Translation rules: Event-B to Eiffel

V. Rivera

November 21, 2016

1 The translation

EB2Eif implements the mapping δ : EventB $\rightarrow$ Eiffel is a transition step to translate Event-B constructs to Eiffel.
ξ translates Event-B Expressions or Predicates to Eiffel.
τ translates Event-B variable's type to Eiffel.

Translating EventB machines (Machine-rule)

Translation rule for Event-B machine M to Eiffel:

$$\frac{\tau(v) = \texttt{Type} \quad \xi(I(s,c,v)) = \texttt{Inv} \quad \delta(\textsf{events}\ e) = \texttt{E} \qquad \delta(\textsf{event}\ initialisation\ \textsf{then}\ A(s,c,v)\ \textsf{end}) = \texttt{Init}}{\begin{array}{l}\delta(\textsf{machine}\ M\ \textsf{sees}\ C \\ \qquad \textsf{variables}\ v \\ \qquad \textsf{invariants}\ label\_inv:\ I(s,c,v) \\ \qquad \textsf{event}\ initialisation\ \textsf{then}\ A(s,c,v)\ \textsf{end} \\ \qquad \textsf{events}\ e \\ \quad \textsf{end}) =\end{array}} \text{(Machine-rule)}$$

```
class M
create initialisation

feature -- Initialisation
   Init
feature -- Events
   E
feature -- Access
   ctx : CONSTANTS
        -- translation of Event-B constants

   v : Type
        -- variable comment
invariant
   label_inv: Inv
end
```

Translating EventB Contexts (Context-rule)

Translation rule for Event-B context C to Eiffel:

$$\dfrac{\begin{array}{l}\delta(\text{axioms } X(s,c)) = \texttt{X}\\ \tau(c) = \texttt{Type}\end{array}}{\begin{array}{l}\delta(\text{Context } \ C\\ \qquad \text{constant } c\\ \qquad \text{set } S\\ \qquad \text{Axioms } X(s,c)\\ \quad \text{end}) =\end{array}} \text{ (Context-rule)}$$

```
class CONSTANTS
feature -- Constants
   c : Type
         -- 'c' comment
      once
         create Type Result
      end
invariant
   X
end
```

Translating EventB carrier sets (CSet-rule)

Translation rule for Event-B carrier set S to Eiffel. In EventB, carrier sets represent a new type defined by the user. It is translated as an Eiffel class:

$$\frac{\tau(s) = \texttt{Type}}{\begin{array}{l}\delta(\textsf{Context}\ \ C\\ \qquad \textsf{constant}\ c\\ \qquad \textsf{set}\ S\\ \qquad \textsf{Axioms}\ X(s,c)\\ \quad \textsf{end}) =\end{array}}\ (\textsf{CSet-rule})$$

```
class S
inherit
    EBSET[Type]
        export
            {NONE}
                singleton, from_set, card, partition,
                pow, pow1, min, max
        redefine
            has, is_empty, default_create, finite, union,
            intersection, difference, is_equal, partition
        end
create
    default_create
feature -- Default Creation

    default_create
        do
            create elements
            is_user_type_def := True
        end

feature -- Redefinition
    has (v: Type): BOOLEAN
            -- FROM EBSET
        do
            Result := True
        end

    is_empty : BOOLEAN
            -- FROM EBSET
        do
            Result := False
        end

    finite : BOOLEAN
            -- FROM EBSET
        do
            Result := False
        end

    union (other: EBSET[Type]): EBSET[Type]
            -- FROM EBSET
        do
            Result := Current
        end

    intersection (other: EBSET[Type]): EBSET[Type]
            -- FROM EBSET
        do
            Result := Current
        end

    difference (other: EBSET[Type]): EBSET[Type]
            -- FROM EBSET
        do
            Result := Current
        end

    is_equal (other: Type): BOOLEAN
            -- FROM EBSET
        do
            Result := True
        end

    partition (sets: ARRAY[Type]): BOOLEAN
            -- FROM EBSET
        do
            Result := False
        end
end
```

Translating initialisation EventB event (init-rule)

Translation rule for Event-B initialisation event *initialisation* to Eiffel:

$$\frac{\xi(A(s,c,v)) = \mathtt{A}}{\begin{array}{l}\delta(\mathsf{event}\ initialisation\\ \quad\quad \mathsf{then}\\ \quad\quad\quad label:\ A(s,c,v)\\ \quad \mathsf{end}) =\\ \\ \quad \mathsf{initialisation}\\ \quad\quad\quad\quad \text{-- evt comment}\\ \quad\quad \textbf{do}\\ \quad\quad\quad \textbf{create}\ \mathsf{ctx}\\ \quad\quad\quad \mathsf{v.assigns}(\mathtt{A})\\ \quad\quad \textbf{ensure}\\ \quad\quad\quad \mathsf{label:\ v.is\_equal}(\textbf{old}\ \mathtt{A})\\ \quad\quad \textbf{end}\end{array}}\ (\mathsf{Init\text{-}rule})$$

Translating EventB events (E-rule)

Translation rule for Event-B events to Eiffel:

$$\frac{\begin{array}{l}\xi(G(s,c,v,x)) = \mathtt{G} \quad \xi(A(s,c,v,x)) = \mathtt{A}\\ \tau(x) = \mathtt{Type}\end{array}}{\begin{array}{l}\delta(\mathsf{event}\ evt\\ \quad\quad \mathsf{any}\\ \quad\quad\quad \mathsf{x}\\ \quad\quad \mathsf{where}\\ \quad\quad\quad label\_guard:\ G(s,c,v,x)\\ \quad\quad \mathsf{then}\\ \quad\quad\quad label\_action:\ A(s,c,v,x)\\ \quad\quad \mathsf{end}) =\\ \\ \quad \mathsf{evt}(\mathsf{x}:\mathtt{Type})\\ \quad\quad\quad\quad \text{-- 'evt' comment}\\ \quad\quad \textbf{require}\\ \quad\quad\quad \mathsf{label\_guard:}\ \mathtt{G}\\ \quad\quad \textbf{do}\\ \quad\quad\quad \mathsf{v.assigns}(\mathtt{A})\\ \quad\quad \textbf{ensure}\\ \quad\quad\quad \mathsf{label\_action:\ v.equals}(\textbf{old}\ \mathtt{A})\\ \quad\quad \textbf{end}\end{array}}\ (\mathsf{E\text{-}rule})$$

2 Appendix A: The EventB mathematical language translation to Eiffel

In the following sections, the following predicates, expressions, variables are given:

EventB	Eiffel translation	
Predicate	P	EBPRED p
Predicate	Q	EBPRED q
Expression	E	EBEXP e
Expression	F	EBEXP f
Set	S	EBSET[$\tau(S)$] s
Set	T	EBSET[$\tau(T)$] t

2.1 The propositional Language

Most of the EventB propositional language is translated to Eiffel constructs as shown in the following table:

EventB constructs	EventB symbol	Eiffel translation
falsity	$\bot$	**False**
	$\top$	**True**
negation	$\neg$	**not**
conjunction	$\wedge$	**and**
disjunction	$\vee$	**or**
implication	$\Rightarrow$	**implies**
bi-implication	$p \Leftrightarrow q$	(p **implies** q) **and** (q **implies** p)

2.2 The predicate Language

EventB constructs	EventB symbol	Eiffel translation
Universally Quantified Predicate	$\forall var\_list \cdot p$	(**create** var_list).for_all(p)
Existentially Quantified Predicate	$\exists var\_list \cdot p$	(**create** var_list).exists(p)
Paired Expression	$E \mapsto F$	**create** {EBPAIR[$\tau(E)$,$\tau(F)$]}.make (e,f)
List of variables	var_list	...
Equality	$E = F$	e.equals (f)
Equality	$E \neq F$	**not** e.equals (f)

2.3 The set-theoretic Language

EventB constructs	**EventB symbol**	**Eiffel translation**
Membership	$E \in S$	s.has (e)
Cartesian Product	$S \times T$	**create** {EBREL[$\tau(S)$,$\tau(T)$]}.cartesian_prod (s, t)
Power set	$\mathbb{P}(S)$	s.power_set
Set Comprehension	$\{x \cdot P \mid E\}$	...
Set Inclusion	$S \subseteq T$	s.is_subset (t)
Set Union	$S \cup T$	s.union (t)
Set Intersection	$S \cap T$	s.intersection (t)
Set Difference	$S \setminus T$	s.difference (t)
Empty Set	$\varnothing$	**create** {EBSET}.empty_set
Generalised Union	$E \in union(S)$	...
Quantified Union	$E \in \bigcup x \cdot P \mid T$	...
Generalised Intersection	$E \in inter(S)$	...
Quantified Intersection	$E \in \bigcap x \cdot P \mid T$	...

The following, shows the translation of a series of binary relation operators: the set of binary relations built on two sets, the domain and range of a binary relation, and then various sets of binary relations. This defines a binary relation in Eiffel

EventB symbol	**Eiffel translation**
$r \in S \overset{*}{\to} T$	r : EBREL[$\tau(s)$,$\tau(t)$] **create** r.default_create (s, t)

Where $\overset{*}{\to} \in \{\rel, \trel, \srel, \strel, \pfun, \tfun, \pinj, \tinj, \psur, \tsur, \tbij\}$. Additionally constraints are added as follows:

EventB constructs	**EventB symbol**	**Eiffel translation**
Set of all total relations	$r \in S \trel T$	r.is_total_rel
Set of all surjective relations	$r \in S \srel T$	r.is_surj_rel
Set of all total and surjective relations	$r \in S \strel T$	r.is_tsurj_rel
Set of all partial functions	$r \in S \pfun T$	r.is_partial_func
Set of all total functions	$r \in S \tfun T$	r.is_total_func
Set of all partial injections	$r \in S \pinj T$	r.is_pinj_func
Set of all total injections	$r \in S \tinj T$	r.is_tinj_func
Set of all partial surjections	$r \in S \psur T$	r.is_psurj_func
Set of all total surjections	$r \in S \tsur T$	r.is_tsurj_func
Set of all bijections	$r \in S \tbij T$	r.is_biject_func

Operations on relations and functions:

EventB constructs	**EventB symbol**	**Eiffel translation**
Domain	$dom(r)$	r.domain
Range	$ran(r)$	r.range
Converse	r^{-1}	r.converse
Domain Restriction	$S \lhd r$	r.domain_restriction
Range Restriction	$r \rhd T$	r.range_restriction
Domain Subtraction	$S ⩤ r$	r.domain_subtraction
Range Subtraction	$r ⩥ T$	r.range_subtraction
Relational Image	$r[U]$	r.rel_image (u)
Forward Composition	$f \mathbin{;} g$	**create** {EBREL_OPER[τ(f.domain),τ(g.range)]} .forward (f, g)
Backward Composition	$f \circ g$	**create** {EBREL_OPER[τ(g.range),τ(f.domain)]} .backward (f, g)
Overriding	$f g$	**create** {EBREL_OPER[τ(f.domain),τ(f.range)]} .override (f, g)
Direct Product	$f \otimes g$	**create** {EBREL_OPER[τ(f.domain), EBREL[τ(f.range),τ(g.range)]]}.direct_product (f, g)
Parallel Product	$f \parallel g$	**create** {EBREL_OPER [EBREL[τ(f.domain),τ(g.domain)], EBREL[τ(f.range),τ(g.range)]]}.parallel (f, g)
Identity	ID	...
First Projection	$\mathrm{prj}_1\, P$	p.prj1
Second Projection	$\mathrm{prj}_2\, P$	p.prj2
Lambda Expression	$\lambda L \cdot P \mid E$	...
Function Invocation	$f(E)$	...