Formalizing the Institution for Event-B in the Coq Proof Assistant

Conor Reynolds(✉)

Maynooth University, Maynooth, Kildare, Ireland
conor.reynolds@mu.ie

Abstract. We formalize a fragment of the theory of institutions sufficient to establish basic facts about the institution *EVT* for Event-B, and its relationship with the institution *FOPEQ* for first-order predicate logic. We prove the satisfaction condition for *EVT* and encode the institution comorphism *FOPEQ* → *EVT* embedding *FOPEQ* in *EVT*.

Keywords: Coq · Event-B · Institution theory

1 Introduction

The theory of institutions [4] was introduced by Joseph Goguen and Rod Burstall to give concrete form to the informal notion of a "logical system", identifying a common structure among the many logics in regular use in computer science. A 2017 paper by Marie Farrell, Rosemary Monahan, and James Power [3] uses the theory of institutions to provide a sound mathematical semantics and modularization constructs for the industrial-strength state-based formal modelling language Event-B [1], providing interoperability with other formalisms. In related work, the Heterogeneous Tool Set (Hets) [7] makes use of institutions to provide heterogeneous specifications.

Event-B has an associated development process for system-level modelling and analysis. Key features include the use of set theory as a modelling notation, the use of refinement to represent systems at different abstraction levels and the use of mathematical proof to verify consistency between refinement levels. The primary purpose of this research is to formalize the work in [3] within the Coq proof assistant, and more generally to provide the rudiments of a Coq library for the theory of institutions.

We build on earlier work formalizing universal algebra in Agda by Emmanuel Gunther, Alejandro Gadea, and Miguel Pagano [5]. However, the purpose of this work is not to provide a comprehensive development of universal algebra; we only develop as much as we need in order to define the institutions for first-order logic and Event-B. We also depend on the development of category theory by John Wiegley at jwiegley/category-theory.

Supported by the Irish Research Council (GOIPG/2019/4529).

While some obligations remain to be formally discharged for the institution *FOPEQ* for first-order predicate logic with equality, our developments for the institution *EVT* for Event-B are complete. We have also encoded the institution comorphism $FOPEQ \rightarrow EVT$, which embeds the simpler *FOPEQ* institution into *EVT*, providing the underlying mathematical language for *EVT*. It remains, however, to prove the naturality condition in our encoding. The formalization is not axiom-free, assuming dependent function extensionality and proof irrelevance. A more careful development might use setoids (as in [2,5]), and in the future we may experiment with grounding these efforts in homotopy type theory.

Throughout this paper, we will assume some familiarity with basic category theory, as well as the first two chapters of [8].

2 The Institution for Event-B

An ***institution*** [4] consists of

- a category Sig of signatures (non-logical syntax);
- a sentence functor $\mathsf{Sen}\colon \mathsf{Sig} \rightarrow \mathsf{Set}$ (logical syntax);
- a model functor $\mathsf{Mod}\colon \mathsf{Sig}^{\mathrm{op}} \rightarrow \mathsf{Cat}$ (semantics for non-logical syntax); and
- a semantic entailment relation $\models_\Sigma \subseteq |\mathsf{Mod}(\Sigma)| \times \mathsf{Sen}(\Sigma)$ for each $\Sigma \in \mathsf{Sig}$,

such that for any signature translation $\sigma\colon \Sigma \rightarrow \Sigma'$, any sentence $\phi \in \mathsf{Sen}(\Sigma)$, and any model $M' \in \mathsf{Mod}(\Sigma')$, the satisfaction condition holds:

$$M' \models_{\Sigma'} \mathsf{Sen}(\sigma)(\phi) \quad \text{iff} \quad \mathsf{Mod}(\sigma)(M') \models_\Sigma \phi \tag{1}$$

This kind of institution is sometimes referred to as a set/cat institution, since the target of Sen is Set and the target of Mod is Cat. To avoid encoding a "category of categories" in Coq, we implement set/set institutions [6].

We will now provide a precise but brief definition for the institution for Event-B, alongside its definition in Coq. For details, we refer the reader to [3]. Throughout, let $\mathsf{Status} = \{\mathsf{ordinary} \leq \mathsf{anticipated} \leq \mathsf{convergent}\}$.

The category of *EVT*-signatures has as objects $\hat{\Sigma} = \langle \Sigma, E, X, X' \rangle$, where Σ is a first-order signature, $E : \mathsf{Status} \rightarrow \mathsf{Type}$ is a status-indexed set of events, and $X, X' : \mathsf{sorts}\ \Sigma \rightarrow \mathsf{Type}$ are sorts-indexed sets of pre- and post-variables, respectively. In Coq, this becomes:

```
Record EvtSignature :=
  { base_sig :> FOSig ;
    events : Status → Type ;
    Vars   : sorts base_sig → Type ;
    Vars'  : sorts base_sig → Type }.
```

An *EVT*-signature morphism $\hat{\Sigma}_1 \rightarrow \hat{\Sigma}_2$ consists of a first-order signature morphism $\sigma\colon \Sigma_1 \rightarrow \Sigma_2$ translating the base signature, along with a function $E_1 \rightarrow E_2$ mapping events in such a way as to preserve the ordering on statuses, and functions $X_1 \rightarrow X_2 \circ \sigma$, $X'_1 \rightarrow X'_2 \circ \sigma$ mapping variables, regarded as morphisms in their respective indexed categories. It is convenient to assume that the initialization event is not in E, so there is no need for the assumption that the initial event is preserved by signature morphisms. If the initialization/event distinction is made at the level of sentences, then we can enforce preservation of the initialization event definitionally.

```
Record EvtSigMorphism Σ Σ' : Type :=
  { on_base_sig :> SignatureMorphism Σ Σ' ;
    on_events : EventMorphism Σ Σ' ;
    on_vars   : Vars Σ → Vars Σ' ∘ on_base_sig ;
    on_vars'  : Vars' Σ → Vars' Σ' ∘ on_base_sig }.
```

EVT-sentences are either *initialization* sentences, $\mathsf{Init}\ \psi$ where $\psi : \mathsf{FOSen}(\Sigma + X')$, or *event* sentences, $\mathsf{Event}\ e\ \psi$ where $\psi : \mathsf{FOSen}(\Sigma + X + X')$. Note that the base signature is expanded to include the EVT-variables as constant operation names. Initialization sentences describe how variables are initially set. Event sentences describe how events change the variables. As a very simple example, given an event inc which increments a variable n, $\mathsf{inc} :\equiv$ **begin** $n := n + 1$ **end**, we write the EVT-sentence $\mathsf{Event}(\mathsf{inc}, n' = n + 1)$, where $n \in X$ and $n' \in X'$ are respectively pre- and post-variables from the ambient Event-B signature. Given an initialization event which starts n at 0, $\mathsf{init} :\equiv$ **begin** $n := 0$ **end**, we write the EVT-sentence $\mathsf{Init}(n' = 0)$. For details on this correspondence, see again [3].

Event-B sentences rely on the ability to construct the expansion of first-order signatures by adjoining a sorts-indexed set of constant operation names, which in Coq we denote by $\mathsf{SigExpand}\ \Sigma\ X$. EVT-sentences can be defined as follows.

```
Inductive EVT Σ : Type :=
| Init  : FOSen (SigExpand Σ (Vars' Σ)) → EVT Σ
| Event : ∀ status, events Σ status
    → FOSen (SigExpand Σ (Vars Σ + Vars' Σ))
    → EVT Σ.
```

An EVT-model consists of a first-order model M and a pair of environments $L : \mathsf{List}(X' \to M)$ and $R : E \to \mathsf{List}(X + X' \to M)$, which are lists of valuations of variables in M. We enforce that L and R_e, for each event e, are nonempty.

```
Record EvtModel Σ :=
  { base_alg :> Algebra Σ ;
    envL : NEList (Vars' Σ → base_alg) ;
    envR : ∀ status,
      events Σ status → NEList (Vars Σ + Vars' Σ → base_alg) }.
```

Let M^θ denote the expansion of a model M by a valuation $\theta : X \to M$. We say that $\langle M, L, R\rangle \models \mathsf{Init}\ \psi$ if for all valuations $\theta \in L$, we have $M^\theta \models \psi$, and we say that $\langle M, L, R\rangle \models \mathsf{Event}\ e\ \psi$ if for all valuations $\theta \in R_e$ we have $M^\theta \models \psi$. This can be written down directly in Coq.

```
Definition interp_evt Σ M ϕ : Prop :=
  match ϕ with
  | Init ψ    => List.Forall (λ θ, AlgExpansion M θ ⊨ ψ) (envL M)
  | Event e ψ => List.Forall (λ θ, AlgExpansion M θ ⊨ ψ) (envR M e)
  end.
```

Now, taking a top-down perspective, we can define institutions in Coq as follows:

```
Class Institution :=
  { Sig : Category ;
    Sen : Sig → SetCat ;
    Mod : Sig^op → SetCat ;
    interp : ∀ Σ : Sig, Mod Σ → Sen Σ → Prop ;

    satisfaction : ∀ (Σ Σ' : Sig) (σ : Σ → Σ')
                     (ϕ : Sen Σ) (M' : Mod Σ'),
      interp M' (fmap[Sen] σ ϕ) ↔ interp (fmap[Mod] σ M') ϕ }.
```

Proving that EVT is an institution amounts to instantiating this class to the above definitions and discharging the generated obligations. The proofs rely on custom induction principles for the dependent records we introduce above, since the induction principles generated by Coq are too strong. For example, if one wishes to prove that two Event-B signature morphisms $\hat{\sigma}$ and $\hat{\sigma}'$ are equal, of course it suffices to prove that they are equal componentwise. Consider equality on the on_vars component. The statement of this equality will depend on a proof $p : \sigma = \sigma'$ that the underlying first-order signature morphisms are equal, which we write $p_*(\mathsf{on\_vars}\ \hat{\sigma}) = \mathsf{on\_vars}\ \hat{\sigma}'$. Notice that this requirement is substantially stronger than necessary; it suffices in this case to know that σ and σ' agree on sorts. Hence, given $p' : \mathsf{on\_sorts}\ \sigma = \mathsf{on\_sorts}\ \sigma'$, we only need to prove $p'_*(\mathsf{on\_vars}\ \hat{\sigma}) = \mathsf{on\_vars}\ \hat{\sigma}'$. This dramatically simplifies the proofs.

3 Future Work

In the future, it will be interesting to investigate Coq's code extraction facilities to generate provably correct code derived from, for example, the institution comorphism $FOPEQ \rightarrow EVT$. We also wish to prove the amalgamation property for EVT, and more generally to build institution-independent constructions and proofs, which we have already explored to some extent for modal logics and linear-time temporal logics. The proofs involved in the definition for first-order predicate logic were rather complicated, but the proofs for EVT often reduced to properties of first-order logic. This suggests that quick progress could be made defining further institutions, verifying their properties, and providing interoperability between represented formalisms represented in our framework.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Capretta, V.: Universal algebra in type theory. In: Bertot, Y., Dowek, G., Théry, L., Hirschowitz, A., Paulin, C. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 131–148. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48256-3_10
3. Farrell, M., Monahan, R., Power, J.F.: An institution for Event-B. In: James, P., Roggenbach, M. (eds.) WADT 2016. LNCS, vol. 10644, pp. 104–119. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72044-9_8

4. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM **39**(1), 95–146 (1992)
5. Gunther, E., Gadea, A., Pagano, M.: Formalization of universal algebra in Agda. Electron. Notes Theor. Comput. Sci. **338**, 147–166 (2018)
6. Mossakowski, T., Goguen, J., Diaconescu, R., Tarlecki, A.: What is a logic? In: Beziau, J.Y. (ed.) Logica Universalis, pp. 111–133. Birkhäuser Basel (2007)
7. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, HETS. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_40
8. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Springer-Verlag (2012)