

Context instantiation plug-in: a new approach to genericity in Rodin^{*}

Guillaume Verdier¹, Laurent Voisin²

¹ LMF/CentraleSupélec, Université Paris-Saclay
² Systerel

June 8th, 2021
9th Rodin User and Developer Workshop

^{*}This work is supported by the French ANR project Event-B Rodin Plus (EBRP, ANR-19-CE25-0010).

Introduction

- ▶ the core Rodin platform has many features, but lacks genericity/type parametricity
- ▶ how to define, prove and use generic mathematical theories or data structures?
 - ▶ implement specific instances (e.g., List_Int) with copy-and-paste... which is cumbersome and error-prone
 - ▶ use the Theory plug-in, which introduces a new type of files, a slightly different syntax, a deployment process, ...
- ▶ as part of the EBRP project, Jean-Raymond Abrial suggested a new, lighter approach: the *context instantiation plug-in* experiments it

Overview

Main idea: start from what people do by hand (“copy-and-paste method”) and provide a plug-in to automate it in a safe way.

- ▶ reuse contexts
- ▶ use carrier sets and constants as generic parameters
 - ▶ carrier sets: type parameters
 - ▶ constants: value/expression parameters
- ▶ prove theorems about these abstract types and constants
- ▶ in other contexts, *instantiate* these generic theorems by replacing carrier sets and constants with concrete types and values

Generic contexts

“Generic” contexts are written as usual and do not need the plug-in. For example:

context sequence

sets S

constants

seq

add

size

axioms

axm1: $seq = \{f \mapsto n \mid n \in \mathbb{N} \wedge f \in 1..n \rightarrow S\}$

axm2: $add = (\lambda x \cdot x \in S \mid (\lambda f \mapsto n \cdot f \mapsto n \in seq \mid f \cup \{n+1 \mapsto x\} \mapsto n+1))$

theorem thm1: $add \in S \rightarrow (seq \rightarrow seq)$

axm3: $size = (\lambda f \mapsto n \cdot f \mapsto n \in seq \mid n)$

theorem thm2: $size \in seq \rightarrow \mathbb{N}$

theorem thm3: $\forall x, f, n \cdot x \in S \wedge (f \mapsto n \in seq) \Rightarrow size(add(x)(f \mapsto n)) = size(f \mapsto n) + 1$

Instantiations

In the most basic setting, users can manually instantiate theorems and describe the instantiation in the comment box: the plug-in checks that the instantiation is valid.

Useful to check existing contexts:

```
context seqInt
constants
  seq
  add
  size
axioms
  @axm1: seq = {f ↦ n | n ∈ ℕ ∧ f ∈ 1 .. n → ℤ} // sequence|S:=ℤ|axm1
  @axm2: add ∈ ℤ → (seq → seq) // sequence|S:=ℤ|thm1
  @axm3: size ∈ seq → ℕ // sequence|S:=ℤ|thm2
  @axm4: ∀x, f, n · x ∈ ℤ ∧ f ↦ n ∈ seq ⇒ size(add(x)(f ↦ n)) = size(f ↦ n) + 1 // sequence|S:=ℤ|thm3
```

Syntax:

- ▶ @ before axiom label
- ▶ *context* | *substitutions* | *theorem* in comment
 - ▶ substitutions: $param_1 := value_1$;; $param_2 := value_2$;; ...

Short demo

Let's look at the previous example in Rodin:

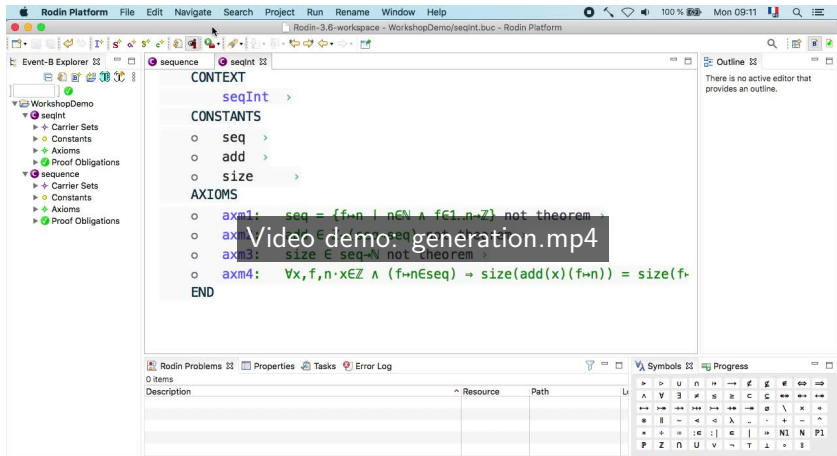
The screenshot shows the Rodin Platform IDE with a BNF specification for a sequence data type. The interface includes a menu bar (File, Edit, Navigate, Search, Project, Run, Rename, Window, Help), a toolbar, and a sidebar with an Event-B Explorer. The main editor displays the following code:

```
CONTEXT
  seqInt >
CONSTANTS
  o seq >
  o add >
  o size >
AXIOMS
  o axm1: seq = {f:n | n ∈ N ∧ f ∈ 1..n · Z} not theorem >
  o axm2: add ∈ seq · N not theorem >
  o axm3: size ∈ seq · N not theorem >
  o axm4: ∀ x, f, n · x ∈ Z ∧ (f · n ∈ seq) ⇒ size(add(x)(f · n)) = size(f · n)
END
```

A video watermark "Video demo: instantiation.mp4" is overlaid on the code. The bottom of the IDE shows a table with columns for Description, Resource, and Path, and a symbols palette.

Generation

For new developments, one can provide only the instantiation in the comment box: the plug-in can generate the instantiated axiom:

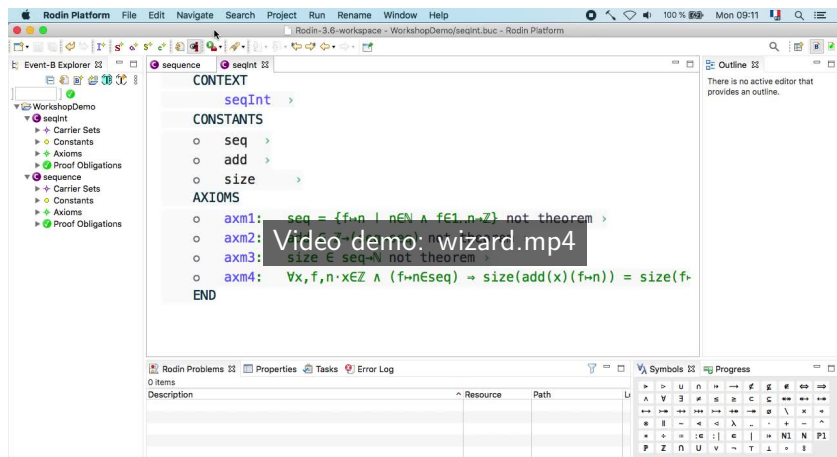


The screenshot shows the Rodin Platform IDE interface. The main editor displays a BNF grammar for a sequence type. The grammar is structured as follows:

```
CONTEXT
  seqInt >
CONSTANTS
  seq >
  add >
  size >
AXIOMS
  axm1: seq = {f·n | n∈N ∧ f∈{1..Z}} not theorem >
  axm2: seq ∈ Seq not theorem >
  axm3: size ∈ Seq → not theorem >
  axm4: ∀x, f, n·x∈Z ∧ (f·n∈seq) ⇒ size(add(x)(f·n)) = size(f·n)
END
```

A semi-transparent watermark "Video demo: generation.mp4" is overlaid on the code. The IDE includes a menu bar (File, Edit, Navigate, Search, Project, Run, Rename, Window, Help), a toolbar, an Event-B Explorer on the left, and a bottom panel with Rodin Problems, Properties, Tasks, Error Log, Symbols, and Progress.

It is also possible to create instantiations through a wizard:



The screenshot shows the Rodin Platform IDE interface. The main editor displays a file named 'seqint.buc' with the following structure:

```
CONTEXT
  seqInt >
CONSTANTS
  seq >
  add >
  size >
AXIOMS
  axm1: seq = {f:n | n ∈ N ∧ f ∈ 1..n} not theorem >
  axm2:
  axm3: size ∈ seq-N not theorem >
  axm4: ∀x,f,n·x ∈ Z ∧ (f → n ∈ seq) → size(add(x)(f → n)) = size(f → n)
END
```

A semi-transparent black box with white text 'Video demo: wizard.mp4' is overlaid on the editor content. The left sidebar shows the 'Event-B Explorer' with a tree view containing 'WorkshopDemo', 'seqInt', 'Carrier Sets', 'Constants', 'Axioms', 'Proof Obligations', 'sequence', 'Carrier Sets', 'Constants', 'Axioms', and 'Proof Obligations'. The bottom status bar includes 'Rodin Problems', 'Properties', 'Tasks', 'Error Log', 'Symbols', and 'Progress'. The 'Progress' panel shows a table with columns 'Description', 'Resource', and 'Path', and a row of navigation icons.

Types and sets

In the sequence example, S is a carrier set, used for *type* substitutions:

- ▶ $S := \mathbb{Z}$ or $S := \mathbb{Z} \times \mathbb{Z}$ are fine, for example
- ▶ $S := \mathbb{N}$ or $S := \mathbb{Z} \leftrightarrow \mathbb{Z}$ are not: they are *sets*, not *types*

We can adapt the generic context to use constant substitution...

context sequence

sets S_type

constants

S

...

axioms

axm1: $S \subseteq S_type$

...

... and then instantiate with $S_type := \mathbb{Z}$;; $S := \mathbb{N}$, for example.

But the plug-in has type inference, so $S := \mathbb{N}$ is sufficient: authors of generic contexts have to take care of it; users do not see it.

Conclusion

The context instantiation plug-in introduces a new approach to genericity in Rodin that

- ▶ is tightly integrated in the core Rodin platform
- ▶ should be very simple to set-up and use
- ▶ can also check existing projects with duplicated code

It is currently in use by members of the EBRP project (see next talks). It should be publicly available soon.

Questions?