Specifications of Software Architectures using
Diagrams of Constructions

Extended abstract of PhD dissertation

Grzegorz Marczyński

Abstract
Formal methods promise the ultimate quality of software artifacts with mathematical proof of their correctness. Algebraic specification is one of such methods, providing formal specifications of system components suitable for verification of correctness of all individual steps in the software development process, and hence of the entire development process and of the resulting program.

In this thesis we propose a new approach to algebraic specifications of software architectures, called diagrams of construction specifications. Constructions, as introduced here, model parameterised modules, with dependency relation captured directly on signature symbols. They give a uniform treatment of first- and higher-order parameterisation, and are equipped with a single sum operation which subsumes the most standard operations on parameterised modules. We introduce specifications for such constructions, study their compositionality properties, and define a notion of refinement for constructor specifications. Diagrams of construction specifications capture design and development of modular software architecture, based on decomposition and refinement of construction specifications.

Throughout the thesis we illustrate new concepts and problems discussed by means of simple examples; a somewhat longer example is also added to summarize our presentation.

Keywords: formal methods, algebraic specifications, software engineering, software architecture, category theory, institution theory

1 Introduction

There are two main roles of software architectures [GS94]: to describe software system decomposition into components and their interconnections, and to define the system development process and its evolution. The goal of such description is to guarantee the important qualities like reusability, scalability, and adaptability.
Formal methods promise the ultimate quality of software artifacts by providing a mathematical proof of their correctness with respect to formally presented requirements. Unfortunately, the use of formal methods in practical software development is still limited to core components of critical systems. The main reason for that situation is the higher cost of formal methods use when compared with popular quality assurance approaches based on testing and good practice. Wider adoption of formal methods requires simplification and automation.

A formal method that we examine in this thesis is algebraic specification. The idea is to provide formal specifications of system components and to prove the correctness of single steps in the development process, thus, by construction, ensuring correctness of the finally composed system.

Algebraic specifications are formalised in the language of category theory [ML98, AHS90] using institution theory [GB84, BG92, ST12, Dia08], abstract formulations of model theory of logical systems. Many different logics have been shown to be institutions

Parameterised programming [Gog96], ACT TWO [EM90]) and CASL architectural specifications [ST88, Mos04] are three representative examples of algebraic specification frameworks aiming at formal development of software systems based on modularisation and reuse.

The basic building blocks in the three approaches are (specifications of) parameterised modules (called modules in ACT TWO and generic units in CASL). Their parameterisation is of first-order functional type on the signature level, i.e. module specifications are signature morphisms together with specifications of parameters and specification of the result. A module realisation (implementation), may be represented as $\lambda X : SP_P . B[X] : SP_R$, where $SP_P$ is a specification of the parameter, $SP_R$ is a specification of the result, $X$ is the formal parameter and $B$ is the body of the module’s realisation, which typically extends $X$.

All three approaches provide basic operations on modules, like composition, instantiation, enrichment and hiding. Every module operation requires additional connection between modules, be it a view or a fitting morphism. A module expression (or result expression) combines modules represented in a module graph (or unit declaration and definition list) into the resulting parameterised module. In all approaches the explicit sharing resolution is required, thus the interconnections between modules in a module graph usually are non-trivial.

The plurality of operations and the need for a module expression to compose a system is a source of potential confusion. The same modules connected via the same views produce different results, depending on the operation that is prescribed to combine them. This overly complicates the specification process, where, on such a high level, the composition should be a simple operation,
without additional unnecessary variation.

All three frameworks mentioned above are equipped with the functional signature-level parameterisation, which makes partial instantiation of modules impossible or requiring some additional work. It is also not evident whether all parameters are actually needed by a parameterised module, because there is no structure of the fine-grained (in)dependency between result symbols and parameter symbols. Thus one needs to assume that all parameters are needed and consider a system incomplete if some of them are missing, even when the missing symbol is not actually required by the result to be complete. As a consequence, also mutual and reflexive instantiations are problematic.

An extension from first-order to higher-order parameterisation, while increasing considerably the expressiveness of the language, requires a tremendous complication of syntax [ST12], which makes such an extended framework unusable from the practical point of view.

We consider the above-presented limitations as potential pitfalls when it comes to real-life use of algebraic specification framework.

In the thesis we define a new approach to software architecture specifications that gives uniform treatment of various types of parameterised modules and their fitting connectives. By providing only a single composition operation we achieve automatic composition of module implementations. Parameterisation is defined on the symbol level, which allows for straightforward expression of partial instantiation and gives a simple, clearly defined sharing between modules. The proposed framework supports the top-down approach to software development by step-wise refinement and gives a very intuitive representation of system architectures in the form of diagrams of system components.

2 Main Contributions

The main contribution of the thesis is introduction of diagrams of construction specifications, defined using terminology from category theory and institution theory. Brief introduction of the main notions from these two domains allows us to formally describe the contributions in an understandable way. Category $\mathbf{C}$ consists of a collection of objects $|\mathbf{C}|$ and, for each pair of objects $o, o' \in |\mathbf{C}|$, a collection of morphisms from the source $o$ to the target $o'$, $\mathbf{C}(o, o')$, such that there exists the associative composition of morphisms with identities. The category dual to $\mathbf{C}$ is the category $\mathbf{C}^{op}$ which has the same objects as $\mathbf{C}$, but the direction of morphisms is reversed. A category is small when the collections of objects and morphisms are sets. Between categories one defines functors that are functions on objects and morphisms such that they preserve identities and composition. In the category of sets $\mathbf{Set}$, the objects are sets and the morphisms
are functions. In the category of small categories \( \text{Cat} \), the objects are categories and the morphisms are functors. A diagram in a category \( \mathbf{C} \) of shape of a small category \( \mathbf{J} \) is a functor \( \mathbf{D} : \mathbf{J} \to \mathbf{C} \). There are limits and colimits of diagrams in categories. A special colimit is a pushout which is a sum of a pair of morphisms with the same source. An institution \( \mathcal{I} = \langle \text{Sig}, \text{Mod}, \text{Sen}, \models \rangle \) consists of a category of signatures \( \text{Sig} \), a model functor \( \text{Mod} : \text{Sig}^{op} \to \text{Cat} \) (for a signature it gives its category of models), a sentence functor \( \text{Sen} : \text{Sig} \to \text{Set} \) (for a signature it gives a set of sentences over it) and a family of satisfaction relations \( \{\models \}_{\Sigma} \) \( \subseteq \text{Mod}(\Sigma) \times \text{Sen}(\Sigma) \) for each \( \Sigma \in \text{Sig} \), such that they meet the satisfaction condition: for each signature morphism \( \sigma : \Sigma \to \Sigma' \), \( \Sigma' \)-model \( M' \) and \( \Sigma \)-sentence \( \varphi \), it holds that \( \text{Mod}(\sigma)(M') \models \Sigma \varphi \) if and only if \( M' \models \Sigma' \text{Sen}(\sigma)(\varphi) \). The equational logic formalised as an institution is named \( \text{EQ} \), the first-order logic with equality is named \( \text{FOEQ} \). Specifications in \( \mathcal{I} \) over a signature \( \Sigma \) are presentations \( \langle \Sigma, \Phi \rangle \), where \( \Phi \) is a set of \( \Sigma \)-sentences and structural specifications obtained by the union operation \( \text{SP} \cup \text{SP}' \), the translation operation \( \sigma(\text{SP}) \) and the hiding operation \( \text{SP}|_{\sigma} \).

In what follows we assume that there is given an institution \( \mathcal{I} \), subject to some technical assumptions met by standard institutions, like \( \text{EQ} \) and \( \text{FOEQ} \). Among others, we require the existence of a functor that for each signature gives the set of its symbols and the existence of the inclusion between signatures.

### 2.1 Signature Fragments and Dependency Structures

The first notion introduced in the thesis are signature fragments, which allow one to distinguish a subset of symbols of the given signature. Signature fragment of \( \Sigma \in \text{Sig} \) is a pair \( \Sigma = \langle A, \Sigma \rangle \), where \( A \) is a set of defined symbols; the remaining symbols from \( \Sigma \) are called assumed symbols. Signature fragments form a category \( \text{Sig}^{frags} \). We define the embedding functor \( \text{Frag} : \text{Sig} \to \text{SigDep} \) which represents a signature as a complete signature fragment, i.e. such that all its symbols are defined, and the completion functor \( \text{Compl} : \text{Sig} \to \text{SigDep} \) which, given a signature fragment, leaves only its second component, i.e. the signature. Another notion introduced in the thesis is a dependency structure put on the symbols from a signature. A signature \( \Sigma \) extended by a dependency structure is a pair \( \Sigma_{<} = \langle \Sigma, \prec \rangle \), where \( \prec \) is a dependency relation which is a strict order on the symbols from \( \Sigma \) such that all descending chains are finite (bounded). Signatures with dependency structure form a category \( \text{SigDep} \) with morphisms (called \( p \)-morphisms) which are monotonic and weakly reflect down-closures of the dependency relation; as a result the dependency structure

\[1\] The definition given here is slightly simplified, but it gives the essence of the signature fragments.
of the symbols in the source and in the target of the morphisms are the same
and there is no way to add new symbols to the dependency structure. Finally
we combine the two above-described notions to obtain signature fragments with
dependencies that form a category \( \text{SigDep}^{\text{frag}} \). Objects in \( \text{SigDep}^{\text{frag}} \) are
marked as \( S \in [\text{SigDep}^{\text{frag}}] \). To simplify the argument we introduce the notion
of a structure below a given symbol, i.e., for a given symbol \( a \) from \( S \), we define
\( S^a \downarrow \) as a signature fragment with dependencies included in \( S \) containing exactly
these symbols that are lower than \( a \) w.r.t. the dependency relation. By \( S^a \downarrow \) we
denote a dependency structure of symbol \( a \) which is a signature fragment with
dependencies that contains all symbols from \( S^a \downarrow \) and additionally symbol \( a \).
This notation is extended to sets of symbols from \( S \) by introducing dependency
structure of set of symbols \( A \) denoted as \( S^A \downarrow \).

2.2 Constructions

Using the above-introduced notions we define constructions which are symbol-
level parameterised modules suitable for uniform representation of the first-
and the higher-order parameterisation. Construction signatures are signature
fragments with dependencies. Assumed symbols are interpreted as symbols from
the parameter of a module. Defined symbols are interpreted as symbols from
the construction result of a module. The example below shows construction
signature and the graph of its dependency structure.

\[
\begin{align*}
S_2 &= \text{sort} \ Nat;
\text{ops} \quad 
\begin{align*}
\& \quad \text{zero} : \ Nat, \\
\& \quad \text{succ} : \ Nat \rightarrow \ Nat,
\& \quad a : \ Nat, \ b : \ Nat,
\& \quad c : \ Nat, \ d : \ Nat;
\text{deps} \quad 
\begin{align*}
\& \quad \text{zero} < \text{succ}, \\
\& \quad \text{succ} < a, \\
\& \quad a < b, \ a < d, \\
\& \quad b < c, \ d < c 
\end{align*}
\end{align*}
\end{align*}
\]

All underlined symbols are assumed. There are two defined and five assumed
symbols in \( S_2 \). Each construction (model) of \( S_2 \) gives interpretation of \( a : Nat \)
and \( c : Nat \). All other symbols are parameters. Dependency structure exactly
describes, which symbols may potentially be used to construct other symbols.
Defined symbol \( a : \text{Nat} \) is dependent only on assumed symbols in its dependency structure, so \( a : \text{Nat} \) is an example of the first-order parameterisation. However, defined symbol \( c : \text{Nat} \) depends on assumed symbols that are dependent on an defined symbol that is dependent on assumed symbols. Such dependency is interpreted as the second-order parameterisation.

A model of a construction signature \( S \), called a construction model and denoted by \( \text{Con} \in [S]^\text{c} \), is defined as a class of models of signature \( \text{Compl}(S) \), subject to the condition that for any defined symbol \( a \in S \) and any two models \( M, M' \in \text{Con} \), if \( M|_S a \downarrow = M'|_S a \downarrow \) then \( M|_S a \downarrow = M'|_S a \downarrow \), where the vertical arrow denotes the reduct of the given model to the signature which is a component of the construction signature. The explanation of the above condition is the following: for every defined symbol \( a \) in the construction signature and for every interpretation of the dependency structure below \( a \) present in the construction model, there is a unique construction of the model of the dependency structure of \( a \) (including \( a \)). A construction specification over a construction signature \( S \) is a pair \( SP = (\widetilde{S}, SP) \), where \( SP \) is a specification over \( \text{Compl}(S) \) in institution \( I \). The satisfaction relation between construction models \( \text{Con} \in [S]^\text{c} \) and construction specifications \( SP = (\widetilde{S}, SP) \) over \( S \) denoted by \( \text{Con} |\text{=}^c SP \), requires satisfaction of the following four conditions:

1. for every defined symbol \( a \) in \( S \) and every model \( M \in \text{Con} \),
   
   if \( M|_S a \uparrow = SP|_S a \uparrow \) then \( M|_S a \downarrow = SP|_S a \downarrow \),

2. for every assumed symbol \( a \) in \( S \) and every model \( M \models SP \),
   
   if \( M|_S a \downarrow \in \text{Con}|_S a \downarrow \) then \( M|_S a \uparrow \in \text{Con}|_S a \uparrow \),

3. for every set of symbols \( A \) in \( S \) and every model \( M \) of \( \text{Compl}(S) \),
   
   if for every \( a \in A \), \( M|_S a \downarrow \in \text{Con}|_S a \downarrow \), then \( M|_S A \downarrow \in \text{Con}|_S A \downarrow \),

4. for every set of symbols \( A \) in \( S \) and every model \( M \in \text{Con} \),
   
   if for every \( a \in A \), \( M|_S a \downarrow = SP|_S a \downarrow \) then \( M|_S A \downarrow = SP|_S A \downarrow \).

Condition (1) guarantees that all defined symbols, i.e. the result of the construction, satisfy the specification, under the condition that the reduct of the model to the dependency structure below these symbols also satisfies the specification. Condition (2) is a kind of completeness condition, which guarantees that all possible interpretations of all assumed symbols are available in the construction, as long as they are consistent with the construction specification. Condition (3) complements the completeness condition (2) and requires that, for all interpretations of a set of symbols, all their combinations be present in
the construction model. Condition (4) guarantees that the specification of symbols does not depend on the specification of the symbols that are not in their dependency structures.

In order to allow for a composition of constructions, we introduce a *sum operation* that works on construction signatures, models and specifications. The sum is a symmetric operation suitable to compose two constructions connected by a fitting span, i.e. a pair of morphisms in \( \text{SigDep}^{\text{frag}} \) with the common source object. There is a requirement put on such pairs, saying that the construction signature which is the source of the fitting span contains no defined symbols. The *sum of construction signatures* is given as the pushout of the fitting span. The *sum of construction models* is the class of models of the signature component of the sum of construction signatures such that all its models reduce to some models from the summand construction models. The *sum of construction specifications* is the union of the translated summand construction specifications. In the thesis we prove the mutual correctness of all three operations under the static *compatibility condition* put on the construction specifications connected by a fitting span. It is also shown how to use the sum operation to express the standard operations on parameterised modules – union, composition, application, partial application, and mutual application.

An another notion introduced in the thesis is *construction specification refinement* denoted by \( \mathcal{SP}_1 \sim \mathcal{S} \rightarrow \mathcal{S} \mathcal{P}_2 \), where \( \mathcal{S} \) is a new kind of morphism between construction signatures, called *construction signature refinement morphism*, which allows for addition of new symbols in dependency structures. This morphism (with some additional assumptions) is used to prove one of the implications of the satisfaction condition: if \( \text{Con}_2 \models \mathcal{S}_1 \mathcal{P} \) then \( \text{Con}_2 \models \mathcal{S}_1 \). It is also shown that the construction specification refinement is in fact the refinement in the standard sense, meaning that for \( \mathcal{S}_1 \rightarrow \mathcal{S}_2 \), if \( \mathcal{S}_1 \sim \mathcal{S} \rightarrow \mathcal{S}_2 \) then for any construction model \( \text{Con}_2 \models \mathcal{S}_1 \mathcal{P} \), it holds that \( \text{Con}_2 \models \mathcal{S}_1 \mathcal{P} \) (under the additional assumption that \( \text{Con} \) contains only models that satisfy \( \mathcal{S} \mathcal{P} \) and that \( \mathcal{S}_1 \) contains only a finite number of symbols). Also the compositionality of the construction signature morphisms w.r.t. the construction signature refinement morphisms is shown in the thesis.

### 2.3 Diagrams of Constructions

All above notions are used to introduce the new approach to specifications of software architectures, called *diagrams of construction specification* (or, *diagrams of constructions*, for short). We introduce the category of construction specifications \( \text{SpecDep}^{\text{con}} \) which has construction specifications as morphisms and \( \text{Sig}^{\text{frag}} \)-morphisms as morphisms, which enables the uniform use of both
construction signature morphisms and construction signature refinement morphisms. A diagram of construction specifications of shape \( J \) is a diagram \( \mathcal{D}: J \rightarrow \text{SpecDep}_{\text{con}} \) such that it can be seen as a connected acyclic directed graph (dag) when all construction signature morphisms have reversed direction, with the single result node, construction signature morphisms used only to form sum squares and construction signature refinement morphisms used only as construction refinements from side nodes of the squares to either single nodes or top nodes of the squares\(^2\). The example below presents a diagram of constructions

\[
\begin{align*}
&\beta_1 \quad SP \\
&\phi_1 \quad SP' \quad \beta_2 \quad SP_2 \\
&SP_1 \quad \sim \quad SP' \quad \beta_1' \quad SP'_{11} \\
&\phi_1' \quad SP'_{12} \quad \phi_2' \quad SP'_{22}
\end{align*}
\]

and the corresponding dag

\[
\begin{align*}
&\beta_1 \quad \beta_2 \\
&\phi_1 \quad \sim \quad \beta_1' \quad \beta_2' \\
&\phi_1' \quad \phi_2'
\end{align*}
\]

Diagrams of constructions are homogenous with construction specifications, construction refinements, and sum squares. Every diagram has a distinguished set of seed nodes which are not a source of any refinement arrow and either are single nodes or side nodes of sum squares. The seed nodes of a diagram of constructions contain the construction specifications that have to be implemented to construct the whole system. In the example above the seed nodes are \( SP_1, SP'_{11} \) and \( SP'_{22} \).

In the thesis we show that it is enough to have a construction model for each seed node of a diagram of constructions to automatically obtain the corresponding diagram of construction models and hence the result of the system construction. Such process of system construction corresponds to the top-down approach to system development.

\(^2\)The given list of conditions is not complete, but reflects the main idea behind diagrams of constructions.
3 Conclusion

In this thesis we presented an approach to development of software architectures via diagrams of constructions. In order to unify the notion of simple and parameterised modules of the first- and higher-order we have equipped the construction signatures with additional dependencies between symbols and the information whether a given symbol is defined or assumed from the outside. It is worth to note that such approach is different from the classical functional type of parameterisation where the parameter symbols are given directly and the symbols defined by a module are those symbols from the result that are not a parameter.

We proposed a new notion of signature fragments, i.e. signatures with distinguished set of defined symbols, and extension of signatures by dependency structure. Using these notions we introduced construction signatures defined as fragments of signatures with dependency structure, where the dependency relation is a bounded strict order. We also proposed construction models defined as classes of models that share the interpretation of defined symbols from the construction signature, subject to the condition that their dependency structure is shared. We defined construction specifications as specifications from the base institution. The introduced satisfaction relation requires that the constructed (defined) symbols satisfy the specification, the parameter (assumed) symbols are represented in all variants matching the specification, any partial instantiation respecting the dependency structure is possible, and the specification respects the dependency structure (does not relate the otherwise independent symbols). We gave the definition of a sum of constructions connected by a pair of morphisms with the same source and we showed that, given construction specifications meeting compatibility conditions, the sum of construction models satisfy the sum of construction specifications. Along the above-described definitions we gave the correspondence between the new concepts and the classical formulation of parameterised modules. We also introduced a notion of construction specifications refinement.

In the thesis we took constructions as the basic building blocks of system architecture, we used the sum operation as the only way of composition and construction specification refinement as the only way of abstraction. Description of the system architecture was formalised as diagrams of construction specifications fulfilling the above assumptions.

At the end of the thesis we gave several ideas for future work, including the concept of the architecture logic suitable to specify properties of diagrams of constructions.
References


