Formalising Liveness Properties in Event-B with the Reflexive EB4EB Framework

Peter Riviere, Neeraj Kumar Singh, Yamine Aït-Ameur, Guillaume Dupont

Formalising Liveness Properties in Event-B with the Reflexive EB4EB Framework

P. Rivière[1], N. K. Singh[1], Y. Aït-Ameur[1], and G. Dupont[1]

INPT-ENSEEIHT/IRIT, University of Toulouse, France
{peter.riviere, nsingh, yamine, guillaume.dupont}@enseeiht.fr

Abstract. The correct-by-construction state-based Event-B formal method lacks the ability to express liveness properties using temporal logic. To address this challenge, two approaches can be envisioned. First, embed Event-B models in another formal method supporting liveness properties verification. This method is cumbersome and error-prone, and the verification result is not guaranteed on the source model. Second, extend Event-B to support the expression of and reasoning on liveness properties, and more generally temporal properties. Following the second approach, in [18], J.-R. Abrial and T. S. Hoang proposed an axiomatisation of linear temporal logic (LTL) for Event-B with a set of proof obligations (POs) allowing to verify these properties. These POs are mathematically formalised, but are neither implemented nor generated automatically. In this paper, using the reflexive EB4EB framework [35,36] allowing for manipulation of the core concepts of Event-B, we propose to formalise and operationalise the automatic generation of proof obligations associated to liveness properties expressed in LTL. Furthermore, relying on trace-based semantics, we demonstrate the soundness of this formalisation, and provide a set of intermediate and generic theorems to increase the rate of proof automation for these properties. Finally, a case study is proposed to demonstrate the use of the defined operators for expressing and proving liveness properties.

Keywords: Proof and state-based methods · Event-B and Theories · Metatheory · Reflexive EB4EB framework · Temporal logic · Liveness properties · Traces and soundness

1 Introduction

Event-B is a formal method based on explicit state expression, refinement and formal proof. It enables the design of complex systems using correct by construction. This method has been used successfully in the design of many complex systems in various engineering areas such as aeronautics [40], railway systems [7,8], health and medicine [38], etc. In particular, it has shown its effectiveness in establishing properties related to system functionalities, safety, security, reachability and compliance with some temporal requirements, and so on.

Event-B models are machines that express state-transition systems using set theory and first-order logic (FOL). A mechanism of proof by induction enables the demonstration of inductive properties based on the preservation of properties at initialization and by each transition (event). Refinement, on the other hand, is defined by a weak simulation relation in which proof obligations guarantee the preservation of behaviours between levels of abstraction. The Rodin platform supports the development of Event-B models. It offers an environment for model editing, automatic and interactive proofs, animation, model checking, etc.

However, Event-B, like all formal methods, lacks some capabilities. Event-B supports the verification of a fragment of temporal logic properties: $\Box$ using invariants and theorem clauses and $\Diamond$ using variants and convergence proof obligations. There is a lack of composition of temporal logic operators, as well as the ability to express and reason about liveness properties. To remedy this absence, two solutions are possible in general. The first solution consists in embedding an Event-B model in another formal method offering the possibility of expressing and reasoning about liveness properties such as TLA$^+$ [23], NuSMV [11], PRISM [21], PAT [41], Spin [20], Uppaal [2], ProB [25] etc. However, tracing the verification results on the source Event-B models is difficult and care must be taken to guarantee the correctness of this embedding. This approach is very popular and is followed by many authors who use other formal methods allowing to express and verify this type of property without worrying about the correctness of the transformation. However, there exist several approaches to ensuring the transformation's correctness [24,16,33,6]. The second solution consists in extending the Event-B method to allow expressing and reasoning about liveness properties. This second approach requires the expression of the semantics and the proof system of the temporal logic in Event-B, as well as establishing the soundness of this extension.

Based on the second approach, JR. Abrial and TS. Hoang [18] proposed an axiomatisation of LTL temporal logic for Event-B in their article entitled *"Reasoning about liveness properties in Event-B"*. This work has defined a set of proof obligations allowing to establish temporal properties such as reachability, progress, persistence or until. However, these proof obligations are mathematically formalised in that paper but are neither implemented nor generated automatically. They must be explicitly described in Event-B by the developer for each model, thus leading to formalization errors. Moreover, their proofs are cumbersome and require too much manual effort to proving them.

Relying on the reflexive EB4EB framework [35,36,37] defined in Event-B, we propose to formalise and operationalise the automatic generation of proof obligations associated with liveness properties expressed in LTL temporal logic. We define an extension of EB4EB including a set of operators expressing these properties on traces. In addition, we demonstrate the soundness of these properties on model traces. Finally, a set of intermediate and generic theorems are also proposed to increase the rate of proof automation.

Context	Machine	Theory

CONTEXT Ctx
SETS s
CONSTANTS c
AXIOMS A
THEOREMS T_{ctx}
END
	MACHINE M
	SEES Ctx
	VARIABLES x
	INVARIANTS $I(x)$
	THEOREMS $T_{mch}(x)$
	VARIANT $V(x)$
	EVENTS
	  EVENT evt
	  ANY α
	  WHERE $G_i(x, \alpha)$
	  THEN
	  $x :\mid BAP(\alpha, x, x')$
	  END
	  ...
	END
		THEORY Th
		IMPORT Th1, ...
		TYPE PARAMETERS E, F, ...
		DATATYPES
		  Type1(E, ...)
		  constructors
		  cstr1(p_1: T_1, ...)
		OPERATORS
		  Op1 <nature> (p_1: T_1, ...)
		  well—definedness $WD(p_1, ...)$
		  direct definition D_1
		AXIOMATIC DEFINITIONS
		  TYPES A_1, ...
		  OPERATORS
		  AOp2 <nature> (p_1: T_1, ...): T_r
		  well—definedness $WD(p_1, ...)$
		  AXIOMS A_1, ...
		THEOREMS T_1, ...
		PROOF RULES R_1, ...
		END

(a)	(b)	(c)

Table 1: Global structure of Event-B Contexts, Machines and Theories

Note that our proposed approach is non-intrusive (self-contained) and does not require the use of any other formal techniques or tools; it is fully formalised in Event-B and mechanised on the Rodin platform.

This paper is organised as follows. Section 2 describes the Event-B modelling language and its Theory plugin extension. Section 3 recalls linear temporal logic, and the EB4EB framework is described in Section 4. Section 5 presents the trace-based semantics of Event-B, and its soundness properties. Section 6 describes a case study that will be used as a running example to show how to use defined LTL operators. Section 7 presents the temporal logic proof rules encoded as EB4EB proof obligations. Their correctness is discussed in Section 8. Section 9 summarises related work, and Section 10 concludes the paper.

2 Event-B

Event-B [1] is a state-based, *correct-by-construction* formal method, where systems are modelled with a set of events representing state changes, using first-order logic (FOL) and set theory.

Contexts and machines (Tables 1.a and 1.b). `Contexts` (Table 1.a) encompass the model's *static* part: *carrier sets* s and *constants* c, as well as their properties, through *axioms* A and *theorems* T_{ctx}. `Machines` (Table 1.b) describe the model's behaviour, using a set of *events* evt, each of which may be guarded G and/or parameterized by α. An event models the evolution of a set of variables x using a Before-After Predicate (BAP) that links the before (x) and after (x') value of the variables. Safety properties are encoded using *invariants* $I(x)$ and *theorems* $T_{mch}(x)$, and *variants* $V(x)$ may be defined to demonstrate the machine's convergence. Model consistency is established by discharging a number of automatically generated POs (Table 2).

Refinements. One strength of Event-B is its *refinement* operation, which is used to transform an abstract model into a more concrete one, adding information (refined states) and behavioural (refined events) details gradually, while

(1) Ctx Theorems (ThmCtx)	$A(s,c) \Rightarrow T_{ctx}$ `(For contexts)`
(2) Mch Theorems (ThmMch)	$A(s,c) \wedge I(x) \Rightarrow T_{mch}(x)$ `(For machines)`
(3) Initialisation (Init)	$A(s,c) \wedge BAP(x') \Rightarrow I(x')$
(4) Invariant preservation (Inv)	$A(s,c) \wedge I(x) \wedge G(x,\alpha) \wedge BAP(x,\alpha,x') \Rightarrow I(x')$
(5) Event feasibility (Fis)	$A(s,c) \wedge I(x) \wedge G(x,\alpha) \Rightarrow \exists x' \cdot BAP(x,\alpha,x')$
(6) Variant progress (Var)	$A(s,c) \wedge I(x) \wedge G(x,\alpha) \wedge BAP(x,\alpha,x') \Rightarrow V(x') < V(x)$

Table 2: Relevant Proof Obligations for Event-B contexts and machines

retaining a similar observational behaviour (simulation relationship). Refinement correctness is established with the help of a gluing invariant, and ensures properties are preserved from the abstract to the concrete model.

Extension with theories. Being based on set theory and FOL, the Event-B formalism is mathematically low-level and thus very expressive. However, it lacks features to build up more complex structures. The theory extension has been proposed to address this issue [9]. A theory is a type of component that makes it possible to define new type-generic datatypes together with constructive and axiomatic operators, specific theorems and axioms and even proof rules (see Table 1.c). The resulting theories consistency can be established by providing witnesses for axioms and definitions, ensuring conservative extensions of Event-B. Once defined, elements of a theory become seamlessly available in an Event-B model and its proofs.

This extension is central for embedding, as data types, *concepts that are unavailable in core Event-B*, similar to Coq [3], Isabelle/HOL [30] or PVS [31]. Many theories have been defined, for supporting real numbers, lists, differential equations and so on.

Well-definedness (WD). Beyond machine-related POs, one key aspect of model consistency is the *well-definedness* (WD) of the expressions involved in it. This notion supplements the one of syntactical correctness with the idea of a formula being "meaningful", i.e. it can always be safely evaluated (e.g., dividing by a constant that is provably non 0). Each formula of a model is associated to a WD PO, usually consisting in checking that operators are correctly used and combined. Once proven, WDs are added to set of hypotheses of other POs.

Note that theories allow designers to provide custom WD conditions for partially defined operators in order to precisely characterise their proper use.

The Rodin Platform. Rodin is an open source integrated development platform for designing, editing and proving Event-B models. It also supports model checking and animation with ProB, as well as code generation. Being based on Eclipse, it also allows the definition of *plug-ins*, including theory extensions. Many provers for first-order logic as well as SMT solvers are plugged to Rodin for helping the proof process.

3 Linear Temporal Logic

This section recalls the principles of linear temporal logic (LTL) following the definition of Manna and Pnueli [26]. Linear temporal logic is defined syntactically

as an extension of propositional logic. A valid LTL formula consists in literals (usually, predicates on the state of the system), the usual logical connectors ($\wedge$, $\vee$, $\neg$ and $\Rightarrow$) as well as *modal operators* $\Box$, $\Diamond$ and $\mathcal{U}$. The semantics of LTL is expressed in terms of *traces* of a system. Given a trace $tr = s_0 \mapsto s_1 \mapsto \ldots$, then tr_i ($i \in \mathbb{N}$) denotes the *suffix* trace of tr, starting from s_i, $tr_i = s_i \mapsto s_{i+1} \mapsto \ldots$

A state that satisfies a predicate P is called a P-state. LTL semantics are given with the following rules:

1. For any state predicate P, $tr \vDash P$ iff s_0 is a P-state.
2. $tr \vDash \phi_1 \wedge \phi_2$ iff $tr \vDash \phi_1$ and $tr \vDash \phi_2$
3. $tr \vDash \phi_1 \vee \phi_2$ iff $tr \vDash \phi_1$ or $tr \vDash \phi_2$
4. $tr \vDash \neg\phi$ iff not $tr \vDash \phi$
5. $tr \vDash \phi_1 \Rightarrow \phi_2$ iff not $tr \vDash \phi_1$ or $tr \vDash \phi_2$
6. $tr \vDash \Box\phi$ iff for all k, $tr_k \vDash \phi$
7. $tr \vDash \Diamond\phi$ iff there exists a i such that $tr_i \vDash \phi$
8. $tr \vDash \phi_1 \mathcal{U} \phi_2$ iff there exists a i such that $tr_i \vDash \phi_2$, and for all $j < i$, $tr_j \vDash \phi_1$

A machine M satisfies a property ϕ, denoted $M \vDash \phi$ if and only if for all traces tr of M, that trace satisfies ϕ ($tr \vDash \phi$).

4 The EB4EB Framework

The EB4EB framework [35,36] proposes to extend the reasoning capabilities of Event-B by enabling the access of Event-B components as first-class citizens within Event-B models (reflection), thereby making it possible to express new reasoning mechanism at the meta-level.

```
THEORY EvtBTheo
TYPE PARAMETERS St, Ev
DATATYPES Machine(St, Ev)
 CONSTRUCTORS
    Cons_machine(
       Event : ℙ(Ev),
       State : ℙ(St),
       Init : Ev, Progress : ℙ(Ev)
       Variant : ℙ(St × ℤ),
       AP : ℙ(St),
       BAP : ℙ(Ev × (St × St)),
       Grd : ℙ(Ev × St),
       Inv : ℙ(St),
       ...)
```

Listing 1: Machine Data type

```
Event_WellCons <predicate>
      (m : Machine(St, Ev))
   direct definition
      partition(Event(m), {Init(m)}, Progress(m))
...
Machine_WellCons <predicate>
      (m : Machine(St, Ev))
   direct definition Event_WellCons(m) ∧ ...
```

Listing 2: Operators to check well-defined data type (static semantics)

Machine structure. Event-B is formalised in an Event-B theory. A machine is represented using the data-type `Machine` (see Listing 1) parameterised by generic types with event labels (`Ev`) and states (`St`). Constructor `Cons_machine` gathers the components of a machine, such as `Event`, `State`, `Grd`, `Inv`, `BAP`, etc.

Well-Construction. A machine built using `Cons_machine` may not be consistent, despite being syntactically correct. Thus, additional operators are defined to encode the *well-construction* of a machine, i.e. the consistency of its components with regard to each others (Listing 2). For instance, `Event_WellCons` ensures that events are partitioned between initialisation and progress events.

Machine Proof Obligations. For any machine expressed in the framework, its associated proof obligations are provided under the form of operators (see Listing 3). Such operators are predicates that rely on the set-theoretical definition of the machine and guarded transition system semantics.

In a particular, for a given machine m the predicate `Mch_INV`(m) holds if and only if the invariants of m hold with regard to m's behaviour, corresponding to PO INV (see Table 2). Following similar principles, every machine-related POs of the Event-B method is formalised in the theory.

Mch_INV_Init <*predicate*> $(m : Machine(St, Ev))$
  direct definition $AP(m) \subseteq Inv(m)$
Mch_INV_One_Ev <*predicate*> $(m : Machine(St, Ev), e : Ev)$
  well−definedness $e \in Progress(m)$
  direct definition $BAP(m)[\{e\}][Inv(m) \cap Grd(m)[\{e\}]] \subseteq Inv(m)$
Mch_INV <*predicate*> $(m : Machine(St, Ev))$
  direct definition
    $Mch\_INV\_Init(m) \land (\forall e \cdot e \in Progress(m) \Rightarrow Mch\_INV\_One\_Ev(m, e))$
...

Listing 3: Well-defined data type operators (behavioural semantics)

Finally, the PO operators are all gathered in a conjunctive expression within the `check_Machine_Consistency` operator (Listing 4), which thus encode the correctness condition for the machine. It uses `Machine_WellCons` as WD condition. At instantiation, it is used as a theorem to ensure machine correctness.

check_Machine_Consistency <*predicate*> $(m : Machine(St, Ev))$
  well−definedness $Machine\_WellCons(m)$
  direct definition $Mch\_INV(m) \land \ldots$

Listing 4: Operator for Event-B machine consistency

Remark. The EB4EB framework makes accessible all the features of Event-B machines, and thus enables the formalisation and verification of the fragment of temporal logic properties already supported by classical Event-B machines: $\Box$ using invariants and theorem clauses and $\Diamond$ using variants and convergence proof obligations. However, it does not support the composition of these operators nor any of the other temporal logic properties.

Instantiation of the meta-theory is used to define specific Event-B machines (instantiation) using the `Cons_machine` constructor. An Event-B context where values for the type parameters `St` and `Ev` are provided.

5 Trace-Based Semantics of Event-B

Establishing the *correctness* of the POs provided in the EB4EB framework requires modelling of Event-B trace-based semantics. We express traces in an Event-B theory and relate them to an EB4EB machine. It becomes possible to prove that a PO defined in EB4EB encodes correctly the property it formalises.

5.1 Semantics: traces of Event-B machines in EB4EB

A machine m consists of state variables and events describing their evolution. A trace tr of m is a sequence of states $tr = s_0 \mapsto s_1 \mapsto \ldots \mapsto s_n \mapsto \ldots$ such that:

1. the initial state s_0 satisfies the after predicate (AP) of the initialisation event
2. each pair of consecutive states s_i, s_{i+1} corresponds to the activation of an event e of m, i.e.: 1) s_i verifies the guard, and 2) $s_i \mapsto s_{i+1}$ verifies the BAP
3. if tr is *finite*, its final state deadlocks (i.e., system cannot progress any more)

In EB4EB, traces are encoded in a theory (Listing 5) extending `EvtBTheo`. They are linked to machines. A trace is a partial function $tr \in \mathbb{N} \nrightarrow St$ such that, for any n in the domain, $tr(n) = s_n$ is the n-th state of the trace.

THEORY *EvtBTraces* **IMPORT** *EvtBTheo*
TYPE PARAMETERS St, Ev
OPERATORS
IsANextState *predicate* $(m : Machine(St, Ev), s : St, sp : St)$
direct definition $\exists e \cdot e \in Progress(m) \land s \in Grd(m)[\{e\}] \land s \mapsto sp \in BAP(m)[\{e\}]$
IsATrace *predicate* $(m : Machine(St, Ev), tr : \mathbb{P}(\mathbb{N} \times St))$
direct definition
$(tr \in \mathbb{N} \to St \lor (\exists n \cdot n \in \mathbb{N} \land tr \in 0..n \to St \land tr(n) \notin Grd(m)[Progress(m)])) \land$
$tr(0) \in AP(m) \land$
$(\forall i, j \cdot i \in dom(tr) \land j \in dom(tr) \land j = i + 1 \Rightarrow IsANextState(m, tr(i), tr(j)))$
...
END

Listing 5: Theory of Event-B Traces

The operator `IsATrace` captures the relation between machines and traces. A transition associated to an event in a trace is defined by the `IsANextState` operator. Considering a machine m and two states s and sp, the operator checks that there exists an event e such that: 1) s verifies the guard of e ($s \in Grd(m)[\{e\}]$), and 2) the pair $s \mapsto sp$ verifies the BAP of e ($s \mapsto sp \in BAP(m)[\{e\}]$).

5.2 Correctness Principle

Soundness properties can be expressed with the formalisation of the semantics using traces, in particular the correctness of the newly defined POs [36]. A generic principle can be stated as follows.

In Listing 6, each PO `[PO]` is associated with a `thm_of_Correctness_of_[PO]` soundness theorem in the `Theo4[PO]Correctness` theory. It states that the `[PO]` predicate definition (see Section 7) implies the PO predicate definition expressed on traces using the `PO_Spec_On_Traces` expression. Such theorems have been proved for each PO introduced in the EB4EB framework.

THEORY Theo4[PO]Correctness **IMPORT** EvtBTraces, Theo4[PO]
TYPE PARAMETERS St, Ev
THEOREMS
thm_of_Correctness_of_[PO] : $\forall m, tr \cdot m \in Machine(St, Ev) \land Machine\_WellCons(m) \land$
$IsATrace(tr, m) \land \ldots \land$ `[PO]`$\boldsymbol{(m, args)} \Rightarrow$ ***PO_Spec_On_Traces(...)***

Listing 6: Liveness Analyses Correctness

Example: Soundness of the Invariant PO (INV). The theorem of Listing 7 states that for any well-constructed machine m, if the invariant PO holds ($Mch\_INV(m)$) then for any trace tr associated to this machine ($IsATrace(tr, m)$), each state of that trace is in the invariant of the machine ($tr(i) \in Inv(m)$).

It has been proved, by induction on the indexes of the traces, using the Rodin platform provers. This principle is applied for all the newly introduced POs, in particular for the temporal logic properties POs introduced in this paper.

```
THEORY EvtBCorrectness IMPORT EvtBTraces, EvtBPO
TYPE PARAMETERS St, Ev
THEOREMS
  thm_of_Correctness_of_Invariant_PO: ∀m, tr · m ∈ Machine(St, Ev)∧
    Machine_WellCons(m) ∧ IsATrace(tr, m) ∧ Mch_INV(m)
      ⇒ (∀i · i ∈ dom(tr) ⇒ tr(i) ∈ Inv(m))
END
```

Listing 7: Theorem of correction of the proof obligation

6 A Case Study: A read write machine

In the original paper [18], the authors used the read-write case study to illustrate their approach. For comparison purposes, we use the same case study.

```
MACHINE RdWrMch
VARIABLES r, w
INVARIANTS
  inv1−2: r ∈ ℕ, w ∈ ℕ
  inv3−4: 0 ≤ w − r, w − r ≤ 3
EVENTS
  INITIALISATION
  THEN
    act1: r, w := 0, 0
  END

  read
  WHERE grd1: r < w
  THEN act1: r := r + 1
  END

  write
  WHERE grd1: w < r + 3
  THEN act1: w := w + 1
  END

END
```

(a)

```
CONTEXT RdWr
SETS Ev
CONSTANTS rdwr, init, read, write
AXIOMS
  axm1: partition(Ev, {init}, {read}, {write})
  axm2: rdwr ∈ Machine(ℤ × ℤ, Ev)
  axm3: Event(rdwr) = Ev
  axm5: State(rdwr) = ℤ × ℤ
  axm6: Init(rdwr) = init
  axm7: Inv(rdwr) = {r ↦ w | r ∈ ℕ ∧ w ∈ ℕ∧
                0 ≤ w − r ∧ w − r ≤ 3}
  axm8: AP(rdwr) = {0 ↦ 0}
  axm9: BAP(rdwr) = {e ↦ (
                (r ↦ w) ↦ (rp ↦ wp)) |
            (e = read ∧ rp = r + 1 ∧ wp = w)
          ∨(e = write ∧ rp = r ∧ wp = w + 1)}
  axm10: Grd(rdwr) = {e ↦ (r ↦ w) |
                (e = read ∧ r < w)∨
                (e = write ∧ w < r + 3)}
  axm11: Progress(rdwr) = {read, write}
  ...
  thm1: check_Machine_Consistency(rdwr)
END
```

(b)

Listing 8: Read write machine in Event-B (a) and instantiation with EB4EB (b)

The system requirements are: **Req1** – The reader process reads data from the buffer; **Req2** – The writer process writes data to the buffer; **Req3** – The reader and the writer share the same buffer; **Req4** – The shared buffer has a fixed size of 3; **Req5** – The system does not stop when data is written and not read; and **Req6** – The reader eventually reads L, $L \in \mathbb{N}$, pieces of data.

Listing 8.a proposes the `RdWrMch` Event-B machine fulfilling the above requirements. The reader (resp. writer) is modelled by variable r (resp. w) corresponding to its position in the buffer and by event *read* (resp. *write*) that represents the associated input/output operation and increments the pointer (**Req1** and **Req2**). The shared buffer is captured by interval $r + 1..w$ (**Req3**). The correct formalisation of the events, i.e. data that has not been written yet is not read and the amount of data in the buffer does not exceed 3 (**Req4**), is guaranteed by invariants *inv3-4*. Listing 8.b shows the context obtained when instantiating the `EvtBTheo` theory (Listing 1) of the EB4EB framework. The `thm1` theorem guarantees the consistency of the `RdWrMch` Event-B machine.

Missing requirements. Req5 and **Req6** are not *safety properties* in the usual sense and are not present in the current model. Event-B does not natively provide explicit constructs for handling them. Additional modelling effort is necessary, like introducing variants and new theorems and altering events.

7 Temporal logic proof rules as EB4EB POs

To support temporal logic properties and handle the missing requirements, we propose an Event-B extension relying on the EB4EB framework. This section presents the formalisation of the liveness properties, introduced in [18], that are missing in core Event-B. For this purpose, we extend the EB4EB framework to introduce the corresponding PO definitions. All the definitions are formalised in the `Theo4Liveness` theory (see Listings 9) extending the `EvtBTheo` theory of EB4EB using a set of operators, defined for each proof rule defined in [18]. Each of these definitions is introduced below. Note that each of the following tables contain two parts, where (a) is from [18] and (b) our corresponding formalization.

```
THEORY Theo4Liveness
IMPORT EvtBTheo
TYPE PARAMETERS Ev , St
...
```

Listing 9: Liveness operators Theory

Notations. For a predicate P on states of St, we define the subset $\hat{P}$ of states satisfying the property P as $\hat{P} = \{x \in St \mid P(x)\}$.

7.1 Liveness properties

This section presents core definitions for expressing formal definition of liveness properties. We first describe the basic building operators.

Machine M Leads From P_1 to P_2, $P_1 \curvearrowright P_2$ (`TLLeads_From_P1_To_P2` operator). For a machine M, given two state formulas P_1 and P_2, we state that M leads from P_1 to P_2 if for every trace of M with two successor states such that $s_i \in \hat{P_1}$ then $s_{i+1} \in \hat{P_2}$. The given property of Table 3(a) is formally defined by the operator `TLLeads_From_P1_To_P2` with a machine m and two set of states $\hat{P_1}$ and $\hat{P_2}$ as parameters. Its direct definition is a predicate $BAP(m)[\{e\}][\hat{P_1} \cap Grd(m)[\{e\}] \cap Inv(m)] \subseteq \hat{P_2}$ stating that for all progress events of machine m that preserve invariant, states of $\hat{P_1}$ lead to $\hat{P_2}$.

The Sequent Rule for $\curvearrowright$	Associated Operator in EB4EB
$P_1 \curvearrowright P_2 \equiv \forall v, v', x \cdot$ $P_1(v) \wedge G(x,v) \wedge A(x,v,v') \Rightarrow P_2(v')$	**TLLeads_From_P1_To_P2** <*predicate*> $(m : Machine(St, Ev),\ \hat{P_1} : \mathbb{P}(St), \hat{P_2} : \mathbb{P}(St))$ **direct definition** $\forall e \cdot e \in Progress(m) \Rightarrow$ $BAP(m)[\{e\}][\hat{P_1} \cap Grd(m)[\{e\}] \cap Inv(m)] \subseteq \hat{P_2}$
(a)	(b)

Table 3: Leads from P1 to P2 encoded in EB4EB

Machine M is Convergent in P, $\downarrow P$ (`TLConvergent_In_P` operator). For a given property P, a machine M is convergent in P if it does not allow for an infinite sequence of P-states (i.e. states satisfying the property P). It is formalised

in Table 4(a) by the predicate operator `TLConvergent_In_P` on machine m, set of states $\hat{P}$ and variant v. The operator's WD condition ensures that the variant is associated to each state. The operator states that, for all progress events e, when its before-after-states s and s' satisfy P, variant v decreases ($v(s') < v(s)$).

The Sequent Rule of $\downarrow$	Associated Operator in EB4EB
$\downarrow P \equiv \forall x, v, v' \cdot$ $(P(v) \land G(x,v) \Rightarrow V(v) \in \mathbb{N}) \land$ $(P(v) \land G(x,v) \land A(x,v,v') \Rightarrow V(v') < V(v))$	**TLConvergent_In_P** <*predicate*> $(m : Machine(St, Ev),\ \hat{P} : \mathbb{P}(St), v : \mathbb{P}(St \times \mathbb{Z}))$ **well—definedness** $v \in St \to \mathbb{Z}$ **direct definition** $\forall e \cdot e \in Progress(m) \Rightarrow ($ $v[\hat{P} \cap Grd(m)[\{e\}] \cap Inv(m)] \subseteq \mathbb{N} \land$ $(\forall s, s' \cdot s \in Inv(m) \land s \in \hat{P} \land$ $s \in Grd(m)[\{e\}] \land s' \in BAP(m)[\{e\}][\{s\}]$ $\Rightarrow v(s') < v(s)))$
(a)	(b)

Table 4: Convergence in P encoded in EB4EB

Machine M is Divergent in P, $\nearrow P$ (`TLDivergent_In_P` operator). Divergence property guarantees that any infinite trace of a machine M ends with an infinite sequence of P-states. The operator `TLDivergent_In_P` of Table 5(a) is identical to the previous convergent operator, except that the variant does not decrease *strictly* ($v(s') \leq v(s)$) allowing divergent sequences of P-states.

The Sequent Rule of $\nearrow$	Associated Operator in EB4EB
$\nearrow P \equiv \forall x, v, v' \cdot$ $(\neg P(v) \land G(x,v) \Rightarrow V(v) \in \mathbb{N}) \land$ $(\neg P(v) \land G(x,v) \land A(x,v,v') \Rightarrow$ $V(v') < V(v)) \land$ $(P(v) \land G(x,v) \land A(x,v,v') \land V(v') \in \mathbb{N} \Rightarrow$ $V(v') \leq V(v))$	**TLDivergent_In_P** <*predicate*> $(m : Machine(St, Ev),\ \hat{P} : \mathbb{P}(St), v : \mathbb{P}(St \times \mathbb{Z}))$ **well—definedness** $v \in St \to \mathbb{Z}$ **direct definition** $TLConvergent\_In\_P(m, St \setminus \hat{P}, v) \land$ $\forall e \cdot e \in Progress(m) \Rightarrow ($ $(\forall s, s' \cdot s \in Inv(m) \land s \in \hat{P}\ \land s \in Grd(m)[\{e\}]$ $\land\ s' \in BAP(m)[\{e\}][\{s\}] \land v(s') \in \mathbb{N}$ $\Rightarrow v(s') \leq v(s)))$
(a)	(b)

Table 5: Divergence in P encoded in EB4EB

Machine M is Deadlock-free in P, $\circlearrowleft P$ (`TLDeadlock_Free_In_P` operator). The deadlock-freeness states that a trace of a machine M never reaches a P-state where no event is enabled. It requires that, in a P-state, at least one event of M is enabled. This property is defined in Table 6(a) and is formalised by the operator `TLDeadlock_Free_In_P` in Table 6(b).

The expression $\hat{P} \cap Inv(m) \subseteq Grd(m)[\ Progress(m)]$ ensures that at least one progress event of the $Progress(m)$ set is enabled in a P-state satisfying the invariant.

The Sequent Rule of $\circlearrowleft$	Associated Operator in EB4EB
$\circlearrowleft P \equiv \forall v \cdot P(v) \Rightarrow \bigvee_i (\exists x \cdot G_i(x,v))$	**TLDeadlock_Free_In_P** <*predicate*> $(m : Machine(St, Ev), \hat{P} : \mathbb{P}(St))$ **direct definition** $\hat{P} \cap Inv(m) \subseteq Grd(m)[Progress(m)]$
(a)	(b)

Table 6: Deadlock-freeness in P encoded in EB4EB

7.2 *Deadlock freeness* $\circlearrowleft$ *P* applied to the Read-Write machine

We illustrate how the operators defined above work in the extended EB4EB framework on the read write case study, with the case of the deadlock-freeness property ensuring requirement **Req5**.

```
CONTEXT RdWrDeadlockFree
EXTENDS RdWr
THEOREMS
  thmDeadlockFreeInP :
    TLDeadlock_Free_In_P(rdwr ,
      {r ↦ w | w ∈ ℤ ∧ r ∈ ℤ ∧ r < w})
END
```

Listing 10: Generation of Proof Obligation of Deadlock_Free_In_ P

A context `RdWrDeadlockFree`, extending the context `RdWr` of Listing 8 is defined with a theorem, `thmDeadlockFreeInP`. This theorem uses the predicate operator `Deadlock_Free_In_P`, previously formalised. Here, the $\hat{P}$ parameter is composed of the pair of state variables $r \mapsto w$ and the property P defined by $w \in \mathbb{Z} \land r \in \mathbb{Z} \land r < w$. Indeed, the machine does not deadlock if it reads less data than it writes. Remember that when a theorem is stated, a PO is automatically generated requiring to prove it.

7.3 Temporal operator proof rules

Section 7.1 presents a formalisation of the basic temporal operators allowing to define liveness properties. This section is devoted to the formalisation of more complex temporal properties, relying on the operators previously defined, like `TLGlobally`, `TLExistence` `TLUntil`, `TLProgress`, and `TLPersistence`. Each of them is defined in the same manner as the previous ones.

Invariance, $\square I$ (`TLGlobally` operator). In Event-B, safety properties are commonly described as invariants. Although this property is already available in core Event-B, it can be formalised in EB4EB as well.

Table 7(a) expresses this property using two sequents. The first one is the inductive invariant proof rule and the second one defines, as theorems, all of the entailed stronger invariants. The `TLGlobally` operator of Table 7(b) defines this property as $Inv(m) \subseteq \hat{I}$; it reuses the native invariant PO of EB4EB.

The Sequent Rule of $\square$	Associated Operator in EB4EB
$\dfrac{\vdash\ init \Rightarrow I \qquad M\ \vdash\ I \rightsquigarrow I}{M\ \vdash\ \square I}$ $\dfrac{\vdash\ J \Rightarrow I \qquad M\ \vdash\ \square J}{M\ \vdash\ \square I}$	**TLGlobally** <*predicate*> $(m : Machine(St, Ev),\ \hat{I} : \mathbb{P}(St))$ **direct definition** $Inv(m) \subseteq \hat{I}$
(a)	(b)

Table 7: Invariance encoded in EB4EB

Existence, $\square\Diamond P$ (`TLExistence` operator). The existence temporal property states that a property P *always eventually* holds for machine M. To express existence $\square\Diamond P$ in a machine M, we rely on convergence and deadlock-freeness. Indeed, the machine shall be convergent on $\neg P$-states, i.e., sometimes $\neg P$ does not hold and $\neg P$-states are not deadlocks. The defined `TLExistence` predicate operator is defined as the conjunction of the two corresponding previously defined operators on a set $\hat{P}$ and a variant v.

The Sequent Rule of $\Box\Diamond$	Associated Operator in EB4EB
$\dfrac{M \vdash \downarrow \neg P \qquad M \vdash \circlearrowleft \neg P}{M \vdash \Box\Diamond P}$	**TLExistence** <*predicate*> $(m : Machine(St, Ev),\ \hat{P} : \mathbb{P}(St), v : \mathbb{P}(St \times \mathbb{Z})$ **well—definedness** $v \in St \rightarrow \mathbb{Z}$ **direct definition** $TLConvergent\_In\_P(m, St \setminus \hat{P}, v) \wedge$ $TLDeadlock\_Free\_In\_P(m, St \setminus \hat{P})$
(a)	(b)

Table 8: Existence encoded in EB4EB

Until, $\Box(P_1 \Rightarrow (P_1 \mathcal{U} P_2))$ (`TLUntil` operator). The *Until* property states that a P_1-state is *always followed eventually* by a P_2-state. Its definition relies on the *leads-to* and *existence* properties we have introduced. The *Until* property requires two antecedents, a leads to from $P_1 \wedge \neg P_2$ to $P_1 \vee P_2$ in the next state and the second is the existence of $\neg P_1 \vee P_2$ (see Table 9(a)). This proof rule is directly formalises using the `TLUntil` operator (see Table 9(b)). It requires two properties P_1 ($\hat{P_1}$ set) and P_2 ($\hat{P_2}$ set) and a variant v. It is defined as the conjunction of the `TLLeads_From_P1_To_P2` and `TLExistence` predicate operators.

The Sequent Rule of $\Box(P_1 \Rightarrow (P_1 \mathcal{U} P_2))$	Associated Operator in EB4EB
$A \equiv (P_1 \wedge \neg P_2) \rightsquigarrow (P_1 \vee P_2)$ $B \equiv \Box\Diamond(\neg P_1 \vee P_2)$ $\dfrac{M \vdash A \qquad M \vdash B}{M \vdash \Box(P_1 \Rightarrow (P_1 \mathcal{U} P_2))}$	**TLUntil**<*predicate*> $(m : Machine(St, Ev),$ $\hat{P_1} : \mathbb{P}(St), \hat{P_2} : \mathbb{P}(St), v : \mathbb{P}(St \times \mathbb{Z})$ **well—definedness** $v \in St \rightarrow \mathbb{Z}$ **direct definition** $Leads\_From\_P1\_To\_P2($ $m, \hat{P_1} \cap (St \setminus \hat{P_2}), \hat{P_1} \cup \hat{P_2})$ $\wedge\ TLExistence(m, (St \setminus \hat{P_1}) \cup \hat{P_2}, v)$
(a)	(b)

Table 9: Until encoded in EB4EB

Progress, $\Box(P_1 \Rightarrow (\Diamond P_2))$ (`TLProgress` operator). Close to the *Until* property, a more general property, namely *Progress* can be defined. It states that always P_1-states reaches $P2$-states. This property does not require P_1 to always hold before reaching P_2-states. To describe this property, an intermediate property P_3 holding before P_2 holds is introduced. It acts as a local invariant between P_1-states and P_2-states.

The Sequent Rule of $\Box(P_1 \Rightarrow \Diamond P_2)$	Associated Operator in EB4EB
$A \equiv \Box(P_1 \wedge \neg P_2 \Rightarrow P_3)$ $B \equiv \Box(P_3 \Rightarrow (P_3 \mathcal{U} P_2))$ $\dfrac{M \vdash A \qquad M \vdash B}{M \vdash \Box(P_1 \Rightarrow (\Diamond P_2))}$	**TLProgress**<*predicate*> $(m : Machine(St, Ev),$ $\hat{P_1} : \mathbb{P}(St), \hat{P_2} : \mathbb{P}(St), \hat{P_3} : \mathbb{P}(St), v : \mathbb{P}(St \times \mathbb{Z})$ **well—definedness** $v \in St \rightarrow \mathbb{Z}$ **direct definition** $TLGlobally(m, \hat{P_3} \cup \hat{P_2} \cup (St \setminus \hat{P_1})) \wedge$ $TLUntil(m, variant, \hat{P_3}, \hat{P_2})$
(a)	(b)

Table 10: Progress encoded in EB4EB

The *Progress* proof rule of Table 10(a) has two antecedents. One states that always $P_1 \wedge \neg P_2 \Rightarrow P_3$ and the second uses the previously defined *Until* property as $\Box(P_3 \Rightarrow (P_3 \mathcal{U} P_2))$. The `TLProgress` predicate operator is the conjunction of the application of the two predicate operators, `Leads_From_P1_To_P2` and `TLUntil` on the $\hat{P_1}$, $\hat{P_2}$ and $\hat{P_3}$ sets and the variant v, encoding the antecedents.

Persistence, $\Diamond\Box P$ (`TLPersistence` operator). *Persistence* is the last property we formalise. It states that a predicate P must eventually hold forever ($\Diamond\Box P$). The two antecedents of the associated proof rule, presented in Table 11(a), state that P-states are divergent $\neg P$-states are deadlock-free. The `TLPersistence` predicate operator is defined as a conjunctive expression of `TLDivergent_In_P` and `TLDeadlock_Free_In_P` operators with the $\hat{P}$ for the property P and the variant v as input parameters.

The Sequent Rule of $\Diamond\Box$	Associated Operator in EB4EB
$\dfrac{M \vdash \nearrow P \qquad M \vdash \circlearrowleft \neg P}{M \vdash \Diamond\Box P}$	**TLPersistence** <*predicate*> $(m : Machine(St, Ev), \hat{P} : \mathbb{P}(St), v : \mathbb{P}(St \times \mathbb{Z})$ **well−definedness** $v \in St \to \mathbb{Z}$ **direct definition** $TLDivergent\_In\_P(m, \hat{P}, variant) \wedge$ $TLDeadlock\_Free\_In\_P(m, St \setminus \hat{P})$
(a)	(b)

Table 11: Persistence encoded in EB4EB

7.4 *Existence* $\Box\Diamond P$ applied to the read write machine

The temporal operators defined in [18] have been successfully formalised in the EB4EB as predicate operators used as theorems to be proved for any Event-B machine.

Here, we show how **Req6** (the reader eventually reads L, $L \in \mathbb{N}$, pieces of data) expressed for the read write case study is fulfilled thanks to the `TLExistence` operator. Like for deadlock freeness in section 7.2, we introduce a new Event-B context `RdWrExistence` (see Listing 11), extending the `RdWr` context of Listing 8, with a theorem stating the existence property. The existence operator is used with a set of states $\{r \mapsto w \mid w \in \mathbb{Z} \wedge r \geq L\}$ and a variant $v = ((L - r) + (L + 3 - w))\}$).

```
CONTEXT RdWrExistence
EXTENDS RdWrDeadlockFree
CONSTANTS L
AXIOMS
  axm1: L ∈ ℕ
  thmExistence: TLExistence(
      rdwr, {r ↦ w | w ∈ ℤ ∧ r ≥ L},
      {(r ↦ w) ↦ v |
          v = ((L − r) + (L + 3 − w))})
END
```

Listing 11: Generation of Proof Obligation of Existence

8 Correctness of the temporal logic properties proof rules

The last step establishes the correctness of our formalisation with respect to the semantics of trace, i.e. the defined proof rules actually hold on the traces of the Event-B machines. The verification principle of Section 5.2 is set up for this purpose. A theory `Theo4LivenessCorrectness` (Listing 12) provides a list of correctness theorems for each of the defined operators. It imports the previously developed theories related to liveness properties `Theo4Liveness` and Event-B traces `EvtBTraces`.

```
THEORY Theo4LivenessCorrectness
  IMPORT Theo4Liveness, EvtBTraces
  TYPE PARAMETERS St, Ev
  ...
```

Listing 12: Theory of correctness

Below, we present the correctness theorem for the `TLExistence` property. All the other theorems are formalised[1] and proved using the Rodin Platform.

[1] `https://www.irit.fr/~Peter.Riviere/models/`

Existence in P correctness theorem $\Box\Diamond P$ (`TLExistence`). The correctness of the existence property follows the principle of Section 5.2. It is supported by the proved `thm_of_correctness_of_Existence` theorem stating that a property P always eventually holds in traces of a machine m. It states that for any well constructed ($Machine\_WellCons(m)$) and consistent ($check\_Machine\_Consistency(m)$) machine, and for any trace tr of this machine satisfying the existence property $TLExistence(m, \hat{P}, variant)$, then for all i there exists j with $j \geq i$ where $tr(j)$ satisfies the property P.

THEOREMS
$thm\_of\_Correctness\_of\_Existence$: $\forall m, tr, v, \hat{P} \cdot v \in STATE \to \mathbb{Z} \wedge$
$m \in Machine(STATE, EVENT) \wedge Machine\_WellCons(m) \wedge$
$check\_Machine\_Consistency(m) \wedge IsATrace(m, tr) \wedge \mathbf{TLExistence(m, \hat{P}, v)}$
$\Rightarrow (\forall i \cdot i \in dom(tr) \Rightarrow (\exists j \cdot j \geq i \wedge j \in dom(tr) \wedge tr(j) \in \hat{P}))$
...

Listing 13: Theorem of correctness of the operators Existence

9 Related Work

Reflexive modelling is present under various forms in formal methods. For instance, the *ASM-Metamodel* API (AsmM) for Abstract State Machines (ASM) has been developed to be able to handle ASM-related concepts. This leads to several extensions, analyses and tools for ASMs [34]. This is also the case when using Mural to modify a VDM specification [4]. Furthermore, the reflexive modelling is also addressed with proof assistants like Coq with MetaCoq [39], Agda [32], PVS [29], HOL [13] and Lean [12] and Event-B with EB4EB [35,36].

Correctness of the Event-B method and its modelling components has been tackled in various previous work. A meta-level study of Event-B context structure is proposed in particular to validate the expected properties of theorem instantiation [5]. Event-B has also been formalised as an *institution* in category theory [15,14], with the aim to facilitate and enable composition of heterogeneous semantics and of different model specifications. Similarly, Event-B has been embedded in Coq [10] in order to establish the correctness of refinement, i.e. that the refinement POs entail the validity of refinement in the trace-based semantics. Last, a form of shallow embedding of Event-B in itself has been proposed and serves as the basis of a methodology for proving the correctness of decomposition and re-composition of Event-B machines [17].

Event-B's methodology is mainly aimed at defining and proving safety properties (that must always hold), or possible convergence. Expressing liveness properties (that must hold at some point [22]) is not as trivial, and many authors address this issue. For Event-B, the ProB model-checker [25] handles Event-B models and enables the expression and verification of liveness properties. Some liveness operators have been formalised to be used in Event-B, together with their related hypotheses [18], making it possible to express *some* liveness properties. However, it is to be noted that liveness properties are not generally preserved by refinement. To address this latter issue, additional conditions on the refinement must be posed, leading to the definition of particular refinement strategies [19],

which are proven to preserve liveness properties through to the concrete model. In addition, the problem of *fairness* has also been studied. For instance, the work of [28] proposes to check fairness of Event-B machines in TLA (on a per-machine basis). Refinement strategies have been defined as well to ensure that fairness and liveness properties are preserved [42].

Our proposed approach is based on the reflexive modelling of Event-B on itself, which is fully integrated into Rodin development environment using the Theory Plugin [9]. Our framework is fully formalised in Event-B and relies solely on FOL and set theory, similar to other approach like MetaCoq [39] with dependent type. Such characteristic makes it possible to *export* models expressed using the framework to any other formalism based on FOL and set theory while preserving the state-transition semantics of the model. Therefore, the issue of the translation of the universe and the semantics' preservation are not related to our work due to the reflexive modelling.

10 Conclusion

This paper has presented a formalisation of liveness properties for Event-B models by encoding LTL temporal logic expressions on the Rodin platform using the reflexive EB4EB framework. LTL logic expressions of properties are formalised within the defined framework. Automatic generation of proof obligations related to the expressed properties and the soundness of the defined proof rules using a trace based semantics have been addressed as well. The proposed approach relies on the definition of algebraic theories offering the capability to define new operators. The read write machine case study was borrowed from [18] to illustrate our approach. Other case studies have been developed as well (Peterson algorithm [37] and behavioural analyses in human computer interaction [27]).

The proposed framework supports non-intrusive analysis for Event-B models, allowing liveness properties to be expressed and verified on any size Event-B formal model and at any refinement level without resorting to any other formal methods. Since our framework allows checking temporal properties at any refinement level, it avoids dealing with the preservation of temporal properties by refinement. Furthermore, the proof process has been enhanced with relevant and proven rewrite rules, which have been incorporated into Rodin tactics, resulting in a high level of proof automation. All the developments illustrated in this paper have been fully formalised and proved using the Rodin platform. They can be accessed on `https://www.irit.fr/~Peter.Riviere/models/`

This work leads to several perspectives. First, we plan to study the capability to allow compositional definitions of LTL properties relying on the defined basic operators. In addition, the proposed approach makes it possible to define other Event-B model analyses or domain specific theories shared by many Event-B models. Last, we believe that our approach can be scaled up to other state based methods provided that a reflexive meta-model is available.

References

1. Abrial, J.R.: Modeling in Event-B: System and software engineering. Cambridge University Press (2010)
2. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal, pp. 200–236. Springer (2004)
3. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series, Springer (2004)
4. Bicarregui, J., Ritchie, B.: Reasoning about VDM developments using the VDM support tool in MURAL. In: VDM '91 - Formal Software Development, 4th International Symposium of VDM Europe, Noordwijkerhout. LNCS, vol. 551, pp. 371–388. Springer (1991)
5. Bodeveix, J., Filali, M.: Event-B Formalization of Event-B Contexts. In: International Conference on Rigorous State-Based Methods (ABZ). LNCS, vol. 12709, pp. 66–80. Springer (2021)
6. Bodeveix, J., Filali, M., Garnacho, M., Spadotti, R., Yang, Z.: Towards a verified transformation from AADL to the formal component-based language FIACRE. Elsevier SCP **106**, 30–53 (2015)
7. Butler, M.J., Dghaym, D., Fischer, T., Hoang, T.S., Reichl, K., Snook, C.F., Tummeltshammer, P.: Formal modelling techniques for efficient development of railway control products. In: International Conference on Reliability, Safety, and Security of Railway Systems (RSSRail). LNCS, vol. 10598, pp. 71–86. Springer (2017)
8. Butler, M.J., Körner, P., Krings, S., Lecomte, T., Leuschel, M., Mejia, L., Voisin, L.: The first twenty-five years of industrial use of the b-method. In: International Conference on Formal Methods for Industrial Critical Systems (FMICS). LNCS, vol. 12327, pp. 189–209. Springer (2020)
9. 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. LNCS, vol. 8051, pp. 67–81. Springer (2013)
10. Castéran, P.: An Explicit Semantics for Event-B Refinements, pp. 155–173. Springer (2021)
11. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource tool for symbolic model checking. In: International Conference on Computer Aided Verification (CAV). p. 359–364. Springer (2002)
12. Ebner, G., Ullrich, S., Roesch, J., Avigad, J., de Moura, L.: A metaprogramming framework for formal verification. ACM PACMPL **1**(ICFP), 34:1–34:29 (2017)
13. Fallenstein, B., Kumar, R.: Proof-producing reflection for HOL - With an application to model polymorphism. In: International Conference on Interactive Theorem Proving (ITP). LNCS, vol. 9236, pp. 170–186. Springer (2015)
14. Farrell, M., Monahan, R., Power, J.F.: An institution for Event-B. In: Recent Trends in Algebraic Development Techniques (IFIP) WG 1.3 International Workshop (WADT). LNCS, vol. 10644, pp. 104–119. Springer (2016)
15. Farrell, M., Monahan, R., Power, J.F.: Building specifications in the Event-B institution. Springer LMCS **18**(4) (Nov 2022)
16. Halchin, A., Ameur, Y.A., Singh, N.K., Ordioni, J., Feliachi, A.: Handling B models in the PERF integrated verification framework: Formalised and certified embedding. Science of Computer Programming, Elsevier **196**, 102477 (2020)

17. Hallerstede, S., Hoang, T.S.: Refinement of decomposed models by interface instantiation. Elsevier SCP **94**, 144–163 (2014)
18. Hoang, T.S., Abrial, J.: Reasoning about liveness properties in event-b. In: International Conference on Formal Engineering Methods, ICFEM. LNCS, vol. 6991, pp. 456–471. Springer (2011)
19. Hoang, T.S., Schneider, S.A., Treharne, H., Williams, D.M.: Foundations for using linear temporal logic in Event-B refinement. Springer FAOC **28**(6), 909–935 (2016)
20. Holzmann, G.: Spin Model Checker, the: Primer and Reference Manual. Addison-Wesley Professional, first edn. (2003)
21. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) International Conference on Computer Aided Verification (CAV). LNCS, vol. 6806, pp. 585–591. Springer (2011)
22. Lamport, L.: Proving the correctness of multiprocess programs. IEEE TSE **3**(2), 125–143 (1977)
23. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc. (2002)
24. Leroy, X., Blazy, S., Kästner, D., Schommer, B., Pister, M., Ferdinand, C.: CompCert - A formally verified optimizing compiler. In: Embedded Real Time Software and Systems (ERTS). SEE (2016)
25. Leuschel, M., Butler, M.J.: ProB: an automated analysis toolset for the B method. Springer International Journal STTT **10**(2), 185–203 (2008)
26. Manna, Z., Pnueli, A.: Adequate proof principles for invariance and liveness properties of concurrent programs. Elsevier SCP **4**(3), 257–289 (1984)
27. Mendil, I., Riviere, P., Ameur, Y.A., 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)
28. Méry, D., Poppleton, M.: Towards an integrated formal method for verification of liveness properties in distributed systems: with application to population protocols. SoSyM **16**(4), 1083–1115 (2017)
29. Mitra, S., Archer, M.: PVS strategies for proving abstraction properties of automata. In: International Workshop on Strategies in Automated Deduction. ENTCS, vol. 125, pp. 45–65. Elsevier (2004)
30. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
31. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: International Conference on Automated Deduction (CADE). LNCS, vol. 607, pp. 748–752. Springer (1992)
32. Paul van der Walt: Reflection in Agda. Master's thesis, University of Utrecht, Department of Computing Science (2012)
33. Pnueli, A., Siegel, M., Singerman, E.: Translation validation. In: International Conference on Tools and Algorithms for Construction and Analysis of Systems(TACAS). LNCS, vol. 1384, pp. 151–166. Springer (1998)
34. Riccobene, E., Scandurra, P.: Towards an interchange language for ASMs. In: International Workshop on Abstract State Machines, Advances in Theory and Practice (ASM). LNCS, vol. 3052, pp. 111–126. Springer (2004)
35. 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)

36. 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)
37. Riviere, P., Singh, N.K., Aït Ameur, Y., Dupont, G.: Standalone Event-B models analysis relying on the EB4EB meta-theory. In: International Conference on Rigorous State Based Methods, ABZ 2023. LNCS, Springer Verlag (2023)
38. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer (2013)
39. Sozeau, M., Anand, A., Boulier, S., Cohen, C., Forster, Y., Kunze, F., Malecha, G., Tabareau, N., Winterhalter, T.: The MetaCoq project. Springer Journal of Automated Reasoning **64**(5), 947–999 (2020)
40. Su, W., Abrial, J.: Aircraft landing gear system: approaches with Event-B to the modeling of an industrial system. Springer International Journal STTT **19**(2), 141–166 (2017)
41. Sun, J., Liu, Y., Dong, J.S., Pang, J.: Pat: Towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) International Conference on Computer Aided Verification (CAV). pp. 709–714. Springer (2009)
42. Zhu, C., Butler, M., Cirstea, C., Hoang, T.S.: A fairness-based refinement strategy to transform liveness properties in Event-B models. Elsevier SCP **225**, 102907 (2023)