Version 4.3.1

Date de diffusion : Février 2016

L'Atelier B 4.3.1 est une version ***Maintenance Edition***, dont l'accès est réservé aux possesseurs d'un contrat de maintenance Atelier B 4 qui disposent ainsi d'un support technique (maintenance corrective) ainsi que d'un accès anticipé aux nouvelles fonctionnalités.

Nouvelles fonctionnalités / Caractéristiques :

L'Atelier B 4.3.1 a été mis à disposition le 18 février 2016.

Cette version corrige 27 anomalies et 2 améliorations sont ajoutées:

- Vérification des règles de codage au sein de l'AtelierB
- L'intégration du model-checker ProB dans le prouveur interactif

Outil de vérification des règles de codage

Un nouveau module permettant de vérifier des règles de codage dans les sources d'un modèle B a été développé[1].

Il consiste en un exécutable « bcrc » (pour « B coding rule checker ») présent dans le répertoire d'installation. Il s'exécute au travers d'une IHM spécifique accessible par menu. L'exécutable « bcrc » peut aussi être lancé en mode commande si nécessaire.

Configuration de la vérification

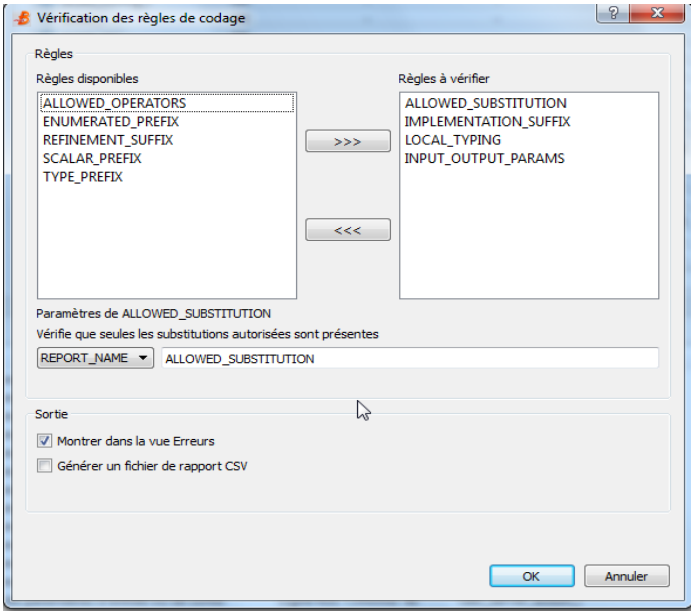
L'action de vérifier les règles de codage se fait dans l'interface au niveau composant, une entrée dans le menu « Composant » y est donc associée. Cette action peut être réalisée sur un ou plusieurs composants simultanément. Les résultats de vérification de cet ensemble de composants seront regroupés.

L'outil s'appuie sur les analyseurs syntaxiques et sémantiques de B présents dans l'AtelierB, par conséquent il doit être lancé sur des composants dont la vérification de type est un succès, afin de présenter un résultat pertinent. S'il est lancé sur des composants présentant des erreurs au sens du type-check, celles-ci seront détectées et affichées dans la vue « Erreurs », mais la vérification des règles de codage ne sera pas effectuée sur ses composants.

La sélection par l'utilisateur de l'action « Vérifier les règles de codage » provoque l'apparition d'une fenêtre de configuration qui permet de sélectionner les règles à vérifier.

Un double-clic sur une règle dans les cadres du haut affiche ses paramètres.

L'utilisateur peut ensuite modifier les valeurs des paramètres. Pour les paramètres qui acceptent plusieurs valeurs parmi un ensemble fini, une liste des différentes valeurs possibles est affichée, et la fenêtre indique par la colorisation de la zone de texte si la valeur entrée est correcte.

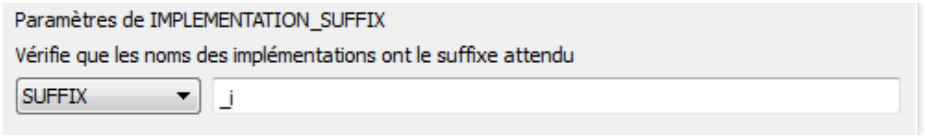


L'IHM propose également de choisir la façon dont seront présentés les résultats de la vérification, soit dans la vue « Erreurs » soit dans un fichier au format CSV, en cochant l'une ou l'autre des deux cases de la fenêtre.

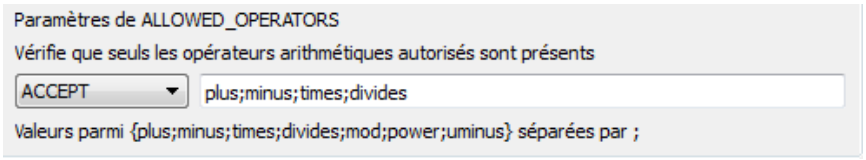
Dans ce cadre il existe un paramètre commun à toutes les règles, appelé « REPORT_NAME ». Il permet de paramétrer, lorsque l'utilisateur choisit de consigner les erreurs dans un fichier, quel est le nom sous lequel la règle apparaîtra.

Ci-dessous est présenté un exemple de paramétrage simple d'une règle. Ici il s'agit du suffixe que doivent avoir les noms des composants de type implémentation.

[1] Avec le support d'Alstom



L'exemple suivant montre lui un paramètre à valeur multiple, ici la liste des opérateurs autorisés dans le code des implémentations.



Le tableau ci-dessous décrit les différentes règles proposées par l'outil, avec leur description ainsi que leur identification au sein de l'interface.

Règles proposées

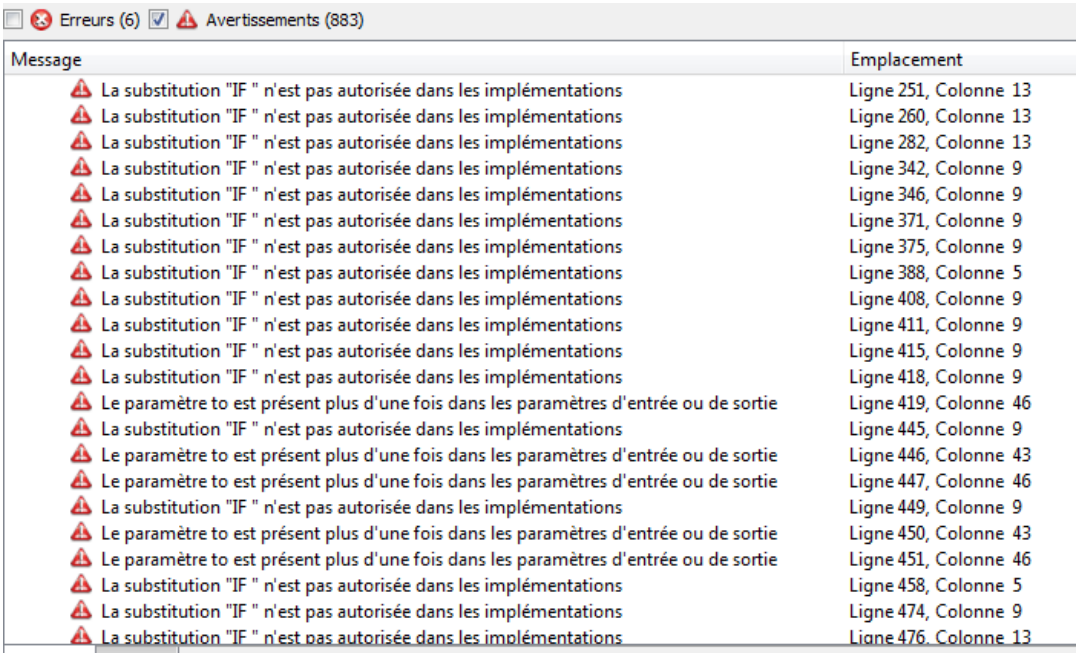
Règle	Description
Suffixe des raffinements (Règle REFINEMENT_SUFFIX)	Cette règle a un paramètre textuel nommé « SUFFIX » dont la valeur par défaut est « _r ». Elle vérifie que le nom des composants de type raffinement (débutant par le mot-clé REFINEMENT) se termine avec le suffixe saisi par l'utilisateur.
Suffixe des implémentations (Règle IMPLEMENTATION_SUFFIX)	Cette règle a un paramètre textuel nommé « SUFFIX » dont la valeur par défaut est « _i ». Elle vérifie que le nom des composants de type implémentation (débutant par le mot-clé IMPLEMENTATION) se termine avec le suffixe saisi par l'utilisateur.
Préfixe des types (Règle TYPE_PREFIX)	Cette règle a un paramètre textuel nommé « PREFIX » dont la valeur par défaut est « T_ ». Elle vérifie que les identificateurs des éléments déclarés dans les clauses SETS des composants commencent avec le préfixe entré par l'utilisateur.
Préfixe des constantes scalaires (Règle SCALAR_PREFIX)	Cette règle a un paramètre textuel nommé « PREFIX » dont la valeur par défaut est « _c ». Elle vérifie que les identificateurs des constantes déclarées dans les clauses ABSTRACT_CONSTANTS ou CONCRETE_CONSTANTS

	dont le type est inclus dans INTEGER se terminent avec le suffixe saisi par l'utilisateur.
Préfixe des valeurs énumérées (Règle ENUMERATED_PREFIX)	Cette règle a un paramètre textuel nommé « PREFIX » dont la valeur par défaut est « e_ ». Elle vérifie que les identificateurs des valeurs énumérées déclarées dans la clause SETS commencent avec le préfixe entré par l'utilisateur.
Opérateurs arithmétiques autorisés (Règle ALLOWED_OPERATORS)	Cette règle a un paramètre textuel à valeur multiple appelé « ACCEPT ». Ce paramètre prend comme valeur une liste d'opérateurs arithmétiques parmi : • 'plus' • 'minus' : soustraction • 'times' : multiplication • 'divides' : division entière • 'mod' : modulo • 'power' : puissance • 'uminus' : Moins unaire La règle vérifie que seuls les opérateurs contenus dans la liste décrite par l'utilisateur sont présents dans le code des implémentations. Par défaut le paramètre ACCEPT a la valeur « plus ;minus ;times ;divides »
Substitutions autorisées (Règle ALLOWED_SUBSTITUTION)	Cette règle a un paramètre textuel à valeur multiple appelé « FORBIDDEN ». Ce paramètre prend comme valeur une liste de type de substitutions parmi : • 'begin' • 'skip' • 'becomes_equal' • 'becomes_such_that' • 'assert' • 'if' • 'case' • 'var' • 'while' La règle vérifie que les substitutions listées par l'utilisateur n'apparaissent pas dans le code des opérations des implémentations (contenu de la clause OPERATIONS). Par défaut le paramètre FORBIDDEN est vide.
Paramètres en doublon (Règle INPUT_OUTPUT_PARAMS)	Un paramètre ne doit pas être présent deux fois dans un appel d'opération.
Typage des variables locales (Règle LOCAL_TYPING)	Les variables déclarées dans une substitution VAR IN doivent être typées dès le début dans un « devient tel que ».

Utilisation des résultats

Il est possible d'utiliser cette fonctionnalité en affichant les erreurs dans la vue principale de l'AtelierB, afin de pouvoir atteindre directement les emplacements concernés dans le modèle et procéder à des corrections. Cela se fait en cochant la case adéquate dans la fenêtre de configuration. Les violations des règles de codage à vérifier sont présentées dans cette vue avec la criticité « Avertissement »

L'exemple suivant présente les résultats d'une vérification dans laquelle a été intégrée la règle sur les paramètres en doublon et celle sur les substitutions, paramétrée pour interdire la substitution IF (valeur « if » pour le paramètre FORBIDDEN).



L'utilisateur peut également choisir de consigner les différentes violations des règles de codage dans un fichier de sortie au format CSV, là encore en cochant la case adéquate dans l'interface. Dans ce cas, seules les véritables erreurs (syntaxe, type) levées par le B compiler lors de l'analyse syntaxique et sémantique des composants à vérifier seront présentes dans la vue principale.

Extensibilité

De nouvelles règles seront intégrées ultérieurement à l'outil, en fonction des besoins constatés par les utilisateurs.

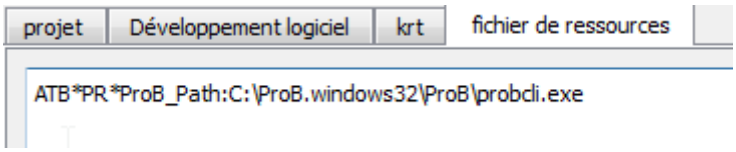
Intégration du model-checker ProB dans le prouveur interactif

Cette version permet à l'utilisateur de lancer le model-checker ProB dans le prouveur en tant que commande interactive utilisable au sein d'une démonstration.

Pour que cette commande puisse être utilisée, ProB doit être installé sur la machine, et la ressource ATB*PR*ProB_Path dans le fichier de ressources AtelierB doit pointer sur l'exécutable procli de l'installation.

Dans le cas contraire l'utilisateur verra apparaître un message indiquant que la ressource n'est pas positionnée quand il essaie d'utiliser la commande.

L'exemple ci-dessous montre le positionnement de la ressource pour l'utilisation de ProB dans un projet créé sous la version Windows de l'AtelierB.



Le nom de la nouvelle commande de preuve interactive est **`prob`**, elle accepte deux syntaxes différentes.

Commande	Description
`prob(n)`	Lance ProB sur le but courant. Le paramètre n est similaire à celui de pp(rp.n), ici la machine passée en paramètre d'entrée à ProB est construite en utilisant les hypothèses fournies par le rp.n.
`prob(n|t)`	Similaire à prob(n) mais limite en plus le temps d'exécution à t secondes.

L'utilisation de cette commande génère une machine qui contient le but à prouver en tant qu'assertion. S'il y a des hypothèses H provenant du **`rp.n`** quand **`prob(n)`** est utilisé pour prouver le but G, elles sont également écrites dans la clause assertions et celle-ci contient alors H => G. les prédicats de typage nécessaires sont ajoutés dans la clause PROPERTIES de cette machine.

ProB est appelé sur cette machine temporaire, dans le mode qui recherche des contre-exemples pour le contenu de la clause ASSERTIONS. 3 cas peuvent alors se produire.

- ProB réussit à tester l'ensemble exhaustif des valeurs pour les variables contenues dans la formule H => G et aucun contre-exemple n'est trouvé : **la branche de preuve courante est prouvée.**
- ProB trouve un contre-exemple pour la clause ASSERTIONS de la machine temporaire : **la commande ne prouve pas la branche, une notification apparaît dans la vue Messages de l'interface graphique**
- ProB ne peut pas parcourir l'ensemble exhaustif des valeurs pour les variables de la clause ASSERTIONS : **la commande échoue à prouver la branche courante**