Formal Engineering Hybrid Systems:
Semantic Underpinnings

Marius C. Bujorianu and Manuela L. Bujorianu
Formal Methods Group
Faculty of Computer Science - EWI
University of Twente,
Enschede, the Netherlands
email: l.m.bujorianu@cs.utwente.nl

March 13, 2008

Abstract

In this work we investigate some issues in applying formal methods to hybrid system development and develop a categorical framework. We study the themes of stochastic reasoning, heterogeneous formal specification and retrenchment. Hybrid systems raise a rich pallets of aspects that need to be investigated, but never the issue of how the multitude of logics, methodologies and tools can be used altogether. We attack this very difficult issue using categorical logic. When applying formal methods hybrid systems, new (formal methods) mathematics can be created. In this sense, we present new developments in categorical logic, inspired by the control engineering way of treating different aspects of systems. As stochastic reasoning has recently seized its importance in modelling and analysing hybrid systems, we present a uniform categorical formalisation of discrete, continuous and stochastic hybrid systems. A categorical, semantic framework is developed in order to help relating different aspects of hybrid system development.

Keywords: control and stochastic hybrid systems; logic; category theory; coalgebra; Markov processes; viewpoints.
1 Introduction

Hybrid systems mix discrete and continuous evolutions, making difficult to apply formal methods to their development process. This is mainly due to the semantic gap between the discrete and continuous semantics and to the complexity of their interaction. The most common way of applying formal methods to hybrid system development was via discrete or logical abstraction. Once a discrete model is produced (for example a timed automaton), many formal theories and tools become available. An alternative approach arise by formal specification: separate logics are used to specify discrete and continuous evolutions, and logics are combined, more or less loosely.

Some approaches to applying formal methods to continuous control and hybrid systems are using category theory as an abstract bridge between discrete and control systems. Originating from general systems theory [33], [41], [40] this methodology was recently used by Pappas, Tabuada and co-workers [23], [39] to defining bisimulation and model checking for continuous systems by transporting the corresponding concepts from (discrete) labelled transition systems. Van Schuppen and Komenda [29], [30], [38] have used coalgebras for studying control systems (but mostly for the discrete event ones).

A more fundamental approach attempts to invent formal methods tailored for hybrid systems. This method was used by logicians who invented logics for hybrid systems (see Davoren et.al. [15], [16]). The main focus is on expressivity rather on developments methods. Paraphrasing Goguen, there is now an explosion of using such logics. This fact is natural because every logic offers a particular perspective on hybrid systems.

Recent developments in hybrid systems emphasize the important role of probabilistic and stochastic reasoning. For example, two new logics and six papers on stochastic hybrid systems have been presented at the last edition of HSCC conference [1]!

All these developments prompt for some important issues regarding the semantic basis of the formal hybrid systems development:

- **Formal heterogeneity**: a mean is necessary to integrate different formal perspectives on a hybrid system being developed. These perspectives could mean using simultaneously different logics or more than one mathematical technique, like stochastic analysis, optimal control, etc.
- **Inventing new formal methods tailored for hybrid systems.**
The former issue is a very difficult one even in the software engineering of discrete systems. It involves categorical formalisation of specification logics (institutions, parchments, charters [21]) and a very complicated theory of morphisms of such structures (logic translation, Grothendieck institutions [34]). This general approach is difficult to apply to hybrid systems mainly because very little logics (except the modal logic from [31]) for hybrid systems admit categorical formalisation as institutions (notoriously, synchronous languages all fail to be institutions), and the translation theory, already very complicated, would get more complex. We propose a solution inspired from practice of hybrid systems algebraic development, which, based on categories of hybrid systems and functors between them, plays the role of logic translation. This method is based on the implicit assumption that syntax can be somehow encoded in the models. Another well developed example of inventing new mathematics from interaction between control and hybrid systems engineering and formal methods is the theory of stochastic analysis with multiformal time [5].

The later issue is treated in this paper in two ways. One way is by presenting retrenchment, a development method resulted from the observation that refinement rules in formal methods fail in control system cases. The new technique is designed considering specifics of control engineering. Another way results from combining the two issues. The foundations of hybrid system engineering approach to heterogeneity constitute a challenge for formal methods community. We present such foundations in categorical logics and these provide new insights in formal heterogeneity in general.

The paper is structured as follows. In the next section we define a categorical formalisation of (stochastic) evolutions of dynamical systems as actions of a monoid on a category. We show that this construction can be viewed in two different, categorical, ways. This observation forms the basis of a categorical approach to formal heterogeneity presented in the following section. In section 4 the categorical theory of dynamical systems is extended to model stochastic hybrid systems by adding coalgebra (modelling the controller) to a monoid action (modelling the plant). In section 5 the model developed in the previous section to a relational model and we use this model to present the retrenchment, and conclusions are drawn in the final section.

Every section starts with a categorical investigation, and the sum of results of all these investigations gives a categorical framework centred on the concept of fibration. Stochastic hybrid systems are modelled as the Grothendieck construction (i.e. an opfibration), in the categorical formali-
isation of specification logics fibrations relate specifications with signatures, and the semantics of model oriented retrenchment is relational (and various kinds of relations can be defined using fibrations).

The paper makes use of category theory at an advanced level. The excellent monograph [?] covers all categorical background we use. Some elementary knowledge of differential equations, Markov process and formal specification language principles is necessary.

2 Categorical Views on State Evolutions

The unified view of dynamics in state base systems, discrete or continuous, using action of monoids is now a classical and well established in systems theory [33]. In this section we propose an extension of this view on Markov processes. In order to this it is necessary to generalize the actions of a monoid from a set to a category. The resulting construction can be seen in two dual ways, as algebra or a transformation (functor), prompting for a new formalization of specification logics for hybrid systems.

Let \( \textbf{SET} \) be the category of sets and functions and \( \textbf{CAT} \) the category of small categories and functors.

2.1 Discrete and continuous deterministic evolutions

Let \((T, +, o)\) be an arbitrary monoid. An action of \( T \) on a set \( S \) [41], [33], [27] (or a monoid action) consists of a function \( \alpha : T \times S \to S \) such that

\[
\alpha(o, s) = s \quad \text{and} \quad \alpha(t_1 + t_2, s) = \alpha(t_1, \alpha(t_2, s))
\]

Consider a deterministic automaton with alphabet \( A \) and transition function

\[
\delta : X \times A \to X.
\]

A standard construction associates an extended transition function (the behavior)

\[
\delta^* : X \times A^* \to X
\]

given by

\[
\delta^*(x, <>) = x
\]
and
\[ \delta^*(x, a \cdot \sigma) = \delta^*(\delta(x, a), \sigma), \sigma \in A^*. \]

The behavior of the automaton can be characterized as an action \( \alpha \) of the free monoid \( A^* \) on the set of states \( X \) by
\[ \alpha(\sigma, x) = \delta^*(x, \sigma). \]

Examples:
- Modules and vector spaces are monoid actions, where \((T, +, o)\) is the monoid of scalars and \(S\) is the set of vectors.
- The flows (i.e. the unique solutions of differential equations \([41]\)) can be characterized as monoid actions.

## 2.2 Stochastic evolutions

The most studied model of systems with uncertain (stochastic) evolution is the Markov process. The associated theory is very rich and, therefore, we need to characterize these processes by simpler algebraic concepts. The stochastic analysis identifies concepts (like infinitesimal generator, semigroup of operators, resolvent of operators) that characterize in an abstract sense the evolutions of a Markov process. Under standard assumptions, all these concepts are equivalent, in the sense that given one concept then all the others can be constructed from it. In the remainder of this subsection we describe how a semigroup of operators can be associated to a Markov process and what is such semigroup from a categorical point of view. In this paper we make the convention that all semigroups we use are associated to a Markov process.

The state space is denoted by \( S \). The most basic assumption is that one can reason about state change using probabilities. In order to do this we need to equip the state space with a measurable space. In the following, a probability space \((\Omega, \mathcal{F}, P)\) is fixed and all \( S \)-valued random variables are defined on this probability space. \( S \) is a locally compact space separable space endowed with its Borel \( \sigma \)-algebra \( \mathcal{B} \) and that \( B(S) \) is the linear space of bounded measurable functions on \( S \).

The trajectories in the state space are modelled by a family of random variables \((X_t)\) where \( t \) denotes the time. The reasoning about state change is carried out by a family of probabilities \( P^s \) one for each state \( s \in S \). The
construction is similar to the coalgebraic reasoning in the semantics of specification languages: the system behavior is described by given for each state the possible evolutions. For Markov processes, we give for each state \( s \) the probability \( P^s(X_t \in A) \) to reach a given set of state \( A \subset S \) (provided that \( A \) is measurable) starting from \( s \). We remark two ingredients that make the difference from the deterministic case: the evolutions are described from an initial state to a set of final set (nondeterminism) and all we know is a probability to have such trajectories (uncertainty). Together, these ingredients form an ‘abstract state’ and abstract state change can be described in a similar way to that from previous section. This is the sense how a semigroup of operators (which will be defined in the following) is abstracting a Markov process.

Suppose that

\[
\mathbb{M} = (\Omega, \mathcal{F}, \mathcal{F}_t, X_t, P^s)
\]

is a homogeneous Markov process with the state space \((S, \mathcal{B})\) and the parameter time \( t \in T \). In most of the cases \( T = [0, \infty) \). The Wiener probability

\[
P^s : (\Omega, \mathcal{F}) \to [0, 1]
\]

is a probability measure such that \( P^s(X_t \in A) \) is \( \mathcal{B} \)-measurable in \( s \in S \) for each \( t \in T \) and \( A \in \mathcal{B} \) and

\[P^s(X_0 = s) = 1.\]

The mathematical expectation corresponding to \( P^s \) is denoted by \( E^s \). Note that to any Markov process we can associate its transition function

\[
p : T \times S \times \mathcal{B} \to [0, 1]
\]

(a family of transition probability measures indexed by time) which is related with the Wiener probability by \( P^s(X_t \in A) = p(t, s, A) \). The semigroup \((P_t)_{t \in T}\) associated to \( \mathbb{M} \) is given by

\[P_t \varphi(s) = E^s \varphi(X_t),\]

where each \( P_t \) maps \( B(S) \) into itself.

In the following, we characterise the semigroup associated to \( \mathbb{M} \) categorically, as a more general action of a monoid. This characterisation is a natural continuation of categorical formalisation activity of evolutions: a more complicated dynamics correspond to a more complex action of the monoid.
The action of the monoid \((T, +, o)\) on the monoid \((P, o, 1)\) is a map
\[
\alpha : T \times S \to S
\]
such that
\begin{enumerate}
\item[(M1)] \(\alpha(t, 1) = 1;\)
\item[(M2)] \(\alpha(t, s_1 \circ s_2) = \alpha(t, s_1) \circ \alpha(t, s_2)\)
\item[(M3)] \(\alpha(o, s) = s;\)
\item[(M4)] \(\alpha(t_1 + t_2, s) = \alpha(t_1, \alpha(t_2, s))\)
\end{enumerate}

The semidirect product of two monoids \((T, +, 0)\) and \((S, o, 1)\) with action \(\alpha : T \times S \to S\) is a monoid that has carrier \(S \times T\) and the structure defined by
\[
(s_1, t_1)(s_2, t_2) = (s_1 \circ \alpha(t_1, s_2), t_1 + t_2).
\]

**Theorem 1** The \((P_t)_{t \in T}\) associated to a Markov process is the semidirect product of the time monoid \((T, +, o)\) on the monoid \((P, o, I)\) of linear operators on \(B(S)\) with \(I\) the identity operator.

### 2.3 From categories to specification

In the following we present two different views on actions of a monoid on a monoid. One is algebraic (in the sense of algebraic specification of data types), in essence denotational, and the other one is more “operational”. Coalgebras, the dual concept of algebra, is more conveniently presented here, but it will be used in section 4.

A well-established approach (originated from Lambek) to the semantics of data types is to regard them as algebras for endofunctors (called signatures) \(\text{Sig} : \text{C} \rightarrow \text{C}\) on a category \(\text{C}\) with suitable structure.

A \(\text{Sig}-\text{algebra}\) is an object \(A\) together with an arrow
\[
a : \text{Sig}[A] \rightarrow A.
\]

\(A\) is called the carrier and \(a\) the structure. \(\text{Sig}\)-algebras form a category, and an initial algebra is an initial object in this category. Inductive data types correspond to initial algebras.

Dually, a \(\text{Sig}-\text{coalgebra}\) is an object \(A\) together with an arrow
\[
c : A \rightarrow \text{Sig}[A].
\]
$A$ is called the *carrier* and $c$ the *dynamics* of the algebra. Coalgebras form a category and the final object of it (called *terminal coalgebra*) is of most interest to us. Dynamics of discrete dynamical systems can be characterised as final coalgebras [29].

Let $\mathbf{C}$ be a category with finite limits. For an object $T \in |\mathbf{C}|$ define an endofunctor $F_T$ by

$$F_T[S] = T \times S$$

and

$$F_T[f] = id_T \times f,$$

where $f : S \rightarrow U$ and $f : T \times S \rightarrow T \times U$.

A morphism (called also *dynamorphisms* [33]) between the $T$–actions $\alpha : T \times S_1 \rightarrow S_1$ and $\beta : T \times S_2 \rightarrow S_2$ is a map $m : S_1 \rightarrow S_2$ such that

$$\alpha ; f = (id_A \times f); \beta.$$

We denote by

$$\mathbf{ACT}_T$$

the category of $T$–actions and by

$$Act : \mathbf{C} \rightarrow \mathbf{CAT}$$

the functor associating to each object $T$ its category of $T$–actions $\mathbf{ACT}_T$.

Consider the forgetful functor

$$U_T : \mathbf{ACT}_T \rightarrow \mathbf{C},$$

defined (as usual) by $U_T[\alpha] = S$ and $U_T[id_T \times f] = f$. In the following we suppose that, for every object $T$ of $\mathbf{C}$, the functor $U_T$ admits a left adjoint

$$Act_T : \mathbf{C} \rightarrow \mathbf{ACT}_T.$$

**Theorem 2** Any $T$–action is an algebra of the endofunctor $F_T$.

The categories of algebras behave contravariantly when the signature is changed (i.e. when considering a natural transformation between signatures, playing the role of signature morphism). This gives rise to a contravariant functor $\phi$ from the category of signatures to the category of algebras (models). This functor will be formalised in the next section as denotational FDT.
A T-flow on S is a functor \( \alpha : \mathbf{C} \to \mathbf{S} \).

Examples

- When \( \mathbf{T} \) is a monoid category \((\mathbf{T}, +, 0)\) and \( \mathbf{S} \) is the discrete category (i.e. a set \( S \)), then the \( T \)-flow \( \alpha \) is the action of the monoid \( T \) on the set \( S \).
- When \( \mathbf{T} \) is a monoid category \((\mathbf{T}, +, 0)\) and \( \mathbf{S} \) is the monoid category of endomorphisms of a monoid \((S, \circ, 1)\) (i.e. \( \mathbf{S} = \text{End}(S, \circ, 1) \)), then the \( T \)-flow \( \alpha \) is the semidirect product of \( S \) and \( T \).

**Proposition 3** The \((P_t)_{t \in \mathbf{T}}\) associated to a Markov process is the Grothendieck construction of the \( \mathbf{T} \)-flow on \( \mathbf{S} \).

The significance of the results enunciated in Theorem 2 and Proposition 3 consists of the possibility of having dual perspective on system dynamics, whenever they are discrete or continuous, deterministic or stochastic. According to Theorem 2, every system evolution can be interpreted as a data type, i.e. for example natural numbers and stochastic system are treated in the same way. This interpretation is specific to denotational semantics, where everything is interpreted as an object in a mathematical structure. The Proposition 3 follows a dual point of view. Every evolution is seen as a transformation of a mathematical structure (the states) according to some rules. This interpretation is specific to operational mathematical semantics. The duality of these interpretations is captured categorically the covariance/contravariance of some functors (capturing the “logic”). This discussion is formalised in next section.

3 Towards a Syntax Independent Approach to Specification Logics

We have seen in the previous section that system dynamics (the evolutions) can be interpreted in two dual ways: as algebras or as transformations (functors). The categorical modeling also suggests a way to be used to formalize specification logics: there is a dual behavior of functors associating a vocabulary (signature) to a specification. First we need to define a category theoretic formalisation of specification languages. After that, we use the formalisation to construct a functorial semantics that can encode the syntax of the logics, such that systems can be thought in terms of categorical models only.
3.1 Categorical logics for dynamical system specification

The first ingredient we need is the structure of specifications in respect with a development relation, like refinement or retrenchment (see Section 5). The basic property of such relations is that they are orders:

\[ sp1 \sqsubseteq sp2 \]

means \( sp2 \) is an improvement (by refinement or by retrenchment) of \( sp2 \). The existence of joins assures the existence of a least developed common improvement (called unification).

A spec space \( (S, \sqsubseteq) \) is a complete join semilattice. Elements of \( S \) are specifications and \( \sqsubseteq \) models a development relation. A spec space morphism is a map \( m : S_1 \rightarrow S_2 \) such that

\[ m(\sqcup F) = \sqcup m(F) \]

for each nonempty subset \( F \) of \( S \). Spec spaces and their morphisms form a category \( S \).

In practice, a specification is constructed in terms of some finitary operations. We model this aspect by the co-algebraic property. A poset \( (S, \sqsubseteq) \) is called co-algebraic iff for every \( s \in S \),

\[ s^\sqsubseteq = \{ t \sqsubseteq s \mid t \text{is compact} \} \]

is directed and

\[ s = s^\sqsubseteq. \]

One can remark that, inverting the order relation, the most familiar concept of algebraic ordered set is obtained. Coalgebraic spec spaces form a full subcategory \( aS \) of \( S \).

In the following we use the categorical formalisation of logics as hyperdoctrines, originating from the work of Lawvere, Johnstone, Pitts et.al. The advantages and subtleties are explained in the excellent survey [35].

By \( \Pr eOrd \) we denote the category of preorders and sup-preserving functions. The powerset endofunctor is denoted \( \phi \). The application of an functor \( F \) to an argument \( X \) is denoted by \( F[X] \).

Fibrations can be understood as a categorical model of indexing operation (on the objects of a category rather on elements of a set). Formally, a fibration
from the total category $E$ to the base category $B$ is a functor

$$Fib : E \to B$$

for which, for every $e \in |E|$ and $u : i \to Fib[e] \in B$, there is a Cartesian lifting of $u$ [?].

A posetal hyperdoctrine $^1$ is a contravariant functor

$$\wp : B^{op} \to \text{PreOrd}_\vee$$

where $B$ is Cartesian (i.e. it has all finite products). We assume that, for each arrow $f \in B$, the monotone function $\wp[f]$ preserves sups and has a left adjoint which is the existential quantification (defining quantification as adjoints of projections is the essence of categorical logic). We also require the Beck-Chevalley and Frobenius conditions [35]. A nonposetal hyperdoctrine mainly differs by being nonposetal, pseudofunctorial (i.e. categorical diagrams are closed up to isomorphism) and defined on the dual category of vocabularies:

$$\wp : B \to \text{MOD}$$

where MOD is some subcategory of $\text{CAT}$, usually the category of Cartesian closed categories. The most significant logical difference between posetal and nonposetal hyperdoctrines is that in the former case different proofs of the same formula are identified, and in the last case not.

Hyperdoctrines form a special case of fibrations.

A denotational Formal Development Technique (denotational FDT for short)

$$(SS, B, \wp, \theta, Sem)$$

consists of

- A subcategory $SS$ of $aS$
- A fibration $f : SS \to B$
- A posetal hyperfibration

$$\wp : B^{op} \to \text{PreOrd}_\vee$$

- a (semantic) functor

$$Sem : SS \to \text{PreOrd}_\vee,$$

---

$^1$In the sixties, the term doctrine was used to denote every equational theory over category of small categories.
such that $Sem[Sp] \subseteq \varphi[F[Sp]]$.

**Example (the specification language Z)** Let $SS$ be the category of $Z$ schemas, ordered by inclusion, and $B$ the category of records of typed variables and constants. Then $\Sigma : SS \to B$ defined by $\Sigma S = S \lor \text{not} S$ (the *signature* of a schema) is a fibration. The posetal hyperfibration is the powerset endofunctor itself. The logic of $Z$ is first order, and it can be modelled as a hyperdoctrine.

**Example (institutions)** Institutions adopt a different order of presenting its components. The category $B$ is the most basic primitive and its objects are called signatures. The category of specifications $aS$ is just $SET$, i.e. no development relation is considered and only the axioms (called formulas) of a theory (we consider here as being specification) are given. The fibration $F$ is defined as the left adjoint of a functor that associates to a signature a set of formulas. The posetal hyperfibration $\varphi$ associates to a signature a set (or a category) of models (remark that $\text{PreOrd}_\varphi$ is not necessary anymore, so it can be replaced by the larger category $\text{CAT}$). The $Sem$ functor is given by a family of (satisfaction) relations.

Institutions are mentioned in general system theory in [40]. Rather of using them, the authors voice the need to generalise them, and category theory is used in order to get that. Other examples include various modal, temporal logics and the Object Constraint Language (OCL).

An *operational Formal Development Technique* (operational FDT for short)

$$(SS, B,F, \varphi, Sem, MOD)$$

has a similar definition as denotational FDT, except the hyperdoctrine component, which is nonposetal.

Important examples of MMD frames are the (higher order) categorical logics of *labelled transition systems* (LTS) and their many variants. An LTS $P$ is a diagram

$$\Sigma_P \to T_P \xrightarrow{\delta} S_P \leftarrow 1$$

in $B$. The operations $\lambda$, $\delta$ and $\varrho$ assign to each transition, respectively, a label, a source and a target. The constant $\iota$ is the initial state. A morphism $\varphi : P \to Q$ of labelled transition systems is a triple of functions $(\Sigma \varphi, \tau \varphi, S \varphi)$. The latter two form a graph morphism, preserving the initial state and the labelling - in the sense that an $a$-labelled transition is mapped to a $\Sigma \varphi(a)$-labelled one. With these kind of morphisms the labelled transition systems
form a category \( \text{LTS} \). It is fibred by the functor

\[
\Sigma_{\text{LTS}} : \text{LTS} \to \text{B},
\]

which projects each labelled transition system to the corresponding alphabet. \( \Sigma_{\text{LTS}} \) is a regular fibration. There is a regular fibration \( \text{LTS}^r \to \text{B} \), spanned by reachable labelled transition systems, where each state can be reached from the initial state. The inclusion \( \text{LTS}^r \hookrightarrow \text{LTS} \) has a right adjoint, i.e. reachable labelled transition systems span a coreflexive subcategory of labelled transition systems.

Many other important examples of lts, like asynchronous automata and Petri nets [4], form MMD frames.

The covariance of the hyperfibration \( \varphi \) means that in the refinement process one gets less models by eliminating formulas from specification. Adding one more formula to a specification means a new building rule for the models of the system to be developed. In an operational FDT the models of specifications are not specified directly, but rules to build them are described instead. This style is specific to meta-modeling. The most prominent examples of this style are the UML metamodeling techniques, the grammars, the graph (algebra) transformation systems and the structured operational semantics [9].

Hybrid systems specification can be carrying out in both types of development frames. A modal logic for hybrid systems is formalised as an institution in [31]. On the other side, hybrid automata form a class of lts, so they admit a refined operational FDT formalisation. The main differences of these approaches reside in the semantic style. In a denotational FDT, all system executions are available and each evolution can be treated individually, therefore a denotational style is invoked. From a logical point of view, the posetal hyperdoctrine assures that two object having the same denotation are equal, which is quit fair! In an operational FDT, full system evolutions are not available, but rules to construct them are provided. Such style correspond to operational semantics: transition rules are provided. For example, in process algebras as CCS and CSP, the semantics of each equation of a specification involves firing one or more structured operational semantics rules. From a logical point of view, system executions are encoded as proofs (this is the case, for example, for the linear logic specification of dynamical systems). The non posetal hyperdoctrine assures that different evolutions in the state space, having common initial and final points, are not identified. Case stud-
ies involving hybrid system specification using this style are provided in [14], [25].

One major impact of the two different categorical formalisation of logic specification is on system composition. Modularity is a very important issue in system development and structuring principles are dual in different development frames: in COD frames systems are composed using pushouts [10], whilst in MMD frames, pullbacks are used instead [4].

3.2 A functorial approach to logic heterogeneity

A structuring category $St(Sp)$ for a specification $Sp$ can be thought of as a category (with some specific properties), which is the ‘smallest’ such category in which $Sp$ can be modelled soundly.

A denotational FDT is called $C$-structured if there is a category $C$ (called the structuring category) with pushouts and with a faithful functor $C \rightarrow \text{CAT}$ such that :

- There are functors $U : C \rightarrow SS$ and $F : SS \rightarrow C$ with $F$ left adjoint to $U$.
- The functor $\text{Sem} : SS^{op} \rightarrow \text{Pr eOrd}_\forall$ is naturally isomorphic to the functor $SS^{op} \rightarrow C^{op} \rightarrow \text{Pr eOrd}_\forall$

where the functor $\setminus / C$ sends an object $c \in |C|$ to the slice category $c / C$.

**Examples**

- the equational logic, where $C$ is the category of categories with product;
- the ADT logic, where $C$ is the category of elementary toposes [9];
- the logic of the typed lambda calculus, where $C$ is the category of Cartesian closed categories;
- the logic of the polymorphic lambda calculus, where $C$ is the category of relatively Cartesian closed categories;
- the Girard’s linear logic, where $C$ is Seely’s category of linear categories.

We say that two specifications are equivalent (relation denoted by using $\equiv$ symbol) if their structuring categories are equivalent.

Every structuring category gives rise to a specification logic, and this process is mutually inverse to that of constructing a structuring category for a given specification logic.

In this paper, we make this correspondence more precise for equational logic, which is at the heart of the most specification languages.
**Proposition 4** For any category $C$ with finite products, we can associate a particular equational specification $Sp[C] = (\text{Sig}, \text{Eqns})$. Moreover, there is a canonical model of $Sp[C]$ in $C$.

The equational specification $Sp[C]$ allows one to reason about the category $C$ thought of as the category of sets and functions.

**Proposition 5** For every equational specification $Sp$ we have $Sp \cong Sp[St[Sp]]$

This statement is extremely useful because it establishes that the categories with finite products provide a representation of the notion of equational specification, which is syntax independent.

The benefits of the type theoretic approach can be used to define specification language translation. For example, the translation

$$Tra : Sp_1 \rightarrow Sp_2$$

of equational specifications can be given as a finite product preserving functor

$$Tr : St[Sp_1] \rightarrow St[Sp_2].$$

Using functor equivalence, this amount gives a functorial model

$$T : Sp_1 \rightarrow St[Sp_2].$$

The methodology of working with hybrid systems modelled heterogeneously using different categorical models (or logics) is illustrated in Figure 1.

## 4 Categorical Models of (Stochastic) Hybrid Systems

We extend in the following the categorical treatment of Markov processes to a model of stochastic hybrid systems. We add coalgebra to the previous model to interpret the switches in the Markov process. In this way we can use object orientation, bisimulation and terminal models (meaning a kind of minimal realization in the sense of system theory) from the coalgebraic based formal methods [13], using a convenient concept of morphism.
Algebras and coalgebras can be understood in terms of relations [10], [27]. One way to do that is to represent the operations of algebra by their graph (in this way algebraic calculus is reduced to set calculus) and coalgebras can be defined as relations between states and transition labels. This point of view is fully developed in [10]. By proceeding in this way, we get a common categorical model [9] with the state and operations style (SO for short) used in model oriented specification and development [17] (associated with FDTs like Z and B). We can formulate a correspondence between hybrid systems and SO style as:

\[ \text{plant evolutions} \leftrightarrow \text{SOs states} \]

and

\[ \text{controller} \leftrightarrow \text{SOs operations}. \]

This interpretation can be extended to a relational framework of hybrid system design (Fig. 2).

### 4.1 Combined algebra and coalgebra models

Combining algebra and coalgebra is a modern tool for algebraic modelling of sophisticated dynamics. Continuous evolutions are treated in this approach as data types.

We proposed the coalgebra and monoid action paradigm as an abstract model of stochastic hybrid systems. In essence, the model consists of
• a small category of states \( S \) (we call the objects of \( S \) the “carrier set” \( S \))
• an initial state \( s^0 \in S \)
• a \( T \cdot \text{flow on } S \), where \( T \) is the monoid of discrete or real time
• a \( F \) coalgebra \( S \rightarrow F[S] \).

We consider a polynomial functor

\[
F[S] = S^T \times A \times S^B,
\]

where the functions \( at : S \rightarrow A \) are called attributes (observers) and the functions

\[
\text{meth} : S \times B \rightarrow S
\]
denote methods. We borrow from [27] the notation

\[
s.meth@t
\]

having the meaning “in state \( s \) let the state evolve for \( t \) units of time (according to the monoid action), and then apply the (coalgebraic) method \( \text{meth} \)”.

All in all, the polynomial functor coalgebras are used to define an object oriented language for controlling the plant (modelled as a monoid action). A morphism between hybrid systems

\[
(S_1, s_1^0, \alpha_1, at_1, \text{meth}_1)
\]

and

\[
(S_2, s_2^0, \alpha_2, at_2, \text{meth}_2)
\]

is defined by a map \( h : S_1 \rightarrow S_2 \) such that

\[
h(s_1^0) = h(s_2^0), f(s).at_2@t = s.at_1@t
\]

and

\[
f(s).\text{meth}_2(b)@t = f(s.\text{meth}_2(b)@t).
\]

There is a subtle aspect regarding the concept of morphism. It does not involve a morphism of monoid actions! Indeed, the internal time steps (i.e. the continuous evolutions) are suitable hidden in the definition of morphism: the attributes (i.e. the external observations) and the methods (i.e. the discrete control) determine a morphism. The main advantage of this approach is that coalgebraic bisimulation, terminality and refinement for hybrid systems are coming for free (from the coalgebraic world).
Jacobs [27] has developed an object oriented specification language for
deterministic hybrid systems having coalgebraic semantics and using action
of a monoid on a set rather on a category. He has also worked out detailed
cases studies where bisimulation and terminality (thought as a “minimal”
realisation) are studied. His definition of morphism is identical equivalent
with the one used here. One could expect different definitions, i.e. stochastic
hybrid system preserve the stochastic structure. This fact is explained by
the above remark: continuous evolutions are hidden!

4.2 Relational models

Hermida and Jacobs [26], in a lengthy paper showed how algebraic and coal-
gebraic semantics (and their associated proof techniques) can be formulated
in categories of relations. This suggests that hybrid systems can be modelled
in a richer, relational framework [19].

A category of homogenous relations $\text{REL}_E$ over a category with pullbacks
$E$ (i.e. between sets having the same type, like the objects of a category $E$)
can be obtained by considering the category of spans factorized by epics [10].

For any relation $\rho \in \text{REL}_E$ we define its $\textsf{precondition}$ as

\[ \text{pre}_\rho = id \cap \rho^{-1} \]

and its $\textsf{image}$ by

\[ \text{im}_\rho = id \cap \rho^{-1} \rho. \]

Obviously, $\text{pre}_\rho$ and $\text{im}_\rho$ can be seen as subobjects in $E$ (for example $\text{pre}_\rho \subseteq id$ and can be identified with the set

\[ \{ c | \{ c \} \in |E| \land (c, c) \in id \cap \rho^{-1} \} \].

A $\textsf{postcondition}$ $\text{post}_\rho$ of $\rho$ is a relation (predicate) between $\text{pre}_\rho$ and $\text{im}_\rho$
such that

\[ \rho = \text{pre}_\rho \land \text{post}_\rho \]

(as predicates).

An $\textsf{abstract data type}$ (ADT for short) is a triple

\[ (St, \text{Init}, \{ Op \}) , \]

where $St$ denotes the state space, $\text{Init}$ denotes the initialization of described
state machine and the set of operations $\{ Op \}$ specify all possible transitions.
A categorical semantics of ADTs [9] considers a category of models $E$ for the state specification $St$, a global element of $E$ for $Init$ and a homogenous relation of $REL_E$ for every operation $Op$. Operation composition is interpreted as relational composition.

The above interpretation is simplified by the assumption that operations do not input and output. The interface of an operation is introduced by splitting the precondition of the operation into a product between states and inputs, and similarly for the outputs (the postcondition become a product this time). But, the semantics of operations fail to be homogenous relations then! Instead, heterogeneous relations should be defined and used instead. Binary relation are predicates whose type annotation corresponds to a product of contexts. If the base category has binary products, the (categorical) logic of relations can be obtained from (categorical) predicate logic considering only those predicates which are relations. Categorically, the category of heterogeneous binary relations $HREL_E$ and homogenous binary relations $REL_E$ arise from $E$ by pullback or change of base in a regular fibration [?]

$Fib$:

\[
\begin{array}{c}
REL_E \xrightarrow{\Delta} HREL_E \rightarrow E \\
\downarrow \quad \quad \downarrow Fib \\
B \xrightarrow{\Delta} A \rightarrow (A,A) \quad \quad B \times B \rightarrow B
\end{array}
\]

Abstract data types form the semantics of classes in Object Z [17] and they are the specification units of B language [?] (where they are called labelled transition systems) and relational frameworks. ADTs have been largely used for control specification: the London’s airport system databases specification is reported in [24] using VDM, a chemical control system specification using B is reported in [6], a radiation therapy machine is specified using Z in [28] and in [32] ADTs are extended with real time and control system inspired concepts. A syntax free approach, using only the notations of relational ADTs, is used in to specify the requirements of USA military fault tolerant flight control systems.

Plant states are modelled as objects in a category and control as a span category. Different spans can be defined as generalised relations. Coalgebraic approach can be recovered using relational coalgebras [7].

The hierarchical (relational) development methodology of hybrid systems is depicted in Fig. 2.
Advantages:
- A sound basis for refinement;
- Wide applicability: most successful languages for hybrid systems are model oriented;
- The construction can be iterated: coalgebras over a topos form a topos. States in model oriented specification are objects of a topos. One can define control over a controlled system: hierarchical approach, mobility, etc.

5 Conclusions and Future Work

This paper proposes new semantic developments in formal methods motivated by hybrid and control systems. A generic approach to formal heterogeneity using category theory is introduced. Functorial semantics is used as a mean of relating logics in a syntax independent way. The concept of
refinement is generalised to retrenchment, and the relevance of this concept to control engineers is explained. A categorical framework for investigating semantic issues related formally engineered methods of developing control and hybrid systems has emerged. The framework is introduced gradually, starting from the, now standard, formalization of (discrete and continuous) dynamical systems as monoid actions. Extending this concept for stochastic processes prompts for a new concept, the flow of a functor, concept that can be characterized as a data type (in algebraic specification tradition, i.e. as an algebra) or as a transformation (in a category of “abstract states”). These interpretation amounts to a new formalization of logics used in the specification of hybrid systems and a new approach to formal heterogeneity, specific to engineers way of reasoning with models. The flow of a functor is combined with a coalgebra to build a model of stochastic hybrid systems. This approach can be generalized even further, to a relational system, which generalizes model oriented development methods. In this setting we present retrenchment, a development method by “refinement”, inspired by control engineering.

The approach we have embarked on is foundational. There are many formal theories and case studies developed by different authors (of this paper and other), and a semantic framework to relate them is necessary. The number of new models, languages, methodologies and tools for hybrid systems is growing very quickly, but no attempt to use many of them altogether and to discuss the semantic issues related to this has been attempted till now.

This paper deliberately does not provide any case study. Instead, references to the main directions in formal development of hybrid systems are provided, and the interest reader can find many detailed case studies in the references provided. In some cases, because of space limitation, we have considered only one the most representative papers of one research direction, so the actual relevant case studies are much richer. We have tried to cite only those approaches that can be semantically embedded in the algebraic framework proposed; there exist formal approaches to specification and refinement of hybrid systems whose semantics are not considered in this paper. In a future work we will propose a case study in air traffic management involving multiple use of specification logics and categorical models.

A situation not covered by the stochastic hybrid system model presented in section 4.1 is when the controller (modelled as a coalgebra) needs to have direct access to the real state space of the Markov process, because it can use only an abstraction of it (via the monoid of operators associated to the
Markov process). In some models [11], transitions take place not conditioned by probabilities, so the real state space need to be accessible. This situation can be remedied using a different coupling mechanism between algebra and coalgebra, as described in [13]. Not a common carrier set is involved, but two state categories related by functors with “nice” properties. Such functors can be identified in the framework we have presented, but full details will be worked out in a future paper. A related issue is the concept of morphism. In the model cited before, the continuous (deterministic and stochastic) evolutions play a very important role, thus their abstraction will produce little significance results. Defining morphisms that preserve some kind of stochastic properties is a very difficult issue.

A very important issue in system development is constituted by the viewpoints. This is a methodology that gives the developers the freedom to work concurrently in the development of the same system, using their own background and favourite specification logics and tools. Viewpoints were coupled with a model theoretic approach (developed in this paper as structuring categories) in a paradigm called correspondence carrying software [9]. Viewpoints need to be integrated to get the full system description. In particular, we need to eliminate redundancy in the integration, as it is the main source of inconsistencies across viewpoints. One way of getting non-redundancy is by using refinement. The least developed common refinement of two viewpoints is called unification. The process of constructing unification finds a natural environment in the algebraic framework proposed in this paper. The categorical duality of development frames means dual construction of unification: in COD frames unification is done by pushout, whilst in MMD frames pullbacks [4] are used. In the relational setting [9], unification of states is by pushout and operation unification by pullback. Unfortunately, in many MMD frames (for example when considering various refinement relations from process algebra, other than trace inclusion) unification can not be constructed, but other integrations are available. Replacing refinement by retrenchment in the definition of unification gives an adequate sense of developing control and hybrid systems using viewpoints.

References


[10] M.C. Bujorianu. *Integration of Specification Languages Using Viewpoints*. In [7].


