as a standalone lecture or to complement a (more theoretical) lecture by a professor from the university or engineering school. In the latter case, the course was often asked to emphasise the industrial use of formal methods, with the course acting as a justification for academic teaching.

6.2 Feedback

Trainees vs students There is sometimes a huge difference between trainees and students. In industry, training is either carried out to address technical difficulties anticipated for the successful completion of a project, or is seen as a reward for professional performance. In almost all cases, the trainee is attentive and diligent during the training. This is not always the case for the student, for whom participation in the course may be compulsory because it is linked to a given curriculum whose content cannot be adapted. It therefore happens that the

[9] https://github.com/hhu-stups/specifications/tree/master/prob-examples/B

behaviour of these two populations (engineers, students) diverges significantly and that the students do not see the point of the course, even if industrial use cases are used as course material.

Handling Abstraction Piaget[8] claimed that only one third of the population is able to handle abstraction. This proportion is somewhat reflected in our courses and training with:

- one or two people dealing with the questions faster than expected and getting ahead of the group in the practical work. These people, when recruited, make excellent practitioners;
- a first group understanding what is being done and why it is being done;
- a second group following the instructions given;
- a third group copying the results obtained by their peers or doing something else.

It should be noted that being a software developer does not imply a facility with formal modelling. Most developers do not have this ability, which is part of the reason why formal methods have difficulty being adopted in the industry when staff are selected solely for their availability and software skills. Our engineers are tested when they are recruited to see which group they belong to so that we don't make the mistake of assigning them to tasks that they will find very difficult to complete successfully.

Formal Models in Real Life Demonstrating the value of formal methods for software development is difficult. You need to be able to learn and use a mathematical language effectively. It requires a willingness to make life difficult by adding properties to the software before it is built. Agile methods and the prior development of software demonstrators undermine this approach. Often the examples presented in the courses are simple (or even simplistic) and do not necessarily allow to apprehend the added value of formal methods. The industrial examples are too large and confidential to be able to provide this insight and convince the learners definitively. With the introduction of the CLEARSY Safety Platform for education, it is possible to bring formal methods closer to the real world. This programmable board allows to specify, implement, prove and execute control logic expressed with B that will interact with the outside world through a simple interface (digital inputs and outputs). For students in formal methods, it shows the concrete applicability of formal techniques to the real world. For students in computer science and embedded systems, it allows them to verify without testing a software development in exchange for an intellectual effort.

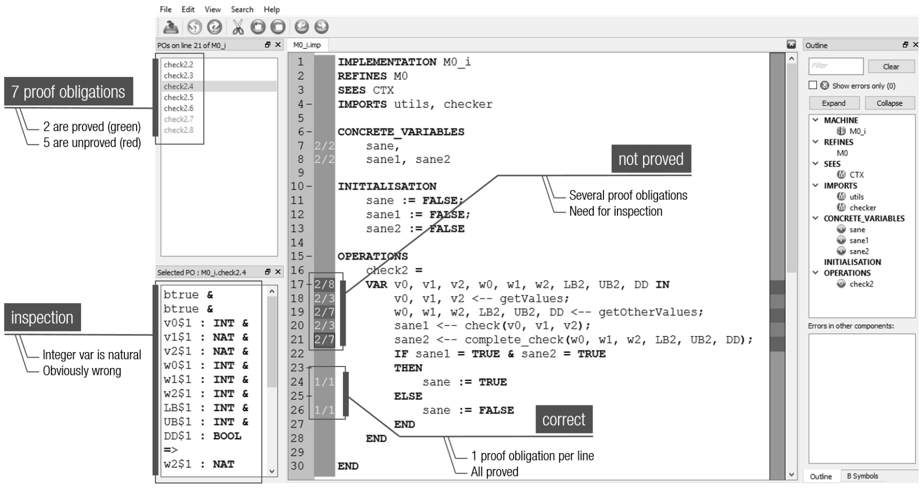
Teaching Proof is Difficult Proof has always been the stumbling block for teaching B. It is rather easy to explain how to model behaviour and properties. It is much more difficult to try to understand why certain points are not automatically proven[10]. This understanding has to be done through the prism of

[10] For cost reasons, the development of the core of the prover was frozen in 1998 to avoid that prover evolutions generate regressions of proof. Indeed, to interactively

multiple proof tools (theorem provers, solvers, model-checkers), which requires proven skills in mathematical proof (and an appetite for the subject). One must be able to determine whether the lack of automatic proof is due to:

– a limitation of the tool - then one must determine whether to modify the model to make it more provable by reformulating the properties and behaviour, or by using certain commands of the proof tool;
– a modelling error - then the model must be modified.

The proof activity is intimately linked to the modelling activity and to be effective must be carried out by the same person. The integration of the proof status in the model editor (Fig. 4) allows the modeller to be aware of the complexity of his modelling in terms of proof. This complexity could be quickly estimated based on the number of proof obligations, their localization and their automatic proof rate. The number of remaining proof obligations is a good measure of the complexity of a model, that has to be confirmed by visual inspection. The connection of Atelier B to external provers has and will improve the automatic proof rate. The difficulty of proof is thus reduced but not eliminated.



```
btrue &
btrue &
v0$1 : INT &
v1$1 : NAT &
v2$1 : NAT &
w0$1 : INT &
w1$1 : INT &
w2$1 : INT &
LB$1 : INT &
UB$1 : INT &
DD$1 : BOOL
=>
w2$1 : NAT
```

```
IMPLEMENTATION M0_i
REFINES M0
SEES CTX
IMPORTS utils, checker

CONCRETE_VARIABLES
    sane,
    sane1, sane2

INITIALISATION
    sane := FALSE;
    sane1 := FALSE;
    sane2 := FALSE

OPERATIONS
    check2 =
    VAR v0, v1, v2, w0, w1, w2, LB2, UB2, DD IN
        v0, v1, v2 <-- getValues;
        w0, w1, w2, LB2, UB2, DD <-- getOtherValues;
        sane1 <-- check(v0, v1, v2);
        sane2 <-- complete_check(w0, w1, w2, LB2, UB2, DD);
        IF sane1 = TRUE & sane2 = TRUE
        THEN
            sane := TRUE
        ELSE
            sane := FALSE
        END
    END

END
```

Fig. 4: Atelier B model editor showing proof status.

Automating Refinement Refinement is at the heart of B. It is a hard point when it comes to transforming non-trivial abstract structures and substitutions into their implementation in one or more steps. The refinement techniques depend on

prove a proof obligation costs about 35 euros. The modification of the proof status of an industrial project following the evolution of the prover could impact tens of thousands of proof obligations and would not be acceptable to the industrialist.

the human modellers and their experience. For the development of the Meteor metro safety automation, MATRA Transport[5] has developed and documented refinement techniques to systematise their use. This resulted in an automatic refinement tool that was later redeveloped for Atelier B. This tool (BART) automates the refinement of a B machine, using an extensible base of refinement rules and an inference engine to apply these rules to an abstract B model. The tool refines the data and then the operations through a process:

- automatic: the engine applies its refinement rule base to the abstract model. It stops when no rule can be applied to the structure/substitution being refined.
- interactive: the modeller must therefore complete the refinement rule base and then restart the automatic process. Refinement is complete when the machine has been successfully transformed.

Neither the tool nor the rules are proven: if the refinement produced is incorrect, it will not be provable. This tool was introduced in the mid-2000s in the hope of making refinement more easily accessible. The expected effect has not been achieved because in fact this tool, which allows the automation of a refinement process, requires a great deal of expertise on the part of the operator, who must know how to refine and model his knowledge in the form of rules.

7 Conclusion and Perspectives

Either training or teaching B are activities difficult to complete satisfactorily. The subject (set theory, first order logic, refinement, proof, etc.) is difficult and can only really concern a part of the audience. Our teaching resources have been enhanced over 25 years to address primarily a professional audience and therefore focus on the modelling of concrete problems/systems. In the meantime, the supporting tools have been improved: the editor integrates proof information while third party provers extend the proof system. New video-based resources have been available while the animation (graphic or not) of models was promoted [3]. Several on-going research projects are aimed at easing the proof process, to make the B modelling more appealing:

- With the project AIDOART [11], Artificial Intelligence could ease the proof process by suggesting proof tactics.
- With the projects BLASST [12] and ICPSA [13], third party provers / solvers could improve proof performances.

One could also imagine having a Big Data based tool offering modelling choices like copilot/Github using OpenAI. In fact, any technological innovation that would simplify the application of the B-method would be welcome to promote learning.

[11] https://www.aidoart.eu/
[12] Enhancing B Language Reasoners with SAT and SMT Techniques
[13] Interoperable and Confident Set-based Proof Assistants

Acknowledgements

The work and results described in this article were partly funded by ECSEL JU under the framework H2020. as part of the project AIDOaRt (AI-augmented automation supporting modelling, coding, testing, monitoring and continuous development in Cyber-Physical Systems).

References

1. Abrial, J.: The B-book - assigning programs to meanings. Cambridge University Press (2005)
2. Abrial, J.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
3. Bendisposto, J., Geleßus, D., Jansing, Y., Leuschel, M., Pütz, A., Vu, F., Werth, M.: ProB2-UI: A Java-Based User Interface for ProB, pp. 193–201 (08 2021)
4. Boulanger, J.: Formal Methods: Industrial Use from Model to the Code. Wiley (2013)
5. Burdy, L., Meynadier, J.M.: Automatic refinement. In: FM'99 workshop – Applying B in an industrial context : Tools, Lessons and Techniques, Toulouse, France, Proceedings (1999)
6. Cataño, N.: Teaching formal methods: Lessons learnt from using event-b. In: Dongol, B., Petre, L., Smith, G. (eds.) Formal Methods Teaching. pp. 212–227. Springer International Publishing, Cham (2019)
7. Istenes, Z.: Experiences of teaching formal methods in higher education. Formal Methods in Computer Science Education (FORMED 2008) (2008)
8. Kramer, J.: Is abstraction the key to computing? Commun. ACM 50, 36–42 (04 2007)
9. Schneider, S.: The B-method (Cornerstones of Computing). Macmillan Education UK (01 2001)
10. Zhumagambetov, R.: Teaching Formal Methods in Academia: A Systematic Literature Review, pp. 218–226 (03 2021)