General Structured Specifications

Sophie Coudert and Gilles Bernot and Pascale Le Gall

L.a.M.I., Université d'Évry, Cours Monseigneur Roméro, 91025 Evry Cedex, France
{coudert,bernot,legall}@lami.univ-evry.fr

Abstract. We propose a definition of structured heterogeneous formal specifications. We focus on the specification structures themselves and we represent them using terms instead of graphs. Then, we take benefit of classical results on terms. For example, the use of substitutions naturally allows to take into account subspecification sharing. Then, we show how some common issues such as proof mechanisms or modularity can be expressed inside our framework. Finally, we give an example of such a general framework of structured specifications.
Keywords: Structured heterogeneous specifications, modularity, general logics, institutions, logical frameworks, inference systems, algebraic specifications, theorem proving.

Introduction

Formal methods in software engineering, and in particular formal specifications, become almost sufficiently well controlled to envisage to apply them to very large size parts of a software project [GW96]. Today, a lot of formal foundations and tools exist to treat specification modules independently, or to treat hierarchical specifications written according to a unique, well established framework: formal languages (e.g., [Jon90,Spi89,Abr95,BG80,Wir94,Gau84,GH93]), advanced theorem provers (e.g., [Gor88,GG91,CCJC+95]), test generation tools (e.g., [DF94,DF93,Mar91]), etc. However, in order to cope with large scale specifications, one of the primary interests is to be able to consider *heterogeneous specifications*. For instance to reuse various software or hardware components in a new system, it implies that their specifications should be reused and combined as they are, thus we have to cope with specifications whose components can be written according to several different frameworks. Moreover, of course, the *structure* itself *of the specifications* becomes a "first class citizen" object of study, as in the so called software architecture approaches [GS93,PW92]. The CoFI initiative for example, gives a large place to structuring issues in the discussions, and there are several research works focused on the structuring primitives of specifications [Wir93,HT94,DGS93]. Extending the ideas of a previous article about heterogeneous specifications [BCLG96][1] and the ideas behind several articles about specification structures [Wir93,HT94,DGS93], this article gives a unified framework to handle both subjects. In other words, we propose a definition of *structured heterogeneous formal specifications* in general.

[1] where it was impossible for a module to import several heterogeneous modules.

Even if the detailed definitions may seem technical, the philosophy behind this article is very simple and based on natural ideas:

- Usually, a hierarchical specification is shown as a directed acyclic graph (DAG) with a root node (the top module), and a classical alternative representation is a tree whose nodes are the specification modules and where equal subtrees represent a unique sub-specification. This last convention allows to retrieve the DAG from the tree (and conversely).
- Syntactic, semantic, and more recently proving [VB95] issues can be handled via recursive considerations along the tree representation of the hierarchical specification. What raises strong difficulties is to manage the sharing aspects between equal subtrees representing equal sub-specifications [Roq94].
- Because substitutions can manage sharing properly, a first natural idea is then to represent structured specifications as terms instead of trees. The advantage would be to take benefit of the wide scientific corpus about terms, substitutions, rewriting, etc. This article is a first proposal in this direction. The first obstacle to this idea is the notion of well-formedness. For example, to represent a specification module $\triangle P$ which uses n specifications $SP_1, ..., SP_n$, it seems natural to write the "term" $use(\triangle P, SP_1, ..., SP_n)$. Unfortunately, if $SP_1, ..., SP_n$ can be viewed as terms easily, it is not the case for $\triangle P$ which is of different nature, and typing issues on the terms representing specifications become unmanageable.
- Things are different if we write $use_{\triangle P}(SP_1, ..., SP_n)$. The top level operation of this term is $use_{\triangle P}$ instead of use, thus there are as many different use operations as different specification modules, and all subterms are structured specifications. Would you believe that such a simple manipulation solves the question? It does! and the rest of the paper, more or less, shows this fact.

Of course, things are a little bit more complex in details. For example, to define semantics of structured heterogeneous specifications, we have to consider a sort of "higher order" substitution where not only variables at the leaves of the term can be substituted, but also operations in the internal nodes of a term. Moreover, we do not only consider a structure as a hierarchical construction via use modules, we also want to encompass all other primitives, such as $forget$, $rename$, etc. The most delicate issue has been to give a large place to heterogeneity in the structure (e.g., in our general approach $SP_1, ..., SP_n$ do not necessarily belong to the same specification framework, and the "output framework" of $operation(SP_1, ..., SP_n)$ can also be another framework). Also, we have been careful to provide a good way to express the proof mechanism within our general framework. One of our contribution is to provide a formalization of the proof principle which consists to "delegate lemmas" along the specification structure (or to dive them according to [HT94]) in order to prove a property at the top level of the specification. For all these concerns, representing structured specifications as terms has been very fruitful.

The terms we consider are simple first order typed terms. The type of a term is simply the signature exported by the specification (according to the "output framework" of the top level operation of the term). One difficulty is that

operations in terms are partial. For example, if we want to consider a structuring operation which "translates" a specification from a framework with exception handling features to a framework without exception handling features, then it is possible that only specifications where all exceptional cases are recovered can be translated. Thus the operation that goes from exception handling to this other framework is partial, it plays the role of a *filter*, and its definition domain can be understood as the well known *proof obligations*, which are often imposed in practice to consider a specification as "well-formed".

1 Preliminary definitions

Our framework will be based on a notion of formalisms similar with the one of general logic introduced in [Mes89], but our requirements about formalisms are weaker. In particular, the notion of signature morphisms is useless for us (and thus, the satisfaction condition) because signatures will now be linked between them through the specification structuring operations. Nonetheless, in the sequel, signature morphisms will serve us to describe classical examples of structured specifications.

Definition 1. *A* specification formalism b *is a tuple* $(Sig_b, sen_b, mod_b, \vdash_b, \models_b)$ *where:*

- Sig_b *is a class whose objects are called* signatures
- sen_b *is a total map from* Sig_b *to Set associating to each signature a set of* sentences *(Set being the category of classes)*
- mod_b *is a total map from* Sig_b *to Class associating to each signature a class of* models *(Class being the category of classes)*
- $\vdash_b$*, called* inference relation*, is a* Sig_b*-indexed family such that for each signature* Σ*,* $\vdash_{b,\Sigma}$ *is a binary relation included in* $\mathcal{P}(sen_b(\Sigma)) \times sen_b(\Sigma)$
- $\models_b$*, called* satisfaction relation*, is a* Sig_b*-indexed family such that for each signature* Σ*,* $\models_{b,\Sigma}$ *is a binary relation included in* $mod_b(\Sigma) \times sen_b(\Sigma)$

and the following properties are satisfied:

- $\vdash_b$ *is reflexive, monotonic and transitive, i.e., respectively*
  - $\forall \Sigma \in Sig_b, \forall \varphi \in sen_b(\Sigma),\ \{\varphi\} \vdash_{b,\Sigma} \varphi$
  - $\forall \Sigma \in Sig_b, \forall \Gamma \subseteq sen_b(\Sigma), \forall \Gamma' \subseteq sen_b(\Sigma), \forall \varphi \in sen_b(\Sigma),$ $\Gamma \vdash_{i,\Sigma} \varphi$ *and* $\Gamma \subseteq \Gamma' \implies \Gamma' \vdash_{b,\Sigma} \varphi$
  - $\forall \Sigma \in Sig_b, \forall \Gamma \subseteq sen_b(\Sigma), \forall \Gamma' \subseteq sen_b(\Sigma), \forall \varphi \in sen_b(\Sigma),$ $\Gamma \vdash_{b,\Sigma} \Gamma'$ *and*[2] $\Gamma \cup \Gamma' \vdash_{b,\Sigma} \varphi \implies \Gamma \vdash_{b,\Sigma} \varphi$
- soundness*: for any* $\Gamma \subseteq sen_b(\Sigma)$ *and any* $\varphi \in sen_b(\Sigma)$*, if* $\Gamma \vdash_{b,\Sigma} \varphi$ *then*[3]

$$\forall M \in mod_b(\Sigma), \quad (M \models_{b,\Sigma} \Gamma) \implies (M \models_{b,\Sigma} \varphi)$$

Notation 1 *We introduce the following notations:*

[2] $\Gamma \vdash_{b,\Sigma} \Gamma'$ means $\forall \varphi \in \Gamma', \Gamma \vdash_{b,\Sigma} \varphi$

[3] $M \models_{b,\Sigma} \Gamma$ means $\forall \varphi \in \Gamma,\ \ M \models_{b,\Sigma} \varphi$

– $\boldsymbol{\mathcal{B}}$ *is the class of all specification formalisms.*
– $Sig = \coprod_{b \in \boldsymbol{\mathcal{B}}} Sig_b$
– $mod : Sig \to$ *Class associates* $mod_b(\Sigma)$ *to each signature* Σ *in* Sig_b
– $sen : Sig \to$ *Set associates* $sen_b(\Sigma)$ *to each signature* Σ *in* Sig_b
– $\vdash \ = \coprod_{b \in \boldsymbol{\mathcal{B}}} \vdash_b$ *and* $\models \ = \coprod_{b \in \boldsymbol{\mathcal{B}}} \models_b$

2 General structured specification framework

Our approach focuses on the structure itself and proposes to see specifications as terms. Each function symbol of a specification term corresponds to a building operation of the specification. We will call such function symbols *structuring nodes*. For example, each basic module will be a constant node and two different enrichment modules are two different nodes. Intuitively, the semantics of these nodes are functions from the imported models towards the models of the resulting specification. Moreover, an inference mechanism is associated to each node. It allows the transmission of properties through the structure. Below, our framework is described by the three usual components: syntax, semantics and inference relation.

2.1 Syntax

A *meta-signature* of a general structured specification framework consists in a class of structuring nodes, each of them being provided with its definition domain:

Definition 2. *A* meta-signature $\boldsymbol{\Theta}$ *is a tuple* $(\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$ *where*

– $\boldsymbol{\mathcal{N}}$ *is a class of* nodes. *Each node is provided with a profile in* Sig^+. *We note* $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma$ *such a node* ($n \in \mathbb{N}$).
– $\boldsymbol{\mathcal{S}pec}$ *is a class of* well structured specifications *which is a subclass of the free term algebra on* $\boldsymbol{\mathcal{N}}$, $T_{\boldsymbol{\mathcal{N}}}$, *closed by sub-terms:*
$\forall \boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma \in \boldsymbol{\mathcal{N}}, \forall \boldsymbol{\tau}_1 \in T_{\boldsymbol{\mathcal{N}},\Sigma_1}, \dots, \forall \boldsymbol{\tau}_n \in T_{\boldsymbol{\mathcal{N}},\Sigma_n}$,
$\boldsymbol{f}(\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n) \in \boldsymbol{\mathcal{S}pec} \Longrightarrow \forall i = 1 \dots n, \boldsymbol{\tau}_i \in \boldsymbol{\mathcal{S}pec}$

<u>*Reminder:*</u> $\boldsymbol{\mathcal{N}}$ *being given, the free term algebra on* $\boldsymbol{\mathcal{N}}$ *is the least* Sig*-indexed family* $T_{\boldsymbol{\mathcal{N}}} = \coprod_{\Sigma \in Sig} T_{\boldsymbol{\mathcal{N}},\Sigma}$ *such that for every* $(\boldsymbol{f} : \Sigma_1, \dots, \Sigma_n \to \Sigma) \in \boldsymbol{\mathcal{N}}$, *if* $\boldsymbol{\tau}_1 \in T_{\boldsymbol{\mathcal{N}},\Sigma_1}$ *and* ... *and* $\boldsymbol{\tau}_n \in T_{\boldsymbol{\mathcal{N}},\Sigma_n}$ ($n \geq 0$), *then* $\boldsymbol{f}(\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n) \in T_{\boldsymbol{\mathcal{N}},\Sigma}$.

Following the previous example, any flat homogeneous presentation $P = (\Sigma, \Gamma)$, according to any specification formalism, can give rise to a node of arity 0, and used to introduce elementary specifications as basic cases in the recursive construction of structured specifications: $basic_P :\to \Sigma$. Also, say in the first order typed equational logic, each signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ can give

rise to $forget_\sigma : \Sigma_2 \to \Sigma_1$, with the obvious meaning as structuring operation. Many other examples can be invented. Notice that the signatures belonging to the profile of a node $\boldsymbol{f}$ do *not* necessarily share the same underlying specification formalism.

Notation 2 *If $\boldsymbol{\tau} \in T_{\boldsymbol{\mathcal{N}},\Sigma}$, we call Σ* the signature *of $\boldsymbol{\tau}$. By convention, the signature of $\boldsymbol{\tau}$ is denoted by $\Sigma_{\boldsymbol{\tau}}$. Moreover, we note $\boldsymbol{\mathcal{S}pec}_\Sigma = \boldsymbol{\mathcal{S}pec} \cap T_{\boldsymbol{\mathcal{N}},\Sigma} = \{T \in \boldsymbol{\mathcal{S}pec} \mid \Sigma_{\boldsymbol{\tau}} = \Sigma\}$*

Let us remark that a specification term is typed by its output signature. Intuitively, the coincidence of the signatures is the minimal requirement to connect two specifications components. However, this requirement is often not sufficient to limit the connections to the licit ones. The additional constraints are expressed by $\boldsymbol{\mathcal{S}pec}$ which is the class of all well-formed specifications.

Definition 3. *Given a meta-signature $\boldsymbol{\Theta} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$, the* syntactical domain *of a node $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma$ is the class $\boldsymbol{\mathcal{D}_f}$ of tuples $(\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n)$ such that $\boldsymbol{f}(\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n)$ belongs to $\boldsymbol{\mathcal{S}pec}$.*

For example, $\boldsymbol{\mathcal{D}}_{forget_\sigma}$ can be chosen as $\boldsymbol{\mathcal{S}pec}_{\Sigma_2}$ entirely (i.e., $forget_\sigma$ is a total function).

Remark 1. $\boldsymbol{\mathcal{N}}$ being given, the class $\boldsymbol{\mathcal{S}pec}$ is entirely determined by the knowledge of the syntactic domain $\boldsymbol{\mathcal{D}_f}$ for each $\boldsymbol{f}$ in $\boldsymbol{\mathcal{N}}$.

2.2 Semantics

Definition 4. *A* node semantics *for a meta-signature $\boldsymbol{\Theta} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$ is a $\boldsymbol{\mathcal{N}}$-indexed family $\boldsymbol{Sem} = \coprod_{\boldsymbol{f} \in \boldsymbol{\mathcal{N}}} \boldsymbol{Sem_f}$ such that for each $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma$ in $\boldsymbol{\mathcal{N}}$, the elements of $\boldsymbol{Sem_f}$, called* implementations *of $\boldsymbol{f}$, are partial functions ν from $mod(\Sigma_1) \times \dots \times mod(\Sigma_n)$ to $mod(\Sigma)$, their definition domain being denoted by D_ν called the* semantic domain *of the node implementation ν.*

Notation 3 *Given $\boldsymbol{\Theta}$ and $\boldsymbol{Sem}$, the class $mod(Sig)$ (Notation 2) is equipped with a partial $\boldsymbol{Sem}$-algebra structure (from Definition 6).*
We note $\boldsymbol{\mathcal{M}od}(\Theta, \boldsymbol{Sem})$ this partial algebra whose carrier is $mod(Sig)$.

Thus, the semantics of a node are partial functions, from the models of the domain signatures of $\boldsymbol{f}$ to the models of the codomain signature of $\boldsymbol{f}$. For example, a natural choice for a node $forget_\sigma$ is the singleton $\boldsymbol{Sem}_{forget_\sigma} = \{mod(\sigma)\}$, the functor $mod(\sigma)$ being often called the forgetful functor in the first order logic framework. A natural choice for a node $basic_P$ could be $mod(P)$, which means that each model satisfying the specification is considered as a correct implementation of the node. If applicable in a given specification formalism, another possibility is to restrict $\boldsymbol{Sem}_{basic_P}$ to finitely generated models for example.

Notice that node implementations ν are *a priori* partial functions. Some models of the signatures of the domain of $\boldsymbol{f}$ may be unacceptable for a given

implementation ν. A rough analogy to make things pragmatically clear is the case of a program module ν which requires to import 32 bits integers and no other implementations of integers[4]. For this reason, $\boldsymbol{\mathcal{M}od}(\Theta, Sem)$ is a (strict) partial algebra.

We can obviously imagine structured specifications $\boldsymbol{\tau}$ such that a given node $\boldsymbol{f}$ (say a module for example) appears several times in the term. As already mentioned in the introduction, to formalize that $\boldsymbol{f}$ should have only *one* implementation in a correct realization of $\boldsymbol{\tau}$ is very difficult when considering specifications as trees or DAGs. Moreover this kind of sharing is generally not considered if the two occurrences of $\boldsymbol{f}$ do not import the same subspecifications. Here, the classical notion of substitution solves the problem easily.

Definition 5. *Given $\boldsymbol{\Theta} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$ and $\boldsymbol{Sem}$, a* $(\Theta, \boldsymbol{Sem})$-realization *is a* partial *substitution $\boldsymbol{\rho} : \boldsymbol{\mathcal{N}} \rightarrow \boldsymbol{Sem}$ such that for $\boldsymbol{f}$ in $\boldsymbol{\mathcal{N}}$, $\boldsymbol{\rho}(\boldsymbol{f})$, when it is defined, belongs to $\boldsymbol{Sem_f}$. We note $\boldsymbol{Sem}(\boldsymbol{\Theta})$ the class of all* $(\Theta, \boldsymbol{Sem})$-realizations.

Notation 4 *Given a $(\Theta, \boldsymbol{Sem})$-realization $\boldsymbol{\rho} : \boldsymbol{\mathcal{N}} \rightarrow \boldsymbol{Sem}$, we note $\overline{\boldsymbol{\rho}} : \boldsymbol{\mathcal{S}pec} \rightarrow \boldsymbol{\mathcal{M}od}(\Theta, \boldsymbol{Sem})$ the* partial *canonical extension of $\boldsymbol{\rho}$ to $\boldsymbol{\mathcal{S}pec}$.*
Notice that $\overline{\boldsymbol{\rho}}(\boldsymbol{f}(\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_n))$ is defined if and only if: $\boldsymbol{\rho}(\boldsymbol{f})$ and all the $\overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_i)$ are defined, and $(\overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_1), \ldots, \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_n))$ belongs to $D_{\boldsymbol{\rho}(\boldsymbol{f})}$, the semantic domain of $\boldsymbol{\rho}(\boldsymbol{f})$.

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

Definition 6. *Given $\boldsymbol{\Theta} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$, $\boldsymbol{Sem}$, and a well structured specification $\boldsymbol{\tau}$ in $\boldsymbol{\mathcal{S}pec}$, the class of all* flattened models *of $\boldsymbol{\tau}$ is:*
$\mathcal{M}od(\boldsymbol{\tau}) = \{M \in mod(\Sigma_{\boldsymbol{\tau}}) \mid \exists \boldsymbol{\rho} \in \boldsymbol{Sem}(\Theta), M = \overline{\boldsymbol{\rho}}(\boldsymbol{\tau})\}$.

2.3 Inference relation

We define heterogeneous inference relations as in [BCLG96]. They allow to heterogeneously infer sentences on the signatures of the codomain of a node using sentences on the signatures of the domain of this node. This is done in a rather natural way. Remind however that the involved signatures do not necessarily all belong to a common specification formalism. There can be up to one different formalism per signature.

Definition 7. *Given a meta-signature $\Theta = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$ and a node $\boldsymbol{f} : \Sigma_1 \ldots \Sigma_n \rightarrow \Sigma \in \boldsymbol{\mathcal{N}}$, a (local)* heterogeneous inference relation *for $\boldsymbol{f}$ is a binary relation* $\vdash\!\sim \subseteq \mathcal{P}(\coprod_{i=1\ldots n} sen(\Sigma_i)) \times sen(\Sigma)$.
In the remainder of this article, we will note indifferently $(\Phi, \varphi) \in \vdash\!\sim$ or $\Phi \vdash\!\sim \varphi$.

[4] even if they fulfill the integer specification under consideration

Definition 8. *Given a meta-signature* $\Theta = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$ *and a node* $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma$ *in* $\boldsymbol{\mathcal{N}}$*, a local heterogeneous inference relation* $\mid\!\sim$ *for* $\boldsymbol{f}$ *is* sound *with respect to a node semantic* $\boldsymbol{Sem}$ *if and only if*

$$\forall (\Phi, \varphi) \in \mid\!\sim,\ \forall \boldsymbol{\tau} = \boldsymbol{f}(\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n) \in \boldsymbol{\mathcal{S}pec},\ \forall \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}) \in \mathcal{M}od(\boldsymbol{\tau}),$$

$$(\forall i = 1 \dots n,\ \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_i) \models \Phi \cap sen(\Sigma_i)) \Longrightarrow \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}) \models \varphi$$

A general structured specification framework is then obtained by considering heterogeneous inference relation for each node.

Definition 9. *A* general structured specification framework *is a tuple* $(\Theta, \boldsymbol{Sem}, \Vdash)$ *where* Θ *is a meta-signature,* $\boldsymbol{Sem}$ *is a node semantics over* Θ *and* $\Vdash$ *is an* $\boldsymbol{\mathcal{N}}$*-indexed family of local heterogeneous inference relations,* $\Vdash = \coprod_{\boldsymbol{f} \in \boldsymbol{\mathcal{N}}} \Vdash_{\boldsymbol{f}}$*, such that for each* $\boldsymbol{f}$ *in* $\boldsymbol{\mathcal{N}}$*,* $\Vdash_{\boldsymbol{f}}$ *is sound with respect to* $\boldsymbol{Sem}$*.*

3 General properties

In this section, we sketch out how our framework allows to easily express some methodology issues. First, we give a proof mechanism following the specification structure. Similar structured proof mechanisms have been already proposed within a unique formalism [Wir93] [HT94], but without addressing the problem of the structuring of models. We also characterize the modularity in our heterogeneous world as the requirement of two fundamental properties: the independence of the module implementations and the preservation of properties through a node.

3.1 Inference system

Here, we propose an inference system for our structured specifications which allows to use properties inherited from sub-specifications in order to prove properties at the top level. There will be two kinds of step, depending on which inference relation will be used. The homogeneous steps will use the inferences relations $\vdash$ of the specification formalisms. They allow to deduce, from properties about one model, other properties about the same model. The heterogeneous steps will use the heterogeneous inference relations. They allow to deduce, from properties about one model, properties about the images of this model by node implementations.

Definition 10. *A general structured specification framework* $(\boldsymbol{\Theta}, \boldsymbol{Sem}, \Vdash)$ *being given, the corresponding* inference system *is the least binary relation* $\Vdash\!\!\Vdash \subseteq \boldsymbol{\mathcal{S}pec} \times sen(Sig)$ *such that, for any* $\boldsymbol{\tau} \in \boldsymbol{\mathcal{S}pec}$

- $\Vdash\!\!\Vdash$ is transitive via $\vdash$*:*
  $\forall \Gamma \subseteq sen(\Sigma_{\boldsymbol{\tau}}),\ \forall \varphi \in sen(\Sigma_{\boldsymbol{\tau}}),\ (\boldsymbol{\tau} \Vdash\!\!\Vdash \Gamma) \wedge (\Gamma \vdash \varphi) \implies (\boldsymbol{\tau} \Vdash\!\!\Vdash \varphi)$.

– $\Vdash\!\!\vdash$ is transitive via $\Vdash$ *: when* $\boldsymbol{\tau}$ *is of the form* $\boldsymbol{f}(\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_n)$
$\forall \Gamma \subseteq \coprod_{i=1\ldots n} sen(\Sigma_{\boldsymbol{\tau}_i}),\ \forall \varphi \in sen(\Sigma_{\boldsymbol{\tau}})$
$[\,(\forall i = 1 \ldots n,\ (\boldsymbol{\tau}_i \Vdash\!\!\vdash \Gamma \cap sen(\Sigma_{\boldsymbol{\tau}_i}))) \wedge (\Gamma \Vdash \varphi)\,] \implies (\boldsymbol{\tau} \Vdash\!\!\vdash \varphi)$

A similar idea is developed in [BCLG96] and a relevant example of heterogeneous proof is given.

Theorem 5. *For any general structured specification framework , the corresponding inference system is monotonic and sound. This means that for any specification* $\boldsymbol{\tau}$ *and for any sentence* φ *in* $sen(\Sigma_{\boldsymbol{\tau}})$*, we have:*

soundness:
– $\boldsymbol{\tau} \Vdash\!\!\vdash \varphi \implies (\forall M \in \mathcal{M}od(\boldsymbol{\tau}),\ M \models \varphi)$
monotony:
– $\forall \Gamma \cup \{\psi\} \subseteq sen(\Sigma_{\boldsymbol{\tau}}),\ (\boldsymbol{\tau} \Vdash\!\!\vdash \Gamma \cup \{\psi\}) \wedge (\Gamma \vdash \varphi) \implies (\boldsymbol{\tau} \Vdash\!\!\vdash \varphi)$.
– *for any node* $\boldsymbol{f}$*, for any* $(\boldsymbol{\tau}_1, \ldots \boldsymbol{\tau}_n)$ *in* $\boldsymbol{\mathcal{D}}_{\boldsymbol{f}}$*, for any* $\Gamma \cup \{\psi\}$ *of* $\coprod_{i=1\ldots n} sen(\Sigma_{\boldsymbol{\tau}_i})$,
$[\,(\forall i = 1 \ldots n,\ (\boldsymbol{\tau}_i \Vdash\!\!\vdash (\Gamma \cup \{\psi\}) \cap sen(\Sigma_{\boldsymbol{\tau}_i}))) \wedge (\Gamma \Vdash \varphi)\,] \implies (\boldsymbol{\tau} \Vdash\!\!\vdash \varphi)$

Proof. The proof of soundness is done by induction on the proofs by using the soundness properties of $\vdash$ and $\Vdash$. The one of monotony is done by inserting a step $\Gamma \cup \{\psi\} \vdash \Gamma$ in the proof (from the reflexivity and monotony properties of $\vdash$).

Let us remark that here, the validity is expressed with respect to the flat models of the specification.

3.2 Stepwise integration

We can easily formalize in our framework the notion of stepwise integration. Intuitively, it means that we can obtain a realization of specification by using some already implemented components and by implementing step by step the missing ones. Such an already existing partial implementation is simply a realization $\boldsymbol{\rho}$, and then, adding component implementations is just extending this realization to these components.

Definition 11. *A general structured specification framework* $(\boldsymbol{\Theta}, \boldsymbol{Sem}, \Vdash)$ *and a realization* $\boldsymbol{\rho}$ *being given, the class of all realizations that* extend $\boldsymbol{\rho}$ *is:*
$\boldsymbol{Sem}(\boldsymbol{\rho}, \boldsymbol{\Theta}) = \{\boldsymbol{\rho}' \in \boldsymbol{Sem}(\boldsymbol{\Theta}) \mid D_{\boldsymbol{\rho}} \subseteq D_{\boldsymbol{\rho}'} \wedge \boldsymbol{\rho}'_{|D_{\boldsymbol{\rho}}} = \boldsymbol{\rho}\}$
Similarly, the class of all flattened models of $\boldsymbol{\tau}$ *that* follow $\boldsymbol{\rho}$ *is:*
$\boldsymbol{\mathcal{M}od}(\boldsymbol{\rho}, \boldsymbol{\tau}) = \{M \in \mathcal{M}od(\boldsymbol{\tau}) \mid \exists \boldsymbol{\rho}' \in \boldsymbol{Sem}(\boldsymbol{\rho}, \boldsymbol{\Theta}),\ M = \overline{\boldsymbol{\rho}'}(\boldsymbol{\tau})\}$

Notation 6 *Given a general structured specification framework* $(\boldsymbol{\Theta}, \boldsymbol{Sem}, \Vdash)$ *a node* $\boldsymbol{f}$ *of* $\boldsymbol{\Theta}$ *and a function* ν *in* $\boldsymbol{Sem}_{\boldsymbol{f}}$*, we denote by* $[\nu/\boldsymbol{f}]$ *the substitution of* $\boldsymbol{Sem}(\boldsymbol{\Theta})$ *verifying* $D_{[\nu/\boldsymbol{f}]} = \{\boldsymbol{f}\}$ *and* $[\nu/\boldsymbol{f}](\boldsymbol{f}) = \nu$.

It denoted the particular realizations which only implement one component. It provides us with a simple way, for example, to point out all the realizations of a specification $\boldsymbol{\tau}$ that share a common implementation ν of a module $\boldsymbol{f}$, with $\mathcal{M}od([\nu/\boldsymbol{f}], \boldsymbol{\tau})$.

3.3 Independent implementations

A classical property which is required for modularity is that the different modules of a specification can be separately implemented. More precisely, a modular composition of specification modules id such that any implementation choice for the modules provides an implementation of the global specification. In our framework, this idea is expressed by the fact that all the possible implementations of a modular node $\boldsymbol{f}$ cope with *any* realization of its potential input specifications ($\boldsymbol{\mathcal{D}_f}$).

Definition 12. *Given a general structured specification framework* $\boldsymbol{\mathcal{F}} = (\boldsymbol{\Theta}, \boldsymbol{Sem}, \Vdash)$ *a node* $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma \in \boldsymbol{\mathcal{N}}$ *of* $\boldsymbol{\Theta}$ *and a function* ν *in* $\boldsymbol{Sem_f}$*, the class of all* $\boldsymbol{\mathcal{F}}$-inputs *of* ν *is denoted by* $Inputs_{\boldsymbol{\mathcal{F}}}(\nu) = \{\boldsymbol{M} \in mod(\Sigma_1) \times \dots \times mod(\Sigma_n) \mid \exists (\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n) \in \boldsymbol{\mathcal{D}_f},\ \exists \boldsymbol{\rho} \in \boldsymbol{Sem}([\nu/\boldsymbol{f}], \boldsymbol{\Theta}),\ \boldsymbol{M} = (\overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_1), \dots, \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_n))\}$.

Definition 13. *Given a general structured specification framework* $\boldsymbol{\mathcal{F}} = (\boldsymbol{\Theta}, \boldsymbol{Sem}, \Vdash)$ *and a node* $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma \in \boldsymbol{\mathcal{N}}$ *of* $\boldsymbol{\Theta}$*,*

- *A function* ν *in* $\boldsymbol{Sem_f}$ *is said to be an* independent implementation *of* $\boldsymbol{f}$ *if and only if all the* $\boldsymbol{\mathcal{F}}$*-inputs of* ν *belong to the semantics domain of* ν*, i.e.* $Inputs_{\boldsymbol{\mathcal{F}}}(\nu) \subseteq D_\nu$*.*
- $\boldsymbol{f}$ *is said to be an* independent node *in* $\boldsymbol{\mathcal{F}}$ *if and only if all implementations of* $\boldsymbol{Sem_f}$ *are independent.*

3.4 Encapsulation

The second requirement for a node to be modular is the preservation of what it imports. From our property-oriented point of view, we would like to preserve in the global model the observable properties of the components. But these properties are not expressed in the same language: the specifications under consideration can belong to different frameworks. So, such a principle of properties preservation has to be expressed by a *heterogeneous* relation $\mathrel{|\!\sim}$ which will serve us as an external reference for the properties we want to preserve. This property preservation mechanism encapsulates properties in two directions, from the imported specification to the global one and reciprocally. This has been done to ensure a possible delegation of lemmas guided by the specification structure. This property preservation according to an external inference relation $\mathrel{|\!\sim}$ is called $\mathrel{|\!\sim}$-encapsulation:

Definition 14. *Given a meta-signature* $\boldsymbol{\Theta} = (\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}pec})$*, a node semantics* $\boldsymbol{Sem}$*, a node* $\boldsymbol{f} : \Sigma_1 \dots \Sigma_n \to \Sigma$ *and a heterogeneous inference relation* $\mathrel{|\!\sim}$ *for* $\boldsymbol{f}$*, a function* ν *of* $\boldsymbol{Sem_f}$ *is said to be* $\mathrel{|\!\sim}$-encapsulated *if and only if:*

for any subset Φ *of* $\coprod_{i=1\dots n} sen(\Sigma_i)$*, if* $\Psi = \{\varphi \in sen(\Sigma) \mid \Phi \mathrel{|\!\sim} \varphi\} \neq \emptyset$*, then*

$$\forall \boldsymbol{\tau} = \boldsymbol{f}(\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_n) \in \boldsymbol{\mathcal{S}pec},\ \forall \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}) \in \boldsymbol{\mathcal{M}od}([\nu/\boldsymbol{f}], \boldsymbol{\tau}),$$

$$\overline{\boldsymbol{\rho}}(\boldsymbol{\tau}) \models \psi \iff \forall i = 1 \dots n,\ \overline{\boldsymbol{\rho}}(\boldsymbol{\tau}_i) \models \Phi \cap sen(\Sigma_i)$$

If for each ν *in* $\boldsymbol{Sem_f}$*,* ν *is* $\mathrel{|\!\sim}$*-encapsulated, then* $\boldsymbol{Sem_f}$ *and* $\boldsymbol{f}$ *are said to be* $\mathrel{|\!\sim}$*-encapsulated too.*

4 *BUFF*: A proposed example of general structured specification framework

In this section, we describe a particular structured specification framework built over "basic formalisms". On such basic formalisms, we propose four kinds of nodes:

- constant $basic_P$ for classical presentations of the form $P = (\Sigma, \Gamma)$;
- $use_{\Delta P}$ for a use primitive based on an enrichment module ΔP;
- $forget_\sigma$ for a restriction primitive based on a signature morphism σ;
- $filter_H$ for a heterogeneous primitive based on a formalism morphism;

This set of structuring primitives is a good compromise between a simple and concise language for structured specifications and the minimal requirements to compose specifications. For homogeneous structures, some authors choose a different distribution of the primitives. In a model-theoretic framework [Wir93], Wirsing propose four primitives: presentation, renaming , restriction on a subsignature and union of two specifications defined on two different signatures. In a proof-theoretic framework [HT94], they also propose presentation and restriction, but the two other primitives are different: extension on a oversignature and union of two specifications defined over a signature. The scope of our choice is roughly the same as the ones of [Wir93,HT94] insofar as each family of primitives allows to build structured specification by either abstracting, renaming or combining subspecifications.

The heterogeneous primitive $filter$ has been introduced in one of our previous articles [BCLG96] in order to compose modules written in different formalisms. It is also similar to the one denoted **derive from** SP **via** γ introduced in [Tar95] which consists in "hiding some components in the specification while preserving the interpretation of the remaining parts according to an institution semi-morphism γ." However, Tarlecki does not tackle the question of a proof mechanism through such a heterogeneous bridge.

Our example of structured specification framework, called[5] *BUFF*, is built over the notion of *basic formalisms* where signature morphisms are introduced in order to cope with the classical meanings of the structuring primitives. But still we do not consider the satisfaction condition since δ-encapsulation encompasses it in the context of structured specifications.

4.1 The category $\mathcal{F}orm$ of basic formalisms

Definition 15 ((Basic formalism)). *A* basic formalism *is a specification formalism* $b = (Sig_b, sen_b, mod_b, \vdash_b, \models_b)$ *such that:*

- Sig_b *is equipped with* signature morphisms *in order to become a category.*
- sen_b *is a functor.*
- mod_b *is a contravariant functor.*

[5] the letters are the first letters of the primitive names

As usual, for any signature morphism σ, $mod(\sigma)$ will be denoted by $\mathcal{U}_\sigma$.

Definition 16 ((Morphism of Specification Formalisms)).
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 specification formalisms. Then a morphism $\mu : a \to b$ consists of

- *A functor $Tr_\mu : Sig_a \to Sig_b$, called* signature transposition functor
- *A Sig_a-indexed family Ext_μ such that $Ext_{\mu,\Sigma} : mod_a(\Sigma) \to mod_b(Tr_\mu(\Sigma))$ 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

such that the following properties are satisfied:

- *Ext_μ is a (partial) natural transformation from mod_a to $mod_b \circ Tr_\mu$*
- *$\Vdash_\mu$ is monotonic*
- $\Vdash$-translation*: for any isomorphism $\sigma : \Sigma \to \Sigma'$ in Sig_a, any $\Gamma \subseteq sen_a(\Sigma)$, and any $\varphi \in sen_b(Tr_\mu(\Sigma))$, $\Gamma \Vdash_{\mu,\Sigma} \varphi$ if and only if $\sigma(\Gamma) \Vdash_{\mu,\Sigma'} Tr_\mu(\sigma)(\varphi)$*
- 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 $M \in mod_a(\Sigma)$, we have $[M \models_a \Gamma \implies Ext_{\mu,\Sigma}(M) \models_b \varphi]$*

This notion of morphisms of specification formalisms allows to characterize the minimal syntactic, semantic and proof requirements on bridges between basic formalisms to define heterogeneous specifications following our approach presented in [BCLG96]. Such bridges between two basic formalisms a and b allow to transpose signatures from a to b, to extract models from a to b and to heterogeneously infer sentences from a to b.

Notation 7 *Given two morphisms $\mu : a \to c$ and $\mu' : c \to b$, if we define the composition $\mu' \circ \mu : a \to b$ in the obvious way, with $\Vdash_{\mu' \circ \mu, \Sigma} = \{(\Phi, \varphi) \in \mathcal{P}(sen_a(\Sigma)) \times sen_b(Tr_{\mu' \circ \mu}(\Sigma)) \mid \exists \Psi \in \mathcal{P}(sen_u(Tr_\mu(\Sigma))), \forall \psi \in \Psi, \Phi \Vdash_{\mu,\Sigma} \psi \wedge \Psi \Vdash_{\mu', Tr_\mu(\Sigma)} \varphi\}$, then basic formalisms forms a category. We denote it by $\mathcal{F}orm$.*

4.2 The meta-signature of *BUFF*

Definition 17. *The meta-signature of BUFF is $\boldsymbol{\Theta}_{BUFF} = (\boldsymbol{\mathcal{N}}_{BUFF}, \boldsymbol{\mathcal{S}pec}_{BUFF})$, where $\boldsymbol{\mathcal{N}}_{BUFF} =$ BASIC⊔USE⊔FORGET⊔FILTER and $\boldsymbol{\mathcal{S}pec}_{BUFF}$ are[6] the smallest classes satisfying the following:*

Basic nodes:
- *A* presentation *of $\mathcal{F}orm$ is a tuple $P = (\Sigma, \Gamma)$, where $\Sigma \in Sig$ and $\Gamma \subseteq sen(\Sigma)$.*
- *A* basic node *is a node of the form $basic_P : \to \Sigma$, where $P = (\Sigma, \Gamma)$ is a presentation of $\mathcal{F}orm$, with $\boldsymbol{\mathcal{D}}_{basic_P} = \mathbb{1}$ (i.e. all "constant" basic specifications belong to $\boldsymbol{\mathcal{S}pec}$)*

[6] remind that $\boldsymbol{\mathcal{S}pec}$ is entirely defined by the knowledge of syntactic domains $\boldsymbol{\mathcal{D}}_{\boldsymbol{f}}$ for all $\boldsymbol{f}$ in $\boldsymbol{\mathcal{N}}$.

– **BASIC** *denotes the class of all basic nodes.*

Use nodes:

– *An* enrichment module *is a tuple* $\triangle P = (\delta_1 : \Sigma_n \to \Sigma, \ldots, \delta_n : \Sigma_n \to \Sigma, \triangle\Gamma, \boldsymbol{\mathcal{D}}_{\triangle P})$ *such that all* δ_i *are signature morphisms of* Sig_b *for some* b *in* $\mathcal{F}orm$, $\triangle\Gamma$ *is a subset of* $sen_b(\Sigma)$ *and* $\boldsymbol{\mathcal{D}}_{\triangle P}$ *is a subset of* $\boldsymbol{\mathcal{S}pec}_{\Sigma_1} \times \ldots \times \boldsymbol{\mathcal{S}pec}_{\Sigma_n}$.

– *A* use node *is a node of the form* $use_{\triangle P} : \Sigma_1 \ldots \Sigma_n$, *where* $\triangle P = (\delta_1 : \Sigma_n \to \Sigma, \ldots, \delta_n : \Sigma_n \to \Sigma, \triangle\Gamma, \boldsymbol{\mathcal{D}}_{\triangle P})$ *is an enrichment module, with* $\boldsymbol{\mathcal{D}}_{use_{\triangle P}} = \boldsymbol{\mathcal{D}}_{\triangle P}$.

– **USE** *denotes the class of all use nodes.*

Forget nodes:

– *A* forget node *is a node of the form* $forget_\sigma : \Sigma_1 \to \Sigma_2$, *where* $\sigma : \Sigma_2 \to \Sigma_1$ *is a signature morphism for some* b *in* $\mathcal{F}orm$, *with* $\boldsymbol{\mathcal{D}}_{forget_\sigma} = \boldsymbol{\mathcal{S}pec}_{\Sigma_1}$.

– **FORGET** *denotes the class of all forget nodes.*

Heterogeneous filter nodes:

– *A* heterogeneous filter *is a tuple* $H = (\mu, \Sigma, \boldsymbol{\mathcal{D}}_H)$ *such that* $\mu : a \to b$ *is a morphism of basic formalisms,* Σ *is a signature belonging to* Sig_a *an* $\boldsymbol{\mathcal{D}}_H$ *is a subset of* $\boldsymbol{\mathcal{S}pec}_\Sigma$.

– *a* heterogeneous filter node *is a node of the form* $filter_H : \Sigma \to Tr_\mu(\Sigma)$ *where* $H = (\mu, \Sigma, \boldsymbol{\mathcal{D}}_H)$ *is a heterogeneous filter, with* $\boldsymbol{\mathcal{D}}_{filter_H} = \boldsymbol{\mathcal{D}}_H$.

– **FILTER** *denotes the class of all heterogeneous filter nodes.*

Let us remark that $forget_\sigma$ when σ is a signature isomorphism, simply amounts to a classical $rename_\sigma$ operation.

4.3 Semantics of *BUFF*

As in [BCLG96], we consider implementations which ensure the preservation of the imported sentences through signature morphisms. They formalize some encapsulation principles and will be used to define semantic and proof aspects of the *use* nodes.

Definition 18. *Given* n *signature morphisms* $(\delta_1 : \Sigma_1 \to \Sigma), \ldots, (\delta_n : \Sigma_n \to \Sigma)$ *in a basic formalism* $(Sig, sen, mod, \Vdash, \models)$, *we note* $\mid\sim_{\delta_1,\ldots,\delta_n}$ *the binary relation on* $\mathcal{P}(\coprod_{i=1\ldots n} sen(\Sigma_i)) \times \Sigma$ *such that*

$$\Phi \mid\sim_{\delta_1,\ldots,\delta_n} \psi \iff \exists i = 1 \ldots n,\ \exists \varphi \in \Phi,\ \psi = \delta_i(\varphi)$$

Definition 19. *The node semantics of BUFF is* $\boldsymbol{Sem}_{BUFF} = \coprod_{\boldsymbol{f} \in \boldsymbol{\mathcal{N}}_{BUFF}} \boldsymbol{Sem}_{\boldsymbol{f}}$, *with*

Basic nodes:

if $\boldsymbol{f}$ *is of the form* $basic_P \in$ **BASIC**, *with* $P = (\Sigma, \Gamma)$, *then* $\boldsymbol{Sem}_{\boldsymbol{f}}$ *is the subclass of* $mod(\Sigma)$ *whose objects satisfy* Γ.

$\boldsymbol{Sem}_{\boldsymbol{f}} = \{M \in mod(\Sigma) \mid M \models \Gamma\}$

Use nodes:
if $\boldsymbol{f}$ *is of the form* $use_{\triangle P} \in$ **USE**, *with* $\triangle P = (\delta_1 : \Sigma_1 \to \Sigma, \ldots, \delta_n : \Sigma_n \to \Sigma, \triangle\Gamma, \boldsymbol{\mathcal{D}}_{\triangle P})$, *then* $\boldsymbol{Sem_f}$ *is the class of all* $\mid\!\sim_{\delta_1,\ldots,\delta_n}$*-encapsulated functions from* $mod(\Sigma_1) \times \ldots \times mod(\Sigma_n)$ *to* $mod(\Sigma, \triangle\Gamma)$ *which are also independent implementations.*

Forget nodes:
if $\boldsymbol{f}$ *is of the form* $forget_\sigma \in$ **FORGET**, *with* $\sigma : \Sigma_2 \to \Sigma_1$, *then:*
if[7] $\forall M \in mod(\Sigma_1),\ \forall \varphi \in sen(\Sigma_2),\ (M \models \sigma(\varphi) \Leftrightarrow \mathcal{U}_\sigma(M) \models \varphi)$, *then* $\boldsymbol{Sem_f} = U_\sigma$, *else* $\boldsymbol{Sem_f} = \emptyset$

Heterogeneous filter nodes:
if $\boldsymbol{f}$ *is of the form* $filter_H \in$ **FILTER**, *with* $H = (\mu : i \to o, \Sigma, \boldsymbol{\mathcal{D}}_H)$, *then:*
if $\forall \boldsymbol{\rho} \in \boldsymbol{Sem}(\boldsymbol{\Theta}),\ \overline{\boldsymbol{\rho}}(\boldsymbol{\mathcal{D}}_H) \subset Dom(Ext_{\mu,\Sigma})$, *then* $\boldsymbol{Sem_f} = \{Ext_{\mu,\Sigma}\}$, *else* $\boldsymbol{Sem_f} = \emptyset$

4.4 Heterogeneous inferences relations for *BUFF*

Definition 20. *The heterogeneous inference relation of BUFF is* $\Vdash_{BUFF} = \coprod_{\boldsymbol{f} \in \boldsymbol{\mathcal{N}}_{BUFF}} \Vdash_{\boldsymbol{f}}$, *with*

Basic nodes:
if $\boldsymbol{f}$ *is of the form* $basic_P \in$ **BASIC**, *with* $P = (\Sigma, \Gamma)$, *then its heterogeneous inference relation is the introduction of the axioms of* Γ*:*
$\Vdash_{\boldsymbol{f}} = \{(\emptyset, \varphi) \mid \varphi \in \Gamma\}$

Use nodes:
if $\boldsymbol{f}$ *is of the form* $use_{\triangle P} \in$ **USE**, *with* $\triangle P = (\delta_1 : \Sigma_n \to \Sigma, \ldots, \delta_n : \Sigma_n \to \Sigma, \triangle\Gamma, \boldsymbol{\mathcal{D}}_{\triangle P})$, *then its heterogeneous inference relation is the introduction of the axioms of* $\triangle\Gamma$ *and the preservation of the properties of the imported specifications:*
$$\Vdash_{\boldsymbol{f}} = \{(\emptyset, \varphi) \mid \varphi \in \triangle\Gamma\} \cup \coprod_{i=1\ldots n} \{(\varphi, (sen(\delta_i))(\varphi)) \mid \varphi \in sen(\Sigma_i)\}$$

Forget nodes:
if $\boldsymbol{f}$ *is of the form* $forget_\sigma \in$ **FORGET**, *with* $\sigma : \Sigma_2 \to \Sigma_1$, *then its heterogeneous inference relation is the preservation of the properties of the imported specifications:*
$\Vdash_{\boldsymbol{f}} = \{((sen(\sigma))(\varphi), \varphi) \mid \varphi \in sen(\Sigma_2)\}$

Heterogeneous filter nodes:
if $\boldsymbol{f}$ *is of the form* $filter_H \in$ **FILTER**, *with* $H = (\mu : i \to o, \Sigma, \boldsymbol{\mathcal{D}}_H)$, *then its heterogeneous inference relation is* $\Vdash_{\boldsymbol{f}} = \Vdash_{\mu,\Sigma}$

Proposition 1. *With the previous notations, the* $use_{\Delta P}$ *nodes are* $\Vdash_{use_{\Delta P}}$*-encapsulated.*
(obvious)

[7] Of course, if the underlying formalism of σ fulfills the satisfaction condition, this property is always satisfied.

5 Conclusion

We have proposed a definition of structured heterogeneous formal specifications allowing to easily deal with implementation sharing. The idea is to see specifications as terms containing structuring "nodes" which are intended to be the structuring operations. Then, semantics of these nodes are in a natural way some functions transforming "imported" models to models of the resulting signature. This approach has the advantage to give a very unified view of the hierarchical structuring mechanisms. It allows us to manipulate them very easily, owning to the well established corpus on terms, substitutions, algebraic morphisms,... It will probably also be useful to deal with grouping of modules or with heterogeneous abstract implementation.

The gist of our article is nothing more, and nothing less: we have tried to exploit well known definitions on terms for the definition of structured heterogeneous specifications in general, and this article is a first proposal in this direction. 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. We will probably rather try to explore more deeply the meaning of several classical theorems on terms (rewriting, etc.) for the side of structured heterogeneous specifications.

References

[Abr95] J.R. Abrial. *The B-Book - Assigning Programs to Meanings*. 1995.

[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.

[BG80] R. Burstall and J. Goguen. The semantics of CLEAR, a specification language. In Springer, editor, *Proc. Advanced Course on Abstract Software Specifications, Berlin*, pages 292–332, 1980.

[CCJC+95] C. Cornes, J. Courant, Filliâtre J-C., G. Huet, P. Manoury, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. The Coq Proof Assistant, Reference Manual, version 5.10. Technical Report 177, INRIA, INRIA-Rocquencourt, 1995.

[DF93] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *FME'93, LNCS 670, Springer Verlag*, pages 268–284, 1993.

[DF94] R. K. Dong and Ph. G. Frankl. The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 3:39, 1994.

[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.

[Gau84] M Gaudel. First introduction to PLUSS. Technical report, LRI, Université de Paris-Sud, 1984.

[GG91] S. Garland and J.V. Guttag. A guide to LP, the larch prover. Technical Report 82, DEC-SRC, 1991.

[GH93] J.V. Guttag and J.J. Horning. LARCH:languages and tools for formal specifications. *Texts and Monographs in Computer Science, Springer-Verlag*, 1993. ISBN 0-387-94006-5/ISBN 3-540-94006-5.

[Gor88] M J C. Gordon. Hol: A proof generating system for higher-ordder logic. In G. Birtwisle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128, Kluwer, Dordrecht, The Netherlands, 1988.

[GS93] D. Garlan and M. Shaw. An introduction to software architecture. In Ambriola and Tortora, editors, *Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Co., 1993.

[GW96] M-C. Gaudel and J. Woodcock, editors. *FME'96: Industrial Benefit and Advances in Formal Methods*, Oxford, 1996. Springer, LNCS 1051.

[HT94] D. Harper, R. andd Sannella and A. Tarlecki. Structured theory presentations and logic representations. *Annals of Pure and Applied Logic*, 67:113–160, 1994.

[Jon90] C.B. Jones. *Systematic Sotfware Development Using VDM*. Prentice-Hall International, 2nd edition, 1990.

[Mar91] B. Marre. *Toward automatic test data set selection using Algebraic Specifications and Logic Programming*. Eigth International Conference on Logic Programming, ICLP'91, Paris, 25-28, MIT Press, 1991.

[Mes89] J. Meseguer. General logics. In North-Holland, editor, *Proc. Logic. Colloquium '87*, Amsterdam, 1989.

[PW92] D.E. Perry and A.L. Wolf. Foundations for the study of software architectures. *ACM SIGSOFT, Software Engineering Notes*, pages 40–52, 1992.

[Roq94] C. Roques. *Modularité dans les spécifications algébriques : théorie et applications*. PhD thesis, Université de Paris-Sud, 1994.

[Spi89] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, Hemel Hempstead, 1989.

[Tar95] A. Tarlecki. Moving between logical systems. In *Recent Trends in Data Type Specification, Oslo*, pages 478–502, 1995.

[VB95] F. Voisin and M. Bidoit. Modular algebraic specifications and the orientation of equations into rewrite rules. In *Recent Trends in Data Type Specification, Oslo*, pages 503–521, 1995.

[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.

[Wir94] M. Wirsing. Algebraic specification languages: An overview. In *Recents Trends in Data Type Specification, S. Margherita, Italy*, pages 81–116, 1994.