Modular Algebraic Specifications and the Orientation of Equations into Rewrite Rules *

Frédéric Voisin [1] and Michel Bidoit [2]

[1] C.N.R.S. U.R.A. 410 and Université de Paris-Sud,
L.R.I., Bât. 490, F-91405 Orsay Cedex, France
[2] C.N.R.S. U.R.A. 1327 and Ecole Normale Supérieure,
L.I.E.N.S., 45 rue d'Ulm, F-75230 Paris Cedex 05, France

Abstract. We consider the problem of orienting equations into rewrite rules in the framework of a modular algebraic specification language. When a specification is built from "modules", the orientation should neither be done globally for all modules, nor independently for each module. The termination ordering we define respects the modules boundaries. Equations of a module are partitioned into two sets: the ones that can be "statically" oriented, in a reusable way, and the ones with potential inter-modules conflicts which are handled in a "dynamic" phase.

1 Introduction

Specification and verification are two activities that are naturally related: one motivation for writing a formal specification, besides getting a deeper understanding of the application at hand, lies in the ability to formally derive some properties from it. Typical examples are the (partial) validation of a specification by verifying that some intended properties are actual consequences of the axiomatization or the proof of some self-consistency property.

When the validation activity is done with the help of automated tools, the axiomatization has usually to be customized - be made "executable" - to fit with the engine of the proof system and to be applied automatically in a safe way. A key part of this customization is the orientation of equations into rewrite rules, and the associated definition of an ordering on terms for proving that the resulting rewrite system is finitely terminating. Not any equation can be turned into a rewrite rule and we are often left with a set of remaining, non-oriented, equations (or more general formulas) to be applied under user guidance.

In this paper we consider "path ordering" methods for turning (conditional) equations into (conditional) rewrite rules, i.e. the definition of a partial ordering on the symbols of the signature that is extended into a partial, well-founded, ordering on terms such that each instance of a left-hand side of a rule is greater than the corresponding right-hand side. More specifically, we consider the most

* Work partially supported by ESPRIT Basic Research Working Group 6112 COMPASS-II and the C.N.R.S. GDR-PRC "Programmation".

classical of these methods, the *recursive path ordering* (*r.p.o.*) [3, 5] which probably offers the best tradeoff between the ability to turn equations into rewrite rules and the ease for the user to define the ordering.

1.1 Modular Orientation of Equations

Except for the smallest examples, an algebraic specification is not written as a flat enumeration of sorts, operations and axioms but is split into pieces - hereafter called "modules" - where each piece introduces part of the knowledge. Modules are glued together by "structuring primitives" (like enrichment, parameterization and instantiation) into a rooted acyclic graph.

We are therefore interested in finding a way to offer a "separate orientation" of modules similar to a separate compilation of modules in programming languages, such that the orderings and rewrite systems resulting from this separate orientation can be safely combined. Note that the problem of finding criteria for the "modular termination" of rewrite systems – given two rewrite systems $\mathcal{R}_1$ and $\mathcal{R}_2$ both being finitely terminating, for whatever reason, can we guarantee that $\mathcal{R}_1 \cup \mathcal{R}_2$ is also finitely terminating? – is not new (c.f. Section 7). However, in a specification framework, one does not start with a collection of finitely terminating rewrite systems but with a collection of sets of equations to orient and one builds at the same time a termination ordering Γ and the associated rewrite system $\mathcal{R}$. We have some flexibility w.r.t. which equations are oriented, and which are not. In practice we are often left with a set of remaining equations (or more general formulas) that are not oriented.

The problem we study is the following: Suppose we have a modular axiomatization, i.e. not a global set of equations but a collection of modules M_i, each with its own set of equations $\Phi(M_i)$. We want to "separately" orient each $\Phi(M_i)$ in a rewrite system $\mathcal{R}(M_i)$ and a set of remaining equations $\mathcal{F}(M_i)$ such that each $\mathcal{R}(M_i)$ is finitely terminating and any arbitrary union of the $\mathcal{R}(M_i)$ is finitely terminating. This generalizes to libraries of modules, some of which shared by several specifications: The separate orientation of the equations of a module must not be altered because we want to import it in another specification.

This modular orientation must work without syntactic restrictions on the axioms and not only for enrichment and sums of modules, but also with more complex structuring primitives like parameterization and instantiation. Since a modular orientation cannot come for free, we have some limitations: We propose to split the orientation task in two steps. A "static" orientation of some of the equations in a module yields a rewrite system that can be freely merged with the rewrite systems of the other modules from the specifications in which it appears. A "dynamic" orientation step handles the orientation of the remaining equations when a specification is actually used, for instance for proving some properties about it. The equations that have not been oriented statically are processed only when the exact context of their use is known, with the possibility to orient them differently, or not at all, depending on the proof at hand.

The work presented here is a first step towards the design of theorem-provers better fit to modular algebraic specifications, i.e. theorem-provers that would take advantage of the modular structure of a specification and reflect it in their internal structure and mechanisms. Such a system would be able to deal with structured blocks of axioms rather than with a flat axiomatization.

The paper is organized as follows: in Section 2 we describe a stylized specification language dealing only with enrichment and sums of modules. In Section 3 we define the "modular orientation" and its properties. Section 4 extends the results of Section 3 to the case of generic modules and their instantiations. In Section 5 we investigate the "dynamic" extension of a modular ordering. Finally, we compare our method with the standard approach in Section 6 and with the approach of "modularity" in the term rewriting community in Section 7.

2 The Specification Language

To describe our method we must fix the structuring primitives we consider. In this paper we select a stylized subset of the PLUSS specification language [2] with enrichment of a module with new sorts and operations, sum of modules,[3] parameterization and instantiation of generic specifications. With respect to PLUSS, we do not consider the "renaming" of sorts and operations since it has no effect on orientation. To avoid the additional technicalities needed for handling parameterization and instantiation, here we consider only enrichment and sums of modules, postponing the two other primitives to Section 4. We give a first abstract syntax for this language.[4] An attribute gives the nature of a module. Its value is always *spec*, but this will change in Section 4 when instantiation and parameterization will be considered. Finally, we assume that, for any specification context or specification library, names uniquely identify modules.

Definition 1 (Simplified abstract syntax of module declarations).
Each module associates a name with an expression built from the following cases:

$\mathbf{M} = \textit{\textbf{spec}}\ (\Delta\Sigma,\ \Delta\Phi)$	– basic specification
$\mathbf{M} = \textit{\textbf{enrich}}\ \mathbf{M1}\ \textit{\textbf{with spec}}\ (\Delta\Sigma,\ \Delta\Phi)$	– enrichment on top of M1
$\mathbf{M} = (\mathbf{M1} + \mathbf{M2},\ \Delta\Phi)$	– sum of modules

We introduce some notations to distinguish between modules and specifications, i.e. graphs of modules rooted by a given module. Modules will be denoted by $M_1, M_2, \ldots$, whereas the specification rooted by M will be denoted by $\mathcal{SP}(M)$. The function $\mathcal{I}$ maps a module to the set of modules it imports and $\mathcal{E}$ is its reflexive closure: $\mathcal{E}(M)$ is the set of modules whose axioms can be used for proving theorems w.r.t. $\mathcal{SP}(M)$. We note $\Sigma(M)$ the (global) signature of $\mathcal{SP}(M)$ and $\Delta\Sigma(M)$ the contribution of M to a global signature, i.e. the part of the signature that is declared in M. Similarly, $\Phi(M)$ and $\Delta\Phi(M)$ are the global axiomatization for $\mathcal{SP}(M)$ and the increment of axiomatization provided by M.

[3] Unlike most existing languages we allow the possibility to add axioms in a sum of modules, to relate operations from both modules.

[4] All the structuring primitives are subject to validity checks, not detailed here, to ensure that a specification is "well-formed", see e.g. [2].

Definition 2 (Semantic rules for the structuring primitives).
We define $\mathcal{I}(M)$, $\Sigma(M)$ and $\Phi(M)$ inductively as follows:

case M = ***spec*** **($\Delta\Sigma$, $\Delta\Phi$)**
$\mathcal{I}(\mathrm{M}) = \emptyset$, $\Sigma(\mathrm{M}) = \Delta\Sigma$, $\Phi(\mathrm{M}) = \Delta\Phi$,
case M = ***enrich*** **M1** ***with spec*** **($\Delta\Sigma$, $\Delta\Phi$)**
$\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{M1})$, $\Sigma(\mathrm{M}) = \Sigma(\mathrm{M1}) \cup \Delta\Sigma$, $\Phi(\mathrm{M}) = \Phi(\mathrm{M1}) \cup \Delta\Phi$,
case M = (M1 + M2, $\Delta\Phi$)
$\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{M1}) \cup \mathcal{E}(\mathrm{M2})$,
$\Sigma(\mathrm{M}) = \Sigma(\mathrm{M1}) \cup \Sigma(\mathrm{M2})$, $\Phi(\mathrm{M}) = \Phi(\mathrm{M1}) \cup \Phi(\mathrm{M2}) \cup \Delta\Phi$.

Modular specifications are built from pieces along some guidelines: not all specifications are "adequately structured". We will not go into the details of the semantics of modularity in specification languages, since most of it is about the models of such specifications, simply requesting here a syntactical criterion: Each operation must be declared exactly once in the specification in which it appears, and the same is asked for sorts. We put no syntactic restriction on the form of the axioms in a module: axioms in an importing module can arbitrarily reference imported operations, as long as the meaning of these operations does not change when the structuring primitive used for that module impose to protect imported operations. This kind of properties, a part of more global semantic constraints, are better left at the semantic level rather than being rigidly enforced by syntactic restrictions on the axioms. In this paper we consider only specifications that are "adequately structured" in the following meaning:

Definition 3. A modular specification $\mathcal{SP}(M)$ is "adequately structured" if for each function symbol f (resp. sort s) in $\Sigma(M)$ there is exactly one module M_i in $\mathcal{E}(M)$ such that f (resp. s) belongs to $\Delta\Sigma(M_i)$.

To the above productions and attributes for modules, we add three parts – Γ^s, $\Delta\mathcal{R}^s$, and $\Delta\mathcal{F}^s$ – to represent the orientation: $\Gamma^s(M)$ is the partial ordering over $\Sigma(M)$ used to prove the finite termination of the rewrite system $\Delta\mathcal{R}^s(M)$ using methods like $r.p.o$, and $\Delta\mathcal{F}^s(M)$ is the complement of $\Delta\mathcal{R}^s(M)$ in $\Delta\Phi(M)$. The superscript recalls that the attributes have been computed "statically", independently of any embedding specification context: $\Gamma^s(M)$ is an ordering that can be recorded with the module and reused in any context. In the next section we define how $\Gamma^s(M)$ can extend the orderings for the modules it imports, while fulfilling our modularity requirements. The sets $\Delta\mathcal{R}^s$ and $\Delta\Phi^s$ appear as derived products of Γ^s: respectively the (maximal) rewrite system obtained by orienting the equations in $\Delta\Phi$ according to Γ^s, and the set of remaining equations. There can exist several Γ^s satisfying our constraints, yielding different rewrite systems. Choosing among the possible Γ^s is not the problem addressed here. In practice, Γ^s is not fixed and $\Delta\mathcal{R}^s$ deduced from it, but both are computed incrementally and depend on the order in which equations are considered, as it is usually the case for existing theorem-provers (e.g. LP [8]) applied to flat axiomatizations.

3 Modular Orientation

Let $\mathcal{SP}(M)$ be a modular specification rooted by module M. The problem to be solved is to define an ordering for the orientation of the equations in $\Delta\Phi(M)$ in a way compatible with the modular structure of the specification. Note that we know the modules that M imports, and we can assume that these imported modules are already modularly oriented, i.e., for each M_i imported by M the ordering $\Gamma^s(M_i)$ is given. We will show that a crucial property of our definition of a modular orientation is that, since all M_i are sub-modules of the same adequately structured specification $\mathcal{SP}(M)$, the orderings $\Gamma^s(M_i)$ are compatible. Therefore, we can use their union to orient the equations of $\Delta\Phi(M)$. Clearly, this will not be enough, since we have to take care of the new symbols introduced by M. The question is therefore to decide how far we can extend these orderings ? A problem is that no assumption can be made about the context(s) in which M will be used: $\mathcal{SP}(M)$ can latter on be used as a sub-specification of several other specifications. For instance, let M' be another specification enriching a module M_i also imported by M, the sum $M + M'$ could reveal conflicting extensions of $\Gamma^s(M_i)$ by M and M'. The orientation of the equations in M must therefore be conservative with respect to the constraints put on imported symbols. However, when specifications are adequately structured, the problem is limited to the part shared by M and M': since $\Delta\Sigma(M)$ and $\Delta\Sigma(M')$ are disjoint, their parts of the ordering cannot interfere, except in the way they relate to $\Sigma(M_i)$. We therefore allow a module to use the ordering that already exists between the operations it imports, but we forbid the addition of new relations, directly or by transitivity, between imported operations.

Definition 4 (Modular orientation). Let $\mathcal{SP}(M)$ be an adequately structured modular specification. $\mathcal{SP}(M)$, or simply M, is said to be "modularly oriented" if:

1. All the modules in $\mathcal{I}(M)$ are recursively modularly oriented,
2. $\Gamma^s(M)$ is a relation over $\Sigma(M)$ s.t. $\Gamma^s(M)^+$ is an ordering relation,
3. In addition, $\Gamma^s(M)$ is defined according to the following rules:[5]
   case M = ***spec*** **($\Delta\Sigma$, $\Delta\Phi$)** : there is no further restriction on $\Gamma^s(\mathrm{M})$.
   case M = ***enrich*** **M1** ***with spec*** **($\Delta\Sigma$, $\Delta\Phi$)**
   $\Gamma^s(\mathrm{M}) =$ any Γ such that $\Gamma^+{}_{|\Sigma(\mathrm{M1})} = \Gamma^s(\mathrm{M1})^+$
   case M = (M1 + M2, $\Delta\Phi$) : $\Gamma^s(\mathrm{M}) = \Gamma^s(\mathrm{M1}) \cup \Gamma^s(\mathrm{M2})$,
4. $\Delta\mathcal{R}^s(M)$ is the part of $\Delta\Phi(M)$ that can be oriented into a finitely terminating rewrite system using $\Gamma^s(M)^+$.

Since there can exist several incomparable orderings – with respect to the inclusion of the corresponding rewrite systems – one cannot ask to get the "largest" ordering. We might ask an extension to be maximal in the sense that no equation that could be oriented by a modular ordering is left unoriented and the ordering contains no unnecessary relation.

[5] We note $f_{|D}$ the restriction of f to D.

The following propositions justify the "modular" qualification to our definition. Proofs are given in the Appendix.

Proposition 5. *Let $\mathcal{SP}(M)$ be an adequately structured modular specification. If all M_i in $\mathcal{I}(M)$ are modularly oriented, then $(\cup\Gamma^s(M_i))^+$ is an ordering relation. Hence M can also be modularly oriented.*

The next proposition is what we are looking for: if all modules are modularly oriented, the union of their associated rewrite systems is finitely terminating:

Proposition 6 (Static union of modules). *Let $\mathcal{SP}(M)$ be an adequately structured, modularly oriented, specification and let $\mathcal{C} \subseteq \mathcal{E}(M)$ be an arbitrary collection of modules of $\mathcal{SP}(M)$. Then $\cup\, \Delta\mathcal{R}^s(M_i)$, for all M_i in $\mathcal{C}$, is finitely terminating.*

4 Parameterization and Instantiation

We now consider parameterization and instantiation. We suppose that the parameter of a generic specification is a module of a special kind.[6] Parameter modules look like ordinary modules but need a specific semantic processing. An enrichment of a generic specification is again a generic specification, with the same formal parameter as the module it enriches (i.e. `G[P]` built on top of `F[P]`, where `P` is a parameter module). The instantiation of a generic specification is a standard specification when the actual parameter is a standard specification as in `F[P => A]`, or is again a generic specification when the actual parameter is a generic specification itself, as in `F[P => G[P']]`, or when it is a formal parameter, usually with additional features, as in `F[P => P']`. In both latter cases `P'` is the parameter of the resulting specification. The formal parameter and the generic module are not imported by the result of an instantiation. An attribute $\mathcal{N}$ records the "nature" of a module: $\mathcal{N}(M) \in \{$*fix, par, gen*[P]$\}$, where P is a *par* module. Since the same information is needed for operations and sorts, we define another attribute $\mathcal{K}$ as a function from the global signature of a module into the range of $\mathcal{N}$: The value of $\mathcal{K}$ for an operation is given by the nature $\mathcal{N}$ of its module of declaration, but changes for the instantiations of generic operations. Finally, we have to describe the effect of an instantiation. Remember that a fitting morphism is a signature morphism from the signature of the parameter module to the signature of the actual parameter that is the identity on non *par* operations or sorts. Given a fitting morphism σ, for each generic operation f the associated instantiation morphism, noted $\overline{\sigma}$, derives a new operation whose arity is obtained by replacing the *par* sorts in the arity of f by their images in σ, leaving *fix* objects unchanged.[7] Similarly, the equations,

[6] We allow a unique formal parameter to simplify the discussion.

[7] In this stylized language, adequate structuring disallows multiple instantiations of a given generic specification in a same module, since $\overline{\sigma}$ leaves "unchanged" – as far as names are concerned – all operations without a parameter sort in their arity. In the

rewrite rules and ordering relations of an instantiation are derived from their generic counterparts.

The controls (not detailed here) associated with the well-formedness of generic specifications forbid to build a sum of a parameter module and a non parameter module, and impose that both generic operands of a sum have the same parameter. Similarly, a generic module cannot be imported by a non generic module or by a generic module with a different parameter. Finally, a parameter module cannot be imported by something that is not a parameter module.

Notation: We note by $f \oplus g$ the function over the union of the domains of f and g that coincides with them on their respective domains, f and g being equal on the intersection of their domains. The use of that notation for combining $\mathcal{N}$ and $\mathcal{K}$ in the following definition is valid, thanks to the "adequate structuring" of the specifications and the above restrictions.

Definition 7 (Extended abstract syntax and semantic rules).
The attributes $\mathcal{I}$, Σ, Φ, $\mathcal{N}$ and $\mathcal{K}$ are inductively defined as follows:

case M = <Kind> ($\Delta\Sigma$, $\Delta\Phi$)
  if <Kind> = *spec* or *par*
    $\mathcal{N}$(M) = <**Kind**>, $\mathcal{K}(\mathrm{M}) = \{f \mapsto \mathcal{N}(\mathrm{M}) \ / \ f \in \Delta\Sigma\}$,
    $\mathcal{I}(\mathrm{M}) = \emptyset$, $\Sigma(\mathrm{M}) = \Delta\Sigma$, $\Phi(\mathrm{M}) = \Delta\Phi$
  if <Kind> = *gen*[P]
    $\mathcal{N}(\mathrm{M}) = gen[\mathrm{P}]$, $\mathcal{K}(\mathrm{M}) = \{f \mapsto \mathcal{N}(\mathrm{M}) \ / \ f \in \Delta\Sigma\} \oplus \mathcal{K}(\mathrm{P})$
    $\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{P})$, $\Sigma(\mathrm{M}) = \Delta\Sigma \cup \Sigma(\mathrm{P})$, $\Phi(\mathrm{M}) = \Delta\Phi \cup \Phi(\mathrm{P})$
case M = *enrich* M1 *with* <Kind> ($\Delta\Sigma$, $\Delta\Phi$)
  if <Kind> = *spec* or *par*
    $\mathcal{N}$(M) = <**Kind**>, $\mathcal{K}(\mathrm{M}) = \{f \mapsto \mathcal{N}(\mathrm{M}) \ / \ f \in \Delta\Sigma\} \oplus \mathcal{K}(\mathrm{M1})$,
    $\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{M1})$, $\Sigma(\mathrm{M}) = \Delta\Sigma \cup \Sigma(\mathrm{M1})$, $\Phi(\mathrm{M}) = \Delta\Phi \cup \Phi(\mathrm{M1})$
  if <Kind> = *gen*[P]
    $\mathcal{N}(\mathrm{M}) = gen[\mathrm{P}]$, $\mathcal{K}(\mathrm{M}) = \{f \mapsto \mathcal{N}(\mathrm{M}) \ / \ f \in \Delta\Sigma\} \oplus \mathcal{K}(\mathrm{M1}) \oplus \mathcal{K}(\mathrm{P})$,
    $\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{M1}) \cup \mathcal{E}(\mathrm{P})$, $\Sigma(\mathrm{M}) = \Delta\Sigma \cup \Sigma(\mathrm{M1}) \cup \Sigma(\mathrm{P})$,
    $\Phi(\mathrm{M}) = \Delta\Phi \cup \Phi(\mathrm{M1}) \cup \Phi(\mathrm{P})$
case M = (M1 + M2, $\Delta\Phi$)
  $\mathcal{N}$(M) is defined as follows:
  - if $\mathcal{N}(\mathrm{M1}) = gen[\mathrm{P}]$, $\mathcal{N}(\mathrm{M2}) = gen[\mathrm{Q}]$ then P = Q and $\mathcal{N}(\mathrm{M}) = gen[\mathrm{P}]$
  - if $\mathcal{N}(\mathrm{M1}) = gen[\mathrm{P}]$, $\mathcal{N}(\mathrm{M2}) = spec$ (or conversely), $\mathcal{N}(\mathrm{M}) = gen[\mathrm{P}]$
  - otherwise we must have $\mathcal{N}(\mathrm{M1}) = \mathcal{N}(\mathrm{M2})$ and $\mathcal{N}(\mathrm{M}) = \mathcal{N}(\mathrm{M1})$,
  $\mathcal{K}(\mathrm{M}) = \mathcal{K}(\mathrm{M1}) \oplus \mathcal{K}(\mathrm{M2})$, $\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{M1}) \cup \mathcal{E}(\mathrm{M2})$,
  $\Sigma(\mathrm{M}) = \Sigma(\mathrm{M1}) \cup \Sigma(\mathrm{M2})$, $\Phi(\mathrm{M}) = \Phi(\mathrm{M1}) \cup \Phi(\mathrm{M2}) \cup \Delta\Phi$.
<Kind> = *par* | *spec* | *gen*[P]

The abstract syntax is finally extended with a rule for instantiation. We explicitely define the increments $\Delta\Sigma$ and $\Delta\Phi$ for instantiations, while in Def. 7

PLUSS language, this is solved by appropriate combinations of instantiation with renaming. Since renaming is viewed as having no effect on the axioms [13], we will not consider this issue here.

their values are clear from the syntax. Hereafter, G_Σ and G_Φ (resp. F_Σ and F_Φ) retrieve the generic (resp. fixed) part of the signature and axiomatization of a generic specification.

Definition 8 (Instantiation). We extend Def. 7 with the following rule:

M = *inst* M1[M2 => M3] *via* σ
$\mathcal{I}(\mathrm{M}) = \mathcal{E}(\mathrm{M3}) \cup (\mathcal{I}(\mathrm{M1}) \setminus \mathcal{E}(\mathrm{M2}))$,[8]
$\mathcal{N}(\mathrm{M}) = \mathit{gen}[\mathrm{M3}]$ if $\mathcal{N}(\mathrm{M3}) = \mathit{par}$ and $\mathcal{N}(\mathrm{M3})$ otherwise,
$\Delta\Sigma(\mathrm{M}) = \overline{\sigma}(G_\Sigma(\mathrm{M1}))$, $\Sigma(\mathrm{M}) = \Sigma(\mathrm{M3}) \cup F_\Sigma(\mathrm{M1}) \cup \Delta\Sigma(\mathrm{M})$,
$\Delta\Phi(\mathrm{M}) = \overline{\sigma}(G_\Phi(\mathrm{M1}))$, $\Phi(\mathrm{M}) = \Phi(\mathrm{M3}) \cup F_\Phi(\mathrm{M1}) \cup \Delta\Phi(\mathrm{M})$,
$\mathcal{K}(\mathrm{M}) = \mathcal{K}(\mathrm{M3}) \oplus \{f \mapsto \mathit{fix} \ / \ f \in F_\Sigma(\mathrm{M1})\} \oplus \{\overline{\sigma}(f) \mapsto \mathcal{N}(\mathrm{M}) \ / \ f \in G_\Sigma(\mathrm{M1})\}$
where:
$G_\Sigma(M) = \{f \in \Sigma(M) \ / \ \mathcal{K}(M)(f) = \mathit{gen}\}$,
$G_\Phi(M) = \cup(\Delta\Phi(M_i))$, $\forall\ M_i \in \mathcal{E}(M)$: $\mathcal{N}(M_i) = \mathit{gen}$,
$F_\Sigma(M) = \Sigma(M) \setminus G_\Sigma(M)$, $F_\Phi(M) = \Phi(M) \setminus G_\Phi(M)$.

In an instantiation we somehow loose part of the modular structure: $\Delta\Phi$ gathers all the generic axioms into a unique piece: consider the example of `G[P]` built on top of `F[P]`, followed by an instantiation `G[P => P']`.

We now consider the problem of the modular orientation of the equations of a generic module. First, note that no assumption can be made about how an ordering relation between parameter operations will be reflected in the ordering relations between their images through fitting morphisms. We must define the ordering in such a way that the rewrite rules in a generic module can be kept as rewrite rules in any instantiation. We will derive by the instantiation morphism the relations for the instantiated operations from the relations on the generic operations and we must be careful not to rule out potential instantiations. This is the case with relations like $h < f < k$ where f is generic and h and k are parameter operations, since it is possible that in an instantiation h and k will be substituted by the same operation. The same situation occurs if one of h or k is a parameter and the other is imported. This generalizes to any use of a relation $h < k$ between a parameter operation h and an imported operation k – without the generic part playing any role – since we can always define a module with an operation h' and a relation $k < h'$ and use that module in an instantiation, substituting h' to h. As long as the relation between h and k is not used to orient a generic equation this is not harmful because the ordering for the parameter part will be discarded in the instantiation. Otherwise, in the instantiation we would have to add the corresponding relation for the image of the parameter operation, creating a cycle in the ordering. Hence, generic equations have to be oriented without relying on relations on parameter operations. We can extend the current ordering with relations between generic and parameter/fixed operations as long as we do not end up by transitivity in one of the previous cases. This is summarized in the following definition, which admits Def. 4 as a sub-case when all operations are *fix*.

[8] In a correct instantiation we must have: $\mathcal{I}(\mathrm{M2}) \subseteq \mathcal{I}(\mathrm{M3})$ except for the *par* modules: all `fix` modules imported by M2 must also be imported by M3.

Definition 9 (Modular orientation). Let $\mathcal{SP}(M)$ be an adequately structured modular specification. $\mathcal{SP}(M)$, or simply M, is said to be "modularly oriented" if:

1. All the modules in $\mathcal{I}(M)$ are recursively modularly oriented,
2. $\Gamma^s(M)$ is a relation over $\Sigma(M)$ s.t. $\Gamma^s(M)^+$ is an ordering relation,
3. $\Gamma^s(M)$ is defined according to the following rules:
   case $\mathbf{M} = \mathcal{N}(\mathbf{M})(\Delta\Sigma,\ \Delta\Phi)$ and
   case $\mathbf{M} =$ ***enrich*** **M1** ***with*** $\mathcal{N}(\mathbf{M})$ $(\Delta\Sigma,\ \Delta\Phi)$
   (a) if $\mathcal{N}(\mathrm{M}) \in \{par,\ spec\}$: $\Gamma^s(\mathrm{M}) =$ any Γ s. t. $\Gamma^+{}_{|\Sigma(\mathrm{M1})} = \Gamma^s(\mathrm{M1})^+$,
   (b) if $\mathcal{N}(\mathrm{M}) = gen[\mathrm{P}]$: $\Gamma^s(\mathrm{M}) =$ any Γ s. t.[9]
      i. $\Gamma^+{}_{|\Sigma(\mathrm{P})} = \Gamma^s(\mathrm{P})^+$, $\Gamma^+{}_{|\Sigma(\mathrm{M1})} = \Gamma^s(\mathrm{M1})^+$
      ii. $\Gamma^+{}_{|\Sigma(\mathrm{P+M1})} = (\Gamma^s(P) \cup \Gamma^s(\mathrm{M1}))^+$
      iii. $\forall f, h, k : h < f < k \in \Gamma^+$:
      $\mathcal{K}(f) = gen \Rightarrow (\mathcal{K}(h),\ \mathcal{K}(k)) \notin \{(par,\ par),\ (fix,\ par),\ (par,\ fix)\}$
      iv. $\forall\ (f < g)$ in the new part of Γ^s: $\mathcal{K}(f)\ = gen \vee \mathcal{K}(g) = gen$
   case $\mathbf{M} =$ **(M1 + M2,** $\Delta\Phi$**)**
   $\Gamma^s(\mathrm{M}) = \Gamma^s(\mathrm{M1}) \cup \Gamma^s(\mathrm{M2})$
   case $\mathbf{M} =$ ***inst*** **M1[M2** `=>` **M3]** ***via*** σ
   $\Gamma^s(\mathrm{M}) = \Gamma^s(\mathrm{M3}) \cup \overline{\sigma}(\Gamma^s(\mathrm{M1}) \setminus \Gamma^s(\mathrm{M2}))$ (*)
4. $\Delta\mathcal{R}^s(M)$ is the part of $\Delta\Phi(M)$ that can be oriented into a finitely terminating rewrite system using $\Gamma^s(M)^+$ when $\mathcal{N}(M) \neq gen[\mathrm{P}]$ and otherwise using $\Gamma^s(M)^+ \setminus \{(h,k) \in \Gamma^s(\mathrm{P})$ s.t. $\mathcal{K}(h) = par \vee \mathcal{K}(k) = par\}$.

Relations on the parameter operations are kept in the ordering of a generic module M so that the rewrite system for the parameter is provable to be finitely terminating with $\Gamma^s(M)$, giving the possibility to use rules from the parameter for proving properties about the generic specification. An instantiation looks like a sum of modules, modulo the morphism $\overline{\sigma}$, except that the ordering on the parameter part is discarded, but Def. 9.4 guarantees that a relation that will not exist in the instantiation has not been used for orienting a rewrite rule derived by the instantiation.

Proposition 10. *Using Definition 9, Propositions 5 and 6 extend to the case of generic specifications and their instantiations.*

One may wonder whether the restrictions for generic modules are not too severe to be of any use. First note that many generic specifications have a very simple parameter with a unique sort and no operation, for which the restriction is void. This is the case of various sorts of containers: bags, sequences,... We can also deal with more elaborate specifications like the ones of "ordered containers" with axioms like `x < y = true => insert(x, cons(y, v)) = cons(x, cons(y, v))` whose orientation with *r.p.o.* only asks for the comparison of the formal operation `<` to the generic operation `insert`, something allowed by our

[9] Note that items (i.) and (ii.) are the same as if M were an enrichment over M1 + P.

definition. Note that when *par* operations are substituted by *fix* ones, some equations that cannot be modularly oriented in the generic specification could be oriented with the ordering for a particular instantiation. To associate an additional orientation step with instantiation that benefits from the information provided by the fitting morphism, replace the equality symbol = by the containment operator $\supseteq$ in line (*) of Def. 9.

5 Dynamic Extension of Modular Orderings

In a modular specification, a proof typically uses only some of the modules (see [11, 15] for earlier works trying to use the structure of a specification for theorem-proving purposes), although often these modules do not make a subspecification: it is rather a slice of a specification, "parameterized" by the lower modules that are not needed for this proof. For a formula to be proved, on the basis of the operations it includes we can gather a collection of modules whose axioms will most probably be needed, but this collection of modules will usually vary during the proof, according to the proof commands issued by the user. Assume we start with only a subset of the modules of a specification, merge their rewrite systems and modular orderings. Then we orient some additional equations. Are we able to dynamically add during the proof a still unloaded module from the specification ?

More precisely: Assume we know the specification $\mathcal{SP}(M)$ w.r.t. which a formula must be proved. Then $\mathcal{E}(M)$ is an upper bound of the set of modules that can be used for that proof. Starting with some $\mathcal{C} \subseteq \mathcal{E}(M)$, we can have a more aggressive orientation procedure for the remaining equations in the modules in $\mathcal{C}$, without making provision for a later extension by a module outside $\mathcal{E}(M)$. Hence, the union of all $\Delta\mathcal{F}^s(M_j)$, for M_j in $\mathcal{C}$, is further split into $\mathcal{R}^d_\mathcal{C}$ and $\mathcal{F}^d_\mathcal{C}$ according to an extension $\Gamma^d_\mathcal{C}$ of $\Gamma^s_\mathcal{C} = \cup\, \Gamma^s(M_j)$, for all $M_j \in \mathcal{C}$. Should this be needed, we can still later add the rewrite system and ordering of any module in $\mathcal{E}(M) \setminus \mathcal{C}$ and start orienting its equations. We can now freely play with the modules from $\mathcal{SP}(M)$. This is summarized in the following proposition:

Proposition 11 (Dynamic addition of modules). *Let $\mathcal{SP}(M)$ be a modularly oriented specification and let $\mathcal{C} \subseteq \mathcal{E}(M)$ be an arbitrary subset of modules. Let $\Gamma^s_\mathcal{C} = \cup\, \Gamma^s(M_i)$, $\mathcal{F}^s_\mathcal{C} = \cup\, \Delta\mathcal{F}^s(M_i)$, $\mathcal{R}^s_\mathcal{C} = \cup\, \Delta\mathcal{R}^s(M_i)$ and $\Sigma_\mathcal{C} = \cup\, \Sigma(M_i)$, where in all unions M_i ranges over $\mathcal{C}$. Let $\Gamma^d_\mathcal{C}$ be an extension over $\Sigma_\mathcal{C}$ of $\Gamma^s_\mathcal{C}$ such that $\Gamma^{d+}_\mathcal{C}$ is an ordering relation, and $\mathcal{R}^d_\mathcal{C}$ be any partition of $\mathcal{F}^s_\mathcal{C}$ into $(\mathcal{R}^d_\mathcal{C}, \mathcal{F}^d_\mathcal{C})$ made according to $\Gamma^d_\mathcal{C}$. Then for any module M_j in $\mathcal{E}(M) \backslash \mathcal{C}$ we have:*

1. *$(\Gamma^s(M_j) \cup \Gamma^d_\mathcal{C})^+$ is an ordering relation*
2. *$\Delta\mathcal{R}^s(M_j) \cup \mathcal{R}^s_\mathcal{C} \cup \mathcal{R}^d_\mathcal{C}$ is finitely terminating.*

6 Monolithic versus Modular Ordering

How does our approach compare with the monolithic approach in which the equations of a specification are considered globally for the orientation ? Starting with a set of user's indications about the ordering, one considers each equation in some unspecified order and dynamically extends the current ordering with the relations needed for the orientation of the new equation, leaving it unoriented if no such relations can extend the current ordering. There are two sources of "variability" for the result: first the resulting ordering and rewrite system depend on the order in which the equations are considered, since usually we do not guess the ordering in advance but we defined it incrementally. Second, there can exist several extensions to orient the current equation.

It is obvious that the modular method cannot be more "powerful" than the monolithic one, since the former is somehow a restriction of the latter. Conversely, not any monolithic ordering can be obtained by the modular approach since an equation can be ordered with the monolithic approach while being postponed by the modular approach and considered again during the dynamic phase. In the meantime we may have made an extension that will forbid the orientation of this equation. For instance, let M_1 be a module with constants `a, b, c, d` and no axiom, and M_2 declare two operations `f` and `g` and the following four axioms, in which the variable `x` forces a left-to-right orientation:

{`f(c,x) = g(c,c,c), g(b,c,x) = f(a,a), f(b,x) = c, f(d,x) = b`}

With *r.p.o.* the first axiom yields `f > g` while the second one can be oriented using either `b > f` and `b > a`, or `c > f` and `c > a`, both extensions being non modular. The third (resp. fourth) equation can be oriented with `f > c` or `b > c` (resp. `f > b` or `d > b`). With the monolithic approach both possibilities for the second equation can be extended into an ordering for the four equations. With the modular approach the orientation of the second axiom is left to the dynamic phase, after the static orientation of the last two axioms, with the unique modular ordering `f > c` and `f > b`. Hence, only three equations are statically oriented, with an ordering that forbid the dynamic orientation of the last equation.

However this formal comparison has little practical value since it ignores the problem of predicting the right order for processing equations or of choosing the right extension. The problem may occur with the monolithic approach if the second equation was put last in the example: we could as easily orient only three equations. Besides, even if we order all the axioms, this puts constraints on `a`, `b` and `c` that can conflict with independent enrichments of M_1, possibly later gathered with M_2. The hypothetical advantage of the monolithic approach is valid only as long as the set Φ of axioms to orient is fixed, but in practice Φ is varying since it can be potentially embedded in larger contexts.

7 Comparison with other "Modular" Approaches

The problem of the "modular termination" of rewriting systems is an active area of research in the term rewriting community for several years [16]. The question has there a somewhat different meaning: given two rewrite systems, each known to be finitely terminating, for whatever reason, find some syntactical criterion that will guarantee that their sum is also terminating. Different unions have been considered, for instance "disjoint", "combined", "composable" or "hierarchical", and there are various criteria to be met by the individual systems for the global termination to hold. Surveys can be found in [4, 6, 12, 10]. When the criteria do not hold, termination cannot be handled separately for each sub-system and must be tackled globally, loosing the modular structure. At first sight, this approach can be thought of as more general than ours since it is not restricted to path ordering methods or "simple termination" [10]: for some of these results no hint is needed about why each system is terminating, although most positive answers hold only for simple termination. In the latter case, some of the proofs are not constructive, i.e. they do not explicitly provide a simplification ordering that proves the termination. Another difference is that the proof of termination for each system is done independently while in our setting we prove the termination of a system using the information provided for the termination proof of the modules on which it is based. With the exception of the work described in [14] that, up to now, deals only with disjoint unions, a major restriction of these methods is that they are based on a syntactic distinction between "constructors" and "defined functions" and consequently on *syntactic restrictions* on the axioms. At a given level of a hierarchy of modules, a "constructor" is an operation symbol that does not appear at the root of the left-hand side of a rule of the current rewrite system. A defined operation is one that is not a constructor. Depending on the approach there are restrictions on the way modules share the operations: for instance the set of defined operations of a module must be disjoint from the set of constructors of another module, or from the set of defined operations, or the system must be "alien-decreasing" [7].

Although "constructor-based" methods form a well-established methodology in algebraic specifications, they are not always pertinent. Being at the top of a new rule does not mean being more defined ! We illustrate that point with two examples: First, when specifying a system, the ease of writing axioms should be favored over executability aspects, yielding more abstract descriptions. This is the case with the modules in Fig. 1, from the *Larch Shared Language Handbook* [9], where `sets`, generated by `{ }` and `ins`, are specified with the membership operator $\in$ and the set extensionality property. Additional operations are then defined in a subsequent module using $\in$ at the root of the left-hand side of the new axioms. Note that the new axioms can be modularly oriented, when boolean operators are minimal in the ordering. The second point is that realistic specifications are made not only with "constructive" definitions for operations but also with additional properties about them. This is especially true in theorem-proving – maybe less in applications closer to functional programming – where an initial axiomatization is enriched with theorems that ease subsequent proofs and for which the "no constructor at the root" rule is too restrictive. One does not want to do every proof with only a kernel of axioms even when the additional axioms are (inductive) consequences of these axioms. This is illustrated by the two modules in Fig. 2. The first module defines standard operations on sequences with the classical axiomatization based on the constructors `cons` and `nil`. The second module adds two operations `rev` and `last`. In addition to the two classical axioms for `rev`, why could we not add the axiom `not(empty(s))`

```
SetBasics(E, C): trait
introduces
{ } : → C;
ins: E, C → C
__ ∈ __ : E, C → Bool;
asserts
C generated by  { }, ins;
C partitioned by ∈;
– i.e. (∀ e: e ∈ c1 = e ∈ c2) ⇒ c1 = c2
e ∈ { } = false;
e1 ∈ ins(e2, c) = (e1 = e2) ∨ (e1 ∈ c)
```

```
Set(E, C): trait M2
includes SetBasics
introduces
__ ∪ __: C, C → C
del: E, C → C
asserts
e ∈(c1 ∪ c2) = (e ∈ c1) ∨ (e ∈ c2);
e1 ∈ del(e2, c) = (e1 ≠e2) ∧ (e1 ∈ c)
```

Fig. 1. Sets in L.S.L.

```
generic spec: Seq[ELEM]
sort: Seq
generated by:
 nil: → Seq
 cons: Elem * Seq → Seq
operations:
 empty: Seq → bool
 hd: Seq → Elem
 tl: Seq → Seq
 _@_: Seq * Seq → Seq
axioms:
 hd(cons(e, s)) = e
 tl(cons(e, s)) = s
 nil @ s = s
 cons(e, s) @ t = cons(e, s @ t)
where: e: Elem; s, t: Seq;
```

```
generic spec: ExtendedSeq[ELEM]
use: Seq[ELEM]
 rev: Seq → Seq
 last: Seq → Elem
axioms:
 last(cons(e, nil)) = e
 not empty(s) => last(cons(e, s)) = last(s)
 rev(nil) = nil
 rev(cons(e, s)) = rev(s) @ cons(e, nil)
 not empty(s) => hd(rev(s)) = last(s)
where: e: Elem; s: Seq;
```

Fig. 2. Sequences in PLUSS

`=> hd(rev(s)) = last(s)` ? As a conditional equation, it has the defined operation `hd` from `Seq` at the root of an axiom of `ExtendedSeq`, something incompatible with standard criteria but neither incompatible with "modularity" (semantically) nor with termination. The two modules neither form a "hierarchical union" [4] since `hd` is "defined" in both modules, nor are they "composable" [12], since each module includes only part of the axioms for `hd`.

Could our approach be extended to simplification orderings beyond path ordering methods ? A candidate might be the "polynomial interpretation" method, c.f. [1] for instance, which is often used to handle associative-commutative operations. For each k-ary operation a polynomial (or a sequence of polynomials) over k variables is given and such interpretations of operations are then extended to terms. However, this method is intrinsically global since it relies not on a relative order between symbols but on a fixed interpretation. An interpretation of the operations that works for the equations in a module will not always work for the equations in another module, although there might exist a third interpretation that would work for both. A modular orientation should guarantee that already oriented equations are still valid and can be combined directly, perhaps at the expense of not orienting some of the new axioms. Without syntactic restrictions on the axioms, the only choice would be to disallow a module to change or extend the interpretation of imported operations, but this may be too severe to be useful. Anyway, even with the monolithic approach the polynomial interpretation method is known to be difficult to apply, and few results for AC operations also exist with the general approach of the rewriting community [7, 14].

8 Conclusion

We have presented a method for handling the orientation of equations into rewrite rules in a way compatible with the modularity properties of algebraic specification languages. Our method does not rely on syntactic restrictions on the axioms that, in practice, cause problems when dealing with complex specifications. It can also handle structuring primitives like parameterization and instantiation that have not yet been really considered in the term rewriting community.

Our method is adapted to a "separate compilation" of modules – in a way similar to what is done for programming modules – by distinguishing between the equations that can be statically oriented, in the sense that the resulting rewrite system can be recorded within the module and reused directly in any valid context, and the equations whose orientation presents potential inter-modules orientation conflicts. The orientation of the latter equations is postponed until the exact context of their use is known – at "binding" time in the terminology of compilers – when modularity constraints no longer matter. The termination ordering we have defined respects the modules boundaries, i.e. it cannot put additional constraints on operations imported from other modules. This ensures that the static parts of a collection of modules can always be safely combined.

This work is a first step towards the design of "modular" theorem-provers, which would reflect and benefit from the modular aspects of specifications. The modular structure of a specification should have an equivalent in the logical representation used by the system and blocks of axioms should be more easily combined than it is the case with existing systems. This work can also ease the design of libraries of modules, in which part of the equations are statically oriented into rewrite rules, in a safe and reusable way.

References

1. A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137 – 159, 1987.

2. M. Bidoit. *Pluss, un langage pour le développement de spécifications algébriques modulaires.* Thèse d'Etat, Université de Paris Sud, 1989.
3. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.
4. N. Dershowitz. Hierarchical termination. In *Workshop on C. T. R. S.*, pages 89 – 105, Jerusalem, Israel, 1994. LNCS 968, Springer Verlag.
5. N. Dershowitz and J.-P. Jouannaud. *Rewrite Systems*, chapter 6, pages 243 – 320. Elsevier, Amsterdam, 1990.
6. M. Fernandez. *Modèles de calcul multi-paradigmes fondés sur la réécriture.* Thèse de doctorat, Université de Paris Sud, 1993.
7. M. Fernandez and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In *Recent Trend in Data Type Specification, 10th Workshop on Specification of Abstract Data Types, joint with the 5th Compass Workshop*, pages 255 – 272, S. Margherita, Italy, 1995. LNCS 906, Springer Verlag.
8. S. Garland and J. V. Guttag. A Guide to LP, The Larch Prover. Technical Report 82, DEC-SRC, 1991.
9. J. V. Guttag and J. J. Horning. *Larch: Languages and Tools for Formal Specification.* Texts and Monographs in Computer Science. Springer-Verlag, 1993. ISBN 0-387-94006-5/ISBN 3-540-94006-5.
10. M.R.K. Krishna Rao. Simple termination of hierarchical combinations of term rewriting systems. In *11th Symposium on Theoretical Aspects of Computer Science*, pages 203 – 223. LNCS 789, Springer Verlag, 1994.
11. R. Nakajima, M. Honda, and Nakahara R. Hierarchical program specification and verification - a many-sorted logical approach. *Acta Informatica*, 14:135–155, 1980.
12. E. Ohlebusch. *Modular Properties of Composable Term Rewriting Systems.* PhD thesis, Universität Bielefeld, Germany, 1994.
13. C. Roques. *Modularité dans les spécifications algébriques: Théorie et Applications.* Thèse de doctorat, Université de Paris Sud, 1994.
14. A. Rubio. Extension orderings. In *Proc. of the 22th ICALP*, Szeged, Hungary, 1995.
15. D.T. Sannella and R. M. Burstall. Structured theories in l.c.f. In *C.A.A.P.*, pages 377 – 391, 1983.
16. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Inf. Process. Lett.*, 25:141–143, 1987.

A Appendix: Proofs

In this section, we note $f \stackrel{\Gamma}{\to} g$ a relation between f and g in Γ, M_f the module – unique in a given specification – that declares the operation f and Γ^s_f – instead of $\Gamma^s(M_f)$ – its associated modular ordering. Since the importation relation defines an ordering between modules, we write $M_1 \leq M_2$ instead of $M_1 \in \mathcal{E}(M_2)$.

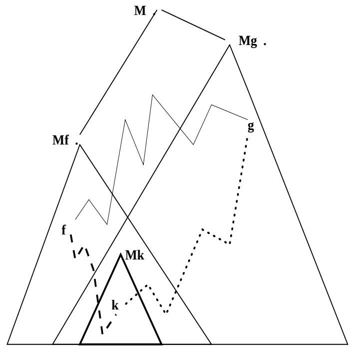
We first state a fact that can be checked by inspection in Def. 9 : A module always knows the ordering relations of the modules it imports.

Fact 1 *Let M_1 and M_2 be two modules:*

- $M_1 \leq M \Rightarrow \Gamma^s(M_1) \subseteq \Gamma^s(M)$

– *if* $\mathcal{N}(M_1) = \text{gen}[M_2]$ *and* $M = \text{inst } M_1[M_2 \Rightarrow M_3]$ via σ *then* $\overline{\sigma}(\Gamma^s(M_1)) \subseteq \Gamma^s(M)$, *and* $\Gamma^s(M_3) \subseteq \Gamma^s(M)$.

We establish a first result for specifications without instantiations: Generic modules are allowed but cannot yet be instantiated.



A module cannot add relations between imported operations. If there exists a chain between f and g within $\Gamma^s(M)$, we can always find a chain between f and g via an operation k in a module shared by $\mathcal{SP}(M_f)$ and $\mathcal{SP}(M_g)$. The chain between f and k (resp. k and g) is entirely within Γ^s_f (resp. Γ^s_g). These two chains will therefore exist in any module importing both M_f and M_g, whether a sub-module of M or not.

Lemma 1 *Let M be a modularly oriented module without any instantiation, and f, g two operations in $\Sigma(M)$, s. t. $f \xrightarrow{\Gamma^s(M)} {}^+ g$, then:*

1. *There exists an operation k s.t. $M_k \leq M_f$, $M_k \leq M_g$ and $f \xrightarrow{\Gamma^s_f} {}^* k \xrightarrow{\Gamma^s_g} {}^* g$.*
2. *For any module M_1 with $M_f \leq M_1$ and $M_g \leq M_1$: $f \xrightarrow{\Gamma^s(M_1)} {}^+ g$.*

Proof. Lemma 1.2 is a consequence of Lemma 1.1 and Fact 1. Lemma 1.1 is proved by induction on the structure of M, where the base step corresponds to empty sets of imported modules. We have three cases to consider:

Cases $M = \mathcal{N}(\text{M})(\Delta\Sigma, \Delta\Phi)$ and $M =$ *enrich* M1 *with* $\mathcal{N}(\text{M})(\Delta\Sigma, \Delta\Phi)$:
We combine these two cases because for generic modules the distinction between the parameter and imported modules is quite arbitrary. As far as ordering is concerned (cf. Def. 9.3) a module, generic or not, can be viewed as an enrichment over a module M_i that is the "sum" of its parameter and imported module, both being possibly empty.

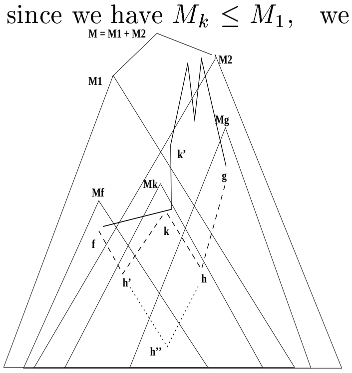
If the enrichment is empty then $M_f = M_g = M$ and $\Gamma^s_f = \Gamma^s_g = \Gamma^s(M)$: take $k = g$. Otherwise, if one of the operation, say f, is declared in $\Delta\Sigma$, i.e. $M = M_f$, we have $\Gamma^s_f = \Gamma^s(M)$ and $M_g \leq M_f$: take $k = g$. If neither f nor g is in $\Delta\Sigma$, then $f \xrightarrow{\Gamma^s(M)} {}^+ g$ iff $f \xrightarrow{\Gamma^s(M_i)} {}^+ g$ and the result is proved by applying the induction hypothesis to M_i.

Case $M = (M_1 + M_2, \Delta\Phi)$**:**
Recall that $\Gamma^s(M) = \Gamma^s(M_1) \cup \Gamma^s(M_2)$. Proceed by induction on the number n of alternations of relations from $\Gamma^s(M_1)$ and $\Gamma^s(M_2)$ used in $f \xrightarrow{\Gamma^s(M)} {}^+ g$. If $n = 0$ the result holds by applying the structural induction hypothesis either on M_1 or M_2, depending on the ordering that is used. If $n \geq 1$, assume for instance that $f \xrightarrow{\Gamma^s(M)} {}^+ g$ starts with a relation from $\Gamma^s(M_1)$ and consider the maximal homogeneous chain in $\Gamma^s(M_1)$: there exists k with $M_k \leq M_1$,

$M_k \le M_2$ – since k is known both in $\Gamma^s(M_1)$ and $\Gamma^s(M_2)$ – and k' in $\Sigma(M_2)$ such that $f \xrightarrow{\Gamma^s(M_1)}{}^+ k \xrightarrow{\Gamma^s(M_2)} k' \xrightarrow{\Gamma^s(M)}{}^+ g$. Note that we have $\Gamma^s_k \subseteq \Gamma^s(M_1)$ and $\Gamma^s_k \subseteq \Gamma^s(M_2)$. Since the chain $k \xrightarrow{\Gamma^s(M)}{}^+ g$ contains $n-1$ alternations, we apply the recurrence hypothesis on it: there exists h, with $M_h \le M_k$ and $M_h \le M_g$ such that $k \xrightarrow{\Gamma^s_k}{}^* h \xrightarrow{\Gamma^s_g}{}^* g$ (in dashed line in the figure). Now, by applying the structural induction hypothesis on M_1, we get an operation h', with $M_{h'} \le M_f$ and $M_{h'} \le M_k$ such that $f \xrightarrow{\Gamma^s_f}{}^* h' \xrightarrow{\Gamma^s_k}{}^* k$ (in dashed line also). If $h' = h$, the result is proved by taking $k = h = h'$. Otherwise, since we have $M_k \le M_1$, we can apply again the structural induction hypothesis now to the sub-chain $h' \xrightarrow{\Gamma^s_k}{}^* k \xrightarrow{\Gamma^s_k}{}^* h$ to get an operation h'', $M_{h''} \le M_{h'}$ and $M_{h''} \le M_h$, with $h' \xrightarrow{\Gamma^s_{h'}}{}^* h'' \xrightarrow{\Gamma^s_h}{}^* h$ (a dotted line). To summarize, we have a chain

$$f \xrightarrow{\Gamma^s_f}{}^* h' \xrightarrow{\Gamma^s_{h'}}{}^* h'' \xrightarrow{\Gamma^s_h}{}^* h \xrightarrow{\Gamma^s_g}{}^* g\ .$$

But $M_{h'} \le M_f \Rightarrow \Gamma^s_{h'} \subseteq \Gamma^s_f$ and $M_h \le M_g \Rightarrow \Gamma^s_h \subseteq \Gamma^s_g$, and h'' is the operation we are looking for. □



We now consider instantiations and prove that they behave like modularly oriented sums of modules:

Let $M = \mathit{inst}\ M_1[M_2 \Rightarrow M_3]\ \mathit{via}\ \sigma$, let M_0 be the module imported by M_1 and suppose that the only instantiation is the one at M. Without loss of generality we can assume that M_0 is not generic. Recall that $\Gamma^s(M) = \Gamma^s(M_3) \cup \overline{\sigma}(\Gamma^s(M_1) \setminus \Gamma^s(M_2))$. We decompose the latter term into $\Gamma^s(M_0) \cup \Delta\Gamma(M_1)$ where $\Delta\Gamma(M_1) = \{(\overline{\sigma}(h), \overline{\sigma}(k)\ /\ (h,k) \in \Gamma^s(M_1) : \mathcal{K}(M_1)(h) = \mathit{gen} \vee \mathcal{K}(M_1)(k) = \mathit{gen}\}$.

For M as above, let $\overline{M} = \mathit{enrich}\ (M_0 + M_3)\ \mathit{with}\ \mathcal{N}(M_3)(\Delta\Sigma(M),\ \Delta\Phi(M))$ where $\Delta\Sigma$ and $\Delta\Phi$ are as in Def. 7. Using Def. 7 and Def. 8 it is clear that M and $\overline{M}$ have the same values for the attributes $\mathcal{I}$, Σ, Φ and $\mathcal{K}$.

Lemma 2 *Let $\overline{\Gamma} = \Gamma^s(M_0 + M_3) \cup \Delta\Gamma(M_1)$. Then*

1. *$\overline{\Gamma} = \Gamma^s(M)$,*
2. *$\overline{\Gamma}$ is a modular orientation for $\overline{M}$.*

Proof. The first part is clear using $\Gamma^s(M_0 + M_3) = \Gamma^s(M_0) \cup \Gamma^s(M_3)$. For the second part, since M_0 and M_3 are modularly oriented, so is their sum. Hence we simply have to check that the enrichment is also modular, i.e.

$$\overline{\Gamma}^+{}_{|\Sigma(M_0+M_3)} = \Gamma^s(M_0 + M_3)^+\ .$$

It is obvious that the first set includes the second one. For the opposite inclusion, consider any chain $f \rightarrow^+ g$ in $\overline{\Gamma}$, where both f and g are in $\Sigma(M_0 + M_3)$. We prove by induction on the number n of $\Delta\Gamma(M_1)$ steps in that chain that $f \rightarrow^+ g$ already exist in $\Gamma^s(M_0 + M_3)$.

case $n = 0$**:** the result is obvious.

case $n > 0$**:** consider the maximal homogeneous sub-parts in $\Gamma^s(M_0 + M_3)$ at both ends: there exist f', g' in $\Sigma(M_0 + M_3)$ and h a generic operation in $\Delta\Sigma(M_1)$ s.t.

$$f \overset{\Gamma^s(M_0+M_3)}{\longrightarrow}{}^* f' \overset{\overline{\Gamma}}{\longrightarrow}{}^* \overline{\sigma}(h) \overset{\overline{\Gamma}}{\longrightarrow}{}^* g' \overset{\Gamma^s(M_0+M_3)}{\longrightarrow}{}^* g$$

where the middle part has n $\Delta\Gamma(M_1)$ steps, and every such step involves an instantiated operation $\overline{\sigma}(h')$ because of Def 3(b)iv .

case $n = 1$**:** this case is void since it requests either $f' = \overline{\sigma}(h)$ or $\overline{\sigma}(h) = g'$ when both operations must belong to $\Sigma(M_0 + M_3)$.

case $n = 2$**:** We have $f' \overset{\Delta\Gamma(M_1)}{\longrightarrow} \overline{\sigma}(h) \overset{\Delta\Gamma(M_1)}{\longrightarrow} g'$. In that case we have either $f' \to h$ in $\Gamma^s(M_1)$, or $f' = \sigma(p)$, for p a *par* operation with $p \to h$ in $\Gamma^s(M_1)$. The situation is similar for g'. Of the four combinations for f' and g', the only one not ruled out by Def.3(b)iii is when $f' \to h \to g'$ is in $\Gamma^s(M_1)$. Then f' and g' are in $\Sigma(M_0 + M_2)$. Since M_1 has been modularly oriented, the chain must exist in $\Gamma^s(M_0) \cup \Gamma^s(M_2)$. Since both operations are *fix* they cannot be declared in the parameter itself and must come either entirely from M_0 or from a module imported in M_2. Using Lemma 1 the chain must come through a module shared by M_0 and the module imported by M_2. Since all non *par* modules imported by M_2 are imported by M_3, using Fact 1, the same chain exists in $\Gamma^s(M_0) \cup \Gamma^s(M_3)$.

case $n > 2$**:** We isolate the first and last $\Delta\Gamma(M_1)$ steps in the chain $f \longrightarrow^* g$:

$$f \overset{\Gamma^s(M_0+M_3)}{\longrightarrow}{}^* f' \overset{\Delta\Gamma(M_1)}{\longrightarrow} \overline{\sigma}(h) \overset{\overline{\Gamma}}{\longrightarrow}{}^* \overline{\sigma}(h') \overset{\Delta\Gamma(M_1)}{\longrightarrow} g' \overset{\Gamma^s(M_0+M_3)}{\longrightarrow}{}^* g$$

If there is at least one operation $k \in \Sigma(M_0 + M_3)$ in $\overline{\sigma}(h) \overset{\overline{\Gamma}}{\longrightarrow}{}^* \overline{\sigma}(h')$ the result is proved by applying the induction hypothesis to the two sub-chains $f' \longrightarrow^* k$ and $k \longrightarrow^* g'$. If there is no such operation, the chain cannot include an operation $\sigma(p)$, where p is a *par* operation, since a chain $f' \to^+ h \to^+ p$ would exists in $\Gamma^s(M_1)$, a situation ruled out by Def.3(b)iii. The last possibility is to have in the chain only instantiated operations: $\overline{\sigma}(h) \to \overline{\sigma}(h_1) \to \ldots \to \overline{\sigma}(h_n) \to \overline{\sigma}(h')$, but then the corresponding chain with generic operations $h \to h_1 \to \ldots \to h_n \to h'$ must exist in $\Gamma^s(M_1)$ and so does $f' \to h \to^+ h' \to g'$. Since $\Gamma^s(M_1)$ is modularly oriented, as in the case $n = 2$, a chain $f' \to^+ g'$ must exist in $\Gamma^s(M_0) \cup \Gamma^s(M_3)$.

In summary, in all cases we have proved the existence of a chain between f and g with all steps in $\Gamma^s(M_0 + M_3)$. □

We now prove Prop. 10, i.e. Prop. 5 and Prop. 6 for arbitrary specifications:

Proposition 5 *Let $\mathcal{SP}(M)$ be an adequately structured modular specification. If all M_i in $\mathcal{I}(M)$ are modularly oriented, then $(\cup\Gamma^s(M_i))^+$ is an ordering relation. Hence M can also be modularly oriented.*

Proof. By induction on the structure of M we prove that it is impossible to have a cycle $f \to^+ g \to^+ f$. The only non straightforward case concerns the sum of modules, since by Lemma 2 instantiations can always be replaced by "equivalent" sums of modules. Assume that $M = (M_1 + M_2, \Delta\Phi)$. By Lemma 1 the only possibility is to have two operations h and k such that $M_h \leq M_f$, $M_h \leq M_g$ and $f \xrightarrow{\Gamma^s_f} {}^* h \xrightarrow{\Gamma^s_g} {}^* g$ on one hand, and $M_k \leq M_f$, $M_k \leq M_g$ with $g \xrightarrow{\Gamma^s_g} {}^* k \xrightarrow{\Gamma^s_f} {}^* f$ on the other hand. However if $h \to {}^+ k$ in Γ^s_g, that relation also holds in any context that imports h and k, in particular in M_f and there would exist a cycle in Γ^s_f and therefore in $\Gamma^s(M_1)$ or $\Gamma^s(M_2)$, a contradiction with the induction hypothesis since these modules were supposed to be modularly oriented. □

Proposition 6 *Let $\mathcal{SP}(M)$ be an adequately structured, modularly oriented, specification and let $\mathcal{C} \subseteq \mathcal{E}(M)$ be an arbitrary collection of modules of $\mathcal{SP}(M)$. Then $\cup\, \Delta\mathcal{R}^s(M_i)$, for all M_i in $\mathcal{C}$, is finitely terminating.*

Proof. The result is a consequence of Lemma 1, Fact 1 and of the incrementality of path ordering methods: when $\Delta\mathcal{R}^s(M)$ has been proved to be finitely terminating using an ordering Γ, the termination property holds with any extension Γ' of Γ. The only case to consider are the instantiations, since the relations associated with operations from the formal parameter are missing, but the equations from the generic specification have been oriented without any reference to them. Therefore, $\overline{\sigma}(\Delta\mathcal{R}^s(M)) \subseteq \Delta\mathcal{R}^s(M')$ is valid, for any instantiation M' of M. □

We finally prove the result for dynamic extensions:

Proposition 11 *Let $\mathcal{SP}(M)$ be a modularly oriented specification and let $\mathcal{C}$, $\mathcal{C} \subseteq \mathcal{E}(M)$, an arbitrary subset of modules. Let $\Gamma^s_{\mathcal{C}} = \cup\, \Gamma^s(M_i)$, $\mathcal{F}^s_{\mathcal{C}} = \cup\, \Delta\mathcal{F}^s(M_i)$, $\mathcal{R}^s_{\mathcal{C}} = \cup\, \Delta\mathcal{R}^s(M_i)$ and $\Sigma_{\mathcal{C}} = \cup\, \Sigma(M_i)$, where in all unions M_i ranges over $\mathcal{C}$. Let $\Gamma^d_{\mathcal{C}}$ be an extension over $\Sigma_{\mathcal{C}}$ of $\Gamma^s_{\mathcal{C}}$ such that $\Gamma^{d+}_{\mathcal{C}}$ is an ordering relation, and $\mathcal{R}^d_{\mathcal{C}}$ be any partition of $\mathcal{F}^s_{\mathcal{C}}$ into $(\mathcal{R}^d_{\mathcal{C}}, \mathcal{F}^d_{\mathcal{C}})$ made according to $\Gamma^d_{\mathcal{C}}$. Then for any module M_j in $\mathcal{E}(M)\backslash\mathcal{C}$ we have:*

1. *$(\Gamma^s(M_j) \cup \Gamma^d_{\mathcal{C}})^+$ is an ordering relation*
2. *$\Delta\mathcal{R}^s(M_j) \cup \mathcal{R}^s_{\mathcal{C}} \cup \mathcal{R}^d_{\mathcal{C}}$ is finitely terminating.*

Proof. We only prove that $(\Gamma^s(M_j) \cup \Gamma^d_{\mathcal{C}})^+$ is acyclic, the termination of the extended rewrite system being proved as above. There are two cases to consider: if $\mathcal{C}$ contains a module M_i such that $M_j \leq M_i$, then $\Gamma^s(M_j) \subseteq \Gamma^s(M_i) \subseteq \Gamma^d_{\mathcal{C}}$ and the result is straightforward. Otherwise, assume that the new ordering has a cycle. Since all modules in $\mathcal{C}$ and M_j are part of M which has been modularly oriented, $\Gamma^s_{\mathcal{C}} \subseteq \Gamma^s(M)$ is acyclic, and any cycle must involve an arc $h \to k$ in $\Gamma^d_{\mathcal{C}} \setminus \Gamma^s_{\mathcal{C}}$, with both h and k in $\Sigma_{\mathcal{C}}$, and an operation f in $\Delta\Sigma(M_j)$ such that $f \to^+ h \to k \to^+ f$. Now, in this cycle take the longest chain around f made only with arcs from the static ordering and consider its endpoints h' and k': both symbols are not in M_j since otherwise the next arc would have been static also, the dynamic extension being defined only on $\Sigma_{\mathcal{C}}$! Such operations exist since at least $h = h'$ and $k = k'$ is possible. We have h' and $k' \notin \Delta\Sigma(M_j)$ such that $f \to^* h' \to^* h \to k \to^* k' \to^* f$. Using Lemma 1, a chain $k' \to^+ h'$ exists in $\Gamma^s_{\mathcal{C}}$ and the addition of $h \to k$ in $\Gamma^d_{\mathcal{C}}$ would not have been possible. □

This article was processed using the LaTeX macro package with LLNCS style