Exploring the limitations of the Atelier B automated prover

Agata Borkowska, UID: 1690550, *MSc in Computer Science, University of Warwick*

***Abstract*—AtelierB is a tool for formal software development through refinement, using the B-method. It incorporates an automated prover, which has been recognized as the most thorough prover for B set theory, and has been used as a basis for many others. Nevertheless it has multiple shortcomings. Various approaches have been suggested and taken to improve its performance, including extensions to the proof rule base, plug-ins and third-party software. In this work we strive to establish the limitations of the prover without such additions, and discover at which point they become necessary. The secondary goal is to improve the robustness of the software without straying from pure B method, and taking into account the ease of use. As a metric of our success, we use the benchmarks proposed by Conchon and Iguernlala [1].**

***Index Terms*—B method, formal verification, abstract machine, proof**

I. Introduction

THE aim of formal specification and verification is to ensure the correctness of software. While overall less popular than quality assurance through testing, it is used in

II. Literature review

III. Project overview

A. The aims of the project

The key aim of this project is to discover the limitations of the Atelier B software. In the previous section, we have mentioned multiple plugins and extensions to the Atelier B software, which were created in order to improve the functionality. They highlight what other users have found lacking in Atelier B and what they have considered in need of improvement. However none of them demonstrate what can be achieved with the original software alone. Hence we strive to assess at what point Atelier B alone becomes insufficient, and the extensions are necessary to successfully verify a project.

An extension to this aim is answering the question: at what point is it necessary to add user-created rules in order to facilitate the proofs in the automated prover? It needs to be stressed that adding user-created rules is not advised unless it is absolutely necessary. Such rules then have to be thoroughly verified by means other than Atelier B software, to ensure that they are in fact correct. An error in an added rule would invalidate the entire proof process. Hence we will strive to circumvent obstacles by all means other than adding proof rules, before resorting to it. We hope to gather such advice and guidelines and make it accessible to others, so that they may be dissuaded from unnecessarily adding proof rules in their work. It is possible that adding proof rules to the Atelier B's rule base may be avoided entirely in the scenarios we have chosen, as they are by no means exhaustive; otherwise we will present the rules and their proofs for the use of others who wish to study the B-method.

We intend to explore various ways of expressing a specification in the B language, paying attention to how seemingly equivalent expression result in different proof obligations being generated by the automated prover, and follow it up by explaining the differences with the information contained in the source texts. This approach is often taken by students and academics beginning their work with the B-method and the Atelier B software. They are unlikely to have a thorough knowledge of the intricacies of the B-method as implemented in the Atelier B software, and would attempt to write a machine first, then search the manuals to explain

We hope to provide them with an explanation for the most commonly encountered proof obligations which are not automatically discharged, and persistent patterns among them. Both understanding the information given by the automated prover and avoiding undischarged proof obligations in the first place is of interest.

The observations arising from our work can be separated into two groups. Firstly there will be points highlighting the intricacies of the B method, which can be reasoned about and explained easily using source texts, primarily Abrial's *The B-book* [5] and Schneider's *The B-Method* [4].

Secondly, there will be disparities between the B method and its implementation in the Atelier B software. Not all of them can be classified as bugs, and some are clearly conscious choices which diverge from the pure theory of the B method. The manuals provided with the software will aid us with comparing the implementation to the theory.

B. The scope of the project

This work focused on abstract machines - they are the first step towards a formally verified implementation of a specification, thus making them the foundation of any project developed using the B method. An implication of this choice is focusing on set-related structures, including relations which are understood as sets of maplets, and operations, such as set comprehension, union, and intersection. These abstract constructs are not allowed in the later stages, where all data structures have to be concrete, and operations deterministic. We will also not discuss proof obligations related to loops, since those arise in the implementation stage.

Another reason for this choice is the distinct lack of literature focusing on this stage of development, in comparison to the refinement and implementation stages.[CITE] The latter stages undeniably generate more proof obligations, since on top of the proof obligations which can appear in the abstract machines, we need to consider those related to relation between a refinement machine and the one being refined.

We shall approach it from an academic rather than industrial point of view, focusing on smaller yet more illustrative examples. The main reason for it is to limit the number of factors affecting the number of generated proof obligations and be able to control them more precisely. An industrial-scale project with hundreds of proof obligations would be rather unwieldy for our purposes. Secondly a person new to formal development and the B-method is more likely to be able to follow clear, exemplar scenarios.

Nevertheless we hope that not only students and researchers will find our work helpful. Being able to minimise the number of proof obligations to be manually discharged has the potential to reduce the time required to complete any project. The constructs and expressions we discuss are the same ones as those used in industry. In fact, we found that the more complex structures in the B language, such as sequences, are rarely used in large-scale projects, as exemplified by [CITE].

IV. Methodology

V. Project Management

A. Choice of scenarios

This work has focused on two generic scenarios and explored various ways of fulfilling a specification for each of them. The specifications were kept deliberately imprecise for two reasons. Firstly, it allowed for an in-depth exploration of the theme. More precise specification would narrow down the options significantly, limiting our findings. Secondly in an industry setting it is not unheard of to have specification documents which leave details up to interpretation or are open-ended.

After exhausting the ways each expressing the specification in the B-method, we created a few more machines for each scenario, which illustrate other variations on the theme, although they diverge further from the original intentions. They served to analyse constructs which are more particular or less suited to the chosen scenarios, but nevertheless not unheard of.

It is important to observe that in large-scale industry projects which apply the B method, the constructs are kept simple and straightforward to avoid obfuscation. Thus in the sections below, dedicated to the chosen scenarios, the examples created were ordered by complexity. The later ones, for example in the case of the Bag Machine those involving sequences or relations, were analysed to compare the number of proof obligations generated next to their simpler variants.

VI. The Bag Machine - various approaches to describing sets and collections

A. Specification

Given a set of items, we want to describe a bag containing a subset of them. Initially, the bag is empty. We can perform the following operations on the bag:

- add an item to the bag
- remove an item from the bag
- find out the number of items in the bag
- find out which items are in the bag
- query whether a given item is in the bag

B. Discussion of the specification

An image of a bag of items was chosen due to its simplicity, although we recognize that it is potentially an unhelpful deviation from the most general description of this scenario, which can be achieved solely in set-theoretical terminology. We argue that this scenario is applicable in many circumstances. For example, one may be asked to develop a system that controls the barriers to a private car park. Then the system would maintain a set of registration numbers of the vehicles permitted to park there, which is a subset of all possible registration numbers. Another example is a library system, where `ITEMS` is the set of books in the library, and `content` are the books a person has on loan. We may want to impose a limit on the total number of items in the subset - bound by the number of spaces in the car park or the maximum number of books permitted to have on loan at the same time. This variation, although not explicitly required by the specification, was analysed in depth among others listed below.

The aim of this scenario is to explore different ways of expressing sets and operations on them. The two sets we are working on are `ITEMS`, consisting of all possible items that can be put in the bag, and `content`, the items contained in the bag at a given point. There are various ways of describing each of these sets.

We take the relation between them firstly to be that of subset, namely `content` $\subseteq$ `ITEMS` or equivalently `content` $\in \mathbb{P}($`ITEMS`$)$. This already shows two different ways of expressing a simple relation like this. Furthermore we may want to impose the limitation that the content of the bag is a finite set, thus arriving at `content` $\in$ `FIN(ITEMS)` in B notation, where `FIN(ITEMS)` denotes all finite subsets of the set of `ITEMS`.

There are other ways of describing the relation between the `ITEMS` and the `content` of the bag. For example latter can be a mapping from a subset of `ITEMS` to the number of times a given item appears in the bag.

At the level of abstract machines, it is permitted to use set comprehension and other operations and properties of sets, according to the Zermelo-Fraenkel set theory.

C. Variants of the Bag Machine

There are various ways of expressing the requirements given in the specification as an abstract machine. The following files are contained in the *Test_scenarios* archive for reference of the reader. We begin by having `ITEMS` as a deferred set - a set which will be defined at some later point of the development process, and `content` as simply a subset of `ITEMS`, and focus on the relation between them. We then move onto different ways of including the set of `ITEMS` in our machine, for example as an enumerated set or as a machine parameter. We finally explore various ways of describing the content of the bag, such as using sequences or relations. The reason for this order of tasks is to begin with the most intuitive implementation of the specification, before discussing less obvious changes to it.

1) Bagmch: is the reference machine which we take to be the core of this scenario. It implements exactly the specification without imposing any non-required conditions, such as the limit on the number of items in the bag. At the same time it includes one condition not explicitly mentioned in the specification, namely:

```
INVARIANT
    content : FIN(ITEMS)
```

The specification requires only that `content` $\subseteq$ `ITEMS`, however in any implementation it is infeasible to have truly infinite sets, thus the software considers them to be erroneous. In the *B Language Reference Manual* [7] we find the following definition for the set of natural numbers: `NAT` $= 0..$`MAXINT`, where `MAXINT` can be set by the user for a given project, although it is usually understood to be $2^{31} - 1$. This definition is not supported by the *B-book*, thus demonstrating a small disparity between the theory of the B-method and its implementation in Atelier B. Nevertheless it is very reasonable for practical purposes.

This machine generates only four proof obligations and all of them are discharged automatically. The first three check that the `INVARIANT` is preserved in the initialisation and by the operations to add and remove items. The last one is as follows:

```
    "Well definedness"
=>
    content: FIN(content)
```

It is concerned with the well-defineness of the operation `howmany`, which returns the number of items in the bag. The well-defineness proof obligations arise when an expression is used which requires certain conditions to be met in order to be well-defined. In this case, `card(content)` is well-defined only if `content` is a finite set. The well-definess conditions for all such expressions can be found in *B Language Reference Manual*. The following machine illustrates the proof obligations generated when the well-definess conditions are not met.

2) Bagmch_unbounded: illustrates the necessity to impose finiteness on the set of items contained in the bag. The sole difference between this and the reference Bag Machine is the statement:

```
INVARIANT
    content <: ITEMS
```

This machine generates the same four proof obligations as the reference machine, however the last one remains undischarged by the automated prover and correctly points the user to a problem in the abstract machine.

It is worth noting that there are multiple ways to denote subsets and finite subsets in these two machines, namely for unbounded subsets `content` $\subseteq$ `ITEMS` is equivalent to `content` $\in$ `POW(ITEMS)`, and for finite sets `content` $\in$ `FIN(ITEMS)` is equivalent to: `content` $\subseteq$ `ITEMS` $\wedge$ `content` $\in$ `FIN(content)`. These variations do not lead to any noticeable differences and do not cause different proof obligations to be generated and discharged, hence we conclude them to be equivalent where noted above.

3) Bagmch_limited: imposes a limit on the number of items that can be included in the bag at once. We define this limit as `max_elem` in the `CONSTANTS` clause of the static part of the machine and give its value in the `PROPERTIES` clause. Other ways of defining this value are shown later.

This restriction impacts the `additem` operation, as we now need to check that adding an item to the bag will not exceed the limit. It can be done in the preconditions section of the operation, as is shown in the example included in the archive, or in an if-else statement. We found these ways to be equivalent and we have elected to stick to the former, since it is less restricting for potential refinement.

There are nine proof obligations generated now. The initialisation and each of the operations to add and remove items results in two: one being a type check, the other making sure that the cardinality of `content` does not exceed `max_elem`. The last three proof obligations are concerned with the well-definess of the expression `card(content)` in the invariant and the operation `sadditem` and `howmany`.

[] If the invariant does not limit the content to the finite subsets, we

VII. Results

In fact it was the initial goal of this project, to create a collection of proof rules which simplify the verification process in the automated prover. This collection was meant to include especially the rules that were found to be needed for the proofs of various scenarios. However as the work has progressed, we have found that it was not necessary to create additional rules, and instead any obstacle could be circumvented with a deeper understanding of the inner workings of the prover.

References

[1] S. Conchon and M. Iguernlala, ”Increasing Proofs Automation Rate of Atelier-B Thanks to Alt-Ergo” in *Proc. 1st Int. Conf. Reliability, Safety and Security of Railway Systems*. Paris, France, 2016, pp. 243-253

[2] *Railway applications - Communication, signalling and processing systems*, EN 50128, 2011

[3] *Atelier B version 4.2 release notes*, ClearSy Sys. Eng., Aix-en-Provence, France, 2014

[4] S. Schneider, *The B-Method: an Introduction*, Basinstoke, Palgrave, 2001

[5] J.-R. Abrial, *The B-book: assigning programs to meanings*, Cambridge, Cambridge Univ. Press, 1996

[6] *Atelier B User Manual*, v. 4.0, ClearSy Sys. Eng., Aix-en-Provence, France

[7] *B Language Reference Manual*, v. 1.8.7, ClearSy Sys. Eng., Aix-en-Provence, France

[8] *Proof Obligations Reference Manual*, v. 3.7, ClearSy Sys. Eng., Aix-en-Provence, France

[9] *Interactive Prover User Manual*, v. 4.0, ClearSy Sys. Eng., Aix-en-Provence, France

[10] *Redaction guide for mathematical rules* v. 1.1, ClearSy Sys.ng., Aix-en-Provence, France, accessible: http://www.atelierb.eu/wp-content/uploads/sites/3/manuels/guide-de-redaction-de-regles-mathematiques-fr.pdf
Translated by D. Déharbe, accessible: http://www.math.pku.edu.cn/teachers/qiuzy/fm_B/Atelier_B/Writing_mathematical_rules.pdf

[11] M. Jacquel et al., ”Verifying B Proof Rules Using Deep Embedding and Automated Theorem Proving” in *Proc. 9th Int. Conf. Software Eng. Formal Methods*, Montevideo, Urugway, Nov. 2011, pp. 253-268

[12] D. Déharbe, ”Integration of SMT-solvers in B and Event-B development environments” in *Sci. Comp. Prog.* vol. 78, Elsevier, March 2013, pp. 310-326

[13] M. Leuschel et al., ”Automated property verification for large scale B models with ProB” in *Proc. 2nd Int. Symp. Formal Methods*, Eindhoven, The Netherlands. 2009, pp. 708-723

[14] V. Medeiros Jr. and D. Déharbe, ”BEval: A Plug-in to Extend Atelier B with Current Verification Technologies” in *Proc. 1st Latin Amer. Workshop Formal Methods*, Buenos Aires, Argentina, 20014, pp. 53-58

Appendix A
The Bag machine