



HAL
open science

Data-types definitions: Use of Theory and Context instantiations Plugins

Peter Riviere, Neeraj Kumar Singh, Yamine Aït-Ameur

► To cite this version:

Peter Riviere, Neeraj Kumar Singh, Yamine Aït-Ameur. Data-types definitions: Use of Theory and Context instantiations Plugins. 9th Rodin User and Developer Workshop collocated with the ABZ 2021 Conference, Jun 2021, Ulm (virtual), Germany. pp.1-6. hal-03487183

HAL Id: hal-03487183

<https://hal.science/hal-03487183>

Submitted on 17 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Data-types definitions: Use of Theory and Context instantiations Plugins

Peter Riviere, Yamine Ait-Ameur, and Neeraj Kumar Singh

IRIT/INPT-ENSEEIH

2 rue Charles Camichel 31071 Toulouse cedex 7. France
{peter.riviere,yamine,neeraj.singh}@toulouse-inp.fr

1 Introduction

In the context of the French national research agency (ANR) EBRP-EventB-Rodin-Plus [4]¹ (Enhancing Event-B and Rodin) project, an extension of the Rodin platform [2] supporting the design of Event-B [1] models has been designed in the form of a plugin [6], namely the *context Instantiation plugin*. It allows the definition of generic contexts and their instantiation to define generic and reusable theories. Instantiable Sets and Constants with their axioms and theorems are defined in a context has been designed. A mechanism for instantiating such generic contexts by importing useful axioms and theorems in another context. A language for describing such instantiations has been defined. It is parsed in order to generate instantiated contexts.

In the work of [3,5], a mathematical extension of Event-B allowing the definition of theories was proposed and implemented in the so-called *Theory Plugin*.

In this paper, we investigate the correspondence between theories formalised in the theory plugin and those formalised in the context instantiation plugin. We present transformation rules that allow us to describe theories through contexts and their instantiation. The goal of these transformations is to provide an additional way to model theories in the core Event-B modelling language.

These correspondences for possible data-type definitions are described further below.

2 Direct definitions of data-types

The correspondence between data-types (non-inductive) and operators defined as *direct definitions* in the Theory Plugin is presented in this section.

2.1 Data-type transformation

Figures 1a and 1b show the correspondence between a data-type defined in a theory with type parameters and constructors (parameterised or not) and a context. Only two type parameters and two constructors have been defined for the sake of clarity in the presentation.

¹ <https://www.irit.fr/EBRP/>

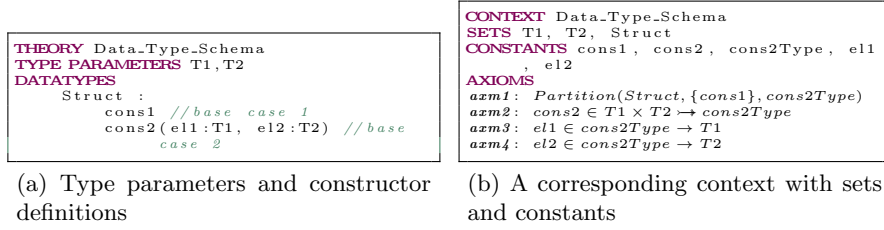


Fig. 1: Data-type correspondence

2.2 Direct definitions of Operators: expressions

As shown on Figures 2a and 2b, theory based direct definitions of operators correspond to partial functions (defined using a lambda expression) where typing and the Well-definedness conditions are used to define the domain of these functions.

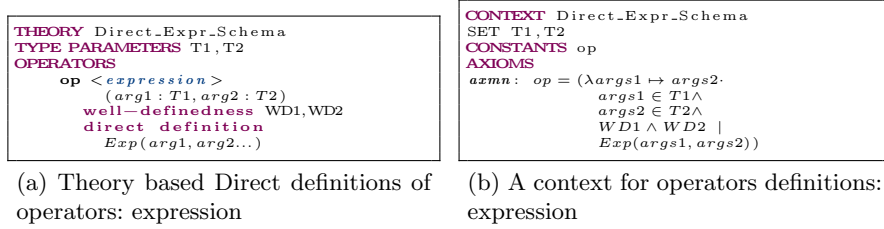


Fig. 2: Direct definitions of operators: expression

2.3 Direct definitions of Operators: predicates

The same principle applies to operators defining predicates as it does to operators defining expressions. The correspondence for operators defined as predicates is shown in Figures 3a and 3b.

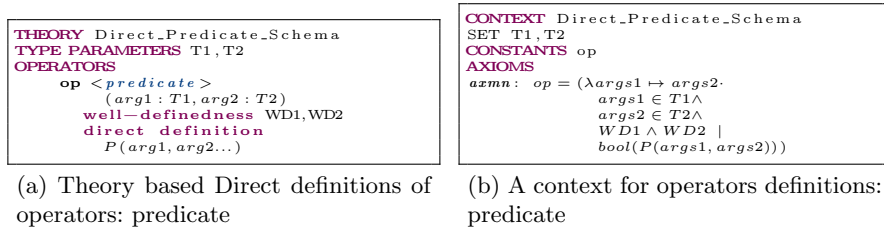


Fig. 3: Direct definitions of operators: predicates

3 Axiomatic data-types definitions

Axiomatic definitions on Figures 4a and 4b correspond to direct definitions in Section 2, except that the expression or predicate is not given in the function definition. In axiom *axmDefOp*, an axiomatically defined operator is formalised as a total function on the domain restricted by the well-definedness conditions and a resulting type (*Res.Type*). Then, in the theory, each axiom that characterises this operator is translated as an axiom in the context.

<pre> THEORY Axm_Schema TYPE PARAMETERS T1, T2 AXIOMS OPERATORS op <expression> (arg1 : T1, arg2 : T2) : res : Res.type well-definedness WD1, WD2... AXIOMS axm1 : Expl(op, ...) ... axmn : Expn(op, ...) </pre>	<pre> CONTEXT Axm_Schema SET T1, T2 CONSTANTS op AXIOMS axmDefOp : op ∈ {args1 ↦ args2. arg1 ∈ T1 ∧ arg2 ∈ T2 ∧ WD1 ∧ WD2} → Res.type axm1 : Expl(op, ...) ... axmn : Expn(op, ...) </pre>
--	--

(a) A theory based definition of axiomatic operators.

(b) A context based definition of axiomatic operators.

Fig. 4: Correspondence for axiomatic definitions of operators.

4 Inductive data-types definitions

In the Event-B modelling language, inductively defined data-types do not have their direct correspondence. This correspondence requires the introduction of a generic definition for inductive structures.

4.1 Generic context for inductive definitions by J.R. Abrial and D. Cansell

To define Theory based inductive data-types correspondence, we use the generic definitions introduced by J.R. Abrial and D. Cansell in the EBRP project.

<pre> CONTEXT SchemaRecGen SETS S_type, B_type CONSTANTS wellfounded, fix, FrSB, S, B AXIOMS axm0 : S ⊆ S_type axm6 : B ⊆ B_type @axm1 : wellfounded = {r · r ∈ S ↔ S ∧ (∀p · p ⊆ S ∧ p ⊆ r[p] ⇒ p = ∅)} @axm2 : fix ∈ (P(S × B) → P(S × B)) → P(S × B) @axm3 : ∀h · h ∈ P(S × B → P(S × B)) ∧ (b · a ⊆ b ∧ b ⊆ S × B ⇒ h(a) ⊆ h(b)) ⇒ fix(h) = h(fix(h)) axm4 : FrSB = {r ↦ g ↦ fr r ∈ wellfounded ∧ g ∈ S × (S → B) → B ∧ (∀x, f · x ∈ S ∧ f ∈ S → B ∧ r[{x}](f) ⇒ x ↦ f ∈ dom(g)) ∧ fr = fix(λp · p ∈ S ↔ B {x, h · x ∈ S ∧ r[{x}] ⊆ h ⊆ p ∧ r[{x}] ⊆ h ∈ r[{x}] → B x ↦ g(x ↦ r[{x}] ⊆ h))}} lem1 : FrSB ∈ {r ↦ g r ∈ wellfounded ∧ g ∈ S(S → B) → B ∧ (∀x, f · x ∈ S ∧ f ∈ S → B ∧ r[{x}] ⊆ dom(f) ⇒ x ↦ f ∈ dom(g))} → (S ↔ B) THEOREMS thm1 : ∀r, g, fr · r ↦ g ↦ fr ∈ FrSB ⇒ fr ∈ S → B thm2 : ∀r, g, fr · r ↦ g ↦ fr ∈ FrSB ⇒ (∀x · x ∈ S ⇒ fr(x) = g(x ↦ r[{x}] ⊆ fr)) </pre>
--

Fig. 5: A generic context with inductive sets definition operator *FrSb*.

The context `SchemaRecGen` of Figure 5 uses the definitions of well-founded relations and the fixpoint operator from contexts not shown in this paper. They are brought up for clarity.

The most important feature is the constant `FrSB` allowing to define the semantics of operators defined on inductive types. It use the type constructors and the fixpoint operator. This function is further applied to formalise the theory based defined inductive types and operators.

4.2 Correspondence schema

Inductive definitions are given in two steps: first the data-type definition using inductive sets definitions and second the operators manipulating this data-type.

Inductive data-type definition. A recursive definition is based on an inductive type, which is depicted in Figures 6a and 6b as a set comprehension in which the inductive properties are encoded and the constants are elements of this set. The `IndType` theory data-type corresponds to the set `IndType`, which is defined from the carrier set `IndTypeTYPE`. The `IndTypeSET` defines the set of n-uplets corresponding to the n constructors of the data-type. In our case, $cons1_El \mapsto cons2_El \mapsto consinduc1_El \mapsto consinduc2_El$.

```

THEORY Ind_Data_Type_Schema
TYPE PARAMETERS T
DATATYPES
IndType :
  cons1
  //base case 1
  cons2 (el : T)
  //base case 2
  consinduc1 (el : IndType)
  // inductive case 1
  consinduc2 (el1 : T, el2 IndType)
  // inductive case 2

```

(a) Inductive type definition

```

CONTEXT Ind_Data_Type_Schema
SETS IndTypeTYPE, T
CONSTANTS IndType, IndTypeSET, cons1, cons2,
  consinduc1, consinduc2
AXIOMS
azm1: IndTypeSET =
  { indtype_El  $\mapsto$  cons1_El  $\mapsto$  cons2_El  $\mapsto$ 
    consinduc1_El  $\mapsto$  consinduc2_El |
    indtype_El  $\subseteq$  IndTypeTYPE  $\wedge$ 
    cons1_El  $\in$  indtype_El  $\wedge$ 
    cons2_El  $\in$  T  $\mapsto$  (indtype_El \
      (ran(consinduc1_El)  $\cup$ 
        ran(consinduc2_El)  $\cup$  {cons1_El}))  $\wedge$ 
    consinduc1_El  $\in$  indtype_El  $\mapsto$  (indtype_El \
      (ran(cons2_El)  $\cup$  ran(consinduc2_El)  $\cup$ 
        {cons1_El}))  $\wedge$ 
    consinduc2_El  $\in$  T  $\mapsto$  (indtype_El \
      (ran(consinduc1_El)  $\cup$  ran(cons2_El)  $\cup$ 
        {cons1_El}))  $\wedge$ 
    ( $\forall tr \cdot cons1\_El \in tr \wedge cons2\_El[T] \subseteq tr \wedge$ 
      consinduc1_El[tr]  $\subseteq$  tr  $\wedge$ 
      consinduc2_2[T  $\times$  tr]  $\subseteq$  tr
       $\Rightarrow$  indtype_El  $\subseteq$  tr)
  }
azm2: IndType  $\mapsto$ 
  cons1  $\mapsto$  cons2  $\mapsto$ 
  consinduc1  $\mapsto$  consinduc2
   $\in$  IndTypeSET

```

(b) Corresponding context with sets and constants

Fig. 6: Correspondence for inductive type

Inductive data-type operator definition. The correspondence for an inductively defined operator is shown in Figures 7a and 7b. The *FrSb* operator is used for the defined inductive data-type *IndType* in both base (with definitions of expressions *exp1* and *exp2*) and inductive cases (with definitions of expressions *ExpInd1* and *ExpInd2*).

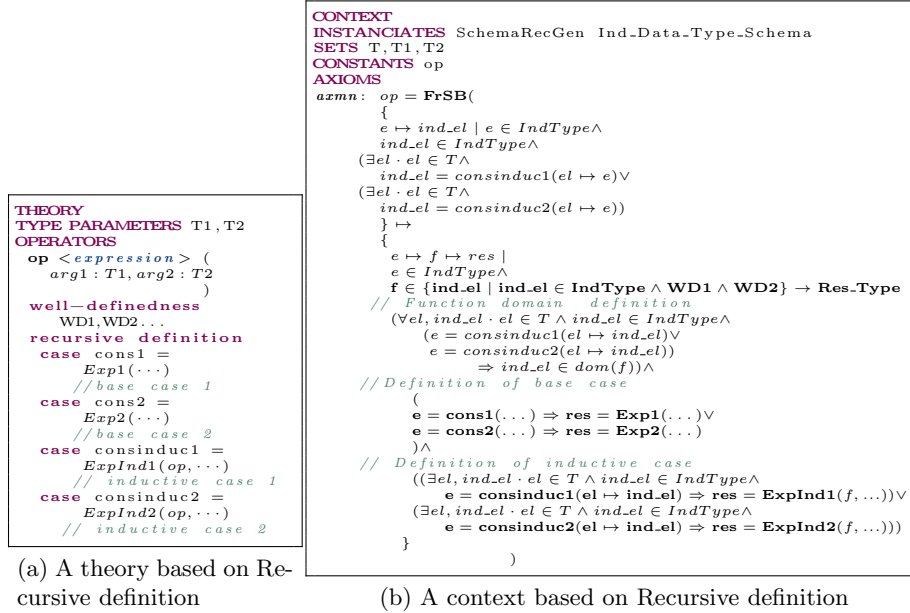


Fig. 7: Corresponding schema of recursive definition of operators

5 Conclusion

We provided a set of correspondences that allow theory-based data types to be translated as contexts. Except for the inductive definitions, which require the use of operators defining inductive sets borrowed from a generic context, this transformation is straightforward.

When we translate theories to context, we obtain context definitions expressed in the native Event-B modeling language, but we lose the structuring and semantic information available in the theories.

References

1. Abrial, J.R.: Modeling in Event-B: system and software engineering. Cambridge University Press (2010)

2. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An open toolset for modelling and reasoning in event-b. *Int. J. Softw. Tools Technol. Transf.* **12**(6), 447–466 (Nov 2010)
3. Butler, M., Maamria, I.: Mathematical extension in Event-B through the rodin theory component (2010)
4. Ebrp, <https://www.irit.fr/EBRP/>
5. Hoang, T.S., Voisin, L., Salehi, A., Butler, M., Wilkinson, T., Beauger, N.: Theory plug-in for rodin 3.x (2017)
6. Verdier, G., Laurent, V.: Context instantiation plug-in: a new approach to genericity in Rodin. Rodin Workshop at ABZ'2021 (2021)