An Institution for Event-B

Marie Farrell, Rosemary Monahan, James Power

An Institution for Event-B

Marie Farrell*, Rosemary Monahan, and James F. Power

Dept. of Computer Science, Maynooth University, Co. Kildare, Ireland

Abstract. This paper presents a formalisation of the Event-B formal specification language in terms of the theory of institutions. The main objective of this paper is to provide: (1) a mathematically sound semantics and (2) modularisation constructs for Event-B using the specification-building operations of the theory of institutions. Many formalisms have been improved in this way and our aim is thus to define an appropriate institution for Event-B, which we call $\mathcal{EVT}$. We provide a definition of $\mathcal{EVT}$ and the proof of its satisfaction condition. A motivating example of a traffic-light simulation is presented to illustrate our approach.

Keywords: Event-B; institutions; refinement; formal methods; modular specification; formal specification

1 Introduction and Motivation

Event-B is an industrial-strength, state-based formalism for system-level modelling and verification, combining set theoretic notation with event-driven modelling. However, Event-B lacks well-developed modularisation constructs and it is not easy to combine specifications in Event-B with those written in other formalisms [6]. Our thesis, presented in this paper, is that the theory of institutions can provide a framework for defining a rich set of modularisation operations and promoting interoperability and heterogeneity for Event-B.

This paper is centered around an illustrative example of a specification written in Event-B, inspired by one in the *Rodin Handbook* [7], which we present in the remainder of Section 1. We define our institution for Event-B, called $\mathcal{EVT}$, in Section 2, prove that it is a valid institution, and define a comorphism between the institution for first-order predicate logic with equality and $\mathcal{EVT}$ in Section 3. In Section 4 we use this institution to recast our Event-B example in modular form using specification-building operators and address refinement, since this is of central importance in Event-B. We summarise our contributions and outline future directions in Section 5.

1.1 Formal Specification of a Traffic-Lights System in Event-B

Figure 1 presents an Event-B machine for a traffic-lights system with one light signalling cars and one signalling pedestrians [2]. The goal of the specification

* This work is funded by Government of Ireland Postgraduate Grant from the Irish Research Council.

is to ensure that it is never the case that both cars and pedestrians receive the "go" signal at the same time (represented by boolean flags on line 2). Machine specifications typically contain variable declarations (line 2), a variant expression (none in this example), invariants (lines 3–6) and event specifications (lines 7–21). *Contexts* in Event-B can be used to model the static properties of a system (constants, axioms and carrier sets). Figure 2 provides a context giving a specification for the data-type $COLOURS$. The axiom on line 5 explicitly restricts the set to only contain the constants red, $green$ and $orange$.

Figure 1 specifies five different events (including a starting event called `Initialisation` defined on lines 8–10). Each event has a guard, specifying when it can be activated, and an action, specifying what happens when the event is activated. For example, the `set_peds_go` event as specified on lines 11–13, has one guard expressed as a boolean expression (line 12), and one action, expressed as an assignment statement (line 13). Moreover, each event has a status, which can be either `ordinary`, `convergent`, or `anticipated`. If the status is different from `ordinary`, then the event is concerned with the variant expression, i.e. with a natural-number expression used in proving termination properties. Our example has no variant so all events have the status `ordinary`.

Figure 3 shows an Event-B machine specification for `mac2` that refines the machine `mac1` (Figure 1). The machine `mac1` is refined by first introducing the new context on line 1 and then by replacing the truth values used in the abstract machine with new values from the carrier set $COLOURS$. This new data type is included into `mac2` using the `SEES` construct on line 1 of Figure 3. During refinement, the user typically supplies a *gluing invariant* relating properties of the abstract machine to their counterparts in the concrete machine [2]. The gluing invariants in Figure 3 (lines 6 and 8) define a one-to-one mapping between the concrete variables introduced in `mac2` and the abstract variables of `mac1`. The concrete variables ($peds\_colour$ and $cars\_colour$) can be assigned either red or $green$, thus the gluing invariants map $true$ to $green$ and $false$ to red.

Event-B permits the addition of new variables and events: `button_pushed` (line 2) and `press_button` (lines 30–31). The existing events from `mac1` are renamed to reflect refinement; for example, the event `set_peds_green` is declared to refine `set_peds_go` (lines 14–15). This event has also been altered via the addition of a guard (line 16) and an action (line 18) that incorporate the functionality of a button-controlled pedestrian light. This example highlights features of the Event-B language, but notice how the same specification has to be provided twice in Figure 1. The events `set_peds_go` and `set_peds_stop` are equivalent, modulo renaming of variables, to `set_cars_go` and `set_cars_stop`. Ideally, writing and proving the specification for these events should only be required once. Our approach addresses these issues as will be seen in Section 4.

1.2 Related Work: Institutions and Modularisation

Originally, Event-B was not equipped with any modularisation constructs. Because of this, several approaches have been suggested for modularising Event-B

```
MACHINE mac1
VARIABLES  cars_go, peds_go
INVARIANTS
   inv1: cars_go ∈ BOOL
   inv2: peds_go ∈ BOOL
   inv3: ¬ (peds_go = true
           ∧ cars_go = true)
EVENTS
  Initialisation ordinary
    then  act1: cars_go := false
          act2: peds_go := false
  Event set_peds_go ≙ ordinary
    when  grd1: cars_go = false
    then  act1: peds_go := true
  Event set_peds_stop ≙ ordinary
    then act1: peds_go := false
  Event set_cars_go ≙ ordinary
    when  grd1: peds_go = false
    then  act1: cars_go := true
  Event set_cars_stop ≙ ordinary
    then act1: cars_go := false
END
```

Fig. 1: Event-B machine specification for a traffic system.

```
CONTEXT ctx1
SETS  COLOURS
CONSTANTS  red, green, orange
AXIOMS
   axm1: partition(COLOURS,
                   {red}, {green}, {orange})
END
```

Fig. 2: Event-B context specification for the colours of a set of traffic–lights.

```
MACHINE mac2  refines mac1  SEES ctx1
VARIABLES  cars_colour, peds_colour,
           buttonpushed
INVARIANTS
   inv1: peds_colour ∈ {red, green}
   inv2: (peds_go = true)
                  ⇔ (peds_colour = green)
   inv3: cars_colour ∈ {red, green}
   inv4: (cars_go = true)
                  ⇔ (cars_colour = green)
   inv5: buttonpushed ∈ BOOL
EVENTS
  Initialisation ordinary
    then  act1: cars_colour := red
          act2: peds_colour := red
  Event set_peds_green ≙ ordinary
  refines  set_peds_go
    when  grd1: cars_colour = red
          grd2: buttonpushed = true
    then  act1: peds_colour := green
          act2: buttonpushed := false
  Event set_peds_red ≙ ordinary
  refines  set_peds_stop
    then  act1: peds_colour := red
  Event set_cars_green ≙ ordinary
  refines  set_cars_go
    when  grd1: peds_colour = red
    then  act1: cars_colour := green
  Event set_cars_red ≙ ordinary
  refines  set_cars_stop
    then  act1: cars_colour := red
  Event press_button ≙ ordinary
    then  act1: buttonpushed := true
END
```

Fig. 3: A refined Event-B machine specification for a traffic system.

specifications. Abrial first proposed two styles of decomposition based on identifying shared variables and shared events [3]. Elaborating these approaches, approximately 8 modularisation plugins have been developed for various versions of Rodin, each offering a different perspective on implementing modularisation. By defining an institution for the Event-B formalism, we can modularise Event-B specifications using specification-building operators [11], and thus provide an approach to developing modular specifications that is consistent with the state of the art in formal specification.

An attempt was previously made to provide an institution and corresponding morphisms for Event-B and UML [4]. However, the definitions of Event-B sentences and models were vague, making it difficult to evaluate their semantics in a meaningful way. Also, the models described resemble the set-theoretic foundations of B specifications, whereas here we concentrate on event-based models. Our presentation of an illustrative example in both Event-B and its modular institutional version is an important element of developing this work.

Our approach provides scope for the interoperability of Event-B and other formalisms via institution (co)morphisms. Those familiar with the institution for UML state machines, $\mathcal{UML}$, may notice that we have based the construction of

our institution for Event-B, $\mathcal{EVT}$, on $\mathcal{UML}$[8]. Both institutions describe state-based formalisms so, by keeping $\mathcal{UML}$ in mind during the development of $\mathcal{EVT}$, it will be possible to design meaningful translations between them in the future.

2 An Institution for Event-B

The theory of institutions, originally developed by Goguen and Burstall in a series of papers originating from their work on algebraic specification, provides a general framework for defining a logical system [5].

Definition 1 (**Institution**)**.** An **institution** $\mathcal{INS}$ for some given formalism will consist of definitions for:

Vocabulary: a category **Sign** whose objects are called signatures and whose arrows are called signature morphisms.

Syntax: a functor $\mathbf{Sen} : \mathbf{Sign} \to \mathbf{Set}$ giving a set $\mathbf{Sen}(\Sigma)$ of *Σ-sentences* for each signature Σ and a function $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ for each signature morphism $\sigma : \Sigma \to \Sigma'$.

Semantics: a functor $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ giving a category $\mathbf{Mod}(\Sigma)$ of *Σ-models* for each signature Σ and a functor $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ for each signature morphism $\sigma : \Sigma \to \Sigma'$.

Satisfaction: for every signature Σ, a satisfaction relation $\models_{\mathcal{INS},\Sigma}$ between Σ-models and Σ-sentences.

An institution must uphold the **satisfaction condition**: for any signature morphism $\sigma : \Sigma \to \Sigma'$ and translations $\mathbf{Mod}(\sigma)$ of models and $\mathbf{Sen}(\sigma)$ of sentences we have for any $\phi \in \mathbf{Sen}(\Sigma)$ and $M' \in |\mathbf{Mod}(\Sigma')|$.

$$M' \models_{\mathcal{INS},\Sigma'} \mathbf{Sen}(\sigma)(\phi) \quad \iff \quad \mathbf{Mod}(\sigma)(M') \models_{\mathcal{INS},\Sigma} \phi$$

There are two basic languages within the Event-B language. The first one is the Event-B mathematical language (propositional/predicate logic, set-theory and arithmetic) and the second is the Event-B modelling language [1]. To represent the latter, we propose a new custom solution; for the former, however, we can use $\mathcal{FOPEQ}$, the institution of first-order logic with equality. Thus, our institution for Event-B is built on $\mathcal{FOPEQ}$.

Definition 2 ($\mathcal{FOPEQ}$**-Signature**)**.** A **signature** in $\mathcal{FOPEQ}$, $\Sigma_{\mathcal{FOPEQ}} = \langle S, \Omega, \Pi \rangle$, is a tuple where S is a set of sort names, Ω is a set of operation names indexed by arity and sort, and Π is a set of predicate names indexed by arity.

Definition 3 ($\Sigma_{\mathcal{FOPEQ}}$**-Sentence**)**.** For any $\Sigma_{\mathcal{FOPEQ}} = \langle S, \Omega, \Pi \rangle$, $\Sigma_{\mathcal{FOPEQ}}$**-sentences** are closed first-order formulae built out of atomic formulae using $\wedge, \vee, \neg, \Rightarrow, \iff, \exists, \forall$. Atomic formulae are equalities between $\langle S, \Omega \rangle$-terms, predicate formulae of the form $p(t_1, \ldots, t_n)$ where $p \in \Pi$ and $t_1, \ldots, t_n$ are terms (with variables), and the logical constants `true` and `false`.

Definition 4 ($\Sigma_{\mathcal{FOPEQ}}$**-Model**)**.** Given a signature $\Sigma_{\mathcal{FOPEQ}} = \langle S, \Omega, \Pi \rangle$, a model over $\mathcal{FOPEQ}$ consists of a carrier set $|A|_s$ for each sort name $s \in S$, a

function $f_A : |A|_{s_1} \times \cdots \times |A|_{s_n} \to |A|_s$ for each operation name $f \in \Omega_{s_1 \ldots s_n, s}$ and a relation $p_A \subseteq |A|_{s_1} \times \cdots \times |A|_{s_n}$ for each predicate name $p \in \Pi_{s_1 \cdots s_n}$, where $s_1, \ldots, s_n$, and s are sort names.

The satisfaction relation in $\mathcal{FOPEQ}$ is the usual satisfaction of first-order sentences by first-order structures.

2.1 Defining $\mathcal{EVT}$

Definition 5 ($\mathcal{EVT}$**-Signature**)**.** A **signature** in $\mathcal{EVT}$ is a five-tuple $\Sigma_{\mathcal{EVT}} = \langle S, \Omega, \Pi, E, V \rangle$ where $\langle S, \Omega, \Pi \rangle$ is a standard $\mathcal{FOPEQ}$-signature as described above, E is a set of events, i.e. of pairs $\langle \textit{event name}, \textit{status} \rangle$ where status belongs to the poset $\{\texttt{ordinary} < \texttt{anticipated} < \texttt{convergent}\}$, and V is a set of sorted variables. We assume that every signature has an initial event, called `Init`, whose status is always `ordinary`.

Notation: We write Σ in place of $\Sigma_{\mathcal{EVT}}$ when describing a signature over our institution for Event-B. For signatures over other institutions than $\mathcal{EVT}$ we will use the subscript notation; e.g. a signature over $\mathcal{FOPEQ}$ is denoted by $\Sigma_{\mathcal{FOPEQ}}$. For a given signature Σ, we access its individual components using a dot-notation, e.g. $\Sigma.V$ for the set V in the tuple Σ.

Definition 6 ($\mathcal{EVT}$**-Signature Morphism**)**.** A **signature morphism** $\sigma : \Sigma \to \Sigma'$ is a five-tuple containing σ_S, σ_Ω, σ_Π, σ_E and σ_V. Here σ_S, σ_Ω, σ_Π are the mappings taken from the corresponding signature morphism in $\mathcal{FOPEQ}$.

- $\sigma_E : \Sigma.E \to \Sigma'.E$ is a function such that for any mapping $\sigma_E\langle e, st \rangle = \langle e', st' \rangle$ we have $st \leq st'$; in addition, σ_E preserves the initial event: in symbols, we have that $\sigma_E\langle \texttt{Init}, \texttt{ordinary} \rangle = \langle \texttt{Init}, \texttt{ordinary} \rangle$.
- $\sigma_V : \Sigma.V \to \Sigma'.V$ is a sort-preserving function on sets of variable names, working similarly to the sort-preserving mapping for constant symbols, σ_Ω.

Definition 7 ($\Sigma_{\mathcal{EVT}}$**-Sentence**)**.** A sentence over $\mathcal{EVT}$ is a pair $\langle e, \phi(\overline{x}, \overline{x}') \rangle$ where e is an event name in the domain of $\Sigma.E$ and $\phi(\overline{x}, \overline{x}')$ is an open $\mathcal{FOPEQ}$-formula over the variables $\overline{x}$ from $\Sigma.V$ and their primed versions $\overline{x}'$.

In the *Rodin Platform*, Event-B machines are presented (syntactically sugared) as can be seen below, where $I(\overline{x})$ represents the invariant over $\overline{x}$.

The variant expression, denoted by $n(\overline{x})$, is used for proving termination properties. Events that have a status of `anticipated` or `convergent` must not increase and strictly decrease the variant expression respectively. Events can have parameter(s) as given by $\overline{p}$. $G(\overline{x}, \overline{p})$ and $W(\overline{x}, \overline{p})$ represent the guard(s) and witness(es) respectively over the variables and parameter(s). Actions are interpreted as before-after predicates i.e. $x := x + 1$ is interpreted as $x' = x + 1$. Thus, $BA(\overline{x}, \overline{p}, \overline{x}')$ represents the action(s) over

```
MACHINE m SEES ctx  refines a
  VARIABLES x̄
  INVARIANTS I(x̄)
  VARIANT n(x̄)
  EVENTS
  Initialisation ordinary
    then  act-name: BA(x̄, x̄')
  Event e ≙ status
    any p̄
    when  guard-name: G(x̄, p̄)
    with  witness-name: W(x̄, p̄)
    then  act-name: BA(x̄, p̄, x̄')
   ⋮
END
```

the parameter(s) $\overline{p}$ and the sets of variables $\overline{x}$ and $\overline{x}'$.

Sentences written in the mathematical language (such as axioms) are interpreted as sentences over $\mathcal{FOPEQ}$. We can include these in specifications over $\mathcal{EVT}$ using the comorphism which will be defined in Section 3. We represent the Event-B event, variant and invariant sentences as sentences over $\mathcal{EVT}$.

For each Event-B invariant sentence $I(\overline{x})$ we form the open $\mathcal{FOPEQ}$-sentence $I(\overline{x}) \wedge I(\overline{x}')$. Since invariants must hold for all events in a machine, each invariant sentence is paired with each event name e for all $\langle e, s\rangle \in \Sigma.E$, where s is an event status. Thus, we form the $\mathcal{EVT}$ sentence $\langle e, I(\overline{x}) \wedge I(\overline{x}')\rangle$.

The variant expression applies to specific events, so we pair it with an event name in order to meaningfully evaluate it. This expression can be translated into an open $\mathcal{FOPEQ}$-term, which we denote by $n(\overline{x})$, and we use this to construct a formula based on the status of the event(s) in the signature Σ.

- For each $\langle e, \texttt{anticipated}\rangle \in \Sigma.E$ we form the sentence $\langle e, n(\overline{x}') \leq n(\overline{x})\rangle$.
- For each $\langle e, \texttt{convergent}\rangle \in \Sigma.E$ we form the sentence $\langle e, n(\overline{x}') < n(\overline{x})\rangle$.

Note that we are assuming the existence of a suitable type for variant expressions and the usual arithmetic interpretation of the predicates $<$ and $\leq$.

Event guard(s) and witnesses are also labelled predicates that can be translated into open $\mathcal{FOPEQ}$-formulae over the variables $\overline{x}$ in V and parameters $\overline{p}$. These are denoted by $G(\overline{x}, \overline{p})$ and $W(\overline{x}, \overline{p})$ respectively. In Event-B, actions are interpreted as before-after predicates, and so they can be translated into open $\mathcal{FOPEQ}$-formulae denoted by $BA(\overline{x}, \overline{p}, \overline{x}')$. Thus for each event we form the formula $\phi(\overline{x}, \overline{x}') = \exists \overline{p} \cdot G(\overline{x}, \overline{p}) \wedge W(\overline{x}, \overline{p}) \wedge BA(\overline{x}, \overline{p}, \overline{x}')$ where $\overline{p}$ are the event parameters. This generates an $\mathcal{EVT}$-sentence of the form $\langle e, \phi(\overline{x}, \overline{x}')\rangle$. The `Init` event, which is an Event-B sentence over only the after variables denoted by $\overline{x}'$, is a special case. In this case, we form the $\mathcal{EVT}$-sentence $\langle \texttt{Init}, \phi(\overline{x}')\rangle$.

There is no formal semantics for Event-B defined in the literature as such. Therefore, we have based our construction of $\mathcal{EVT}$-models on the notion of a mathematical model as described by Abrial [1, Ch. 14]. In these models the state is represented as a sequence of variable-values and models are defined over before and after states. We interpret these states as sets of variable-to-value mappings in our definition of $\mathcal{EVT}$-models.

Definition 8 (Σ-***State***$_A$)**.** For any given $\mathcal{EVT}$-signature Σ we define a **Σ-state** of an algebra A as a set of (sort appropriate) variable-to-value mappings whose domain is the set of sort-indexed variable names $\Sigma.V$. We define the set $State_A$ as the set of all such Σ-states. By "sort appropriate" we mean that for any variable x of sort s in V, the corresponding value for x should be drawn from $|A|_s$, the carrier set of s given by the $\mathcal{FOPEQ}$-model A.

Definition 9 ($\Sigma_{\mathcal{EVT}}$**-Model**)**.** Given $\Sigma = \langle S, \Omega, \Pi, E, V\rangle$, $\mathbf{Mod}(\Sigma)$ provides a category of models, where a **model** over Σ is a tuple $\langle A, L, R\rangle$. A is a $\Sigma_{\mathcal{FOPEQ}}$-model, and the non-empty initialising set $L \subseteq State_A$ provides the states after the `Init` event. Then for every event name $e \in dom(E)$, other than `Init`, we define $R.e \subseteq State_A \times State_A$ where for each pair of states $\langle s, s'\rangle$ in $R.e$, s provides values for the variables x in V, and s' provides values for their primed versions x'. Then $R = \{R.e \mid e \in dom(E)\ and\ e \neq \texttt{Init}\}$.

Intuitively, a model over Σ maps every event name $e \in dom(\Sigma.E)$ to a set of variable-to-value mappings over the carriers corresponding to the sorts of each of the variables $x \in \Sigma.V$ and their primed versions x'. In cases where there are no variables in $\Sigma.V$, L is the singleton $\{\{\}\}$.

For example, given the event e on the right, with natural number variable x and boolean variable y we construct the variable to value mappings:

```
Event e ≙
  when  grd1:   x<2
  then  act1:   x := x + 1
        act2:   y := false
```

$$R_e = \left\{ \begin{array}{ll} \{x \mapsto 0, y \mapsto false, x' \mapsto 1, y' \mapsto false\}, & \{x \mapsto 0, y \mapsto true, x' \mapsto 1, y' \mapsto false\}, \\ \{x \mapsto 1, y \mapsto false, x' \mapsto 2, y' \mapsto false\}, & \{x \mapsto 1, y \mapsto true, x' \mapsto 2, y' \mapsto false\} \end{array} \right\}$$

The notation used above is interpreted as *variable name* $\mapsto$ *value* where the value is drawn from the carrier set corresponding to the sort of the variable name given in $\Sigma.V$. We note that trivial models be excluded as the initialising set L is never empty. In cases where there are no variables in $\Sigma.V$, L is the singleton $L = \{\{\}\}$.

The reduct of an $\mathcal{EVT}$-model $M = \langle A, L, R\rangle$ along an $\mathcal{EVT}$-signature morphism $\sigma : \Sigma \to \Sigma'$ is given by $M|_\sigma = \langle A|_\sigma, L|_\sigma, R|_\sigma\rangle$. Here $A|_\sigma$ is the reduct of the $\mathcal{FOPEQ}$-component of the $\mathcal{EVT}$-model along the $\mathcal{FOPEQ}$-components of σ. $L|_\sigma$ and $R|_\sigma$ are based on the reduction of the states of A along σ, i.e. for every Σ'-state s of A, that is for every sorted map $s : \Sigma'.V \to |A|$, $s|_\sigma$ is the map $\Sigma'.V \to |A|$ given by the composition $\sigma_V; s$. This extends in the usual manner from states to sets of states and to relations on states.

Satisfaction: In order to define the satisfaction relation for $\mathcal{EVT}$, we describe an embedding from $\mathcal{EVT}$-signatures and models to $\mathcal{FOPEQ}$-signatures and models. Given an $\mathcal{EVT}$-signature $\Sigma = \langle S, \Omega, \Pi, E, V\rangle$ we form the following two $\mathcal{FOPEQ}$-signatures:

- $\Sigma^{(V,V')}_{\mathcal{FOPEQ}} = \langle S, \Omega \cup V \cup V', \Pi\rangle$ where V and V' are the variables and their primed versions, respectively, that are drawn from the $\mathcal{EVT}$-signature, and represented as 0-ary operators with unchanged sort. The intuition here is that the set of variable-to-value mappings for the free variables in an $\mathcal{EVT}$-signature Σ are represented by adding a distinguished 0-ary operation symbol to the corresponding $\mathcal{FOPEQ}$-signature for each of the variables $x \in V$ and their primed versions.
- Similarly, for the initial state and its variables, we construct the signature $\Sigma^{(V')}_{\mathcal{FOPEQ}} = \langle S, \Omega \cup V', \Pi\rangle$.

Given the $\mathcal{EVT}$ Σ-model $\langle A, L, R\rangle$, we construct the $\mathcal{FOPEQ}$-models:

- For every pair of states $\langle s, s'\rangle$, we form the $\Sigma^{(V,V')}_{\mathcal{FOPEQ}}$-model expansion $A^{(s,s')}$, which is the $\mathcal{FOPEQ}$-component A of the $\mathcal{EVT}$-model, with s and s' added as interpretations for the new operators that correspond to the variables from V and V' respectively.
- For each initial state $s' \in L$ we construct the $\Sigma^{(V')}_{\mathcal{FOPEQ}}$-model expansion $A^{(s')}$ analogously.

For any $\mathcal{EVT}$-sentence over Σ of the form $\langle e, \phi(\overline{x}, \overline{x}')\rangle$ we create a corresponding $\mathcal{FOPEQ}$-formula by replacing the free variables with their corresponding operator symbols. We write this (closed) formula as $\phi(\overline{x}, \overline{x}')$.

Definition 10 (**Satisfaction Relation**). For any $\mathcal{EVT}$-model $\langle A, L, R\rangle$ and $\mathcal{EVT}$-sentence $\langle e, \phi(\overline{x}, \overline{x}')\rangle$, where e is an event name other than `Init`, we define:

$\langle A, L, R\rangle \models_{\Sigma} \langle e, \phi(\overline{x}, \overline{x}')\rangle \iff \forall \langle s, s'\rangle \in R.e \cdot A^{(s,s')} \models_{\Sigma^{(V,V')}_{\mathcal{FOPEQ}}} \phi(\overline{x}, \overline{x}')$

Similarly, we evaluate the satisfaction condition of $\mathcal{EVT}$-sentences of the form $\langle \texttt{Init}, \phi(\overline{x}')\rangle$ as follows:

$\langle A, L, R\rangle \models_{\Sigma} \langle \texttt{Init}, \phi(\overline{x}')\rangle \iff \forall s' \in L \cdot A^{(s')} \models_{\Sigma^{(V')}_{\mathcal{FOPEQ}}} \phi(\overline{x}')$

Theorem 1 (Satisfaction Condition). *Given $\mathcal{EVT}$ signatures Σ_1 and Σ_2, a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$, a Σ_2-model M_2 and a Σ_1-sentence ψ_1, the following satisfaction condition holds:*

$$\mathbf{Mod}(\sigma)(M_2) \models_{\mathcal{EVT}_{\Sigma_1}} \psi_1 \quad \iff \quad M_2 \models_{\mathcal{EVT}_{\Sigma_2}} \mathbf{Sen}(\sigma)(\psi_1)$$

Proof. Let M_2 be the model $\langle A_2, L_2, R_2\rangle$, and ψ_1 the sentence $\langle e, \phi(\overline{x}, \overline{x}')\rangle$. Then the satisfaction condition is equivalent to

$$\forall \langle s, s'\rangle \in R_2|_{\sigma}.e \cdot (A_2|_{\sigma})^{(s,s')}|_{\sigma} \models_{\mathcal{FOPEQ}_{\Sigma^{(V_1,V_1')}_{\mathcal{FOPEQ}}}} \phi(\overline{x}, \overline{x}')$$

$$\iff \forall \langle s, s'\rangle \in R_2.\sigma_E(e) \cdot A_2^{(s,s')} \models_{\mathcal{FOPEQ}_{\Sigma^{(V_2,V_2')}_{\mathcal{FOPEQ}}}} Sen(\sigma)(\phi(\overline{x}, \overline{x}'))$$

Here, validity follows from the validity of satisfaction in $\mathcal{FOPEQ}$. We prove a similar result for initial events in the same way. □

Pragmatics of Specification Building in $\mathcal{EVT}$: We represent an Event-B specification, such as that for `mac1` in Figure 1, as a presentation over $\mathcal{EVT}$. For any signature Σ, a Σ-presentation is a set of Σ-sentences. A model of a Σ-presentation is a Σ-model that satisfies all of the sentences in the presentation [5]. Thus, for a presentation in $\mathcal{EVT}$, model components corresponding to an event must satisfy all of the sentences specifying that event. This incorporates the standard semantics of the *extends* operator for events in Event-B where the extending event implicitly has all the parameters, guards and actions of the extended event but can have additional parameters, guards and actions [3].

An interesting aspect is that if a variable is not assigned to within an action, then a model for the event may associate a new value with this variable. Some languages deal with this using a *frame condition*, asserting implicitly that values for unmodified variables do not change. In Event-B such a condition would cause complications when combining presentations, since variables unreferenced in one event will be constrained not to change, and this may contradict an action for them in the other event. As far as we can tell, the informal semantics for the Event-B language do not require a frame condition, and we have not included one in our definition.

3 Relating $\mathcal{FOPEQ}$ and $\mathcal{EVT}$

Initially, we defined the relationship between $\mathcal{FOPEQ}$ and $\mathcal{EVT}$ to be a duplex institution formed from a restricted version of $\mathcal{EVT}$ ($\mathcal{EVT}_{res}$) and $\mathcal{FOPEQ}$

where $\mathcal{EVT}_{res}$ is the institution $\mathcal{EVT}$ but does not contain any $\mathcal{FOPEQ}$ components. Duplex institutions are constructed by enriching one institution, in this case $\mathcal{EVT}_{res}$, by the sentences of another, in this case $\mathcal{FOPEQ}$, using an institution semi-morphism [5, 11]. This approach would allow us to constrain $\mathcal{EVT}_{res}$ by $\mathcal{FOPEQ}$ and thus facilitate the use of $\mathcal{FOPEQ}$-sentences in an elegant way. However, duplex institutions are not supported in HETS [9], and therefore we opt for a comorphism which embeds the simpler institution $\mathcal{FOPEQ}$ into the more complex institution $\mathcal{EVT}$ [11].

Definition 11 (The institution comorphism ρ). We define $\rho : \mathcal{FOPEQ} \to \mathcal{EVT}$ to be an institution comorphism composed of:

- The functor $\rho^{Sign} : \mathbf{Sign}_{\mathcal{FOPEQ}} \to \mathbf{Sign}_{\mathcal{EVT}}$ which takes as input a $\mathcal{FOPEQ}$-signature of the form $\langle S, \Omega, \Pi \rangle$ and extends it with the set $E = \{\langle \texttt{Init ordinary} \rangle\}$ and an empty set of variable names V. $\rho^{Sign}(\sigma)$ works as σ on S, Ω and Π, it is the identity on the `Init` event and the empty function on the empty set of variable names.
- The natural transformation $\rho^{Sen} : \mathbf{Sen}_{\mathcal{FOPEQ}} \to \rho^{Sign}; \mathbf{Sen}_{\mathcal{EVT}}$ which pairs any closed $\mathcal{FOPEQ}$-sentence (given by ϕ) with the `Init` event name to form the $\mathcal{EVT}$-sentence $\langle \texttt{Init}, \phi \rangle$. As there are no variables in the signature, we do not require ϕ to be over the variables $\overline{x}$ and $\overline{x}'$.
- The natural transformation $\rho^{Mod} : (\rho^{Sign})^{op}; \mathbf{Mod}_{\mathcal{EVT}} \to \mathbf{Mod}_{\mathcal{FOPEQ}}$ is such that for any $\mathcal{FOPEQ}$-signature Σ,
$$\rho^{Mod}_{\Sigma}(Mod(\rho^{Sign}(\Sigma))) = \rho^{Mod}_{\Sigma}(\langle A, L, \varnothing \rangle) = A$$

Theorem 2. *The institution comorphism ρ is defined such that for any $\Sigma \in |\mathbf{Sign}_{\mathcal{FOPEQ}}|$, the translations $\rho^{Sen}_{\Sigma} : \mathbf{Sen}_{\mathcal{FOPEQ}}(\Sigma) \to \mathbf{Sen}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))$ and $\rho^{Mod}_{\Sigma} : \mathbf{Mod}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma)) \to \mathbf{Mod}_{\mathcal{FOPEQ}}(\Sigma)$ preserve the satisfaction relation. That is, for any $\psi \in \mathbf{Sen}_{\mathcal{FOPEQ}}(\Sigma)$ and $M' \in |\mathbf{Mod}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))|$*

$$\rho^{Mod}_{\Sigma}(M') \models_{\mathcal{FOPEQ}_{\Sigma}} \psi \iff M' \models_{\mathcal{EVT}_{\rho^{Sign}(\Sigma)}} \rho^{Sen}_{\Sigma}(\psi) \qquad (*)$$

Proof. By definition 11, $M' = \langle A, L, \varnothing \rangle$, $\rho^{Mod}_{\Sigma}(M') = A$ and $\rho^{Sen}_{\Sigma}(\psi) = \langle \texttt{Init}, \psi \rangle$. Therefore, we transform $(*)$ into

$$A \models_{\mathcal{FOPEQ}_{\Sigma}} \psi \iff M' \models_{\mathcal{EVT}_{\rho^{Sign}(\Sigma)}} \langle \texttt{Init}, \psi \rangle$$

Then, by the definition of satisfaction in $\mathcal{EVT}$ (Definition 10)

$$A \models_{\mathcal{FOPEQ}_{\Sigma}} \psi \iff A^{(s')} \models_{\mathcal{FOPEQ}_{(\rho^{Sign}(\Sigma))^{(V')}_{\mathcal{FOPEQ}}}} \psi$$

We deduce that $\Sigma = (\rho^{Sign}(\Sigma))^{V'}_{\mathcal{FOPEQ}}$, since there are no variable names in V' and thus no new operator symbols are added to the signature. As there are no variable names in V', $L = \{\{\}\}$, so we can conclude that $A^{(s')} = A$. Thus the satisfaction condition holds. □

For a Σ-specification written over $\mathcal{FOPEQ}$ we can use the specification building operator `_ with` $\rho : Spec_{\mathcal{FOPEQ}}(\Sigma) \to Spec_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))$ to interpret this as a specification over $\mathcal{EVT}$ [11]. This results in a specification with just the `Init` event and no variables, containing $\mathcal{FOPEQ}$-sentences that hold in the initial

state. This process is used to represent contexts, specifically their axioms, which are written over $\mathcal{FOPEQ}$ as sentences over $\mathcal{EVT}$.

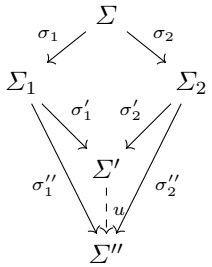
In cases where a specification is enriched with new events, then the axioms and invariants should also apply to these new events. One approach to this would require a new kind of $\mathcal{EVT}$-sentence for invariants, which we denote by $\langle inv, \phi(\overline{x}, \overline{x}') \rangle$, these are applied to all events in the specification when evaluating the satisfaction condition. We do not present these details fully here due to space concerns.

3.1 Pushouts and Amalgamation

We ensure that the institution $\mathcal{EVT}$ has good modularity properties by proving that $\mathcal{EVT}$ admits the amalgamation property: all pushouts in $\mathbf{Sign}_{\mathcal{EVT}}$ exist and every pushout diagram in $\mathbf{Sign}_{\mathcal{EVT}}$ admits *weak* model amalgamation [11].

Proposition 1. *Pushouts exist in* $\boldsymbol{Sign}_{\mathcal{EVT}}$.

Proof. Given two signature morphisms $\sigma_1 : \Sigma \to \Sigma_1$ and $\sigma_2 : \Sigma \to \Sigma_2$ a pushout is a triple $(\Sigma', \sigma_1', \sigma_2')$ that satisfies the universal property: for all triples $(\Sigma'', \sigma_1'', \sigma_2'')$ there exists a unique morphism $u : \Sigma' \to \Sigma''$ such that the diagram on the left below commutes. Our pushout construction follows $\mathcal{FOPEQ}$ for the elements that $\mathcal{FOPEQ}$ has in common with $\mathcal{EVT}$. In $\mathbf{Sign}_{\mathcal{EVT}}$ the additional elements are E and V.



- *Set of ⟨event name, status⟩ pairs E:* The set of all event names in the pushout is the pushout in **Set** on event names only. Then, the status of an event in the pushout is the supremum of all statuses of all events that are mapped to it. Since signature morphisms map $\langle \texttt{Init}, \texttt{ordinary} \rangle$ to $\langle \texttt{Init}, \texttt{ordinary} \rangle$ the pushout does likewise. The universality property for E follows from that of **Set**.
- *Set of sort-indexed variable names V:* The set of sort-indexed variable names in the pushout is the pushout in $\mathcal{FOPEQ}$ for the sort components and the pushout in **Set** for the variable names. This is a similar construction to the pushout for operation names in $\mathcal{FOPEQ}$ as these also have to follow the sort pushout. Thus, the universality property for V follows from that of **Set** and the $\mathcal{FOPEQ}$ pushout for sorts.

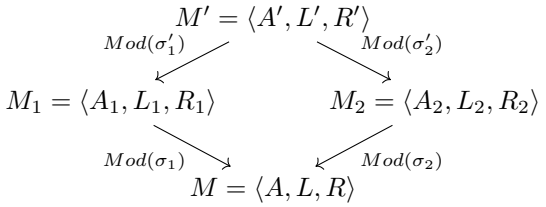
□

Proposition 2. *Every pushout diagram in* $\boldsymbol{Sign}_{\mathcal{EVT}}$ *admits weak model amalgamation.*

We decompose this proposition into two further subpropositions:

Proposition 2(a). *For* $M_1 \in |\boldsymbol{Mod}(\Sigma_1)|$ *and* $M_2 \in |\boldsymbol{Mod}(\Sigma_2)|$ *such that* $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, *there exists a model (the amalgamation of* M_1 *and* M_2*)* $M' \in |\boldsymbol{Mod}(\Sigma')|$ *such that* $M'|_{\sigma_1'} = M_1$ *and* $M'|_{\sigma_2'} = M_2$.

Proof. Consider the commutative diagram with signature morphisms $\sigma_1, \sigma_2, \sigma'_1$ and σ'_2 below:



We construct $M' = \langle A', L', R' \rangle$ as follows. A' is the $\mathcal{FOPEQ}$-model (amalgamation of A_1 and A_2) over $\mathcal{FOPEQ}$. We construct the initialising set L' by amalgamating L_1 and L_2 to get the set of all possible combinations of variable mappings, while respecting the amalgamations induced on variable names via the pushout V'. We construct the relation R', which is the amalgamation of R_1 and R_2, in a similar manner. □

Proposition 2(b). *For any two model morphisms $f_1 : M_{11} \to M_{12}$ in $\mathbf{Mod}(\Sigma_1)$ and $f_2 : M_{21} \to M_{22}$ in $\mathbf{Mod}(\Sigma_2)$ such that $f_1|_{\sigma_1} = f_2|_{\sigma_2}$, there exists a model morphism (the amalgamation of f_1 and f_2) called $f' : M'_1 \to M'_2$ in $\mathbf{Mod}(\Sigma')$, such that $f'|_{\sigma'_1} = f_1$ and $f'|_{\sigma'_2} = f_2$.*

We have omitted this proof but it can be found on our webpage[1].

4 Modularising Event-B Specifications

Our definition of $\mathcal{EVT}$ allows the restructuring of Event-B specifications using the standard specification-building operators for institutions [11]. Thus $\mathcal{EVT}$ provides a means for writing down and splitting up the components of an Event-B system, facilitating increased modularity for Event-B specifications. Figure 4 contains heterogeneous structured specifications corresponding to the Event-B machine `mac1` defined in Figure 1. Since HETS is our target platform, where each institution is represented as a "logic", we use its notation and implementation of the logic for $\mathcal{CASL}$ to represent the $\mathcal{FOPEQ}$ components of our specifications.

Lines 1–6: TWOBOOLS can be presented as a pure $\mathcal{CASL}$ specification, declaring two boolean variables constrained to have different values.

Lines 7–17: LIGHTABSTRACT is a specification in the $\mathcal{EVT}$ logic for a single traffic light that extends (using keyword then) TWOBOOLS which is first translated via the comorphism ρ into a specification over $\mathcal{EVT}$. It contains the events `set_go` and `set_stop`, with the constraint that a light can only be set to "go" if its opposite light is not set to "go". We use "thenAct" in place of the "then" Event-B keyword to distinguish from the "then" specification-building operator.

Lines 18–32: The specification MAC1 combines (using keyword and) two versions of LIGHTABSTRACT, each with a different signature morphism (σ_1 and σ_2) mapping the specification variables and event names to those in the Event-B machine. The where notation used on lines 22–32 is just a convenient presentation of the signature morphisms, it is not part of the syntax of the specification language that we use in HETS.

[1] http://www.cs.nuim.ie/∼mfarrell/extended.pdf

```
1  logic CASL
2  spec TwoBools =
3    Bool
4    then
5      ops i_go, u_go : Bool
6      . ¬ (i_go = true ∧ u_go = true)

7  logic EVT
8  spec LightAbstract =
9    TwoBools with ρ
10   then
11     Initialisation ordinary
12       thenAct act1 : i_go := false
13     Event set_go ≙ ordinary
14       when grd1: u_go = false
15       thenAct act1: i_go := true
16     Event set_stop ≙ ordinary
17       thenAct act1: i_go := false
```

```
18 logic EVT
19 spec mac1 =
20   (LightAbstract with σ1)
21     and (LightAbstract with σ2)
22   where
23     σ1 = {i_go ↦ cars_go, u_go ↦ peds_go,
24           ⟨set_go, ordinary⟩
25             ↦ ⟨set_cars_go, ordinary⟩,
26           ⟨set_stop, ordinary⟩
27             ↦ ⟨set_cars_stop, ordinary⟩}
28     σ2 = {i_go ↦ peds_go, u_go ↦ cars_go,
29           ⟨set_go, ordinary⟩
30             ↦ ⟨set_peds_go, ordinary⟩,
31           ⟨set_stop, ordinary⟩
32             ↦ ⟨set_peds_stop, ordinary⟩}
```

Fig. 4: A modular institution-based presentation corresponding to the abstract machine `mac1` in Fig 1.

We get a presentation over the institution $\mathcal{EVT}$ for `mac1` by flattening out the structuring. Notice that the specification for each individual light had to be explicitly written down twice in the Event-B machine in Figure 1 (lines 11–15 and lines 16–20). In our modular institution-based presentation we only need one light specification and simply supply the required variable and event mappings. In this way, $\mathcal{EVT}$ provides a more flexible degree of modularity than is currently present in Event-B.

4.1 Refinement in the $\mathcal{EVT}$ Institution

Event-B supports three forms of machine refinement: the refinement of event internals (guards and actions) and invariants; the addition of new events; and the decomposition of an event into several events [2]. It is therefore essential for any formalisation of Event-B to be capable of capturing refinement.

In general for institutions, a refinement from an abstract specification A to some concrete specification C is defined using model-class inclusion as $|Mod(C)| \subseteq |Mod(A)|$ when $Sig[A] = Sig[C]$. In Event-B, new variable or event names cannot be added if the signatures stay the same. This provides only one option: strengthen the formulae in event definitions, which will result in at most the same number of models. This accounts for the first form of refinement in Event-B. Both of the other forms of refinement in Event-B cause the signatures to change i.e. the set of events will get larger when adding or decomposing events. In the case when the signatures are different, we can define a signature morphism $\sigma : Sig[A] \to Sig[C]$ from which we can construct the model reduct $Mod(\sigma) : Mod(C) \to Mod(A)$. We can thus restrict the concrete model to only contain elements of the abstract signatures by applying the model reduct before evaluating the subset relation defined above.

4.2 A Modular, Refined Specification

Figure 5 contains a presentation over $\mathcal{EVT}$ corresponding to the main elements of the Event-B specification MAC2 presented in Figures 2 and 3. Here, we present three $\mathcal{CASL}$ specifications and three $\mathcal{EVT}$ specifications.

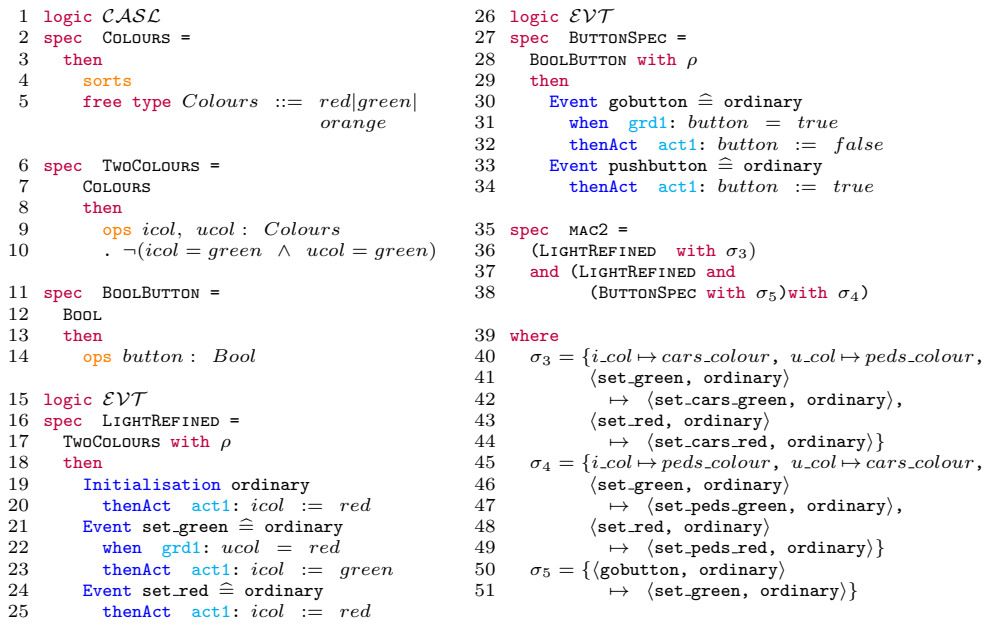


Fig. 5: A modular institution-based presentation corresponding to the refined machine `mac2` specified in Fig 3.

Lines 1–10: We specify the *Colours* data type with a standard $\mathcal{CASL}$ specification, as can be seen in Figure 2. The specification TwoColours describes two variables of type *Colours* constrained to be not both green at the same time. This corresponds to the gluing invariants on lines 5 and 7 of Figure 3. The specification modularisation constructs used in Figure 5, allow these properties to be handled distinctly and in a manner that facilitates comparison with the TwoBools specification on lines 1–6 of Figure 4.

Lines 15–25: A specification for a single light is provided in LightRefined which uses TwoColours to describe the colour of the lights. As was the case with LightAbstract in Figure 4, the specification makes clear how a single light operates. An added benefit here is that a direct comparison with the abstract specification can be done on a per-light basis.

Lines 11–14, 26–34: The specifications BoolButton and ButtonSpec account for the part of the mac2 specification that requires a button. These details were woven through the code in Figure 3 (lines 2, 8, 16, 18, 29, 30) but the specification-building operators allow us to modularise the specification and group these related definitions together, clarifying how the button actually operates.

Lines 35–51: Finally, to bring this all together we combine a copy of LightRefined with a specification corresponding to the sum (and) of LightRefined and ButtonSpec with appropriate signature morphisms. This second specification combines the event `gobutton` in ButtonSpec with the event `set_green` in LightRefined thus accounting for `set_peds_green` in Figure 3. One small

```
1  refinement REF : Bool to Colours =
2      Bool  ↦ Colours,
3      true  ↦ green,
4      false ↦ red
5      i_go ↦ icol,
6      u_go ↦ ucol,
7      ⟨set_peds_go, ordinary⟩
8         ↦ ⟨set_peds_green, ordinary⟩,
9      ⟨set_peds_stop, ordinary⟩
10        ↦ ⟨set_peds_red, ordinary⟩,
11     ⟨set_cars_go, ordinary⟩
12        ↦ ⟨set_cars_green, ordinary⟩,
13     ⟨set_cars_stop, ordinary⟩
14        ↦ ⟨set_cars_red, ordinary⟩
15 end
```

Fig. 6: Defining the refinement relationships between the concrete and abstract presentations.

issue involves making sure that the name replacements are done correctly, and in the correct order, hence the bracketing on lines 37–38 is important.

The combination of these specifications involves merging two events with different names: `gobutton` from ButtonSpec with the event `set_green` from LightRefined. To ensure that these differently-named events are combined into an event of the same name we use the signature morphism σ_5 to give `gobutton` the same name as `set_green` before combining them. Ensuring that the events have the same name allows the and operator to combine both events' guards and actions and the morphism σ_4 to name the resulting event `set_peds_green`. The resulting specification also contains the event `pushbutton`. The labels given to guards/actions are syntactic sugar to make the specification aesthetically resemble the usual Event-B notation for guards/actions.

Figure 6 uses the refinement syntax available in Hets to specify each of the refinements in the specification of the concrete machine `mac2`:

Lines 2–4: define the data refinement of *Bool* into *Colours*, with an appropriate mapping for the values.

Lines 5–6: define the refinement of the two boolean variables into their corresponding variables of type *Colour*. In combination with lines 2–4, this corresponds to the gluing invariants on lines 5 and 7 of Figure 3.

Lines 7–14: define the refinement relation between the four events: this corresponds to the refines statements on lines 14, 20, 23 and 27 of Figure 3.

5 Conclusion and Future Work

Currently, the core benefit of $\mathcal{EVT}$, our institution for Event-B, is the increased modularity of Event-B specifications via the use of specification-building operators. The concept of refinement, central to Event-B, is also well-developed in the theory of institutions, and we have shown how this can be applied here. Devising meaningful institutions and corresponding morphisms to/from Event-B provides a mechanism not only for ensuring the safety of a particular specification but also, via morphisms, a potential for integration with other formalisms. Interoperability and heterogeneity are significant goals in the field of software engineering, and we believe that the work presented in this paper provides a basis for the integration of Event-B with other formalisms defined in this way.

The Heterogeneous Tool-Set Hets provides a framework for heterogeneous specifications where each formalism is represented as a logic and understood in the theory of institutions [9]. Our logic for $\mathcal{EVT}$ utilises the already existing

institution $\mathcal{CASL}$ [10] to account for the $\mathcal{FOPEQ}$ parts of the $\mathcal{EVT}$ institution thus taking advantage of the interoperability/heterogeneity supplied by Hets. $\mathcal{CASL}$ provides sorts and predicates like those written in lines 4–6 from Figure 4.

At present we can parse, statically analyse and combine specifications written over $\mathcal{EVT}$. Future work includes developing comorphisms to translate between $\mathcal{EVT}$ and other logics in Hets as well as integrating with the provers currently available in Hets (e.g. Isabelle). Comorphisms between these theorem provers and $\mathcal{EVT}$ will allow us to prove our specifications correct in Hets. We envisage that development should take place here to fully take advantage of the prospects for interoperability. A translation from Event-B to $\mathcal{EVT}$ in the future will not only enable us to fully utilise both the *Rodin Platform* and Hets, but will also provide a translational semantics for Event-B using the theory of institutions.

Acknowledgements The authors would like to acknowledge the reviewers for their helpful comments and Ionut Tutu for his assistance with the presentation of the technical details of our institution for Event-B.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.
2. J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, 2010.
3. J.-R. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1-2):1–28, 2007.
4. A. Achouri and L. Jemni Ben Ayed. UML activity diagram to Event-B: A model transformation approach based on the institution theory. In *Information Reuse and Integration*, pages 823–829, Aug. 2014.
5. J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
6. A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Supporting reuse in Event-B development: Modularisation approach. In *Abstract State Machines, Alloy, B and Z*, volume 5977 of *LNCS*, pages 174–188, 2010.
7. M. Jastram and P. M. Butler. *Rodin User's Handbook: Covers Rodin V.2.8*. CreateSpace Independent Publishing Platform, USA, 2014.
8. A. Knapp, T. Mossakowski, M. Roggenbach, and M. Glauer. An institution for simple UML state machines. In *Fundamental Approaches to Software Engineering*, volume 9033 of *LNCS*, pages 3–18, 2015.
9. T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set, Hets. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 519–522, 2007.
10. P. D. Mosses, editor. *CASL Reference Manual*, volume 2960 of *LNCS*. Springer, 2004.
11. D. Sanella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Springer, 2012.