

# Hierarchical Heterogeneous Specifications

Sophie Coudert<sup>1</sup>, Gilles Bernot<sup>2</sup>, and Pascale Le Gall<sup>2</sup> \*

<sup>1</sup> I.R.I.N., Université de Nantes, Rue de la Houssinière, 44332 Nantes, France

`Sophie.Coudert@irin.univ-nantes.fr`

<sup>2</sup> L.a.M.I., Université d'Évry, Cours Monseigneur Roméro, 91025 Évry, France

`{bernot,legall}@lami.univ-evry.fr`

**Abstract.** We propose a definition of hierarchical heterogeneous formal specifications, where each module is specified according to its own homogeneous logic. We focus on the specification structure which we represent by a term in order to take benefit of classical knowledge on terms. For example, substitutions solve implementation sharing of modules. Then, we show how proof mechanisms can be expressed inside our framework. Our proof system involves both the homogeneous inference relations associated to the logics of modules and property inheritance relations associated to the structuring primitives. Heterogeneous primitives allow to move from one logic to another. We sketch out the specification of a travel agency given according to our particular framework of structured specifications. We demonstrate on this specification how a heterogeneous proof can be handled.

**Keywords:** formal specification, structured specification, proof theory, heterogeneous specification, heterogeneous proof, logical framework, inference system, modularity, algebraic specification.

## Introduction

Today, a number of formal foundations and tools exist to treat specification modules independently, or hierarchical specifications: formal languages, theorem provers, test generation tools, etc. Moreover the *structure of the specifications* becomes a privileged object of study, as in the software architecture approaches [PW92]. The CoFI initiative for example [CoF96], which is defining a common language core for algebraic specifications, gives a large place to structuring issues [BST99]. Also several studies focus on the structuring primitives of algebraic specifications [Wir93,HST94,DGS93]. Bringing into operations formal methods and specifications on complete industrial projects requires reusability issues. In order to *reuse* various already implemented components, we should take them *as they are*, with their formalisms. The question addressed by this paper is precisely how to combine such heterogeneous components.

---

\* This work was partly supported by the ESPRIT-IV Working Group 22704 ASPIRE, the ESPRIT-IV Working Group 23531 FIREworks and the French “PRC-GDR de programmation.”

We propose a general framework allowing to take into account both classical structuring primitives ([Wir93,HST94,DGS93]) and special translation primitives that introduce heterogeneity in specifications. We deal with structuring primitives, *uniformly* according to the syntactic, semantic and proof considerations. For this, we abstract them by the notion of *constructor* and we look for characterizing minimal requirements allowing us to combine them into a hierarchical heterogeneous specification (HHS in the sequel) in a coherent way.

From a syntactic viewpoint, we represent structured specifications as terms with specification modules as function symbols (constructors). For example, we denote the enrichment of  $n$  specifications  $SP_1, \dots, SP_n$  with a module  $\Delta P$  by the term  $use_{\Delta P}(SP_1, \dots, SP_n)$ . There are as many different *use* constructors as different enrichment modules  $\Delta P$ . All other primitives, such as *forget*, *rename*, as well as the translation primitive *het*, are constructors. Moreover, each constructor is provided with an application domain giving which specification terms it may import. Then, specifications are simply terms with respect the application domains. Equal subterms of such a specification represent a unique sub-specification.

From a semantic viewpoint, the models of a specification are defined on its exported signature so that they belong to the top level constructor logic. Following previous works on modularity issues ([Bid87,NOS95]), constructor semantics are sets of functions (*implementations*) associating to the imported models a model over the exported signature. The ability to manage multiple importations from different logics, as well as the sharing aspects are solved by using semantic substitution mechanisms.

From a proof-theoretical viewpoint, we associate to each module an inference mechanism especially devoted to capture the rôle of the constructor w.r.t. to the transmitted properties from the imported specifications to the global specification. We then formalize the proof principle which consists in *delegating lemmas* along the specification structure ([HST94] uses the term “*diving*”) in such a way that pieces of homogeneous proofs can be reused.

The main contribution of our HHS framework is that the syntax, semantic and proof-theoretical parts of a specification can be systematically derived from those of the constructors (structuring primitives) occurring in it. This modelling easily admits heterogeneous components within a specification. We illustrate our purpose with a simplified HHS of a travel agency and a significant heterogeneous proof about it. The specification combines about a dozen modules and 3 formalisms, and heterogeneous components are glued together using heterogeneous bridges.

## 1 Homogeneous Logics

### 1.1 Definition

As usual, system components are specified by means of a set of sentences over their signature, according to their formalism. Homogeneous formalisms are simi-

lar to general logics [Mes89], but our requirements are weaker<sup>1</sup> because they are minimal with respect to our needs for HHS.

**Definition 1.** A homogeneous logic  $l$  is a tuple  $(Sig_l, sen_l, mod_l, \vdash_l, \models_l)$  where:

- $Sig_l$  is a class whose objects are called signatures
- $sen_l$  is a map from  $Sig_l$  to Set associating to each signature a set of sentences (Set being the class of sets)
- $mod_l$  is a map from  $Sig_l$  to Class associating to each signature a class of models (Class being the class of classes)
- $\vdash_l$ , called inference relation, is a  $Sig_l$ -indexed family such that for each signature  $\Sigma$ ,  $\vdash_{l,\Sigma}$  is a binary relation included in  $\mathcal{P}(sen_l(\Sigma)) \times sen_l(\Sigma)$
- $\models_l$ , called satisfaction relation, is a  $Sig_l$ -indexed family such that for each signature  $\Sigma$ ,  $\models_{l,\Sigma}$  is a binary relation included in  $mod_l(\Sigma) \times sen_l(\Sigma)$

and  $\vdash_l$  is reflexive, monotonic, transitive and sound i.e., respectively<sup>2</sup>

- $\forall \Sigma \in Sig_l, \forall \varphi \in sen_l(\Sigma), \{\varphi\} \vdash_{l,\Sigma} \varphi$
- $\forall \Sigma \in Sig_l, \forall \Gamma \subseteq sen_l(\Sigma), \forall \Gamma' \subseteq sen_l(\Sigma), \forall \varphi \in sen_l(\Sigma),$   
 $\Gamma \vdash_{l,\Sigma} \varphi$  and  $\Gamma \subseteq \Gamma' \implies \Gamma' \vdash_{l,\Sigma} \varphi$
- $\forall \Sigma \in Sig_l, \forall \Gamma \subseteq sen_l(\Sigma), \forall \Gamma' \subseteq sen_l(\Sigma), \forall \varphi \in sen_l(\Sigma),$   
 $\Gamma \vdash_{b,\Sigma} \Gamma'$  and  $\Gamma \cup \Gamma' \vdash_{l,\Sigma} \varphi \implies \Gamma \vdash_{l,\Sigma} \varphi$
- for any  $\Gamma \subseteq sen_l(\Sigma)$  and any  $\varphi \in sen_l(\Sigma)$ , if  $\Gamma \vdash_{l,\Sigma} \varphi$  then

$$\forall M \in mod_l(\Sigma), (M \models_{l,\Sigma} \Gamma) \implies (M \models_{l,\Sigma} \varphi)$$

**Notation** –  $\mathcal{L}$  is the class of all homogeneous logics.

- $Sig = \coprod_{l \in \mathcal{L}} Sig_l$  (the class of all the signatures<sup>3</sup>)
- $mod : Sig \rightarrow Class$  associates  $mod_l(\Sigma)$  to each signature  $\Sigma$  in  $Sig_l$
- $sen : Sig \rightarrow Set$  associates  $sen_l(\Sigma)$  to each signature  $\Sigma$  in  $Sig_l$
- $\vdash = \coprod_{l \in \mathcal{L}} \vdash_l$  and  $\models = \coprod_{l \in \mathcal{L}} \models_l$

Our HHS example specifying a travel agency will involve three different homogeneous logics, respectively FG, OBS and ND. FG is the classical typed equational logic [EM85], finitely generated for a subset of the sorts. It allows both to use structural induction on these sorts and to import non finitely generated logics. OBS is a formalism with observational semantics [BH96] and is also a typed equational logic, not finitely generated, with set-theoretic equality for observable sorts (declared in the signature) and observational equality (based on observable contexts) for the other sorts. ND is a formalism devoted to specify non deterministic behaviours [WM95]. Non deterministic signatures are the

<sup>1</sup> and we have actually used examples of homogeneous logics which do not satisfy the satisfaction condition [BLGA94]

<sup>2</sup>  $\Gamma \vdash_{b,\Sigma} \Gamma'$  means  $\forall \varphi \in \Gamma', \Gamma \vdash_{b,\Sigma} \varphi$  and  $M \models_{b,\Sigma} \Gamma$  means  $\forall \varphi \in \Gamma, M \models_{l,\Sigma} \varphi$

<sup>3</sup> as usual,  $\coprod$  stands for the disjoint union

same as those of typed logic. The difference is that terms evaluate to *sets* of values: all the possible values of non deterministic executions. The equality ( $\doteq$ ) is strong: two terms are equal if evaluated to the same singleton. There are two other predicates: a strong difference ( $\neq$ ), which requires disjoint evaluations, and an inclusion ( $\prec$ ). Sentences are clauses and a useful remark for our purpose is that in deterministic models (where all terms are evaluated in singletons),  $\doteq$  and  $\prec$  have the same meaning, and Horn clauses can be seen as conditional positive sentences. While FG and OBS are nearby formalisms (see for example [Pad96]), ND is really a foreign formalism w.r.t. FG and OBS because of the possible multi-evaluations of terms.

## 1.2 Heterogeneous Bridges Between Logics

Our approach for formal interoperability is based on a natural idea: we use special translation modules which are new components added to the pre-existent homogeneous ones. These new components allow a module in a formalism to import a specification in another formalism. Such translations could be based on some previous works (e.g. [Mes89, AC94, Tar96]) which establish strong relationships between formalisms by means of mappings. All these mappings preserve the signature morphisms (and their semantic counterpart) which are used to model the structure. In other words these works intend to preserve the structure and all the properties through the translation so that we can retrieve them in a global flat model of the specification. Such a point of view in practice leads to translation modules which are defined from a less expressive formalism to a more expressive one (in the sense of a logical framework according to [Mes97]).

In order to define an applicable general framework, we adopt a more flexible definition of such translation modules: formalism mappings will carry models without any reference to signature morphisms, provided that the underlying translations carry some clear intuition.

### Definition 2.

Let  $a = (Sig_a, sen_a, mod_a, \vdash_a, \models_a)$  and  $b = (Sig_b, sen_b, mod_b, \vdash_b, \models_b)$  be homogeneous logics. A heterogeneous bridge  $\mu : a \rightarrow b$  consists of

- A total function  $Tr_\mu : Sig_a \rightarrow Sig_b$ , called signature transposition function
- A  $Sig_a$ -indexed family  $Ext_\mu$  such that  $Ext_{\mu, \Sigma} : mod_a(\Sigma) \rightarrow mod_b(Tr_\mu(\Sigma))$  are partial functions called model extraction functions
- A family  $\Vdash_\mu$  indexed by  $Sig_a$ , such that  $\Vdash_{\mu, \Sigma}$  is a binary relation included in  $\mathcal{P}(sen_a(\Sigma)) \times sen_b(Tr_\mu(\Sigma))$ , called heterogeneous inference bridge

satisfying the following properties:

- $\Vdash_\mu$  is monotonic
- heterogeneous soundness: for any  $\Sigma \in Sig_a$ , for any  $\Gamma \subseteq sen_a(\Sigma)$ , for any  $\varphi \in sen_b(Tr_\mu(\Sigma))$ , if  $\Gamma \Vdash_{\mu, \Sigma} \varphi$  then for all  $M \in mod_a(\Sigma)$ , if  $Ext_{\mu, \Sigma}(M)$  is defined, then we have:  $[M \models_a \Gamma \implies Ext_{\mu, \Sigma}(M) \models_b \varphi]$

Homogeneous logics provided with heterogeneous bridges form a category, heterogeneous bridge composition being defined in an obvious way. Some previous works on maps between formalisms ([CM97, AC92, GB92, SS96, Mes89]) allow us to rather easily derive heterogeneous bridges (e.g. [AC94], see also [BCLG96] for a more precise discussion). Let us point out that we do not look for establishing some fine relationships between logics, but on the contrary, we look for pragmatic ones, any *ad hoc* definition of heterogeneous bridge is in fact suitable as soon as one can associate an intuitive meaning to it. For example, the partiality of  $Ext_{\mu, \Sigma}$  allows translations from a more expressive formalism to a less expressive one.

Heterogeneous bridges between the logics FG, OBS and ND are defined in an obvious way. From FG to OBS, we forget in the source signature the declaration of the finitely generated sorts and all sorts in the target signature are observable. Models and sentences are preserved, by replacing the set-theoretic equal  $=$  by the observational one  $\approx$ . So,  $\Phi_{FG} \Vdash \varphi_{OBS}$  iff  $\varphi_{OBS} \in \Phi_{FG} [\approx / =]$ . From OBS to FG, we forget the observation part of the signature, no sort is finitely generated and we quotient the source model by the observational equivalence [BH96]. Conversely, we have  $\Phi_{OBS} \Vdash \varphi_{FG}$  iff  $\varphi_{FG} \in \Phi_{OBS} [= / \approx]$ . From FG to ND, we forget the finitely generated sorts in the signature, values become singletons for the models, and we translate the sentences. The reverse bridge is exactly symmetric on *de facto* deterministic models: it allows to transmit the deterministic part of a specification after having forgotten the rest. We obtain a bridge from OBS to ND by combining the previous ones.

## 2 An Example of HSS: A Travel Agency

Our travel agency selects for its customers the operator which potentially offers the less expensive path from a city to another city. We have classical modules for booleans and positive integers (**Bool** and **Int**). A module **Map** specifies cities and direct links between them. A module **Netwk** specifies networks as sets of links. Then, a module **Path** defines paths in a network as lists of links verifying certain properties. A module **Oper** specifies two operators, each of them owning its own network. Finally, the agency is specified in the module **Agency**. Each module has its own “natural” specification language. Most of them (**Bool**, **Int**, **Map**, **Path**, **Agency**) are specified according to FG. The networks (**Netwk**) are sets, specified in OBS which is especially adequate for this. **Oper** is specified according to ND. Indeed, as in general routing problems, there can be several paths for a given travel, so that an operator proposes them in a non deterministic way.

## 3 HHS Theory

### 3.1 Syntax

We unify all the different notions of specification modules and building primitives by the notion of *specification constructor* [NOS95].

**Definition 3.** A constructor universe  $\Theta$  is a tuple  $(\mathcal{C}, \mathbf{Spec})$  where

- $\mathcal{C}$  is a class of constructors. Each constructor is provided with a profile in  $\text{Sig}^+$ . We denote such a constructor by  $f : \Sigma_1 \dots \Sigma_n \rightarrow \Sigma$  ( $n \in \mathbb{N}$ ).
- $\mathbf{Spec}$  is a class of well structured specifications which is a subclass of the free term algebra on  $\mathcal{C}$ ,  $\mathcal{T}_{\mathcal{C}}$ .  $\mathbf{Spec}$  is closed by sub-terms:
 
$$\forall f : \Sigma_1 \dots \Sigma_n \rightarrow \Sigma \in \mathcal{C}, \forall \tau_1 \in \mathcal{T}_{\mathcal{C}, \Sigma_1}, \dots, \forall \tau_n \in \mathcal{T}_{\mathcal{C}, \Sigma_n},$$

$$f(\tau_1, \dots, \tau_n) \in \mathbf{Spec} \implies \forall i = 1 \dots n, \tau_i \in \mathbf{Spec}$$

Reminder:  $\mathcal{C}$  being given, the free term algebra on  $\mathcal{C}$  is the least  $\text{Sig}$ -indexed family  $\mathcal{T}_{\mathcal{C}} = \prod_{\Sigma \in \text{Sig}} \mathcal{T}_{\mathcal{C}, \Sigma}$  such that for every  $(f : \Sigma_1, \dots, \Sigma_n \rightarrow \Sigma) \in \mathcal{C}$ , if  $\tau_1 \in \mathcal{T}_{\mathcal{C}, \Sigma_1}$  and  $\dots$  and  $\tau_n \in \mathcal{T}_{\mathcal{C}, \Sigma_n}$  ( $n \geq 0$ ), then  $f(\tau_1, \dots, \tau_n) \in \mathcal{T}_{\mathcal{C}, \Sigma}$ . Notice that  $\mathcal{T}_{\mathcal{C}}$  exists even if  $\mathcal{C}$  is a class [DD77].

By convention, a constructor  $f$  will be denoted by  $c_{text}$  where  $c$  indicates the nature of the constructor (e.g. *use* for an enrichment module) and *text* gives the specific informations of the module (e.g. the exported signature, the axioms, etc), possibly by means of an intermediate identifier (e.g.  $\Delta P$ ). A flat homogeneous presentation  $P = (\Sigma, \Gamma)$  can give rise to a constructor of arity 0,  $basic_P : \rightarrow \Sigma$ . Also, say in the first order typed equational logic, each signature morphism  $\sigma : \Sigma_1 \rightarrow \Sigma_2$  gives rise to  $forget_\sigma : \Sigma_2 \rightarrow \Sigma_1$ , with the obvious meaning of hiding operations as structuring operation. As a third example, we can consider heterogeneous constructors from a logic to another one based on the heterogeneous bridges presented earlier (similar to the deriving primitive introduced in [Tar96]). Each heterogeneous bridge  $\mu$  allows to define a family of constructors  $f = het_{\mu, \Sigma}$  provided with  $\Sigma \rightarrow Tr_\mu(\Sigma)$  as profile.

**Notation** If  $\tau \in \mathcal{T}_{\mathcal{C}, \Sigma}$ , we call  $\Sigma$  the signature of  $\tau$ . By convention, the signature of  $\tau$  is denoted by  $\Sigma_\tau$ . Moreover, we note  $\mathbf{Spec}_\Sigma = \mathbf{Spec} \cap \mathcal{T}_{\mathcal{C}, \Sigma} = \{ \tau \in \mathbf{Spec} \mid \Sigma_\tau = \Sigma \}$

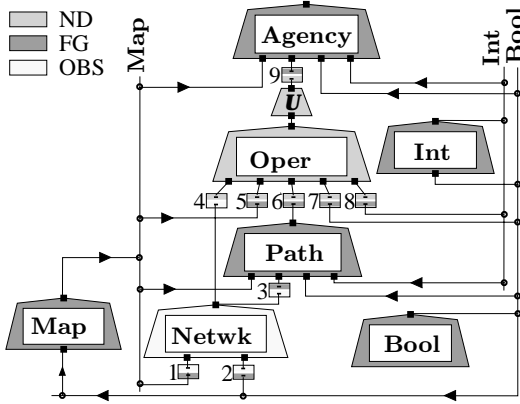
The coincidence of the signatures is a minimal requirement to connect two specification components but some specification modules are intended to be combined only with some peculiar subspecifications. For example the module *useNetwk* can only import a **Map** specification such that the provided equality is reflexive, symmetric and transitive. It is the reason why  $\mathbf{Spec}$  determines a domain for each constructor.

**Definition 4.** Given a constructor universe  $\Theta = (\mathcal{C}, \mathbf{Spec})$ , the syntactical domain of a constructor  $f : \Sigma_1 \dots \Sigma_n \rightarrow \Sigma$  is the class  $\mathcal{D}_f$  of tuples  $(\tau_1, \dots, \tau_n)$  such that  $f(\tau_1, \dots, \tau_n)$  belongs to  $\mathbf{Spec}$ .

The point is that one should be able to specify a constructor without knowing the formalisms of all its subcomponents, but only the useful properties inherited from the syntactical domain and expressed on the input profile.

We can now give all the significant syntactic elements of the travel agency example: we first give its hierarchical structure by a specification term.

The structure of the specification is built on a *basic* constructor for the **Bool** presentation which does not import anything and *use* constructors for the enrichment modules (**Int**, **Map**, **Netwk**, **Path**, **Oper**, **Agency**).



Nine heterogeneous constructors (*het*) are needed to connect these heterogeneous components. Moreover, a *forget* constructor  $U$  extracts the deterministic part of the non-deterministic specification  $\overline{\text{Oper}}$ , before applying the heterogeneous constructor numbered 9. According to our framework, this specification and its subspecifications are terms on

these constructors:

$$\overline{\text{Bool}} = \text{basic}_{\text{Bool}}, \quad \overline{\text{Int}} = \text{use}_{\text{Int}}(\overline{\text{Bool}}), \quad \overline{\text{Map}} = \text{use}_{\text{Map}}(\overline{\text{Bool}}),$$

$$\overline{\text{Netwk}} = \text{use}_{\text{Netwk}}(\text{het}_1(\overline{\text{Map}}), \text{het}_2(\overline{\text{Bool}}))$$

$$\overline{\text{Path}} = \text{use}_{\text{Path}}(\overline{\text{Map}}, \text{het}_3(\overline{\text{Netwk}}), \overline{\text{Bool}}, \overline{\text{Int}})$$

$$\overline{\text{Oper}} = \text{use}_{\text{Oper}}(\text{het}_4(\overline{\text{Netwk}}), \text{het}_5(\overline{\text{Map}}), \text{het}_6(\overline{\text{Path}}), \text{het}_7(\overline{\text{Bool}}), \text{het}_8(\overline{\text{Int}}))$$

$$\overline{\text{Agency}} = \text{Agency}(\overline{\text{Map}}, \text{het}_9(\text{forget}_U(\overline{\text{Oper}})), \overline{\text{Bool}}, \overline{\text{Int}})$$

We now make precise the contents of the main *use* constructors. **Map** specifies cities ( $\mathbf{A}, \mathbf{B}, \mathbf{C} : \rightarrow \text{city}$ ) and direct links ( $\mathbf{l}(\_, \_) : \text{city} \times \text{city} \rightarrow \text{link}$ ).

<u>Path, FG</u>
<p><b>Inputs:</b>                      1:Map, 2:Netwk, 3:Bool, 4:Int,  <b>Sorts:</b> <i>list</i>  <b>Operations:</b>                      *[_] : <i>link</i> → <i>list</i>;                      *_ :: _ : <i>link</i> × <i>list</i> → <i>list</i>;                      p() : <i>list</i> × <i>city</i> × <i>city</i> × <i>net</i> → <i>bool</i>;                      \$( ) : <i>list</i> → <i>nat</i>;  <b>Axioms:</b> ...</p>
<u>Agency, FG</u>
<p><b>Inputs:</b> 1:Map, 2:Oper, 3:Bool, 4:Int  <b>Operations:</b> sl() : <i>city</i> × <i>city</i> → <i>oper</i>;  <b>Axioms:</b>                      1: <math>\widehat{\mathbf{O}_2, c, c'} &gt; \widehat{\mathbf{O}_1, c, c'} \Leftrightarrow \text{sl}(c, c') = \mathbf{O}_1</math></p>

**Netwk** specifies sets of links (sort *net*, with  $\emptyset : \rightarrow \text{net}$  as empty-set and  $\_ \diamond \_ : \text{link} \times \text{net} \rightarrow \text{net}$  as insertion). These sets are observable through membership ( $\_ \in \_ : \text{link} \times \text{net} \rightarrow \text{bool}$ ): booleans are observable. Paths in a network are non-empty lists of consecutive links:  $\mathbf{p}(p, c, c', n)$  means that  $p$  is a path from  $c$  to  $c'$  in the network  $n$ ; their cost ( $\$( )$ ) is their length. We only give the signature of the module **Path**. Conventionally, a star before a symbol points a generator. The **Oper** and **Agency** modules are relevant for

the part of the proof we consider later on. For readability, we have simplified some notations; for example, boolean functions are written like predicates (omitting the “=  $\mathbf{T}$ ”), we abbreviate by  $\Leftrightarrow$  the corresponding set of conditional positive axioms *between* boolean values, etc.

<u>Oper, ND</u>
<b>Inputs:</b>
1: <b>Netwk</b> , 2: <b>Map</b> , 3: <b>Path</b> , 4: <b>Bool</b> , 5: <b>Int</b>
<b>Sorts:</b> <i>oper</i>
<b>Operations:</b>
$\mathbf{O}_1 : \rightarrow \textit{oper}$ ; $\mathbf{O}_2 : \rightarrow \textit{oper}$ ;
$\mathbf{nw}() : \textit{oper} \rightarrow \textit{net}$ ;
$\mathbf{pt}() : \textit{oper} \times \textit{city} \times \textit{city} \rightarrow \textit{list}$ ;
$\widehat{\rightarrow, -} : \textit{oper} \times \textit{city} \times \textit{city} \rightarrow \textit{nat}$ ;
<b>Axioms:</b>
1: $\mathbf{O}_1 \doteq \mathbf{O}_1$ 2: $\mathbf{O}_2 \doteq \mathbf{O}_2$
3: $\mathbf{nw}(\mathbf{O}_1) \doteq \mathbf{l}(\mathbf{A}, \mathbf{B}) \diamond \mathbf{l}(\mathbf{B}, \mathbf{C}) \diamond \mathbf{l}(\mathbf{C}, \mathbf{A}) \diamond \emptyset$
4: $\mathbf{nw}(\mathbf{O}_2) \doteq \mathbf{l}(\mathbf{C}, \mathbf{B}) \diamond \mathbf{l}(\mathbf{B}, \mathbf{A}) \diamond \mathbf{l}(\mathbf{A}, \mathbf{C}) \diamond \emptyset$
5: $\mathbf{p}(\mathbf{pt}(o, c, c'), c, c', \mathbf{nw}(o)) \doteq \mathbf{T}$
6: $\mathbf{l}(\mathbf{A}, \mathbf{B}) \prec \mathbf{pt}(\mathbf{O}_1, \mathbf{A}, \mathbf{B})$
7: $\mathbf{\$}(\mathbf{pt}(o, c, c')) \geq \widehat{o, c, c'}$
8: $\widehat{o, c, c'} \prec \mathbf{\$}(\mathbf{pt}(o, c, c'))$

The **Oper** specification defines two operators,  $\mathbf{O}_1$  and  $\mathbf{O}_2$ , each of them having its own network ( $\mathbf{nw}()$ ). They propose paths from a city to another in a non-deterministic way ( $\mathbf{pt}()$ ). A function ( $\widehat{\rightarrow, -}$ ) gives the best cost an operator proposes for a given travel, and then, the agency selects ( $\mathbf{sl}()$ ) the operator with the best optimum. This is possible because the optimum function, which is *de facto* deterministic, will be preserved by the *forget* constructor and then imported by **Agency**.

In our proof example,  $\mathbf{O}_1$ , who proposes a direct path from **A** to **B**, will be selected.

### 3.2 Semantics

Several previous works [Bid87, NOS95] already presented the meaning of enrichment modules as a class of functions from the models of the imported specifications to models of the global one. We consider also primitives in the same way, and the semantics of a constructor  $\mathbf{f}$  are defined by means of partial functions, from the models of the domain signatures of  $\mathbf{f}$  to the models of the exported signature of  $\mathbf{f}$ :

**Definition 5.** A constructor semantics for a constructor universe  $\Theta = (\mathcal{C}, \mathbf{Spec})$  is a  $\mathcal{C}$ -indexed family  $\mathbf{Sem} = \coprod_{\mathbf{f} \in \mathcal{C}} \mathbf{Sem}_{\mathbf{f}}$  such that for each  $\mathbf{f}$  :

$\Sigma_1 \dots \Sigma_n \rightarrow \Sigma$  in  $\mathcal{C}$ , the elements of  $\mathbf{Sem}_{\mathbf{f}}$ , called implementations of  $\mathbf{f}$ , are partial functions  $\nu$  from  $\text{mod}(\Sigma_1) \times \dots \times \text{mod}(\Sigma_n)$  to  $\text{mod}(\Sigma)$ . Each definition domain  $D_\nu$  is called the semantic domain of the constructor implementation  $\nu$ .

Let us give 3 examples to illustrate this idea. First, the semantics of a homogeneous presentation  $\textit{basic}_P$  can be seen as a set of constant functions of arity zero, one for each flat model of  $\text{Mod}(P)$ . Second, a natural choice for a classical constructor  $\textit{forget}_\sigma$  is  $\mathbf{Sem}_{\textit{forget}_\sigma} = \{\text{mod}(\sigma) : \text{mod}(\Sigma_2) \rightarrow \text{mod}(\Sigma_1)\}$ .  $\text{mod}(\sigma)$  is often called the “forgetful functor” and is usually denoted by  $\mathcal{U}_\sigma$ . So, the semantics of  $\textit{forget}_\sigma$  is reduced to  $\{\mathcal{U}_\sigma\}$ . The semantics of heterogeneous bridge constructors  $\mathbf{f} = \textit{het}_{\mu, \Sigma} : \Sigma \rightarrow \text{Tr}_\mu(\Sigma)$  are given by the corresponding model extraction functions:  $\mathbf{Sem}_{\textit{het}_{\mu, \Sigma}} = \{\text{Ext}_{\mu, \Sigma}\}$ .

**Definition 6.** Given  $\Theta = (\mathcal{C}, \mathbf{Spec})$  and  $\mathbf{Sem}$ , a  $(\Theta, \mathbf{Sem})$ -realization is a partial substitution  $\rho : \mathcal{C} \rightarrow \mathbf{Sem}$  such that  $\rho(\mathbf{f})$ , when it is defined, belongs to  $\mathbf{Sem}_{\mathbf{f}}$ . We note  $\mathbf{Sem}(\Theta)$  the class of all  $(\Theta, \mathbf{Sem})$ -realizations.

**Notation** Given a  $(\Theta, \mathbf{Sem})$ -realization  $\rho : \mathcal{C} \rightarrow \mathbf{Sem}$ , we note  $\bar{\rho} : \mathbf{Spec} \rightarrow \text{mod}(\text{Sig})$  the partial canonical extension of  $\rho$  to  $\mathbf{Spec}$ . Notice that  $\bar{\rho}(\mathbf{f}(\tau_1, \dots, \tau_n))$  is defined if and only if:  $\rho(\mathbf{f})$  and all the  $\bar{\rho}(\tau_i)$  are defined, and  $(\bar{\rho}(\tau_1), \dots, \bar{\rho}(\tau_n))$  belongs to  $D_{\rho(\mathbf{f})}$ , the semantic domain of  $\rho(\mathbf{f})$ .

The partiality of  $\rho$  should not confuse the reader here: it simply means that to realize a specification  $\tau$ , it is not necessary to implement *all* other constructors that do not belong to  $\tau$ . On the contrary, the partiality of  $\bar{\rho}$  is significant. It means that incompatible implementations of the constructors of  $\tau$  never result into a realization of  $\tau$ . Finally, we can give the following definition:

**Definition 7.** Given  $\Theta = (\mathcal{C}, \mathbf{Spec})$ ,  $\mathbf{Sem}$ , and a well structured specification  $\tau$  in  $\mathbf{Spec}$ , the class of all flattened models of  $\tau$  is:  
 $\text{Mod}(\tau) = \{M \in \text{mod}(\Sigma_\tau) \mid \exists \rho \in \mathbf{Sem}(\Theta), M = \bar{\rho}(\tau)\}$ .

Our substitution mechanism ensures that a constructor  $\mathbf{f}$  still keeps the same implementation even for two occurrences which do not import the same subspecifications. It encompasses approaches dedicated to subspecification sharing within an homogeneous setting.

### 3.3 Inference Relation

We define heterogeneous inference relations similarly as [Wir93,HST94] did for homogeneous specifications. We define below a general mechanism of property transmission through a constructor. A HHS theory is then obtained by considering a sound heterogeneous inference relation for each constructor.

**Definition 8.** A HHS theory is a tuple  $(\Theta, \mathbf{Sem}, \Vdash)$  where  $\Theta$  is a constructor universe,  $\mathbf{Sem}$  is a constructor semantics over  $\Theta$  and  $\Vdash$  is a  $\mathcal{C}$ -indexed family of local heterogeneous inference relations,  $\Vdash = \prod_{\mathbf{f} \in \mathcal{C}} \Vdash_{\mathbf{f}}$ , such that for each constructor  $\mathbf{f} : \Sigma_1 \dots \Sigma_n \rightarrow \Sigma \in \mathcal{C}$ ,  $\Vdash_{\mathbf{f}}$  is a binary relation included in  $\mathcal{P}(\prod_{i=1 \dots n} \text{sen}(\Sigma_i) \times \text{sen}(\Sigma))$ , sound with respect to  $\mathbf{Sem}$ :

$$\begin{aligned} \forall (\Phi, \varphi) \in \Vdash_{\mathbf{f}}, \forall \tau = \mathbf{f}(\tau_1, \dots, \tau_n) \in \mathbf{Spec}, \forall \bar{\rho}(\tau) \in \text{Mod}(\tau), \\ (\forall i = 1 \dots n, \bar{\rho}(\tau_i) \models \Phi \cap \text{sen}(\Sigma_i)) \implies \bar{\rho}(\tau) \models \varphi \end{aligned}$$

## 4 Heterogeneous Structured Proofs

### 4.1 Inference System

There are two kinds of step in our heterogeneous inference system: homogeneous steps ( $\vdash$ ) and constructor steps ( $\Vdash$ ).

**Definition 9.**  $(\Theta, \mathbf{Sem}, \Vdash)$  being given, the corresponding inference system is the least binary relation  $\Vdash \subseteq \mathbf{Spec} \times \text{sen}(\text{Sig})$  such that, for any  $\tau \in \mathbf{Spec}$

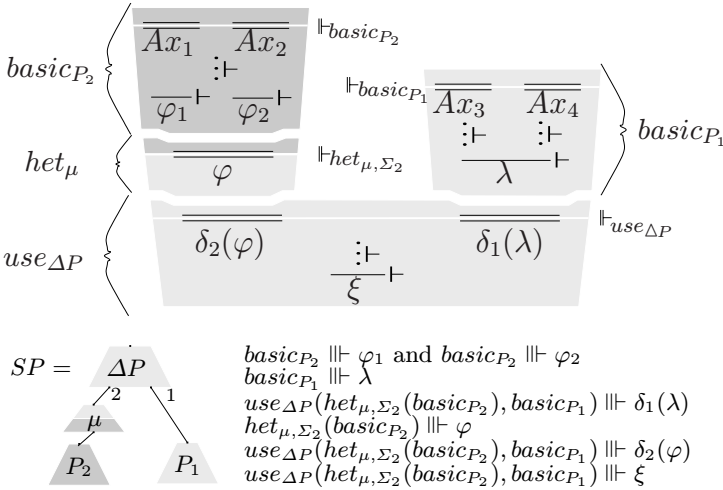
–  $\Vdash$  is transitive via  $\vdash$ :

$$\forall \Gamma \subseteq \text{sen}(\Sigma_\tau), \forall \varphi \in \text{sen}(\Sigma_\tau), (\tau \Vdash \Gamma) \wedge (\Gamma \vdash \varphi) \implies (\tau \Vdash \varphi).$$

–  $\Vdash$  is transitive via  $\Vdash$  : when  $\tau$  is of the form  $\mathbf{f}(\tau_1, \dots, \tau_n)$

$$\forall \Gamma \subseteq \prod_{i=1 \dots n} \text{sen}(\Sigma_{\tau_i}), \forall \varphi \in \text{sen}(\Sigma_{\tau}) \\ [(\forall i = 1 \dots n, (\tau_i \Vdash \Gamma \cap \text{sen}(\Sigma_{\tau_i}))) \wedge (\Gamma \Vdash \varphi)] \implies (\tau \Vdash \varphi)$$

With such steps, we can deduce a sentence  $\xi$  from the axioms of a heterogeneous specification  $SP$  ( $SP \Vdash \xi$ ). The figure below illustrates how the structure of the proof follows the term structure of the specification. It also introduces our conventional proof notations. The different grey scales represent different logics. The white interstices separate the input/output sides in the structure. Properties get over it with the  $\Vdash_f$ -relations (doubles lines). On this shape we inherit  $\delta_1(\lambda)$  from  $P_1$  and  $\delta_2(\varphi)$  from  $P_2$  to prove  $\xi$  in the  $\Delta P$  module, using its homogeneous inference  $\vdash$  (simple lines). The properties inherited from  $P_2$  are translated through the inference associated to the constructor  $het_{\mu, \Sigma_2}$ .



**Proposition 10.** For any HHS theory, the corresponding inference system is sound. This means that for any specification  $\tau$  and for any sentence  $\varphi$  in  $\text{sen}(\Sigma_{\tau})$ , we have:  $\tau \Vdash \varphi \implies (\forall M \in \text{Mod}(\tau), M \models \varphi)$

The structure of a heterogeneous proof deliberately follows the structure of the specification. A direct consequence is that if a module  $\mathbf{f}_1$  is based on a poorly expressive logic, and if it imports a module  $\mathbf{f}_2$  based on a powerful logic, then  $\mathbf{f}_1$  will be unable to pass some properties of  $\mathbf{f}_2$ , even to a higher level module  $\mathbf{f}_0$  based on the same logic as  $\mathbf{f}_2$ . So,  $\Vdash$  is intrinsically not complete<sup>4</sup>. A heterogeneous proof is neither harder nor easier than a structured proof w.r.t a purely homogeneous hierarchical specification. The only specific knowledges are the constructor inferences. For *basic* and *use* constructors, it is simply the axiom introduction and/or the transmission of all the imported properties through the involved signature morphisms.

<sup>4</sup> [HST94] already pointed out that their proof search procedure was not complete for many-sorted equational logic and first-order logic with equality.





$$\frac{\overline{\overline{o, c, c' \prec \$(\text{pt}(o, c, c'))}} \quad ax_0^8 \quad \overline{\overline{A \doteq A}} \quad REF \ h_5 \ u_O^2 \quad \overline{\overline{B \doteq B}} \quad REF \ h_5 \ u_O^2 \quad \overline{\overline{O_2 \doteq O_2}} \quad ax_O^2}{\widehat{O_2, A, B \prec \$(\text{pt}(O_2, A, B))} \quad @10} \quad sub$$

$$\frac{\quad @10 \quad @9}{\widehat{O_2, A, B \geq 2} \quad @11} \quad rg^3$$

## 5 Conclusion

Our approach of hierarchical heterogeneous structuring is resolutely both formal and pragmatic. Our first motivation was to manage heterogeneous libraries of already implemented modules, for reuse purposes: in the French telecommunication project ECOS/CNET, hardware components specified in VHDL have to cooperate in a system with software components.

From this point of view, we do not want to look for equivalence between formalisms, or completeness of proofs, but a good compromise between a sufficient power to prove what is usually needed and a reasonable simplicity in order to preserve legibility and *tractability*. For this reason, we do not require the heterogeneous bridges to be optimal in any sense.

One of the main contributions of this article is to systematically represent HHS by terms. It gives a very unified view of the hierarchical structuring mechanisms and allows us to manipulate HHS very easily, owing to the well established corpus on terms, substitutions, etc. It provides us with a well suited framework for “in the large” issues, and contributes to get a clearer view of how to heterogeneously combine formal methods in software engineering.

The agency example outlined here has been inspired by a more general routing problem in telecommunication networks. It is of course considerably simplified with respect to the original problem, however it remains representative enough to illustrate our approach. It indicates that all classical algebraic specifications approaches can be almost freely heterogeneously mixed together. This result is already very encouraging and significant. Netherless, we have tried to allow a broader scope by reducing as much as possible the hypotheses about homogeneous logics. This will hopefully allow to specialize our heterogeneous framework to model oriented specifications as well (Z,B,VDM,...). Our current researches exploit this flexibility and the term approach facilities. With our approach, it becomes possible to rigorously treat the heterogeneous refinements of a module. It amounts to replace a specification constructor by a “piece of term” which can be itself heterogeneous. A next question is possibly: “is it possible to handle object oriented structures in a similar manner?” (which is a bit out of the scope of this paper). Following the heterogeneous proof method of this article, we would also like to invent what integration testing of HHS could be [LGA96]. Details on definitions, specifications and proofs outlined in this article can be found in [Cou98,CLGB97].

## References

- AC92. E. Astesiano and M. Cerioli. Relationships between logical frameworks. In *Recent Trends in Data Type Specification*, volume 655, pages 101–126, Dourdan, 1992. LNCS. [111](#)
- AC94. E. Astesiano and M. Cerioli. Multiparadigm specification languages: a first attempt at foundations. *Semantics of Specification Languages*, Workshops in Computing, pages 168–185. Springer Verlag, 1994. [110](#), [111](#)
- BCLG96. G. Bernot, S. Coudert, and P. Le Gall. Towards heterogeneous formal specifications. In *AMAST'96, Munich*, volume 1101, pages 458–472. Springer, LNCS, 1996. [111](#)
- BH96. M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165 (1):3–55, 1996. [109](#), [111](#)
- Bid87. M. Bidoit. The stratified loose approach : a generalization of initial and loose semantics. In *Recent Trends in Data Type Specification, Gullane, Scotland*, pages 1–22. Springer-Verlag LNCS 332, July 1987. [108](#), [114](#)
- BLGA94. G. Bernot, P. Le Gall, and M. Aiguier. Label algebras and exceptions handling. *Journal of Science of Computer Programming*, 23:227–286, 1994. [109](#)
- BST99. M. Bidoit, D. Sannella, and A. Tarlecki. Architectural specification in casl. In *AMAST'98, Amazonia-Manaus*, volume to appear. Springer, LNCS, 1999. [107](#)
- CLGB97. S. Coudert, P. Le Gall, and G. Bernot. An example of heterogeneous structured specification. Université d'Évry, Report 28-1997, 1997. [117](#), [119](#)
- CM97. M. Cerioli and J. Meseguer. May I borrow your logic? *Theoretical Computer Sciences*, 173(2):311–347, 1997. [111](#)
- CoF96. CoFI. Common framework initiative. EATCS Bulletin, 1996. [107](#)
- Cou98. S. Coudert. Hiérarchie et hétérogénéité dans les spécifications formelles. Forthcoming Thesis, Université d'Évry, France, 1998. [117](#), [119](#)
- DD77. R. Douady and A. Douady. *Algèbre et théories galoisiennes, Tome 1 (Algèbre)*. CEDIC, Nathan, Paris, 1977. [112](#)
- DGS93. R. Diaconescu, J. Goguen, and P. Stefaneas. Logical support for modularisation. In G. Huet and G. Plotkin, editors, *Proc. Workshop on Types and Logical Frameworks*, pages 83–130, 1993. [107](#), [108](#)
- EM85. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and initial semantics*, volume 6. Springer-Verlag, EATCS Monographs on Theoretical Computer Science, 1985. [109](#)
- GB92. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. [111](#)
- HST94. R. Harper, D. Sannella, and A. Tarlecki. Structured theory presentations and logic representations. *Annals of Pure and Applied Logic*, 67, 1994. [107](#), [108](#), [108](#), [115](#), [116](#)
- LGA96. P. Le Gall and A. Arnould. Formal specifications and test: Correctness and oracle. In *Recent Trends in Data Type Specification, Oslo, Norway, September 1995*, pages 342–358. Springer-Verlag LNCS 1130, 1996. [119](#)
- Mes89. J. Meseguer. General logics. In *Proc. Logic. Colloquium '87*, Amsterdam, 1989. North-Holland. [109](#), [110](#), [111](#)
- Mes97. J. Meseguer. Membership algebra as a logical framework for equational specification. In *Recent Trends in Data Type Specification*, volume 1376, pages 18–61, Tarquinia, LNCS, 1997. [110](#)

- NOS95. M. Navarro, F. Orejas, and A. Sanchez. On the correctness of modular systems. *Theoretical Computer Science*, 140:139–177, 1995. [108](#), [111](#), [114](#)
- Pad96. P. Padawitz. Swinging data types: Syntax, semantics, and theory. In *Recent Trends in Data Type Specifications, Oslo, Norway, September 1995*, pages 409–435. Springer-Verlag LNCS 1130, 1996. [110](#)
- PW92. D.E. Perry and A.L. Wolf. Foundations for the study of software architectures. *ACM SIGSOFT, Software Engineering Notes*, pages 40–52, 1992. [107](#)
- SS96. A. Salibra and G. Scollo. Interpolation and compactness in categories of pre-institutions. *Mathematical Structures in Computer Science*, 6:261–286, 1996. [111](#)
- Tar96. A. Tarlecki. Moving between logical systems. In *Recent Trends in Data Type Specifications, Oslo*, pages 478–502. Springer-Verlag LNCS 1130, 1996. [110](#), [112](#)
- Wir93. M. Wirsing. Structured specifications: syntax, semantics and proof calculus. In Brauer W. Bauer F. and Schwichtenberg H., editors, *Logic and Algebra of Specification*, pages 411–442. Springer, 1993. [107](#), [108](#), [115](#)
- WM95. M. Walicki and S. Meldal. A complete calculus for the multialgebraic and functional semantics of nondeterminism. *ACM Transactions on Programming Languages and Systems*, 17: 2, p. 366-393, 1995-03, 1995. [109](#), [121](#)

## Annex

Inference system for ND ([\[WM95\]](#)):

$$\begin{array}{l}
 \text{R1: } \frac{\text{a- } \overline{x \neq y, x \doteq y} \text{ } rg1}{C_t^x \quad D, s \doteq t} \quad \text{b- } \overline{x \neq t, x \prec t} \text{ } rg1 \quad x, y \in \mathcal{V} \\
 \text{R2: } \frac{C_s^x, D \text{ } rg2}{C_t^x \quad D, s \prec t} \\
 \text{R3: } \frac{C_s^x, D \text{ } rg3}{C, s \preceq t \quad D, s \neq t} \quad x \text{ not in a right-hand side of } \prec \text{ in } C. \\
 \text{R4: } \frac{C, D}{C, D} \text{ } cut \quad (\preceq \text{ being either } \doteq \text{ or } \prec) \\
 \text{R5: } \frac{C}{C, e} \text{ } wea \\
 \text{R6: } \frac{C, x \neq t}{\vdash C_t^x} \text{ } eli \quad x \in \mathcal{V} - \mathcal{V}[t], \text{ at most one } x \text{ in } C
 \end{array}$$

Derived rules:

$$\begin{array}{l}
 \text{PER: } \frac{\text{a- } \overline{x \doteq x} \text{ } per}{C} \quad \text{b- } \overline{t \prec t} \text{ } per \\
 \text{REN: } \frac{C^x}{C^y} \text{ } ner \\
 \text{SUB: } \frac{C \quad D, t \doteq t}{C_t^x, D} \text{ } sub \\
 \text{INTR: } \frac{C_t^y}{C_x^y, x \neq t} \text{ } inc \quad y \text{ not in a right-hand side of } \prec
 \end{array}$$