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 [7]. 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* [8], 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 [3]. 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 [3]. 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 [4]. 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 [14], 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 [5]. 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 using 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 $\mathcal{UML}$. 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 [6].

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

Semantics: *a functor **Mod** : $\textbf{\textit{Sign}}^{op} \to \textbf{\textit{Cat}}$ giving a category $\textbf{\textit{Mod}}(\Sigma)$ of Σ-models for each signature Σ and a functor $\textbf{\textit{Mod}}(\sigma) : \textbf{\textit{Mod}}(\Sigma') \to \textbf{\textit{Mod}}(\Sigma)$ 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 $\textbf{\textit{Mod}}(\sigma)$ of models and $\textbf{\textit{Sen}}(\sigma)$ of sentences we have for any $\phi \in \textbf{\textit{Sen}}(\Sigma)$ and $M' \in |\textbf{\textit{Mod}}(\Sigma')|$.*

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

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 [2]. 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 $\langle$event name, status$\rangle$ where status belongs to the poset $\{$`ordinary` $<$ `anticipated` $<$ `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 $\sigma_E\langle$`Init`, `ordinary`$\rangle = \langle$`Init`, `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 $\overline{x}$.

```
MACHINE m SEES ctx  refines a
  VARIABLES $\overline{x}$
  INVARIANTS $I(\overline{x})$
  VARIANT $n(\overline{x})$
  EVENTS
  Initialisation ordinary
    then  act-name: $BA(\overline{x}, \overline{x}')$
  Event e $\widehat{=}$ status
    any $\overline{p}$
    when  guard-name: $G(\overline{x}, \overline{p})$
    with  witness-name: $W(\overline{x}, \overline{p})$
    then  act-name: $BA(\overline{x}, \overline{p}, \overline{x}')$
       ⋮
END
```

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 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 [2, 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 $R = \{R.e \mid e \in dom(E) \text{ and } e \neq \texttt{Init}\}$.*

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 $\langle$`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:*

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

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 [6]. 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 [4].

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 [6, 14]. 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 [11], and therefore we opt for a comorphism which embeds the simpler institution $\mathcal{FOPEQ}$ into the more complex institution $\mathcal{EVT}$ [14].

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 20)

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

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 $\mathcal{EVT}$ [14]. 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 [14].

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.

$$\begin{tikzcd}
& \Sigma \arrow[dl, "\sigma_1"'] \arrow[dr, "\sigma_2"] & \\
\Sigma_1 \arrow[dr, "\sigma'_1"] \arrow[ddr, "\sigma''_1"'] & & \Sigma_2 \arrow[dl, "\sigma'_2"'] \arrow[ddl, "\sigma''_2"] \\
& \Sigma' \arrow[d, dashed, "u"] & \\
& \Sigma'' &
\end{tikzcd}$$

– *Set of* $\langle$*event name, status*$\rangle$ *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 $\langle$`Init,ordinary`$\rangle$ to $\langle$`Init`, `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:

$$\begin{tikzcd}
 & M' = \langle A', L', R' \rangle \arrow[dl, "Mod(\sigma_1')"'] \arrow[dr, "Mod(\sigma_2')"] & \\
M_1 = \langle A_1, L_1, R_1 \rangle \arrow[dr, "Mod(\sigma_1)"'] & & M_2 = \langle A_2, L_2, R_2 \rangle \arrow[dl, "Mod(\sigma_2)"] \\
 & M = \langle A, L, R \rangle &
\end{tikzcd}$$

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 $\textbf{Mod}(\Sigma_1)$ and $f_2 : M_{21} \to M_{22}$ in $\textbf{Mod}(\Sigma_2)$ 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 $\textbf{Mod}(\Sigma')$, 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 [14]. 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

```
logic CASL
spec  TwoBools =
  Bool
  then
    ops i_go, u_go : Bool
    . ¬ (i_go = true ∧ u_go = true)

logic EVT
spec  LightAbstract =
  TwoBools with ρ
  then
    Initialisation ordinary
      thenAct  act1 : i_go := false
    Event set_go ≙ ordinary
      when  grd1: u_go = false
      thenAct  act1: i_go := true
    Event set_stop ≙ ordinary
      thenAct  act1: i_go := false
```

```
logic EVT
spec  mac1 =
  (LightAbstract with σ1)
    and (LightAbstract with σ2)
  where
    σ1 = {i_go ↦ cars_go, u_go ↦ peds_go,
          ⟨set_go, ordinary⟩
            ↦ ⟨set_cars_go, ordinary⟩,
          ⟨set_stop, ordinary⟩
            ↦ ⟨set_cars_stop, ordinary⟩}
    σ2 = {i_go ↦ peds_go, u_go ↦ cars_go,
          ⟨set_go, ordinary⟩
            ↦ ⟨set_peds_go, ordinary⟩,
          ⟨set_stop, ordinary⟩
            ↦ ⟨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 [3]. 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.

```
 1 logic CASL
 2 spec  Colours =
 3   then
 4     sorts
 5     free type Colours  ::=  red|green|
                               orange

 6 spec  TwoColours =
 7     Colours
 8     then
 9       ops icol,  ucol :  Colours
10       . ¬(icol = green  ∧  ucol = green)

11 spec  BoolButton =
12   Bool
13   then
14     ops button :  Bool

15 logic EVT
16 spec  LightRefined =
17   TwoColours with ρ
18   then
19     Initialisation ordinary
20       thenAct  act1: icol  :=  red
21     Event set_green ≙ ordinary
22       when  grd1: ucol  =  red
23       thenAct  act1: icol  :=  green
24     Event set_red ≙ ordinary
25       thenAct  act1: icol  :=  red
```

```
26 logic EVT
27 spec  ButtonSpec =
28   BoolButton with ρ
29   then
30     Event gobutton ≙ ordinary
31       when  grd1: button  =  true
32       thenAct  act1: button  :=  false
33     Event pushbutton ≙ ordinary
34       thenAct  act1: button  :=  true

35 spec  mac2 =
36   (LightRefined  with σ3)
37   and (LightRefined and
38         (ButtonSpec with σ5)with σ4)

39 where
40   σ3 = {i_col ↦ cars_colour,  u_col ↦ peds_colour,
41         ⟨set_green, ordinary⟩
42           ↦  ⟨set_cars_green, ordinary⟩,
43         ⟨set_red, ordinary⟩
44           ↦  ⟨set_cars_red, ordinary⟩}
45   σ4 = {i_col ↦ peds_colour,  u_col ↦ cars_colour,
46         ⟨set_green, ordinary⟩
47           ↦  ⟨set_peds_green, ordinary⟩,
48         ⟨set_red, ordinary⟩
49           ↦  ⟨set_peds_red, ordinary⟩}
50   σ5 = {⟨gobutton, ordinary⟩
51           ↦  ⟨set_green, ordinary⟩}
```

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 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.

```
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.

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 [11]. Our logic for $\mathcal{EVT}$ utilises the already existing institution $\mathcal{CASL}$ [1] 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.

A $\mathcal{FOPEQ}$, the insitution for first-order predicate logic with equality

Definition 12 ($\mathcal{FOPEQ}$-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 13 ($\Sigma_{\mathcal{FOPEQ}}$-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 14 ($\Sigma_{\mathcal{FOPEQ}}$-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 $f_A : |A|_{s_1} \times \cdots \times |A|_{s_n} \rightarrow |A|_s$ 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.

B Defining $\mathcal{EVT}$, an institution for Event-B

There are two basic languages in Event-B, these are the Event-B mathematical language (propositional/predicate logic, set-theory and arithmetic) and the Event-B modelling language [2]. We use $\mathcal{FOPEQ}$, the institution for first-order logic with equality, to represent this mathematical language. Therefore, $\mathcal{EVT}$, our formalisation of Event-B in terms of institutions is based on splitting an Event-B specification into two parts:

- A data part, which can be defined using some standard institution such as that for algebra or first-order logic. We have chosen $\mathcal{FOPEQ}$, the institution for first order predicate logic with equality [14], since it most closely matches the kind of data specification needed.
- An event part, which defines a set of events in terms of formula constraining their before- and after- states. Our specification here is closely based on $\mathcal{UML}$, an institution for UML state machines [10]. We also draw some inspiration from the institution for CSPCASL, $\mathcal{CSPCASL}$, [13].

Note that we will write Σ in place of $\Sigma_{\mathcal{EVT}}$ when describing a signature over our institution for Event-B, $\mathcal{EVT}$. Where we speak of signatures over other institutions than $\mathcal{EVT}$ we will use the subscript notation. For example a signature over the institution for first-order predicate logic with equality will be denoted $\Sigma_{\mathcal{FOPEQ}}$.

B.1 Sign, the category of $\mathcal{EVT}$ signatures

In this section we define and prove that signatures over $\mathcal{EVT}$ and their respective morphisms form a category. In particular, we prove that we can compose signature morphisms, that this composition is associative and that there exists an identity signature morphism. The category **Sign** describes the structure of the vocabulary that can be used in an Event-B specification.

Definition 15 ($\mathcal{EVT}$-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 $\langle event\ name, status\rangle$ 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`*.*

Definition 16 ($\mathcal{EVT}$-Signature Morphism) *We define the* ***signature morphism*** *$\sigma : \Sigma \to \Sigma'$ to be 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 $\mathcal{FOPEQ}$ with composition, associativity of composition and identities as in $\mathcal{FOPEQ}$. In particular,*

$\sigma_S : \Sigma.S \to \Sigma'.S$ is a function mapping sort names to sort names.
$\sigma_\Omega : \Sigma.\Omega \to \Sigma'.\Omega$ is a family of functions mapping operation names in Ω, respecting the arities and result sorts.
$\sigma_\Pi : \Sigma.\Pi \to \Sigma'.\Pi$ is a family of functions mapping the predicate names in Π, respecting the arities and sorts.
$\sigma_E : \Sigma.E \to \Sigma'.E$ 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.

Lemma 1. *Signatures and signature morphisms define a category* ***Sign****. The objects are signatures and the arrows are signature morphisms*

Proof. Let $\Sigma = \langle S, \Omega, \Pi, E, V\rangle$ be a signature where $\langle S, \Omega, \Pi\rangle$ is a signature over $\mathcal{FOPEQ}$, $\Sigma_{\mathcal{FOPEQ}}$, the institution for first-order logic with equality [14]. E is a set of $\langle event\ name, status\rangle$ pairs and V is a set of sort indexed variable names.

Composition of signature morphisms:

- Signature morphisms can be composed, the composition of σ_S, σ_Ω and σ_Π is as in $\mathcal{FOPEQ}$ so we are only concerned here with the composition of σ_E and σ_V.

σ_E: Event names are not concerned with sort/arity so the only restriction is on pairs of the form $\langle \texttt{Initialisation}, \texttt{ordinary} \rangle$ for which, it is easy to see that the composition holds for σ_E.

σ_V: Variable names are sort indexed so σ_V utilises σ_S on these sorts and σ_V is order-preserving.

$$\begin{aligned}
&\sigma_2(\sigma_1(v : s)) \\
&\qquad\qquad = \sigma_2((\sigma_{1_V}(v) : \sigma_{1_S}(s))) \\
&\qquad\qquad = ((\sigma_{2_V}(\sigma_{1_V}(v)) : (\sigma_{2_S}(\sigma_{1_S}(s))))
\end{aligned}$$

Let $\sigma_1 : \Sigma_1 \to \Sigma_2$ and $\sigma_2 : \Sigma_2 \to \Sigma_3$. We can check that $\sigma_2 \circ \sigma_1$ is actually a morphism.

- For all $\langle e_1, st_1 \rangle \in \Sigma_1.E_1$ we have that $\sigma_1(\langle e_1, st_1 \rangle) \in \Sigma_2.E_2$ and for all $\langle e_2, st_2 \rangle \in \Sigma_2.E_2$ we have that $\sigma_2(\langle e_2, st_2 \rangle) \in \Sigma_3.E_3$. Therefore $\sigma_2(\sigma_1(\langle e_1, st_1 \rangle)) \in \Sigma_3.E_3$ so

$$\forall \langle e_1, st_1 \rangle \in \Sigma_1.E_1 \Rightarrow \sigma_2 \circ \sigma_1(\langle e_1, st_1 \rangle) \in \Sigma_3.E_3$$

- For $(v_1 : s_1) \in \Sigma_1.V_1$ we have that $\sigma_1((v_1 : s_1)) \in \Sigma_2.V_2$ and for $(v_2 : s_2) \in \Sigma_2.V_2$ then $\sigma_2((v_2 : s_2)) \in \Sigma_3.V_3$. Therefore $\sigma_2(\sigma_1((v_1 : s_1))) \in \Sigma_3.V_3$ so

$$(v_1 : s_1) \in \Sigma_1.V_1 \Rightarrow \sigma_2 \circ \sigma_1((v_1 : s_1)) \in \Sigma_3.V_3$$

– Composition of signature morphisms is associative, i.e.

$$(\sigma_3 \circ \sigma_2) \circ \sigma_1 = \sigma_3 \circ (\sigma_2 \circ \sigma_1)$$

For $\langle e, st \rangle \in \Sigma.E$: $\sigma_2 \circ \sigma_1(\langle e, st \rangle) = \sigma_2(\sigma_1(\langle e, st \rangle))$ and so $\sigma_3 \circ (\sigma_2 \circ \sigma_1)(\langle e, st \rangle) = \sigma_3(\sigma_2(\sigma_1(\langle e, st \rangle)))$ by the definition of composition. This is equal to $\sigma_3 \circ \sigma_2 \circ (\sigma_1(\langle e, st \rangle))$ which is the same as $(\sigma_3 \circ \sigma_2) \circ \sigma_1(\langle e, st \rangle)$.

For $(v : s) \in \Sigma.V$: $\sigma_2 \circ \sigma_1((v : s)) = \sigma_2(\sigma_1((v : s)))$ and so $\sigma_3 \circ (\sigma_2 \circ \sigma_1)((v : s)) = \sigma_3(\sigma_2(\sigma_1((v : s))))$ by the definition of composition. Similar to the above, this is also equal to $(\sigma_3 \circ \sigma_2) \circ \sigma_1((v : s))$.

Identity morphism for signatures: For any signature Σ, there exists an identity signature morphism $id_\Sigma : \Sigma \to \Sigma$.

id_E and id_V are such that $id_E(\langle e, st \rangle) = \langle e, st \rangle$ and $id_V((v : s)) = (v : s)$.This morphism satisfies the signature morphism condition since

$$\langle e, st \rangle \in \Sigma.E \Rightarrow id_E(\langle e, st \rangle) \in E \quad \wedge \quad ((v : s) \in \Sigma.V \Rightarrow id_V((v : s)) \in V$$

$\square$

B.2 The functor Sen, giving $\mathcal{EVT}$ sentences

In this section we prove that **Sen** is a functor that provides for each signature a set of sentences and for each signature morphism a function mapping the corresponding sentences. In particular, we prove that **Sen** preserves composition and identity of signature morphisms. The functor **Sen** describes the syntax of an $\mathcal{EVT}$-sentence over a specific vocabulary given by Σ.

Definition 17 ($\Sigma_{\mathcal{EVT}}$-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 $\overline{x}'$.*

Lemma 2. *There is a functor **Sen** : **Sign** $\rightarrow$ **Set** giving for each signature Σ a set of sentences (objects in the category **Set**) and for each signature morphism $\sigma : \Sigma_1 \rightarrow \Sigma_2$ (arrows in the category **Sign**) a function $Sen(\sigma) : Sen(\Sigma_1) \rightarrow Sen(\Sigma_2)$ (arrows in the category **Set**) translating sentences.*

Proof. This proof can be decomposed into three subcomponents as follows:

Sentence morphisms: **Sen** is a functor therefore it is necessary to map the signature morphisms to corresponding functions over sentences. The functor maps morphisms to sentence morphisms respecting sort, arity and initialisation events. The domain and codomain of $Sen(\sigma)$ are their respective images under σ.

Composition of sentence morphisms: We can show that

$$Sen(\sigma_2 \circ \sigma_1) = Sen(\sigma_2) \circ Sen(\sigma_1)$$

$Sen(\sigma_2) \circ Sen(\sigma_1)$ is the application of the signature morphism σ_1 to a sentence composed with the application of σ_2 and since signature morphisms can be composed this is the same as $Sen(\sigma_2 \circ \sigma_1)$.

Identity morphism for sentences: Let id_{Σ_1} be an identity signature morphism as defined in Lemma 1. Since signature morphisms already preserve identity and $Sen(id_{\Sigma_1})$ is the application of the identity signature morphisms to every element of the sentence, then the identities are preserved.

□

B.3 The functor Mod, giving $\mathcal{EVT}$ models

In this section we prove that every signature corresponds to a category of models with model morphisms as arrows (Lemma 3). Then, for each signature morphism we define the model reduct as a functor from models over one signature to models over another (Lemma 4). Finally, we prove that **Mod** is actually a functor (Lemma 5). The functor **Mod** formally defines the semantics of an Event-B specification by intuitively assigning values to all variables (including the after variables described during an event) as described below. We begin by providing a couple of definitions that will be required in order to carry out the proofs of the above mentioned lemmas.

Definition 18 (*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 19 (**$\Sigma_{\mathcal{EVT}}$-Model**) *Given $\Sigma = \langle S, \Omega, \Pi, E, V\rangle$, **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 $e \neq$ **`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 $\{\{\}\}$.

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

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:

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

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 $L = \{\{\}\}$.

Lemma 3. *For any signature Σ there is a category of models **Mod**(Σ) where the objects in the category are models and the arrows are model morphisms.*

Proof. There are three components to this proof as follows:

Model morphisms: In $\mathcal{FOPEQ}$ a model morphism $h : A_1 \to A_2$ is a family of functions $h = \langle h_s :: |A_1|_s \to |A_2|_s\rangle_{s\in S}$ which respects the sorts and arities of the operations and predicates. $\mathcal{EVT}$ models have the form $\langle A, L, R\rangle$ so $\mathcal{EVT}$-morphisms are given by the $\mathcal{FOPEQ}$-morphisms for A applied to the set L and the relation R.

Thus there is a model morphism $\mu : \langle A_1, L_1, R_1\rangle \to \langle A_2, L_2, R_2\rangle$ if there is a $\mathcal{FOPEQ}$-model morphism $h : A_1 \to A_2$ and we extend this to the states in the set L_1 and in the relation R_1. That is, for any element $\{b_1 \mapsto a_1, ..., b_n \mapsto a_n\} \in R.e_1$ in R_1 we have

$$\{h(b_1) \mapsto h(a_1), ..., h(b_n) \mapsto h(a_n)\} \in R.e_2$$

in R_2. A similar construction follows for $L1$. The composition of model morphisms, their associativity and identity derives from that of $\mathcal{FOPEQ}$.

Composition of model morphisms: Let $M_i = \langle A_i, L_i R_i \rangle$ be a model and $h_i : M_i \rightarrow M_{i+1}$ be a model morphism where $i \in \{1, 2, 3...\}$

Composition of model morphisms is associative:

$$\begin{aligned}
(h_3 \circ h_2) \circ h_1 &= h_3 \circ (h_2 \circ h_1) \\
(h_3 \circ h_2)(h_1(M_1)) &= h_3 \circ (h_2(h_1(M_1))) \\
(h_3 \circ h_2)(M_2) &= h_3 \circ (h_2(M_2)) \\
h_3(h_2(M_2)) &= h3(h_2(M_2)) \\
h_3(M_3) &= h_3(M_3) \\
M_4 &= M_4
\end{aligned}$$

Identity morphism for models: For any model M_i there exists an identity model morphism $h_{id} : M_i \rightarrow M_i$. If $M_i = \langle A_i, L_i, R_i \rangle$ then $h_{id}(M_i) = \langle A_i, L_i, R_i \rangle$

□

Lemma 4. *For each signature morphism $\Sigma : \Sigma_1 \rightarrow \Sigma_2$ the model reduct is a functor* ***Mod***(σ) *from Σ_2-models to Σ_1-models.*

Note that each Σ-state$_A$ is a set of variable-to-value mappings of the form

$$\{v_1 \mapsto val_1, \ldots, v_n \mapsto val_n\}$$

where $v_1, \ldots, v_n \in V$.

Proof. Let $M_2 = \langle A_2, L_2, R_2 \rangle$ be a Σ_2-model respectively. Then the reduct $M_2|_\sigma$ collapses the model to only contain signature items supported by Σ_1 and consists of the tuple $M_2|_\sigma = \langle A_2|_\sigma, L_2|_\sigma, R_2|_\sigma \rangle$ such that

- $A_2|_\sigma$ is the reduct of the $\mathcal{FOPEQ}$-component of the $\mathcal{EVT}$-model along the $\mathcal{FOPEQ}$-components of $\sigma : \Sigma \rightarrow \Sigma'$.
- $L_2|_\sigma$ and $R_2|_\sigma$ are based the reduction of the states of A_2 along σ.

Given $e \in dom(E1) \wedge e \neq$ `Initialisation` and $R.\sigma(e) \in R_2$

$$R.\sigma(e) = \{s_1, ..., s_m\}$$

where each s_i is a Σ_2-state$_{A_2}$ $(1 \leq i \leq m)$ is of the form

$$\{\sigma(v_1) \mapsto val_1, ..., \sigma(v_n) \mapsto val_n\}$$

with $v_1, ..., v_n \in V$.
Then for each $e \in dom(E1) \wedge e \neq$ `Initialisation` and $R.e \in R_2|_\sigma$ we have

$$R.e = \{s_1|_\sigma, ..., s_m|_\sigma\}$$

where each $s_i|_\sigma$ $(1 \leq i \leq m)$ is of the form

$$\{v_1 \mapsto val_1, ..., v_n \mapsto val_n\}$$

Preservation of composition for model reducts: Given model morphisms $h_1 : M_1 \to M_2$, $h_2 : M_2 \to M_3$: we must show $(h_2 \circ h_1)|_\sigma = h_2|_\sigma \circ h_1|_\sigma$.

For any $R.e \in R$, $(h_2 \circ h_1)|_\sigma = (h_2 \circ h_1)|_\sigma(R.e)$ which by definition of composition of morphisms is $(h_2)|_\sigma \circ ((h_1)|_\sigma(R.e))$ which equals $((h_2)|_\sigma \circ (h_1)|_\sigma)(R.e)$ which is $h_2|_\sigma \circ h_1|_\sigma$

Preservation of identities for model reducts: The reduct of the identity is the identity.

Let id_{M_2} be an identity Σ_2-morphism then $id_{M_2}|_\sigma$ is an identity Σ_1-morphism h_1 defined by $h_1(r_e) = id_{M_2}|_\sigma(r_e) = r_e$ for any $r_e \in R$ and $e \in dom(E) \wedge e \neq$ `Initialisation`.

For the components belonging to A these proofs follow the corresponding proofs in $\mathcal{FOPEQ}$. □

Lemma 5. *There is a functor* ***Mod*** *giving a category* ***Mod***(Σ) *of models for each signature* Σ*, and for each signature morphism* $\sigma : \Sigma_1 \to \Sigma_2$ *a functor* ***Mod***(σ) *from* Σ_2*-models to* Σ_1*-models.*

Proof. Proving that **Mod** is a functor:

For each $\sigma : \Sigma_1 \to \Sigma_2$ in **Sign** there is an arrow in $\mathbf{Sign}^{op}$ going in the opposite direction. By Lemma 4, the image of this arrow in $\mathbf{Sign}^{op}$ is $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma_2) \to \mathbf{Mod}(\Sigma_1)$ in **Cat**. By Lemma 3, the image of a signature **Sign** is an object $\mathbf{Mod}(\Sigma)$ in **Cat**. Therefore, domain and codomain of the image of an arrow are the images of the domain and codomain respectively.

Preservation of composition: $\mathbf{Mod}(\sigma_2 \circ \sigma_1) = \mathbf{Mod}(\sigma_2) \circ \mathbf{Mod}(\sigma_1)$
Let $\sigma_1 : \Sigma_1 \to \Sigma_2$ and $\sigma_2 : \Sigma_2 \to \Sigma_3$ be signature morphisms and let $M_i = \langle A_i, L_i, R_i \rangle$ be a model over Σ_i and let h_i be a Σ_i-model morphism. $i \in \{1, 2, 3\}$.

- $M_3|_{\sigma_2 \circ \sigma_1} = (M_3|_{\sigma_2})|_{\sigma_1}$
  By definition of reduct $M_3|_{\sigma_2} = \langle A_3, L_3, R_3 \rangle|_{\sigma_2} = \langle A_2, L_2, R_2 \rangle = M_2$.
  Then $(M_3|_{\sigma_2})|_{\sigma_1} = M_2|_{\sigma_1} = \langle A_2, L_2, R_2 \rangle|_{\sigma_1} = \langle A_1, L_1, R_1 \rangle = M_1$.
  By composition of signature morphisms $\sigma_2 \circ \sigma_1 : \Sigma_1 \to \Sigma_3$. So $M_3|_{\sigma_2 \circ \sigma_1} = \langle A_3, L_3, R_3 \rangle|_{\sigma_2 \circ \sigma_1} = \langle A_1, L_1, R_1 \rangle = M_1$
  Therefore $M_3|_{\sigma_2 \circ \sigma_1} = (M_3|_{\sigma_2})|_{\sigma_1}$

- $h_3|_{\sigma_2 \circ \sigma_1} = (h_3|_{\sigma_2})|_{\sigma_1}$
  Proof similar to above.

Preservation of identities: Let id_{Σ_1} be an identity signature morphism as defined in Lemma 1. Since signature morphisms already preserve identity and $Mod(id_{\Sigma_1})$ is the application of the identity signature morphisms to every part of the model, then the identities are preserved.

□

B.4 The Satisfaction relation for $\mathcal{EVT}$

This satisfaction relation is a relation between $\mathcal{EVT}$-sentences and $\mathcal{EVT}$-models. This satisfaction relation describes what it means for an $\mathcal{EVT}$-sentence to be satisfied using the variable to value mappings defined by any particular $\mathcal{EVT}$-model.

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 20 (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 $\langle$`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 3 (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.

B.5 Comorphism

We define the relationship between $\mathcal{EVT}$ and $\mathcal{FOPEQ}$ as a comorphism embedding of the simpler institution $\mathcal{FOPEQ}$ into the more complex institution $\mathcal{EVT}$.

Definition 21 *Given two institutions $INS = \langle Sign, Sen, Mod, \langle \models_\Sigma \rangle_{\Sigma \in |Sign|} \rangle$ and $INS' = \langle Sign', Sen', Mod', \langle \models'_{\Sigma'} \rangle_{\Sigma' \in |Sign'|} \rangle$. An institution comorphism $\rho : INS \rightarrow INS'$ consists of :*

- *a functor $\rho^{Sign} : Sign \rightarrow Sign'$.*
- *a natural transformation $\rho^{Sen} : Sen \rightarrow \rho^{Sign}; Sen'$ that is, for each $\Sigma \in |Sign|$, a function $\rho^{Sen}_\Sigma : Sen(\Sigma) \rightarrow Sen'(\rho^{Sign}(\Sigma))$ such that the following diagram commutes for every $\sigma : \Sigma_1 \rightarrow \Sigma_2$ in Sign.*

$$\begin{tikzcd}
\Sigma_2 & Sen(\Sigma_2) \arrow[r, "\rho^{Sen}_{\Sigma_2}"] \arrow[d, "Sen(\sigma)"] & Sen'(\rho^{Sign}(\Sigma_2)) \arrow[d, "Sen'(\rho^{Sign}(\sigma))"] \\
\Sigma_1 \arrow[u, "\sigma"] & Sen(\Sigma_1) \arrow[r, "\rho^{Sen}_{\Sigma_1}"] & Sen'(\rho^{Sign}(\Sigma_1))
\end{tikzcd}$$

- *a natural transformation $\rho^{Mod} : (\rho^{Sign})^{op}; Mod' \rightarrow Mod$ that is, for each $\Sigma \in |Sign|$, a functor $\rho^{Mod}_\Sigma : Mod'(\rho^{Sign}(\Sigma)) \rightarrow Mod(\Sigma)$ such that the following diagram commutes for every $\sigma : \Sigma_1 \rightarrow \Sigma_2$ in Sign.*

$$\begin{tikzcd}
\Sigma_2 & Mod'(\Sigma_2) \arrow[r, "\rho^{Mod}_{\Sigma_2}"] \arrow[d, "Mod'(\sigma)"] & Mod(\rho^{Sign}(\Sigma_2)) \arrow[d, "Mod(\rho^{Sign}(\sigma))"] \\
\Sigma_1 \arrow[u, "\sigma"] & Mod'(\Sigma_1) \arrow[r, "\rho^{Mod}_{\Sigma_1}"] & Mod(\rho^{Sign}(\Sigma_1))
\end{tikzcd}$$

such that for any $\Sigma \in |Sign|$, the translations $\rho^{Sen}_\Sigma : Sen(\Sigma) \rightarrow Sen'(\rho^{Sign}(\Sigma))$ of sentences and $\rho^{Mod}_\Sigma : Mod'(\rho^{Sign}(\Sigma)) \rightarrow Mod(\Sigma)$ of models preserve the satisfaction relation, that is, for any $\psi \in Sen(\Sigma)$ and $M' \in |Mod'(\rho^{Sign}(\Sigma)|$

$$\rho^{Mod}_\Sigma(M') \models_\Sigma \psi \iff M' \models'_{\rho^{Sign}(\Sigma)} \rho^{Sen}_\Sigma(\psi)$$

Based on this definition we define the comorphism between $\mathcal{FOPEQ}$ and $\mathcal{EVT}$ as follows:

Theorem 4. *$\rho : \mathcal{FOPEQ} \to \mathcal{EVT}$ is an institution comorphism which consists 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 $E = \{\langle \texttt{Init}\ \texttt{ordinary}\rangle\}$ 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 $\overline{x}'$. For each $\Sigma \in |Sign_{\mathcal{FOPEQ}}|$, a function $\rho^{Sen}_{\Sigma} : Sen_{\mathcal{FOPEQ}}(\Sigma) \to Sen_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))$ such that the following diagram commutes for every $\sigma : \Sigma_1 \to \Sigma_2$ in $Sign_{\mathcal{FOPEQ}}$.*

$$\begin{tikzcd}
\Sigma_2 & Sen_{\mathcal{FOPEQ}}(\Sigma_2) \arrow[r, "\rho^{Sen}_{\Sigma_2}"] \arrow[d, "Sen_{\mathcal{FOPEQ}}(\sigma)"] & Sen_{\mathcal{EVT}}(\rho^{Sign}(\Sigma_2)) \arrow[d, "Sen_{\mathcal{EVT}}(\rho^{Sign}(\sigma))"] \\
\Sigma_1 \arrow[u, "\sigma"] & Sen_{\mathcal{FOPEQ}}(\Sigma_1) \arrow[r, "\rho^{Sen}_{\Sigma_1}"] & Sen_{\mathcal{EVT}}(\rho^{Sign}(\Sigma_1))
\end{tikzcd}$$

- *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,*
  $$\rho^{Mod}_{\Sigma}(Mod(\rho^{Sign}(\Sigma))) = \rho^{Mod}_{\Sigma}(\langle A, L, \varnothing\rangle) = A$$
  For each $\Sigma \in |Sign_{\mathcal{FOPEQ}}|$, a functor $\rho^{Mod}_{\Sigma} : Mod_{\mathcal{EVT}}(\rho^{Sign}(\Sigma)) \to Mod_{\mathcal{FOPEQ}}(\Sigma)$ such that the following diagram commutes for every $\sigma : \Sigma_1 \to \Sigma_2$ in $Sign_{\mathcal{FOPEQ}}$.

$$\begin{tikzcd}
\Sigma_2 & Mod_{\mathcal{EVT}}(\Sigma_2) \arrow[r, "\rho^{Mod}_{\Sigma_2}"] \arrow[d, "Mod_{\mathcal{EVT}}(\sigma)"] & Mod_{\mathcal{FOPEQ}}(\rho^{Sign}(\Sigma_2)) \arrow[d, "Mod_{\mathcal{FOPEQ}}(\rho^{Sign}(\sigma))"] \\
\Sigma_1 \arrow[u, "\sigma"] & Mod_{\mathcal{EVT}}(\Sigma_1) \arrow[r, "\rho^{Mod}_{\Sigma_1}"] & Mod_{\mathcal{FOPEQ}}(\rho^{Sign}(\Sigma_1))
\end{tikzcd}$$

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 $M' \in |\mathbf{Mod}_{\mathcal{EVT}}(\rho^{Sign}(\Sigma)|$

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

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 20)

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

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. $\square$

For a specification written over $\mathcal{FOPEQ}$ we can use the specification-building operator $\_\_$ `with` $\rho : Spec_{\mathcal{FOPEQ}}(\Sigma) \rightarrow Spec_{\mathcal{EVT}}(\rho^{Sign}(\Sigma))$ where $\Sigma \in |\textbf{Sign}|$ to interpret $Spec_{\mathcal{FOPEQ}}(\Sigma)$ as a specification over $\mathcal{EVT}$.

C Modularisation: Pushouts and Amalgamation in $\mathcal{EVT}$

One of our primary aims is to use the theory of institutions in order to provide access to an array of formally defined modularisation constructs, namely *specification-building operators*, for Event-B. We have successfully represented Event-B in the institution $\mathcal{EVT}$ and in the theory of institutions, pushouts and amalgamation are required for any institution to have good modularity properties with respect to the specification building operators [12]. In fact, (weak) amalgamation properties are a required for good parametrisation behaviour [14]. In this section we prove that $\mathcal{EVT}$ has pushouts and the (weak) amalgamation property.

An institution has the weak amalgamation property if all pushouts in **Sign** exist and every pushout diagram in **Sign** admits weak amalgamation [14].

We split the proof of amalgamation into two lemmas as follows:

Lemma 6. *Pushouts exist in **Sign***

Proof. Given two signature morphisms $\sigma_1 : \Sigma \to \Sigma_1$ and $\sigma_2 : \Sigma \to \Sigma_2$ a pushout is a triple $(\Sigma', \sigma_1', \sigma_2')$ with $\sigma_1' \circ \sigma_1 = \sigma_2' \circ \sigma_2$ that satisfies the universal property: for every other such triple $(\Sigma'', \sigma_1'', \sigma_2'')$ with $\sigma_1'' \circ \sigma_1 = \sigma_2'' \circ \sigma_2$ there exists a unique morphism $u : \Sigma' \to \Sigma''$ such that the following diagram commutes:

$$\begin{tikzcd}
 & & \Sigma \arrow[dll, "\sigma_1"'] \arrow[drr, "\sigma_2"] & & \\
\Sigma_1 \arrow[drr, "\sigma_1'"] \arrow[ddrr, "\sigma_1''"'] & & & & \Sigma_2 \arrow[dll, "\sigma_2'"'] \arrow[ddll, "\sigma_2''"] \\
 & & \Sigma' \arrow[d, dashed, "u"] & & \\
 & & \Sigma'' & &
\end{tikzcd}$$

Given some signature $\Sigma = \langle S, \Omega, \Pi, E, V \rangle$, and signature morphisms $\sigma_1 : \Sigma \to \Sigma_1, \sigma_2 : \Sigma \to \Sigma_2$ we will construct the pushout $(\Sigma', \sigma_1', \sigma_2')$. Since $\mathcal{FOPEQ}$ admits amalgamation and is semi-exact, all pushouts exist in $\mathbf{Sign}_{\mathcal{FOPEQ}}$ and the **Mod** functor maps them to pullbacks in **Cat** [14]. Hence, our pushout construction follows $\mathcal{FOPEQ}$ for the elements that $\mathcal{FOPEQ}$ has in common with $\mathcal{EVT}$. In $\mathbf{Sig}_{\mathcal{EVT}}$ the only additional elements are E and V, and for each of these the pushouts are derived from **Set** those of $\mathcal{FOPEQ}$.

– *Set of* $\langle$*event name, status*$\rangle$ *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.

□

Lemma 7. *Every pushout diagram in **Sign** admits weak model amalgamation*

Consider the following diagram in **Sign**:

$$\begin{tikzcd}
 & \Sigma' & \\
\Sigma_1 \arrow[ur, "\sigma_1'"] & & \Sigma_2 \arrow[ul, "\sigma_2'"'] \\
 & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] &
\end{tikzcd}$$

This diagram admits weak model amalgamation if:

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

(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) $f' : M_1' \to M_2'$ in $\mathbf{Mod}(\Sigma')$ such that $f'|_{\sigma_1'} = f_1$ and $f'|_{\sigma_2'} = f_2$.

We handle both of these conditions separately by splitting this lemma further into two sublemmas:

Lemma 7(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:

$$\begin{tikzcd}
 & M' = \langle A', L', R' \rangle \arrow[dl, "Mod(\sigma_1')"'] \arrow[dr, "Mod(\sigma_2')"] & \\
M_1 = \langle A_1, L_1, R_1 \rangle \arrow[dr, "Mod(\sigma_1)"'] & & M_2 = \langle A_2, L_2, R_2 \rangle \arrow[dl, "Mod(\sigma_2)"] \\
 & M = \langle A, L, R \rangle &
\end{tikzcd}$$

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.

Specifically, starting from any $R.e = \{s_1, ..., s_m\} \in R$ where $s_1, ..., s_m$ are states of the form

$$\{v_1 \mapsto val_1, ..., v_n \mapsto val_n\}$$

we construct the corresponding $R'.\sigma'(e)$ in R' so that the following diagram commutes:

$$\begin{tikzcd}
 & R'.\sigma'(e) = \{\ldots,\{\sigma'(v_1) \mapsto h'(val_1),\ldots,\sigma'(v_n) \mapsto h'(val_n)\},\ldots\} \in R' \arrow[dl, "Mod(\sigma'_1)"'] \arrow[dr, "Mod(\sigma'_2)"] & \\
R_1.\sigma_1(e) = \{\ldots,\{\sigma_1(v_1) \mapsto h_1(val_1),\ldots,\sigma_1(v_n) \mapsto h_1(val_n)\},\ldots\} \in R1 \arrow[dr, "Mod(\sigma_1)"'] & & R_2.\sigma_2(e) = \{\ldots,\{\sigma_2(v_1) \mapsto h_2(val_1),\ldots,\sigma_2(v_n) \mapsto h_2(val_n)\},\ldots\} \in R2 \arrow[dl, "Mod(\sigma_2)"] \\
 & R.e = \{\ldots,\{v_1 \mapsto val_1,\ldots,v_n \mapsto val_n\},\ldots\} \in R &
\end{tikzcd}$$

Here $h' = (h_1 + h_2)$, the corresponding function over the carrier-sets in M' obtained from $\mathcal{FOPEQ}$, and $\sigma' = (\sigma_1' \circ \sigma_1) + (\sigma_2' \circ \sigma_2)$ the mapping for variable and event names obtained from the corresponding construction in **Sign**.

□

Lemma 7(b) *For any two model morphisms $f_1 : M_{11} \to M_{12}$ in $\textbf{\textit{Mod}}(\Sigma_1)$ and $f_2 : M_{21} \to M_{22}$ in $\textbf{\textit{Mod}}(\Sigma_2)$ 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 $\textbf{\textit{Mod}}(\Sigma')$, such that $f'|_{\sigma_1'} = f_1$ and $f'|_{\sigma_2'} = f_2$.*

Proof. Here, we are given the morphisms f_1 and f_2 and their common reduct f_0, and must construct f' so that the following diagram commutes:

$$\begin{tikzcd}
 & f' : M_1' \to M_2' \arrow[dl, "Mod(\sigma_1')"'] \arrow[dr, "Mod(\sigma_2')"] & \\
f_1 : M_{11} \to M_{12} \arrow[dr, "Mod(\sigma_1)"'] & & f_2 : M_{21} \to M_{22} \arrow[dl, "Mod(\sigma_2)"] \\
 & f_0 : M_{01} \to M_{02} &
\end{tikzcd}$$

Since each $\mathcal{EVT}$-model has a $\mathcal{FOPEQ}$ model as its first component, each of the $\mathcal{EVT}$-model morphisms f_0, f_1, f_2 and f' must have an underlying model morphism in $\mathcal{FOPEQ}$, which we denote f_0^-, f_1^-, f_2^- and f'^- respectively. To build the amalgamation for $\mathcal{EVT}$-models we must show how to extend these to cover the data states of the $\mathcal{EVT}$-models. This $\mathcal{EVT}$-model morphism follows the underlying $\mathcal{FOPEQ}$-model morphism on sort carrier sets for the values in the data states.

Given by $R.e \in R$, suppose we start with any f_0-maplet of the form

$$\begin{aligned}&\{\ldots, \{v_1 \mapsto val_1, \ldots, v_n \mapsto val_n\}, \ldots\}\\ &\mapsto \{\ldots, \{v_1 \mapsto f_0^-(val_1), \ldots, v_n \mapsto f_0^-(val_n)\}, \ldots\}_e \in f_0\end{aligned}$$

where f_0^- is the underlying map on data types from the $\mathcal{FOPEQ}$ model morphism.

Then the original two functions in f_1 and f_2 must have maplets of the form

$$\begin{aligned}&\{\ldots, \{\sigma_1(v_1) \mapsto h_1(val_1), \ldots, \sigma_1(v_n) \mapsto h_1(val_n)\}, \ldots\}\\ &\mapsto \{\ldots, \{\sigma_1(v_1) \mapsto f_1^-(h_1(val_1)), \ldots, \sigma_1(v_n) \mapsto f_1^-(h_1(val_n))\}, \ldots\} \in f_1\end{aligned}$$

and

$$\begin{aligned}&\{\ldots, \{\sigma_2(v_1) \mapsto h_2(val_1), \ldots, \sigma_2(v_n) \mapsto h_2(val_n)\}, \ldots\}\\ &\mapsto \{\ldots, \{\sigma_2(v_1) \mapsto f_2^-(h_2(val_1)), \ldots, \sigma_2(v_n) \mapsto f_2^-(h_2(val_n))\}, \ldots\} \in f_2\end{aligned}$$

where f_1^- and f_2^- are again the data type maps from the underlying $\mathcal{FOPEQ}$ model morphism, and h_1 and h_2 are obtained from $Mod(\sigma_1)$ and $Mod(\sigma_2)$.

We then can construct the elements of the model morphism f', which is the amalgamation of f_1 and f_2, as f'-maplets of the form:

$$\begin{aligned}&\{\ldots, \{\sigma'(v_1) \mapsto h'(val_1), \ldots, \sigma'(v_n) \mapsto h'(val_n)\}, \ldots\} \\ &\mapsto \{\ldots, \{\sigma'(v_1) \mapsto f'^-(h'(val_1)), \ldots, \sigma'(v_n) \mapsto f'^-(h'(val_n))\}, \ldots\} \ \in \ f'\end{aligned}$$

As before, $h' = (h_1 + h_2)$, the corresponding function over the carrier-sets in M' obtained from $\mathcal{FOPEQ}$, and $\sigma' = (\sigma_1' \circ \sigma_1) + (\sigma_2' \circ \sigma_2)$ the mapping for variable and event names obtained from the corresponding construction in **Sign**. Here $f^{'-} = f_1^- + f_2^-$ is the amalgamation from the corresponding diagram for model morphisms in $\mathcal{FOPEQ}$, which ensures that the data states are mapped to corresponding states in the model M_2'.

□

D An Institution-Based Semantics for Event-B

D.1 Syntax of Event-B

Our objective is to define a translational semantics for Event-B by representing Event-B specifications as specifications over $\mathcal{EVT}$, our institution for Event-B [9]. There are two basic languages in Event-B, these are the Event-B mathematical language (propositional/predicate logic, set-theory and arithmetic) and the Event-B modelling language [2]. We decompose the Event-B modelling language into two further languages; the infrastructure language and the superstructure language. In order to translate Event-B into the institutional framework we divide the constructs of the Event-B language into three layers, each layer corresponding to one of its three component languages, as shown in Figure 7.

- At the base of Figure 7 is the Event-B mathematical language. The institution for first order predicate logic with equality, $\mathcal{FOPEQ}$, is embedded via comorphism into the institution for Event-B, $\mathcal{EVT}$. The semantics that we have defined translates the constructs of this mathematical language into corresponding constructs over $\mathcal{FOPEQ}$.
- At the next level is the Event-B infrastructure, which consists of those language elements used to define variables, invariants, variants and events. These are translated into sentences over $\mathcal{EVT}$.
- At the topmost level is the Event-B superstructure which deals with the definition of Event-B machines and contexts, as well as their relationships (refines, sees, extends). These are translated into presentations over $\mathcal{EVT}$.

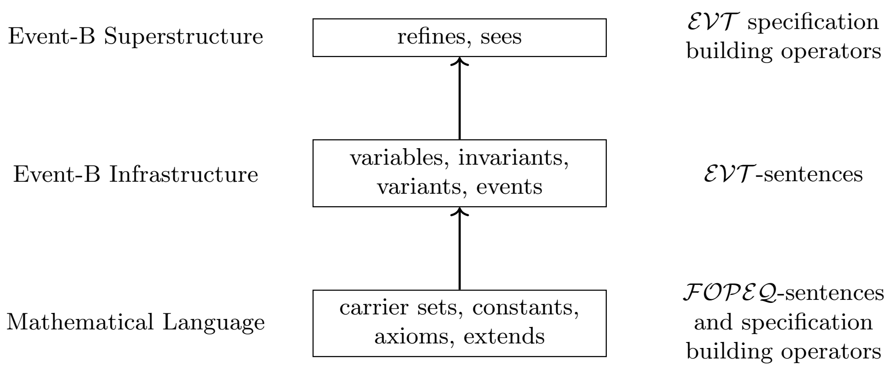
Event-B Superstructure
refines, sees
$\mathcal{EVT}$ specification building operators
Event-B Infrastructure
variables, invariants, variants, events
$\mathcal{EVT}$-sentences
Mathematical Language
carrier sets, constants, axioms, extends
$\mathcal{FOPEQ}$-sentences and specification building operators

Fig. 7: We split the Event-B syntax into three components: superstructure, infrastructure and a mathematical language

The abstract syntax for Event-B is described briefly in [2] and we provide a more detailed version in Figure 8. A *Specification* consists of any number of *Machine* and *Context* definitions. The nonterminals *predicate* and *expression* are not defined in Figure 8. These are part of the Event-B mathematical language, and in our translation these syntactic elements will be supplied by $\mathcal{FOPEQ}$, the institution for first order predicate logic with equality, with predicates corresponding to $\mathcal{FOPEQ}$-formulae and expressions corresponding to $\mathcal{FOPEQ}$-terms.

Both machines and contexts allow the user to specify *theorems* which are used to generate proof obligations. Since these must be consequences of the specification and do not add any constraints, we omit them from further discussion here. We order things in a slightly different manner to the standard in Event-B in that we use *MachineBody*,*EventBody* and *ContextBody* to refer to the non-superstructure elements of a machine, event or context.

Based on the syntax defined in Figure 8, we define the semantics of each of the Event-B infrastructure sentences by describing a mechanism to translate them into $\mathcal{EVT}$-sentences. In order to carry out such a translation we first extract the corresponding $\mathcal{EVT}$ signature $\Sigma = \langle S, \Omega, \Pi, E, V \rangle$ from any given Event-B specification. Contexts can be represented entirely by the underlying mathematical language and thus translated into specifications over $\mathcal{FOPEQ}$. In section D.2, we define the interface in Figure 9 in order to facilitate the use of some $\mathcal{FOPEQ}$ operations and semantic functions.

D.2 A $\mathcal{FOPEQ}$ Interface

The Event-B formalism is parametrised by an underlying mathematical language and $\mathcal{EVT}$, our institution for Event-B is parametrised by $\mathcal{FOPEQ}$, the institution for first-order predicate logic with equality in Figure 9. We define a $\mathcal{FOPEQ}$ interface in order to facilitate the use of its operations and semantic functions within our semantic definition of Event-B using $\mathcal{EVT}$. The description of the operations defined in Figure 9 is contained within the figure.

The semantic function $\mathbb{P}_\Sigma$ described in Figure 9 takes a labelled predicate and outputs a $\mathcal{FOPEQ}$-sentence (*Σ-formula*). The semantic function $\mathbb{T}_\Sigma$ takes an expression and returns a *Σ-term*. These functions are used later to translate Event-B predicates and expressions into *Σ-formulae* and *Σ-terms* respectively.

The purpose of the semantic function $\mathbb{M}$ is to take two lists of identifiers and a list of labelled predicates and, use these to form the $\mathcal{FOPEQ}$ signature $\langle S, \Omega, \Pi \rangle$. The reason for this is that when extracting a signature from a context (described in Figure 12) carrier sets are interpreted as sorts and used to form S. The constants and axioms are used to form Ω and Π. We assume that $\mathbb{M}$ provides this translation.

For simplicity, we assume that it is possible to use Event-B identifiers in $\mathcal{FOPEQ}$ and $\mathcal{EVT}$. Also, when we reference a *Σ-formula* in Figure 9 we mean a possibly open $\mathcal{FOPEQ}$-formula over the signature given by Σ. We only return a closed $\mathcal{FOPEQ}$-formula when applying $\mathbb{P}_\Sigma$ to axiom sentences since they form closed predicates in Figure 12.

Specification	::=	(*Machine* $\mid$ *Context*)$^+$
Machine	::=	`machine` *identifier*
		[`refines` *identifier*]
		[`sees` *identifier*$^+$]
		MachineBody
		`end`
MachineBody	::=	`variables` *identifier*$^+$
		`invariants` *LabelledPredicate**
		[`theorems` *LabelledPredicate*$^+$]
		[`variant` *expression*]
		`events` *InitEvent Event**
LabelledPredicate	::=	*label* : *predicate*
InitEvent	::=	`event Initialisation`
		`status ordinary`
		[`then` *LabelledPredicate*]
		`end`
Event	::=	`event` *identifier*
		`status` *Stat*
		[`refines` *identifier*$^+$]
		EventBody
		`end`
EventBody	::=	[`any` *identifier*$^+$]
		[`where` *LabelledPredicate*]
		[`with` *LabelledPredicate*]
		[`then` *LabelledPredicate*]
Stat	::=	`ordinary` $\mid$ `convergent` $\mid$ `anticipated`
Context	::=	`context` *identifier*
		[`extends` *identifier*$^+$]
		ContextBody
		`end`
ContextBody	::=	[`sets` *identifier*$^+$]
		[`constants` *identifier*$^+$]
		[`axioms` *LabelledPredicate*$^+$]
		[`theorems` *LabelledPredicate*$^+$]
identifier	::=	`String`
label	::=	`String`

Fig. 8: The Event-B syntax is parametrised by first order logic as indicated by our use of the nonterminals *predicate* and *expression*. These will be mapped to $\mathcal{FOPEQ}$-formulae and terms respectively.

$\mathcal{FOPEQ}$ **Operations**

- `F.and` : Σ*-formula** $\rightarrow$ Σ*-formula*
  This corresponds to the logical conjunction ($\wedge$) of formulae in $\mathcal{FOPEQ}$ which results in the formation of a formula.

- `F.lt` : Σ*-formula* $\times$ Σ*-formula* $\rightarrow$ Σ*-formula*
  This operation takes two formulae and returns a formula corresponding to arithmetic less than ($<$).

- `F.leq` : Σ*-formula* $\times$ Σ*-formula* $\rightarrow$ Σ*-formula*
  This operation takes two formulae and returns a formula corresponding to arithmetic less than or equal to ($\leq$).

- `F.exists` : $identifier^*$ $\times$ Σ*-formula* $\rightarrow$ Σ*-formula*
  This operation takes a sequence of identifiers and a formula and returns a formula corresponding to the existential quantification of the identifiers over the input formula.

- `F.`ι : $identifier^*$ $\rightarrow$ Σ*-formula* $\rightarrow$ Σ*-formula*
  This operation takes a list of identifiers and formula and returns the input formula with the names of all the free variables (as given by the list of identifiers) primed.

$\mathcal{FOPEQ}$ **Semantic Functions**

- $\mathbb{P}_{\Sigma}$: *LabelledPredicate* $\rightarrow$ Σ*-formula*

- $\mathbb{T}_{\Sigma}$: *expression* $\rightarrow$ Σ*-term*

- $\mathbb{M} : identifier^* \times identifier^* \times LabelledPredicate^* \rightarrow |\mathbf{Sign}_{\mathcal{FOPEQ}}|$

Fig. 9: The $\mathcal{FOPEQ}$ interface provides access to a range of operations and semantic functions which we assume to exist. These are used throughout our semantic definitions in figures 10, 11 and 12.

$$Env \;= Id \rightarrow |\mathbf{Sign}|$$

$$
\begin{array}{ll}
\mathbb{D}: & Specification \rightarrow Env \rightarrow Env \\
\mathbb{D} & [\![\langle\ \rangle]\!]\ \xi = \xi \\
\mathbb{D} & [\![hd :: tl]\!]\ \xi = \mathbb{D}\ ([\![tl]\!])\ (\mathbb{D}\ [\![hd]\!]\ \xi)
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{D}: & Machine \rightarrow Env \rightarrow Env \\
\mathbb{D} & \left[\!\!\left[\begin{array}{l}\texttt{machine}\ m \\ \texttt{refines}\ a \\ \texttt{sees}\ ctx_1, \ldots, ctx_n \\ mbody \\ \texttt{end}\end{array}\right]\!\!\right] \xi = \xi \cup \{[\![m]\!] \mapsto (\langle S, \Omega, \Pi, E, V\rangle \cup r(\xi[\![a]\!]))\} \\
 & where \\
 & \quad \langle S, \Omega, \Pi\rangle = \{(\xi\ [\![ctx_1]\!]) \cup\ \ldots\ \cup (\xi\ [\![ctx_n]\!])\} \\
 & \quad \langle E, V, RA\rangle = \mathbb{D}[\![mbody]\!] \\
 & \quad r : |\mathbf{Sign}| \rightarrow |\mathbf{Sign}| \\
 & \quad r(\xi[\![a]\!]) = \texttt{let}\ \ \Sigma_a = \xi[\![a]\!]\ \ \texttt{in}\ \ \langle \Sigma_a.S, \Sigma_a.\Omega, \Sigma_a.\Pi, RA \lhd \Sigma_a.E, \Sigma_a.V\rangle
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{D}: & MachineBody \rightarrow \langle E, V, \{identifier\}\rangle \\
\mathbb{D} & \left[\!\!\left[\begin{array}{l}\texttt{variables}\ v_1, \ldots, v_n \\ \texttt{invariants}\ i_1, \ldots, i_n \\ \texttt{theorems}\ t_1, \ldots, t_n \\ \texttt{variant}\ n \\ \texttt{events}\ e_{init}\ e_1, \ldots, e_n\end{array}\right]\!\!\right] = \langle E, V, RA\rangle \\
 & where \\
 & \quad E = \{def[\![e_{init}]\!], def[\![e_1]\!], \ldots, def[\![e_n]\!]\} \\
 & \quad V = \langle [\![v_1]\!], \ldots, [\![v_n]\!]\rangle \\
 & \quad RA = ref[\![e_{init}]\!] \cup ref[\![e_1]\!] \cup \ldots \cup ref[\![e_n]\!]
\end{array}
$$

$$
\begin{array}{ll}
def : & Event \rightarrow identifier \times Stat \\
def & [\![\texttt{event}\ e, \texttt{status}\ s, \texttt{refines}\ e_1, \ldots, e_n, \cdots \texttt{end}]\!] = \langle [\![e]\!], s\rangle \\
def & [\![e_{init}]\!] = \langle \texttt{Initialisation}, \texttt{ordinary}\rangle
\end{array}
$$

$$
\begin{array}{ll}
ref : & Event \rightarrow \{identifier\} \\
ref & [\![\texttt{event}\ e, \texttt{status}\ s, \texttt{refines}\ e_1, \ldots, e_n, \cdots \texttt{end}]\!] = \{[\![e_1]\!], \ldots, [\![e_n]\!]\} \\
ref & [\![e_{init}]\!] = \{[\![e_{init}]\!]\}
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{D}: & Context \rightarrow Env \rightarrow Env \\
\mathbb{D} & \left[\!\!\left[\begin{array}{l}\texttt{context}\ ctx \\ \texttt{extends}\ ctx_1, \ldots, ctx_n \\ cbody \\ \texttt{end}\end{array}\right]\!\!\right] \xi = \xi \cup ([\![ctx]\!] \mapsto (\mathbb{D}[\![cbody]\!] \cup \xi[\![ctx_1]\!] \cup\ \ldots\ \cup \xi[\![ctx_n]\!]))
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{D}: & ContextBody \rightarrow |\mathbf{Sign}_{\mathcal{FOPEQ}}| \\
\mathbb{D} & \left[\!\!\left[\begin{array}{l}\texttt{sets}\ s_1, \ldots, s_n \\ \texttt{constants}\ c_1, \ldots, c_n \\ \texttt{axioms}\ a_1, \ldots, a_n \\ \texttt{theorems}\ t_1, \ldots, t_n\end{array}\right]\!\!\right] = \langle S, \Omega, \Pi\rangle \\
 & where \\
 & \quad \langle S, \Omega, \Pi\rangle = \mathbb{M}(([\![s_1]\!], \ldots, [\![s_n]\!]), ([\![c_1]\!], \ldots, [\![c_n]\!]), ([\![a_1]\!], \ldots, [\![a_m]\!]))
\end{array}
$$

Fig. 10: The semantics of signature extraction uses the interface described in Figure 9 in order to extract signature components from the definition of a *ContextBody*. Context signatures are over $\mathcal{FOPEQ}$ and machine signatures are over $\mathcal{EVT}$.

D.3 Extracting the Signature Σ

We define an environment Env to map machine/context names to signatures, since, due to the superstructure components, machines/contexts can refer to other machines/contexts. In all further definitions we use ξ to denote an environment as defined by Env. We define the overloaded semantic function $\mathbb{D}$ in Figure 10 to extract the environment from a given specification.

We map $\mathbb{D}$ through the list of machines and contexts that make up an Event-B specification. $\mathbb{D}$ extracts the signature from machines and contexts. The function def extracts the pair (event name, status) for each event in the machine and these pairs form the E component of the signature. The status is paired with each event name in order to correctly form variant sentences which will be discussed in Section D.5. The function ref forms the set of events that a particular concrete event refines. We use this function to remove the names of the refined abstract events, and the status that each is paired with, from the abstract machine's signature before we combine it with the concrete signature. We use the domain anti-restriction operator $\lhd$ to achieve this.

Once we have formed the environment, we can then define a systematic translation from specifications in Event-B to specifications over $\mathcal{EVT}$. We take a top-down approach to this translation which is comprised of two parts.

- The first semantic mapping (in Figure 11) that we provide is from the superstructure components of an Event-B specification to presentations over $\mathcal{EVT}$ (for machines) and presentations over $\mathcal{FOPEQ}$ (for contexts) in section D.4.
- The second semantic mapping (in Figure 12) that we define is from the Event-B infrastructure sentences (invariants, variants, events and axioms) to sentences over $\mathcal{EVT}$ (for invariants, variants and events) and sentences over $\mathcal{FOPEQ}$ (for axioms) in section D.5.

D.4 Defining the Semantics of Event-B Superstructure sentences using $\mathcal{EVT}$

Based on the syntax defined in Figure 8 we have identified a number of constructs that form the Event-B superstructure language, these are:

- `extends` $context\_identifier^+$
- `refines` $machine\_identifier$
- `sees` $context\_identifier^+$
- `refines` $event\_identifier^+$

In this section, we define a semantics for the Event-B superstructure language using specification building operators. In Figure 11 we define the semantic function $\mathbb{B}$ to translate Event-B specifications written using the superstructure language to presentations over $\mathcal{EVT}$ that use the specification building operators defined in the theory of institutions [14].

We translate a specification as described by Figure 8 into a *presentation* over the institution $\mathcal{EVT}$.

Definition 22 (Presentation) *For any Σ, a Σ-presentation is a pair $\langle \Sigma, \Phi \rangle$ where $\Phi \subseteq \boldsymbol{Sen}(\Sigma)$. $M \in |\boldsymbol{Mod}(\Sigma)|$ is a model of a Σ-presentation $\langle \Sigma, \Phi \rangle$ if $M \models \Phi$ [14].*

The construct that enables a context to extend others is used in Event-B to add more details to a context. Since a context in Event-B only refers to elements of the $\mathcal{FOPEQ}$ component of an $\mathcal{EVT}$ signature it is easy to see that we can translate this using the specification building operator then. then is used to enrich the signature with new sorts/operations etc [14]. It is possible to extend more than one context. In this case the resulting context contains all constants and axioms of all extended contexts and the additional specification of the extending context itself [8]. To give a semantics for this using the specification building operators we and all extended contexts and use then to incorporate the extending context itself. The specification building operator and takes the sum of two specifications that can be written over different signatures. It is the most straight forward way to combine specifications over different signatures [14].

In Event-B machines may see contexts. This construct is used to add a context(s) to a machine so that the machine can refer to elements of the context(s). We have shown that the relationship between $\mathcal{FOPEQ}$ and $\mathcal{EVT}$ is that of a comorphism. This enables us to directly use $\mathcal{FOPEQ}$-sentences, as given by the context in this case, in an $\mathcal{EVT}$-presentation. We use the specification building operation with ρ which indicates translation by an institution comorphism ρ [14]. The resulting machine specification is heterogeneous as it links two institutions by an institution comorphism.

An Event-B machine can refine at most one other machine and there are two types of machine refinement: superposition and data refinement [8]. then accounts for both of these types of refinement because either new signature components or constraints on the data (gluing invariants) are added to the specification. In Figure 11 the semantic function $\mathbb{A}_\Sigma$ is used to process the events in the concrete machine which refine those in the abstract machine.

Event refinement in Event-B is superposition refinement [8]. By superposition refinement all of the components of the corresponding abstract event are implicitly included in the refined version. This approach is useful for gradually adding more detail to the event. In $\mathcal{EVT}$, we have not prohibited multiple definitions of the same event name. When there are multiple definitions we combine them by taking the conjunction of their respective formulae which will constrain the model. As mentioned above, when refining an abstract machine we use the semantic function $\mathbb{A}_\Sigma$, in Figure 11 to process the events in the concrete machine which refine those in the abstract machine. $\mathbb{A}_\Sigma$ in turn calls the semantic function $\mathbb{R}_\Sigma$. $\mathbb{R}_\Sigma$ gets the abstract machine signature and restricts its event component to those events contained in the `refines` clause of the event definition using domain restriction. This new signature is included in the abstract via the signature morphism σ_h. We then form the signature morphism σ_m which is the identity on the sort, operation, predicate and variable components of Σ_h. σ_m maps each of the abstract event signature components that are being refined by the concrete event e_c to the signature component corresponding to e_c. The

For an Event-B specification denoted by SP, the environment is given by $\xi = \mathbb{D}[\![SP]\!]\xi_0$ *where* ξ_0 *is the empty environment.*

$$
\begin{array}{ll}
\mathbb{B}: & \mathit{Specification} \to \mathit{Env} \to \langle|\mathbf{Pres}|\rangle \\
\mathbb{B} & [\![\langle\ \rangle]\!]\ \xi = \langle\ \rangle \\
\mathbb{B} & [\![hd :: tl]\!]\ \xi = (\mathbb{B}\ [\![hd]\!]\ \xi) :: (\mathbb{B}\ [\![tl]\!]\ \xi)
\end{array}
$$

$$\mathbb{B}: \quad \mathit{Machine} \to \mathit{Env} \to |\mathbf{Pres}_{\mathcal{EVT}}|$$

$$
\mathbb{B}\ \left[\!\!\left[\begin{array}{l}
\texttt{machine}\ m \\
\texttt{refines}\ a \\
\texttt{sees}\ ctx_1, \ldots, ctx_n \\
mbody \\
\texttt{end}
\end{array}\right]\!\!\right] \xi = \left\langle \Sigma, \left[\begin{array}{l}
\texttt{spec}\ [\![m]\!]\ \texttt{over}\ \mathcal{EVT} = \\
([\![ctx_1]\!]\ \texttt{and}\ \ldots\ \texttt{and}\ [\![ctx_n]\!])\ \texttt{with}\ \rho \\
(\texttt{and}\ \ \mathbb{A}_{\Sigma}[\![mbody]\!][\![a]\!]\xi)^{*} \\
\ \texttt{then} \\
\quad \mathbb{S}_{\Sigma}[\![mbody]\!] \\
\mathit{where} \\
\quad \rho : \mathcal{FOPEQ} \to \mathcal{EVT}
\end{array}\right] \right\rangle
$$

$$\mathit{where}\ \Sigma = \xi[\![m]\!]$$

*only included if the **refines** clause is nonempty

$$\mathbb{A}_{\Sigma}: \mathit{MachineBody} \to \mathit{identifier} \to \mathit{Env} \to \mathit{Sen}(\Sigma)$$

$$
\mathbb{A}_{\Sigma}\ \left[\!\!\left[\begin{array}{l}
\texttt{variables}\ v_1, \ldots, v_n \\
\texttt{invariants}\ i_1, \ldots, i_n \\
\texttt{theorems}\ t_1, \ldots, t_n \\
\texttt{variant}\ n \\
\texttt{events}\ e_{init},\ e_1, \ldots, e_n
\end{array}\right]\!\!\right] [\![a]\!]\xi = \mathbb{R}_{\Sigma}[\![e_1]\!][\![a]\!]\xi\ \texttt{and}\ \ldots\ \texttt{and}\ \mathbb{R}_{\Sigma}[\![e_n]\!][\![a]\!]\xi
$$

$$\mathbb{R}_{\Sigma}: \mathit{Event} \to \mathit{identifier} \to \mathit{Env} \to \mathit{Sen}(\Sigma)$$

$$
\mathbb{R}_{\Sigma}\ \left[\!\!\left[\begin{array}{l}
\texttt{event}\ e_c \\
\texttt{status}\ s \\
\texttt{refines}\ e_1, \ldots, e_n \\
ebody \\
\texttt{end}
\end{array}\right]\!\!\right] [\![a]\!]\xi = \begin{array}{l}
\texttt{let} \\
\ \Sigma_a = \xi[\![a]\!], \\
\ \Sigma_h = \langle \Sigma_a.S, \Sigma_a.\Omega, \Sigma_a.\Pi, \{[\![e_1]\!], \ldots, [\![e_n]\!]\} \lhd \Sigma_a.E, \Sigma_a.V\rangle, \\
\ \sigma_h : \Sigma_h \hookrightarrow \Sigma_a, \\
\ \sigma_m : \Sigma_h \to \Sigma \\
\ \sigma_m = \langle \Sigma_h.S \hookrightarrow \Sigma.S, \Sigma_h.\Omega \hookrightarrow \Sigma.\Omega, \Sigma_h.\Pi \hookrightarrow \Sigma.\Pi, \\
\qquad\qquad\qquad \Sigma_h.E \mapsto \{[\![e_c]\!]\} \lhd \Sigma.E, \Sigma_h.V \hookrightarrow \Sigma.V\rangle \\
\texttt{in} \\
\ ([\![a]\!]\ \texttt{hide via}\ \sigma_h)\ \texttt{with}\ \sigma_m
\end{array}
$$

$$\mathbb{B}: \quad \mathit{Context} \to \mathit{Env} \to |\mathbf{Pres}_{\mathcal{FOPEQ}}|$$

$$
\mathbb{B}\ \left[\!\!\left[\begin{array}{l}
\texttt{context}\ ctx \\
\texttt{extends}\ ctx_1, \ldots, ctx_n \\
cbody \\
\texttt{end}
\end{array}\right]\!\!\right] \xi = \left\langle \Sigma, \left[\begin{array}{l}
\texttt{spec}\ [\![ctx]\!]\ \texttt{over}\ \mathcal{FOPEQ} = \\
[\![ctx_1]\!]\ \texttt{and}\ \ldots\ \texttt{and}\ [\![ctx_n]\!] \\
\ \texttt{then} \\
\quad \mathbb{S}_{\Sigma}[\![cbody]\!]
\end{array}\right] \right\rangle
$$

$$\mathit{where}\ \Sigma = \xi[\![ctx]\!]$$

Fig. 11: We define the semantics of Event-B superstructure sentences by translating them into presentations over $\mathcal{EVT}$ using the semantic function $\mathbb{B}$ and the specification building operators defined in the theory of institutions. Note that objects of **Pres** are of the form $\langle \Sigma, \Phi \rangle$ for a signature Σ and $\Phi \subseteq \mathit{Sen}(\Sigma)$.

resulting sentence uses hide via and with to apply these signature morphisms (σ_h and σ_m) in the correct way.

D.5 Defining the Semantics of Event-B Infrastructure sentences using $\mathcal{EVT}$

In this section we define a translation from Event-B infrastructure sentences to sentences over $\mathcal{EVT}$. We translate the axiom sentences that are found in Event-B contexts to sentences over $\mathcal{FOPEQ}$ as they form part of the underlying Event-B mathematical language in Figure 7.

We define an overloaded meaning function, $\mathbb{S}_\Sigma$, for specifications in Figure 12. $\mathbb{S}_\Sigma$ takes as input a specification and returns a set of sentences over $\mathcal{EVT}$ ($Sen_{\mathcal{EVT}}(\Sigma)$) for machines and a set of sentences over $\mathcal{FOPEQ}$ ($Sen_{\mathcal{FOPEQ}}(\Sigma)$) for contexts. When applying $\mathbb{S}_\Sigma$ to a machine (resp. context) we also define semantic functions for processing invariants, variants and events (resp. axioms). These are given by $\mathbb{I}_\Sigma, \mathbb{V}_\Sigma$ and $\mathbb{E}_\Sigma$ in Figure 12. Axioms are predicates that can be translated into closed $\mathcal{FOPEQ}$-formulae using the semantic function $\mathbb{P}_\Sigma$ which is defined in the interface in Figure 9.

Given a list of invariants $i_1, \ldots, i_n$ we define the semantic function $\mathbb{I}_\Sigma$ in Figure 12. Each invariant, i, is a *LabelledPredicate* from which we form the open $\mathcal{FOPEQ}$-sentence $\texttt{F.and}(\mathbb{P}_\Sigma[\![i]\!], \texttt{F.}\iota(\Sigma.V)(\mathbb{P}_\Sigma[\![i]\!]))$. Each invariant sentence is paired with each event name e where $(e, s) \in E$ and s is the status corresponding to event e. This is due to the fact that invariants in Event-B are shared globally by all events in a machine.

Given a variant expression n, we define the semantic function $\mathbb{V}_\Sigma$ in Figure 12. The variant is only relevant for specific events so we pair it with an event name in order to meaningfully evaluate the variant expression. An event whose status is `convergent` must strictly decrease the variant expression. An event whose status is `anticipated` must not increase the variant expression. This expression can be translated into an open $\mathcal{FOPEQ}$-term using the semantic function $\mathbb{T}_\Sigma$ as described in Figure 9. From this we form a formula based on the status of the event(s) in the signature Σ. Event-B machines are only permitted to have one variant [8].

In Figure 12 we define the semantic function $\mathbb{E}_\Sigma$ to process a given event definition. Event guard(s) and witnesses are predicates that can be translated via $\mathbb{P}_\Sigma$ into open $\mathcal{FOPEQ}$-formulae denoted by G and W respectively in Figure 12. In Event-B, actions are interpreted as before-after predicates e.g. $x := x + 1$ is interpreted as $x' = x + 1$. Therefore, actions can also be translated via $\mathbb{P}_\Sigma$ into open $\mathcal{FOPEQ}$-formulae denoted by BA in Figure 12. Thus the semantics of an *EventBody* definition is given by the semantic function $\mathbb{F}_\Sigma$ which returns the formula `F.exists` (p, `F.and`(G, W, BA)) where p are the event parameters.

A context can exist independently of a machine and is written as a specification over $\mathcal{FOPEQ}$. Thus, we translate an axiom sentence directly as a $\mathcal{FOPEQ}$-sentence which is a closed Σ-formula using the semantic function $\mathbb{P}_\Sigma$ given in Figure 9. Axiom sentences are closed $\mathcal{FOPEQ}$-formulae (elements of

$$\mathbb{S}_\Sigma : \quad \mathit{MachineBody} \to \mathit{Sen}_{\mathcal{EVT}}(\Sigma)$$

$$\mathbb{S}_\Sigma \quad \left[\!\!\left[\begin{array}{l}\texttt{variables } v_1, \ldots, v_n \\ \texttt{invariants } i_1, \ldots, i_n \\ \texttt{theorems } t_1, \ldots, t_n \\ \texttt{variant } n \\ \texttt{events } e_{init},\ e_1, \ldots, e_n\end{array}\right]\!\!\right] = \left(\begin{array}{l}\mathbb{I}_\Sigma[\![i_1]\!] \cup \ldots \cup \mathbb{I}_\Sigma[\![i_n]\!] \cup \mathbb{V}_\Sigma[\![n]\!] \\ \cup\ \ \mathbb{E}_\Sigma[\![e_{init}]\!] \cup \mathbb{E}_\Sigma[\![e_1]\!] \cup \ldots \cup \mathbb{E}_\Sigma[\![e_n]\!]\end{array}\right)$$

$$\mathbb{I}_\Sigma : \quad \mathit{LabelledPredicate} \to \mathit{Sen}_{\mathcal{EVT}}(\Sigma)$$
$$\mathbb{I}_\Sigma[\![i]\!] \quad = \{\langle [\![e]\!], \texttt{F.and}(\mathbb{P}_\Sigma[\![i]\!], \texttt{F.}\iota(\Sigma.V)(\mathbb{P}_\Sigma[\![i]\!]))\rangle \mid (e, s) \in E\}$$

$$\mathbb{V}_\Sigma : \quad \mathit{expression} \to \mathit{Sen}_{\mathcal{EVT}}(\Sigma)$$
$$\mathbb{V}_\Sigma[\![n]\!] = \begin{array}{l}\{\langle [\![e]\!], \texttt{F.lt}(\texttt{F.}\iota(\Sigma.V)(\mathbb{T}_\Sigma[\![n]\!]), \mathbb{T}_\Sigma[\![n]\!])\rangle \mid (e, \texttt{convergent}) \in E\} \\ \cup\{\langle [\![e]\!], \texttt{F.leq}(\texttt{F.}\iota(\Sigma.V)(\mathbb{T}_\Sigma[\![n]\!]), \mathbb{T}_\Sigma[\![n]\!])\rangle \mid (e, \texttt{anticipated}) \in E\}\end{array}$$

$$\mathbb{E}_\Sigma : \quad \mathit{InitEvent} \to \mathit{Sen}_{\mathcal{EVT}}(\Sigma)$$

$$\mathbb{E}_\Sigma : \quad \left[\!\!\left[\begin{array}{l}\texttt{event Initialisation} \\ \texttt{status ordinary} \\ \texttt{then } act_1, \ldots, act_n \\ \texttt{end}\end{array}\right]\!\!\right] = \{\langle \texttt{Initialisation}, BA\rangle\}$$

where

$BA = \texttt{F.and}(\mathbb{P}_\Sigma[\![act_1]\!], \ldots, \mathbb{P}_\Sigma[\![act_n]\!])$

$$\mathbb{E}_\Sigma : \quad \mathit{Event} \to \mathit{Sen}_{\mathcal{EVT}}(\Sigma)$$

$$\mathbb{E}_\Sigma : \quad \left[\!\!\left[\begin{array}{l}\texttt{event } e \\ \texttt{status } s \\ \texttt{refines } e_1, \ldots, e_n \\ \mathit{ebody} \\ \texttt{end}\end{array}\right]\!\!\right] = \{\langle [\![e]\!], \mathbb{F}_\Sigma[\![\mathit{ebody}]\!]\rangle\}$$

$$\mathbb{F}_\Sigma : \quad \mathit{EventBody} \to \Sigma\text{-}\mathit{formula}$$

$$\mathbb{F}_\Sigma : \quad \left[\!\!\left[\begin{array}{l}\texttt{any } p_1, \ldots, p_n \\ \texttt{where } grd_1, \ldots, grd_n \\ \texttt{with } w_1, \ldots, w_n \\ \texttt{then } act_1, \ldots, act_n\end{array}\right]\!\!\right] = \texttt{F.exists}(p, \texttt{F.and}(G, W, BA))$$

where

$p = \langle [\![p_1]\!], \ldots, [\![p_n]\!]\rangle$
$G = \texttt{F.and}(\mathbb{P}_\Sigma[\![grd_1]\!], \ldots, \mathbb{P}_\Sigma[\![grd_n]\!])$
$W = \texttt{F.and}(\mathbb{P}_\Sigma[\![w_1]\!], \ldots, \mathbb{P}_\Sigma[\![w_n]\!])$
$BA = \texttt{F.and}(\mathbb{P}_\Sigma[\![act_1]\!], \ldots, \mathbb{P}_\Sigma[\![act_n]\!])$

$$\mathbb{S}_\Sigma : \quad \mathit{ContextBody} \to \mathit{Sen}_{\mathcal{FOPEQ}}(\Sigma)$$

$$\mathbb{S}_\Sigma \quad \left[\!\!\left[\begin{array}{l}\texttt{sets } s_1, \ldots, s_n \\ \texttt{constants } c_1, \ldots, c_n \\ \texttt{axioms } a_1, \ldots, a_n \\ \texttt{theorems } t_1, \ldots, t_n\end{array}\right]\!\!\right] = \{\mathbb{P}_\Sigma[\![a_1]\!], \ldots, \mathbb{P}_\Sigma[\![a_n]\!]\}$$

Fig. 12: We provide a semantics for Event-B infrastructure sentences by translating them into sentences over $\mathcal{EVT}$, denoted $\mathit{Sen}_{\mathcal{EVT}}(\Sigma)$, for machines and sentences over $\mathcal{FOPEQ}$, denoted $\mathit{Sen}_{\mathcal{FOPEQ}}(\Sigma)$, for contexts. We use the interface operations and semantic functions described in Figure 9 throughout this translation.

$Sen_{\mathcal{FOPEQ}}(\Sigma)$) which are interpreted as a valid sentences in $\mathcal{EVT}$ using the comorphism ρ.

References

1. CASL reference manual. In P. D. Mosses, editor, *The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *LNCS*. 2004.
2. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, New York, NY, USA, 1st edition, 2010.
3. 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.
4. 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.
5. 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.
6. J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
7. 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.
8. M. Jastram and P. M. Butler. *Rodin User's Handbook: Covers Rodin V.2.8.* CreateSpace Independent Publishing Platform, USA, 2014.
9. A. Kleppe. *Software language engineering: creating domain-specific languages using metamodels.* Pearson Education, 2008.
10. 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.
11. 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.
12. T. Mossakowski and M. Roggenbach. Structured CSP - a process algebra as an institution. In *Recent Trends in Algebraic Development Techniques*, volume 4409 of *LNCS*, pages 92–110. 2007.
13. M. Roggenbach. Csp-casla new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.
14. D. Sanella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development.* Springer, 2012.

References

1. CASL reference manual. In P. D. Mosses, editor, *The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *LNCS*. 2004.
2. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, New York, NY, USA, 1st edition, 2010.
3. 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.
4. 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.
5. 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.
6. J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
7. 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.
8. M. Jastram and P. M. Butler. *Rodin User's Handbook: Covers Rodin V.2.8.* CreateSpace Independent Publishing Platform, USA, 2014.
9. A. Kleppe. *Software language engineering: creating domain-specific languages using metamodels.* Pearson Education, 2008.
10. 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.
11. 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.
12. T. Mossakowski and M. Roggenbach. Structured CSP - a process algebra as an institution. In *Recent Trends in Algebraic Development Techniques*, volume 4409 of *LNCS*, pages 92–110. 2007.
13. M. Roggenbach. Csp-casla new integration of process algebra and algebraic specification. *Theoretical Computer Science*, 354(1):42–71, 2006.
14. D. Sanella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development.* Springer, 2012.