Extending EB4EB for Parameterised Events

Peter Rivière[1], Neeraj Kumar Singh[2],
Guillaume Dupont[2], Yamine Aït Ameur[2]

[1] *JAIST - Japan Advanced Institute of Science and Technology, Ishikawa, Japan*
[2] *INPT-ENSEEIHT/IRIT, University of Toulouse, France*
priviere@jaist.ac.jp
{nsingh, guillaume.dupont, yamine}@enseeiht.fr

EB4EB [5,6], standing for *Event-B for Event-B*, is a framework that supports the formalisation of Event-B [1] models using first-order logic (FOL) and set-theory. This framework can handle machine elements as formulas, thus the EB4EB framework enables the definition of new specific proof obligations and *analyses* [8,4,7]. In the earlier formalisation of the EB4EB framework only *states* and *events* were handled, limiting the expressive reasoning power of the framework. In this paper, we present an overview of an extension of the EB4EB framework to support parameterised events [9], an important feature of Event-B. This extension is not straightforward in EB4EB. Indeed, the typing system supported by Event-B theories [2,3] is not rich enough to describe such extension in a constructive manner as for the other Event-B features formalised in EB4EB. The proposed solution, described in this paper, consists in defining an axiomatic formalisation of event parameters definitions.

EB4EB Extension. Listing 1 presents the extended version of the EB4EB meta-theory. The `EvtBTheoPar` includes a new type parameter, `PARAM`, for abstracting the type of event parameters. The main difference between the former EB4EB meta-theory and this one is the definition of the *destructors*. The $Machine$ data-type still has the same signature as well as a single constructor ($Cons\_machine$); however, this constructor only has two arguments (and thus two destructors): $Event$ identifying the events of the machine and $State$ identifying its state.

The usual destructors of the machine data-type (Inv, $Progress$, etc...) are defined *as axiomatic operators*, so that they can be free from the limitation of data-types regarding type parameters.

The $PARAM$ is defined as a type bound in the operators but not in the $Machine$ data-type, serving as a means to universally quantifying it independently from the data-type definition. Specifically, when the same operator (e.g., BAP_par) is used in two different contexts within the same machine $m : Machine(STATE, EVENT)$, both instances must reference the same sets for $STATE$ and $EVENT$, as these are fixed by the machine's type. However, the sets for $PARAM$ do not need to be identical, since they are not constrained by the machine's type. This allows each occurrence of $Param$, BAP_par, and Grd_par to have different types, even when used within the same machine.

THEORY EvtBTheoPar
TYPE PARAMETERS $STATE, EVENT, PARAM$
DATATYPES

$Machine(STATE, EVENT)$
CONSTRUCTORS
$Cons\_machine($
$Event : \mathbb{P}(EVENT),$
$State : \mathbb{P}(STATE))$
AXIOMATIC DEFINITIONS
OPERATORS
Param <*expression*> $(m : Machine(STATE, EVENT), e : EVENT) : \mathbb{P}(PARAM)$
Grd_par <*expression*>
$(m : Machine(STATE, EVENT), e : EVENT) : \mathbb{P}(PARAM \times STATE)$
BAP_par <*expression*>
$(m : Machine(STATE, EVENT), e : EVENT) : \mathbb{P}((PARAM \times STATE) \times STATE)$
...
AXIOMS ...

Listing 1. Machine Data-type with parameter

Instantiation principle for parameters. The instantiation of the *EvtB-TheoPar* theory from Listing 1, which introduces parameters, requires the definition of a set of axioms that encode an Event-B machine. The approach involves specifying the different components of the machine through definition axioms– predicates of the form $Op(m, \ldots) = Expr$.

CONTEXT EvtInstantiationSchema
AXIOMS
...
AxmParEv.1 : $Param(m, ev1) = T_{Par}^{ev1}$ ...
AxmParEv.n: $Param(m, evn) = T_{Par}^{evn}$...
AxmParGrd.1 : $Grd\_par(m, ev1) = \{par \mapsto s \mid par \in T_{Par}^{ev1} \wedge \ldots\}$ ...
AxmParGrd.n: $Grd\_par(m, evn) = \{par \mapsto s \mid par \in T_{Par}^{evn} \wedge \ldots\}$ ...
AxmParBAP.1 : $BAP\_par(m, ev1) = \{(par \mapsto s) \mapsto sp \mid par \in T_{Par}^{ev1} \wedge \ldots\}$ ...
AxmParBAP.n: $BAP\_par(m, evn) = \{(par \mapsto s) \mapsto sp \mid par \in T_{Par}^{evn} \wedge \ldots\}$ ...

Listing 2. Event instantiation schema

Machine POs. In addition to the introduction of event parameters and axiomatic definition, we must also update the defined PO operators. The proof obligations have been updated (defined axiomatically); the operators defining the invariant preservation PO are shown in Listing 3. The PO is divided into two parts: the base case and the induction case with the event. Note that the base and induction cases take into account type homogeneity in their axiomatic definitions and associated POs, respectively. The axiomatic definition of the data-type machine allows for including the parameter `PARAM` type in the proof obligation definition for specific events. Other POs operators are also updated in the same way.

OPERATORS
...
Mch_INV_One_Ev_Par_Def ***predicate*** $(m : Machine(STATE, EVENT),$
$e : EVENT)$
well–definedness $e \in Progress(m)$
direct definition
$BAP\_par(m, e)[(Param(m, e) \times Inv(m)) \cap Grd\_par(m, e)] \subseteq Inv(m)$
Mch_INV_Init ***predicate*** $(m : Machine(STATE, EVENT))$
direct definition
$AP(m) \subseteq Inv(m)$
Mch_INV ***predicate*** $(m : Machine(STATE, EVENT))$
direct definition
$Mch\_INV\_Init(m) \wedge (\forall e \cdot e \in Progress(m) \Rightarrow Mch\_INV\_One\_Ev\_Def(m, e))$
...

Listing 3. Well defined Data-type operators with parameter (behavioural semantics)

Conclusion. Unlike the original EB4EB framework, which employs constructive definitions for all types and operators within the associated Event-B theory, our approach utilises axiomatic definitions for event parameters. This allows for the instantiation of the theory to define various parameters with differing sets as their types, providing greater flexibility. Our approach has been applied to several examples to demonstrate the flexibility, reliability, and scalability of the extended EB4EB framework in terms of modelling, expressive power, and simplification of the proof process. More details can be found in [9].

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., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Tech. rep. (2009), `http://deploy-eprints.ecs.soton.ac.uk/216/`
3. Butler, M.J., Maamria, I.: Practical theory extension in Event-B. In: Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. pp. 67–81 (2013)
4. Mendil, I., Riviere, P., Aït Ameur, Y., Singh, N.K., Méry, D., Palanque, P.A.: Non-intrusive annotation-based domain-specific analysis to certify event-b models behaviours. In: 29th Asia-Pacific Software Engineering Conference, APSEC. pp. 129–138. IEEE (2022)
5. Riviere, P., Singh, N.K., Aït Ameur, Y.: EB4EB: A Framework for Reflexive Event-B. In: International Conference on Engineering of Complex Computer Systems, ICECCS 2022. pp. 71–80. IEEE (2022)
6. Riviere, P., Singh, N.K., Aït Ameur, Y.: Reflexive Event-B: Semantics and Correctness the EB4EB Framework. IEEE Transactions on Reliability pp. 1–16 (2022)
7. Riviere, P., Singh, N.K., Aït Ameur, Y., Dupont, G.: Formalising liveness properties in event-b with the reflexive EB4EB framework. In: NFM. Lecture Notes in Computer Science, vol. 13903, pp. 312–331. Springer (2023)
8. Riviere, P., Singh, N.K., Aït-Ameur, Y., Dupont, G.: Standalone Event-B models analysis relying on the EB4EB meta-theory. In: ABZ. Lecture Notes in Computer Science, vol. 14010, pp. 193–211. Springer (2023)
9. Rivière, P., Singh, N.K., Aït-Ameur, Y., Dupont, G.: Extending the EB4EB framework with parameterised events. Sci. Comput. Program. **243**, 103279 (2025). https://doi.org/10.1016/J.SCICO.2025.103279, `https://doi.org/10.1016/j.scico.2025.103279`