



Edition	1
Revision	0
Number of pages	28
Status	Review

# Third-Party Provers in Atelier B:

## Manual

	Redaction	Verification	Approval
Nom	D. Déharbe		
Date			

# Revision Table

Version	Date	Contributors	Contribution
1.0	xx/02/2021	D. Déharbe (Clearsy)	Initial version

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Principles</b>	<b>7</b>
2.1	Proof mechanisms . . . . .	7
2.2	Trust issues . . . . .	8
2.3	Disproving . . . . .	8
<b>3</b>	<b>Atelier B Assets and Configuration</b>	<b>9</b>
3.1	Third-party provers . . . . .	9
3.2	Writers and Readers . . . . .	10
3.3	Proof mechanisms . . . . .	10
3.3.1	Available proof mechanisms . . . . .	10
3.3.2	GUI Tool Bar Configuration . . . . .	11
<b>4</b>	<b>Project-level Configuration</b>	<b>15</b>
4.1	Migrating a project to NG mode . . . . .	15
4.1.1	Setting a project in NG mode in the GUI . . . . .	16
4.1.2	Setting a project in NG mode in the CLI . . . . .	16
4.2	Managing Proof Mechanisms . . . . .	17
4.2.1	Managing Proof Mechanisms in the GUI . . . . .	17
4.2.2	Managing Proof Mechanisms in the CLI . . . . .	17
<b>5</b>	<b>Using External Provers in the GUI</b>	<b>19</b>
5.1	Proving . . . . .	19
5.2	Replaying proofs . . . . .	20
5.3	Concurrency . . . . .	20
5.4	Metrics . . . . .	20
<b>6</b>	<b>Using External Provers in the CLI</b>	<b>23</b>

<b>A XML Format for Proof Mechanisms</b>	<b>25</b>
<b>B Creating New Proof Mechanisms</b>	<b>27</b>
B.1 Constraints on writer tools . . . . .	27
B.2 Constraints on reader tools . . . . .	28

# Chapter 1

## Introduction

Since version 4.7 of Atelier B, third-party automatic theorem provers can be used to discharge the generated proof obligations. This document presents this new feature.

- Chapter 2 presents the framework that has been designed to integrate external provers capabilities in B-based development processes. The remaining chapters refer to concepts presented in this part of the document, so read it first entirely before proceeding to the rest of the document!
- Third-party provers are not distributed with Atelier B and needs to be installed separately. Interacting with them requires to convert proof obligations to their format and to interpret their output to update the status of the proof obligations. Chapter 3 describes the components and settings in Atelier B that perform such interaction.
- Each project needs to be set up to allow for the use of external provers. This configuration is described in 4.
- The GUI of Atelier B has been modified to use external provers. The available functionalities are described in 5.
- It is also possible to use external provers in the CLI program `bbatch`. The available commands are described in 6.

The external prover integration framework is flexible. It is possible to customize the existing framework and even to extend it by connecting new provers. Appendices A and B contain technical resources necessary for such customizations and extensions.



# Chapter 2

## Principles

Atelier B is capable of integrating third-party automatic theorem provers, with different input and output formats. The use of external provers is optional and is defined at the project level.

### 2.1 Proof mechanisms

All the interactions with external provers is realized with so-called *proof mechanisms*. So a proof mechanism essentially packages an external prover so that it may be applied to the proof obligations of a component.

When the user applies a proof mechanism to a component, all the proof obligations in that component that are still unproved are passed to the external prover and the result of the external prover are used to update the status of these proof obligations.

Actually a proof mechanism can package several external provers, or several calls to the same external provers, with different parameters. Then they are applied in turn, only on the remaining unproved proof obligations.

Since the external provers do not know the format of the proof obligations in Atelier B, these need to be first translated to one of the input format of the external prover. Also, each external tool may have a different output format. So, Atelier B needs to be able to interpret such output to update the status of the proof obligations. These needs are addressed in a proof mechanism by embedding the call to a prover into a so-called driver.

A *driver* is composed of three elements:

**prover** The third-party automatic theorem prover.

**writer** The program translating proof obligations to the input format of the prover

**reader** The program interpreting the prover output.

So, a proof mechanism is essentially a sequence of such drivers. Most proof mechanisms packaged with Atelier B contain only one driver though.

## 2.2 Trust issues

The B method is essentially used industrially to develop safety-critical software components. As such, it is necessary that the tools are qualified according to the role they have in the development process.

If a proof mechanism is not qualified, it can still be used in Atelier B, e.g. because it makes it easier to identify proof obligations that are still wrong (because the design is not finished)

When a proof mechanism is thus not qualified, then the proof obligations that are proved with it are classified as *Unreliably Proved*, a new proof status in Atelier B. When a proof obligation has this status, then it is highly probable that it is provable. In the design phase where some parts of the components are not finished, this makes it easier to identify which proof obligations are not yet provable.

## 2.3 Disproving

Some automatic theorem provers are not only able to prove, but also to disprove. When a proof obligation is not valid, and an external prover is able to show that it is not valid, then the status of the proof obligation is *Disproved*. This indicates directly that there the B component being verified has an error.



# Chapter 3

## Atelier B Assets and Configuration

### 3.1 Third-party provers

Atelier B is distributed with proof mechanisms that use the following third-party automatic theorem provers:

- CVC4: joint project of Stanford University and University of Iowa. Available under the BSD 3-clause license.
- Z3: from Microsoft Research. Available under the MIT License.

These provers are not distributed together with Atelier B, but they may be downloaded from their respective web sites for a variety of platforms. Their use is subject to their respective licences.

To use a proof mechanism, the referenced prover(s) must be installed separately.

The proof mechanisms use Atelier B resources to access the third-party provers. When such a prover has been installed, its path shall be used to set the corresponding resource. The resources are the following :

- CVC4: `ATB*Proof*CVC4`
- Z3: `ATB*Proof*Z3`

As with other resources, they may be set at the installation level and at the project level, in the `AtelierB` file.

To use a proof mechanism, the resource corresponding to the prover(s) must be set.

In case the prover requires a dynamic library at execution-time, the path to the library must be added in the execution environment of Atelier B. This procedure is platform-dependent.

## 3.2 Writers and Readers

Atelier B is distributed with programs that play the role of writer and reader in proof mechanisms. Each writer has an associated reader. They are:

- `ppTransSmt` and `smt_solver_reader` are companion tools for any provers compliant with the SMT-LIB 2.6 format, e.g. CVC4 and Z3. The writer handles all the operators of the B language.
- `pog2smt` and `simple_smt_solver_reader` are companion tools for any provers compliant with the SMT-LIB 2.6 format, e.g. CVC4 and Z3. The writer handles only those operators of the B language that have an equivalent in an SMT-LIB logic.

Further details on writers and readers may be found in appendix B.

## 3.3 Proof mechanisms

### 3.3.1 Available proof mechanisms

The proof mechanisms are defined as XML-based files that are stored in the *proof mechanism directory*, i.e. the directory `press/pm` of the distribution.

The following table lists all the proof mechanisms that are available with the distribution, together with the resources they depend on.

proof mechanism	resource
<code>cvc4_ddrp1_pp</code>	ATB*Proof*CVC4
<code>cvc4_pp</code>	ATB*Proof*CVC4
<code>cvc4_simple</code>	ATB*Proof*CVC4
<code>smtlib_simple</code>	ATB*Proof*CVC4 and ATB*Proof*Z3
<code>smtpp_rp0</code>	ATB*Proof*CVC4 and ATB*Proof*Z3
<code>smtpp_rp1</code>	ATB*Proof*CVC4 and ATB*Proof*Z3
<code>z3_ddrp1_pp</code>	ATB*Proof*Z3
<code>z3_pp</code>	ATB*Proof*Z3
<code>z3_simple</code>	ATB*Proof*Z3

Notice that some mechanisms requiring setting two resources. The reason is that these mechanisms try up to two automatic provers on each proof obligation.

In addition to providing access to different automatic provers, these proof mechanisms vary according to two additional features : hypotheses filtering, and logic encoding.

**Hypotheses filtering** This consists in selecting a subset of the hypotheses in the proof obligation to produce the input of the automatic prover. This selection is based on the fact that, historically, a proof obligation in B has three parts: global hypotheses (which may be in the hundreds), local hypotheses and a goal. In the interactive prover, the initial view of a proof obligation only contains the local hypotheses and the goal. There are four possible filters:

1. The rp0 filter, where only the local hypotheses and the goal are translated.
2. The ddrp1 filter, that contains the goal plus all the global hypotheses that share a symbol with the goal.
3. The rp1 filter, that contains the result of the rp0 filter plus all the global hypotheses that share a symbol with this result.
4. No filter, i.e., all the hypotheses and the goal are translated.

**Accuracy** This feature provides the possibility to abstract parts of the proof obligation. There are two levels of accuracy:

1. In the simple case, the set theory operators are left uninterpreted in the input of the automatic prover. This feature is only available for proof mechanisms embedding SMT solvers, and is suitable for the proof obligations where no set-based reasoning intervenes in the demonstration of the goal.
2. In the faithful case, the semantics of all the B operators is preserved in the input of automatic prover.

proof mechanism	filtering	accuracy
cvc4_ddrp1_pp	ddrp1	faithful
cvc4_pp	no	faithful
cvc4_simple	no	simple
smtlib_simple	no	simple
smtpp_rp0	rp0	faithful
smtpp_rp1	rp1	faithful
z3_ddrp1_pp	ddrp1	faithful
z3_pp	no	faithful
z3_simple	no	simple

### 3.3.2 GUI Tool Bar Configuration

Once the proof mechanisms of interests have been identified, the corresponding provers installed, and the corresponding resources set, these proof mechanisms may be added to the tool bar of the GUI.

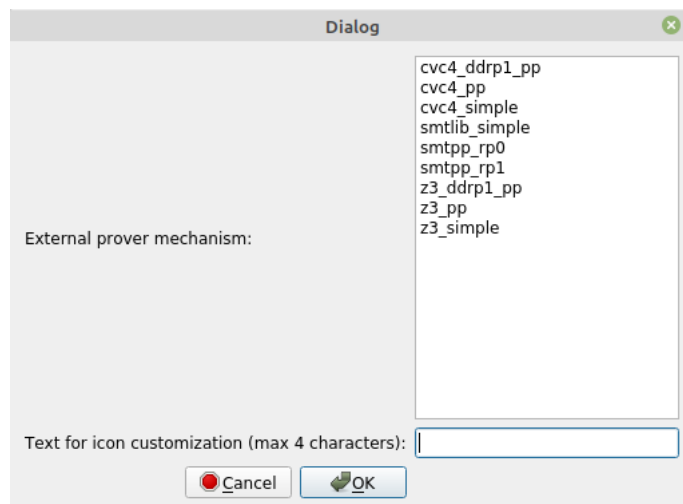
For each proof mechanism, an icon may be added to the tool bar. The user will then be able to use this icon to call the corresponding proof mechanism on the selected components.

To add a proof mechanism icon to the tool bar:

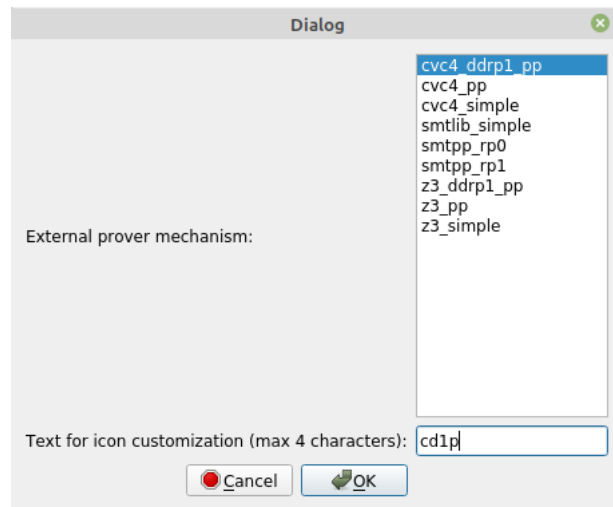
1. Click on the rightmost icon + of the tool bar.



2. An icon creation dialog pops up.



3. Select the proof mechanism from the list.
4. Provide an identifier (up to 4 characters) for this proof mechanism. This identifier will be used to produce the icon.
5. Click the OK button.



6. The icon appears in the tool bar.



Further details on proof mechanisms may be found in appendix B. Also, the description of the XML-based format for proof mechanisms can be accessed in the Manuals section of the Help menu in the GUI.



# Chapter 4

## Project-level Configuration

At the level of each project, two actions are required to use external provers:

1. Set the project to NG mode (described in § 4.1);
2. Add proof mechanisms to the project (described in § 4.2).

### 4.1 Migrating a project to NG mode

The use of external provers required to make changes to the structure of the project database in Atelier B, so once a project has been set up to use external provers, it is not possible to revert this action automatically.

To use a proof mechanism in a project, the project must be in so-called NG mode. Once in NG mode, a project cannot be reverted automatically to non NG mode.

When a project is in NG mode:

1. The proof obligation generator is the new proof obligation generator.
2. The primary format for proof obligation files is the XML-based format POG (instead of the theory language-based format PO).
3. The primary format for proof obligation status file is the XML-based format POS (instead of the theory language-based format PMI).
4. The files produced by the proof mechanism writers are stored in the database, but are immediately deleted after they have been used by the provers.
5. The resource `ATB*ATB*Project_Mode_NG` is set to `TRUE`.

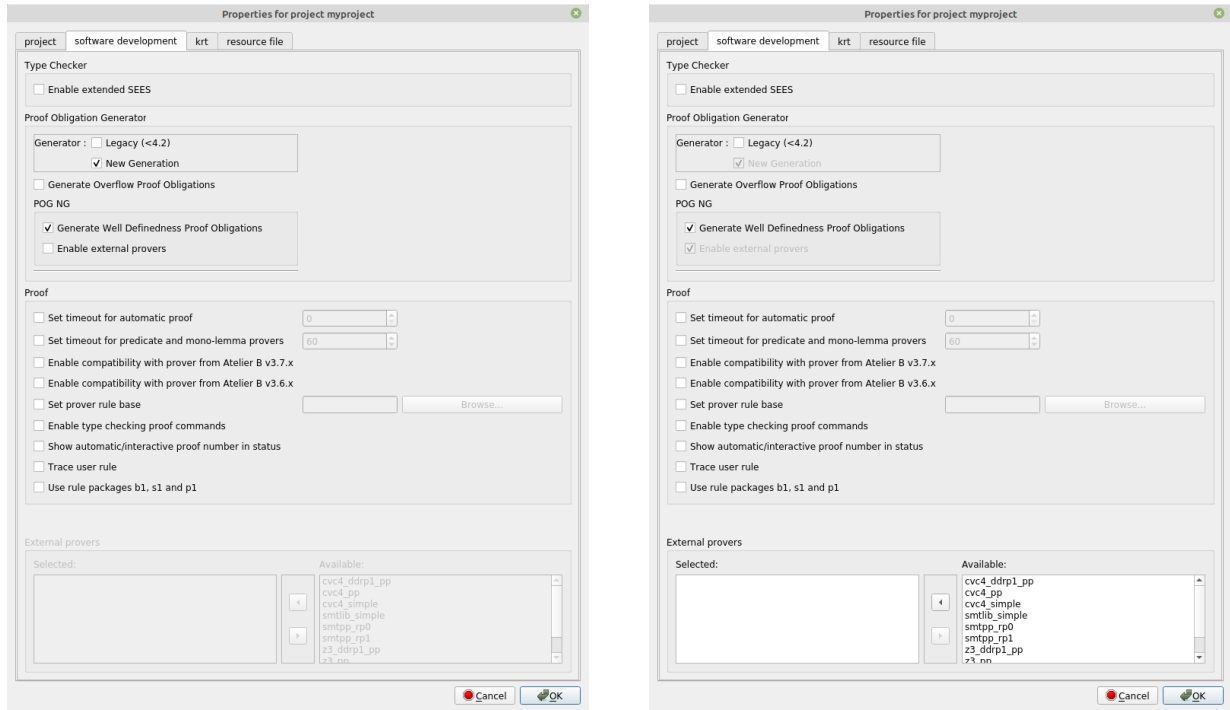


Figure 4.1: Migrating project to NG mode by checking the “Enable external provers”. This action is irreversible.

Setting the resource `ATB*ATB*Project_Mode_NG` to `TRUE` in the `AtelierB` file of a project migrates it to NG mode. The GUI and the CLI also provide means to migrate the project to NG mode.

### 4.1.1 Setting a project in NG mode in the GUI

To migrate a project to NG mode in the GUI, do the following steps (see also figure 4.1):

1. In the project view, right-click on the project and select “Properties”.
2. Click on the “software development” tab to put it to the foreground.
3. Check the box “Enable external provers”.

### 4.1.2 Setting a project in NG mode in the CLI

To migrate a project to NG in the CLI, do the following steps:



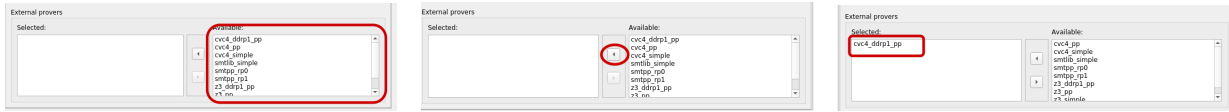


Figure 4.2: The views and buttons for managing proof mechanisms: the list of available proof mechanisms (left), the button to add a proof mechanism to the project (middle), the list of project proof mechanisms (right).

1. Open the project (command `op`).
2. Type the command `migrate_project` (short form: `mip`).

## 4.2 Managing Proof Mechanisms

The proof mechanisms stored in the directory `press/pm` are not usable in a project until they have been explicitly added to the project. This section describes how to manage the proof mechanisms available in a project.

### 4.2.1 Managing Proof Mechanisms in the GUI

The proof mechanisms of a project may be managed through the “Properties” dialog, in the “software development” tab.

External provers management is at the bottom of this tab. On the left, the “Selected” view lists all the proof mechanisms available in the project. On the right, the “Available” view lists all the proof mechanisms available in the installation but not in the project. The buttons between the two views add and remove the selected proof mechanism to the project (see figure 4.2).

### 4.2.2 Managing Proof Mechanisms in the CLI

The CLI commands to manage proof mechanisms are the following :

**add\_proof\_mechanism m** Includes **m** to the list of proof mechanisms authorized in the current project.

The files involved are:

- The database file of the project is updated to include the proof mechanism.
- The proof mechanism file `m.xml` must be present in the proof mechanism directory (see 3.3).

Short form: **apm m**

**remove\_prove\_mechanism m** Removes **m** from the list of proof mechanisms authorized in the current project.

The files involved are:

- The project database file of the project is updated to suppress the proof mechanism.

Short form: **rpm m**

**show\_project\_proof\_mechanisms** The command lists proof mechanisms authorized in the current project.

The information is taken from the project database file.

Short form: **sppm**

**show\_proof\_mechanisms** The command lists all proof mechanisms available in the installation.

The information is taken from the proof mechanism directory (see 3.3).

Short form: **spm**

# Chapter 5

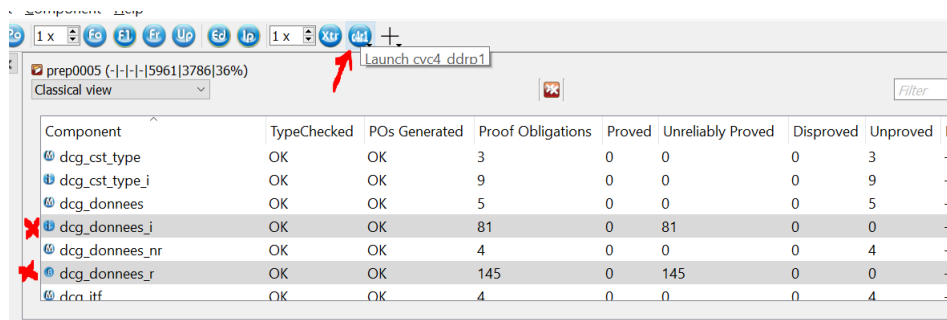
## Using External Provers in the GUI

The section describes how to use external provers to perform the proof activities in a project.

### 5.1 Proving

The actions described in this section require that the proof mechanisms have been added to the tool bar (see section 3.3.2) and that the proof mechanisms have been added to the project (see section 4.1.2).

To use proof mechanisms for discharging proof obligations on the selected components, click on the icon corresponding to that proof mechanism.



While the proof mechanism is running on the selected components, the corresponding tasks are listed in the task view.

Project	Component	Action	Status	Messages	Server
prep0005	dca_if.donnees_r		Running	Initialisation ...	localhost
prep0005	dca_if.donnees_i		Waiting		

When the tasks complete, the components view is updated to display the result of the proof mechanism. In that example, the proof mechanism is not trusted. So, even though all proof obligations were shown valid by the proof mechanism, they are counted as `Unreliably Proved`.

Component	TypeChecked	POs Generated	Proof Obligations	Proved	Unreliably Proved	Disproved	Unproved	B0 Checked
dcg_cst_type	OK	OK	3	0	0	0	3	-
dcg_cst_type_i	OK	OK	9	0	9	0	0	-
dcg_donnees	OK	OK	5	0	0	0	5	-
dcg_donnees_i	OK	OK	81	0	81	0	0	-
dcg_donnees_nr	OK	OK	4	0	0	0	4	-
dcg_donnees_r	OK	OK	145	0	145	0	0	-
dcg_itf	OK	OK	4	0	0	0	4	-

## 5.2 Replaying proofs

When proof obligations of a component have been discharged thanks to proof mechanisms, and the component has been unproved, the same proof mechanisms can be automatically applied to the component thanks to the “external replay” button found in the tool bar.



## 5.3 Concurrency

The execution of each proof mechanism spawns processes to execute translations and external provers. By default, the number of concurrently spawned processes is 1, but it is possible to spawn concurrently more than one process. This is controlled by a following widget located in the tool bar.



## 5.4 Metrics

The project status dialog contains an additional tab with the details of the results for each driver of each proof metrics. The information is displayed in a tabular format as follows:

État du projet prep0005

?

Proof status

External provers metrics

Component	General							Summary								
	Po	Pr	Utr	Dis	Unp	Ext	ATB	Pr	Utr	Dis	Unk	Unp	Pr	Dis	Unk	Unp

Here the project `prep0005` has only a single proof mechanism, called `cvc4_pp`. This proof mechanism has a single driver, named `cvc4`.





## Chapter 6

# Using External Provers in the CLI

The following commands are available in the `bbatch` command-line interface:

**extprove *c m* (0|1)** where *c* is a component; *m* is a proof mechanism.

Applies the proof mechanism *m* to the component *c*. Multiple components can be proved in a single command using `*` as a wildcard. The proof mechanism must be authorized in the current project.

If the third parameter is 1, then only drivers tagged as fast are executed, otherwise all drivers are executed.

This command reads the proof obligation file of the component, and writes secondary proof obligation files in a format suitable for the provers used by *m*. These files are stored in the directory `bdp/m`. They are temporary and deleted upon completion of the command.

Short form: **xtp**

**extreplay *c [m]*** where *c* is a component and *m* is a mechanism.

If *m* is provided, the command replays the proof mechanism *m* on the component *c*. Otherwise, the command replays all external proof mechanisms already applied to the component.

The replay command may be applied to multiple components by using the `*` wildcard.

Short form: **xtr**

**extmetrics** This is a project-level command. It displays a table of proof statuses of each component, giving details about the result of each proof mechanism.

Short form: **xtm**

**concurrency** [**m**] , where **N** is an optional positive number.

The concurrency level is the number of concurrent threads used by the commands ‘extprove’ and ‘extreplay’ to:

1. write secondary proof obligation files
2. call external provers on these secondary proof obligation files

When **N** is given, sets the concurrency level.

When **N** is not given, prints the concurrency level.

The default concurrency level is 1.

It can be set with resource `ATB*External_Proof*Concurrency`.

Short form: **co**



# Appendix A

## XML Format for Proof Mechanisms

The XML format for proof mechanisms is defined with an XML Schema named `proof-mechanism`. Its current version is 1.0. The documentation for this format is available as an HTML page in the GUI in the Manuals section of the Help menu. It is also available in the directory `documentation/formats` in the installation of Atelier B.



# Appendix B

## Creating New Proof Mechanisms

Creating a proof mechanism requires writing an XML file complying with the `proof-mechanism` XML Schema (see A). Of course the tools (writer, prover, reader) that compose the driver(s) making up this schema should also be available (see 2).

We recommend using the `extprove` command line program, distributed with Atelier B, to test that a proof mechanism behaves as expected. Indeed, the primary functionality of `extprove` is to interpret a proof mechanism on a component.

So it is possible to create new proof mechanisms by reutilizing the tools distributed with Atelier B that implement the translation between Atelier B's formats and that of existing automated provers. Such tools target the `whyml` and `SMT-LIB` format and it is thus very easy to create new proof mechanisms with automatic provers recognizing these formats.

It is also possible to target automatic provers recognizing other formats, or to design writer tools with a different level of accuracy than those currently available. The writer and reader tools are required to respect a number of conventions so that they are compatible with the algorithm implementing the evaluation of the proof mechanism in Atelier B.

### B.1 Constraints on writer tools

The *input* of a writer tool should be the file containing the proof obligations of a B component. Such files are produced by the proof obligation generator in Atelier B and they are compliant with the `pog` format. This format is XML-based. The documentation and the XML Schema for this format are available in the GUI in the Manuals section of the Help menu, and also in the directory `documentation/formats` in the installation of Atelier B.

The writer shall recognize the following parameters:

**-i ipath** The path of the input `pog` file is **ipath**.

- o **opath** The resulting translation shall be stored in the file **opath** (the contents of this file shall be in the format recognized by the prover).
- a **i j** The *j*-th child element tagged `Simple_Goal` in the *i*-th element tagged `Proof_Obligation` of the input file shall be translated. There might be several such parameters in case the solver support proving several goals in a single execution.

So each `-a` parameter given to the writer corresponds to a proof obligation of a component. The prover is expected to produce one result for each such proof obligation *in the same order as the `-a` parameters*. It is then the role of the reader tool to interpret this result and translate it to a unique format, as explained in the next section.

## B.2 Constraints on reader tools

When the prover is executed on the file produced by the writer, the standard output channel of the prover is piped to the standard input channel of the reader. For each goal processed by the prover, the reader shall produce on its standard output channel a single line containing one of the following strings (and nothing else): `proved`, `disproved` and `unknown`. The order of the status printed shall correspond to the order of the `-a` parameters given to the writer.

The status of the component is updated using these strings, according to the policy specified in the proof mechanism.