Atelier B

# MDELTA

## User Manual

version 2.0

CLEARSY
SYSTEM ENGINEERING

ATELIER B
MDELTA User Manual
version 2.0

CLEARSY
Maintenance ATELIER B
Parc de la Duranne
320 avenue Archimède
Les Pléiades III - Bât. A
13857 Aix-en-Provence Cedex 3
France

Tel 33 (0)4 42 37 12 70
Fax 33 (0)4 42 37 12 71

mail : contact@atelierb.eu

# Contents

# Chapter 1

# Introduction

A mathematical predicate can be :

- true
- or false

provided that it is **well-defined**.

The ill-definedness of a predicate[1](or of an expression contained in this predicate) calls into question the correctness of a proof involving this predicate.

Atelier B V3.7 does not have any means of its own to detect these meaningless expressions[2].

MLE do not allow to guarantee the full validity of a proof work performed with Atelier B V3.7.

This issue is addressed by the production of well-definedness lemmas by the mdeltatool and by their automatic or manual proof.

This document defines what are meaningless (or potentially meaningless) expressions. It lays down how you can use the mdelta tool to generate the well-definedness lemmas that you will have to prove in order to ensure the correctness of your B development[3], as far as well-definedness is concerned.

[1]The verification that a predicate or expression is well-typed is performed by the type-checker. This verification is independent of the well-definedness verification. Well-definedness is only checked for well-typed predicates or expressions.

[2]MLE in abbreviated form (see § Terminology)

[3]The B method is described in [1].

# Chapter 2

# Terminology

## 2.1 Abbreviations

**PO:** proof obligation.

**POG:** proof obligation generator.

**PRB:** prover rule base.

**PDB:** project database directory.

**PatchProver:** File containing user-provided rules that can be used, for a whole project, in addition to the PRB rules.

**Pmm file:** File ending in '.pmm' and containing user-provided rules that can be used, for a component, in addition to the PRB rules.

**Pmi file:** File ending in '.pmi' and containing, in particular, the interactive commands entered by the user, for every PO of a component.

**PO file:** File ending in '.po' and containing the PO of one component.

**Component:** this term applies equally to an abstract machine, a refinement or an implementation in the B sense.

**MLE:** Meaningless Expression

**PMLE:** Potentially Meaningless Expression

## 2.2 Definition of terms used

Logic Solver language: The language used to implement some of Atelier B tools and, in particular, proof rules.

Proof obligation: Logical predicate generated by Atelier B from a component written in the B language[1], that has to be proved in order to guarantee the component correctness.

[1]The B language is described in [2].

Meaningless Expression: B language expression that cannot be given any mathematical interpretation, like a partial function applied to a value outside of its domain or the minimum of an empty set.

Potentially meaningless expression: B language expression requiring extra conditions to be meaningful (these conditions are listed in table 3.1, page 8).

Well-definedness lemma: logical predicate that has to be proved to ensure that a predicate is well-defined.

# Chapter 3

# Problem presentation

## 3.1 Example

Let us consider the predicate

$$y = \frac{x}{\frac{x+8}{c}} \tag{3.1}$$

This predicate can be true or false, provided that it is well-defined. If this predicate is not well-defined, it is then impossible to associate it a true or false value. This ill-definedness means that at least one operator of the predicate has at least one operand that does not belong to its domain.

Predicate 3.1 is obviously arithmetical by nature. We consider that this predicate has been type-checked (operation performed by the type checker) and that y, x, and c are integers.

The operators appearing in predicate 3.1 are:

- equality
  An equality a=b is well-defined on the condition that a and b are well-defined
- addition
  An addition a+b is well-defined on the condition that a and b are well-defined
- integer division
  An integer division a/b is well-defined on the condition that a and b are well-defined and b is non-zero.

We will then have to check that:

- x and c are well-defined (and it is the case since we have supposed that y, x and c are integers)
- the denominator of $x/(x + 8/c)$ is non-zero
- the denominator of $8/c$ is non-zero

To establish that predicate 3.1 is well-defined, we then have to prove the following lemmas

$$c \neq 0 \tag{3.2}$$

$$(x+8)/c \neq 0 \tag{3.3}$$

The predicate context has to contain these lemmas in the form of hypotheses or enable to deduce them.

If it is not the case, predicate 3.1 is potentially ill-defined.

## 3.2 Well-definedness conditions

The well-definedness conditions[1] are listed in the table below.

| Expression | Well-definedness condition |
|---|---|
| $a^b$ | $a \in \mathbb{N} \land b \in \mathbb{N}$ |
| $a \textsf{ mod } b$ | $b \in \mathbb{N}_1 \land a \in \mathbb{N}$ |
| $a/b$ | $b \in \mathbb{Z}_1$ |
| $\Pi(x).(P\|E)$ | $\{x\|P\} \in \mathbb{F}(\{x\|P\})$ |
| $\Sigma(x).(P\|E)$ | $\{x\|P\} \in \mathbb{F}(\{x\|P\})$ |
| $\textsf{max}(S)$ | $S \cap \mathbb{N} \in \mathbb{F}(\mathbb{N}) \land S \neq \emptyset$ |
| $\textsf{min}(S)$ | $S \cap (\mathbb{Z} - \mathbb{N}) \in \mathbb{F}(\mathbb{Z}) \land S \neq \emptyset$ |
| $\textsf{card}(S)$ | $S \in \mathbb{F}(S)$ |
| $\textsf{inter}(U)$ | $U \neq \emptyset$ |
| $\bigcap(x).(P\|E)$ | $\{x\|P\} \neq \emptyset$ |
| $r^n$ | $n \in \mathbb{N}$ |
| $f(x)$ | $x \in \textsf{dom}(f) \land f \in \textsf{dom}(f) \;\;\twoheadrightarrow\;\; \textsf{ran}(f)$ |
| $\textsf{perm}(S)$ | $S \in \mathbb{F}(S)$ |
| $\textsf{conc}(s)$ | $s \in \textsf{seq}(\textsf{ran}(s)) \land \forall x.(x \in \textsf{dom}(s) \;\Rightarrow\; s(x) \in \textsf{seq}(\textsf{ran}(s(x))))$ |
| $s \frown t$ | $s \in \textsf{seq}(\textsf{ran}(s)) \land t \in \textsf{seq}(\textsf{ran}(t))$ |
| $\textsf{size}(s)$ | $s \in \textsf{seq}(\textsf{ran}(s))$ |
| $\textsf{rev}(s)$ | $s \in \textsf{seq}(\textsf{ran}(s))$ |
| $s \leftarrow e$ | $s \in \textsf{seq}(\textsf{ran}(s))$ |
| $e \rightarrow s$ | $s \in \textsf{seq}(\textsf{ran}(s))$ |
| $\textsf{tail}(s)$ | $\textsf{size}(s) \geq 1 \land s \in \textsf{seq}(\textsf{ran}(s))$ |
| $\textsf{first}(s)$ | $\textsf{size}(s) \geq 1 \land s \in \textsf{seq}(\textsf{ran}(s))$ |
| $\textsf{front}(s)$ | $\textsf{size}(s) \geq 1 \land s \in \textsf{seq}(\textsf{ran}(s))$ |
| $\textsf{last}(s)$ | $\textsf{size}(s) \geq 1 \land s \in \textsf{seq}(\textsf{ran}(s))$ |
| $s \uparrow n$ | $n \in 0\,..\,\textsf{size}(s) \land s \in \textsf{seq}(\textsf{ran}(s))$ |
| $s \downarrow n$ | $n \in 0\,..\,\textsf{size}(s) \land s \in \textsf{seq}(\textsf{ran}(s))$ |

Figure 3.1: Potentially meaningless expressions

## 3.3 Example of well-definedness lemma generation

Let us consider the $x/((x+8)/c)$ expression.

This expression is well-defined if

[1]This conditions come from [3] and [4].

- $x$ is well-defined (true since it's a variable)
- $(x+8)/c$ is well-defined if
  - $x+8$ is well-defined if
    - $x$ is well-defined (true since it's a variable)
    - 8 is well-defined (true since it's a literal value)
  - $c$ is well-defined (true since it's a variable)
  - $c$ is non-zero (see table3.1, third line)
- $(x+8)/c$ is non-zero (see the table3.1, third line)

The well-definedness lemmas of expression $x/(x+8/c)$ are therefore

$$\begin{array}{l} H \vdash x+8 \neq 0 \\ H \vdash c \neq 0 \end{array}$$

where $H$ refers to the current hypothesis context.

## 3.4 MLE location

MLE can be found, for a B project, in:

- B components,
- project-level user-provided rules,
- component-level user-provided rules,
- interactive proof commands.

## 3.5 The mdelta tool for checking well-definedness

The mdelta tool does not exactly detect MLE. When applied to a project, it generates the well-definedness lemmas for all the project PMLE and attempts to prove them automatically. The result is available in a "child" B project that is automatically created or updated.

Files affected by the PMLE search are:

- B component files (in normal form, files ending in '.nf')
- Patchprover file,
- Pmm files,
- Pmi and Po files (Po files are necessary to get context information.)

The well-definedness lemmas are gathered and their origin is identifiable (PMLE can be tracked down).

# Chapter 4

# Using mdelta

## 4.1 When you should use it

This tool can be applied on any B project, anytime during its development, without the proof being finished. Nonetheless, all the components must have been type-checked successfully.

We strongly advise you to use mdelta on two particular occasions in the B project development life:

- When the project has been informally validated (all PO seem correct but no formal proof has been performed)
- and when the proof work has formally validated the project (some PMLE may have been introduced during the interactive proof process through the use of commands like `ah`, `ph` or `se` or user-provided rules)

## 4.2 Step I: Running mdelta

To launch the tool, use the following command:

```
mdelta <DELTA_CHOICE> <ATB_HOME> <ATB_RESSOURCE> <PROJECT_NAME>
```

with:

- DELTA_CHOICE: processing options:
  - -r: to apply the tool to user-provided rules (Pmm and PatchProver files),
  - -p: to apply the tool to interactive proof commands ('.pmi' and '.po' files),
  - -b: to apply the tool to B components ('.nf' files),
  - any combination of the three options above,
  - -a: to apply the tool with all the options enabled (equivalent to -r -p -b).
- ATB_HOME: installation directory of Atelier B
- ATB_RESSOURCE: complete path of the Atelier B global resource file
- PROJECT_NAME: project declared in Atelier B to which we want to apply the tool

### 4.2.1 Example

*mdelta -a /home/atelierb/AB /home/atelierb/AB/AtelierB DeltaTest*

### 4.2.2 Output

The whole output of this step is written in the PDB directory of the PROJECT_NAME project in the form of the following files:

- For the PMLE of a B component named 'X' (option *-a* or *-b* has been selected):
  - X_wd.po: file containing the well-definedness lemmas
  - X_wd.pmi: file containing the interactive proof work to be done
- For the PMLE of the interactive proof commands of a B component named 'X' (option *-a* or *-p* has been selected):
  - X_pmi.po
  - X_pmi.pmi
- For the PMLE of the PatchProver file (option *-a* or *-r* has been selected):
  - PatchProver.lemmas
  - PatchProver.unresolved: contains expressions that could not be analysed because of a too imprecise syntax
  - PatchProver.res: contains the lemmas of PatchProver.lemmas and their status after the application of the predicate prover (proved or unproved).
- For the PMLE of each Pmm file (option *-a* or *-r* has been selected):
  - X.lemmas
  - X.unresolved
  - X.res
- A script file called *createDelta.sh* (see Step III) when the *-a* option has been selected.

## 4.3 Step II: Verifying the application of mdelta

You have to verify that Step I went smoothly by examining the displayed messages. If some of the messages indicate a problem, you have to:

- Analyse the origin of the problem and correct it
- start again step I

## 4.4 Step III: Creating the *delta_PROJECT* B project

### 4.4.1 Manual creation

To use Atelier B on the well-definedness lemmas generated during the previous step, you have to create a "fake" B project. The proof obligations of this project will then correspond to the well-definedness lemmas.

To obtain this project, it is sufficient to create a project with its *pdb*, *src* and *lang* directories, add to it the **empty**[1] machines X_wd.mch and X_pmi.mch (corresponding to the previously generated X_wd.pmi, X_wd.po, X_pmi.pmi and X_pmi.po files), type-check them and apply the proof obligations generator (it will not generate any PO as the machines are empty) to every one of them.

Once done, you have to replace the '.pmi' and '.po' files generated by the proof obligations generator by those created by mdelta.

The well-definedness lemmas are now available for an interactive proof work.

### 4.4.2 Automatic creation

The *createDelta.sh* script, generated during step I with the *-a* option, allows to create a *delta_PROJECT* B project so as to:

- display from Atelier B the well-definedness lemmas
- perform interactive proof work on this lemmas

You just have to type in

```
sh ./createDelta.sh
```

### 4.4.3 Script Output

- If a *delta_PROJECT* project already exists:
  - important data ('.po', '.pmi', '.lemmas', '.res' and '.unresolved' files) from the *delta_PROJECT* PDB are backed up in the same place, (files are renamed with the backup date appended)
  - the project is deleted
- the *delta_PROJECT* project is created again with the "fake" but necessary (for the proof work of the well-definedness lemmas) components.
- Files created during step I are moved to the *delta_PROJECT* project PDB directory.

## 4.5 Step IV: Proving the well-definedness lemmas

You can now begin to check informally and/or prove formally the well-definedness lemmas:

[1]A component X is considered to be empty if it is reduced to "MACHINE X END"

- for the lemma related to components or interactive proof commands, just use Atelier B as usual after opening the *delta_PROJECT*B project.
- As far as the lemmas related to user-provided rules (included in the files ending in 'lemmas') are concerned, theses lemmas cannot be handled through Atelier B V3.7. The files ending in '.res' contain the output of the application of the predicate prover to these lemmas (performed during step I), that is to say their proof status (Proved or Unproved). The unproved lemmas must then be proved manually by the user who will have to provide a formal proof document.
- The rules from the '.unresolved' files must be proved manually too. But before the manual proof, two operations must be performed. First, the user must translate these rules into mathematical lemmas[2] likely to be proved. Once translated, the user must deduce from them the corresponding well-definedness lemmas (using table 3.1). The formal proof of these lemmas makes up the last step of the '.unresolved' files handling.

[2] Use [5]

# Bibliography

[1] B-Book, Jean-Raymond ABRIAL (1996).

[2] B Language - Reference Manual, version 1.8.5, ClearSy.

[3] Partial logics reconsidered: a conservative approach, Formal aspect of computing, Olaf OWE (1993).

[4] Traitement des expressions dépourvues de sens de la théorie des ensembles. Application à la méthode B (Thèse de Doctorat), Lilian BURDY.

[5] Rule writing guide, version 1.0, ClearSy.