# Interfacing Automatic Proof Agents in Atelier B: Introducing "iapa"*

Lilian Burdy

ClearSy System Engineering, Aix-en-Provence, France

lilian.burdy@clearsy.com

David Déharbe[†]

david.deharbe@clearsy.com

ClearSy System Engineering, Aix-en-Provence, France

Étienne Prun

ClearSy System Engineering, Aix-en-Provence, France

etienne.prun@clearsy.com

The application of automatic theorem provers to discharge proof obligations is necessary to apply formal methods in an efficient manner. Tools supporting formal methods, such as Atelier B, generate proof obligations fully automatically. Consequently, such proof obligations are often cluttered with information that is irrelevant to establish their validity.

We present iapa, an "Interface to Automatic Proof Agents", a new tool that is being integrated to Atelier B, through which the user will access proof obligations, apply operations to simplify these proof obligations, and then dispatch the resulting, simplified, proof obligations to a portfolio of automatic theorem provers.

## 1  Introduction

Historically, the B Method[1] was introduced in the late 80's to design provably safe software. Promoted and supported by RATP, the B method and Atelier B, the tool implementing it, have been successfully applied to the industry of transportation leading to a worldwide implementation of the B technology for safety critical software, mainly as automatic train controllers for subways.

The development of such controllers corresponds to a "big" industrial project. To give an idea of the size of such a development, a train controller is composed of different software components communicating together. Taking from a real example, the size of the critical parts of a real-life controller is around 500.000 lines of B which give after translation 300.000 lines of Ada and 160.000 generated proof obligations. The proof, already mainly automated, of those proof obligations is a substantial part of the development cost for such projects. Limiting these costs by using more efficient provers, or by using more efficiently provers is a real concern in industry.

Atelier B comes with two provers developed in the 90's. Recently the ProB model checker [8] has been added as a prover that can be called during an interactive proof session. The BWare project[10] [7] aims to provide a mechanized framework to apply automated theorem provers, such as first order provers and SMT (Satisfiability Modulo Theories) solvers on proof obligations coming from the development of industrial applications using the B method. This approach produces proof obligations in the format of Why3 [3], which is used then responsible for calling different automatic theorem provers with the adequate input, interpreting their output and synthesizing a verification result.

---

SMT provers are routinely used by any tool that has to deal with a logic-based verification task. Notably, they have already been added as plugins in Rodin[6], another IDE for Event-B, which is a formal method closely related to the B method. Concerning BWare, promising first results[9] [4] have already been published.

We present here the integration of such automated provers in Atelier B considering the specificities of proof obligations produced by industrial software developments, described in Section 3. The basic elements of this integration are the Why3-based bridge to automated theorem provers developed in the BWare project and a new approach for efficiently selecting the relevant parts of a proof obligation. The principles of this latter aspect are presented in Section 4. The implementation of this functionality in a graphical user interface is then presented in Section 5.

## 2   Presenting the B method

The development of a project using the B method comprises two activities that are closely linked: writing formal texts and proving these same texts.

The writing activity consists in writing the specifications for *abstract machines* using a high level mathematical formalism. In this way, a B specification comprises data (that may be expressed among other ways using integers, Boolean values, sets, relations, functions or sequences), *invariant properties* that relate to the data (expressed using first order logic), and finally services that describe the initialization and possible evolutions of the data (data transformations being expressed as substitutions). The proof activity for a B specification comprises performing a number of demonstrations in order to prove the establishment and conservation of invariant properties in the specification (e.g. it is necessary to prove that a service call retains the invariant properties). The generation of such proof obligations is based essentially on the transformation of predicates using substitutions.

The development of an abstract machine continues during the successive *refinement* steps. Refining a specification consists in reformulating it so as to provide more and more concrete solutions, but also to enrich it. The proof activity relating to refinements performs a number of static checks and demonstrations in order to prove that the refinement is a valid reformulation of the specification.

## 3   Proof obligations

Verifying program by proving verification conditions (called proof obligations here) leads to deal with huge lemmas. This problem is not specific to the B Method, indeed the same observation is done, for example, in [5] for verification conditions issued from C program verification.

Concerning the B Method[1], Figure 1 shows a proof obligation template. One can see that hypotheses are collected in many clauses of many components. As an example, let us consider the declaration of constants and their properties: as the B Method is a modular software development method, constants are usually declared in some specific components. When a function needs to use a constant, the component in which the constant is declared is *seen*[1] and with the needed constant come all the properties of all the constants of the component. So all the proof obligations will have as hypotheses the properties of all the constants of the seen component even if only few of them are relevant for the proof.

---

[1]The SEES link is used to reference within a component, an abstract machine instance, to access its constituents (sets, constants and variables) without modifying them.

*"Machine parameter constraints"* ∧
*"Properties of constants of previous refinements"* ∧
*"Properties of refinement constants"* ∧
*"Properties of constants of components seen"* ∧
*"Properties of constants of components included"* ∧
*"Invariants and assertions of included components"* ∧
*"Invariants and assertions of components seen"* ∧
*"Invariants and assertions of the vertical development"* ∧
*"Precondition of the abstract operation"*
⇒
*"Precondition of the refinement operation"* ∧
*"Refinement operation applied to the negation of the specified operation applied to the negation of the invariant"*

Figure 1: Refinement proof obligation template

The incremental approach to refinement is another source of growth in the size of the proof obligations. Indeed, in each new refinement, all the proof obligations will include the contexts from all the components that come before in the refinement chain.

In the real project described previously, the average number of hypotheses for a lemma is around 2000 formulas. Some proof obligations can contain more that 4000 hypotheses. Of course, not all of them are necessary in the proof of each goal. To use efficiently automated provers on such lemmas, we argue that it is necessary to filter relevant hypotheses. This is the motivation for the development of an "Interface to Automatic Proof Agents" (IAPA), giving the users of Atelier B the means to build scripts constructing a mini-lemma from a generated proof obligation, and to submit such mini-lemmas to the provers.

## 4 Core functionality in iapa

Atelier B already contains an interface to discharge proof obligations: the interactive prover, where users spend most of their time. Some functionalities in iapa are similar to those found in the interactive prover and will be familiar to the users.

The iapa tool is invoked within Atelier B on a given component, once the proof obligations of that component have been generated (as with the interactive prover). In Atelier B, all the proof obligations of a component are available in a single file, and can be either in a legacy format called the "theory language" or in a XML-based format. Only the latter contains typing annotations and this format has been chosen in the BWare initiative as the basis for interfacing with the Why3 platform. Thus, iapa reads the proof obligations of a given component from the XML-based file.

In the iapa interface, proof obligations are grouped according to their origin in the corresponding B component (assertions, initialization, operations). Navigating through these proof obligations is a first core functionality available in iapa, and its principles mimick those of the interactive prover. Regarding this aspect, one important difference with respect to the interactive prover is that well-definedness proof obligations are presented together with those proof obligations instead of in a separate project.

Formally, a proof obligation is a pair $(\Gamma, \varphi)$, where $\Gamma$ is a set of hypotheses and $\varphi$ is the goal. The main goal of iapa is to assist the user in selecting the relevant information in the current proof obligation. This consists in building a new proof obligation $(\Gamma', \varphi)$ such that $\Gamma' \subseteq \Gamma$. In iapa, the new, simplified, proof obligation is termed the *lemma*. The interface also contains means to submit lemmas to a portfolio of automatic theorem provers.

One notable requirement of iapa is that the steps leading to the construction of a lemma for a given proof obligation can be applied automatically to other proof obligations. This is achieved by means of two kinds of entities: *contexts* and *lexicons*, that the user has to manipulate and combine in order to build a lemma.

A context $\gamma \subseteq \Gamma$ is a set of hypotheses that originates from the proof obligation $(\Gamma, \varphi)$. When a proof obligation is loaded in iapa, a number of contexts are pre-defined:

- all contains all the hypotheses;
- local contains all the hypotheses that are local in the proof obligation, i.e. hypotheses stemming from conditions found in the operation corresponding to the proof obligation;
- global contains all but the local hypotheses;
- a number of contexts that correspond to the different sections in a B component (properties, invariants, etc.);
- B definitions contains hypotheses on pre-defined sets such as implementable integers.

A lexicon $l$ is a set of free identifiers of the original proof obligations. Assuming fv returns the set of free identifiers in a formula, then $l \subseteq \bigcup \{\text{fv}(\psi) \mid \psi \in \Gamma \cup \{\varphi\}\}$. Initially, there is a single pre-defined lexicon, named goal, and containing $\text{fv}(\varphi)$ (the free identifiers in the goal).

At any time, the state of iapa contains the following elements:

- $(\Gamma, \varphi)$ the current proof obligation;
- $\mathscr{C}$: a set of contexts ($\forall x \in \mathscr{C}, x \subseteq \Gamma$);
- $\mathscr{L}$: a set of lexicons ($\forall l \in \mathscr{L}, l \subseteq \bigcup \{\text{fv}(\psi) \mid \psi \in \Gamma \cup \{\varphi\}\}$);
- $c$: a current context ($c \in \mathscr{C}$);
- $l$: a current lexicon ($l \in \mathscr{L}$);
- $S$: a set of selected hypotheses ($S \subseteq \Gamma$).

The pre-defined values for $\mathscr{C}$ and $\mathscr{L}$ are as described previously, those of $c$, $l$ and $S$ are local, goal, and $\emptyset$, respectively. Then the commands on contexts and lexicons currently implemented in iapa are the following:

- ah adds the hypotheses in the current context to the set of selected hypotheses;
- dh removes the hypotheses in the current context from the set of selected hypotheses;
- chctx(ctx) sets the current context to ctx;
- chlex(lex) sets the current lexicon to lex;
- mklex creates a new lexicon with the free identifiers of the current context;
- mklex(i1,..., in) creates a new lexicon with the given identifiers;
- mkctx(Some) creates a new context containing the hypotheses in the current context that have at least one free identifier in the current lexicon;

| command | effect | condition |
|---|---|---|
| ah | $S := S \cup c$ | |
| dh | $S := S \setminus c$ | |
| chctx(ctx) | $c := \mathsf{ctx}$ | $\mathsf{ctx} \in \mathscr{C}$ |
| chctx(lex) | $l := \mathsf{lex}$ | $\mathsf{lex} \in \mathscr{L}$ |
| mklex | $\mathscr{L} := \mathscr{L} \cup \{\mathsf{fv}(c)\}$ | $\mathsf{fv}(c) \neq \emptyset$ |
| mklex(i1,...,in) | $\mathscr{L} := \mathscr{L} \cup \{\{\mathsf{i1}, \cdots, \mathsf{in}\}\}$ | $\mathsf{i1} \in l \cdots \mathsf{in} \in l$ |
| mkctx(Some) | $\mathscr{C} := \mathscr{C} \cup \{\{h | h \in c \wedge \mathsf{fv}(h) \cap l \neq \emptyset\}\}$ | $\{h | h \in c \wedge \mathsf{fv}(h) \cap l \neq \emptyset\} \neq \emptyset$ |
| mkctx(All) | $\mathscr{C} := \mathscr{C} \cup \{\{h | h \in c \wedge l \subseteq \mathsf{fv}(h)\}\}$ | $\{h | h \in c \wedge l \subseteq \mathsf{fv}(h)\} \neq \emptyset$ |
| mkctx(h1,...,hn) | $\mathscr{C} := \mathscr{C} \cup \{\{\mathsf{h1}, \cdots, \mathsf{hn}\}\}$ | $\mathsf{h1} \in c \cdots \mathsf{hn} \in c$ |

Table 1: Formalization of iapa commands.

| script | description |
|---|---|
| ah | builds the lemma containing only local hypotheses |
| chctx(all) & ah | builds the lemma identical to proof obligation |
| mklex & chctx(all) & mkctx(Some) & ah | builds the lemma with hypotheses containing an identifier in the local hypotheses |

Table 2: Example iapa scripts (& being used as command separator).

- mkctx(All) creates a new context containing the hypotheses in the current context such that their free identifiers includes the current lexicon;

- mkctx(h1,...,hn) creates a new context containing the given hypotheses.

The condition and effect of the execution of these commands are summarized in Table 1, and some illustrative scripts are presented in Table 2. The following section presents the iapa interface, including how the user can play such commands.

# 5   The iapa tool

The core functionality is presented in a graphical user interface that has to be launched from Atelier B's main window on a given component. The functionalities are offered both by textual and point-and-click means. Figure 2 contains a screenshot of the initial contents of the window. At that point, two views are populated: *Provers* and *Proof obligations*. The contents of the latter is found in Atelier B's database. The contents of the former is obtained by querying the automatic provers currently installed. Since, at the moment, the access to these provers is realized through Why3, this information is found automatically using Why3 and its auto-configuration facilities. Besides the menu and the tool bar found at the top of the window, the *Command* section contains a widget containing a command-line interface to iapa. Here, the user has already typed the command ne, which, when executed, will open the next proof obligation.

Opening a proof obligation results in filling the *Goal*, *Context manager* and *Lexicon manager* sections, and enables actions corresponding to the core iapa functionalities. Figure 3 shows the contents of the iapa window after the user has managed to complete a proof after having selected some hypotheses in the context (using here one of the scripts presented in Table 2) and started the provers on the resulting lemma with the pr command. Scenarios where the proof obligation is not found valid correspond to the
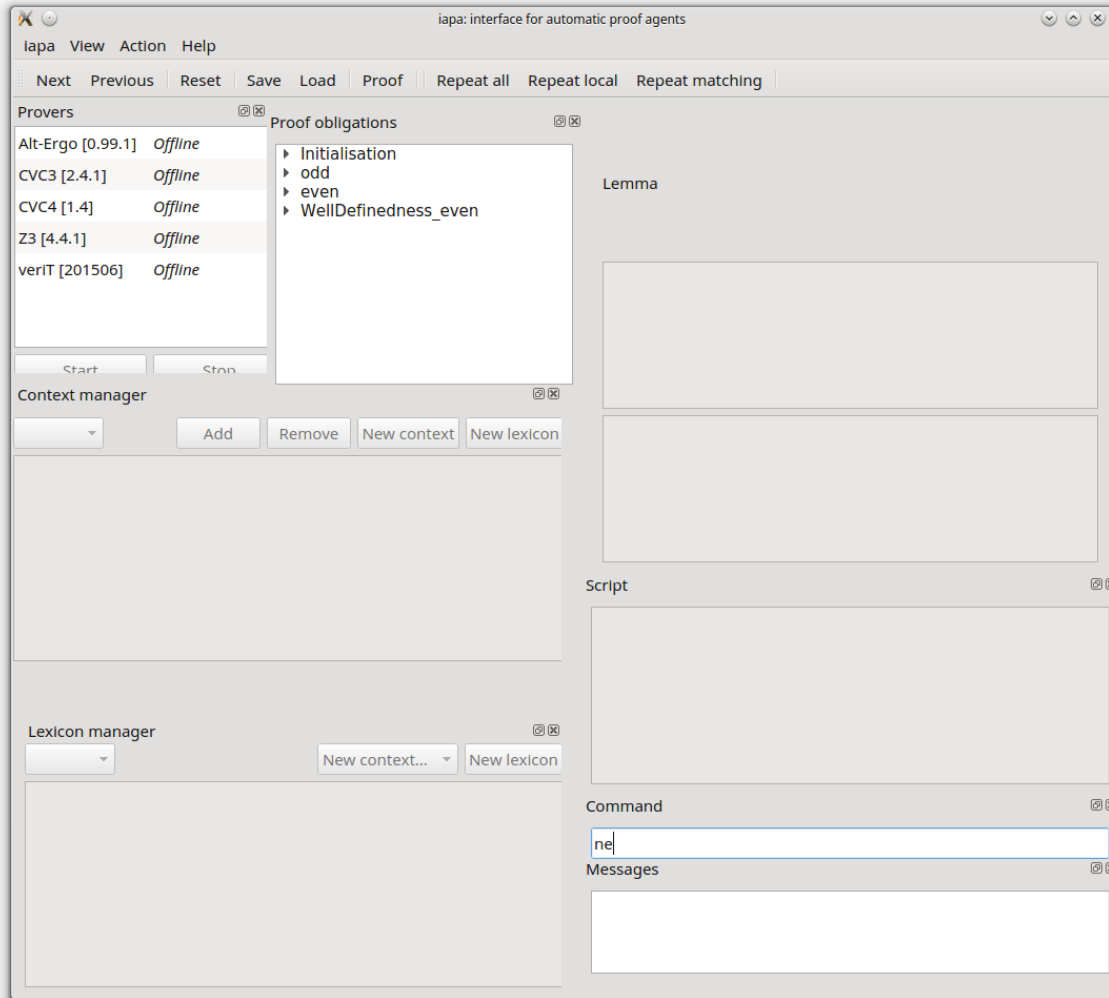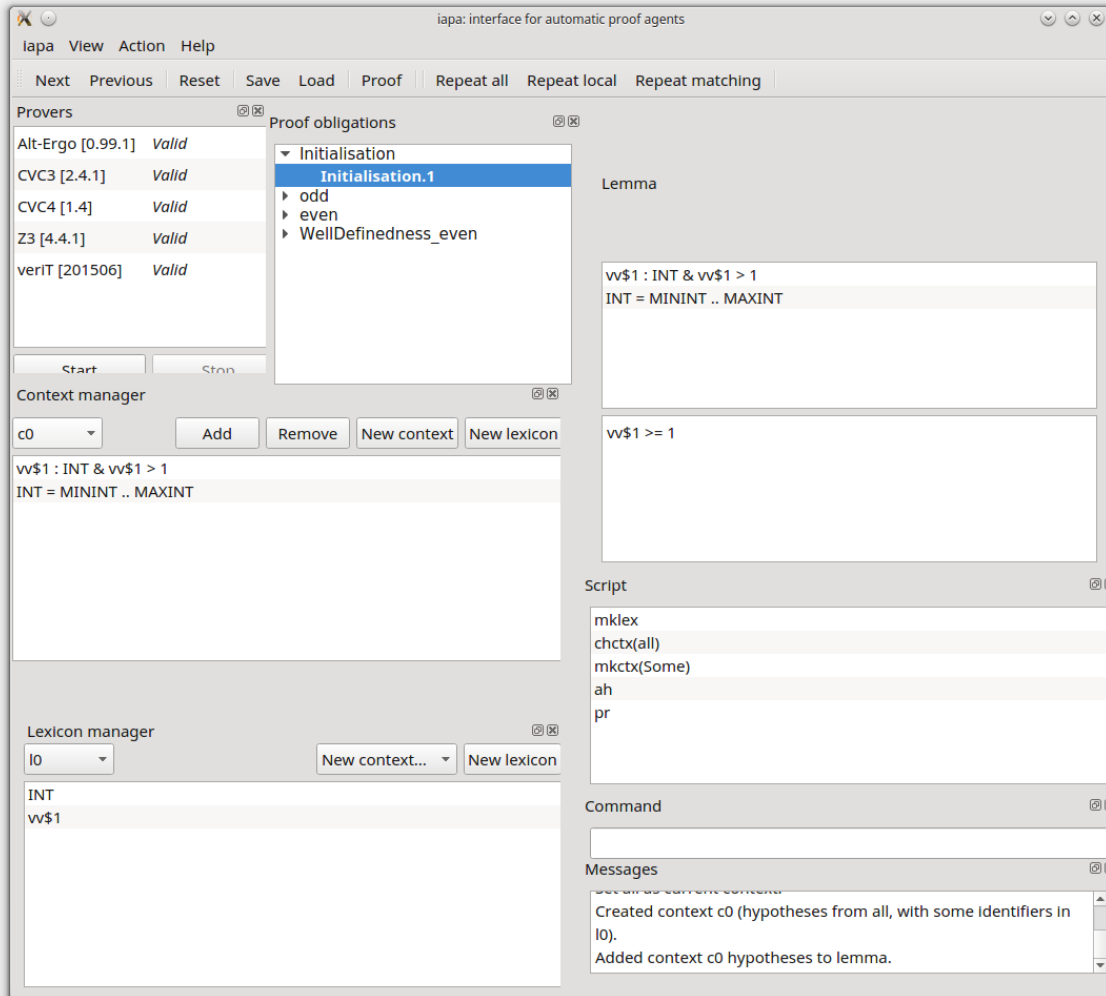
Figure 2: Initial contents of the iapa window.

Figure 3: Contents of the iapa window during a session.

following cases: the original proof obligation is not valid, the user did not select a relevant subset of hypotheses, or the tools did not find the proof.

In any case, the steps realized by the user are saved and displayed in the *Script* section. Also the *Messages* section is dedicated to the display of feed-back information.

## 6   Conclusions and future work

This paper presents an on-going work to reduce the cost of discharging proof obligations when applying formal methods in an industrial environment. This work is embodied in iapa, a prototype for an extension of Atelier B aiming at both integrating additional proof engines and offering hypotheses selection facilities to the user.

The effectiveness of the approach will be assessed through a systematic evaluation on a representative set of industrial projects. The results of this evaluation will decide whether iapa is eventually deployed together with the distributions of Atelier B. We also plan to improve the usability of iapa, by adding hypotheses selection criteria based on formula patterns, and also taking user feed-back into account.

Certification is another important aspect of tooling in an industrial setting for safety-critical systems. The historical provers in Atelier B have been certified but certifying new tools is costly. We forecast some solutions to address this issue in iapa. First, since some automated theorem provers are proof-producing, we envision using the proofs thus produced to build proofs that can be played by the certified provers. Second, since redundancy is also a mean to achieve desired safety levels, a second tool chain could be developed. It would bypass Why3 and generate proof obligations directly in the input language of the automatic provers. An approach similar to [6], targeting the SMT-LIB format [2] is a good candidate.

## References

[1] Jean-Raymond Abrial (2005): *The B-book - assigning programs to meanings*. Cambridge University Press.

[2] Clark Barrett, Pascal Fontaine & Cesare Tinelli (2015): *The SMT-LIB Standard: Version 2.5*. Technical Report, Department of Computer Science, The University of Iowa. Available at www.SMT-LIB.org.

[3] François Bobot, Jean-Christophe Filliâtre, Claude Marché & Andrei Paskevich (2011): *Why3: Shepherd your herd of provers*. In: *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pp. 53–64.

[4] Sylvain Conchon & Mohamed Iguernelala (2014): *Tuning the Alt-Ergo SMT Solver for B Proof Obligations*. In Yamine Ait Ameur & Klaus-Dieter Schewe, editors: *Abstract State Machines, Alloy, B, TLA, VDM, and Z: 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 294–297, doi:10.1007/978-3-662-43652-3_27.

[5] Jean-François Couchot & Thierry Hubert (2007): *A Graph-based Strategy for the Selection of Hypotheses*. In: *FTP'07, Int. Workshop on First-Order Theorem Proving*, Liverpool, UK.

[6] David Déharbe, Pascal Fontaine, Yoann Guyot & Laurent Voisin (2012): *SMT Solvers for Rodin*. In: *Proceedings of the Third International Conference on Abstract State Machines, Alloy, B, VDM, and Z*, ABZ'12, Springer-Verlag, Berlin, Heidelberg, pp. 194–207, doi:10.1007/978-3-642-30885-7_14.

[7] David Delahaye, Catherine Dubois, Claude Marché & David Mentré (2014): *The BWare Project: Building a Proof Platform for the Automated Verification of B Proof Obligations*. In: *Abstract State Machines, Alloy, B, VDM, and Z (ABZ)*, pp. –, doi:10.1007/978-3-662-43652-3_26.

[8] Michael Leuschel & Michael Butler (2003): *ProB: A Model Checker for B*. In Keijiro Araki, Stefania Gnesi & Dino Mandrioli, editors: *FME 2003: Formal Methods: International Symposium of Formal Methods*

*Europe, Pisa, Italy, September 8-14, 2003. Proceedings*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 855–874, doi:10.1007/978-3-540-45236-2_46.

[9] David Mentré, Claude Marché, Jean-Christophe Filliâtre & Masashi Asuka (2012): *Discharging Proof Obligations from Atelier B using Multiple Automated Provers*. In Steve Reeves & Elvinia Riccobene, editors: *ABZ - 3rd International Conference on Abstract State Machines, Alloy, B and Z, Lecture Notes in Computer Science* 7316, Springer, Pisa, Italy, pp. 238–251, doi:10.1007/978-3-642-30885-7_17. Available at `https://hal.inria.fr/hal-00681781`.

[10] BWare team (2012): *The BWare Project*. Available at `http://bware.lri.fr/`.