Reuse and Integration of Specification Logics: The Hybridisation Perspective

Luis S. Barbosa, Manuel A. Martins, Alexandre Madeira and Renato Neves

Abstract Hybridisation is a systematic process along which the characteristic features of hybrid logic, both at the syntactic and the semantic levels, are developed on top of an arbitrary logic framed as an institution. It also captures the construction of first-order encodings of such hybridised institutions into theories in first-order logic. The method was originally developed to build suitable logics for the specification of reconfigurable software systems on top of whatever logic is used to describe local requirements of each system's configuration. Hybridisation has, however, a broader scope, providing a fresh example of yet another development in combining and reusing logics driven by a problem from Computer Science. This paper offers an overview of this method, proposes some new extensions, namely the introduction of full quantification leading to the specification of dynamic modalities, and exemplifies its potential through a didactical application. It is discussed how hybridisation can be successfully used in a formal specification course in which students progress from equational to hybrid specifications in a uniform setting, integrating paradigms, combining data and behaviour, and dealing appropriately with systems evolution and reconfiguration.

Keywords Software specification · Hybrid logic · Hybridization

L.S. Barbosa (✉) · A. Madeira · R. Neves
HASLab - INESC TEC & University of Minho, Braga, Portugal
e-mail: luis.s.barbosa@inesctec.pt

A. Madeira
e-mail: amadeira@inesctec.pt

R. Neves
e-mail: rjneves@inescporto.pt

M.A. Martins
CIDMA - Department of Mathematics, University of Aveiro, Aveiro, Portugal
e-mail: martins@ua.pt

T. Bouabana-Tebibel and S.H. Rubin (eds.), *Theoretical Information Reuse and Integration*, Advances in Intelligent Systems and Computing 446,
DOI 10.1007/978-3-319-31311-5_1

1 Introduction

Hybrid logic [5, 10, 14, 31] adds to a modal language the ability to name, or to explicitly refer to, specific states of the underlying Kripke structure. This is done through the introduction of propositional symbols of a new sort, called *nominals*, each of which is true at exactly one possible state. Sentences are then enriched in two directions. On the one hand, nominals are used as simple sentences, each of them holding exclusively in the state it names. On the other hand, explicit reference to states is provided by sentences such as $@_i\, \rho$, stating the validity of ρ at the state named i.

Hybrid logic was originally introduced by A. Prior in his book [46], and later revisited, in the school of Sofia, by Passy and Tinchev [47], awakening a broad interest within the modal logic community along the 90s. Our own interest in this generalisation of modal logic was triggered by a concrete problem in (rigorous) software engineering—the specification of *reconfigurable* software systems. The qualifier *reconfigurable* is used for systems whose execution modes, and not only the values stored in their internal memory, may change in response to the continuous interaction with the environment. Such systems behave differently in different modes of operation, or *configurations*, and commute between them along their lifetime.

Present such is more the norm than the exception. A typical, everyday example is offered by cloud based applications that elastically react to client demands. Another example is a modern car in which hundreds of electronic control units must operate in different modes depending on the current situation—such as driving on a highway or finding a parking spot. Switching between these modes is an intuitive example of a dynamic reconfiguration. As a matter of fact, reconfigurability, together with related issues like self-adaptation or context-awarness, became a main research topic [48], in the triple perspective of foundations, methods and technologies.

Clearly, the dynamics of reconfiguration of a software system can be described by some sort of transition system, whose states represent configurations and transitions are triggered by whatever conditions enforce a switch of configurations. However, one needs also to capture the specific, *local* requirements which characterise each configuration and distinguish one from the others. Formally, such different behaviours can be modelled by imposing additional structure upon the states of the transition system which expresses the overall dynamics.

This path was explored in our previous work [35] on a specification methodology for reconfigurable systems. The basic insight is that, starting from a classical state-machine specification, each state, regarded as a possible system's configuration, is equipped with a rich mathematical structure to describe its functionality. Technically, specifications become structured state-machines whose states denote algebras or first order structures, rather than sets. Such a specification should be able to make assertions both about the transition dynamics and, locally, about each particular configuration. This explains why hybrid logic was chosen as the *lingua franca* for the envisaged methodology. One may therefore specify (local) properties of specific configurations in the system or even assert the equality between two particular configurations, something that is beyond what can be said in a modal language. Modalities, however, capture state transitions, providing a way to specify the *global* dynamics of reconfigurability.

For the working software architect, the relevant question goes a step forward: the envisaged methodology should be *independent* of whatever logic is found appropriate to express *local* requirements for each configuration. Actually, specific problems do require specific logics to describe their configurations (e.g., equational, first-order, fuzzy, etc.). Therefore, instead of choosing a particular version of hybrid logic, the method proposed in [35] starts by choosing a specific logic to express requirements at the configuration level. This is later taken as the *base* logic on top of which the characteristic features of hybrid logic are developed.

Such a process along which the characteristic features of hybrid logic, both syntactical and semantical, are developed on top of a given logic, in a parametric way, is called *hybridisation*, and was proposed in Madeira Ph.D. thesis [34], whose core results were published in Refs. [22, 23, 39]. Going generic entailed the need for a proper abstract foundation. Therefore, the whole approach is framed in the context of the theory of institutions of Goguen and Burstall [20, 25], each logic (base and hybridised) being treated abstractly as an institution.

As discussed in the sequel, hybridisation techniques not only offer a main conceptual tool for dealing with reconfigurable systems, but are also valuable in designing innovative teaching approaches in Software Engineering.

Aims. In such a context, this paper has a triple objective. First of all, it offers an overview of this method, emphasising conceptual exposition, rather than the purely technical style the interested reader may find in the references above. Secondly it exemplifies its potential through a *didactical* application, as a follow up to the original workshop paper [38]. The focus is on how the method can provide ways of reusing and integrating different specification logics in an undergraduate course on formal software specification. This leads to the design of a new course along which students progress from equational to hybrid specifications in a uniform setting, integrating paradigms, combining data and behaviour, and dealing appropriately with systems evolution and reconfiguration. Finally, it extends the method in two directions: (i) computational support for the translation of system's requirements in the format of boilerplates to $\mathcal{H}$CASL; (ii) introduction of full quantification in the method providing a way to specify dynamic modalities and, in general, the change 'on-the-fly' of the transition relation.

Paper structure. The hybridisation method is described and illustrated in the next section. Section 3 discusses the integration of the method in the HETS platform, therefore providing effective tool support to (some families of) hybridised specifications. Its didactical use in an introductory course to formal software specification is the subject of Sects. 4 and 5. Section 6 extends the method to deal with full quantification, which forms the main, original contribution of the paper. Finally, Sect. 7 reviews related work in the area of combination of logics and concludes pointing out current research directions.

2 The Hybridisation Method

2.1 Institutions

An *institution* is an abstract formalisation of a logical system, encompassing syntax, semantics and satisfaction. The concept was put forward by Goguen and Burstall, in the end of the Seventies, in order to "*formalise the formal notion of logical systems*", in response to the "*population explosion among the logical systems used in Computing Science*" [25].

The universal character of institutions proved effective and resilient as witnessed by the wide number of logics it was able to formalise. Examples range from the usual logics in classical mathematical logic (propositional, equational, first order, etc.), to the ones underlying specification and programming languages or used for describing particular systems from different domains. Well-known examples include *probabilistic logics* [9], *quantum logics* [18], *hidden and observational logics* [6, 8], *coalgebraic logics* [15], as well as logics for reasoning about *process algebras* [42], *functional* [50, 52] and *imperative programing languages* [52].

The theory of institutions (see [20] for an extensive account) was motivated by the need to abstract from the particular details of each individual logic and to characterise fundamental concepts, such as satisfaction and combination of logics, in very general terms. This lead to the development of a solid *institution-independent specification theory*, on which, structuring and parameterisation mechanisms, required to scale up software specification methods, are defined 'once and for all', irrespective of the concrete logic used in each application domain.

Formally, an institution

$$I = \left(\mathrm{Sign}^{\mathcal{I}}, \mathrm{Sen}^{\mathcal{I}}, \mathrm{Mod}^{\mathcal{I}}, (\models^{I}_{\Sigma})_{\Sigma\in|\mathrm{Sign}^{\mathcal{I}}|}\right)$$

consists of a category $\mathrm{Sign}^{\mathcal{I}}$ of *signatures* and *signature* morphisms; a functor $\mathrm{Sen}^{\mathcal{I}}$, $\mathrm{Sen}^{\mathcal{I}} : \mathrm{Sign}^{\mathcal{I}} \to \mathbb{S}et$, giving for each signature a set of *sentences* over that signature; another functor $\mathrm{Mod}^{\mathcal{I}} : (\mathrm{Sign}^{\mathcal{I}})^{op} \to \mathbb{C}AT$, providing for each signature Σ a category of *Σ-models* and *Σ-(model) homomorphisms*, and, finally, a satisfaction relation.

Note that each morphism of signatures $\varphi : \Sigma \to \Sigma' \in \mathrm{Sign}^{\mathcal{I}}$ induces a semantic map, i.e., a functor $\mathrm{Mod}^{\mathcal{I}}(\varphi) : \mathrm{Mod}^{\mathcal{I}}(\Sigma') \to \mathrm{Mod}^{\mathcal{I}}(\Sigma)$ called the *reduct functor*, whose effect is to cast a model of Σ' as a model of Σ. Therefore, the satisfaction relation $\models^{I}_{\Sigma} \subseteq |\mathrm{Mod}^{\mathcal{I}}(\Sigma)| \times \mathrm{Sen}^{\mathcal{I}}(\Sigma)$, for each $\Sigma \in |\mathrm{Sign}^{\mathcal{I}}|$, verifies the following condition, which, for each signature morphism φ, entailing a syntactic transformation, captures the basic principle of *truth invariance under change of notation* [25]:

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

2.2 The Method

This section reviews the *hybridisation* method proposed in [23, 39]. The method enriches a base (arbitrary) institution I with hybrid logic features and the corresponding Kripke semantics. The result is still an institution, $\mathcal{H}I$, called the *hybridisation of* I. In the sequel we concentrate in a simplified version, i.e., quantifier-free and non-constrained, of the general method, to convey the basic intuitions.

At the syntactic level the base signatures are enriched with nominals and polyadic modalities. Therefore, the category of *I-hybrid signatures*, denoted by $\mathrm{Sign}^{\mathcal{H}I}$, is defined as the direct (cartesian) product of categories of the original category of signatures $\mathrm{Sign}^{\mathcal{I}}$ and that of signatures of *REL*, the sub-institution of (the institution of) first order logic, without non-constant operation symbols, Sign^{REL}. Signatures of the hybridised institution combine those of I with a set of constants Nom for *nominals* and a set of relational symbols Λ to represent *modalities*. $\mathcal{H}I$ signatures are, thus, triples $(\Sigma, \mathrm{Nom}, \Lambda)$, with signature morphisms $\varphi = (\varphi_{\mathrm{Sig}}, \varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}}) : (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda')$, defined component-wise: the first component is inherited from I and the others simply map nominals and modalities while preserving the arities of the latter.

The second step in the method is to enrich the base sentences accordingly. The sentences of the base institution I and the nominals in Nom are taken as atoms and composed with the Boolean connectives, the modalities in Λ, and satisfaction operators indexed by nominals. For example, for a n-ary modality λ, a nominal i and $\mathcal{H}I$-sentences $\rho, \rho_1, \rho_2 \ldots, \rho_n$, the following are also sentences in $\mathcal{H}I$: $[\lambda](\rho_1, \ldots, \rho_n)$, $\langle\lambda\rangle(\rho_1, \ldots, \rho_n)$ and $@_i\rho$.

Given a $\mathcal{H}I$-signature morphism φ, the translation of sentences $\mathrm{Sen}^{\mathcal{HI}}(\varphi)$ is defined structurally: e.g.,

$$
\begin{aligned}
\mathrm{Sen}^{\mathcal{HI}}(\varphi)(i) &= \varphi_{\mathrm{Nom}}(i)\\
\mathrm{Sen}^{\mathcal{HI}}(\varphi)(@_i\rho) &= @_{\varphi_{\mathrm{Nom}}(i)}\mathrm{Sen}^{\mathcal{HI}}(\rho) \text{ and}\\
\mathrm{Sen}^{\mathcal{HI}}(\varphi)([\lambda](\rho_1, \ldots, \rho_n)) &= [\varphi_{\mathrm{MS}}(\lambda)](\mathrm{Sen}^{\mathcal{HI}}(\rho_1), \ldots, \mathrm{Sen}^{\mathcal{HI}}(\rho_n))
\end{aligned}
$$

Models of $\mathcal{H}I$ can be regarded as (Λ-)Kripke structures whose worlds are I-models. Formally, they are pairs (M, W) where W is a (Nom, Λ)-model in *REL* and M is a function which assigns to each state $w \in W$ a model $M(w) \in |\mathrm{Mod}^{\mathcal{I}}(\Sigma)|$. We denote $M(w)$ simply by M_w.

In each world (M, W), W_n provides an interpretation for nominal n, whereas relation W_λ interpretes modality λ. The reduct definition is lifted from the base institution: the reduct of a Δ'-model (M', W') along a signature morphism $\varphi : \Delta \to \Delta'$ is the Δ-model (M, W) such that W is the $(\varphi_{\mathrm{Nom}}, \varphi_{\mathrm{MS}})$-reduct of W' (i.e., $|W| = |W'|$, $W_n = W'_{\varphi_{\mathrm{Nom}}(n)}$, for each nominal n, and $W_\lambda = W'_{\varphi_{\mathrm{MS}}(\lambda)}$ for each modality in Λ).

Finally, the satisfaction relation for the hybridised institution resorts to the one in the base institution for sentences in I, i.e.,

- $(M, W) \models^w \rho$ iff $M_w \models^I \rho$ when $\rho \in \mathrm{Sen}^I(\Sigma)$,

captures the semantics of nominals

- $(M, W) \models^w i$ iff $W_i = w$, when $i \in \mathrm{Nom}$
- $(M, W) \models^w @_j\rho$ iff $(M, W) \models^{W_j} \rho$

and modalities, as in

- $(M, W) \models^w [\lambda](\xi_1, \ldots, \xi_n)$ iff, for any $(w, w_1, \ldots, w_n) \in W_\lambda$, $(M, W) \models^{w_i} \xi_i$ for some $1 \leq i \leq n$

and is defined as usual for the Boolean connectives.

The main result is that $\mathcal{H}I$ effectively constitutes an institution [39]. The next step is the systematic characterisation of encodings of the hybridised institution $\mathcal{H}I$ into the institution of many sorted first-order logic (*FOL*) building on existent encodings of the base institution I into *FOL*. This is discussed below in Sect. 3.

2.3 Examples

Propositional logic. Propositional logic gives rise to a well-known institution *PL* whose signatures are sets of propositional symbols and signature morphisms are functions between them. Models assign truth values to propositions and interpret propositional sentences, built with the Boolean connectives, in the usual way.

The hybridisation of the institution of propositional logic *PL* introduces nominals and modalities resulting in an institution whose sentences are generated by

$$\rho ::= \ \rho_0 \mid i \mid @_i\rho \mid \rho \odot \rho \mid \neg\rho \mid \langle\lambda\rangle(\rho, \ldots, \rho) \mid [\lambda](\rho, \ldots, \rho)$$

where ρ_0 is a sentence inherited from *PL*, $\odot = \{\vee, \wedge, \Rightarrow\}$, and i and λ stand, respectively, for a nominal and a modality symbol. Note there is a double level of connectives in the sentences: one coming from base *PL*-sentences and another introduced by the hybridisation process. However, they "semantically collapse" and, hence, no distinction between them needs to be done (see [23] for details). A $\mathcal{H}PL$ model has a transition structure to interpret each added modality. Each world comes equipped with a *PL*-model, i.e., a particular subset of propositions holding locally.

As one would expect, restricting signatures to those with just a single unary modality results in the usual institution for classical hybrid propositional logic [14].
Propositional fuzzy logic. Many-valued logics [26] generalise classic logics by replacing, as their *truth domain*, the 2-element Boolean algebra, by larger sets structured as complete residuated lattices. A residuated lattice includes an associative, monotonic binary operation $\otimes$, with the biggest element as the identity and such that there exists an element $x \Rightarrow z$ verifying $y \leq (x \Rightarrow z)$ iff $x \otimes y \leq z$. They were originally formalised as institutions in [21].

Given a complete residuated lattice L, an institution MVL_L is defined based on *PL*-signatures, but whose sentences are pairs (ρ, p) formed by an element p of L

and a PL-sentence ρ defined over the usual Boolean connectives and $\otimes$. Models are functions evaluating propositions on the lattice, rather than on the Boolean domain. Accordingly, a sentence (ρ, p) is satisfied in a model M if p is less or equal the evaluation of sentence ρ in M.

This institution captures many many-valued logics discussed in the literature. For instance, taking L as the Łukasiewicz arithmetic lattice over the closed interval [0, 1], where $x \otimes y = 1 - max\{0, x + y - 1)\}$ (and $x \Rightarrow y = min\{1, 1 - x + y\}$), yields the standard *propositional fuzzy logic*.

The institution obtained through the hybridisation of MVL_L, for a fixed L, is similar to $\mathcal{H}PL$ but for two aspects: sentences are defined as in $\mathcal{H}PL$ but taking sentences (ρ_0, p) as atomic; and a function assigning to each proposition a value in L, is associated to each world.

Note that expressivity increases even in the restricted case of a (one-world) standard semantics. Differently from what happens in the base logic, where each sentence is tagged by a L-value, in the hybridised institution expressions may involve different L-values, as in, for example, $(\rho, p) \wedge (\rho', p')$. The reason for this is the introduction of Boolean connectives by the hybridisation process.

Equational logic. Signatures in the institution EQ of equational logic are pairs (S, F) where S is a set of sort symbols and $F = \{F_{\underline{\mathrm{ar}} \to s} \mid \underline{\mathrm{ar}} \in S^*, s \in S\}$ is a family of sets of operation symbols indexed by arities $\underline{\mathrm{ar}}$ (for the arguments) and sorts s (for the results). Signature morphisms map both components in a compatible way. A model for a given signature is an algebra interpreting each sort symbol as a carrier set and each operation symbol as a function; model morphisms are, of course, homomorphisms of algebras. Sentences are universal quantified equations $(\forall X)t = t'$ and the satisfaction relation is the usual Tarskian satisfaction defined recursively on the structure of the sentences.

The hybridisation of EQ gives rise to an institution $\mathcal{H}EQ$ whose signatures are triples $((S, F), \mathrm{Nom}, \Lambda)$ and the sentences are defined as in the previous examples, but taking (S, F)-equations $(\forall X)t = t'$ as atomic base sentences instead. Models are Kripke structures with a (local) (S, F)-algebra associated to each world.

3 Hybridisation at Work

Hybridised logics provide an interesting framework to specify and reason about reconfigurable software systems. As explained above, models for reconfigurable software can be regarded as structured transition systems, whose states represent individual configurations with whatever structure they have to bear in concrete applications. Transitions, on the other hand, correspond to the admissible reconfigurations. For example, if local requirements are captured equationally, as they often are in formal specification methods, distinct configurations can be modelled by distinct algebras. Clearly, specifications are given equationally, based on EQ-signatures. Nominals identify the “relevant” configurations, and reconfigurations amount to state transitions. Therefore, one resorts to equations tagged with the satisfaction operators to specify configurations; plain equations to specify the system global properties and modal features to specify its reconfiguration dynamics.

The key ingredient to make these ideas appealing for the working software engineer is the existence of computer-based support for reasoning about specifications in logics obtained by hybridisation. Technically, this amounts to the existence of tools to transport specifications from a logical system to another, with more effective proof support. This is done through the systematic characterisation of encodings of hybridised institutions into *FOL*, the institution of *many sorted first-order logic*. In this section we discuss such encodings and the tool support they provide on top the HETS platform [40].

3.1 First-Order Encodings

As mentioned above, for each institution "encodable" in *FOL* theories, there is a method to construct an encoding from its hybridisation to *FOL*. Therefore, a wide variety of computer assisted provers for first order logic can be "borrowed" to reason about specifications in the new, hybridised logics.

Technically such encodings extend the classical *standard translation* of modal logic into the (one-sorted) first order logic [53], more precisely, of its hybrid version [10], to the encodings of hybridised institutions into *FOL*.

The standard translation from hybrid propositional logic $\mathcal{H}PL$ into the (one-sorted) first-order logic introduces a new sort to encode the state space, interprets nominals as constants, modalities as binary relations, and propositions as unary predicates encoding the validity of each proposition in each state. Brauner [14] extends this encoding in devising the translation from hybrid first order logic $\mathcal{H}FOL$ to *FOL*. Basically, he introduces a new universe as an extra sort in the signature, and "flattens" the universes, operations and predicates of the (local) *FOL*-models to an unique (global) *FOL*-model. Local functions and predicates become parametric over states, and the state universes distinguished with a sort-family of definability predicates. Intuitively, whenever m belongs to the universe of w, $\pi(w, m)$ and $\sigma(w, m) = b$ means that $\pi(m)$ and $\sigma(m) = b$ hold in state w. The restriction of this global model M to the local universes, operations and predicates of a fixed word w, gives rise to a "slice of M", say $M|_w$, i.e., a local *FOL*-model which represents (and coincides with) M_w.

A similar method, based on a state-parametric construction, is used in our context to lift *I2FOL* to $\mathcal{H}I2FOL$. Thus, all the signatures and sentences targeted by *I2FOL* become parametric on states. A slice $M|_w$ corresponds now to the "*FOL*-interpretation" of the local I-model M_w, which can be recovered using *I2FOL*. Actually, this process can be understood as a *combination of logic encodings* between the standard translation of hybrid logic into *FOL* and other encodings into *FOL*.

Such encodings are required to be conservative "theoroidal comorphisms" [27, 41], i.e., they are supposed to map signatures to theories. Conservativity, i.e., requirement that models are translated through surjections, is a sufficient condition to use such maps as actual encodings. In particular, this is necessary in order to borrow from *FOL* proof resources in a sound and complete way. This entails the need for an abstract characterisation of conservativity which appeared in [23]. This reference also extends the method originally proposed in [39] for generating first-order encodings in hybridised institutions to theories, constrained models and quantified sentences.

Constrained models provide a very general way to introduce sharing constraints into the picture. Those are traditionally modelled via the so-called "rigid" syntactic entities, which means that some sorts, functions, or predicates are designated as "rigid" and consequently their interpretations are invariant across possible worlds. Constrained models are indispensable for having encodings into first-order logic, more precisely to reflect the consequence relation (see [22] for a detailed account).

3.2 Implementation in the HETS *Platform*

Encodings, as discussed above, provide the right path to transport specifications from a logical system to another offering more effective, computer-based proof support. HETS has been described as a "motherboard" of logics where different "expansion cards" can be plugged in. These are individual logics (with their particular analysers and proof tools) as well as logic translations. To make them *compatible*, logics are formalised as institutions and translations as comorphisms. Therefore, the integration of hybrid specifications in the HETS platform is legitimate, since all formal requirements (e.g., that institutions exist, that comorphisms can be defined, etc.) are already guaranteed by the hybridisation process itself.

This implementation was done along two different directions, both documented in [43]. Firstly the general hybridisation method was incorporated in HETS , making available parsing and static analysis for the hybridisation of any base institution already supported by this platform. Secondly, the encoding along the comorphism $\mathcal{H}$CASL $\rightarrow$ CASL was implemented, offering effective tool support for proofs on a number of $\mathcal{H}$CASL-sub-institutions, namely $\mathcal{H}PL$ and $\mathcal{H}FOL$. Institution $\mathcal{H}$CASL consists of the hybridisation of the institution for CASL [36], the platform *lingua franca*, with the models restricted to those with common realisation of sorts in all the states and of the quantified variables. This provides for free the proof support environment of a particularly well established logic. The implementation of the hybridisation method in HETS proved an effective and flexible way to prove properties of hybrid specifications and thus to support the design method in [35, 37].

3.3 An Example

Figure 1 depicts the setting for a toy, yet illustrative example of a hybrid specification and its encoding. The system is a "swinging" calculator with only one operation which can be interpreted in two possible modes. In one of them it adds two natural numbers,

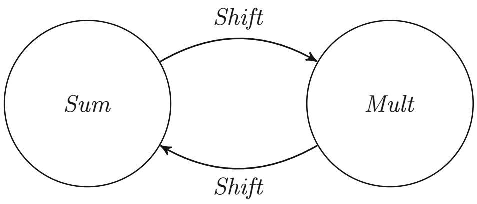


Fig. 1 The swinging calculator

in the other multiplies them. One switches between these two modes through the *Shift* command.

The underlying Kripke frame is specified as follows:

modalities *Shift*
nominals *Sum*, *Mult*
$@Sum \neg Mult$
$Sum \vee Mult$
$@Sum\ (\langle Shift\rangle\ Mult \wedge [Shift]\ Mult)$
$@Mult\ (\langle Shift\rangle\ Sum \wedge [Shift]\ Sum)$

The first axiom rules out models where *Sum* and *Mult* would collapse into each other. The second one restricts to models which admit at most two possible modes. Thus all valid Kripke frames for this example will have precisely the two desired modes of operation. Transitions between them (i.e., the reconfiguration dynamics) are characterised by the last two sentences. The "reconfigurable" operation is declared in the calculator's "global" signature:

op $\_\_\#\_\_ : Nat \times Nat \to Nat$

Global properties of the calculator, for example # commutativity and associativity, can be specified as follows,

$\forall\ n, m, p : Nat$
$\bullet\ n \# m = m \# n$
$\bullet\ (n \# m) \# p = n \# (m \# p)$

The behaviour of #, however, needs to be defined locally, i.e. relative to each possible mode of operation, *Sum* and *Mult*. Thus,

$\forall\ n, m : Nat$
$\bullet\ @Sum\ n \# 0 = n$
$\bullet\ @Sum\ n \# suc(m) = suc(n \# m)$
$\bullet\ @Mult\ n \# 0 = 0$
$\bullet\ \exists\ p, q : Nat$
  $\bullet\ @Mult\ n \# suc(m) = p \wedge @Sum\ n \# q = p \wedge @Mult\ n \# m = q$

which concludes the specification. Note that the last sentence represents the equation $n * (m + 1) = n + (n * m)$, where $+$ and $*$ are, respectively, the usual addition and multiplication of natural numbers. The translation of these axioms to CASL proceeds as described above, with the introduction of a new sort to encode the state space upon which nominals are interpreted as constants (Wrl_Sum and Wrl_Mult, respectively). The translation of the two axioms characterising the behaviour of # in the *Sum* mode is as follows:

$$\begin{array}{l}\forall\ world : World\\ \bullet\ \forall\ n : Nat\ \bullet\ (\#(Wrl\_Sum, n\ 0(Wrl\_Sum))) = n\end{array}$$

$$\begin{array}{l}\forall\ world : World\\ \bullet\ \forall\ n, m : Nat\\ \quad \bullet\ (\#(Wrl\_Sum, n, suc(Wrl\_Sum, m)))\\ \quad\ \ = (suc(Wrl\_Sum, (\#(Wrl\_Sum, n, m))))\end{array}$$

The next step is to check for properties. For illustration purposes, consider the three properties below. The first one states monotonicity of addition; the second the cyclic character of the *Shift* modality; and the third represents the equation $n + n = n * 2$.

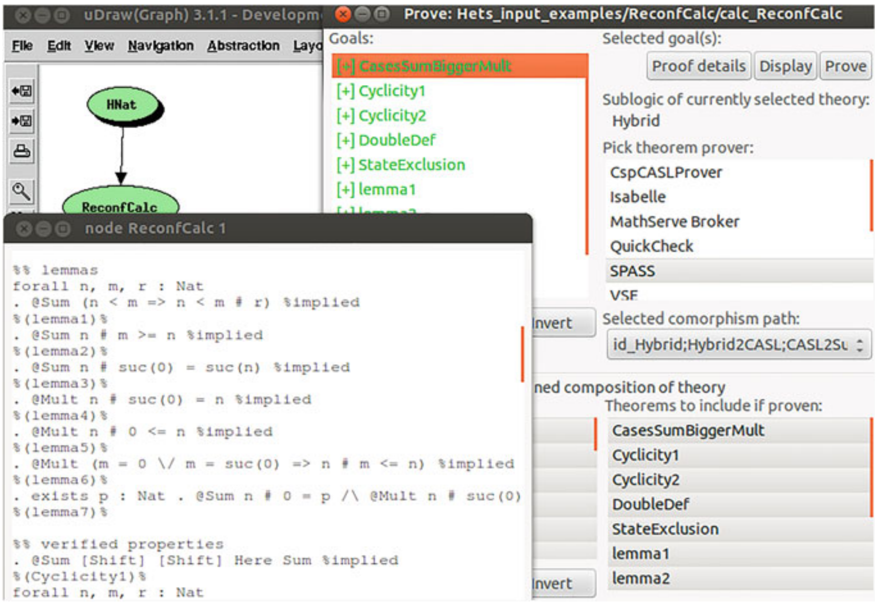
$$\begin{array}{ll}\forall\ n, m, r : Nat & \\ \bullet\ @Sum\ (n < m \Rightarrow n < m\ \#\ r) & \%1\%\\ \bullet\ \exists\ p : Nat & \\ \quad \bullet\ @Sum\ n\ \#\ m = p \wedge @Sum < Shift > < Shift > n\ \#\ m = p & \%2\%\\ \bullet\ \exists\ p : Nat\ \bullet\ @Sum\ n\ \#\ n = p \Rightarrow @Mult\ n\ \#\ suc(suc(0)) = p & \%3\%\end{array}$$

The CASL-translations computed for these properties are, respectively,

$$\begin{array}{ll}\forall\ world : World & \\ \bullet\ \forall\ n, m, r : Nat & \\ \quad \bullet\ <(Wrl\_Sum, n, m) & \\ \quad\ \ \Rightarrow\ <(Wrl\_Sum, n, & \\ \qquad\qquad (\#(Wrl\_Sum, m, r : Nat))) & \%1\%\end{array}$$

$$\begin{array}{ll}\forall\ world : World & \\ \bullet\ \forall\ n, m : Nat & \\ \quad \bullet\ \exists\ p : Nat & \\ \qquad \bullet\ (\#(Wrl\_Sum, n, m)) = p & \\ \qquad\ \ \wedge\ \neg\ \forall\ world0 : World & \\ \qquad\qquad \bullet\ Acc\_Shift(Wrl\_Sum, world0) & \\ \qquad\qquad\ \ \Rightarrow\ \forall\ world1 : World & \\ \qquad\qquad\quad \bullet\ Acc\_Shift(world0, world1) & \\ \qquad\qquad\quad\ \ \Rightarrow\ \neg\ (\#(world1 : World, n, m)) = p & \%2\%\end{array}$$

$$\begin{array}{l}\forall\ world : World\\ \bullet\ \forall\ n : Nat\\ \quad \bullet\ \exists\ p : Nat\end{array}$$



```
%% lemmas
forall n, m, r : Nat
. @Sum (n < m => n < m # r) %implied
%(lemma1)%
. @Sum n # m >= n %implied
%(lemma2)%
. @Sum n # suc(0) = suc(n) %implied
%(lemma3)%
. @Mult n # suc(0) = n %implied
%(lemma4)%
. @Mult n # 0 <= n %implied
%(lemma5)%
. @Mult (m = 0 \/ m = suc(0) => n # m <= n) %implied
%(lemma6)%
. exists p : Nat . @Sum n # 0 = p /\ @Mult n # suc(0)
%(lemma7)%

%% verified properties
. @Sum [Shift] [Shift] Here Sum %implied
%(Cyclicity1)%
forall n, m, r : Nat
```

Fig. 2 A HETS session for the swinging calculator

- $(\#(Wrl\_Sum, n, n)) = p$
  $\Rightarrow (\#(Wrl\_Mult, n, suc(Wrl\_Mult, suc(Wrl\_Mult, 0(Wrl\_Mult)))))$
  $= p$ %3%

Once translated, all these properties are easily proved by one of the provers plugged into the HETS platform, for example SPASS. Figure 2 registers an HETS session relative to this example showing the proof window, part of the model theory, and the specification graph.

3.4 From Boilerplates to $\mathcal{H}$CASL Specifications

In order to facilitate the use of hybridised logics in real world specification projects, a language of boilerplates for modelling requirements of reconfigurable systems was proposed by the authors [37]. In the discipline of requirements engineering, a *boilerplate* [29] is defined as a simplified, normative English text, intended to capture software requirements in a controlled way. It is supposed to be highly reusable and amenable to some form of computer-based simulation.

The term derives from steel manufacturing, where it refers to steel rolled into large plates for use in steam boilers. The intuition is that a boilerplate has been time-tested and is "strong as steel" suitable for repeated reuse. Our starting point in the above cited paper was that *the use of "controlled natural language" for requirements elicitation is a successful practice in industry and, despite of its informal character, provides an interesting starting point towards more formal approaches* [37].

This approach is extended in the present paper by providing a systematic translation scheme of this language of boilerplates to hybridised specifications in $\mathcal{H}$CASL. Once the system's requirements are captured by a collection of boilerplates which, taken jointly, specify a structured transition system, a formal specification is generated in $\mathcal{H}$CASL. The latter can then be handled through HETS . Its states, corresponding to different *configurations*, or *modes of execution*, are endowed with a specific description of the functionality available locally. The boilerplates define globally the relevant modes of execution and the transition structure, as well as, at the local level, the interface of services available and their properties.

The role of this tool is illustrated through the *swinging calculator* example discussed above. Figure 3 shows a fragment of the relevant requirements captured as boilerplates. The language comprises different classes of boilerplates to deal with different kinds of requirements. Figure 4 contains the translator output, i.e., the derived $\mathcal{H}$CASL specification. At this stage both texts offer no difficulty and the reader can appreciate the translation process. Note, however, that specifications of real systems can become rather complex, which advises the use of boilerplates. On the other hand, it should also be mentioned that not all design features can be suitably expressed through boilerplates, a few of them requiring some fine tuning directly over the specification. A complete account of the language of boilerplates is given in the paper mentioned above [37].

```
System's interface is defined by {
 sorts Nat
 op __#__ : Nat * Nat -> Nat
 op 0 : Nat
}.

System has events Shift.
System has modes Sum, Mult.

Property Mult does not hold in mode Sum.
Either mode Sum is active or mode Mult is active.

System changes from Sum to Mult through event Shift.
System may change from Sum to Mult through event Shift.
System changes from Mult to Sum through event Shift.
System may change from Mult to Sum through event Shift.

Property forall n,m, p: Nat. n # (m # p) = (n # m) # p holds in all modes.
Property forall n, m: Nat. n # m = m # n  holds in all modes.
Property forall n,m: Nat. n # 0 = n holds in mode Sum.
```

Fig. 3 Requirements for the swinging calculator encoded in boilerplates

```
logic Hybrid
spec X =
 sorts Nat
 op __#__ : Nat * Nat -> Nat
 op 0 : Nat

modalities Shift
nominals Mult,Sum

 . @ Sum  not Here Mult
 . Here Sum \/ Here Mult
 . @ Sum < Shift > Here Mult
 . @ Sum [ Shift ] Here Mult
 . @ Mult < Shift > Here Sum
 . @ Mult [ Shift ] Here Sum
 .  forall n,m, p: Nat. n # (m # p) = (n # m) # p
 .  forall n, m: Nat. n # m = m # n
 . @ Sum  forall n,m: Nat. n # 0 = n
end
```

Fig. 4 The derived $\mathcal{H}$CASL specification

The first boilerplate describes the system interface at each local state. Then the relevant configurations (`Sum` and `Mult`) are declared as well as the event labelling the transition from one to the other. The definition of the configurations proceeds with the third group of boilerplates which describes a number of properties to be respected. The transition structure is described afterwards; notice how expression “changes” is translated to a “diamond” modality (emphasising that an effective transition will take place), whereas expression “may change” leads to a “box” modality: the event under consideration, if present, can only result in such a transition. Finally, the last lines in Fig. 3 are examples of boilerplates for capturing properties of the system’s functionality at different configurations.

4 An Application to the Design of a Specification Course

The ideas behind hybridisation and hybridised logics were further tested in the design of a specification course in the curriculum of the Computer Science undergraduate degree at Universidade Minho, Portugal. The underlying motivation was to explore a uniform framework for specifying system’s requirements either *functional* (i.e., relative to the meaning of individual services or operations) or *behavioural* (i.e., relative to its overall evolution and reaction to external stimulus), and to emphasise a strong connection between *modelling* and *verification*.

The course rationale. The course has a standard typology: a lecture per week (1 h), an exercises class devoted to pen-and-pencil resolution of exercises previously proposed and their discussion (2 h) and a laboratory session with the HETS system (1 h). Students work in groups of two elements.

The course develops around a triangle whose vertices are repeatedly revisited: the *models*, the *languages* in which such models and their properties are expressed and the *satisfaction relation* between them, which enables property verification and design assessment. Another methodological option concerned the adoption of a *generic framework*, in which progressively more elaborated requirements could be represented, in contrast to one with a narrower scope or clearly oriented to a particular specification style. This has the advantage of focusing students and enhancing their ability to work at higher abstraction levels.

This favoured the choice of an institutional approach and the hybridisation method described in the previous sections, computationally supported by the HETS framework.

The course structure. As expected, the course targets *reconfigurable* systems, whose components may evolve in time through a number of different stages or modes of operation, in which specific service configurations are made available through their interfaces. The envisaged teaching/learning process develops around three specification stages: *algebraic*, *modal* and *hybrid*. The idea is to cover the whole spectrum of basic specification logics in three course units, all of them sharing HETS as the common tool support. A fourth unit in the syllabus explores a number of case-studies in the project of reconfigurable systems. The course illustration in Sect. 5 is taken from this last unit. Before that, let us review the *rationale* under each of them.

The algebraic stage. At a first stage each system *configuration* is specified axiomatically as a "stand-alone" *algebraic theory*; its model being a concrete algebra satisfying such a theory. Component's functionality is therefore given in terms of input-output relations modeling operations on *data*. This stage covers the classical concepts in algebraic specification, namely those of *signature*, *sentence*, *equation* and equational reasoning, *model* and *satisfaction of an equation*. The envisaged learning outcome is the ability to master these concepts and capture informal requirements about component's functionality by defining a (syntactic) *universe of discourse* and formulating properties as axioms.

The modal stage. The second stage emphasises the *reactive* nature of the systems at hands. Component's evolution is modelled by a transition system: a configuration changes in response to a particular event in the system. Modal logics are introduced as specification languages for state transition systems. Modal formulas are evaluated inside such systems, at a particular state, and modal operators disclose access to information stored at other states accessible from the current one via a suitable transition. The main learning outcome is to make students familiar with the modal framework and the meaning of modalities as a language to specify transition structures.

The hybrid stage. The third stage starts with a crucial observation: functional and transitional behaviour are strongly interconnected in practice as the functionality offered by the system, at each moment, may depend on the stage of its evolution. This entails the need for

- enriching the basic modal language with the ability to refer to *individual* states, regarded as possible system's configurations or modes of operation;
- distinguishing *global* behaviour (in the underlying transition system) from *local* behaviour expressed, at each state, by a particular specification.

The first requirement leads to the introduction of *nominals* as explicit references to specific states of the underlying transition system. Conceptually this exposes students to another basic and pervasive notion in Computer Science, that of *naming*. Hybrid logics [10] are the appropriate tool for this last stage in the course. The need for formulating specific *local* requirements, on the other hand, imposes extra structure upon states. Actually, different states are interpreted as different *modes* of operation and each of them is equipped with an algebraic specification of the corresponding functionality. Technically, specifications become *structured* state-machines, where states are specified as *algebras*, rather than as *sets*.

As mentioned in the previous section, HETS provides for free the proof support environment needed for this course. The boilerplates translator introduced in the previous section can also be used in the course to directly generate $\mathcal{H}$CASL specifications. Its pedagogical value, in training students to write specifications, is greatly appreciated. It should be stressed, however, that, in despite of the crucial role played by institution theory in this approach, no familiarity with institutions is required from students.

5 A Glimpse of a Course Session

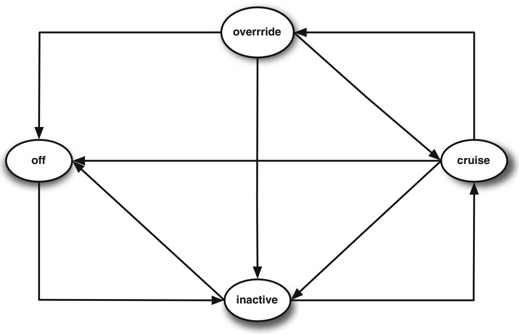
The course contents and methodology are better understood through the presentation of a typical problem addressed first in the exercises class and later in the laboratory, in the last stage of the course. For space limitations we only focus on a fragment of the original problem. The example, small but self-contained, is taken from a description of requirements for an *automatic cruise control* (ACC) system summarised in [30] as follows:

> The mode class CruiseControl contains four modes, Off, Inactive, Cruise, and Override. At any given time, the system must be in one of these modes. Turning the ignition on causes the system to leave Off mode and enter Inactive mode, while turning the cruise control level to const when the brake is off and the engine running causes the system to enter Cruise mode. (…) Once cruise control has been invoked, the system uses the automobile's actual speed to determine whether to set the throttle to accelerate or decelerate the automobile, or to maintain the current speed (…)To override cruise control (i.e., enter Override), the driver turns the lever to off or applies the brake.

These requirements are captured by the state machine depicted in Fig. 5 and expressed in *hybrid propositional logic* ($\mathcal{H}PL$).

A modality *next* is introduced to denote the state-machine accessibility relation. Nominals in set {*off*, *inactive*, *override*, *cruise*} correspond to the operation modes mentioned in the requirements. The first element students can formally capture within the logic is the transition structure, as in, for example,

Fig. 5 The transition structure



- (T_1) $@_{\mathit{off}}\ \langle \mathit{next}\rangle\ \mathit{inactive}$
- (T_2) $@_{\mathit{override}}\ (\langle \mathit{next}\rangle\ \mathit{off} \wedge \langle \mathit{next}\rangle\ \mathit{inactive} \wedge \langle \mathit{next}\rangle\ \mathit{cruise})$

Local properties can also be expressed through the satisfaction operator $@_i$, for each nominal i, to refer to the corresponding state. For instance, the requirement that the ignition is off when the system is in the *off* mode, while it is *on* and the engine running (*EngRunning*) in the *cruise* mode, is modelled by

- (L_1) $@_{\mathit{off}}(\neg \mathit{IgnOn})$
- (L_2) $@_{\mathit{cruise}}(\mathit{IgnOn} \wedge \mathit{EngRunning})$

Symbols *EngRunning* and *IgnOn*, with a self-explanatory designation, are propositions whose validity is discussed in each configuration (state). Others are used in the sequel. Definitional properties can also be captured, as in

- (A_1) $\mathit{LeverOff} \Leftrightarrow \neg\ \mathit{LeverCons}$
- (A_4) $\mathit{HighSpeed} \Rightarrow \neg\ \mathit{CruiseSpeed} \wedge \neg\ \mathit{LowSpeed}$

The second step in the case study is to equip each state of the underlying transition system with a first-order structure, to model its local functionality. Therefore, hybrid structures are enriched with a family of first-order structures indexed by the set of states, i.e., they become structures (M, W) where function M defines a family $(M_w)_{w\in|W|}$ of first-order structures over the same signature and universe (constraint necessary for the conservativity of the $\mathcal{H}FOL2FOL$ encoding). Each M_w models the system's behaviour at state $w \in W$. Note that at state w each first order formula is evaluated in the structure M_w. Properties are now expressed in a hybrid first order language $\mathcal{H}FOL$ whose detailed presentation we omit here (but see [35]). We focus instead on the sort of properties students are supposed to formulate. An algebraic specification is used to model system's functionality. This entails the need for introducing data types able to support the envisaged notions of *time*, *speed* and *acceleration*.

spec TIMESORT =INT
with sort *Int* $\mapsto$ *time*, **ops** 0 $\mapsto$ *init*, *suc* $\mapsto$ *after* **end**
spec SPEEDSORT =INT **with sort** *Int* $\mapsto$ *speed* **end**
spec ACELLSORT =INT **with sort** *Int* $\mapsto$ *accel* **end**

Operation *Pedal* models the accelerations applied by the driver at each moment. On the other hand, *Automatic* captures accelerations applied on the engine by the ACC, and *CurrentSpeed* records the current speed. Finally, constant *MaxCruiseSpeed* represents the maximum speed allowed on the ACC mode:

spec ACCSIGN =
  TIMESORT **and** SPEEDSORT **and** ACELLSORT
then ops $Pedal : time \to accel;$
  $Automatic : time \to accel;$
  $Speed : speed \times accel \to speed;$
  $CurrentSpeed : time \to speed;$
  $MaxCruiseSpeed : speed$

Students are asked to identify properties that globally hold, in all possible configurations, and the ones which model local requirements. In the first group we have, for example,

$\forall\, s : speed; a : accel; t : time$
$\bullet\ (G_1)\ \ Speed(s, a) \geq 0$
$\bullet\ (G_2)\ \ CurrentSpeed(t) = 0 \wedge Pedal(t) \geq 0 \Rightarrow$
$CurrentSpeed(after(t)) \geq 0$
$\bullet\ (G_3)\ \ Pedal(t) > 0 \Leftrightarrow CurrentSpeed(t) < CurrentSpeed(after(t))$
$\bullet\ (G_4)\ \ Speed(s, a) = s \Leftrightarrow a = 0$
$\bullet\ (G_5)\ \ CurrentSpeed(after(t)) = Speed(CurrentSpeed(t), Pedal(t))$

Local properties refer to specific configurations. For example, in state *off*, *Speed* and *Pedal* are null and no other operation in the interface react. Thus,

$\forall\, t : time; s : speed; a : accel$
$\bullet\ (L^1_{off})\ \ @_{off} CurrentSpeed(t) = 0$
$\bullet\ (L^2_{off})\ \ @_{off} Speed(s, a) = 0$

On the other hand, in state *inactive*, speed and acceleration depend on the accelerations automatically introduced in the system, i.e.,

$\forall\, s : speed; a : accel$
$\bullet\ (L^1_{inactive})\ \ @_{inactive} Speed(s, a) = s + a$

$\forall\, t: time; s : speed; a : accel$
$\bullet\ (L^{1'}_{cruise})\ \ @_{cruise}[CurrentSpeed(t) > MaxCruiseSpeed \Rightarrow Automatic(after(t)) < 0]$
$\bullet\ (L^{2'}_{cruise})\ \ @_{cruise}[CurrentSpeed(t) \leq MaxCruiseSpeed \Leftrightarrow Automatic(after(t)) = 0]$
$\bullet\ (L^3_{cruise})\ \ @_{cruise} Speed(s, a) = s + a$
$\bullet\ (L^4_{cruise})\ \ @_{cruise} Pedal(t) \geq 0 \Rightarrow Pedal(t) = Automatic(t)$

An interesting feature in this example is that properties local to states *override* and *off* do coincide. The system's behaviour on both states only differs in what concerns the definition of the allowed transitions. Actually, students may now be invited to revisit the specification of the transition system presented above. It turns out that some propositions may be re-stated by means of properties of local states. For instance,

$\forall$ *t*: *time*;
• (L_1) $@_{cruise}[CurrentSpeed(t) = 0 \Rightarrow \langle next\rangle^u(inactive \wedge CurrentSpeed(after(t)) = 0)]$

where $\langle\lambda\rangle^u\rho$ abbreviates $\langle\lambda\rangle\rho \wedge [\lambda]\rho$.

Finally, in the laboratory session students are invited to translate hybrid to first order specifications and use Hets to animate them. On translating to $\mathcal{H}FOL2FOL$ we end up with the following signature (see Fig. 6):

ops
$Speed^* : st^* \times speed \times accel \to speed;$
$Pedal^* : st^* \times time \to accel;\dots$
pred
$next : st^* \times st^*;\ IgnOn^* : st^*;\ \dots$

where global properties are universally quantified, and local properties take as an argument the respective nominal. For instance, global properties (G_1) and (G_2) are translated into

$\forall\, s : speed;\ w : st^*;\ a : accel; t : time$
• (G_{1^*}) $\geq^*(w, Speed^*(w, s, a), 0^*(w))$
• (G_{2^*}) $CurrentSpeed^*(w,t) = 0^*(w) \wedge \geq^*(w, Pedal^*(w,t), 0^*(w))$.

and local properties (L^1_{off}) and (L^4_{cruise}), into

$\forall\, t : time$
• $(L^{1^*}_{off})$ $CurrentSpeed^*(off,t) = 0^*(off)$
• $(L^{4^*}_{cruise})$ $\geq^*$
$(cruise, Pedal^*(cruise,t), 0^*(cruise)) \Rightarrow Pedal(cruise,t) = Automatic^*(cruise,t)$.

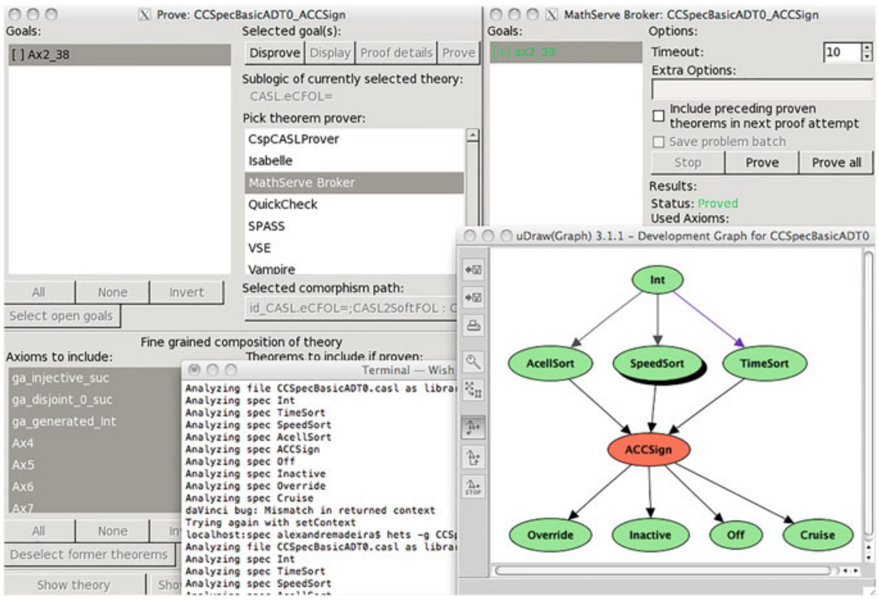
6 A Step Ahead: The Power of Quantification

6.1 Introducing Full Quantification

This section introduces a new, major extension to the method surveyed in the previous sections to support quantification. This requires the inclusion of another parameter in the method: a *quantification space*.[1] $\mathcal{D}^{\mathcal{H}I}$ for $\mathrm{Mod}^{\mathcal{H}\mathcal{I}}$.

In the institutional framework, as a subclass of $\mathrm{Sign}^{\mathcal{H}\mathcal{I}}$, quantification morphisms consist of triples $\chi = (\chi_{\mathrm{Sig}}, \chi_{\mathrm{Nom}}, \chi_{\mathrm{MS}}) : (\Sigma, \mathrm{Nom}, \Lambda) \to (\Sigma', \mathrm{Nom}', \Lambda')$. Each of these components is responsible for a particular kind of quantification. We are particularly interested in inclusion morphisms, which are the ones that give rise to standard

[1] Quantification spaces are extensively discussed in Madeira's thesis [34], as well as in a joint paper with Diaconescu [23].



```
Analyzing file CCSpecBasicADT0.casl as librai
Analyzing spec Int
Analyzing spec TimeSort
Analyzing spec SpeedSort
Analyzing spec AcellSort
Analyzing spec ACCSign
Analyzing spec Off
Analyzing spec Inactive
Analyzing spec Override
Analyzing spec Cruise
daVinci bug: Mismatch in returned context
Trying again with setContext
localhost:spec alexandremadeira$ hets -g CCS
Analyzing file CCSpecBasicADT0.casl as librai
Analyzing spec Int
Analyzing spec TimeSort
Analyzing spec SpeedSort
```

Fig. 6 A HETS session

quantifications. For example, considering $\chi_{\text{Nom}} : \text{Nom} \hookrightarrow \text{Nom} + Y$, for Y a finite set of constants, and considering χ_{MS} and χ_{Sig} the identity morphisms, we obtain the standard state quantification that can be found in the literature.

In the (fully) quantified version of the method proposed in this paper, the set $\text{Sen}^{\mathcal{HI}}(\Delta)$ is enriched with the sentence $(\forall\chi)\rho$, for any $\chi : \Delta \to \Delta' \in \mathcal{D}^{\mathcal{HI}}$ and $\rho \in \text{Sen}^{\mathcal{HI}}(\Delta')$. Similarly, the translation of sentences is extended, in each morphism $\varphi : \Delta \to \Delta_1$, by $\text{Sen}^{\mathcal{HI}}(\varphi)\big((\forall\chi)\rho\big) = (\forall\chi(\varphi))\text{Sen}^{\mathcal{HI}}(\varphi[\chi])(\rho)$. Finally, in what concerns the satisfaction relation, we consider

- $(M, W) \models^w (\forall\chi)\rho$ iff $(M', W') \models^w \rho$

for any (M', W') such that $\text{Mod}^{\mathcal{HI}}(\chi)(M', W') = (M, W)$. Existential quantification is introduced in a similar way.

In standard logical terminology, given an inclusion morphism

$$\chi = (\chi_{\text{Sig}}, \chi_{\text{Nom}}, \chi_{\text{MS}}) : (\Sigma, \text{Nom}, \Lambda) \to (\Sigma', \text{Nom}', \Lambda')$$

where $\chi_{\text{Nom}} : \text{Nom} \hookrightarrow \text{Nom} + U$ and $\chi_{\text{MS}} : \Lambda \hookrightarrow \Lambda + Y$, for finite sets $U = \{u_1, u_2, \ldots, u_n\}$ and $Y = \{y_1, y_2, \ldots, y_m\}$, the new sentence $(\forall\chi)\rho$ may be written as $\forall_{u_1,u_2,\ldots,u_n} \forall_{y_1,y_2,\ldots,y_m} \rho$. Moreover, one can say that $(M, W) \models^w (\forall\theta)\rho$ iff, for any θ-expansion $(M, W)^\theta$ of (M, W), one has $(M, W)^\theta, w \models \rho$.

Quantified sentences play a major role in specification theory. Actually,

- Quantification coming from the base institution can be used to specify local configurations.
- Quantification over nominals, makes possible to express properties about the system's global state space. This is particularly useful, for instance, to express the existence of configurations satisfying a given requirement.
- Quantification over modalities, finally, constitutes a rather powerful form of quantification useful to express enabling/disabling of reconfigurations.

The last two types of quantification are explored below as a very general way to introduce dynamic modalities. Specifically, quantification over nominals and over modalities makes possible to express paradigmatic changes on the relational model, like *swapping* and *sabotage*. This is done at minimal cost and in a very general way which captures several approaches in the literature which are specific to particular situations.

6.2 Effects and Dynamic Modalities

Suppose you take a train and start planning your trip as you go. With a proper map the task is quite straightforward. But *what if the transportation system breaks down, and a malevolent demon starts canceling connections, anywhere in the network?* This question appears in the motivation section of van Benthem seminal paper on sabotage logic [54]. The scenario is as follows: there is a transition structure (the map, a graph) over which sentences are interpreted as usual in modal logic; however this may change dynamically while being traversed.

Sabotage logic is an example of a modal logic equipped with modalities that can change the accessibility relation of the underlying Kripke model along the evaluation of a formula. In particular, edges are deleted. Adding new edges or swapping existent ones are further examples of effects leading to logics which, over time, have found interesting applications in describing and reasoning about dynamic aspects of phenomena. Some recent papers [1–3] explore specific instances of these ideas further witnessing their relevance to application areas ranging from reconfigurable software specifications to changing obligations contexts in epistemic logics. In these logics the meaning of the basic modal operators remains unchanged, but new ones, suitably called *dynamic modalities*, are introduced to encode specific changes in the accessibility relation.

Our approach aims at going a step forward. Instead of formulating new, tailor-made logics for each family of effects, we resort to the fully quantified hibridisation of the Triv institution, in which the typical dynamic modalities in the literature can be captured in a uniform way and within a unique logic. The introduction of quantification over modality symbols allows not only a suitable encoding of effects, like reversing or deleting transitions, but also the precise specification of their scope (e.g., the whole or part of the accessibility relation) and the point of application (e.g., anywhere, relative to the current evaluation point, an edge between specific named states, etc.). This goes beyond and generalises current approaches in the literature. The only work we are aware of with a similar spirit, but through a different way, is a very recent paper by Areces et al. [4] which proposes a characterisation of what the authors call *relation-changing modal operators*. Actually, our approach differs from the one above, by the ability to express a bigger diversity of effects. The reason is that we resort to an abstract hybrid logic and, through nominals, it is possible to express changes in specific points of the relational structure.

Besides providing a uniform setting to discuss dynamic modalities, and, more generally, *effects* over Kripke models, the main advantage of the approach introduced here is the possibility to characterise typical results in the study of these logics in a generic way, for example a general notion of bisimulation parametric on the effect. Finally note that, in the approach proposed here, and contrary to what appears in the literature, models remain standard Kripke structures, no actual updating taking place in the accessibility relation. The effect of dynamic modalities is to expand the original relation into a new, updated one and, then, to hand it over the current evaluation point.

6.2.1 Effects and Events

An *effect* $E(X, Y, x, y)$ captures a specific transformation, or update, of an accessibility relation X in a Kripke model. It can be regarded as a *macro* relating two accessibility relations X and Y. For example the *swap effect*, which inverts in Y the orientation of an edge in X, is specified as

$$\textbf{(Swap)} \qquad Sw(X, Y, x, y) \overset{abv}{=} @_x\langle X\rangle y \wedge @_y\langle Y\rangle x$$

The *sabotage* effect, which ignores in Y the edge (x, y) of X, is given by

$$\textbf{(Sabotage)} \qquad Sg(X, Y, x, y) \overset{abv}{=} @_x\langle X\rangle y \wedge \neg @_x\langle Y\rangle y$$

Enriching X with a specific new edge, is expressed through the *bridge* effect:

$$\textbf{(Bridge)} \qquad Bg(X, Y, x, y) \overset{abv}{=} \neg @_x\langle X\rangle y \wedge @_x\langle Y\rangle y$$

Weaker forms of the two latter effects can also be considered:

$$\textbf{(Conditional Sabotage)} \qquad PSg(X, Y, x, y) \overset{abv}{=} @_x\langle X\rangle y \to \neg @_x\langle Y\rangle y$$

$$\textbf{(Conditional Bridge)} \qquad PBg(X, Y, x, y) \overset{abv}{=} \neg @_x\langle X\rangle y \to @_x\langle Y\rangle y$$

An effect can act upon a given, specific edge (x, y), or a set of edges. This is called the range (**rng**) of an effect—*exclusive* (denoted by **o**) or *partial* (**p**). Once this specified for a particular effect, the resulting expression is called an *event*. Formally, given an effect E, an E-event $E_{\mathbf{rng}}(X, Y, x, y)$ with $\mathbf{rng} \in \{\mathbf{p}, \mathbf{o}\}$ is a sentence in $\mathcal{H}Triv$ such that

- $E_{\mathbf{p}}(X, Y, x, y) \overset{abv}{=} E(X, Y, x, y) \wedge Ex_E(X, Y, x, y)$
- $E_{\mathbf{o}}(X, Y, x, y) \overset{abv}{=} E(X, Y, x, y) \wedge U(X, Y, x, y)$

where

- $Ex_E(X, Y, x, y) \overset{\text{def}}{=} (\forall s, v)\big((@_s\langle X\rangle v \leftrightarrow @_s\langle Y\rangle v) \vee (E(X, Y, s, v) \wedge @_s x)\big)$
- $U(X, Y, x, y) \overset{\text{def}}{=} (\forall s, v)\big((@_s\langle X\rangle v \leftrightarrow @_s\langle Y\rangle v) \vee (@_s x \wedge @_v y)\big)$

Intuitively, expression $Ex_E(X, Y, x, y)$ asserts that an edge with source in x can only be updated, on going from X to Y, as result of effect E. Apart from this, relations X and Y remain equal. Expression $U(X, Y, x, y)$, on the other hand, establishes that any modification affects exclusively the pair of states x and y.

Let us illustrate this construction with the event **o**-*swap* for edge (x, y):

$$\begin{aligned} Sw_{\mathbf{o}}(X, Y, x, y) &\overset{\text{def}}{=} Sw(X, Y, x, y) \wedge U(X, Y, x, y) \\ &= \big(@_x\langle X\rangle y \wedge @_y\langle Y\rangle x\big) \wedge (\forall s, v)\big(@_s\langle X\rangle v \leftrightarrow @_s\langle Y\rangle v \ \vee (@_s x \wedge @_v y)\big) \end{aligned}$$

where relation Y is constructed by swapping exactly the edge (x, y) of X. The partial range version of this event, **p**-*swap*, is

$$\begin{aligned} Sw_{\mathbf{p}}(X, Y, x, y) \overset{abv}{=}\ & Sw(X, Y, x, y) \wedge Ex_{Sw}(X, Y, x, y) \\ =\ & \big(@_x\langle X\rangle y \wedge @_y\langle Y\rangle x\big) \wedge \\ & (\forall s, v)\big((@_s\langle X\rangle v \leftrightarrow @_s\langle Y\rangle v) \vee (Sw(X, Y, s, v) \wedge @_s x)\big) \\ =\ & \big(@_x\langle X\rangle y \wedge @_y\langle Y\rangle x\big) \wedge \\ & (\forall s, v)\big((@_s\langle X\rangle v \leftrightarrow @_s\langle Y\rangle v) \vee ((@_s\langle X\rangle v \wedge @_v\langle X\rangle s) \wedge @_s x)\big) \end{aligned}$$

As expected, the new accessibility relation Y is identical to X, but on a number of swapped edges with source in x. The result of a partial *swap* and a partial *sabotage* event is depicted in Figs. 8 and 9, respectively (over the same relation X depicted in Fig. 7).

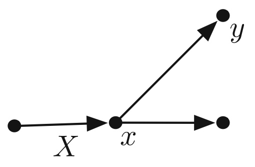


Fig. 7 The original relation X

Fig. 8 *X* swapped

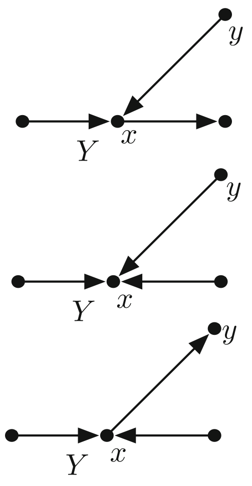
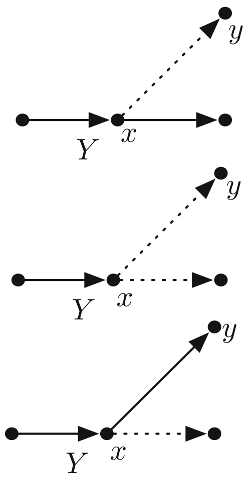


Fig. 9 *X* sabotaged



6.2.2 Dynamic Modalities

Dynamic modalities are built from the *events* introduced in the previous section. Please note that there is no actual update of the accessibility relation. A dynamic modality expands the original model with a new, modified relation with reference to which evaluation proceeds. For any event $E_{\mathbf{rng}}(X, Y, x, y)$, two dynamic modalities are defined: a *local* and a *global* modality. The first one is defined by

$$\textbf{(Local)} \quad \ll E_{\mathbf{rng}}(X) \gg_l \rho \overset{\text{def}}{=} (\exists Y, x, y)\big(x \wedge E_{\mathbf{rng}}(X, Y, x, y) \wedge @_y \rho^Y_X\big)$$

where Y, x, y are variables not occurring in ρ.

The intuition is that event E is performed in possible edges whose source is the current evaluation point, which then changes through a transition over an updated edge. The global modality, on the other hand, is defined by

$$\textbf{(Global)} \quad \ll E_{\mathbf{rng}}(X) \gg_g \rho \overset{\text{def}}{=} (\exists Y, x, y)\big(E_{\mathbf{rng}}(X, Y, x, y) \wedge \rho_X^Y\big)$$

where Y, x, y are variables not occurring in ρ. In this case the event is performed at some point in the model and the current evaluation point does not change. Observe that substitution ρ_X^Y represents the "shift" between the original relation X by the "updated" one Y.

As usual, corresponding boxed dynamic modalities are obtained through

$$[[E_{\mathbf{rng}}(X)]]_l \, \rho \overset{\text{def}}{=} (\forall Y, x, y)\big((x \wedge E_{\mathbf{rng}}(X, Y, x, y)) \to @_y \rho_X^Y\big)$$
$$[[E_{\mathbf{rng}}(X)]]_g \, \rho \overset{\text{def}}{=} (\forall Y, x, y)\big(E_{\mathbf{rng}}(X, Y, x, y) \to \rho_X^Y\big)$$

where Y, x, y are variables not occurring in ρ and ρ_X^Y is the sentence obtained by substituting all the occurrences of X by Y. As expected, for any formula $\rho \in Fm(\text{Nom}, \text{Prop}, \Lambda)$, correspondences

$$\neg \ll E_{\mathbf{rng}}(X) \gg_l \neg\rho \leftrightarrow [[E_{\mathbf{rng}}(X)]]_l \, \rho$$

and

$$\neg \ll E_{\mathbf{rng}}(X) \gg_g \neg\rho \leftrightarrow [[E_{\mathbf{rng}}(X)]]_g \, \rho$$

hold.

7 Concluding

The hybridisation method discussed in this paper can be broadly understood as a specific way of combining logics at the model theoretical level. Actually, it classifies as *a tool for simplifying problems involving heterogeneous reasoning*, a common ingredient to this family of methods according to the corresponding entry in the *Stanford Encyclopedia of Philosophy* [16]. The same entry stresses the role of Computer Science applications as a main driving force for research in obtaining new logic systems from old: *One of the main areas interested in the methods for combining logics is software specification. Certain techniques for combining logics were developed almost exclusively with the aim of applying them to this area.* [16].

More specifically, hybridisation is a form of asymmetric combination of logics in the sense that specific features of hybrid logic are developed "on top" of another logic. This follows the pattern of, and to a certain extent extends, previous work by Diaconescu and Stefaneas [24] on "modalisation" of institutions, which endows systematically institutions with Kripke semantics for standard modalities. The institutional setting [7] in which we worked offers a suitable framework to discuss the generation of new logics from old, and to identify the sort of properties preserved or reflected along such a process. As in many other areas of theoretical Computer Science, going categorial means going generic.

In the following paragraphs we briefly discuss some directions for future work. The first is concerned with the extension of the educational application of the hybridisation method described above. The other two are specific research challenges on pushing forward the method reviewed in this paper.

A curricular challenge. Sects. 4 and 5 introduced the *rationale* for a somehow not very standard introductory course to software specification with hybrid(ised) logics. Building on an institution-based framework kept implicit along the lectures, the course aims at conducting students through two orthogonal paradigms (equational and hybrid) which are then combined in a common specification framework.

The approach underlying the course is based on a particular instance of the hybridisation method. However, other possible "hybridisations" (e.g., of institutions of multialgebras or partial algebras) are suitable to explore a wide range of exercises in a similar spirit. Moreover, the course skills may be easily expanded into new directions: for instance, functional and imperative programming languages may be presented as institutions (see [52]) whose hybridisation may be used to develop reconfigurable algorithms. On a different note, a two-level hybridisation of a base logic, as discussed in [34], provides modalities and nominals at two different levels: local and global. This seems a suitable setting to talk about reconfigurable software applications whose local configurations are also described by transition systems. More generally, models become *hierarchical* transition systems. In [44], the authors have also presented the logic underlying ALLOY [32] in an institutional setting. This paves the way to hybridising ALLOY and combining in the course the use of the traditional ALLOY model finder with theorem proving (in HETS) in an integrated way.

Beyond reconfigurability, hybridised logics may provide flexible frameworks to address related problems in software design, namely those concerning adaptation and software evolution.

Hybridisation for quantitative reasoning. Specification frameworks for *quantitative reasoning*, dealing for example with weighted or probabilistic transition systems, emerged recently as a main challenge for software engineers. This witnesses a shift from classical models of computation, such as labeled transition systems, to similar structures where quantities can be handled. Examples include weighted [19], hybrid [28, 33] or probabilistic [49] automata, as well as their coalgebraic rendering (e.g., [51]). An interesting topic to pursue is taking up this "quantitative" challenge within the context of the hybridisation process itself. The simplest move in such a direction proceeds by instantiation. In this case quantitative reasoning is just reflected and expressed at the *local* level of concrete, specific configurations. A complementary path may focus on generalising the underlying semantic structures, replacing the *REL*-component in models by coalgebras over suitable categories of probability distributions, metric, or topological spaces.

Calculus. Comparing the calculus for hybrid propositional logic in Ref. [14] with the one for hybrid first-order logic in [13], a common structure pops out: both "share" rules involving sentences with nominals and satisfaction operators (i.e., formulas of a "hybrid nature") and have specific rules to reason about "atomic sentences" that come from the base institution. Hence, it makes sense to consider the development of a general proof calculus for hybrid institutions on top of the calculus of the corresponding base institution, in the style of [12, 17]. Somehow anticipating the general construction, a calculus for equational hybrid logic was proposed in [11].

Recent work [45] reports preliminary general results in this direction. In particular, it is shown that, whenever the base logic has the usual Boolean connectives, hybridisation preserves decidability, and furthermore, the generated calculus is sound and complete whenever the one for the base logic is. These results have not only a theoretical interest on their own, but also pave the way for new approaches to tool supported verification.

Acknowledgments This work is financed by the ERDF—European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation—COMPETE 2020 Programme, and by National Funds through the FCT (Portuguese Foundation for Science and Technology) within project `POCI-01-0145-FEDER-006961`. M. Martins was further supported by project `UID/MAT/04106/2013`. A. Madeira and R. Neves research was carried out in the context of a post-doc and a Ph.D. grant with references `SFRH/BPD/103004/2014` and `SFRH/BD/52234/2013`, respectively. L.S. Barbosa is also supported by `SFRH/BSAB/113890/2015`.

References

1. Areces, C., Fervari, R., Hoffmann, G.: Moving arrows and four model checking results. In: Ong, L., de Queiroz, R. (eds.) Proceedings of the 19th International Workshop on Logic. Language, Information and Computation (WoLLIC 2012). Lecture Notes in Computer Science, vol. 7456, pp. 142–153. Springer, Buenos Aires, Argentina (2012)
2. Areces, C., Fervari, R., Hoffmann, G.: Tableaux for relation-changing modal logics. In: Proceedings of Frontiers of Combining Systems 2013, Nancy, France, Sept 2013
3. Areces, C., Fervari, R., Hoffmann, G.: Swap logic. Logic J. IGPL **22**(2), 309–332 (2014)
4. Areces, C., Fervari, R., Hoffmann, G.: Relation-changing modal operators. Logic J. IGPL **23**(4), 601–627 (2015)
5. Areces, C., ten Cate, B.: Hybrid logics. In: Blackburn, P., Wolter, F., van Benthem, J. (eds.) Handbook of Modal Logic. Studies in Logic and Practical Reasoning, vol. 3, pp. 822–868. Elsevier (2007)
6. Burstall, R., Diaconescu, R.: Hiding and behaviour: an institutional approach. In: Roscoe, W. (ed.) A Classical Mind: Essays in Honour of C.A.R. Hoare, pp. 75–92. Prentice-Hall (1994)
7. Burstall, R.M., Goguen, J.A.: The semantics of CLEAR, a specification language. In: Bjørner, D. (ed.) Abstract Software Specifications (1979 Copenhagen Winter School, 22 Jan–2 Feb 1979), Lecture Notes in Computer Science, vol. 86, pp. 292–332. Springer (1980)
8. Bidoit, M., Hennicker, R.: Constructor-based observational logic. J. Log. Algebr. Program. **67**(1–2), 3–51 (2006)
9. Beierle, C., Kern-Isberner, G.: Looking at probabilistic conditionals from an institutional point of view. In: Kern-Isberner, G., Rödder, W., Kulmann, F. (eds.) Conditionals, Information, and

Inference (Revised Selected Papers of WCII 2002, Hagen, Germany, 13–15 May 2002), Lecture Notes in Computer Science, vol. 3301, pp. 162–179. Springer (2005)
10. Blackburn, P.: Representation, reasoning, and relational structures: a hybrid logic manifesto. Logic J. IGPL **8**(3), 339–365 (2000)
11. Barbosa, L.S., Martins, M.A., Carreteiro, M.: A Hilbert-style axiomatisation for equational hybrid logic. J. Logic Lang. Inf. **23**(1), 31–52 (2014)
12. Borzyszkowski, T.: Logical systems for structured specifications. Theor. Comput. Sci. **286**(2), 197–245 (2002)
13. Braüner, T.: Natural deduction for first-order hybrid logic. J. Logic Lang. Inf. **14**(2), 173–198 (2005)
14. Braüner, T.: Hybrid Logic and Its Proof-Theory. Applied Logic Series. Springer (2010)
15. Cîrstea, C.: An institution of modal logics for coalgebras. J. Log. Algebr. Program. **67**(1–2), 87–113 (2006)
16. Carnielli, W., Coniglio, M.E.: Combining logics. In: Zalta, E.N. (ed.) *The Stanford Encyclopedia of Philosophy*. Winter 2011 edn. (2011)
17. Codescu, M., Găină, D.: Birkhoff completeness in institutions. Logica Universalis **2**(2), 277–309 (2008)
18. Caleiro, C., Mateus, P., Sernadas, A., Sernadas, C.: Quantum institutions. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday. Lecture Notes in Computer Science, vol. 4060, pp. 50–64. Springer (2006)
19. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. **380**(1–2), 69–86 (2007)
20. Diaconescu, Răzvan: Institution-independent Model Theory. Studies in Universal Logic. Birkhäuser Basel (2008)
21. Diaconescu, R.: On quasi-varieties of multiple valued logic models. Math. Log. Q. **57**(2), 194–203 (2011)
22. Diaconescu, R.: Quasi-varieties and initial semantics in hybridized institutions. J. Logic Comput. (2015)
23. Diaconescu, R., Madeira, A.: Encoding hybridized institutions into first-order logic. Math. Struct. Comput. Sci. 1–44 (2015) (in print)
24. Diaconescu, R., Stefaneas, P.S.: Ultraproducts and possible worlds semantics in institutions. Theor. Comput. Sci. **379**(1–2), 210–230 (2007)
25. Goguen, J.A., Burstall, R.M.: Institutions: abstract model theory for specification and programming. J. ACM **39**(1), 95–146 (1992)
26. Gottwald, S.: A Treatise on Many-Valued Logics. Studies in Logic and Computation vol. 9. Research Studies Press (2001)
27. Goguen, J.A., Roşu, G.: Institution morphisms. Formal Asp. Comput. **13**(3–5), 274–307 (2002)
28. Henzinger, T.A.: The theory of hybrid automata. In: 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96, New Brunswick, New Jersey, USA, 27–30 July 1996), pp. 278–292 (1996)
29. Hull, M.E.C., Jackson, K., Dick, J.: Requirements Engineering, 2nd edn. Springer Verlag (2005)
30. Heitmeyer, C.L., Kirby, J., Labaw, B.G.: The SCR method for formally specifying, verifying, and validating requirements: tool support. In: Richards Adrion, W., Fuggetta, A., Taylor, R.N., Wasserman, A.I. (eds.) Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, 17–23 May 1997, pp. 610–611. ACM (1997)
31. Indrzejczak, A.: Modal hybrid logic. Logic Logical Philos. **16**, 147–257 (2007)
32. Jackson, D.: Software Abstractions (Logic, Language, and Analysis), 2nd edn. MIT Press (2011)
33. Lynch, N.A., Segala, R., Vaandrager, F.W., Weinberg, H.B.: Hybrid i/o automata. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop, 22–25 Oct 1995, Ruttgers University, New Brunswick, NJ, USA). Lecture Notes in Computer Science, vol. 1066, pp. 496–510. Springer (1995)

34. Madeira, A.: Foundations and techniques for software reconfigurability. Ph.D. thesis, Universidades do Minho, Aveiro and Porto (Joint MAP-i Doctoral Programme), July 2013
35. Madeira, A., Faria, J.M., Martins, M.A., Barbosa, L.S.: Hybrid specification of reactive systems: an institutional approach. In: Barthe, G., Pardo, A., Schneider, G. (eds.) Software Engineering and Formal Methods (SEFM 2011, Montevideo, Uruguay, 14–18 Nov 2011). Lecture Notes in Computer Science, vol. 7041, pp. 269–285. Springer (2011)
36. Mossakowski, T., Haxthausen, A., Sannella, D., Tarlecki, A.: CASL: the common algebraic specification language: semantics and proof theory. Comput. Inf. **22**, 285–321 (2003)
37. Madeira, A., Martins, M.A., Barbosa, L.S.: Boilerplates for reconfigurable systems: a language and its semantics. In: Du Bois, A.R., Trinder, P. (eds.) Programming Languages—17th Brazilian Symposium, SBLP 2013, Brasília, Brazil, 3–4 Oct 2013. Proceedings. Lecture Notes in Computer Science, vol. 8129, pp. 75–89. Springer (2013)
38. Martins, M.A., Madeira, A., Barbosa, L.S., Neves, R.: Paradigm integration in a specification course. In: Joshi, J., Bertino, E., Thuraisingham, B.M., Liu, L. (eds.) Proceedings of 15th IEEE International Conference on Information Reuse and Intergration, IRI 2014, Redwood City, CA, USA, 13–15 Aug 2014, pp. 492–499. IEEE Press (2014)
39. Martins, M.A., Madeira, A., Diaconescu, R., Barbosa, L.S.: Hybridization of institutions. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) Algebra and Coalgebra in Computer Science (CALCO 2011, Winchester, UK, 30 Aug–2 Sept 2011). Lecture Notes in Computer Science, vol. 6859, pp. 283–297. Springer (2011)
40. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, Hets. In: Grumberg, O., Huth, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007—Braga, Portugal, 24 Mar—1 Apr 2007). Lecture Notes in Computer Science, vol. 4424, pp. 519–522. Springer (2007)
41. Mossakowski, T.: Different types of arrow between logical frameworks. In: auf der Heide, F.M., Monien, B. (eds.) Automata, Languages and Programming (ICALP96, Paderborn, Germany, 8–12 July 1996). Lecture Notes in Computer Science, vol. 1099, pp. 158–169. Springer (1996)
42. Mossakowski, T., Roggenbach, M.: Structured CSP—a process algebra as an institution. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) Recent Trends in Algebraic Development Techniques (Revised Selected Papers of WADT 2006, La Roche en Ardenne, Belgium, 1–3 June 2006). Lecture Notes in Computer Science, vol. 4409, pp. 92–110. Springer (2006)
43. Neves, R., Madeira, A., Martins, M.A., Barbosa, L.S.: Hybridisation at work. In: Heckel, R., Milius, S. (eds.) Algebra and Coalgebra in Computer Science—5th International Conference, CALCO 2013, Warsaw, Poland, 3–6 Sept 2013. Proceedings. Lecture Notes in Computer Science, vol. 8089, pp. 340–345 (2013)
44. Neves, R., Madeira, A., Martins, M.A., Barbosa, L.S.: An institution for alloy and its translation to second-order logic. In: Bouabana-Tebibel, T., Rubin, S.H. (eds.) Integration of Reusable Systems [extended versions of the best papers presented at IEEE International Conference on Information Reuse and Integration and IEEE International Workshop on Formal Methods Integration, San Francisco, CA, USA, Aug 2013]. Advances in Intelligent Systems and Computing, vol. 263, pp. 45–75. Springer (2013)
45. Neves, R., Madeira, A., Martins, M.A., Barbosa, L.S.: Completeness and decidability results for hybrid(ised) logics. In: Braga, C., Martí-Oliet, N. (eds.) Formal Methods: Foundations and Applications - 17th Brazilian Symposium, SBMF 2014, Maceió, AL, Brazil, 29 Sept–1 Oct 2014. Lecture Notes in Computer Science, vol. 8941, pp. 146–161. Springer (2015)
46. Prior, A.N.: Past, Present and Future. Oxford University Press (1967)
47. Passy, S., Tinchev, T.: An essay in combinatory dynamic logic. Inf. Comput. **93**(2), 263–332 (1991)
48. Ciocarlie, H., Szepesia, R.: An overview on software reconfiguration. Theory Appl. Math. Comput. Sci. **1**, 74–79 (2011)
49. Segala, R.: A compositional trace-based semantics for probabilistic automata. In: Lee, I., Smolka, S.A. (eds.) Concurrency Theory (CONCUR'95—Philadelphia, PA, USA, 21–24 Aug 1995). Lecture Notes in Computer Science, vol. 962, pp. 234–248. Springer (1995)

50. Schröder, L., Mossakowski, T.: HasCasl: integrated higher-order specification and program development. Theor. Comput. Sci. **410**(12–13), 1217–1260 (2009)
51. Sokolova, A.: Probabilistic systems coalgebraically: a survey. Theor. Comput. Sci. **412**(38), 5095–5110 (2011)
52. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. Monographs on Theoretical Computer Science, an EATCS Series. Springer (2012)
53. van Bentham, J.: Modal Logic and Classic Logic. Humanities Press (1983)
54. van Benthem, J.: An essay on sabotage and obstruction. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 2605, pp. 268–276. Springer (2005)