Automated transformation of Event-B machines into EB4EB deep instances[1]

Rémy Sangoï, Yamine Aït-Ameur, Guillaume Dupont, Peter Rivière, Neeraj K. Singh

IRIT, Toulouse INP – ENSEEIHT, Toulouse, France
JAIST, Nomi, Japan

Rodin Workshop 2026 – May 2026

[1]This work was supported by grant ANR-19-CE25-0010 (EBRP Project)

Context

- EB4EB [RSA24] is a formal reflexive meta-modelling of Event-B within Event-B
- This framework enables the formalisation of Event-B machines as *a set of FOL/set theoretic formulas*
- Machines become *first-class citizen*: we can quantify on them and their components

  $\Rightarrow$ allows defining new *proof obligations*! (among other things)
- Supports two instantiation "modes": deep and shallow

One problem with EB4EB deep instances is that they have to be written *by hand*
In fact, they may be derived from an existing machine, almost syntactically
$\Rightarrow$ we can automate this transformation!

The EB4EB framework

Idea: meta-model Event-B machine as a datatype in a theory

```
THEORY EvtBTheo
TYPE PARAMETERS STATE, EVENT
DATATYPE Machine(STATE, EVENT)
  Cons_machine(
    Event: ℙ(EVENT),
    State: ℙ(STATE),
    Init: EVENT
    Progress: ℙ(EVENT),
    AP: ℙ(STATE)
    Grd: ℙ(EVENT × STATE)
    BAP: ℙ(EVENT × (STATE × STATE)),
    Inv: ℙ(STATE),
    Thm: ℙ(STATE),
    Variant: ℙ(STATE × ℤ),
    Ordinary: ℙ(EVENT),
    Convergent: ℙ(EVENT)
  )
```

```
MACHINE M
VARIABLES v
INVARIANTS
  inv1: v ∈ STATE
  inv2: Inv(v)
  thm1: Thm(v) theorem
VARIANT Exp(v)
EVENTS
  INITIALISATION
  THEN v :| AP(v')

  -- events other than init are "progress"
  Event ordinary
  WHERE Grd(v)
  THEN v :| BAP(v, v')
END
```

This enables writing formulas such as:

$$\forall m, e, s, sp \cdot m \in \mathbf{Machine}(STATE, EVENT) \land e \in Event(m) \land s \in STATE \land sp \in STATE \land$$
$$s \in Inv(m) \land s \in Grd(m)[\{e\}] \land (s \mapsto sp) \in BAP(m)[\{e\}] \Rightarrow sp \in Inv(m)$$

(or, more simply:)

$$\forall m, e \cdot m \in \mathbf{Machine}(STATE, EVENT) \land e \in Event(m) \Rightarrow$$
$$BAP[\{e\}][Inv(m) \cap Grd(m)[\{e\}]] \subseteq Inv(m)$$

Machine well-construction and consistency

A number of predicates are defined to check a machine is "well constructed", and to associate POs with it

BAP_WellCons *predicate*
$(m : Machine(STATE, EVENT))$
direct definition $\mathrm{dom}(BAP(m)) = Progress(m)$
Grd_WellCons *predicate*
$(m : Machine(STATE, EVENT))$
direct definition $\mathrm{dom}(Grd(m)) = Progress(m)$
Event_WellCons *predicate*
$(m : Machine(STATE, EVENT))$
direct definition $\mathrm{partition}(Event(m), \{Init(m)\}, Progress(m)\}$
...
Machine_WellCons *predicate*
$(m : Machine(STATE, EVENT))$
direct definition $BAP\_WellCons(m) \wedge Grd\_WellCons(m) \wedge Event\_WellCons(m) \wedge$
...

Mch_INV_Init *predicate*
$(m : Machine(STATE, EVENT))$
direct definition $AP(m) \subseteq Inv(m)$
Mch_INV_One_Ev *predicate*
$(m : Machine(STATE, EVENT), e : EVENT)$
well-definedness $e \in Progress(m)$
direct definition
$BAP(m)[\{e\}][Inv(m) \cap Grd(m)[\{e\}]] \subseteq Inv(m)$
Mch_INV *predicate* $(m : Machine(STATE, EVENT))$
direct definition $Mch\_INV\_Init(m) \wedge (\forall e \cdot e \in Progress(m) \Rightarrow Mch\_INV\_One\_Ev(m, e))$
...

Then, the theory provides the predicate that, given a well-constructed machine, yields the conjunction of all the POs generated from it

check_Machine_Consistency *predicate* $(m : Machine(STATE, EVENT))$
well-definedness Machine_WellCons(m)
direct definition $Mch\_THM(m) \wedge Mch\_INV(m) \wedge Mch\_FIS(m) \wedge Mch\_VARIANT(m) \wedge Mch\_NAT(m)$

An example

MACHINE Clock
VARIABLES m, h
INVARIANTS
  inv1: $m \in \mathbb{N} \land h \in \mathbb{N}$
  inv2: $m < 60 \land h < 24$
EVENTS
  INITIALISATION
  THEN $m, h :\mid m' = 0 \land h' = 0$

  tick_min
  WHERE $m < 59$
  THEN $m :\mid m' = m + 1$

  tick_hour
  WHERE $m = 59 \land h \neq 23$
  THEN $m, h :\mid m' = 0 \land h' = h + 1$

  tick_day
  WHERE $m = 59 \land h = 23$
  THEN $m, h :\mid m' = 0 \land h' = 0$
END

CONTEXT ClockInstance
SETS Ev
CONSTANTS $clock$, $init$, $tick\_min$, $tick\_hour$, $tick\_day$
AXIOMS
  axm1: $\text{partition}(Ev, \{init\}, \{tick\_min\}, \{tick\_hour\}, \{tick\_day\})$
  axm2: $clock \in Machine(\mathbb{Z} \times \mathbb{Z}, Ev)$
  axm3: $State(clock) = \mathbb{Z} \times \mathbb{Z}$
  axm4: $Event(clock) = Ev$
  axm6: $Inv(clock) = \{m \mapsto h \mid m \in \mathbb{N} \land h \in \mathbb{N} \land m < 60 \land h < 24\}$
  axm5: $Init(clock) = init$
  axm6: $Grd(clock) = \{e \mapsto (m \mapsto h) \mid$
    $(e = tick\_min \land m < 59) \lor$
    $(e = tick\_hour \land m = 59 \land h \neq 23) \lor$
    $(e = tick\_day \land m = 59 \land h = 23)$
    $\}$
  axm7: $Evt(clock) = \{e \mapsto ((h \mapsto m) \mapsto (h' \mapsto m')) \mid$
    $(e = tick\_min \land m' = m + 1 \land h' = h) \lor$
    $(e = tick\_hour \land m' = 0 \land h' = h + 1) \lor$
    $(e = tick\_day \land m' = 0 \land h' = 0)$
    $\}$
  ...
  thm1: $check\_Machine\_consistency(m)$ *theorem*
END

This is this correspondence we want to make!

The transformation

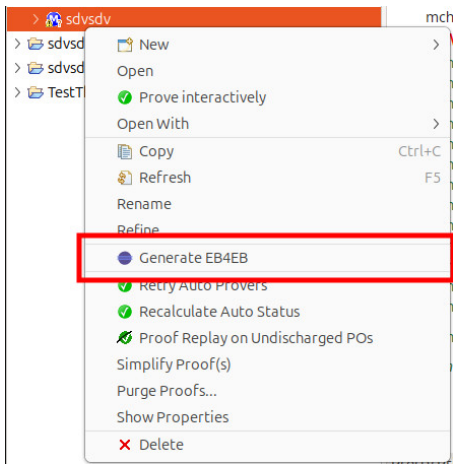
Main ideas:

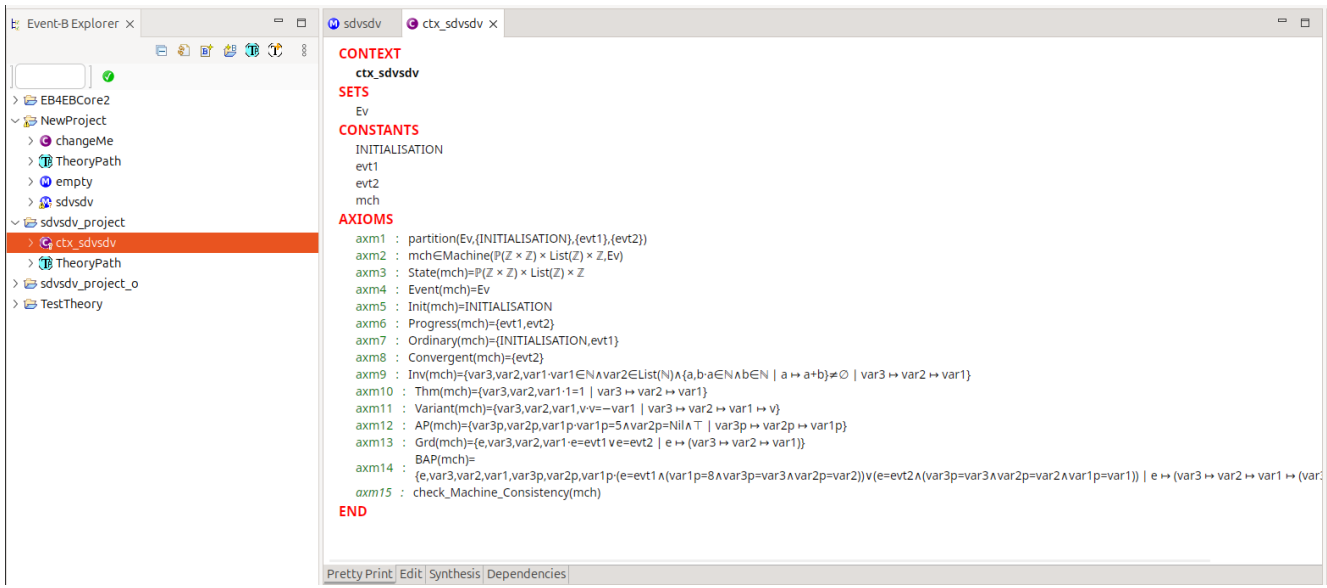
- find and type variables $\Rightarrow$ constitute the state space
- for each event, normalize BAP as :| and add unchanged variables ($x' = x$)
- create and populate Ev (provide a constant for each event label)
- create $m \in Machine$ and the axioms $State(m) = \ldots$, $Event(m) = \ldots$, etc.
- create theorem $clock\_Machine\_consistency(m)$

This transformation was implemented in a Rodin plug-in[2]

[2] `https://github.com/r-sangoi/custom_po_eb4eb_rodin`

Screenshots





Defining new POs

EB4EB enables defining new POs (or *analyses*)

```
THEORY Theo4PO IMPORT EvtBTheo
TYPE PARAMETERS STATE, EVENT, T_args
OPERATORS
  [PO]_Definition predicate (m: Machine(EVENT, STATE), args: T_args)
    ...
  check_machine_[PO] predicate (m: Machine(EVENT, STATE), args: T_args)
    well-definedness condition Machine_WellCons(m) ∧ ...
    direct definition [PO]_Definition(m, args)
END
```

Usage:

```
CONTEXT MachinePO EXTENDS MachineInstance
AXIOMS
  axm1: check_machine_*)[PO](m, args)
END
```

Defining new POs – Example

```
THEORY Theo4Deadlock IMPORT EvtBTheo
TYPE PARAMETERS STATE, EVENTB
OPERATORS
  DeadlockFreeness_Definition predicate (m: Machine(STATE, EVENT))
    direct definition Inv(m) ⊆ Grd(m)[Progress(m)]
  check_Machine_DeadlockFreeness predicate (m: Machine(STATE, EVENT))
    well-definedness condition Machine_WellCons(m)
    direct definition DeadlockFreness_Definition(m)
END
```

```
CONTEXT ClockDeadlockFreeness EXTENDS ClockInstance
AXIOMS
  thmDeadlockFreeness: check_Machine_DeadlockFreeness(m)
END
```

Can we, given the PO definition, generate something like that?

Annotation system and new POs

Idea: let the user define new POs and annotate Rodin components

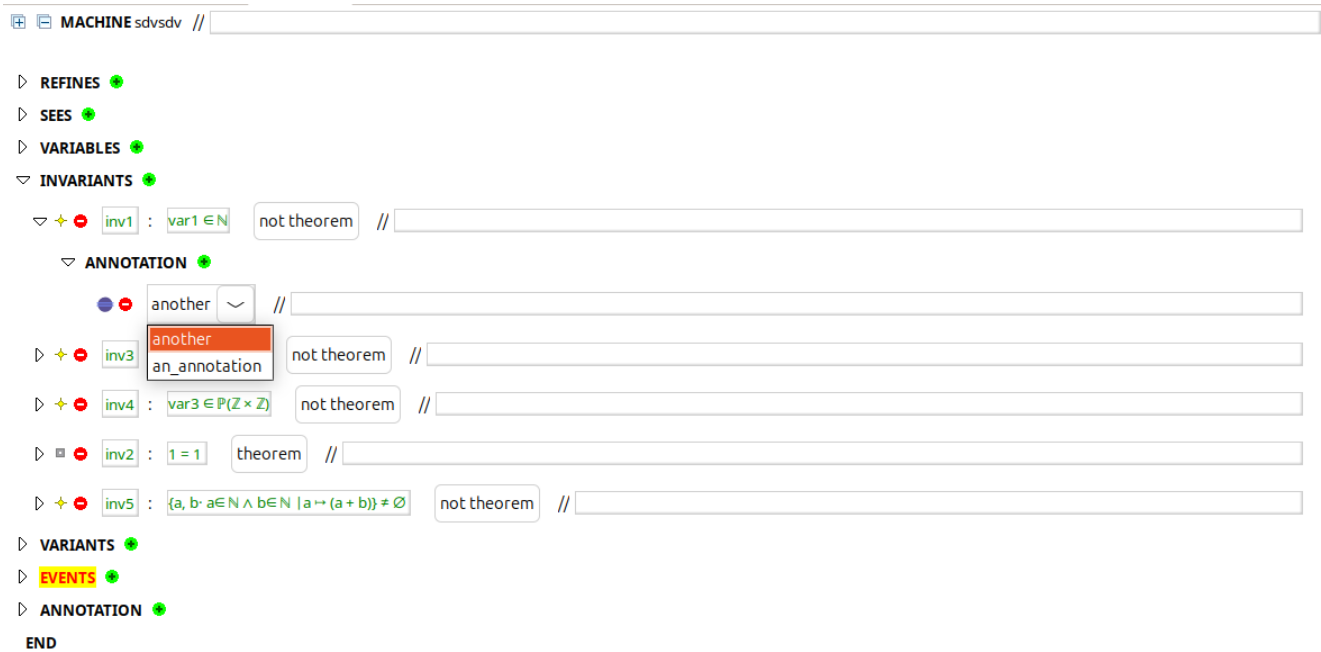
Annotation are defined in a dedicated annotation component:

The content of the annotation is defined through predicates in a theory:

•**DeadlockFreeness_Definition** : DeadlockFreeness_Definition(m : Machine(STATE,EVENT)) **PREDICATE PREFIX**
direct definition
DeadlockFreeness_Definition(m : Machine(STATE,EVENT)) ≙ Inv(m)⊆Grd(m)[Progress(m)]
•**check_Machine_DeadLockFreeness** : check_Machine_DeadLockFreeness(m : Machine(STATE,EVENT)) **PREDICATE PREFIX**
well-definedness condition
Machine_WellCons(m)
direct definition
check_Machine_DeadLockFreeness(m : Machine(STATE,EVENT)) ≙ DeadlockFreeness_Definition(m)

Usage and result

Once defined, the annotation becomes accessible in the model (for the specific type of element it is attached to)



As a result, the PO predicate associated with the annotation is added:

```
CONTEXT Machine_Deep
...
  inv1: m ∈ Machine(STATE, Ev)
...
  thm1: check_Machine_consistency(m) theorem
  thm2: check_Machine_DeadLockFreeness(m) theorem -- From Machine annotation an_annotation
END
```

Assessment

Current work:

- the transformation Event-B $\rightarrow$ EB4EB context is functional
- the annotation system is not complete yet... (we mostly have the skeleton, we are missing the mechanics)

Future work:

- do it in the other direction: generate an Event-B machine from an EB4EB instance
- support "invited" semantics to specialised EB4EB models (hybrid, probabilistic, etc.)

Part I

Bibliography

References I

[RSA24] Peter Rivière, Neeraj Kumar Singh, and Yamine Aït-Ameur. "Reflexive Event-B: Semantics and Correctness, the EB4EB Framework". In: *IEEE Trans. Reliab.* 73.2 (2024), pp. 835–850. DOI: 10.1109/TR.2022.3219649.