Multiparadigm Specification Languages: a First Attempt at Foundations *

Egidio Astesiano and Maura Cerioli
DISI–Dipartimento di Informatica e Scienze dell'Informazione,
Università di Genova, Viale Benedetto XV,
6132 Genova, Italy,
e-mail: {astes,cerioli}@disi.unige.it

Abstract

This paper is a first attempt at a formal foundation of specification languages allowing their basic modules to be defined in several formalisms. More precisely a rigorous notion of a compositional tool for importing/exporting specifications between two instances of one specification metalanguage on different basic algebraic frameworks is proposed.

Adopting the notion of institution as a synonym for formalism, we introduce and develop the concept of *simulation* of an institution by another. Then we deal with the simulation of basic and structured specifications, introducing the concept of *simulation independent metalanguage*, a generalization of institution independent languages, which allows "putting together theories *from different formalisms* to make specifications". Since simulation generalizes the notion of implementation and allows relating implementations in different formalisms, a third dimension is added to the well known horizontal and vertical compositions of specifications, typical of Clear and ASL.

1 Motivations

Most software systems needed to solve concrete problems are far too large to be handled by human minds without the support of a rigorous methodology. Formal specifications, providing tools for modularity and refinement (see e.g. [9, 13]), facilitate reuse and maintenance of produced software.

As a result both of theoretical investigations and of preliminary attempts at applications, the nature of the algebras, the logics used to define the classes of admissible realizations of a data type and even the notion of signature have been changed, w.r.t. the pioneering papers of the ADJ group, more or less recently, producing a considerable proliferation of specification formalisms. Indeed this work has grown out of the experience of the first author who has been involved with his group in various formal specification projects, where different

*This work has been partially supported by Esprit-BRA W.G. n.6071 COMPASS, Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of C.N.R. (Italy), MURST-40% Modelli della computazione e dei linguaggi di programmazione

formalisms, like partial conditional logic, order-sorted logic, first-order structures have been used, depending on the target application.

From a pragmatic point of view, the existence of a number of different formalisms is pretty reasonable, because each one of them may be the more comfortable to work within, depending on the problem under examination, the field tradition, the available tools and (not least) the personal taste. However, from the point of view of a specification language user, the ability of supporting modularity and refinement is essential in order to allow reuse of specifications. Thus it is important (not to say crucial) to assemble, possibly at different levels of implementative detail, specification modules in different formalisms. Rephrasing the title of a landmark paper by Burstall and Goguen [3], the issue is "putting together theories *from different formalism* to make specifications".

This paper presents a formal approach to this translation problem, showing how it deals with modularity and refinement. It is worth noting that, in order to support the stepwise refinement, we regard specifications as classes of *possibly non-isomorphic* models, so that, by fixing implementative details at the different design levels, the model class is restricted, until only one model (up to isomorphisms) remains, that is the required realization. Thus we consider specification languages based on this loose approach (see e.g. ASL [15], or CLEAR [4]) more than languages where specifications describe (the isomorphism class of) one model (see e.g. VDM [10], Z [16], OBJ [8]).

Adopting the notion of institution, developed by Burstall and Goguen (see e.g. [6]) in order to define the semantics of the CLEAR language, as a rigorous counterpart of the notion of formalism, we use the concept of simulation (see e.g. [1, 5] of an institution by another (section 2): if μ is a simulation of $\mathcal{I}$ by $\mathcal{I}'$, then $\mathcal{I}'$ has at least the same expressive power as $\mathcal{I}$ and moreover μ indicates how $\mathcal{I}$ can be translated into $\mathcal{I}'$. Intuitively a simulation μ of $\mathcal{I}$ by $\mathcal{I}'$ codes the syntax of $\mathcal{I}$, i.e. the signatures and the sentences, into the syntax of $\mathcal{I}'$ in a consistent way w.r.t. the semantics, in the sense that a class of models in $\mathcal{I}'$ is chosen to represent the models of $\mathcal{I}$. Particularly interesting are *logical* simulations, that are those for which the class of models in $\mathcal{I}'$ chosen to represent the models of $\mathcal{I}$ are the model class of a set of $\mathcal{I}'$ sentences, because for such simulations there is an immediate correspondence between the basic specifications (i.e. the model classes of sets of sentences) of the two frames.

Then the basic notion of simulation between institutions is extended to deal with simple and structured specifications. In particular using any simulation of a framework $\mathcal{I}$ by another $\mathcal{I}'$, every specification language defined on $\mathcal{I}$ is allowed to import, in a rigorous way, specifications defined for the $\mathcal{I}'$ paradigm. However this capability is in some sense rough, as the structure of the imported specification, if any, is lost by the importing process. In order to improve the import process, the first step is using, instead of two generic languages on $\mathcal{I}$ and $\mathcal{I}'$, instances of one common metalanguage whose semantics is uniformly defined for both frameworks so that preserving the structure has a formal meaning; for this aim the notion of institution independent metalanguage by Sannella and Tarlecki (see e.g. [14]) is perfectly fitting. However institution independence is not sufficient for every translation to be compatible with the structure of the language and has to be refined to that of *(logical) simulation independent operation* (section 3), which is the basis for (logical) simulation independent metalanguages. Such metalanguages can structure specifications defined in several frameworks, provided that the translations of these input specifications into a common framework are given by means of (logical) simulations of the original paradigms by the new chosen one. Although this approach is very promising, non every detail has been fixed yet and in particular it is still under investigation which can be the most suitable set of (logical) simulation independent operators, depending on the actual specification languages used in practice.

Finally, in connection with the refinement problem, it is shown how simulation adds a third dimension to the well known horizontal and vertical composition of implementations, thus allowing the composition of software modules not only from different formalisms, but also at different levels of abstraction.

The concept of simulation plays a central role in the Ph.D. thesis [5], where many applications are explored and an exhaustive comparison with related works is made. In [1] it is shown how simulations may provide a tool for comparing frameworks and translating deductive systems. In the present paper, corresponding roughly to chapter 5 in [5], the emphasis is on multiparadigm specification languages.

Following the suggestions of the referees, all technical details have been summarized in an appendix, keeping the exposition at a semiformal level.

2 The concept of simulation

To introduce the concept of simulation (see e.g. [1]) and the corresponding notation from an intuitive point of view, we begin with an informal example, which is the reduction of partial algebra logic with ground conditional formulas built on existential equations ($t = t'$ holds iff t and t' denote the same element), from now on $\mathcal{PAR}$, to the conditional fragment of typed logic, from now on $\mathcal{TL}$, making explicit the definedness of elements, by definedness predicates; this technique has been used in practice in order to make deduction for partial specifications by tools defined for the total case, as in [12] (and analogously for the homogeneous total frame, see e.g. [11]).

Example 2.1 Let us fix a many-sorted signature Σ with sorts S and function symbols F. We define the translation of Σ into a typed logic signature $\Phi(\Sigma)$, by setting $\Phi(\Sigma) = (S, F, P')$, where P' contains only the typing predicates and a binary relation, playing the role of existential equality, i.e. $P'_s = \{D_s\}$ $P'_{ss} = \{\_eq_e\_\}$, where the symbol $\_$ denotes the place of the arguments in an infix notation, and $P'_w = \emptyset$ for all $w \notin \{s, ss\}$.

Any partial conditional equation over Σ can be translated into a total equivalent one over $\Phi(\Sigma)$; indeed it is sufficient to substitute the predicate eq_e for the existential equality. Thus let us consider a partial positive conditional formula $\xi = (t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \supset t = t')$ over Σ and define $\alpha_\Sigma(\xi) = (t_1\, eq_e t'_1 \wedge \ldots \wedge t_n\, eq_e t'_n \supset t\, eq_e t')$.

To illustrate in which sense $\alpha_\Sigma(\xi)$ is equivalent to ξ, a class $dom(\mu)_\Sigma$ of total algebras with predicates (first-order structures) is chosen, which soundly represents the partial algebras and s.t. a total algebra satisfies $\alpha_\Sigma(\xi)$ iff the partial algebra represented by it satisfies ξ. Informally, the idea is to distinguish any carrier of a total algebra in two parts, the "defined" elements and the junk, by means of the definedness predicates and to impose that the predicate eq_e corresponds to the restriction of the (usual) equality to the defined part; moreover, as in the partial frame functions are *strict*, i.e. the definedness of a function application result implies the definedness of each argument. More formally a total algebra A' is a sound representation of a partial algebra A, we write $A = \beta_\Sigma(A')$, iff both $a\, eq_e^{A'}\, b$ is equivalent to $\{D_s{}^{A'}(a), D_s{}^{A'}(b), a = b\}$ for all elements a and b and every sort s, and $D_s{}^{A'}(f^{A'}(a_1, \ldots, a_n))$ implies $D_{s_i}{}^{A'}(a_i)$ for all $i = 1, \ldots, n$, for every function symbol $f \in F_{s_1 \ldots s_n, s}$. If A' satisfies this condition, then A is the partial algebra whose carrier of sort s is $D_s{}^{A'}$ and f^A is the restriction of $f^{A'}$ to $s_1^A \times \ldots \times s_n^A$ for every functions symbol f. It is easy to check that A' satisfies $\alpha_\Sigma(\xi)$ iff A satisfies ξ. Note that total algebras differing only on elements which do not satisfy the definedness predicates represent the same partial algebra.

Thus, for every many-sorted signature Σ a typed logic signature $\Phi(\Sigma)$ and two functions are defined: α_Σ, which translates many sorted positive conditional sentences on Σ into Horn-Clauses on $\Phi(\Sigma)$ built on definedness and existential equality predicates, and β_Σ, which partially translates total first-order structures on $\Phi(\Sigma)$ into partial algebras on Σ and is surjective, as it is immediate to check.

Since the change of notation, via signature morphisms, has a great relevance in the algebraic approach and in particular for specification languages, being used for example to bind the actual to the formal parameters in parameterized specifications and to "put theories together to make specifications", we have to investigate the compatibility between the coding functions α_Σ and β_Σ defined for any signature Σ and the changes of notation.

Let $\bar{\sigma}\colon \Sigma_1 \to \Sigma_2$ be a morphism of many-sorted signatures, i.e. a pair of functions $\sigma\colon S_1 \to S_2$, renaming the sorts, and $\phi\colon F_1 \to F_2$ translating function symbols in a consistent way w.r.t. the sort renaming (i.e. if $f\colon s_1 \times \ldots \times s_n \to s$, then $\phi(f)\colon \sigma(s_1) \times \ldots \times \sigma(s_n) \to \sigma(s)$). Then $\bar{\sigma}$ naturally induces a typed logic signature morphism (σ, ϕ, π) from $\Phi(\Sigma_1)$ into $\Phi(\Sigma_2)$, defined by $\pi(D_s) = D_{\sigma(s)}$ and $\pi(eq_e) = eq_e$ for any $s \in S$. It is easy to check that the translation of sentences is compatible with signature morphisms, i.e. that ${}_{\Sigma_1}(\bar{\sigma}(\xi)) = \Phi(\bar{\sigma})({}_{\Sigma_1}(\xi))$, where the application of a signature morphism to a sentence is the usual renaming of function (and predicate) symbols.

Instead the partiality of the translation of algebras makes the compatibility between the algebra translations and signature morphisms delicate. Indeed it is intuitive to expect that the translation along a signature morphism of a total algebra simulating a partial algebra simulates the translation of that partial algebra; more formally, recalling that algebras are translated along signature morphisms in a countervariant direction into their *reduct*, we have that if $A' \in dom(\mu)_{\Sigma_2}$, then $A'_{|\Phi(\bar{\sigma})} \in dom(\mu)_{\Sigma_1}$ and $(\beta_{\Sigma_2}(A'))_{|\bar{\sigma}} = \beta_{\Sigma_1}(A'_{|\Phi(\bar{\sigma})})$. But it may be that $A' \notin dom(\mu)_{\Sigma_1}$ only because the interpretation of one function symbol f in A' is non-strict and that such an f is dropped by $\Phi(\bar{\sigma})$, so that $A'_{|\Phi(\bar{\sigma})} \in dom(\mu)_{\Sigma_1}$ and hence the converse of the first implication does not hold. Therefore we have a weaker condition (called *partial-naturality*) for algebras than the one for sentences.

Generalizing this example, we have that every framework consists of a category **Sign** of *signatures* (representing the symbols for types, operations and such, together with the admissible changes of notation) and, for every signature Σ, of a set of *sentences* $Sen(\Sigma)$ and of a category $Mod(\Sigma)$ of *models*; $Mod(\Sigma)$ and $Sen(\Sigma)$ are related by the *validity relation* $\models_\Sigma$: $A \models_\Sigma \xi$ states that the model A satisfies the sentence ξ. It is important to notice that the validity relation is invariant under change of notation, i.e. the reduct of a model along a signature morfism satisfies a formula iff the model satisfies the translation of the formula along the same signature morfism. This notion of logical framework is formalized by the *institutions* by Goguen and Burstall (see e.g. [6]), whose definition is recalled in appendix.

Let us abstract from the above construction the general aspects of the coding of a *new* formalism $\mathcal{I}$ into an *old* one $\mathcal{I}'$, as in the example the "new" partial framework was reduced the the "old" total; a *simulation* $\mu: \mathcal{I} \to \mathcal{I}'$ consists og:

- ◇ of a translation $\Phi: \mathbf{Sign} \to \mathbf{Sign}'$ of the new signatures in terms of the old ones;
- ◇ a translation $\alpha_\Sigma: Sen(\Sigma) \to Sen'(\Phi(\Sigma))$, for every signature Σ, coding the new sentences into the old ones on the corresponding signature;
- ◇ a partial surjective mapping $\beta_\Sigma: Mod'(\Phi(\Sigma)) \to Mod(\Sigma)$, for every signature Σ, of the old models into the new ones on the corresponding signature.

Moreover the translation of sentences has to be *natural* w.r.t. the signature morphisms, i.e. the following diagram commutes for every $\sigma: \Sigma \to \Sigma'$:

$$\begin{tikzcd}
Sen(\Sigma) \arrow[r, "\alpha_\Sigma"] \arrow[d, "Sen(\sigma)"'] & Sen'(\Phi(\Sigma)) \arrow[d, "Sen'(\Phi(\sigma))"] \\
Sen(\Sigma') \arrow[r, "\alpha_{\Sigma'}"'] & Sen'(\Phi(\Sigma'))
\end{tikzcd}$$

Analogously the translation of models has to be *partially-natural* w.r.t. the signature morphisms, i.e. if the lower path of the following diagram exists, then the upper path exists too and they are equal for every $\sigma: \Sigma \to \Sigma'$:

$$\begin{tikzcd}
Mod'(\Phi(\Sigma)) \arrow[r, "\beta_\Sigma"] & Mod(\Sigma) \\
Mod'(\Phi(\Sigma')) \arrow[r, "\beta_{\Sigma'}"'] \arrow[u, "Mod'(\Phi(\sigma))"] & Mod(\Sigma') \arrow[u, "Mod(\sigma)"']
\end{tikzcd}$$

This scheme generalizes to the frame of institutions by lifting maps to the proper categorical objects, taking care of the delicate points due to the partiality of model translation, and requiring that the only non-categorical structure, i.e. the validity relation, is preserved (see Def. A.2).

3 Simulations and modularity

In order to support stepwise refinement, it seems crucial that a specification is the collection of its possible realizations, i.e. of algebras satisfying appropriate requirements, expressed at the first stage by axioms in the framework and then by more general properties. Indeed such realizations are non-necessarily isomorphic, so that more decisional details can be fixed at every refinement step, restricting the class of possible models toward a completely determined model, hopefully defined in a (pseudo)executable language. Thus, as the modularity principle requires the ability of "putting together" specifications by means of structuring operations, we want to extend the capabilities of specification languages by allowing the different specifications that have to be composed to be defined in different frameworks. Therefore simulations, dealing only with the basic objects of a frame (i.e. signatures, sentences and models) have to be extended to work on specifications, i.e. classes of models, and the compatibility with structuring operations has to be investigated.

3.1 Basic Specifications

Informally a specification is the collection of the admissible models of a data type; formally it is completely determined by a class of algebras over one signature. Thus for every institution $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$, a *specification functor* $Spec_{\mathcal{I}}: \mathbf{Sign} \to \mathbf{Cat}^{\mathbf{Op}}$ is defined, associating each signature Σ with the power set of its model class, i.e. $Spec_{\mathcal{I}}(\Sigma) = \wp(|Mod(\Sigma)|)$.

It is worth noting that the above construction is a particular case of building a new kind of objects starting from the ones explicitly given in the definition of institution, like models, signatures and so on. In the next subsection we will face the problem from a more general point of view.

Having built a new kind of objects, a new component of the simulation dealing with them has to be defined and is called from now on γ, possibly decorated. As the construction of specifications relies on algebras, the modularity principle requires that γ is analogously based on β, i.e. that if $\gamma(sp)$ is defined, then $\gamma(sp) = \beta(sp)$, where $\beta(sp) = \{\beta(A) \mid A \in sp\}$; but it is not obvious which specifications have to be translated.

The minimal requirement that has to be imposed so that $\gamma(sp)$ is defined is that every model $A \in sp$ belongs to the domain of the simulation, because this condition guarantees that the validity relation extended to specifications is reflected. Indeed, defining $sp \models \phi$ iff $A \models \phi$ for all $A \in sp$, it is always true that $sp' \models' \alpha(\phi)$ implies $\beta(sp') \models \phi$, because if $A' \models' \alpha(\phi)$, then $\beta(A') \models \phi$ for any A' s.t. $\beta(A')$ is defined, by definition of simulation. But in general $\beta(sp') \models \phi$ does not imply $sp' \models' \alpha(\phi)$, because A' may exist s.t. $A' \not\models' \alpha(\phi)$ and $\beta(A')$ is not defined, so that the condition of validity preservation by simulation does not apply. Of course the totality of β over sp', which is not necessary in the general case, in many significant cases is needed.

Besides requiring β to be total on sp, many other conditions can be imposed in order that a specification translation fits special purposes. This lack of canonicity leads to define, for every simulation $\mu: \mathcal{I} \to \mathcal{I}'$, a *specification extension* of μ to be any partially-natural transformation $\delta: Spec_{\mathcal{I}'} \cdot \Phi \to Spec_{\mathcal{I}}$ s.t. if $\delta(sp)$ is defined then $sp \subseteq dom(\beta)$ and $\delta(sp) = \beta_{\Sigma}(sp)$.

Among the many possible specification extensions, a particular role is played by the *maximal* specification extension γ of μ (one for each signature Σ), defined by:
if $sp \subseteq dom(\mu)_\Sigma$, then $\gamma_\Sigma(sp) = \beta_\Sigma(sp)$, else $\gamma_\Sigma(sp)$ is undefined.

Since β is surjective on models, there exists at least one $sp' \in Spec_{\mathcal{I}'}(\Phi(\Sigma))$ for each $sp \in Spec_{\mathcal{I}}(\Sigma)$ s.t. $\gamma_\Sigma(sp') = sp$. Such an sp' consists of $\{A' \mid \beta_\Sigma(A') \in sp\}$. This formalizes the intuitive idea that an institution is simulated by another one if each of its specifications is "implemented" by at least one specification of the other institution.

Using simulations, then, specifications can be imported/exported between institutions; thus every specification language L defined for a frame represented by an institution $\mathcal{I}$ can be enriched by constructs of the form

$$\texttt{import}\ sp\ \texttt{via}\ \mu$$

where sp is a specification in an institution $\mathcal{I}'$ (possibly defined using another language L') and μ is a simulation from $\mathcal{I}$ into $\mathcal{I}'$. This capability is reminiscent of the feature common to most programming environment, allowing a program in a language to use an external module defined in another language.

Let us analyze a standard example of use of such an "import" feature: the specification of the natural numbers with sum and product is enriched by a *predecessor* operation. Let us consider how this can be realized in different formalisms. First of all consider the usual specification of natural numbers in a total many-sorted paradigm.

spec Nat =
  sorts N
  opns $z\colon \to N$
    $s\colon N \to N$
    $\_+\_\colon N \times N \to N$
    $\_*\_\colon N \times N \to N$
  axioms $x + z = x$
    $x + s(y) = s(x + y)$
    $x * z = z$
    $x * s(y) = x + (x * y)$

Now suppose that this specification has to be enriched by a *partial* predecessor operation; as the chosen paradigm is *total*, we have to deal with the elements obtained as application of the predecessor operation to zero and to erroneous terms. There are two possibilities, neither of them pleasant: either no axioms are imposed on the errors, so that in most models, and in particular in the initial one, several distinguished "junk" elements are present, or many axioms have to be given in order to propagate the error, making the specification innecessarily long, and moreover this propagation is incompatible with the original specification, against every modularity principle. Consider indeed the following naive enrichement of the specification Nat.

spec $Nat_\perp$ = **enrich** Nat **by**
  opns $err\colon \to N$
    $p\colon N \to N$
  axioms $p(s(x)) = x$
    $p(z) = err$
  – error propagation axioms
    $s(err) = err$

$p(err) = err$
$x + err = err$
$err + x = err$
$x * err = err$
$err * x = err$

It is easy to check that from $err * x = err$ and $x * z = z$, $err = z$ follows, and then, by the error propagation axioms, every term is identified with err, so that the unique (up to isomorphism) term-generated model of the specification is the trivial one. To avoid this problem the unique possibility is introducing an "ok" predicate, false on the errors, and restricting the application of the original axioms to the "ok" elements, by conditional axioms (see e.g. [7]), so that the "parameter" Nat is changed by its enrichment.

But there is an obvious embedding ϵ of total into partial algebras that can be formalized as a simulation, so that the Nat specification can be translated along ϵ and then easily enriched in the partial framework with the successor operation.

The simulation ϵ consists of the identity translation $\bar{\Phi}$ of signatures, the identity translation $\bar{\alpha}$ of sentences and the embedding $\bar{\beta}$ of the partial algebras satisfying the family of *total axioms* $D_s(f(x_1, \ldots, x_n))$, one for each operation symbol $f: s_1 \times \ldots \times s_n \rightarrow s$ of the signature, where the x_i are distinct variables of sort s_i, respectively. As it is immediate to check, the partial algebras satisfying such axioms are all and only the total algebras on the same signature, so that $\bar{\beta}$ is well defined and surjective.

Now, using the "import_along_a_simulation" feature, we can easily define the required enrichment.

spec Nat_P = **enrich** **import** Nat **along** ϵ **by**
opns $p: N \rightarrow N$
axioms $p(s(x)) = x$

Although, as the above example shows, the capability of importing specifications along simulations is a first step toward assembling specifications from different paradigms, it is quite unsatisfactory, because the possible structure of the imported specification is lost, as in the above example the axiomatic characterization of Nat. In order to get a more powerful kind of import, the structuring operations available in either language should be translated into the other language and the simulation used to import actual specifications should be compatible with such a structure.

3.2 Structured Specifications

In order to attain the capability of translating structured specifications along simulations in a way that the structure is preserved, the languages in the two frameworks should be instances of one common metalanguage, so that the intuitive meaning of "preserving the structure" can be rigorously formalized. Thus we need an *institution independent* metalanguage (see e.g. [14]). The sorts of this metalanguage may be both *basic*, i.e. implicitly defined by the concept of institution, like the sort of signatures, of algebras and so on, and *derived*, i.e. built from the basic ones using categorical and set-theoretic concepts, like the specifications which come from the algebras applying the powerset functor. Analogously the operations of this metalanguage are defined only involving the usual categorical and set-theoretic language, so that the interpretation of sorts and operations in any institution is standard.

Let us start with a few examples of institution independent operations, imported more or less from [14] and generalizing some ASL constructs.

models is the basic operation of every formalism, usually implicit; it takes in input a theory (Σ, Γ) and yields the class of models of Γ, i.e. $models(\Sigma, \Gamma) = \{A \mid A \in Mod(\Sigma), A \models \Gamma\}$; for example $models(Nat)$ denotes the variety of total algebras satisfying the axioms on $+$ and $*$. Note that the semantics of *models* can be defined for any institution, as it only involves the notion of validity, that is one of the institution ingredients.

$\_+\_$ takes in input two specifications on the same signature and yields the specification whose models are models of both, i.e. $sp + sp' = \{A \mid A \in sp \text{ and } A \in sp'\}$. In standard frameworks, where signatures consist of sorts, (predicates) and operations and signature morphisms are changes of notation, this operation allows the implementations of different operations to be independently developed, getting several specifications on one signature, eachone with a subset of the operations axiomatically described and the remaining free (without conditions imposed on them) and then, summing the subspecifications, one specification is obtained where all operations are axiomatized.

translate _ *by* _ *in* _ takes in input a Σ-specification sp, a signature morphism $\sigma\colon \Sigma \to \Sigma'$ and a Σ'-specification sp' and yields the subclass of sp' models whose σ-reduct is in sp, i.e.

$$translate\ sp\ by\ \sigma\ in\ sp' = \{A' \mid A' \in sp', Mod(\sigma)(A') \in sp\};$$

In standard frameworks the *translate* operation is used both to rename the operation symbols (by a bijective σ) and to embed a specification in another with larger signature, so that parts of a big specification can be developed on smaller signatures and then lifted by *translate* to the original signature so that the $+$ operation applies.

derive _ *by* _ takes in input a Σ'-specification sp' and a signature morphism $\sigma\colon \Sigma \to \Sigma'$ and yields the class of σ-reducts of sp' models, i.e.

$$derive\ sp'\ by\ \sigma = \{Mod(\sigma)(A') \mid A' \in sp'\}.$$

In standard frameworks the *derive* operation is used to hide implementative parts (if σ is a non-surjective embedding, then the sorts and/or operations in $\Sigma' - \Sigma$ concur to build the Σ-models, but become invisible in the reducts $Mod(\sigma)(A')$).

Using these operations more familiar operations can be defined, for example

$$\textbf{enrich}\ sp\ \textbf{by sorts}\ S'\ \textbf{opns}\ Op'\ \textbf{axioms}\ Ax'$$

for a specification sp on the signature (S, Op), corresponds to $translate\ sp\ by\ \iota\ in\ sp'$, where $sp' = models((S \cup S', Op \cup Op'), Ax')$ and ι is the embedding of (S, Op) into $(S \cup S', Op \cup Op')$.

Let us consider again the example of natural numbers from the previous section. The specification **import** Nat **along** ϵ can now be more formally expressed by $\beta^{-1}(models(Nat))$; but also the theory Nat can be translated along the simulation ϵ, as we will see in the sequel, by a theory extension Θ and hence we have an *a priori* different specification $models(\Theta(Nat))$. Obviously the latter possibility is easier to deal with, because it is a basic specification, so that, for example, deductive tools can be used for rapid prototyping.

Thus it would be interesting to have conditions guaranteeing that $\bar{\beta}^{-1}(models(Nat)) = models(\Theta(Nat))$; for this aim we can play with the choice of theory and specification extensions. Let us consider as specification extension the maximal one; for every logical simulation $\mu = (\Phi, \alpha, \beta)$, and hence in particular for ϵ, there are two standard possibilities for the definition of the theory extension:

- ◇ any theory (Σ, Γ) in the domain is translated by the *complete theory extension* $\Psi^\bullet$ into the theory $(\Phi(\Sigma), (\alpha(\Gamma) \cup th(\mu, \Sigma))^\bullet)$ in the codomain, where $th(\mu, \Sigma)$ is a set of sentences defining the domain of μ and $\_^\bullet$ denotes the validity closure, i.e. $\Delta^\bullet = \{\epsilon \mid A \models \epsilon \text{ for all } A \models \Delta\}$;
- ◇ any theory (Σ, Γ) in the domain is translated by the *trivial theory extension* Ψ into the theory $(\Phi(\Sigma), \alpha(\Gamma))$ in the codomain (this choice is available for non-logical simulations as well).

With the trivial theory extension $models(\Psi(Nat))$ has many non-total models, and hence $\bar{\beta}^{-1}(models(Nat)) \neq models(\Psi(Nat))$, while it is easy to check that with the complete theory extension $\bar{\beta}^{-1}(models(Nat)) = models(\Psi^\bullet(Nat))$.

In general it can be proved that this property holds for any logical simulation, i.e. that given a logical simulation $\mu = (\Phi, \alpha, \beta)$, for every theory th in the domain of μ translating specifications along the maximal specification extension and theories along the complete theory extension $models(th) = \gamma(models(\Psi^\bullet(th)))$.

This last property can be seen as an instantiation of a general property, quite reminiscent of the homomorphism condition, that can be rephrased for every institution independent operation (see Def. A.5); for example for the $+$ operation it becomes $\beta(sp + sp') = \beta(sp) + \beta(sp')$, for the *translate* it becomes $\beta(translate\ sp\ by\ \Phi(\sigma)\ in\ sp') = translate\ \beta(sp)\ by\ \sigma\ in\ \beta(sp')$ and so on.

If, for an institution independent operation *op*, this generalized homomorphism condition is satisfied by every (logical) simulation, then we call *op (logical) simulation independent*. In our opinion simulation independent operations are the crucial point in building a specification language allowing the basic modules to be defined in different frameworks, as they guarantee that the composition of specifications is well behaving w.r.t. their translation between frameworks. In particular this condition suffices for the interpretation of every (closed) term of the specification language in the domain of the simulation to be translated in the interpretation of the same term in the codomain, so that structured proofs are preserved, the expressive power of the language is unaffected, and the import mechanism can be seen as an easier way for defining specifications. Thus if the language enjoys the property that every closed term can be reduced to a basic specification, i.e. to an axiomatic description, (as many languages do), then simulation independency guarantees that the imported specification can be reduced to basic specifications as well.

It can be directly checked that with the complete theory extension and the maximal specification extension the *models*, *translate _ by _ in _* and *derive _ by _* operations are logical simulation independent.

It is worth noting that the _+_ operation is not logical simulation independent w.r.t. this choice of extensions. Indeed a model A may exists in $\gamma(sp) + \gamma(sp')$ for which a $B \in sp$ and a $B' \in sp'$ exist s.t. $\beta(B) = A = \beta(B')$, but both $B \notin sp$ and $B' \notin sp'$ and more in general there does not exist a $C \in sp + sp'$ s.t. $\beta(C) = A$, so that $A \notin \gamma(sp + sp')$. As the _+_ operation is very useful for the modular approach to the specification of complex data types, it can be interesting to change the notion of specification extension, by allowing only specifications closed w.r.t. the simulation, i.e. those sp s.t. $A \in sp$ and $\beta(A) = \beta(B)$ imply $B \in sp$, to be mapped. In the sequel such a specification extension will be called *closed* and denoted by $\gamma^\bullet$. With this definition the _+_ operation becomes logical simulation independent, and so are the *models* and the *translate _ by _ in _*; but the *derive _ by _* is not, as in general it does not preserve the closure w.r.t. simulation, so that the result of *derive sp by* $\mu(\sigma)$ cannot be translated, although sp can be and hence the homomorphism condition is lost.

4 Simulations and the third dimension of implementation

In the literature a concept of implementation is largely used in different contexts, which is based on the idea of *refinement*. So a specification sp_2 implements a specifications sp_1, denoted by $sp_1 \rightsquigarrow sp_2$, iff in sp_2 more details have been fixed and hence sp_2 has less models than sp_1, i.e. $sp_2 \subseteq sp_1$ (see e.g. [15, 17]); this concept may also be extended to functions on specifications, by saying that a function f *implements* a function g iff for every possible argument sp we have $g(sp) \rightsquigarrow f(sp)$, and hence to parameterized specifications, viewed as denotation of functions. For an introductory exposition of the subject see e.g. section 8.1 of [17].

A relevant result (see fact. 8.1.1 of [17]) is the double composability law of implementation for specification-building operations monotonic w.r.t. the set-inclusion: in the "vertical" sense we have that if $sp \rightsquigarrow sp_1$ and $sp_1 \rightsquigarrow sp_2$, then $sp \rightsquigarrow sp_2$, by the transitivity of the subset relation, while in the "horizontal" sense we have that if a parameterized specification p_2 implements another parameterized specification p_1 of the same type, and if an actual parameter sp for p_1 is implemented by a specification sp_1, then $p_1(sp) \rightsquigarrow p_2(sp_1)$.

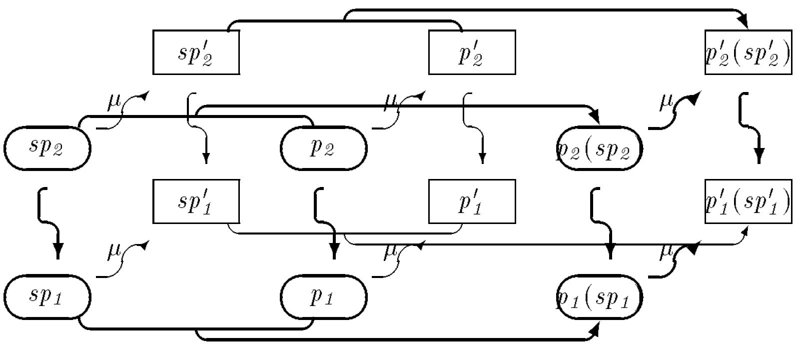
Since every specification extension is monotonic w.r.t. the set inclusion for all simulations μ, the implementation relation is translated by simulation from the old to the new frame, whatever notion of specification extension is chosen. Moreover if closed specification extension is chosen, then it is also preserved in the opposite direction; in this case the restriction of the old implementation relation to the domain of $\gamma^\bullet$ coincides with the new implementation relation (see the appendix for the formal statement).

Using simulations the concept of implementation is generalized, involving models in two institutions: given a simulation $\mu\colon \mathcal{I} \to \mathcal{I}'$ and two specification $sp \in Spec_{\mathcal{I}}(\Sigma)$ and $sp' \in Spec_{\mathcal{I}'}(\mu(\Sigma))$, we say that sp is *μ-implemented* by sp', denoted by $sp \overset{\mu}{\rightsquigarrow} sp'$, iff it is implemented in the standard sense by the translation of sp' along μ. This generalizes in the obvious way to functions and parameterized specifications (see Def. A.9).

Note that in the particular case that $\mathcal{I} = \mathcal{I}'$ and μ is the identity, $\overset{\mu}{\rightsquigarrow}$ coincides with $\rightsquigarrow$ and hence every result for $\overset{\mu}{\rightsquigarrow}$ applies also to $\rightsquigarrow$.

The vertical and horizontal composability for the usual implementation relation can be generalized to deal with simulations, in the sense that the implementation along the composition of two simulations is the composition of the implementation along the two given simulations and that all monotonic functions satisfy the "horizontal" composition w.r.t. the relation $\overset{\mu}{\rightsquigarrow}$, too.

Thus a more suggestive diagram than the usual one can be proposed, where every path is an implementation (possibly via simulation) arrow and three dimensions are present: horizontally and vertically moving within an institution, while along the third dimension different institutions are connected.



It is worth noting the difference between this approach and the one in [2]. Indeed here implementation is defined as a relation between specifications in different institutions, while in [2] an institution is proposed whose sentences represent the implementation inside a basic institution.

Conclusions

We have presented a preliminary attempt to state the foundation for specification metalanguages allowing their specification to be defined in different formalisms. In particular it has been shown how, using simulation in order to translate modules from one framework into another one, specifications can be declared "external" by a language and imported from a different paradigm. The issue becomes more complex if we want to preserve the structure of the imported specification; indeed, for the problem to be meaningful, the importing and exporting languages have to be instances of one metalanguage and the extension of simulations to the language metasorts have to be carefully tailored to fit the operations. Although it is not difficult to find extensions working for a significant subset of the most common specification building operations, a completely satisfactory choice is missing and this point is still under investigation.

Acknowledgments. We wish to thank Joseph Goguen and Josè Meseguer for their patient attention, encouragement and useful criticism.

References

[1] E. Astesiano and M. Cerioli. Relationships between logical frames. In *Recent Trends in Data Type Specification*, number 655 in Lecture Notes in Computer Science, pages 126–143, Berlin, 1993. Springer Verlag.

[2] C. Beierle and A. Voss. Viewing implementations as an institution. In D. Pitt, A. Poigné, and D. Rydeheard, editors, *Proceedings of Category Theory and Computer Science*, number 283 in Lecture Notes in Computer Science, pages 196–218, Berlin, 1987. Springer Verlag.

[3] R. Burstall and J. A. Goguen. Putting theories together to make specifications. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 1045–1058, Cambridge, 1977.

[4] R. Burstall and J. A. Goguen. The semantics of clear, a specification language. In D. Bjørner, editor, *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, number 86 in Lecture Notes in Computer Science, pages 292–332, Berlin, 1980. Springer Verlag.

[5] M. Cerioli. *Relationships between Logical Formalisms.* PhD thesis, Universities of Pisa, Genova and Udine, 1993.

[6] J. Goguen and R. Burstall. Introducing institutions. In E. Clarke and D. Kozen, editors, *Logics of Programs Workshop*, number 164 in Lecture Notes in Computer Science, pages 221–256, Berlin, 1984. Springer Verlag.

[7] J. Goguen, J. Thatcher, and Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, pages 80–149. Prentice-Hall, 1976.

[8] J. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Lab., SRI International, 1988.

[9] C. Hoare. Proof of correctness of data representation. *Acta Informatica*, 1:271–281, 1972.

[10] C. Jones. *Systematic Software Development using VDM.* Prentice Hall International, 1990.

[11] V. Manca, A. Salibra, and G. Scollo. Equational type logic. *Theoretical Computer Science*, 77:131–159, 1990. Special Issue dedicated to AMAST'89.

[12] M. Navarro, P. Nivela, F. Orejas, and A. Sanchez. On translating partial to total specifications with applications to theorem proving for partial specifications. Technical Report LSI-89-21, Universitat Politecnica de Catalunya, 1990.

[13] D. Parnas. A technique for software module specification. *Communications of A.C.M.*, 15, 1972.

[14] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.

[15] D. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation. In M. Karpinski, editor, *International Conference on Foundations of Computation*, number 158 in Lecture Notes in Computer Science, pages 413–427, Berlin, 1983. Springer Verlag.

[16] J. Spivey. *The Z Notation: a Reference Manual.* Prentice-Hall, New-York, 1989.

[17] M. Wirsing. Algebraic specification. In *Handbook of Theoretical Computer Science.* North Holland, 1990.

A Formal Foundations

In this appendix we collect the formal definitions of the concepts and the rigorous statements of results informally presented in the paper.

A.1 Institutions and simulations

We recall first the notion of institution.

Def. A.1 [[6] def.14] An *institution* $\mathcal{I}$ consists of a category **Sign** of *signatures*, a functor Sen: **Sign** $\rightarrow$ **Set** giving the set of *sentences* over a given signature, a functor Mod: **Sign** $\rightarrow$ $\mathbf{Cat}^{\mathbf{op}}$ giving the category (sometimes called the variety) of *models* of a given signature (the arrows in $Mod(\Sigma)$ are called the *model morphisms*) and a satisfaction relation[1]

$$\models \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$$

for each Σ in **Sign**, sometimes denoted $\models_\Sigma$, such that for each morphism $\phi: \Sigma_1 \rightarrow \Sigma_2$ in **Sign**, the *Satisfaction Condition*

$$M' \models Sen(\phi)(\xi) \iff Mod(\phi)(M') \models \xi$$

holds for each M' in $|Mod(\Sigma_2)|$ and each ξ in $Sen(\Sigma_1)$. $\square$

Since models are partially mapped, the usual notion of natural transformation is insufficient to describe the translation of the (*old*) model functor and we have to explicitly deal with the *partiality* of each component of this "partially"-natural transformation.

Def. A.2 Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and $\mathcal{I}' = (\mathbf{Sign}', Sen', Mod', \models')$ be institutions. Then a *simulation* $\mu: \mathcal{I} \rightarrow \mathcal{I}'$ consists of

- ◇ a functor $\Phi: \mathbf{Sign} \rightarrow \mathbf{Sign}'$;
- ◇ a natural transformation $\alpha: Sen \rightarrow Sen' \cdot \Phi$,
- ◇ a surjective *partially-natural* transformation $\beta: Mod' \cdot \Phi \rightarrow Mod$, that is a family of functors $\beta_\Sigma: dom(\mu)_\Sigma \rightarrow Mod(\Sigma)$, where $dom(\mu)_\Sigma$ is a (non-necessarily full) subcategory of $Mod'(\Phi(\Sigma))$, s.t. β_Σ is surjective on $|Mod(\Sigma)|$ and the family is partially-natural, i.e. for any signature morphism $\sigma \in \mathbf{Sign}(\Sigma_1, \Sigma_2)$
$$Mod'(\Phi(\sigma))(dom(\beta)_{\Sigma_2}) \subseteq dom(\beta)_{\Sigma_1}$$
and $Mod(\sigma) \cdot \beta_{\Sigma_2}$ is the restriction of $\beta_{\Sigma_1} \cdot Mod'(\Phi(\sigma))$ to $dom(\mu)_{\Sigma_2}$;

s.t. the following *satisfaction condition* holds:

$$A \models \alpha_\Sigma(\xi) \iff \beta_\Sigma(A) \models \xi$$

for all $A \in |dom(\mu)_\Sigma|$ and all $\xi \in Sen(\Sigma)$.

If for every $\Sigma \in |\mathbf{Sign}|$ a set $th(\Sigma, \mu)$ of $\mathcal{I}'$-sentences on $\Phi(\Sigma)$ exists s.t. the class of algebras satisfying $th(\Sigma, \mu)$ is the object class of $dom(\mu)_\Sigma$ and $dom(\mu)_\Sigma$ is a full sub-category of $Mod'(\Phi(\sigma))$, then μ is called *logical*. $\square$

It is easy to check that $\mu^\sharp: \mathcal{PAR} \rightarrow \mathcal{TL}$, whose components were informally sketched in example 2.1, is a simulation, that is logical if open formulas are considered in $\mathcal{TL}$.

---

[1] for any category **C** we denote by $|\mathbf{C}|$ the class of the objects of **C**.

A.2 Simulating Specifications

Def. A.3 Let $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ be an institution. The *specification functor*, $Spec_{\mathcal{I}}: \mathbf{Sign} \to \mathbf{Cat}^{\mathbf{Op}}$ is the composition of Mod with the power functor, i.e.

- $\diamond$ $Spec_{\mathcal{I}}(\Sigma)$ is the partially ordered category w.r.t. the class inclusion having as objects $\wp(|Mod(\Sigma)|)$, for all $\Sigma \in |\mathbf{Sign}|$;
- $\diamond$ $Spec_{\mathcal{I}}(\sigma)(sp) = \{Mod(\sigma)(A) \mid A \in sp\}$ for all $\sigma \in \mathbf{Sign}(\Sigma_1, \Sigma_2)$ and all $sp \in Spec_{\mathcal{I}}(\Sigma_2)$. □

Def. A.4 Let $\mu: \mathcal{I} \to \mathcal{I}'$ be a simulation. A *specification extension* of μ is any partially-natural transformation $\alpha: Spec_{\mathcal{I}'} \cdot \mu \to Spec_{\mathcal{I}}$ s.t. if $\alpha(sp)$ is defined then $sp \subseteq dom(\beta)$ and $\alpha(sp) = \beta_\Sigma(sp)$.

The *maximal specification extension* γ of μ is defined by:
if $sp \subseteq dom(\beta)$, then $\gamma_\Sigma(sp) = \beta_\Sigma(sp)$, else $\gamma_\Sigma(sp)$ is undefined. If no ambiguity arises γ_Σ will be denoted by γ or, simply, by μ. □

While it is clear what a metalanguage based on a categorical and set-theoretical language is, the only way to define it completely formally seems to be explicitly enumerating which sorts and operations are allowed; thus we limit ourselves to a semi-formal level and propose a scheme of construction of a generic metalanguage using as paradigmatic examples the operations of [14].

A.2.1 A building scheme for an institution independent language.

Let X be a set of variables[2], which will be evaluated in $|\mathbf{Sign}|$ for all institutions $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$.

- $\diamond$ Starting from the elements of X a set MS of *metasorts* is built, only using categorical and set-theoretical concepts; for example for any $\Sigma_1, \Sigma_2 \in X$ consider the metasort $Sign(\Sigma_1, \Sigma_2)$ of signature morphisms from Σ_1 into Σ_1.
- $\diamond$ A set MF of metaoperations of arity in MS is built, only using categorical and set-theoretical concepts; for example for any $\Sigma \in X$ consider the metaoperation $models: \wp \cdot Sen(\Sigma) \to Spec(\Sigma)$, associating with any set of sentences the class of its models.
- $\diamond$ Let us fix an institution $\mathcal{I} = (\mathbf{Sign}, Sen, Mod, \models)$ and a valuation $V: X \to |\mathbf{Sign}|$; with each symbol of MS and each operation symbol in MF the corresponding standard interpretation is associated; for example $(Sign(\Sigma_1, \Sigma_2))^{\mathcal{I},V} = \mathbf{Sign}(V(\Sigma_1), V(\Sigma_2))$ and $(s_i^{\mathcal{I},V}: \wp(Sen(V(\Sigma))) \to Spec_{\mathcal{I}}(V(\Sigma))$ on Ax yields $\{A \mid A \models \alpha,$ for all $\alpha \in Ax\}$. □

For each new sort we need to know the way it has to be translated w.r.t. a generic simulation μ, as we have seen in the case of specifications, where specification extensions had been defined to translate them. Analogously to the definition of the metalanguage, in order to define the new components of simulations we start from symbols to denote the components dealing with signatures, sentences and models, which will be evaluated to the components of the simulation, and use them to formally define an extension of a generic

[2] Since in any significant example we have seen, the language is based only on metavariables of sort *signatures*, we use a set of variables for sake of simplicity; but there are no problems using a family of variable sets indexed on the basic elements of institutions.

simulation by means of the categorical and set-theoretic metalanguage. The only requirement made for the choice of the extensions is that the composition of the (chosen) extensions of two composable simulations is the extension of the composition itself.

In the sequel both the symbol for the extension of metasort s of a simulation and its evaluation on a concrete simulation will be denoted by μ^s.

A.2.2 Simulating Structured Specifications

With the help of a metalanguage, institutions are provided of algebraic structure; thus we are looking for conditions guaranteeing that the simulation extensions are behaving like homomorphisms of this new structure. As the extensions of a simulation for the derived metasorts can be partial or countervariant, the standard formulation of many-sorted homomorphisms, $h_s(op^{\mathcal{I}}(a_1,\ldots,a_n)) = op^{\mathcal{I}'}(h_{s_1}(a_1),\ldots,h_{s_n}(a_n))$, has to be worked out carefully. It is easier starting from the following equivalent formulation of the homomorphism condition:

$$a'_1 = h_{s_1}(a_1) \wedge \ldots \wedge a'_n = h_{s_n}(a_n) \supset h_s(op^{\mathcal{I}}(a_1,\ldots,a_n)) = op^{\mathcal{I}'}(a'_1,\ldots,a'_n);$$

indeed it is sufficient to modify the premises, allowing that, depending on the co/counter-variance of the s component of the homomorphism, either $a'_i = h_{s_i}(a_i)$, or $a_i = h_{s_i}(a'_i)$ are used.

Def. A.5 Let $\mu{:}\mathcal{I} \rightarrow \mathcal{I}'$ be a simulation and $L = (MS, MF)$ be an institution independent metalanguage on variables X. Then μ is an *L-homomorphism* iff a_i related by μ to a'_i implies $op^{\mathcal{I},V}(a_1,\ldots,a_n)$ related by μ to $op^{\mathcal{I}',\mu\cdot V}(a'_1,\ldots,a'_n)$, where two elements a and a' of the same metasort s are related by μ iff one of them is the image of the other one along the extension μ^s, for all valuations $V{:}X \rightarrow |\mathbf{Sign}|$ and all $op \in MF_{s_1\ldots s_n,s}$.

Let L be an institution independent metalanguage and M be a class of simulations. Then L is *M-independent* iff μ is an L-homomorphism for each $\mu \in M$ and it is *(logical) simulation independent* iff μ is an L-homomorphism for all (logical) simulations μ. □

It is easy to check that L-homomorphisms are well behaving w.r.t. the composition of simulations, the union and the operational closure of languages.

A.3 Implementation and Simulation

Def. A.6

- ◇ A specification sp_2 *implements* a specifications sp_1, denoted by $sp_1 \rightsquigarrow sp_2$, iff both sp_1 and sp_2 are on the same signature and $sp_2 \subseteq sp_1$.
- ◇ Let $f, g{:}\, Spec_{\mathcal{I}} \rightarrow Spec_{\mathcal{I}}$ be functions on specifications; then f *implements* g, denoted by $g \rightsquigarrow f$ iff $g(sp) \rightsquigarrow f(sp)$ for all specifications sp.
- ◇ Let $p_1 = \lambda X : \Sigma_{Par}.sp_1(\_)$ and $p_2 = \lambda X : \Sigma_{Par}.sp_2(\_)$ be terms of the same sort on some (institution independent) metalanguage; then p_2 *implements* p_1, denoted by $p_1 \rightsquigarrow p_2$, iff $sp_1[sp] \rightsquigarrow sp_2[sp]$ for all specifications sp. □

Prop. A.7 Let sp, sp_1 and sp_2 be specifications and p_1, p_2 be parameterized specifications in a metalanguage whose specification-building operations are monotonic w.r.t. the set-inclusion.

1. If $sp \rightsquigarrow sp_1$ and $sp_1 \rightsquigarrow sp_2$, then $sp \rightsquigarrow sp_2$;
2. If $sp \rightsquigarrow sp_1$, $p_1 \rightsquigarrow p_2$ and sp is an actual parameter of p_1 (i.e. $p_1(sp)$ is defined), then $p_1(sp) \rightsquigarrow p_2(sp_1)$.

Proof. See fact. 8.1.1 of [17]. $\square$

Prop. A.8 Let $\mu: \mathcal{I} \to \mathcal{I}'$ be a simulation, sp'_1 and sp'_2 belong to $Spec_{\mathcal{I}'}(\mu(\Sigma))$ s.t. both $sp'_1, sp'_2 \subseteq dom(\beta)$; if $sp'_1 \rightsquigarrow sp'_2$, then $\gamma(sp'_1) \rightsquigarrow \gamma(sp'_2)$. Moreover if $\gamma^\bullet(sp'_1)$ and $\gamma^\bullet(sp'_2)$ are defined, then $sp'_1 \rightsquigarrow sp'_2$ iff $\gamma^\bullet(sp'_1) \rightsquigarrow \gamma^\bullet(sp'_2)$.

Proof. If $sp'_1 \rightsquigarrow sp'_2$, then $sp'_2 \subseteq sp'_1$ and hence $\gamma(sp'_1) \rightsquigarrow \gamma(sp'_2)$, as

$$\gamma(sp'_2) = \beta(sp'_2) \subseteq \beta(sp'_1) = \gamma(sp'_1).$$

Let us assume that $\gamma^\bullet(sp'_1)$ and $\gamma^\bullet(sp'_2)$ are defined; then, analogously to the previous point, $sp'_1 \rightsquigarrow sp'_2$ implies $\gamma^\bullet(sp'_1) \rightsquigarrow \gamma^\bullet(sp'_2)$. Vice versa if $\gamma^\bullet(sp'_1) \rightsquigarrow \gamma^\bullet(sp'_2)$, then $\beta^{-1}(\gamma^\bullet(sp'_2)) \subseteq \beta^{-1}(\gamma^\bullet(sp'_1))$, i.e., by the condition of definedness of $\gamma^\bullet$, $sp'_1 \rightsquigarrow sp'_2$, as

$$sp'_2 = \beta^{-1}(\gamma^\bullet(sp'_2)) \subseteq \beta^{-1}(\gamma^\bullet(sp'_1)) = sp'_1. \square$$

Def. A.9 Let $\mu: \mathcal{I} \to \mathcal{I}'$ be a simulation, $sp \in Spec_{\mathcal{I}}(\Sigma)$, $sp' \in Spec_{\mathcal{I}'}(\mu(\Sigma))$; then sp is *μ-implemented* by sp', denoted by $sp \overset{\mu}{\rightsquigarrow} sp'$, iff $sp' \subseteq dom(\beta)$ and $\beta(sp') \subseteq sp$, i.e. iff $\gamma(sp')$ is defined and $sp \rightsquigarrow \gamma(sp')$.

Let $f: Spec_{\mathcal{I}}(\Sigma_1) \to Spec_{\mathcal{I}}(\Sigma_2)$ and $f': Spec_{\mathcal{I}'}(\mu(\Sigma_1)) \to Spec_{\mathcal{I}'}(\mu(\Sigma_2))$ be functions; then f is *μ-implemented* by f', denoted by $f \overset{\mu}{\rightsquigarrow} f'$, iff $f(sp) \overset{\mu}{\rightsquigarrow} f'(\beta^{-1}(sp))$ for all $sp \in Spec_{\mathcal{I}}(\Sigma)$.

Let $p_1 = \lambda X : \Sigma_{Par}.sp_1(\_)$ and $p_2 = \lambda X : \Sigma_{Par}.sp_2(\_)$ be terms of the same sort on some institution independent metalanguage; then p_1 is *μ-implemented* by p_2, denoted by $p_1 \overset{\mu}{\rightsquigarrow} p_2$, iff ${p_1}^{\mathcal{I},V} \overset{\mu}{\rightsquigarrow} {p_2}^{\mathcal{I}',\mu \cdot V}$ for all valuations V for the free variables of p_1 and p_2 in $\mathcal{I}$. $\square$

Prop. A.10 Let $\mathcal{I}$, $\mathcal{I}'$ and $\mathcal{I}''$ be institutions, $\mu: \mathcal{I} \to \mathcal{I}'$ and $\nu: \mathcal{I}' \to \mathcal{I}''$ be simulations. The following conditions hold:

1. $sp \overset{\mu}{\rightsquigarrow} sp'$ and $sp' \overset{\nu}{\rightsquigarrow} sp''$ implies $sp \overset{\nu \cdot \mu}{\rightsquigarrow} sp''$ for all $sp \in Spec_{\mathcal{I}}(\Sigma)$, all $sp' \in Spec_{\mathcal{I}'}(\mu(\Sigma))$ and all $sp'' \in Spec_{\mathcal{I}''}(\nu(\mu(\Sigma)))$.
2. $sp \overset{\mu}{\rightsquigarrow} sp'$ and $f \overset{\mu}{\rightsquigarrow} f'$ implies $f(sp) \overset{\mu}{\rightsquigarrow} f'(sp')$ for all $sp \in Spec_{\mathcal{I}}(\Sigma_1)$, all $sp' \in Spec_{\mathcal{I}'}(\mu(\Sigma_1))$ all monotonic $f: Spec_{\mathcal{I}}(\Sigma_1) \to Spec_{\mathcal{I}}(\Sigma_2)$ and $f': Spec_{\mathcal{I}'}(\mu(\Sigma_1)) \to Spec_{\mathcal{I}'}(\mu(\Sigma_2))$.

Proof.

1. Since $sp' \subseteq dom(\beta)$ and $\beta'(sp'') \subseteq sp'$, $sp'' \subseteq dom(\nu \cdot \mu)$. Moreover if $sp' \overset{\nu}{\rightsquigarrow} sp''$, then $\beta'_{\Phi(\Sigma)}(sp'') \subseteq sp'$, and if $sp \overset{\mu}{\rightsquigarrow} sp'$, then $\beta_\Sigma(sp') \subseteq sp$, so that $\beta_\Sigma(\beta'_{\Phi(\Sigma)}(sp'')) \subseteq sp$, i.e. $sp \overset{\nu \cdot \mu}{\rightsquigarrow} sp''$.
2. By definition of $\overset{\mu}{\rightsquigarrow}$, it is sufficient to show that $\beta_\Sigma(f'(sp')) \subseteq f(sp)$. By definition of $\overset{\mu}{\rightsquigarrow}$, $f \overset{\mu}{\rightsquigarrow} f'$ implies that $f'({\beta_\Sigma}^{-1}(sp_1)) \overset{\mu}{\rightsquigarrow} f(sp_1)$, so that $\beta_\Sigma(f'({\beta_\Sigma}^{-1}(sp_1)) \subseteq f(sp_1)$ for all $sp_1 \in Spec_{\mathcal{I}}(\Sigma_1)$; thus, for $sp_1 = \beta_\Sigma(sp')$, $\beta_\Sigma(f'({\beta_\Sigma}^{-1}(\beta_\Sigma(sp'))) \subseteq f(\beta_\Sigma(sp'))$. Since $sp \overset{\mu}{\rightsquigarrow} sp'$, $sp' \subseteq dom(\beta)$ and hence $sp' \subseteq {\beta_\Sigma}^{-1}(\beta_\Sigma(sp'))$, so that, as f' is monotonic, $f'(sp') \subseteq f'({\beta_\Sigma}^{-1}(\beta_\Sigma(sp')))$. Finally from $sp \overset{\mu}{\rightsquigarrow} sp'$, i.e. $\beta_\Sigma(sp') \subseteq sp$, and from the monotony of f, the thesis follows. $\square$