Increasing Proofs Automation Rate of **Atelier-B** Thanks to **Alt-Ergo**

Sylvain Conchon[1,2] and Mohamed Iguernlala[3,1]

[1] LRI, Université Paris-Sud, Orsay F-91405
[2] INRIA Saclay – Ile-de-France, Toccata, Orsay, F-91893
[3] OCamlPro SAS, Gif-sur-Yvette F-91190

Abstract. In this paper, we report on our recent improvements in the Alt-Ergo SMT solver to make it effective in discharging proof obligations (POs) translated from the Atelier-B framework. In particular, we made important modifications in its internal data structures to boost performances of its core decision procedures, we improved quantifiers instantiation heuristics, and enhanced the interaction between the SAT solver and the decision procedures. We also introduced a new plugin architecture to facilitate experiments with different SAT engines, and implemented a profiling plugin to track and identify "bottlenecks" when a formula requires a long time to be discharged, or makes the solver *timeout*. Experiments made with more than 10,000 POs generated from real industrial B projects show significant improvements compared to both previous versions of Alt-Ergo and Atelier-B's automatic main prover.

Keywords: SMT solvers, B Proof Obligations, B Method

1 Introduction

The use of formal techniques to assess that software components conform to given requirements is gaining an increasing interest in the industrial world during the last decades. Indeed, when software are deployed in critical safety domains such as aeronautics, medical fields, and transportation, a high level of confidence is required because a bug may cause very costly damage.

Among formal frameworks, Atelier-B [4] is an industrial software development tool that implements the B method, a method based on abstract machines and refinement techniques. Roughly speaking, a developer starts a B project by designing an initial and most abstract version of a program, called abstract machine, that includes the formal specifications of the design's goal. Then, at each refinement step, the machine is turned into a more concrete one by adding more details about data structures or algorithms. At the end, a C/C++ or Ada code is produced. Each refinement step generates a set of mathematical formulas, called proof obligations (POs), that includes all the properties of the abstract machine and of the refinement process. These POs have to be proven coherent.

An industrial B project usually generates thousands of POs. Each PO is itself a big first-order formula that requires complex reasoning, such as Set theory and arithmetic, to be proved. In practice, proving manually the POs is a long and boring exercise. Therefore, the success and financial profitability of a B project strongly rests on the ability of proving resulting POs *automatically*.

Since its earlier versions, Atelier-B integrates a home-made automatic prover, which is mainly dedicated to B's Set theory [1] with some limited support for linear arithmetic. When a PO is not proved automatically, the user can either try to guide the proof interactively, or add some proof rules to the context of the solver, and prove the soundness of these rules later on. Consequently, to reduce the price of B software, a solution would be to increase proofs automation rate and to lower the number (cost) of manual proofs.

In order to increase proof automation, several research projects have started the integration of SMT (Satisfiability Modulo Theories) solvers. SMT solvers are recent and efficient automatic theorem provers built on top of a satisfiability (SAT) solver and a combination of decision algorithms for first-order theories of interest such as the free theory of equality and linear arithmetic over integers and rationals. In some application domains, these tools are extended with instantiation techniques to handle universally quantified formulas.

In recent years, we worked on the improvement of our SMT solver, called Alt-Ergo [3], in the context of the BWare ANR project [10]. BWare aims, among other things, at integrating SMT solvers as back-ends of Atelier-B. This relies on the Why3 [8] platform. Its main idea consists in translating Atelier-B proof obligations into Why3's logical language, combining them with a model of the B Set theory (also written in Why3's logical language) and feeding the result of the combination into different external solvers, among of which Alt-Ergo.

This paper presents our recent developments in Alt-Ergo that significantly improved its effectiveness in discharging POs coming from Atelier-B. Our enhancements include (1) new efficient data structures that boost performances of Alt-Ergo's core, (2) better heuristics for instantiating polymorphic quantified formulas coming from B model, and (3) a better interaction between the SAT solver and the decision procedures components. We also introduced a new plugin architecture to facilitate experiments with different SAT engines and provided a new experimental CDCL-based SAT solver. In addition, to be able to track and identify "bottlenecks" in our prover, we implemented a profiling plugin that allows us to observe the behavior of internal components of Alt-Ergo when a formula requires a long time to be discharged, or makes the solver *timeouts*.

We evaluated our improvements on a benchmark of more than 10,000 POs generated from four industrial B projects. The results are very promising and show a significant progression of current versions of Alt-Ergo compared to both previous releases of our solver and to Atelier-B's automatic main prover.

In Section 2, we present Alt-Ergo and its applications. Section 3 explains how B POs are enriched with a Set theory model and translated into Alt-Ergo. Section 4 details our benchmarks' characteristics and compares them with our existing test-suite. Section 5 describes some of our developments to improve Alt-Ergo for B POs, and Section 6 presents an experimental evaluation of these improvements. In Section 7, we conclude and discuss related and future works.

2 The Alt-Ergo SMT Solver

Alt-Ergo is an automatic solver of mathematical formulas designed for program verification. It is based on Satisfiability Modulo Theories (SMT). Solvers of this family have made impressive advances and became very popular during the last decade. They are now used in various domains such as hardware design, software verification and formal testing.

Alt-Ergo is used as a back-end of different tools and in various settings, in particular via the Why3 platform. For instance, the Frama-C suite relies on it to prove POs generated from C code, and the SPARK toolset uses it to check POs produced from Ada programs. Alt-Ergo is also used to prove POs issued from cryptography protocols verification and from the Cubicle model-checker. Recently, we started to use it to discharge POs coming from Atelier-B.

The simplified architecture of Alt-Ergo is shown in Figure 2. The SAT solver interacts with the decision procedures to look for a model for the ground part of the input formula. If a fix-point is reached and unsatisfiability is not deduced, it asks the "Axioms Instances" part for some ground consequences of quantified formulas (axioms). Generated instances are added to the SAT's context and the interaction with the "Decision Procedures" part continues. The latter component provides a combination of decision algorithms for a collection of built-in theories including the free theory of equality with uninterpreted symbols, linear arithmetic over integers and rationals, fragments of non-linear arithmetic, and enumerated and records datatypes.

```
type 'a set
logic add: 'a , 'a set → 'a set
logic mem: 'a , 'a set → prop

axiom mem_add:
  ∀ x,y: 'a. ∀ s: 'a set.
    mem(x,add(y,s)) ↔
    (x = y or mem(x,s))

logic a, b: int
logic s: int set

goal g: a = b+1 → mem(a-1,add(b,s))
```

Fig. 1: An example in Alt-Ergo's syntax

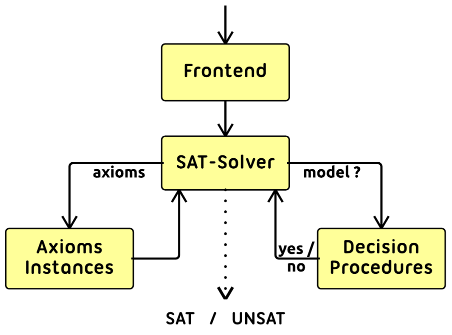

Fig. 2: Alt-Ergo's simplified architecture

Alt-Ergo's native input language is a polymorphic first-order logic *à la ML* modulo theories, a very suitable language for expressing formulas generated in the context of program verification. For instance, the toy example shown in Figure 1 declares an abstract polymorphic type `'a set`, some function and constant symbols (`add`, `mem`, `a`, `b` and `s`), one axiom (`mem_add`) that specifies the meaning of membership over `add`, and a formula to be discharged (a goal) that involves arithmetic and uninterpreted function symbols.

3 From B Proof Obligations to Alt-Ergo

One of the objectives of BWare is to connect additional automatic provers to Atelier-B to increase its proofs automation rate and to lower the cost of manual proofs. This goal is achieved via the translation scheme given in Figure 3:

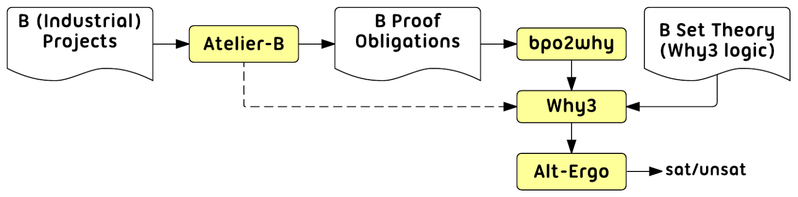

Fig. 3: Translating B proof obligations to Alt-Ergo's native input language

1. First, the POs produced by Atelier-B are translated into Why3's logic using bpo2why [9]. The latter tool has been extended during the project to cover a larger part of B constructs. A small PO and its corresponding Why3 translation are given in [9] (Fig. 2 and Fig. 4 respectively).
2. Datatypes and function symbols declarations, as well as the axioms of the B Set theory, do not appear in the original POs because they are built-in for Atelier-B's main prover. To make them explicit for Why3, a prelude that contains these information is written and appended to every translated PO. An overview of the content of this file is given in [9] (Fig. 7).
3. At this point, Why3 can be used to produce POs for a wide range of solvers in different formats (TPTP, SMT2, Alt-Ergo's native input language, ...).

Note that, a new proof obligations generator that is able to directly output POs in Why3's logic has been developed in Atelier-B during the project.

4 Benchmarks Characteristics

Quite at the beginning of the BWare project, we had a test-suite[1] of 12,831 POs obtained from four industrial B projects. The POs were previously discharged automatically or interactively in Atelier-B. They were translated to Why3 using bpo2why. Two benchmarks (called RCS3 and DAB) were provided by *Mitsubishi Electric R&D Centre Europe*. They are generated from B implementations of an automated teller machine and a software that controls a railway level crossing system, respectively. Two additional benchmarks (called p4 and p9) were provided by *ClearSy*, and were obfuscated.

Every PO is composed of three parts: the first one is a large set of declarations and axioms (universally quantified formulas). It results from the translation of the B Set theory prelude to Alt-Ergo's syntax. The second part is made of huge (in size) predicate definitions describing parts of the B state machines. It is part

[1] A first release is available here: `http://bware.lri.fr/index.php/Benchmarks`

of the original B formula. The last part is the “goal” we would like to prove. It is a ground formula involving the predicates of the second part. The concatenation of the two first parts will be called “the context” of the PO.

A quick inspection of the POs shows that they are made of equalities over uninterpreted function symbols and atoms involving enumerated data types. A small portion of atoms contains arithmetic and records. Compared to our older benchmarks, the average number of axioms, as well as the size of the POs are much larger in this new test-suite, as summarized below:

	VSTTE	Why3	Hi-Lite	RCS3	DAB	p4	p9
number of POs	125	4490	15993	2259	860	9342	371
avg. # of axioms	32	57	115	395	303	304	332
avg. size (KB)	8	12	36	907	252	258	420

At the beginning of the project, we ran state-of-the-art SMT solvers that can handle the POs of our test-suite. Those solvers have been running without any particular options or configurations. We used a 64-bit machine with a quad-core Intel Xeon processor at 3.2 GHz and 24 GB of memory. Time (*resp.* memory) limit was set to 60 seconds (*resp.* 2 GB) per PO. The results are shown in the table below, as well as the automation success rate of Atelier-B’s main prover (denoted B-PR) for these projects. Note that, for the sake of equity, B proof obligations were first split to obtain one goal per file, before they were given to main prover.

prover	version	RCS3	DAB	p4	p9
Alt-Ergo	0.95.2	2226 (98.7%)	822 (95.6%)	8402 (89.9%)	213 (57.4%)
Z3	4.3.1	2191 (97.1%)	716 (83.3%)	7974 (85.4%)	162 (43.7 %)
CVC3	2.4.1	2203 (97.6%)	684 (79.5%)	7981 (85.4%)	108 (29.1%)
B-PR	4.2	(90.1%)	(95.7%)	(83.0%)	(96.2%)

The evaluation shows that it is not immediate to obtain a substantial gain of performances by using SMT solvers to discharge B proof obligations. Without a specific tuning for B, SMT solvers compete equally with Atelier-B’s prover on the test-suite. We describe in the next section the main improvements we made in Alt-Ergo to increase its success rate on these benchmarks.

5 Tuning Alt-Ergo for B Proof Obligations

We now provide a non-exhaustive list of modifications we made in Alt-Ergo to augment its proofs success rate on BWare POs. But, we will start by describing our profiling plugin that allowed us to quickly localize sources of inefficiency.

5.1 Spying the solver

During our investigations to improve Alt-Ergo, we had to instrument several parts of its code to print some information and understand what is happening inside it. We ended by writing a profiling plugin that records relevant data and prints them in an appropriate way, with a negligible overhead when it is deactivated. Currently, information are printed in "text mode" and refreshed periodically.

When profiling, the user can switch between four views. The first view shows the progression of some global counters such as the current decision and instantiation levels, the total number of decisions and instantiations, the number of generated instances, the number of Boolean (*resp.* theories) simplifications and conflicts, the number of case-splits, etc.

The second one is a "matrix view" where the lines contain around twenty of the most used (time consuming) functions and the columns are labeled with the most important modules of Alt-Ergo. In every cell, the accumulated time spent in each function of every module is shown. This view allowed us to realize that, contrary to what we thought at the beginning of the project, arithmetic reasoning is very costly for some p4 proof obligations. In fact, arithmetic modules take more than 80% of the solver's time on these POs. An enhancement of corresponding algorithms increased both the success rate and the execution time of Alt-Ergo.

The third view prints the stack of currently activated modules and functions. To differentiate successive calls to the same function, we associate a fresh stamp to every new call. This allows us to detect when a function is slow or looping thanks to the repetition of the same stamp ID after two successive prints.

Finally, the fourth view is dedicated to axioms instantiation. For every axiom, this view shows: the number of generated, kept and ignored instances, the number of instances that participated in a conflict, and the number of "consumed" and "produced" ground terms by the instances. Actually, an axiom that produces a huge number instances or terms may have a bad (*i.e.* too permissive) trigger, so choosing another trigger or completely disabling the axiom may alleviate the solver's context and permit to do the proof.

5.2 Improving internal data-structures

During our first investigations to improve Alt-Ergo, we noticed that the representation of literals and formulas were not optimal, and that some normalizations were missing. This prevents the solver from making some straightforward simplifications, from getting the best from hash-consing techniques, and from doing some operations in constant time without allocating (*e.g.* computing the negation of a formula). To fix these issues, we reimplemented the internal data-structures for literals and formulas. In addition, we hash-consed internal data-structures of the decision procedures. This enabled the use of hash-consed based comparison to build sets and maps over these structures and induced an important speedup.

5.3 Improving the SAT solver

In general, SAT reasoning is cheaper than theories reasoning. So, to improve the interaction between the SAT and the theories, we made some modifications to delay calls to decision procedures as much as possible and to make all possible deductions at SAT level first. This is done when assuming unit facts, deciding a literal, or when performing Boolean constraints propagation modulo theories. A similar distinction is also made inside decision procedures: reasoning with equalities is, in general, much faster than processing inequalities.

In addition, we modified Alt-Ergo's architecture to enable the use of different SAT solvers provided as plugins, implemented a new CDCL-based solver, and enriched the default SAT solver with some modern decision heuristics.

5.4 Better axioms instantiation heuristics

During our investigations, we ran Alt-Ergo with profiling support on our POs and noticed a large number of axioms instantiation, a high activity of the decision procedures, and an important workload for the SAT engine. A further

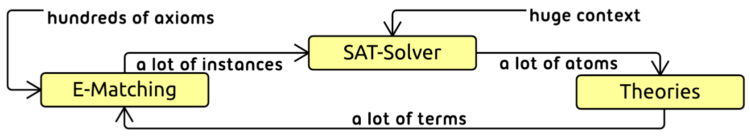

Fig. 4: Interaction of different components of Alt-Ergo on B proof obligations

investigation revealed that this is due to the hundreds of axioms and the huge context that implies thousands of generated instances, as shown in Figure 4.

In order to limit the number of generated instances at each matching round, we modified Alt-Ergo to only consider terms that appear in the current active branch (model) of the SAT engine when instantiating. We also added some normalizations to detect and eliminate redundant (equivalent) instances.

Another improvement is related to E-matching technique: in SMT solvers, matching process is performed modulo the set of known equalities. For instance, if x is a variable, $f(g(x)), f(g(a)), g(b)$ are terms, and $g(a) = g(b)$ is a known equality by the decision procedures, then E-matching a trigger [2] $f(g(x))$ against $f(g(a))$ will produce two solutions $\sigma_1 = \{x \mapsto a\}$ and $\sigma_2 = \{x \mapsto b\}$, while simple syntactic matching would only generate σ_1. Actually, the number of solutions for the matching problem is directly related to the number of generated instances. Consequently, we added the ability to disable the generation of new instances modulo known ground equalities via a new option, called `-no-Ematching`. This choice is justified by the fact that, while E-matching is not really mandatory to discharge more POs for BWare benchmarks, disabling this feature makes Alt-Ergo regress on our older benchmarks coming from Why3 and SPARK.

5.5 Save (replay with) the context used for a proof

Another important feature we added in Alt-Ergo is the ability to identify and save, for a discharged PO, a reasonably small over-approximation of the names of axioms that are useful to do the proof. The overhead due to the activation of this

[2] Triggers are terms with variables that prevent the instantiation of quantified formulas unless they "match" some ground terms present in the decision procedures.

feature (via option `-save-used-context`) is, in general, small compared to the benefits: saved information can be used to quickly replay the proofs (with Alt-Ergo via option `-replay-used-context`, or with another prover), as we demonstrate it in the next section.

6 Experimental Evaluation

In order to measure the impact of our improvements on BWare's test-suite, we considered two evaluation axes. For the first axis, we varied the time given to the solver for each PO: we used small timeouts that are adequate for an online integration (2 and 10 seconds per PO), and bigger timeouts, suitable for an offline integration (60 and 600 seconds per PO). For the second axis, we varied solver's options and resolution strategies:

1. the first strategy uses Alt-Ergo without particular options,
2. the second one uses the solver with the restricted options "`-no-Ematching -nb-triggers 1`": picking one trigger per axiom (default value is two) and disabling matching modulo equality will restrict the number of generated instances,
3. the third one is a portfolio approach that uses a dozen of configurations on a PO as long as it is not proved. This strategy is rather intended to be used offline, as timeout is set per configuration. Used configurations are listed in the figure below:

```
'-nb-triggers 1 -no-Ematching'
'-nb-triggers 1'
''
'-no-tcp'
'-no-theory'
'-nb-triggers 10'
'-greedy'
'-sat-plugin satML-plugin.cmxs -nb-triggers 1 -no-Ematching'
'-sat-plugin satML-plugin.cmxs -nb-triggers 1'
'-sat-plugin satML-plugin.cmxs'
```

   Basically, we restrict (*e.g.* `-nb-triggers 1`) or modify (*e.g.* `-no-Ematching`, `-nb-triggers 10`) some solver's capabilities, or use alternative implementations of some components (*e.g.* `-sat-plugin satML-plugin.cmxs`) to hopefully discharge a PO. Option `-greedy` enables the use of all the terms of the SAT solver when instantiating instead of those appearing in the current model only, option `-no-tcp` disables the simplification of disjunctions in the SAT modulo theories, while `-no-theory` completely disables theory reasoning.

For our experiments, we used the latest private release of Alt-Ergo (v. 1.10). The results are reported in the tables below (D = default strategy, R = restricted strategy, P = portfolio strategy, B = results of Atelier-B's main prover, and O = results of Alt-Ergo v. 0.95.2).

We can draw many conclusions from these results:

- even with a time limit of 2 seconds, the default strategy of Alt-Ergo 1.10 solves more POs than version 0.95.2 (which was ran in default mode with a time limit of 60 seconds),
- the restricted strategy is, in general, faster and solves more POs than the default one (except for RCS3, and for DAB with a time limit of 600 seconds),
- whatever the chosen timeout for Alt-Ergo, the portfolio strategy is always the fastest, and has the best resolution rate. This is as expected since the first and the second strategies are just particular configurations of the third one (in which timeout was set per configuration),
- more generally, we made substantial progress for both resolution time and the number of discharged POs compared to Alt-Ergo 0.95.2, in particular for projects p4 and p9,
- Atelier-B's main prover is still better on p9 even if we compare it to portfolio approach. The reason is that, contrary to p4 project that involves a lot of arithmetic reasoning, a substantial part of p9 POs necessitates lemmas superposition to be proven quickly. Unfortunately, E-matching is not suitable for that, and superposition calculus is currently lacking in Alt-Ergo.

	c.	RCS3	DAB	p4	p9
2	D	98.8%(2230/794s)	95.8%(824/ 138s)	94.7%(8849/ 1876s)	64.4%(239/ 126s)
	R	97.8%(2206/780s)	98.0%(843/ 135s)	98.8%(9230/ 1880s)	66.6%(247/ 124s)
10	D	99.0%(2233/811s)	95.8%(824/ 138s)	97.7%(9124/ 2940s)	67.4%(250/ 183s)
	R	97.8%(2207/783s)	98.0%(843/ 135s)	99.3%(9273/ 2126s)	73.6%(273/ 261s)
60	D	99.0%(2233/811s)	97.0%(834/ 518s)	98.3%(9179/ 4480s)	71.2%(264/ 691s)
	R	97.8%(2207/782s)	98.0%(843/ 135s)	99.3%(9274/ 2158s)	77.9%(289/ 547s)
600	D	99.0%(2233/811s)	99.0%(851/1789s)	98.8%(9231/13375s)	72.2%(268/1554s)
	R	97.8%(2207/782s)	98.0%(843/ 135s)	99.3%(9278/ 2542s)	82.8%(307/3064s)
60	B	90.1%	95.7%	83.0%	96.2%
	O	98.7%	95.6%	89.9%	57.4%

We also notice for DAB project that increasing timeout of the portfolio approach does not allow to discharge more POs. This may be due to two reasons: either the triggers computed for the remaining formulas are not suitable to do the proofs, or the proofs require superposition calculus.

	c.	RCS3	DAB	p4	p9
2	P	99.2%(2237/795s)	99.2%(853/ 138s)	99.4%(9289/ 1968s)	71.2%(264/ 142s)
10	P	99.2%(2237/795s)	99.2%(853/ 138s)	99.5%(9292/ 2204s)	81.7%(303/ 353s)
60	P	99.3%(2240/844s)	99.2%(853/ 138s)	99.5%(9295/ 2272s)	86.2%(320/1007s)
600	P	99.3%(2240/844s)	99.2%(853/ 138s)	99.6%(9303/ 4729s)	90.8%(337/3402s)

We made a second experiment to measure the impact of saving the "names of axioms" that have been used to discharge a PO, and of replaying the proof with the pruned context. For that, we used the default configuration of Alt-Ergo 1.10 and a time limit of 600 seconds. The results are reported in the table below. We notice that we have a small overhead when option `-save-used-context` is activated, compared to the results we got with the default strategy, and that around twenty POs are not proved anymore. However, thanks to the information saved when this option is activated, all proofs replay succeeded quite faster.

	RCS3	DAB	p4	p9
-save	99.0%(2233/821s)	99.0%(851/3015s)	98.7%(9216/15454s)	72.0%(267/1250s)
-replay	100% (776 s)	100% (124 s)	100% (1742 s)	100% (101 s)

7 Conclusion and Future Works

This paper describes our improvements in the Alt-Ergo SMT solver to increase its proofs success rate on formulas coming from the Atelier-B framework. Our experimental results show a substantial progression of Alt-Ergo 1.10 compared to older versions and to Atelier-B's main prover. It turns out that B proof obligations have some specificities that should be taken into account to obtain a good success rate. Note that, the integration of SMT solvers in the Rodin [11] platform to discharge proof obligations coming from Event-B [1] has already been investigated [7]. However, the translation scheme that has been employed is quite different from BWare's (static expansion of Set theory constructs before generating a PO for an SMT solver). It would be interesting to investigate the use of BWare's translation technique within Rodin. This could be achieved by adapting the investigations of [2] to use BWare axiomatization.

In the near future, we plan to investigate the integration of built-in support for a fragment of the B Set theory in Alt-Ergo via the extension of our rewriting-based frameworks AC(X) [5] and CC(X) [6]. This would improve resolution time and offer some nice completeness properties on this fragment. The extension of Alt-Ergo with superposition calculus would also increase proofs success rate. In addition, we identified some components of our solver that necessitate further improvements (*e.g.* triggers inference module). Other possible lines of work include the use of benchmarks coming directly from Atelier-B's new POs generator, and the extension of the B Set theory prelude[3]. Yet, a new project containing 60,000 POs has been recently translated to Why3's logic. It constitutes another interesting challenge for Alt-Ergo.

Last, but not least, a previous release of Alt-Ergo (version 0.94) has already been qualified for a usage in avionic area (DO-178B). It would be worth considering the ability to qualify a new version for a usage in the railway domain.

[3] The prelude is still under development, and some axioms may be missing to discharge a PO, or may be written in an "unsuitable" way for the solvers.

References

1. J.-R. Abrial. *The B-book - assigning programs to meanings.* Cambridge University Press, 2005.
2. D. Adjepon-Yamoah, A. Romanovsky, and A. Iliasov. A reactive architecture for cloud-based system engineering. In *Proceedings of the 2015 International Conference on Software and System Process*, ICSSP 2015, pages 77–81, New York, NY, USA, 2015. ACM.
3. F. Bobot, S. Conchon, E. Contejean, M. Iguernlala, S. Lescuyer, and A. Mebsout. *Alt-Ergo, version 0.99.1.* CNRS, Inria, Université Paris-Sud 11, and OCamlPro, Dec. 2014. `http://alt-ergo.lri.fr/`, `http://alt-ergo.ocamlpro.com/`.
4. ClearSy System Engineering. *Atelier B User Manual, version 4.0.* `http://tools.clearsy.com/wp-content/uploads/sites/8/resources/User_uk.pdf`.
5. S. Conchon, E. Contejean, and M. Iguernelala. Canonized Rewriting and Ground AC Completion Modulo Shostak Theories : Design and Implementation. *Logical Methods in Computer Science*, 8(3), 2012.
6. S. Conchon, E. Contejean, J. Kanig, and S. Lescuyer. CC(X): Semantic Combination of Congruence Closure with Solvable Theories. *Electronic Notes in Theoretical Computer Science*, 198(2):51–69, May 2008.
7. D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin. Integrating SMT solvers in rodin. *Sci. Comput. Program.*, 94:130–143, 2014.
8. J.-C. Filliâtre and A. Paskevich. Why3 - Where Programs Meet Provers. In *ESOP*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, 2013.
9. D. Mentré, C. Marché, J.-C. Filliâtre, and M. Asuka. Discharging Proof Obligations from Atelier B Using Multiple Automated Provers. In *ABZ*, volume 7316 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2012.
10. The BWare Project, 2012. `http://bware.lri.fr/`.
11. L. Voisin and J.-R. Abrial. The Rodin Platform Has Turned Ten. In Y. Ait Ameur and K.-D. Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, volume 8477 of *LNCS*, pages 1–8. Springer Berlin Heidelberg, 2014.