Version 4.3.1

Date of diffusion : February 2016

The Atelier B 4.3.1 is a ***Maintenance Edition*** version, which access is restricted to Atelier B 4 maintenance contract holders (corrective maintenance, anticipated access to new features/tools).

New Functionalities / Characteristics:

Atelier B 4.3.0 has been released on February 18th, 2016

This version fixes 27 bugs and 2 improvements are included:

- Checking of coding rules inside B models.
- Integration of ProB model-checker inside the interactive prover interface

B coding rule checking tool

A new module which allows the user to perform coding rule checking on B models has been developed[1].

It consists of one executable called « bcrc » (standing for « B coding rule checker ») which is present in AtelierB installation directory. It is launched through a dedicated GUI which can be opened with a menu entry. The « bcrc » executable can also be launched in command line if needed.

Configuring the rule checking

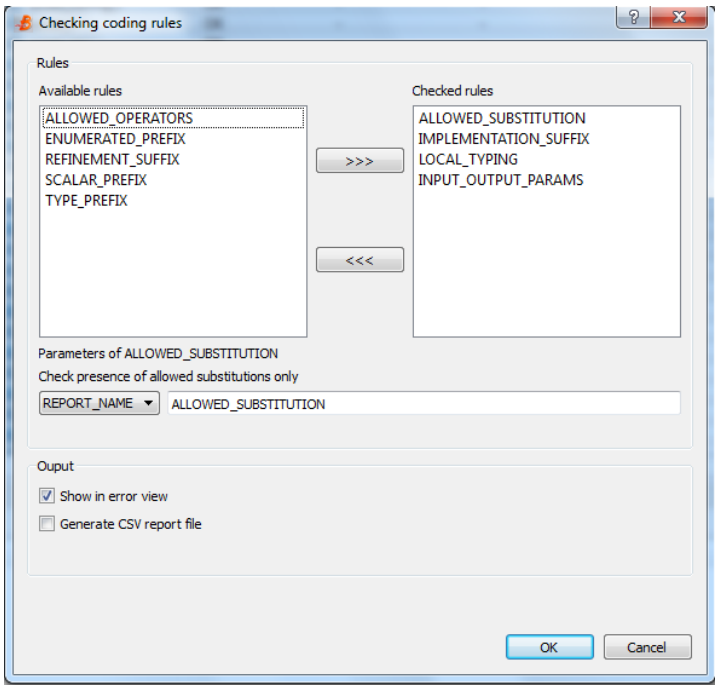
Checking coding rules in the AtelierB GUI is a component level action, an entry in « Component » menu has been added. This action may be performed on one or several components at a time. Results of rule checking on several components will be gathered.

The tool is based on syntactical and semantical analyzers provided in AtelierB, so it should be launched on components which can be successfully type-checked. If it is used on some incorrect components, type-check errors will be displayed in the "Error" view, but the coding rule checking will not be performed on these components.

Selecting « Check coding rules » action pops up a configuration window which shows the rules that can be verified.

Double-clicking on a rule in one of the two top frames of the window displays the rule parameters.

The user can then modify parameter values. For parameters accepting several values among a finite set, a list of different possible values is displayed, and the text field shows by its colorization if current value is correct.

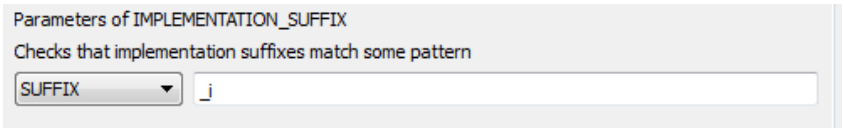


The GUI contains also two check boxes used to specify how results of coding rule checking must be shown. These results can be either displayed in the "Error" view of AtelierB main GUI, or written in an output CSV file.

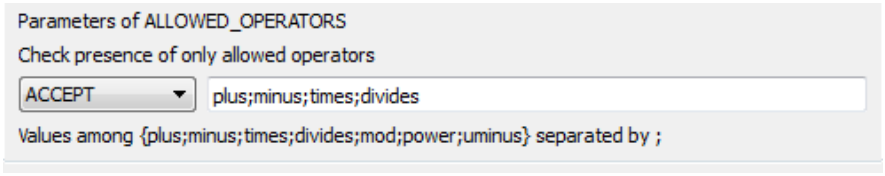
All rules accept a parameter called "REPORT_NAME" which is used to modify the name the rule will be described with if the user chose to log coding rule violations in a file.

Below is an example of setting a simple string parameter. In this case this is the suffix that implementation names must end with.

[1] With support of Alstom



And then we present an example of a multiple value parameter, here the arithmetic operators that are allowed in implementations



Below table describes all rules provided by the tool, as their identifier inside the GUI.

Provided rules

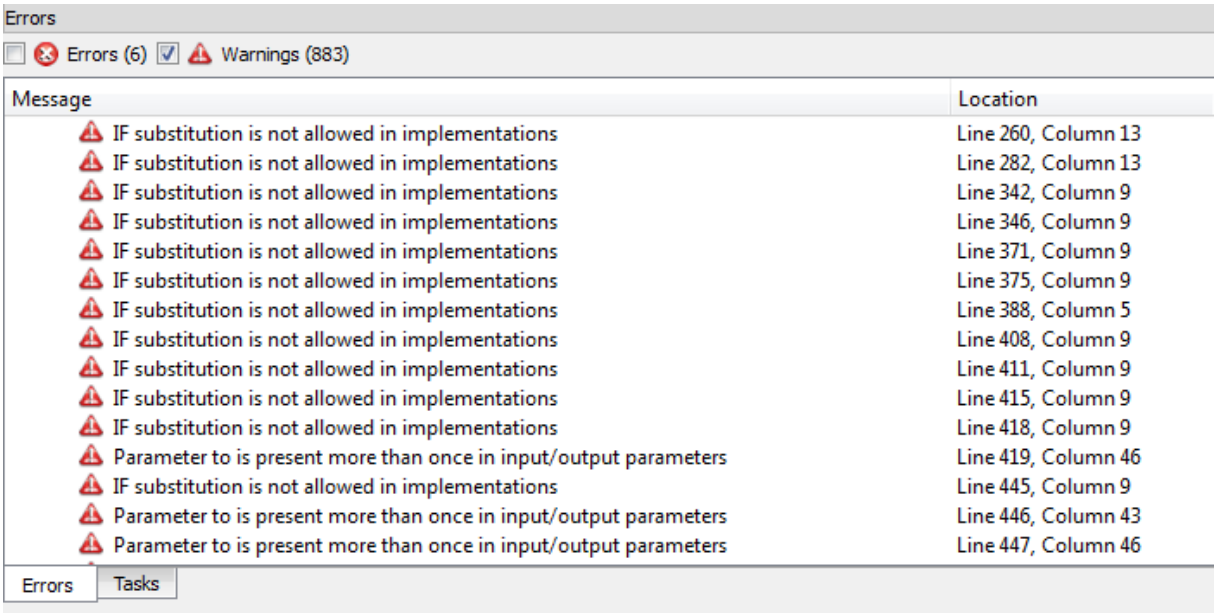
Rule	Description
Refinement suffix (REFINEMENT_SUFFIX rule)	This rule accept the string parameter named "SUFFIX". Its default value is "_r". The rule checks that the name of refinement components (beginning with REFINEMENT keyword) ends with the suffix chosen by user.
Implementation suffix (IMPLEMENTATION_SUFFIX rule)	This rule accepts the string parameter named "SUFFIX". Its default value is "_i". The rule checks that the name of implementation components ends with the suffix chosen by user.
Type prefix (TYPE_PREFIX rule)	This rule accepts the string parameter named "PREFIX". Its default value is "T_". The rule checks that identifiers of elements declared in component SETS clauses start with the prefix chosen by user.
Scalar constant prefix (SCALAR_PREFIX rule)	This rule accepts the string parameter named "PREFIX". Its default value is "c_". The rule checks that identifiers of constants declared in ABSTRACT_CONSTANTS or CONCRETE_CONSTANTS with a type included in INTEGER start with the prefix chosen by user.
Enumerated value prefix (ENUMERATED_PREFIX rule)	This rule accepts the string parameter named "PREFIX". Its default value is "e_". The rule checks that identifier of enumerated values declared in SETS clauses of components start with the prefix chosen by user.
Authorized arithmetic operators (ALLOWED_OPERATORS rule)	This rule accepts the multiple value parameter named "ACCEPT". Its value is a list of arithmetic operators among : • 'plus' • 'minus' : susbtraction

	• 'times' : multiplication • 'divides' : integer division • 'mod' : modulo • 'power' • 'uminus' : Unary minus The rule checks that only arithmetic operators chosen by user in the list are present in implementations. The default value for the « ACCEPT » list is « plus ;minus ;times ;divides ».
Authorized substitutions (ALLOWED_SUBSTITUTION rule)	This rule accepts the multiple value parameter named "FORBIDDEN". Its value is a list of substitution types among: • 'begin' • 'skip' • 'becomes_equal' • 'becomes_such_that' • 'assert' • 'if' • 'case' • 'var' • 'while' The rule checks that substitutions of the list do not appear in implementations (only in code parts of OPERATIONS clause). By default FORBIDDEN parameter is empty.
Parameter present twice in operation calls (INPUT_OUTPUT_PARAMS rule)	A parameter cannot be used twice in an operation call.
Local variable typing (LOCAL_TYPING rule)	Variable declared in VAR IN substitutions must be typed at the beginning in « becomes such that » substitutions.

Displaying the results

Results of this checking functionality can be displayed as errors in the main « Error » view of AtelierB, so that associated locations in the model can be reached by the user in order to directly write a correction. This is done by checking the suitable box in the configuration pop-up. Violations of coding rules are displayed in this view with criticality «Warning ».

Below example shows results of a coding rule verification, including the rules INPUT_OUTPUT_PARAMS (parameters must not be present twice in operation calls) and ALLOWED_SUBSTITUTION (only allowed substitutions can be present in implementations) configured to forbid IF substitution – with value "if" for parameter FORBIDDEN.



User can also choose to write the violations of coding rules in an output CSV file by checking the suitable box in the GUI. In this case only real typing or syntax errors provided by B compiler during analysis of components to check are displayed in the main view.

Extensibility

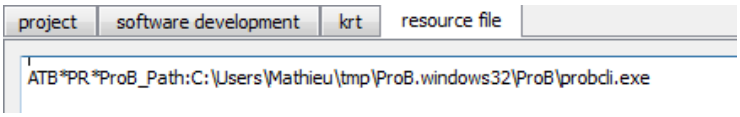
New rules will be included in the tool, depending on new needs defined by users

Integration of ProB model-checker in the interactive prover

This new version provides a way for the user to launch the ProB model-checker in the interactive prover as an interactive command which can be used inside a proof.

For this command to be used, ProB must be installed on the computer, and the resource ATB*PR*ProB_Path in AtelierB resource file must contain the path to procli executable. If not, the user will get a message « The Prob_Path resource is not set ».

This example shows how to set the resource in a B project created with Windows version of AtelierB.



The name of the new interactive proof command is **`prob`**, and it has two different syntaxes.

Command	Description
`prob(n)`	Launches ProB on the current goal. The parameter n is similar to the one in pp(rp.n), here the machine given to ProB as input is built using hypothesis provided by rp.n

`prob(n|t)`	Similar to prob(n) but limits the running time to t seconds

Using this new command actually generates a machine containing the goal as an assertion. If there are some hypothesis H coming from the **`rp.n`** when **`prob(n)`** is used to prove the goal G, they are also written in the assertions clause of this machine so that it will contain H => G. Predicate needed for typing are written in the PROPERTIES clause of this machine.

Then ProB is called with this temporary machine, in the mode which searches for counter examples for ASSERTIONS clause content. Then 3 cases may occur:

- ProB can check the exhaustive set of values for the variables contained in the formula H => G and no counter example is found: **current proof branch is proved**
- ProB finds a counter example for the temporary machine assertion: **the command fails to prove the branch, a notification is written in Message view of the GUI**
- ProB cannot process the exhaustive set of value for the variables of the assertion clause: **the command fails to prove current branch**