Modularising and Promoting Interoperability for Event-B Specifications using Institution Theory

Marie Farrell, Rosemary Monahan & James F. Power

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

Introduction & Motivation

Event-B is an industrial-strength language for system-level modelling and verification that combines an event-based logic with basic set theory.

- Event-B supports formal refinement, which allows a developer to write an abstract specification of a system and gradually add complexity.
- The *Rodin Platform*, an IDE for Event-B, ensures the safety of system specifications and refinement steps by generating appropriate proof-obligations, and then discharging these via support for various theorem provers [2].

Limitations of Event-B

Modularity: Event-B lacks well-developed modularisation constructs and it is not easy to combine specifications in Event-B with those written in other formalisms [1]. Notice how, in Figure 1 the same specification has to be provided twice. The events `set_peds_go` and `set_peds_stop` are equivalent, modulo renaming of variables, to `set_cars_go` and `set_cars_stop`.

Interoperability: When developing software using Event-B, it is at least necessary to transform the final concrete specification into a different language to get an executable implementation. Current approaches to interoperability in Event-B consist of a range of *Rodin*-based plugins to translate to/from Event-B, but these often lack a solid logical foundation.

Adding Event-B to the theory supermarket

- We have identified the theory of institutions as a suitable metalogical framework in which to provide a specification of the Event-B specification language.
- In order to represent a formalism/logic using institutions, the syntax and semantics for the formalism must first be defined and verified in a uniform way using some basic constructs from category theory [3].
- It is necessary to verify that the resulting metalogical structure is actually a valid institution. This is ensured by proving the satisfaction condition which states in formal terms the basic maxim of institutions, that "truth is invariant under change of notation".

Building an Institution for Event-B, $\mathcal{EVT}$

Our institution, $\mathcal{EVT}$, for Event-B consists of the following definitions:

- A signature over $\mathcal{EVT}$ describes the permitted vocabulary to use when writing Event-B specifications, consisting of names for sorts, operations, predicates, events and variables. Signature morphisms provide a mechanism for moving between vocabularies and mapping the corresponding sentences and models in a similar fashion.
- A sentence over $\mathcal{EVT}$ is an Event-B specification written using this vocabulary. Such sentences can be evaluated in a model.
- An $\mathcal{EVT}$ model consists of possible before-after value pairs for each variable in each event.

Further details and proofs can be found on our website:
`http://www.cs.nuim.ie/~mfarrell`

A Modular Traffic Light System

By defining $\mathcal{EVT}$ and carrying out the appropriate proofs, we gain access to an array of generic specification building operators [3]. These facilitate the combination (and, $+$, $\cup$), extension (then), hiding (hide via, reveal) and renaming via signature morphism (with) of specifications. 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 2 is a presentation (set of sentences) over the institution $\mathcal{EVT}$ corresponding to the Event-B machine `mac1` defined in Figure 1.

Our Contributions

Modularity: Representing Event-B in this way provides us with a mechanism for combining and parameterising specifications. Most importantly, these constructs are formally defined, a crucial issue for a language used in formal modelling.

Interoperability: Institution comorphisms can be defined enabling us to move between different institutions, thus providing a mechanism by which a specification written over one institution can be represented as a specification over another. Devising meaningful institutions and corresponding morphisms to/from Event-B provides a mechanism for not only ensuring the safety of a particular specification but also, via morphisms, a platform for integration with other formalisms and logics.

```
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
       act1: cars_go := false
       act2: peds_go := false
  Event set_peds_go ≙
    when  grd1: cars_go = false
    then  act1: peds_go := true
  Event set_peds_stop ≙
       act1: peds_go := false
  Event set_cars_go ≙
    when  grd1: peds_go = false
    then  act1: cars_go := true
  Event set_cars_stop ≙
       act1: cars_go := false
```

Fig. 1: Event-B machine specification for a traffic system, with each light controlled by boolean flags.

```
spec TwoBools over FOPEQ
  Bool then
    ops     i_go, u_go : Bool
    preds   ¬ (i_go = true ∧ u_go = true)
spec LightAbstract over EVT
  TwoBools then
    Initialisation
         act1 : i_go := false
     Event set_go ≙
      when grd1: u_go = false
      then act1: i_go := true
     Event set_stop ≙
      then act1: i_go := false
spec mac1 over EVT
  (LightAbstract with σ1) and (LightAbstract with σ2)
  where
  σ1 = {i_go ↦ cars_go, u_go ↦ peds_go,
     set_go ↦ set_cars_go, set_stop ↦ set_cars_stop}
  σ2 = {i_go ↦ peds_go, u_go ↦ cars_go,
     set_go ↦ set_peds_go, set_stop ↦ set_peds_stop}
```

Fig. 2: A modular institution-based presentation corresponding to the abstract machine `mac1` in Fig 1.

[1] 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.

[2] M. Jastram and P. M. Butler. *Rodin User's Handbook: Covers Rodin V.2.8*. CreateSpace Independent Publishing Platform, USA, 2014.

[3] D. Sanella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Springer, 2012.