Records

Vitaly Savicks, Colin Snook, Michael Butler

University of Southampton, Southampton SO17 1BJ, UK
{vs02, cfs, mjb}@ecs.soton.ac.uk

1 Introduction

The Records plug-in allows users to declare structured types within an Event-B context using the form-based editor UI. This is achieved by extending the Rodin database with new syntactical features to support the declaration of structured types called records. Once declared, a record can be used like any other set. Records can be sub-typed so that new fields are added to a declared subset of records. They may also be extended so that new fields are added to all such records. Records may be closed to prevent further extension or subtyping. This is necessary if update and make functions are required to assist in specifications that use the records. Declarations of records exist in the unchecked Event-B context – either abstract or concrete – and can be used as normal types e.g. carrier sets inside a machine or refinement, but are translated to the standard Event-B constructs in the checked version of the corresponding context. The translation is done during static checking performed by the Records plug-in's SC modules. The latter add extra axioms on records – feasibility, closure, make/update functions – to the checked version of the context. The axioms can be used later in the prover and referenced by semantically interpretable labels.

This feature has been developed as part of the Deploy project[1] in response to requests from our industrial partners.

2 Syntax of Records

The syntax of a structured type declaration introduced by Records plug-in is the following:

```
RECORD DECLARATIONS
R
SUPERTYPES
S
FIELDS
e type E
f type F
END
```

[1] DEPLOY: EU Project IP-214158, www.deploy-project.eu

This example creates a new structured type R as a subtype of an abstract type S (record or carrier set), thus inheriting its fields if present, as well as introduces new fields e and f of types E and F, specific to type R. The syntactic elements however exist only in the form-based editor. When parsed by extended static checker record declarations translate to Event-B constructs: record types translate to sets and fields – to projection functions:

```
SETS R
CONSTANTS e, f
AXIOMS
R ⊂ S
e ∈ R → E
f ∈ R → F
```

Later in refinements a declared record can be extended with new fields. The syntax of a record extension declaration is the following:

```
RECORD EXTENSIONS
R
FIELDS
g type G
END
```

In terms of Event-B an extension of a particular record simply adds a projection function for every introduced field. The example shown adds a new field g of type G to the previously declared record R, which translates to:

```
CONSTANTS g
AXIOMS
g ∈ R → G
```

3 UI Extensions

The Records plug-in extends the Rodin database with new record and field element types. The Event-B form-based context editor is extended with Records and Record Extensions declaration sections that include structural elements for record type, subtype, extension and field declarations. Records can subtype either previously declared records or carrier sets. To prevent a particular record from further extension or subtyping it can be closed with a toggle control. The pretty-print viewer and Event-B navigator are extended to support Records. An extension to the EMF framework for Event-B has been provided (to be released) so that EMF based tools can be extended to support Records. The EMF based structured editor Rose is automatically extended to support Records, based on the generated EMF item-providers.

4 Using Records in Modelling and Proving

Records are declared in the Record Declarations section of a Context. A user may declare a new record or create a subtype of an existing record or carrier set which is already defined in the abstract contexts or in the same context, but importantly before the current declaration. A super-type is referenced in the Supertypes property of a record declaration. In the Fields section of a record declaration, new fields are introduced. A field's type must be specified as an expression that gives a basic type. This is because it will not be possible to verify that after a record is declared it is either left open for further sub-typing or extension, or it is closed to finalising its structure. An extension to an existing record is declared in the Record Extensions section using an Extends clause. Each extension contributes new fields to an extended record type, with more than one extension possible for a single record. Once declared, a record becomes a structured type available to other typed elements in current and extended contexts and machines that use them. Function application is used to refer to the value of a record's field. After a context with record declarations is saved and static-checked (built) each record declaration and extension is translated to Event-B constructs in the checked context. A feasibility axiom of the following form is generated for each record type and its extension to ensure the existence of a record for every possible combination of its field values:

$$e \otimes f \in R \twoheadrightarrow (E \times F)$$

here for a record R with two fields e and f of types E and F. For a closed record an axiom is generated to indicate that there is a unique record for each combination of field values. This is required to ensure that the constructor function that follows is valid :

$$e \otimes f \in R \rightarrowtail (E \times F)$$

A constructor function that constructs a record from a tuple of field components is also generated. It is both injective and surjective to allow the generation of all possible records, different for each unique combination of components, and consists of the following axioms:

$$make\_R \in (E \times F) \rightarrowtail R$$

$$\forall x, y \cdot e(make\_R(x \mapsto y)) = x$$

$$\forall x, y \cdot f(make\_R(x \mapsto y)) = y$$

In addition a set of convenience update-functions for each field of a closed record is generated using the constructor function e.g. for the field e of example record R:

$$update\_R\_e \in R \times E \to R$$

$$\forall r, x \cdot update\_R\_e(r \mapsto x) = make\_R(x \mapsto f(r))$$

All the generated axioms have interpretable labels that consists of record name, field name and a name of the axiom if still ambiguous. Labels are used in the prover to refer to a particular axiom. Generated functions are used in modelling to construct records and update their fields.