Teaching an Old Dog New Tricks

The Drudges of the Interactive Prover in Atelier B

Lilian Burdy and David Deharbe(✉)

ClearSy Systems Engineering, 320, av Archimède – Les Pléiades III,
13857 Aix-en-Provence CEDEX 3, France
{lilian.burdy,david.deharbe}@clearsy.com

Abstract. This paper presents an evolution of an industrial proof support framework that integrates state-of-the-art technologies without compromising the existing tool qualification status. Third-party provers produce proof rules that may be applied by the legacy system and verified using a certified approach. This approach has been implemented in Atelier B, a formal-methods based IDE for the development of software components and for the modeling of systems.

1 Introduction

The industrial applications of formal methods rely on the formal verification of conditions, e.g., invariant preservation. In case the specification logic is undecidable, human interaction is required to discharge proof obligations. This is the case of the B method and Event-B, two closely related formal methods, based on a first-order language with integer arithmetics and set theory. So even though automatic theorem provers are available, their application requires their users to interact with proof assistants. Atelier B, an integrated development environment for both the B method and Event-B, includes custom automatic and interactive provers. Increasing the success rate of automatic provers and decreasing the amount of user interaction are key to reducing the cost of formal methods and increasing their application in the industry. This paper addresses the former approach.

One main industrial application of formal methods in the industry is the development of safety-critical, software-based, systems. Mostly, formal methods are used when mandated or recommended by an industrial standard (e.g. EN50128 [3]). In that context, all the tool support must be qualified. Obtaining such a qualification has a significant cost and then usually only applies to a specific version of the tool. Atelier B has been qualified to formally develop software components by large industrial partners in the railway industry. This paper presents an extension to Atelier B that improves its support for interactive proof without compromising the certification obtained by the existing code base.

M. Butler et al. (Eds.): ABZ 2018, LNCS 10817, pp. 415–419, 2018.
https://doi.org/10.1007/978-3-319-91271-4_33

2 Technical Background

Atelier B and Proof. In Atelier B, proof obligations (POs) are produced automatically, two proof engines being available to check them. One solver, called *pr*, is a conditional term rewrite engine written in a Prolog-like notation, called the *theory language*, together with a default set of rules. Users may also write their own rules, which need also be verified eventually. The second solver, called *pp* for *predicate prover*, was developed to prove such rules and takes an (incomplete) tableaux-based approach. Both have been certified[1]. For rules that *pp* cannot handle, the certified verification procedure is to first prove the rule with a mathematical demonstration and then have a third-party independent expert validate this demonstration. Note that *pp* is also available to solve proof obligations.

In interactive mode, the user is presented with a PO composed of a goal, local hypotheses and global hypotheses. In the course of an interactive session, the goal and the hypotheses evolve, new POs are created and the current PO may be discharged. Examples of commands are: rewrite the goal using an equality from the hypotheses; call a built-in expression simplifier; apply either *pr* or *pp* on the current goal; instantiate a universally quantified hypothesis; apply a given rule (or set of rules); case split on a condition. The execution of some commands produce new PO, e.g., case splits and instantiations. A PO is discharged when a so called terminal rule is applied. An example of terminal rule is: `binhyp(a) => a or b`. Here `a` and `b` are so-called jokers and stand for terms, `=>` is a delimiter in the rule between the conditions (to the left) and the conclusion (to the right). Such a rule can be applied if the current PO goal matches `a or b`, i.e., it is a disjunction, and the term matching `a` is found in the hypotheses (that is the semantics of the `binhyp` operator).

Leveraging Automatic Provers in Interactive Proof Assistant. Since Atelier B has been originally developed, mechanical theorem proving has seen a lot of progress and powerful automatic provers are now available, such as SMT solvers [2,4]. The area of formal methods has also made efforts to use such tools to address its own verification challenges (see, e.g. [5,6]). Our goal has been to take advantage of some of these advances in Atelier B without compromising certification.

We follow the approach taken in the general purpose proof system Isabelle [7], extended with "sledgehammer" [8], a command to invoke external solvers on the current PO and, when successful, to reconstruct an Isabelle proof script from their output. Our take on applying this approach in the legacy proof system of Atelier B is an extension we named *the drudges of the interactive prover*.

3 The Drudges of the Interactive Prover: Principles

Even though the logic of POs in Atelier B is rich and undecidable, it is often the case that, during the course of an interactive proof, proving the current goal

[1] E.g., sanctioned for software development by RATP for line 14 (operated completely automatically).

only requires arguments that may be cast in a decidable logic, such as propositional logic, equality and integer arithmetics. For such goals, the layered solving architecture found in most SMT solvers is particularly fit. However this is not how Atelier B solvers function, and they sometimes fail to prove automatically POs that only require propositional reasoning or substitution of equals.

So our approach consists in applying SMT solvers to *an abstraction of the current goal*, i.e., a simplified version where the set operators are uninterpreted (i.e., their semantics is lost). Such simplified POs contain the declarations for the symbols from the original PO, then a series of assertions. There is one such assertion for each hypothesis (and so, all hypotheses are thus abstracted) and one assertion with the negation of the goal. All these assertions are labeled. If an SMT solver finds the simplified PO to be unsatisfiable, the original PO is valid. Then, all we have to do is build a rule that can be applied by the solver of Atelier B. This rule needs to be logically sound, applicable to the current PO, and as general as possible.

To this end, the SMT solver is then queried to obtain an unsatisfiable subset of the assertions, using the `get-unsat-core` functionality [2]. The solver then returns the set of the labels of the assertions it used to conclude unsatisfiability. Now all needs to be done is to build a rule from the logical formulas associated with these labels. Assuming the solver is sound, such a rule is valid. Also, by construction, it is applicable to the current PO. To make it more general, terms are replaced with jokers. The issue here is where jokers are introduced. To illustrate this point, consider the following PO:

$$\{\cdots; 0 \leq fn(s_3); s_1 = s_2 \vee s_1 = s_3; 0 \leq fn(s_2); \cdots \vdash fn(s_1) \leq fn(s_2) + fn(s_3)\},$$

where the hypotheses are to the left of $\vdash$ and the goal is to the right. Only the hypotheses returned by `get-unsat-core` are shown. Consider the proof rule, built with these formulas:

```
   binhyp(0<=fn(s3)) &
   binhyp(s1 = s2 or s1 = s3) &
   binhyp(0<=fn(s2)
=> fn(s1) <= fn(s2)+fn(s3)
```

This rule is sound but it only applies to POs where the goal and some hypotheses are identical to those in the rule.

To gain generality, we can replace (sub)-terms with jokers, by recursing over the structure of the given formulas, keeping the operators known to the SMT solver, and by introducing a joker for each different sub-term. The result is the following rule:

```
   binhyp(0<=a) &
   binhyp(b=c or b=d) &
   binhyp(0<=e)
=> f <= e+a
```

However, the abstraction here is too coarse and the resulting rule is no longer sound.

However, when jokers are introduced one level deeper in the syntactic structure, the rule obtained is sound, applicable to the original PO, and as general as possible given the initial problem.

```
   binhyp(0 <= a(b)) &
   binhyp(c=d or c=b) &
   binhyp(0 <= a(d))
=> a(c) <= a(d)+a(b)
```

This discussion illustrates the main principles in the generation of proof rules. Jokers are introduced at a given level, and if the rule is sound, the process stops, otherwise it is repeated one level deeper. This process is guaranteed to end at most when the level is the height of the original PO. The verification of the soundness is carried out by the same SMT solver that was applied to the original proof obligation.

4 The Drudges of the Interactive Prover: Application

The functionality is in release 4.5 of Atelier B. The user sets the preferences of the interactive prover to create drudges. A *drudge* is an SMT solver with settings to enable quantifiers and arithmetics reasoning. These settings guide the procedure that simplifies the POs for the SMT solver; e.g., disabling quantifiers forces all quantified sub-formulas to be abstracted.

The new functionality can be run in the interactive prover either with a command `smt` or with the click of a button. In case the interaction fails to prove the PO, a message is printed to the console. Otherwise, the following steps take place. First, a "most-general", sound, proof rule is created as described above. Second, the rule is added to the "pmm" file, a file containing the rules for the current component, in a section named `SMT_Rules` (actually, the rule is added only if no other equivalent rule is already present). Third, a command to apply the rules from `SMT_Rules` is issued to the Atelier B solver. The current PO is then proved. The interactive prover either loads the next proof obligation or signals completion of the session. In practice, the `smt` command is successful as soon as the proof only involves arithmetics, equalities and properties of Boolean operators. The pmm file of the component then contains all the rules that have been created through this command, thus guaranteeing compliance with certification requirements.

All the proof rules introduced in a development process should be verified. This includes the rules created by drudges. For this activity, the *pp* prover is applicable and practice shows that it is able to verify many such rules. For the rest, a manual proof must be performed.

5 Conclusion and Future Work

We extend the proof assistant of Atelier B with a command to run SMT solvers in a proof session. The main requirements to use a solver in our framework are

efficiency and a function returning the hypotheses used in a proof. This extension then creates and applies automatically proof rules.

For certification, these rules need to be verified. The *pp* solver of Atelier B is able to check only part of these rules. The remainder need to be verified and validated manually. We envision the following alternatives: to put a bridle to the function by only adding rules that are checked with *pp*; to develop a more efficient rule checker; to translate the proofs generated by the SMT solvers into a human-readable proof. Future work also includes: to evaluate axiomatizing set operators for the SMT solvers, to experiment with SMT solvers handling set operators (namely, CVC4 [1]), to use other classes of solvers (e.g., TPTP provers).

References

1. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
2. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa (2017)
3. CENELEC - EN 50128: Railway applications-communication, signaling and processing systems-software for railway control and protection systems (2011)
4. Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. J. Satisf. Boolean Model. Comput. **9**, 207–242 (2016)
5. Couchot, J., Déharbe, D., Giorgetti, A., Ranise, S.: Scalable automated proving and debugging of set-based specifications. J. Braz. Comput. Soc. **9**(2), 17–36 (2004)
6. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: Integrating SMT solvers in Rodin. Sci. Comput. Program. **94**, 130–143 (2014)
7. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, vol. 2283. Springer Science & Business Media, Berlin (2002). https://doi.org/10.1007/3-540-45949-9
8. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In: PAAR@ IJCAR, pp. 1–10 (2010)