# Reasoning about complex requirements in a uniform setting*

Manuel A. Martins[1], Alexandre Madeira[2,1,3], Luís S. Barbosa[2]

[1] Department of Mathematics, University of Aveiro
[2] Department of Informatics & CCTC, Minho University
[3] Critical Software S.A., Portugal

**Abstract.** The paper formulates $\mathcal{HEQ}$, an institution for hybrid equational logic to provide a uniform setting to express and reasoning about different sorts of properties of complex software. It is also shown how, through the definition of a suitable comorphism to $\mathcal{FOL}$, this can be integrated in Hets, providing suitable tool support for teaching and research. The whole exercise was motivated by the need to unify, in a single undergraduate course in a Computer Science curriculum, the specification of data and behavioural constraints of reconfigurable systems.

## 1 The problem

Fundamental infrastructures of modern societies, including those related to financial, health, education, energy and water supply, are critically based on information systems, which are assumed to be trustworthy. Moreover, our way of living depends on software whose reliability is crucial for our own work, security, privacy, and quality of life. This explains why the quest for programs whose correctness could be established by mathematical reasoning, which has been around for a long time as a research agenda, has recently emerged as a key concern for industry, who is becoming aware of the essential role played by proofs and the associated relevance given to formal logic. At present, at least in what concerns safety-critical systems, *proofs pay the rent*: they are no more an academic activity or an exotic detail, but simply part of the business.

But software is large and complex, deals with a multitude of different concerns, has to meet requirements formulated (and verified) at different abstraction levels. A basic distinction is drawn between *behavioural* and *data* aspects. The former relates to mechanisms (e.g., *processes*) which control manipulation of data. While processes are dynamic and active, data is static and passive. Typically, the emergent behaviour of a software system is determined by the concurrent execution of several processes which exchange data in order to influence each other's behaviour.

* The authors manifest their gratitude to Răzvan Diaconescu for his suggestions and support in the preparation of this paper. This work was partially supported by *FCT*, under contract `PTDC/EIA-CCO/108302/2008`, *Centro de Investigação e Desenvolvimento em Matemática e Aplicações* of University of Aveiro, and doctoral grant `SFRH/BDE/33650/2009` supported by *FCT* and *Critical Software S.A., Portugal*.

Mathematically, this symmetry between *data* and *behavioural* structures can be traced down to the duality between *initial* algebras and *final* coalgebras, which provide their abstract descriptions [14]. From an educational point of view, although disguised in a number of different designations, both approaches are part of a typical Computer Science undergraduate curricula: abstract behavioural structures are usually studied in a Process Algebra course (often on top of a previous course on languages and automata); abstract data structures are covered in algebraic specification courses. The latter are typically concerned with the concept of *abstract data type*, entailing a family of methods [6,15] which constitutes a large and mature body of knowledge and active research in the triple dimension of foundations, methodologies and applications.

These two approaches are usually kept separated in the curriculum. Even if a number of attempts to integrate data and behaviour specifications do exist, as in LOTOS [9] or mCRL2 [8], they are often introduced as inhabitants of different galaxies, dealing with orthogonal problems through essentially different methods.

But such a lack of integration inside the curriculum is not the only problem. Actually, most approaches to software modeling, based either on an algebraic or coalgebraic perspective, are 'static' in the sense that the specification fixes the component semantics once and for all. In most cases, however, and most typically in service-oriented applications, what a software component may offer at each stage may depend on its own evolution and history. That is to say, software components are often *evolving structures* which may change from on mode of operation to another, entailing corresponding updates in what counts, at each mode or stage, as a valid description of their behaviour.

Can a rigorous discipline of software development, able not only to combine data and behavioural issues, but also to deal appropriately with systems evolution and reconfiguration, be devised for teaching at undergraduate level? Such is the problem addressed in this paper. It comes from a concrete context: the reorganization of undergraduate degrees in Computer Science motivated by the implementation of the Bologna Agreement in Portugal. This entailed the split of traditional 5-years courses in Bachelor (3 years) and Master (2 years) degrees. The latter are usually vertical in specific domains of Computer Science. Bachelor degrees, on the other hand, entailed the need for integrating courses in core curricular areas (such as software specification and design) which requires the introduction of methodologies with a *common background* and reasonable *tool support* for increased experimental work.

A suitable answer to this challenge has to proceed at two levels: that of general enough semantical structures, on the one hand, and of expressive logics to capture properties of such structures, on the other. The approach proposed in this paper characterizes an institution [7,4] for *hybrid equational logic*, which enriches a classical modal setting with the ability to reference (properties of) specific points in the system space state. This entails a powerful specification logic endowed with a suitable class of models, implicitly capturing algebraic and coalgebraic properties, and a satisfaction relation. Such an institutional rendering, which is new to the best of our knowledge, pays off in terms of tool support for specifications, as discussed below.

## 2 The approach: *states-as-algebras* and hybrid languages

*The setting.* From a didactical point of view the problem students are supposed to deal with by the end of a course in Software Specification is that of modeling and reasoning *reconfigurable components*. These are components which may evolve in time through a number of different stages or modes of operation, to which correspond different configurations of the services made available through its interface. Each *configuration* is specified axiomatically as an *algebraic theory*; its model being a concrete algebra satisfying such a theory. The component evolution, on the other hand, is modeled by a transition system: a configuration changes in response to a particular event in the system. Both aspects are taken into account in the definition of a hybrid model in the following section.

The envisaged logic to express requirements on such structures, on the other hand, has to deal with global and local properties. The former are essentially *modal*, to capture the component evolution through different configurations. The latter should be able to refer to specific states in the system and characterizing the semantics of operations at each stage.

Modal logic is not enough as it does not allow explicit references to specific states. Hybrid logic [1], however, overcomes this limitation by introducing symbols, called *nominals* to reference states, i.e., in our case, to identify component's configurations. This is achieved through a family of connectives $@_i$, indexed by nominals $i$: intuitively $@_i\, p$ states the validity of $p$ at the state identified by nominal $i$. The syntax of the *equational hybrid logic*, discussed in the following section, is given by

$$WFF := \; i \,|\, t = t' \,|\, \neg\varphi |[\lambda]\varphi \,|\, @_i\varphi \,|\, \varphi \wedge \psi \,|\, \varphi \vee \psi \,|\, \varphi \rightarrow \psi$$

where $\lambda$ ranges over modal operators. The logic can be seen as a fragment of hybrid first-order logic obtained by taking equations as extra atoms instead of all first-order formulas (cf. [2]). Note that the usual propositional variables are implicitly considered as special equations.

*A specification example.* A small, elementary example may help to illustrate the kind of specifications we want to be able to deal with. Consider a calculator with two states, say the +-state and the ×-state, on which an operation denoted by the $\star$ symbol stands, respectively, for sum or multiplication of natural numbers. Additionally, the calculator exhibits another operation, $shift$, that leads from one configuration to the other.

This calculator may be viewed as a transition system that alternates between + and ×-states by the application of $shift$. Each of its states is associated to a $\Sigma$-algebra, where $\Sigma$ is the one-sorted signature consisting of one sort $\{nat\}$ and the following set of operation symbols $\{0 :\rightarrow nat; suc : nat \rightarrow nat; p : nat \rightarrow nat; \star : nat \times nat \rightarrow nat\}$.

Considering $\Lambda = \{\Box\}$ and $\mathrm{Nom} = \{\times, +\}$, to denote the + and ×-states, we are able to express *local* properties like $@_+ \star(n, 0) \approx n$, $@_+ \star(n, suc(0)) \approx suc(n)$, $@_\times \star(n, 0) \approx 0$ and $@_\times \star(n, suc(0)) \approx n$. Modal or transition properties, on the other hand, resort to $\Lambda$. For example, $\Box + \leftrightarrow \times$ and $\star(n, 0) \approx n \to \Box \star(n, 0) \approx 0$.

*Going 'institutional'.* Dealing with this sort of specifications entails the need for a *uniform* specification framework in which both equational properties of data types, modal properties of transitions and local properties of states can be expressed and verified. The canonical way to do it is through the notion of an *institution* [7,4], as an abstract representation of a logical system, encompassing syntax, semantics and satisfaction. Let us recall here the formal definition: An *institution* $\big(\mathrm{Sign}^{\mathcal{I}}, \mathrm{Sen}^{\mathcal{I}}, \mathrm{Mod}^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma \in |\mathrm{Sign}^{\mathcal{I}}|}\big)$ consists of

- a category $\mathrm{Sign}^{\mathcal{I}}$ whose objects are called *signatures*.
- a functor $\mathrm{Sen}^{\mathcal{I}} : \mathrm{Sign}^{\mathcal{I}} \to \mathsf{Set}$ giving for each signature a set whose elements are called *sentences* over that signature.
- a functor $\mathrm{Mod}^{\mathcal{I}} : (\mathrm{Sen}^{\mathcal{I}})^{op} \to \mathsf{CAT}$, giving for each signature $\Sigma$ a category whose objects are *$\Sigma$-models*, and whose arrows the corresponding *$\Sigma$-morphisms*, and
- a *satisfaction relation* $\models^{\mathcal{I}}_{\Sigma} \subseteq |\mathrm{Mod}^{\mathcal{I}}(\Sigma)| \times \mathrm{Sen}^{\mathcal{I}}$ for each $\Sigma \in |\mathrm{Sen}^{\mathcal{I}}|$.

such that for each morphism $\varphi : \Sigma \to \Sigma' \in \mathrm{Sign}^{\mathcal{I}}$ , the satisfaction condition

$$M' \models^{\mathcal{I}}_{\Sigma'} \mathrm{Sen}^{\mathcal{I}}(\varphi)(\rho) \text{ iff } \mathrm{Mod}^{\mathcal{I}}(\varphi)(M') \models^{\mathcal{I}}_{\Sigma} \rho \tag{1}$$

holds for each $M' \in |\mathrm{Mod}^{\mathcal{I}}(\Sigma')|$ and $\rho \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$. A well-known example, upon which $\mathcal{HEQ}$ will be built in the sequel, is $\mathcal{EQ} = \big(\mathrm{Sign}^{\mathcal{EQ}}, \mathrm{Sen}^{\mathcal{EQ}}, \mathrm{Mod}^{\mathcal{EQ}}, (\models^{\mathcal{EQ}}_{\Sigma})_{\Sigma \in |\mathrm{Sign}^{\mathcal{EQ}}|}\big)$, the institution of equational logic.

Institutions provide a suitable setting to do *abstract specification theory* [16], structuring any kind of specifications through combinators which are institution-independent, i.e. not tied to a specific logic system. In Casl [12], for example, such combinators allow the construction of basic specifications, by defining a signature and a set of sentences, the union of specifications, and the derivation and translation of specifications along signature morphisms. The use of this set of (abstract) combinators, allows to approach, in a uniform way and trough the same theory, systems expressed in completely different logics. Naturally, what can be inferred or verified for a particular specification depends on the institution in which it is formulated.

A step further towards a uniform, institution-independent setting, provides heterogeneous, *multi-institution* specifications. One takes unstructured specification on specific institutions as basic units, that are structured and combined via adequate logical translations. These maps plays, therefore, a central role, being treated as *first-class citizens* in, e.g., [11]. Such maps lift specifications expressed within different institutions to a common level. Thus any tools, namely proof assistants, available for the target institution, can be borrowed to the source one. Heterogenous specifications are currently supported by Hets [13] and *CafeObj*[5]. The former integrates parsers, static analysers and provers for individual logics, and manages heterogeneous proofs resorting to the so-called graphs of logics, i.e., graphs whose nodes are institutions and, whose edges, are adequate translations between them, known as *institution comorphisms*. Formally, a comorphism between institutions $\big(\mathrm{Sign}^{\mathcal{I}}, \mathrm{Sen}^{\mathcal{I}}, \mathrm{Mod}^{\mathcal{I}}, (\models^{\mathcal{I}}_{\Sigma})_{\Sigma\in|\mathrm{Sign}^{\mathcal{I}}|}\big)$ and $\big(\mathrm{Sign}^{\mathcal{I}'}, \mathrm{Sen}^{\mathcal{I}'}, \mathrm{Mod}^{\mathcal{I}'}, (\models^{\mathcal{I}'}_{\Sigma})_{\Sigma\in|\mathrm{Sign}^{\mathcal{I}'}|}\big)$ consists of a triple $(\Phi, \alpha, \beta)$ where

- $\Phi : \mathrm{Sign}^{\mathcal{I}} \to \mathrm{Sign}^{\mathcal{I}'}$ is a functor
- $\alpha : \mathrm{Sen}^{\mathcal{I}} \Rightarrow \mathrm{Sen}^{\mathcal{I}'} \circ \Phi$ is a natural transformation,
- $\beta : \mathrm{Mod}^{\mathcal{I}'} \circ \Phi^{op} \Rightarrow \mathrm{Mod}^{\mathcal{I}}$ is a natural transformation such that

for any $\Sigma \in |\mathrm{Sign}^{\mathcal{I}}|$, $\rho \in \mathrm{Sen}^{\mathcal{I}}$ and $M' \in \mathrm{Mod}^{\mathcal{I}'}(\Phi(\Sigma))$,

$$M' \models^{\mathcal{I}'}_{\Phi(\Sigma)} \alpha_{\Sigma}(\rho) \text{ iff } \beta_{\Sigma}(M') \models^{\mathcal{I}}_{\Sigma} \rho. \tag{2}$$

A paradigmatic example is the comorphism between $\mathcal{FOL}$, the institution of first-order logic, and $\mathcal{EQ}$ obtained by the encoding first-order relations as boolean functions [4]. We are now in conditions to formally define the specification language intuitively suggested on Section 1, and show that it defines an institution, the *hybrid equational logic institution*, $\mathcal{HEQ}$.

## 3 An institution for hybrid equational specifications

The institution $\mathcal{HEQ}$ is defined as

$$\mathcal{HEQ} = (\mathrm{Sign}^{\mathcal{HEQ}}, \mathrm{Sen}^{\mathcal{HEQ}}, \mathrm{Mod}^{\mathcal{HEQ}}, (\models^{\mathcal{HEQ}}_{\Delta})_{\Delta\in|\mathrm{Sign}^{\mathcal{HEQ}}|}) \tag{3}$$

Its *category of signatures*, $\mathrm{Sign}^{\mathcal{HEQ}}$, takes as objects triples $\langle F, \mathrm{Nom}, \Lambda\rangle$, where $F$ is a signature of $\mathcal{EQ}$ and $\Lambda, \mathrm{Nom}$ are disjoint sets of *modalities* and *nominals*. Morphisms are triples $\varphi = (\varphi_{\mathrm{Sig}}, \varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}})$ with $\varphi_{\mathrm{Sig}}$ a morphism in $\mathcal{EQ}$ between $F$ and $F'$ and $\varphi_{\mathrm{Nom}} : \mathrm{Nom} \to \mathrm{Nom}'$ and $\varphi_{\mathrm{MS}} : \Lambda \to \Lambda'$ are functions. The *sentences functor* $\mathrm{Sen}^{\mathcal{HEQ}}$, maps a signature $\Delta = \langle F, \mathrm{Nom}, \Lambda\rangle$ on the smaller set which contains the $F$-equations and nominals in Nom and it is closed for the boolean connectives $\{\neg, \vee, \wedge, \to\}$ and the satisfaction operator $@_i, i \in \mathrm{Nom}$. Formally,

- $\mathrm{Sen}^{\mathcal{EQ}}(F) \subseteq \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$;
- $\mathrm{Nom} \subseteq \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$;
- for any $\rho, \rho' \in \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$, $\neg\rho, \rho \vee \rho', \rho \wedge \rho', \rho \to \rho' \in \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$
- $@_i\rho \in \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$ for any $\rho \in \mathrm{Sen}^{\mathcal{HEQ}}(\Sigma)$ and $i \in \mathrm{Nom}$;
- $[\lambda]\rho \in \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$, for any $\lambda \in \Lambda, \rho \in \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$.

A signature morphism $\langle F, \mathrm{Nom}, \Lambda\rangle \xrightarrow{\varphi} \langle F', \mathrm{Nom}', \Lambda'\rangle$ induces a sentence translation $\mathrm{Sen}^{\mathcal{EQ}}(\langle F, \mathrm{Nom}, \Lambda\rangle) \xrightarrow{\mathrm{Sen}^{\mathcal{EQ}}(\varphi)} \mathrm{Sen}^{\mathcal{EQ}}(\langle F', \mathrm{Nom}', \Lambda'\rangle)$ recursively defined by

- $\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\rho) = \mathrm{Sen}^{\mathcal{I}}(\varphi_{\mathrm{Sig}})(\rho)$ for any $\rho \in \mathrm{Sen}^{\mathcal{EQ}}(F)$;

- $\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(i) = \varphi_{\mathrm{Nom}}(i)$;
- $\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\neg\rho) = \neg\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\rho)$;
- $\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\rho \odot \rho') = \mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\rho) \odot \mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\rho')$, $\odot \in \{\vee, \wedge, \to\}$;
- $\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(@_i\rho) = @_{\varphi_{\mathrm{Nom}}(i)}\mathrm{Sen}^{\mathcal{HEQ}}(\rho)$;
- $\mathrm{Sen}^{\mathcal{HEQ}}(\varphi)([\lambda]\rho) = [\varphi_{\mathrm{MS}}(\lambda)]\mathrm{Sen}^{\mathcal{HEQ}}(\rho)$;

For each $\langle F', \mathrm{Nom}', \Lambda'\rangle \in |\mathrm{Sign}^{\mathcal{HEQ}}|$, the *category of models* $\mathrm{Mod}^{\mathcal{HEQ}}(F', \mathrm{Nom}', \Lambda')$ has the following structures as objects:

$$\mathcal{A} = \langle S, \mathrm{state} : \mathrm{Nom} \to S, (R_\lambda \subseteq S^2)_{\lambda\in\Lambda}, (A_s)_{s\in S}\rangle, \tag{4}$$

where $S$ is a *set of states*; $\mathrm{state} : \mathrm{Nom} \to S$ is a function that assigns nominals to states; for each $\lambda \in \Lambda$, $R_\lambda \subseteq S^2$ is a binary relation, called a *modality*, and $(A_s)_{s\in S}$ is a $S$-family of $F$-algebras over the same carrier. A morphism between models $\langle S, \mathrm{state} : \mathrm{Nom} \to S, (R_\lambda \subseteq S^2)_{\lambda\in\Lambda}, (A_s)_{s\in S}\rangle$ and $\langle S', \mathrm{state}' : \mathrm{Nom} \to S', (R'_\lambda \subseteq S'^2)_{\lambda\in\Lambda}, (A'_s)_{s\in S'}\rangle$ consists of a pair $\langle \mathrm{h_{St}}, \mathrm{h_{Mod}}\rangle$, where and $\mathrm{h_{Mod}}$ is an $S$-family $\left(\mathrm{h_{Mod}}_s : A_s \to A'_{\mathrm{h_{St}}(s)}\right)_{s\in S}$ of algebras morphisms and $\mathrm{h_{St}} : S \to S'$ is a function such for any $s, s' \in S$ and for any $\lambda \in \Lambda$, $(s, s') \in R_\lambda$ implies that $(\mathrm{h_{St}}(s), \mathrm{h_{St}}(s')) \in R'_\lambda$, and for any $n \in \mathrm{Nom}$, $\mathrm{state}'(n) = \mathrm{h_{St}}(\mathrm{state}(n))$.

The *reduct* of a $\Delta'$-model $\mathcal{A}' = \langle S', \mathrm{state}' : \mathrm{Nom}' \to S', (R'_\lambda \subseteq S^2)_{\lambda\in\Lambda'}, (A'_s)_{s\in S}\rangle$, along $\varphi : \Delta \to \Delta'$, denoted by $\mathrm{Mod}^{\mathcal{HEQ}}(\varphi)(\mathcal{A}')$, consists of the $\Delta$-model $\langle S, \mathrm{state} : \mathrm{Nom} \to S, (R_\lambda \subseteq S^2)_{\lambda\in\Lambda}, (A_s)_{s\in S}\rangle$ where $S = S'$, $\mathrm{state}(n) = \mathrm{state}'(\varphi_{\mathrm{Nom}}(n))$ for any $n \in \mathrm{Nom}$, $R_\lambda = R'_{\varphi_{\mathrm{MS}}(\lambda)}$ for any $\lambda \in \Lambda$ and $A_s = \mathrm{Mod}^{\mathcal{I}}(\varphi_{\mathrm{Sig}})(A'_s)$ for any $s \in S$.

Finally, we have a $|\mathrm{Sign}^{\mathcal{EQ}}|$-family of relations $\models_\Delta \subseteq \mathrm{Mod}^{\mathcal{EQ}}(\Delta) \times \mathrm{Sen}^{\mathcal{EQ}}(\Delta)$, recursively defined, for each $\mathcal{A} = \langle S, \mathrm{state} : \mathrm{Nom} \to S, (R_\lambda \subseteq S^2)_{\lambda\in\Lambda}, (A_s)_{s\in S}\rangle, \in \mathrm{Mod}^{\mathcal{EQ}}(\Delta)$, and for any $s \in S$, $\rho, \rho' \in \mathrm{Sen}^{\mathcal{EQ}}(\Delta)$, $e \in \mathrm{Sen}^{\mathcal{EQ}}(F)$ and $i, j \in \mathrm{Nom}$ as follows:

- $\mathcal{A} \models^s e$ iff, $A_s \models^{\mathcal{EQ}} e$; $\qquad \mathcal{A} \models^s i$ iff, $\mathrm{Nom}(s) = i$;
- $\mathcal{A} \models^s @_j\rho$ iff $\mathcal{A} \models^{\mathrm{state}(j)} \rho$;
- $\mathcal{A} \models^s [\lambda]\rho$ iff, $\mathcal{A} \models^w \rho$ for any $(s, w) \in R_\lambda$;

with the obvious definition for $\vee$, $\wedge$ and $\to$. The following theorem, which is proved by induction on the structure of sentences (the interested reader is referred to [10] for proofs), completes the presentation of $\mathcal{HEQ}$.

**Theorem 1.** *Let $\Delta = (F, \mathrm{Nom}, \Lambda)$ and $\Delta' = (F', \mathrm{Nom}', \Lambda')$ two hybrid signatures and $\varphi : \Delta \to \Delta'$ an hybrid signature morphism. Then, for any $\rho \in \mathrm{Sen}^{\mathcal{HEQ}}(\Delta)$ and for any $\mathcal{A}' = \langle S, \mathrm{state} : \mathrm{Nom} \to S, (R_\lambda \subseteq S^2)_{\lambda\in\Lambda}, (A_s)_{s\in S}\rangle \in |\mathrm{Mod}^{\mathcal{HEQ}}(\Delta')|$, $\mathrm{Mod}^{\mathcal{HEQ}}(\varphi)(\mathcal{A}') \models^s \rho$ iff $\mathcal{A}' \models^s \mathrm{Sen}^{\mathcal{HEQ}}(\varphi)(\rho)$, for all $s \in S$.*

As announced, it is possible to establish translations between hybrid logic and the classic first order logic. A standard procedure [3] translates hybrid formulas to the first-order logic by transforming functions and relations local to each state to global functions and relations parametrized by states. On the present section, we enlighten this phenomena, defining a comorphism between $\mathcal{HEQ}$ and $\mathcal{FOL}$. This result is fundamental for our approach as it brings to scene all reasoning power of first order logic. Moreover, it provides the key for the integration of $\mathcal{HEQ}$ on the HETS framework. We sketch in the sequel its basic structure. The relevant comorphism is defined as $(\Phi, \alpha, \beta) : \mathcal{HEQ} \to \mathcal{FOL}$ where, functor $\Phi : \mathrm{Sign}^{\mathcal{HEQ}} \longrightarrow \mathrm{Sign}^{\mathcal{FOL}}$, mapping $(F, \Lambda, \mathrm{Nom})$ to $(\{W, U\}, \bar{F}, \bar{R})$, is defined by $\bar{F} = \{x_i :\to W | i \in \mathrm{Nom}\} \cup \{\bar{f} : W \times U^n \to U | f \in F_n\}$ and $\bar{R} = R_\Lambda$. The natural transformation $\beta : \Phi^{op} \circ \mathrm{Mod}^{\mathcal{FOL}} \Rightarrow \mathrm{Mod}^{\mathcal{HEQ}}$ maps each $(M, M_{\bar{F}}, M_{\bar{R}}) \in \mathrm{Mod}\big((\{W, U\}, \bar{F}, \bar{R})\big)$ on

$$(S, \mathrm{state}, R_\Lambda, (M_s)_{s \in S}) \xleftarrow[(\beta_{F,\Lambda,\mathrm{Nom}})]{} (M, M_{\bar{F}}, M_{\bar{R}}) \ ,$$

where $S = M_W$, $\mathrm{state}(i) = x_i^M$, $i \in \mathrm{Nom}$, $R_\Lambda = R_\Lambda^M$ and $M_s = \langle M_U, F^{M_s} \rangle$, where for any $f \in F_n$ and each $u_i \in U$, $i \leq n$, $f^{M_s}(u_1, \ldots, u_n) = \bar{f}^M(s, u_1, \ldots, u_n)$.

The natural transformation $\alpha : \mathrm{Sen}^{\mathcal{HEQ}} \Rightarrow \mathrm{Sen}^{\mathcal{FOL}} \circ \Phi$ is defined for each $(F, \mathrm{Nom}, \Lambda)$-sentence by $\alpha(\rho) = (\forall x)\alpha_x(\rho)$,

$$\begin{aligned}
\alpha_x(\forall X\, t \approx t') &= \forall X\, \mathcal{T}_x(t) \approx \mathcal{T}_x(t') \\
\alpha_x(i) &= x_i \approx x, && i \in \mathrm{Nom} \\
\alpha_x(@_i \rho) &= \alpha_x(\rho)[x_i/x], && i \in \mathrm{Nom} \\
\alpha_x([\lambda]\rho) &= (\forall y)[(x, y) \in R_\lambda \to \alpha_y(\rho)], && \lambda \in \Lambda \\
\alpha_x(\neg\rho) &= \neg\alpha_x(\rho) \\
\alpha_x(\rho \odot \rho') &= \alpha_x(\rho) \odot \alpha_x(\rho'),\ \odot \in \{\vee, \wedge, \to\}
\end{aligned}$$

where for each $f(t_1, \ldots, t_n) \in T_F$, $\mathcal{T}_x(f(t_1, \ldots, t_n)) = \bar{f}(x, \mathcal{T}_x(t_1), \ldots, \mathcal{T}_x(t_n))$. We may, finally, state the basic result:

**Theorem 2.** *Let $\Delta \in |\mathrm{Sign}^{\mathcal{HEQ}}|$, $\rho \in \mathrm{Sen}^{\mathcal{HEQ}}$ and $M' \in \mathrm{Mod}^{\mathcal{FOL}}(\Phi(\Delta))$. Then, for $\alpha$ and $\beta$ defined as above we have that,*

$$\beta_\Delta(M') \models_\Delta^{\mathcal{HEQ}} \rho \text{ iff } M' \models_{\Phi(\Delta)}^{\mathcal{FOL}} \alpha_\Delta(\rho). \tag{5}$$

Back to our running example, we encode an $\mathcal{HEQ}$-specification in $\mathcal{FOL}$ by mapping $\Delta = \langle \Sigma, \{\Box\}, \{+, \times\} \rangle$ to signature $\Phi(\Delta)$ with the set of sorts $\{nat, states\}$ and the set of operations $\{\bar{0} : states \to nat;\ \overline{suc} : states \times nat \to nat; \overline{p} : states \times nat \to nat;\ c_+ :\to states;\ c_\times :\to states\}$. In order to understand how the translation $\alpha$ works, we present three examples:

$$\begin{aligned}
\alpha\big(p(suc(n)) \approx n\big) \ &= \forall s : states\ (\alpha_s[p(suc(n) \approx n)) \\
&= \forall s : states\ \overline{p}(s, \overline{suc}(s, n)) \approx n \\
\alpha([shift]+ \leftrightarrow \times) \ &= \forall s : states\ [\alpha_s([shift]+ \leftrightarrow \times)] \\
&= \forall s : states\ [\alpha_s([shift]+) \leftrightarrow \alpha_s(\times)] \\
&= \forall s : states\ [\forall v : states\ ((s, v) \in R_{shift} \to c_+ \approx v) \leftrightarrow c_\times \approx s] \\
\alpha(@_\times \star (n, 0) \approx 0) &= \bar{\star}(c_\times, n, \bar{0}(c_\times)) \approx \bar{0}(c_\times)
\end{aligned}$$

## 4 Concluding

The paper suggested an approach to define and reason about complex specifications resorting to a hybrid logic with equations which was formalized as an institution. Moreover it presented a comorphism to $\mathcal{FOL}$ which caters for its encoding in HETS, as well as in theorem provers based in first order languages.

The impact of such a smooth, uniform setting, with suitable tool support, in teaching software specification at undergraduate level, seems promising, although it is still to early to assess. It can be said, however, that it completely meets our initial aims: integrating in a single course on Software Specification the ability to state and reason, in a single formal framework, about functional and behavioural, global and local properties of complex software, with suitable tool support.

## References

1. P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of IGPL*, 8(3):339–365, 2000.
2. P. Blackburn and M. Marx. Tableaux for quantified hybrid logic. In *Methods for modalities 2, workshop proceedings, November 29-30, 2001. ILLC*, pages 38–52. Springer Verlag, 2002.
3. T. Braüner. Natural deduction for first-order hybrid logic. *Journal of Logic, Language and Information*, 14:173, 2005.
4. R. Diaconescu. *Institution-independent Model Theory.* Birkhäuser Basel, 2008.
5. R. Diaconescu and K. Futatsugi. Logical foundations of cafeobj. *Theoretical Computer Science*, 285:289–318, 2002.
6. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* Springer Verlag, 1985.
7. J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39:95–146, January 1992.
8. J. F. Groote and *et al.* The mcrl2 toolset. In *Proc. Int. Workshop on Advanced Software Development Tools and Techniques*, 2008.
9. L. Logrippo, T. Melanchuk, and R. J. Du Wors. The algebraic specification language lotos: an industrial experience. In *Proceedings Int. Conf. on Formal methods in software development*, pages 59–66. ACM, 1990.
10. M. A. Martins, A. Madeira, R. Diaconescu, and L. S. Barbosa. Hybridization of institutions. Technical report, CCTC, Minho University (submitted to a conference), 2011.
11. T. Mossakowski. Foundations of heterogeneous specification. In *WADT 2002, 16th Inter. Workshop on Recent Trends in Algebraic Development Techniques, Revised Selected Papers, LNCS*, pages 359–375. Springer, 2003.
12. T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. CASL: The common algebraic specification language: Semantics and proof theory. *Computing and Informatics*, 22:285–321, 2003.
13. T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set, hets. In *13th Int. Conf. Tools and algorithms for the construction and analysis of systems*, TACAS'07, pages 519–522, Berlin, Heidelberg, 2007. Springer-Verlag.
14. J. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
15. D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, (9):229–269, 1997.
16. A. Tarlecki. Abstract specification theory: An overwiew. In *Models, Algebras, and Logics of Engineering Software,M. Broy, M. Pizka eds.*, NATO Science Series, Computer and Systems Sciences, VOL 191, pages 43–79. IOS Press, 2003.