Computer Science Journal of Moldova, vol.**23**, no.2(68), 2015

Structuring of Specification Modules (extended)*

Răzvan Diaconescu

Abstract

This paper has two goals. One goal is to provide a brief introduction to the concept of modularisation in the context of formal specifications. The other goal is to survey some recent developments in this area, including parameter instantiation with sharing and module systems for behavioural specifications.

1 Composition of Specification Modules

1.1 Formal methods in software engineering

The field of formal methods for software and hardware systems is without alternative in safety-critical or security areas where one cannot take the risk of failure. Moreover formal methods are crucial in ensuring a smooth development of large software systems as well as their evolution and maintainability. In general one may distinguish between two classes of formal methods:

1. model checking; and
2. formal specification.

While the former has gained certain prominence, being especially succesfull in discovering errors in the system development process, only the latter may fully guarantee correctness.

* This work has been supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0439.

1.2 Formal specification

In simple terms, formal specification activity can be described at two levels:

- At *specification* level, the system is specified axiomatically in a formal logical system (not necessarily a conventional one).
- At *verification* level, the properties of the system are derived as theorems, getting a fully formal proof.

Perhaps the most prominent classes of formal specifications are those that are based upon algebraic principles (in a wide sense of the term). While traditionally *algebraic specification* [3] is based upon a general form of algebra (such as the equational logic of many-sorted algebra), modern algebraic specification employs a wide variety of logical systems of higher levels of sophistication. Modern algebraic specification systems include CASL [2], CafeOBJ [20], Maude [10], Specware [29], etc. Heterogeneous environments [20, 32] constitute a recent integrating trend in logic-based formal specification that provides a very flexible approach when choosing the appropriate language and formalism. The theoretical infrastructure developed over many decades of research in formal specification has been exported also to other areas of computing science such as ontologies (e.g. [30]) or declarative programming.

In some cases there is a rather strong integration between the specification and the verification levels (like in OBJ, CafeOBJ, Maude, etc.). These are systems that in some sense are most faithful to the original algebraic specification culture that has equational logic at its core. The algebraic features of those systems smoothen up significantly the verification process by integrating techniques with good computational properties such as rewriting. In a way such specification languages, also called *executable* languages, may function quite well as very high level (declarative) programming languages. However equational logic poses some limitations in the specification power. In other cases (such as CASL, Specware, etc.) there is not a direct integration between the specification and the verification levels, allowing for more sophisticated logics meaning significant gains in specification power. In these cases the formal verification methodologies may involve external automatic provers (SPASS [41], etc.) and/or proof assistants (Coq, Isabelle, etc.). The difference between these two lies at the level of human involvement, while in the former situations this is minimal, in the case of the proof assistants the formal proofs are human guided.

An important distinctive feature of formal/algebraic specifications is its rock solid mathematical foundations based upon the principle of specification languages being rigorously based upon underlying formal logical systems such that each language constructs reflect rigorously as a mathematical concept in the underlying logic. Modern specification theory is based upon Goguen and Burstall's theory of *institutions* [24, 14] which is a general *category theoretic* [31] approach to logic. This makes the general theory independent of the actual choice of a logical system, which means great uniformity and applicability to a wide variety of specification languages.

1.3 Modular specifications

Modularisation is the only way to cope with the complexity of formal specifications of large software systems. It greatly enhances readability and reusability of specification code. Moreover structuring techniques also reduce the complexity of the formal verification process.

The work on specification modules has a long history that starts with the specification language Clear developed by Goguen and Burstall [9]. That laid general founding principles for module systems of formal specifications that have been realised by a multitude of subsequent languages and systems, notably including programming languages such as ADA or ML.

The main structuring constructs include

- Module imports (various kinds);
- Module sum/aggregation;
- Renamings;
- Information hiding;
- Parameterised (generic) modules and parameters instantiation by *views*;
- Complex module expressions;

The current modularisation theory and methodologies include many developments that have been fuelled by the actual practice in formal specification and by the design of new modern languages and systems supporting this activity. Moreover the modularisation technology has been exported from the area of formal specifications to other computer science areas, e.g. ontologies (the OMG standard *OntoIOp*).

Two quite major ideas have shaped the current approaches to specification modules.

1.3.1 Institution theory

The first idea is to abstract away from the details of the logic underlying the actual specification language. The module composition techniques are largely independent of the logical formalism employed by the respective specification language; in fact this important observation constituted the main idea behind the language Clear and has triggered the conception of institution theory [24] as an abstract framework for modularisation. This very general mathematical study of formal logical systems, with emphasis on semantics, is based upon a mathematical definition for the informal notion of logical system, called *institution*. This definition accommodates not only well established logical systems but also very unconventional ones and moreover it has served and it may serve as a template for defining new ones. Institution theory approaches logic from a relativistic, non-substantialist perspective, quite different from the common reading of logic. This is not opposed to the established logic tradition, since it rather includes it from a higher abstraction level. However the real difference is made at the level of methodology, top-down (in the case of institution theory) versus bottom-up (in the case of conventional logic tradition). Institution theory has had a strong impact in computer science and logic for over more than three decades (see [16]), and it continues to attract an ever growing interest in institution theory by computer scientists and (more recently) logicians.

Let us very briefly recall here the main concept of institution theory.

Definition 1 (Institutions) *An* institution *is a tuple* $(Sign, Sen, Mod, (\models_\Sigma)_{\Sigma \in |Sign|})$ *that consists of*

- *a category Sign whose objects are called* signatures*,*
- *a functor* $Sen\colon Sign \to \mathbb{S}\mathrm{et}$ *(to the category of sets) giving for each signature a set whose elements are called* sentences *over that signature,*
- *a (contravariant) functor* $Mod\colon (Sign)^{op} \to \mathbb{CAT}$ *(to the 'category' of categories), giving for each signature* Σ *a category whose objects are called* Σ-models*, and whose arrows are called* Σ-(model) homomorphisms*, and*
- *a relation* $\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$ *for each* $\Sigma \in |Sign|$*, called the* satisfaction relation*,*

such that for each morphism $\varphi\colon \Sigma \to \Sigma' \in Sign$*, the* Satisfaction Condition

$$M' \models_{\Sigma'} Sen(\varphi)(\rho) \text{ if and only if } Mod(\varphi)(M') \models_\Sigma \rho \qquad (1)$$

holds for each $M' \in |Mod(\Sigma')|$ *and* $\rho \in Sen(\Sigma)$*.*

The literature (e.g. [14, 39]) shows myriads of logical systems from computing or from mathematical logic captured as institutions. In fact, an informal thesis underlying institution theory is that any 'logic' may be captured by the above definition. While this should be taken with a grain of salt, it certainly applies to any logical system based on satisfaction between sentences and models of any kind.

1.3.2 Core specification building operators

A second important idea envisaged in many works (e.g. [38, 4, 39], etc.) is to have a core set of specification building operators, with a clearly defined semantics, that can be used to define composition operators in actual module systems. The most prominent such set of specification building operators has been introduced in [38]; we recall them below in the more modern form of [8].

Given any institution $(\mathit{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ with designated classes of signature morphisms $\mathcal{T}$ and $\mathcal{D}$ the class of the $(\mathcal{T}, \mathcal{D})$-*structured specifications* [8] is the least class such that

- it contains all finite *presentations*, i.e. pairs (Σ, E) with Σ signature and E finite set of Σ-sentences; we also define
  - $\Phi(\Sigma, E) = \Sigma$ and
  - $\mathit{Mod}(\Sigma, E) = \{M \in |\mathit{Mod}(\Sigma)| \mid M \models E\}$,
- if SP_1 and SP_2 are structured specifications such that $\Phi(\mathrm{SP}_1) = \Phi(\mathrm{SP}_2)$, then $\mathrm{SP}_1 \cup \mathrm{SP}_2$ is also a structured specification and we define
  - $\Phi(\mathrm{SP}_1 \cup \mathrm{SP}_2) = \Phi(\mathrm{SP}_i)$ and
  - $|\mathit{Mod}(\mathrm{SP}_1 \cup \mathrm{SP}_2)| = |\mathit{Mod}(\mathrm{SP}_1) \cap \mathit{Mod}(\mathrm{SP}_2)|$,
- if SP is a structured specification and $(\varphi \colon \Phi(\mathrm{SP}) \to \Sigma') \in \mathcal{T}$, then $\mathrm{SP} \star \varphi$ is structured specification and
  - $\Phi(\mathrm{SP} \star \varphi) = \Sigma'$ and
  - $|\mathit{Mod}(\mathrm{SP} \star \varphi)| = \{M' \mid \mathit{Mod}(\varphi)(M') \in \mathit{Mod}(\mathrm{SP})\}$, and
- if SP' is a structured specification and $(\varphi \colon \Sigma \to \Phi(\mathrm{SP}')) \in \mathcal{D}$, then $\varphi \Box \mathrm{SP}'$ is structured specification and
  - $\Phi(\varphi \Box \mathrm{SP}') = \Sigma$ and
  - $|\mathit{Mod}(\varphi \Box \mathrm{SP}')| = \{M' {\restriction_\varphi} \mid M' \in \mathit{Mod}(\mathrm{SP})\}$.

Given a structured specification SP as above $\Phi(\mathrm{SP})$ is its *signature* $\Phi(\mathrm{SP})$ and $\mathit{Mod}(\mathrm{SP})$ is its *class of models.* The signature and the class of models of a specification SP represents its semantics.

When the actual specification language provides tight semantics capabilities, then an initial or final semantics operator is also typically included. However occasionally (see [18]) these specification building operators do not suffice, therefore other operators have also to be considered.

1.3.3 Inclusion systems

At the end of this section we would like to briefly present a category theoretic device that plays an important technical role in the theory of specification modules.

Inclusion systems were introduced in [23] with the aim of supporting an abstract general study of structuring specifications and programming modules that is independent of any underlying logic. While computer science is usually a heavy consumer of mathematics, inclusion systems can be seen as opposite, i.e. a rare contribution of computer science to pure mathematics. Inclusion systems have been used in a series of general module algebra studies such as [23, 27, 14], and also in institutional model theory studies [14, 36, 13, 1, 14]. Inclusion systems capture categorically the concept of set-theoretic inclusion in a way reminiscent of the manner in which the rather notorious concept of factorisation system [7] captures categorically the set-theoretic injections; however, in many applications the former are more convenient than the latter. The definition below can be found in the recent literature on inclusion systems (see, e.g. [14]), and differs slightly from the original one of [23].

Definition 2 (Inclusion system) *An* inclusion system *for a category $\mathbb{C}$ is a structure $\langle \mathcal{I}, \mathcal{E} \rangle$ such that $\mathcal{I}$ and $\mathcal{E}$ are broad subcategories of $\mathbb{C}$ satisfying the following two properties:*

- *$\mathcal{I}$ is a partial order (with the ordering relation denoted by $\subseteq$), and*
- *every arrow f in $\mathbb{C}$ can be factored uniquely as $f = e_f; i_f$, with $e_f \in \mathcal{E}$ and $i_f \in \mathcal{I}$.*

In [12] it was shown that the category $\mathcal{I}$ of abstract inclusions determines the category $\mathcal{E}$ of abstract surjections. In this sense, [12] gives an explicit equivalent definition of inclusion systems that relies only on the category $\mathcal{I}$ of abstract inclusions.

The standard example of inclusion system is that from $\mathbb{S}$et, with set theoretic inclusions in the role of abstract inclusions and surjective functions in the role of abstract surjections. The literature contains many other examples of inclusion systems for the categories of signatures and for the categories of models of various institutions from logic or specification theory.

2 Recent Developments

In this section we survey some recent developments in the theory of modular formal specifications. Three topics are considered: general theory, genericity and sharing, and module systems for behavioural specifications.

2.1 General theory

Pivotal developments in the institutional theory of structured specifications (such as [38, 23] etc.) provide an abstract treatment of the underlying logic as an abstract institution but consider fixed sets of concrete structuring operators. To overcome this limitation, but also to achieve unification between the Goguen-Burstall [24, 23] and the Sannella-Tarlecki [38, 39] approaches to the semantics of structured specifications, the recent paper [15] introduces a second level of institution-independence by treating the class of the structured specifications together with their model theory as an abstract institution. The relationship between the level of structured specifications and the level of the underlying logic is axiomatized by a special kind of institution morphism. The following definition recalls from [15] the main concept of this theory.

Definition 3 (Structured institution) *An institution* $\mathcal{I}' = (\mathit{Sign}', \mathit{Sen}', \mathit{Mod}', \models')$ *is said to be* structured *over a base institution* $\mathcal{I} = (\mathit{Sign}, \mathit{Sen}, \mathit{Mod}, \models)$ *through a* structuring functor Φ*, or* $(\Phi, \mathcal{I})$-structured*, when*

- Φ *is a functor* $\mathit{Sign}' \to \mathit{Sign}$*,*
- *for every* $\mathcal{I}'$*-signature* Σ'*,* $\mathit{Sen}(\Phi(\Sigma')) = \mathit{Sen}'(\Sigma')$*, and similarly, for every* $\mathcal{I}'$*-signature morphism* φ'*,* $\mathit{Sen}(\Phi(\varphi')) = \mathit{Sen}'(\varphi')$*,*

- *for every $\mathcal{I}'$-signature Σ', $Mod'(\Sigma')$ is a full subcategory of $Mod(\Phi(\Sigma'))$ such that the diagram below commutes for every $\mathcal{I}'$-signature morphism $\varphi\colon \Sigma'_1 \to \Sigma'_2$, and*

$$\begin{tikzcd}
Mod'(\Sigma'_1) \arrow[r, "\subseteq"] & Mod(\Phi(\Sigma'_1)) \\
Mod'(\Sigma'_2) \arrow[r, "\subseteq"'] \arrow[u, "Mod'(\varphi)"] & Mod(\Phi(\Sigma'_2)) \arrow[u, "Mod(\Phi(\varphi))"']
\end{tikzcd}$$

- *for every $\mathcal{I}'$-signature Σ', Σ'-model M' and Σ'-sentence ρ,*

$$M' \models'_{\Sigma'} \rho \quad \textit{if and only if} \quad M' \models_{\Phi(\Sigma')} \rho.$$

Several examples of structured institutions are presented in some detail in [15]. One of the most important ones is that of the structured specifications of Sect. 1.3.2. In that example, *Sign'* is the category of the structured specifications, *Mod'*(SP) is *Mod*(SP) as defined in Sect. 1.3.2 and the satisfaction relation of $\mathcal{I}'$ is inherited from $\mathcal{I}$ in the obvious way.

Recent papers such as [15, 40, 11, 19, 17] develop elements of modularisation theory in this abstract framework.

2.2 Genericity and Sharing

Generic or parameterised modules represent one of the most important module composition techniques because they can be (re)used in various ways by appropriate instantiations of their parameters. In simple terms, a parameterised module (denoted SP(P)) can be regarded as a module import $P \to$ SP, with P being its *parameter* and SP being its *body*. Instantiation of parameterised modules is performed through interpretations of their parameters. In the specification literature they are usually called *views* and they are syntactic mappings $v\colon P \to \mathrm{SP}_1$ that satisfy two conditions:

1. they match consistently the signature of the parameter P to the signature of the value SP_1 (which is also a specification module);

technically this amounts to the fact that v is a *signature morphism*; and

2. any implementation[1] of SP_1 has to be a an implementation of the parameter P via the interpretation of the syntactic entities given by v.

Given a parameterised module $\mathrm{SP}(P)$ and a view $v\colon P \to \mathrm{SP}_1$ the *instance* $\mathrm{SP}(P \Leftarrow v)$ is commonly defined by the using the *pushout* technique (cf. [9, 39], etc.) from category theory [31], which informally can be explained as a 'user-defined' form of union. This is a two-steps process. First consider the underlying signatures of the specifications and compute a pushout in the category of signatures:

$$\begin{tikzcd}
\Phi(P) \arrow[r, "\subseteq"] \arrow[d, "\Phi(v)"'] & \Phi(\mathrm{SP}) \arrow[d, "v'"] \\
\Phi(\mathrm{SP}_1) \arrow[r, "i'"'] & \Sigma'
\end{tikzcd}$$

At the second stage we built the actual instance

$$\begin{tikzcd}
P \arrow[r] \arrow[d, "v"'] & \mathrm{SP} \arrow[d, "v'"] \\
\mathrm{SP}_1 \arrow[r, "i'"'] & \mathrm{SP}(P \Leftarrow v)
\end{tikzcd} \tag{2}$$

by defining

$$\mathrm{SP}(P \Leftarrow v) = (\mathrm{SP}_1 \star i') \cup (SP \star v').$$

This requires that, minimally, there are unions and translations available as building operators.

While this is the traditional approach to parameter instantiation which is widely employed by actual specification formalisms (e.g. [9, 2, 20], etc.) and also by programming languages such as ML and

[1] Mathematically speaking, 'implementations' are treated as models or interpretations in the underlying logic.

other languages influenced by it, it still raises several technical issues of important methodological significance and that can be summarised as follows:

1. Since the pushout construction is unique only up to isomorphic renaming, actual implementations of module systems involving parameters provide ad-hoc constructions for the results of parameter instantiations. But how can we ensure in general that there exists an instantiation such that $\mathrm{SP}_1 \to \mathrm{SP}(P \Leftarrow v)$ behaves like an import, and moreover would this be uniquely defined?
2. In order to avoid technical complications the actual specification systems commonly dismiss the sharing between the body (SP) and the instance (SP_1), a situation that in practice constitutes a real restriction, as for example it may lead to duplication of the same data.
3. In the case of multiple parameters, for example $\mathrm{SP}(P_1, P_2)$, we can instantiate them sequentially (first $(\mathrm{SP}(P_1 \Leftarrow v_1)(P_2)$ and next $\mathrm{SP}(P_1 \Leftarrow v_1)(P_2 \Leftarrow v_2)$, or the other way around) or in parallel by regarding SP as parameterised by a single parameter, $\mathrm{SP}(P_1 + P_2 \Leftarrow v_1 + v_2)$. Are the two sequential instantiations and the parallel one equivalent methods in the sense of yielding isomorphic results?

The main technicalities involved in the answer to these questions include both a reshape of the definition of parameter instantiation and a general property of the signatures of the specification language formulated. In brief the traditional pushout square (2) has to be redefined as

$$\begin{tikzcd}
P \cup \mathrm{SP}_1 \arrow[r, "\subseteq"] \arrow[d, "v \cup 1_{\mathrm{SP}_1}"'] & \mathrm{SP} \cup \mathrm{SP}_1 \arrow[d, "v'"] \\
\mathrm{SP}_1 \arrow[r, "i"'] & \mathrm{SP}(P \Leftarrow v)
\end{tikzcd} \tag{3}$$

(where $v \cup 1_{\mathrm{SP}_1}$ implies that v is identity on the part shared between P and SP_1)

and the signatures have to enjoy the following general property: for any signature morphism $\varphi\colon \Sigma \to \Sigma_1$ and any inclusion $\Sigma \subseteq \Sigma'$

$$\begin{tikzcd}
 & \Sigma\cap\Omega \arrow[rr, "\subseteq"] \arrow[d, "\subseteq"] \arrow[dd, bend right=60, "\subseteq"'] & & \Sigma'\cap\Omega \arrow[d, "\subseteq"'] \arrow[dd, bend left=60, "\subseteq"] & \\
\textcircled{1} & \Sigma \arrow[rr, "\subseteq"] \arrow[d, "\varphi"'] \arrow[drr, phantom, "\textcircled{3}"] & & \Sigma' \arrow[d, "\varphi'"] & \textcircled{2} \\
 & \Sigma_1 \arrow[rr, "\subseteq"'] & & \Sigma'_1 &
\end{tikzcd}$$

there exists a pushout ③ such that for any signature Ω if ① holds then ② holds too.

In the works [18, 15, 40] the concepts of inclusion ($\subseteq$), union ($\cup$), intersection ($\cap$), disjointness, etc. are treated abstractly within the framework of 'inclusion systems' as presented in Sect 1.3.3. In this way the solution proposed is general and can be applied to almost all existing specification formalisms (with the notable exception of behavioural specifications discussed below). The required property above holds naturally for all specification formalisms of interest, often in a stronger form than actually required (see [18, 40]). The generality of this solution implies also that it can be employed also by new specification formalisms to be defined in the future.

2.3 Module systems for behavioural specifications

Modern algebraic specification theory and practice has extended the traditional many-sorted algebra-based specification to several new paradigms. One of the most promising is behavioural specification, which originates from the work of Horst Reichel [33, 34] and can be found in the literature under names such as hidden algebra [25, 26], observational logic [5, 28], coherent hidden algebra [21] and hidden logic [35]. Behavioural specification characterises how objects (and systems) *behave*, not how they are implemented. This new form of abstraction can be very powerful for the specification and verification of software systems since it naturally embeds other useful paradigms such as concurrency, object-orientation, constraints, nondeterminism, etc. (see [26] for details). In the tradition of algebraic specification, the behavioural abstraction is achieved by using specification with hidden sorts and a behavioural concept of satisfaction based on the idea of indistinguishability of states that are observationally the same, which also generalizes process algebra and transition systems (see [26]). An important effort has been undertaken to develop languages and systems supporting the behavioural extension of conventional or less conventional algebraic specification techniques; these include CafeOBJ [20, 22], CIRC [37] and BOBJ [35]. In other situations, behavioural specification, although not directly realized at the level of the language definition, is employed as a mere methodological device [6]. In all cases there is the unavoidable need of a structuring mechanism for behavioural specifications.

However the modularisation of behavioural specifications poses specific challenges with respect to the standard modularisation techniques. Some important properties that in general are taken for granted do not hold in the case of behavioural specifications, for example the basic operation of union (aggregation) of behavioural specifications is only partial. The root cause of these problems lies in the 'encapsulation condition' on the signature morphisms, which prohibits new behavioural operations on old hidden sorts (i.e. that correspond to sorts of the source signature). Therefore the basic compositionality properties of behavioural specifications hold in a partial rather than total algebra style form. For example (see [19]) the associativity of union of behavioural specifications

$$(\mathrm{SP} \cup \mathrm{SP}') \cup \mathrm{SP}'' = \mathrm{SP} \cup (\mathrm{SP}' \cup \mathrm{SP}'')$$

means that either both members are defined and are equal semantically or else that neither of them is defined.

In the case of the instantiation of parameterised behavioural specifications the pushout square (2) is replaced with the following pushout square:

$$\begin{tikzcd}
P \cup (\mathrm{SP} \cap \mathrm{SP}_1) \arrow[r, "\subseteq"] \arrow[d, "v \cup \mathrm{id}"'] & \mathrm{SP} \arrow[d] \\
\mathrm{SP}_1 \arrow[r] & \mathrm{SP}'_1
\end{tikzcd} \tag{4}$$

when $P \cup \mathrm{SP}_1$ is defined. When $\mathrm{SP} \cup \mathrm{SP}_1$ is defined too, (4) can be replaced by the technically more convenient (3). However in this case the condition underlying (3) is stronger than that of (4) (note that according to [19] while unions of behavioural specifications are partial, intersections are total).

3 Conclusions and Future Research

We have presented some important elements of modularisation theory in formal specification, including important mathematical tools (institutions, inclusion systems) and methods (pushout-style parameterisation). In the second part of the paper we have discussed recent developments such as two-layered institution-independence, upgrade to pushout-style, multiple parameters, and behavioural specification. The latter three topics have been developed in close collaboration with Ionuţ Ţuţu.

At this moment there is ongoing work on parameterisation with sharing for behavioural specifications, on general views for parameter instantiation, etc.

References

[1] Marc Aiguier and Fabrice Barbier. An institution-independent proof of the Beth definability theorem. *Studia Logica*, 85(3):333–359, 2007.

[2] Edigio Astesiano, Michel Bidoit, Hélène Kirchner, Berndt Krieg-Brückner, Peter Mosses, Don Sannella, and Andrzej Tarlecki.

CASL: The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002.

[3] Jan Bergstra, Jan Heering, and Paul Klint. *Algebraic Specification.* Association for Computing Machinery, 1989.

[4] Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.

[5] Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Behavioural and abstractor specifications. *Sci. Comput. Program.*, 25(2-3):149–186, 1995.

[6] Michel Bidoit, Donald Sannella, and Andrzej Tarlecki. Observational interpretation of CASL specifications. *Mathematical Structures in Computer Science*, 18(2):325–371, 2008.

[7] Francis Borceux. *Handbook of Categorical Algebra.* Cambridge University Press, 1994.

[8] Tomasz Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286(2):197–245, 2002.

[9] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *1979 Copenhagen Winter School on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer, 1980.

[10] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude - A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science.* Springer, 2007.

[11] Ionuţ Ţuţu. Comorphisms for structured institutions. *Information Processing Letters*, 113(894–900), 2013.

[12] Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems. *Mathematical Structures in Computer Science*, 7(2):195–206, 1997.

[13] Răzvan Diaconescu. Elementary diagrams in institutions. *Journal of Logic and Computation*, 14(5):651–674, 2004.

[14] Răzvan Diaconescu. *Institution-independent Model Theory.* Birkhäuser, 2008.

[15] Răzvan Diaconescu. An axiomatic approach to structuring specifications. *Theoretical Computer Science*, 433:20–42, 2012.

[16] Răzvan Diaconescu. Three decades of institution theory. In Jean-Yves Béziau, editor, *Universal Logic: an Anthology*, pages 309–322. Springer Basel, 2012.

[17] Răzvan Diaconescu. On the existence of translations of structured specifications. *Information Processing Letters*, 115(1):15–22, 2015.

[18] Răzvan Diaconescu and Ionuţ Ţuţu. On the algebra of structured specifications. *Theoretical Computer Science*, 412(28):3145–3174, 2011.

[19] Răzvan Diaconescu and Ionuţ Ţuţu. Foundations for structuring behavioural specifications. *Journal of Logical and Algebraic Methods in Programming*, 83(3–4):319–338, 2014.

[20] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[21] Răzvan Diaconescu and Kokichi Futatsugi. Behavioural coherence in object-oriented algebraic specification. *Universal Computer Science*, 6(1):74–96, 2000. First version appeared as JAIST Technical Report IS-RR-98-0017F, June 1998.

[22] Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. *Theoretical Computer Science*, 285:289–318, 2002.

[23] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993. Proceedings of a Workshop held in Edinburgh, Scotland, May 1991.

[24] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[25] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification*, volume

785 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 1994.

[26] Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.

[27] Joseph Goguen and Grigore Roşu. Composing hidden information modules over inclusive institutions. In Olaf Owe, Stein Krogdahl, and Tom Lyche, editors, *From Object-Orientation to Formal Methods*, volume 2635 of *Lecture Notes in Computer Science*, pages 96–123. Springer, 2004.

[28] Rolf Hennicker and Michel Bidoit. Observational logic. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology*, number 1584 in LNCS, pages 263–277. Springer, 1999. Proc. AMAST'99.

[29] Kestrel Institute. *Specware system and documentation*, 2003. `www.specware.org`.

[30] Oliver Kutz, Till Mossakowski, and Dominik Lücke. Carnap, Goguen, and the hyperontologies - logical pluralism and heterogeneous structuring in ontology design. *Logica Universalis*, 4(2):255–333, 2010.

[31] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, second edition, 1998.

[32] T. Mossakowski, C. Maeder, and K. Lütich. The heterogeneous tool set. In *Lecture Notes in Computer Science*, volume 4424, pages 519–522. 2007.

[33] Horst Reichel. Behavioural equivalence – a unifying concept for initial and final specifications. In *Proceedings, Third Hungarian Computer Science Conference*. Akademiai Kiado, 1981. Budapest.

[34] Horst Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras.* Clarendon, 1987.

[35] Grigore Roşu. *Hidden Logic.* PhD thesis, University of California at San Diego, 2000.

[36] Grigore Roşu. Axiomatisability in inclusive equational logic. *Mathematical Structures in Computer Science*, 12(5):541–563, 2002.

[37] Grigore Roşu and Dorel Lucanu. Circular coinduction: A proof theoretical foundation. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Algebra and Coalgebra in Computer Science*, volume 5728 of *Lecture Notes in Computer Science*, pages 127–144, 2009.

[38] Donald Sannella and Andrzej Tarlecki. Specifications in an arbitrary institution. *Information and Control*, 76:165–210, 1988.

[39] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specifications and Formal Software Development.* Springer, 2012.

[40] Ionuţ Ţuţu. Parameterisation for abstract structured specifications. *Theoretical Computer Science*, 517:102–142, 2014.

[41] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topic. Spass version 2.0. In *Proceedings of the 18th International Conference on Automated Deduction*, CADE-18, pages 275–279, London, UK, 2002. Springer-Verlag.

Received September 11, 2015

Răzvan Diaconescu

Simion Stoilow Institute of Mathematics of Romanian Academy
Romania
E–mail: `Razvan.Diaconescu@imar.ro`