Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.A. Bergstra, J. Heering, P. Klint

Module algebra

Computer Science/Department of Software Technology Report CS-R8617 May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Module Algebra

J.A. Bergstra
Department of Computer Science, University of Amsterdam
Department of Philosophy, University of Utrecht

J. Heering
Department of Software Technology, Centre for Mathematics and Computer Science

P. Klint
Department of Software Technology, Centre for Mathematics and Computer Science
Department of Computer Science, University of Amsterdam

An axiomatic algebraic calculus of modules is given which is based on the operators *combination/union*, *export*, *renaming*, and *taking the visible signature*. Four different models of module algebra are discussed and compared.

1986 CR Categories: D.2.0 [**Software Engineering**]: Requirements/Specifications - *Languages;* D.2.2 [**Software Engineering**]: Tools and Techniques - *Modules and Interfaces;* D.3.3 [**Programming Languages**]: Language Constructs - *Modules;* F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages - *Algebraic Approaches to Semantics.*

1980 Mathematics Subject Classification: 68B10 [**Software**]: Analysis of Programs - *Semantics.*

Key Words & Phrases: algebraic specification, first-order specification, signature, module algebra, module composition, signature expression, module expression, Craig interpolation lemma, information hiding, abstraction, export, union of modules, renaming, visible signature.

Note: Partial support received from the European Communities under ESPRIT projects 348 (Generation of Interactive Programming Environments - GIPE) and 432 (An Integrated Formal Approach to Industrial Software Development - METEOR).

Note: This paper has been submitted for publication elsewhere.

1. Introduction

1.1. General

The study of modules and modularization is one of the central issues in software engineering. Three notions are basic to an understanding of modularization as a software engineering technique:

(1) *Information hiding/abstraction.* Modules generally contain *hidden* (*auxiliary*, *local*, *internal*, *invisible*, . . .) items without which it would be very difficult or even impossible to specify them. These items must remain inaccessible from the outside so as not to spoil the intended semantics of the module [P72]. Examples are the hidden variables and functions that have to be introduced when specifying data types in programming languages, and the hidden sorts and functions needed in initial algebra specifications of data types (see Section 5.4). It should be noted that the notion of "hiding the representation" discussed by Meseguer & Goguen in the context of algebraic specification [MG86] is different from the notion of information hiding. Representation independence is achieved by considering isomorphism classes of models instead of single

Report CS-R8617
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

(concrete) models, but this does not help in hiding auxiliary sorts and functions.

(2) *Compositionality of module operations.* Modules can be adapted and combined by means of various operations like renaming of hidden items and importing a module in another one. Each such operation should preferably be a simple (but at least an effectively computable) operation on the textual representation (presentation) of modules. Import of a module in another module, for instance, should correspond to textual substitution plus renaming of hidden items to avoid name clashes. Simplicity at the textual level is not enough, however. The textual operation should have a semantical counterpart which is comparable in simplicity, i.e., module operations should be *compositional* [J86]. If these two (tough) requirements can be met, computations involving modules become both practicable and meaningful.

(3) *Reusability of modules.* Some modules can be used as part of many programs or specifications. These are said to be *reusable*. Such modules resemble constructs in programming or specification languages, which are also highly reusable. (From this viewpoint a programming language is nothing but a coherent collection of reusable constructs.) Reusability can be enhanced by choosing the right module composition operations, but the requirement of compositionality imposes a restriction on the module operations that are acceptable. For instance, creating a new module by editing the text of an existing one is also a very general form of reuse, of course, but this will not normally correspond to a comparably simple change in the semantics of the module and hence not to a valid module operation.

1.2. Outline of this paper

Each specification module (at least implicitly) contains a syntax part defining the language used in it. Composition of modules entails, first of all, composition of the corresponding languages and hence composition of the corresponding syntax definitions. In principle, these may be arbitrary grammars, but in this paper we limit ourselves to *signatures* defining strongly typed first-order expression languages. In SECTION 2.1 we discuss signatures in general terms, and in SECTION 2.2 we give an initial/final algebra specification of the algebra of signatures. Basic operators of this algebra are *renaming* $(.)$, *combination/union* $(+)$, and *intersection* $(\cap)$.

In SECTION 3.1 the definition of the algebra of signatures is extended to a definition of the basic algebra of first-order logic modules $BMA[fol]$. The main operators of this algebra are *taking the visible signature* (Σ), *renaming* $(.)$, *combination/union* $(+)$, and *export* $(\Box)$. (We do not discuss parametrization in this paper.) In SECTION 3.2 we prove a normal form theorem for closed module expressions. In SECTION 3.3 we introduce *hiding* and *common export* operators. The former is complementary to export, while the latter is a generalization of export allowing a rather elegant axiomatization. In SECTION 3.4 we discuss four well-known types of construction/development steps, namely *abstraction*, *enrichment*, *extension* and *refinement*, from the viewpoint of module algebra.

In SECTION 4 four different models for $BMA[fol]$ are given:

(1) the initial algebra $\mathbb{I}(BMA[fol])$,
(2) the algebra $\mathbb{M}(fol)$ of full model classes of modules,
(3) the algebra $\mathbb{M}_C(fol)$ of classes of countable models of modules, and
(4) the algebra $\mathbb{T}(fol)$ of theories of modules.

We show that there are homomorphisms $\mathbb{M}(fol) \to \mathbb{M}_C(fol)$ and $\mathbb{M}_C(fol) \to \mathbb{T}(fol)$, and also that $\mathbb{M}(fol) \ncong \mathbb{M}_C(fol) \ncong \mathbb{T}(fol)$.

In SECTIONS 5.1-2 the expressive power of equational logic, (*eql*), conditional equational logic (*ceql*), first-order logic (*fol*), and equational logic in the presence of Booleans (*eql* + *bool*) are compared with each other. The implications of our results for equational logic and initial algebra semantics are briefly discussed in SECTION 5.3, while SECTION 5.4 gives an overview of related results in the field of algebraic specification.

Finally, a series of examples of modular algebraic specifications using the operators of *BMA* are given in SECTION 6.

1.3. Related work

The introduction of composition/construction operators for modular specifications is, of course, not new. Such operators occur, for instance, in CLEAR [BG80], OBJ2 [FGJM85], OBSCURE [LOE85], and PLUSS [GAU85]. In particular, the operators *union*, *export* and *forget* in PLUSS are similar to our operators $+$, $\Box$ and Δ. GANZINGER [GAN83], KLAEREN [KLA83], and EHRIG & MAHR [EM85] have given a category theoretic treatment of the $+$-operator in the context of initial/final algebra semantics.

A structure theory of algebraic specifications based on a set of construction operators was given by KAPLAN [KAP83], LIPECK [LIP83], and WIRSING [W83]. The work of LIPECK is also based on category theory, but WIRSING uses first-order logic and model theory as his point of departure. Our approach is similar to that of WIRSING. In fact, the full model class semantics $\mathbb{M}(fol)$ was discussed by him in [W83] and several laws of $BMA[fol]$ can be identified there, although not yet in a uniform setting. The importance of the CRAIG interpolation lemma in the context of specification languages was pointed out by MAIBAUM, VELOSO & SADLER [MVS85, MS85], who used it to characterize the composability of implementations. We obtain a conditional distributive law (E4)

$$x = (\Sigma(Y) \cap \Sigma(Z)) + x' \;\Rightarrow\; x \Box (Y + Z) = (x \Box Y) + (x \Box Z),$$

which, in the context of a first-order logic interpretation of module expressions, is equivalent to the CRAIG interpolation lemma.

In BERGSTRA, HEERING & KLINT [BHK85] we experimented with an algebraic specification formalism similar to OBJ or PLUSS. Our motivation for the present work was both dissatisfaction with the import and export mechanisms we used there and the feeling that we needed a firmer foundation for our formalism.

As far as we know the following points in our paper are new:

(1) the specification of the algebra of signatures;
(2) the laws of $BMA[fol]$;
(3) the normal form theorem for closed module expressions;
(4) the models $\mathbb{M}_C(fol)$ and $\mathbb{T}(fol)$ of $BMA[fol]$ (with the understanding that $\mathbb{M}(fol)$ has already been discussed by WIRSING [W83]);
(5) the fact that equations and conditional equations have equal power for a variety of different semantics;
(6) the fact that in the presence of *bool* equations are as powerful as full first-order logic.

2. SIGNATURES

2.1. General

The language in which the axioms of a specification are expressed consists of a logical and a non-logical part. The latter is defined by the *signature* of the specification. The signatures of many-sorted algebraic and first-order specifications (the only ones we consider in this paper) are sets of declarations of *sorts*, *typed constants*, and *typed functions*.

FIGURE 2.1 shows a simple example of a signature *Sig* both in textual and graphical form. Because the constant symbols 0 and function symbol S are declared more than once with different types, they are said to be *overloaded*. The circles in the graphical representation correspond to sorts while the arrows denote constants or functions. In general, the types of n-adic function symbols ($n \geqslant 2$) are not uniquely determined by the graphical representation, but only up to an arbitrary permutation of the argument sorts.

sorts N, L
constant $0 : N$
function $S : N \to N$
constant $0 : L$
functions
$i : N \to L$
$f : L \times L \to L$
$S : L \to L$

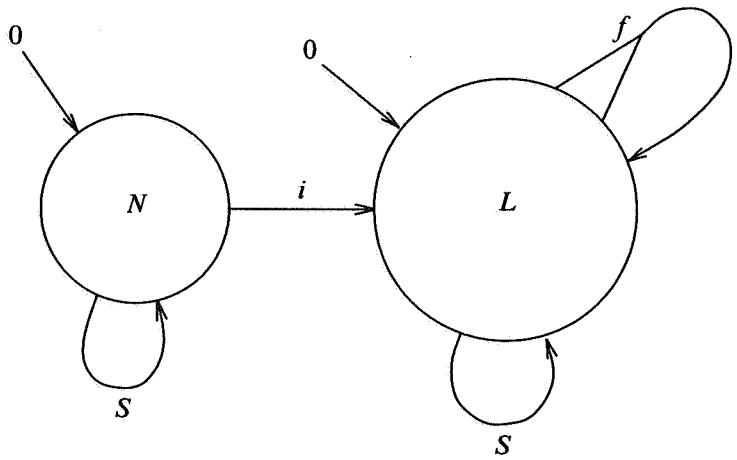

FIGURE 2.1. Example of a signature *Sig* - textual and (almost) equivalent graphical representation.

Let $\mathcal{T}(\Sigma)$ be the set of *correctly and explicitly typed* expressions (terms) which can be formed from the constant and function symbols declared in a signature Σ plus the typed first-order variable symbols declared in some separate variable declaration, and let $\mathcal{L}(\Sigma)$ be the set of correctly and explicitly typed first-order formulas over Σ. Some expressions belonging to $\mathcal{T}(Sig)$ (FIGURE 2.1) are

0^N

$S^{N\to N}(0^N)$

$S^{L\to L}(f^{L\times L\to L}(x^L,y^L))$,

where x and y are variables of sort L. Some expressions *not* in $\mathcal{T}(Sig)$ are

0	(not explicitly typed)
$S^{L\to L}(0^N)$	(incorrectly typed)
$f^{L\times L\to L}(0^L)$	(f is not a monadic function).

Usually, most of the explicit typing is redundant. For instance, the $\mathcal{L}(Sig)$-identities

$S^{L\to L}(0^L) = 0^L$

$S^{L\to L}(i^{N\to L}(n^N)) = i^{N\to L}(S^{N\to N}(n^N))$

$S^{L\to L}(f^{L\times L\to L}(x^L,y^L)) = f^{L\times L\to L}(S^{L\to L}(x^L),S^{L\to L}(y^L))$

can in principle be abbreviated to

$$S(0^L) = 0$$

$$S(i(n)) = i(S(n))$$

$$S(f(x,y)) = f(S(x),S(y)),$$

because all types except that of 0 and S in the first identity can be deduced from the context in which the constant and function symbols occur. This example shows that if all explicit typing is dropped the intended typing cannot always be inferred mechanically. Also, type inference is context-dependent and, even if initially possible, may become impossible if the context is widened by combining the module in question with another one. In SECTION 3.5 we introduce a notation that allows us to drop the explicit typing from axioms in many cases.

2.2. The algebra of signatures

Composition of specification modules entails, first of all, composition of the corresponding signatures. Hence, we first give an initial/final algebra specification of the algebra of signatures (FIGURES 2.2 and 2.3). Signatures are basically *sets of atomic signatures*. The latter are declarations of a single sort or function. The primary operations on signatures are renaming (.), *combination/union* (+), and *intersection* ($\cap$).

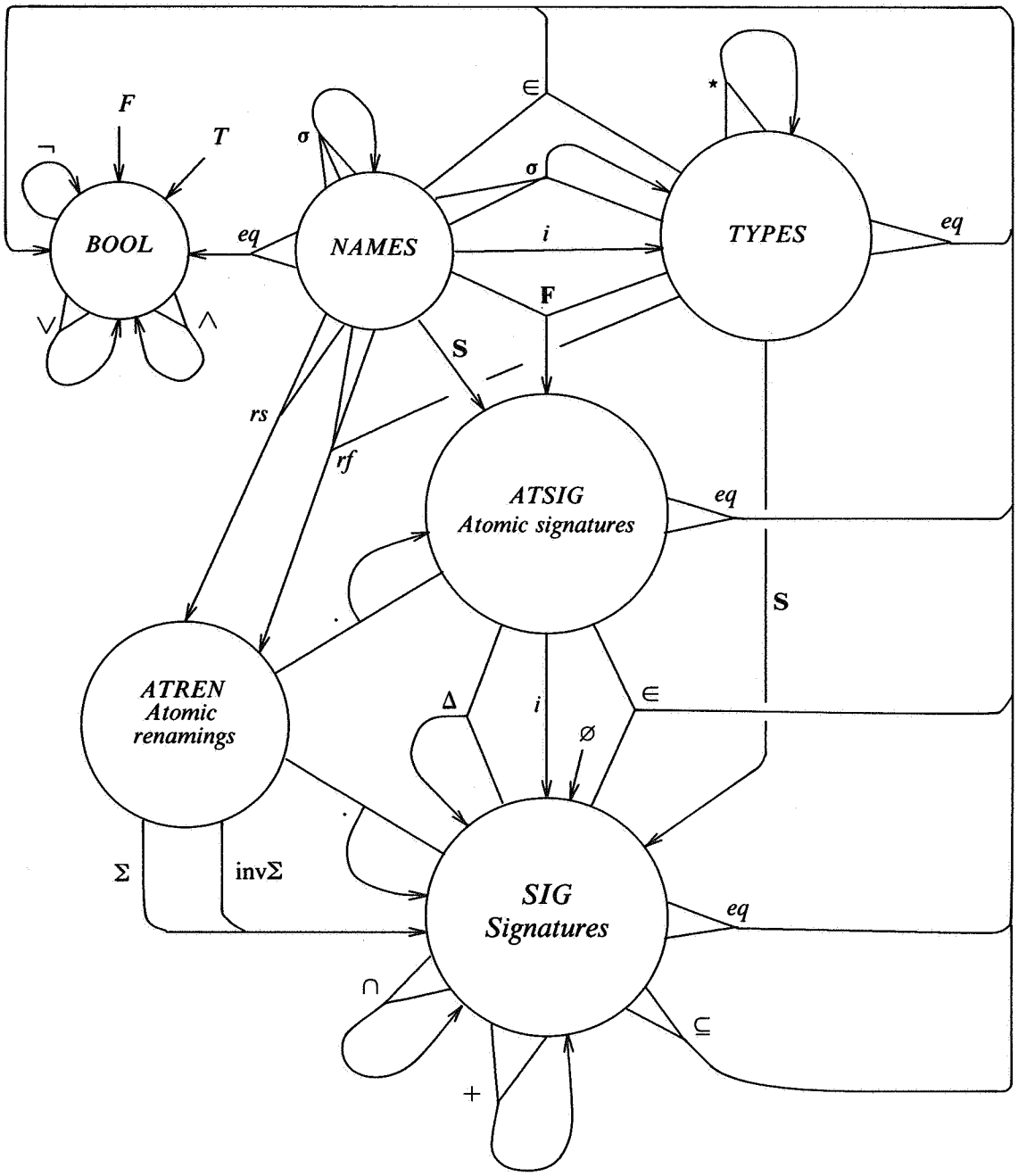

FIGURE 2.2. The signature of the algebra of signatures.

specification *Booleans*
begin

sort $BOOL$
constants $F,T : BOOL$
functions
$\neg : BOOL \rightarrow BOOL$
$\vee\ ,\ \wedge : BOOL \times BOOL \rightarrow BOOL$
variables $X,Y,Z : BOOL$
axioms
$\neg F = T$
$\neg\neg X = X$
$X \vee T = T$
$X \vee F = X$
$X \vee \neg X = T$
$(X \vee Y) \vee Z = X \vee (Y \vee Z)$
$X \vee Y = Y \vee X$
$X \vee X = X$
$X \wedge Y = \neg(\neg X \vee \neg Y)$
$(X \vee Y) \wedge Z = (X \wedge Z) \vee (Y \wedge Z)$

end *Booleans*

specification *Signatures*
begin

import *Booleans*
parameter *Names*
begin
sort $NAMES$
functions
$eq : NAMES \times NAMES \rightarrow BOOL$ (*Equality*)
$\sigma : NAMES \times NAMES \times NAMES \rightarrow NAMES$ (*Elementary renaming*)
variables $l,m,n : NAMES$
requirements
$eq(l,l) = T$
$l \neq m \ \Rightarrow\ eq(l,m) = F$

$\sigma(l,m,l) = m$
$\sigma(l,m,m) = l$
$eq(l,n) = F \;\&\; eq(m,n) = F \;\Rightarrow\; \sigma(l,m,n) = n$

end *Names*

sort *TYPES* (*Sequences of one or more names*)

functions

$i : NAMES \rightarrow TYPES$ (*Injection*)
$\star : TYPES \times TYPES \rightarrow TYPES$ (*Concatenation*)
$\sigma : NAMES \times NAMES \times TYPES \rightarrow TYPES$ (*Renaming*)
$\in : NAMES \times TYPES \rightarrow BOOL$ (*Membership*)
$eq : TYPES \times TYPES \rightarrow BOOL$ (*Equality*)

variables

$l,m,n : NAMES$
$t,u,v : TYPES$

axioms

$(t\star u)\star v = t\star(u\star v)$

$\sigma(l,m,i(n)) = i(\sigma(l,m,n))$
$\sigma(l,m,t\star u) = \sigma(l,m,t)\star\sigma(l,m,u)$

$l \in i(m) = eq(l,m)$
$l \in (t\star u) = (l \in t) \vee (l \in u)$

$eq(i(l),i(m)) = eq(l,m)$
$eq(i(l)\star t, i(m)\star u) = eq(l,m) \wedge eq(t,u)$
$eq(i(l),t\star u) = F$
$eq(t\star u, i(l)) = F$

sort *ATSIG* (*Atomic signatures*)

functions

$\mathbf{S} : NAMES \rightarrow ATSIG$ (*Sort constructor*)
$\mathbf{F} : NAMES \times TYPES \rightarrow ATSIG$ (*Constant/function constructor*)
$eq : ATSIG \times ATSIG \rightarrow BOOL$ (*Equality*)

variables

$l,m : NAMES$
$t,u : TYPES$

axioms

$eq(\mathbf{S}(l),\mathbf{S}(m)) = eq(l,m)$
$eq(\mathbf{S}(l),\mathbf{F}(m,t)) = F$
$eq(\mathbf{F}(l,t),\mathbf{S}(m)) = F$
$eq(\mathbf{F}(l,t),\mathbf{F}(m,u)) = eq(l,m) \wedge eq(t,u)$

sort *ATREN* (*Atomic renamings*)

functions

$rs : NAMES \times NAMES \rightarrow ATREN$ (*Sort renaming constructor*)

$rf : NAMES \times NAMES \times TYPES \rightarrow ATREN$ (*Function renaming constructor*)

$. : ATREN \times ATSIG \rightarrow ATSIG$ (*Apply atomic renaming*)

variables

$l,m,n : NAMES$

$t,u : TYPES$

axioms

$rs(l,l) = rs(m,m)$
$rs(m,m) = rf(l,l,t)$
$rf(l,l,t) = rf(m,m,u)$
} (*Identify all identity renamings*)

$rs(l,m) = rs(m,l)$

$rf(l,m,t) = rf(m,l,t)$

$rs(l,m).\mathbf{S}(n) = \mathbf{S}(\sigma(l,m,n))$

$rs(l,m).\mathbf{F}(n,t) = \mathbf{F}(n,\sigma(l,m,t))$

$rf(l,m,t).\mathbf{F}(n,t) = \mathbf{F}(\sigma(l,m,n),t)$

$eq(t,u) = F \Rightarrow rf(l,m,t).\mathbf{F}(n,u) = \mathbf{F}(n,u)$

$rf(l,m,t).\mathbf{S}(n) = \mathbf{S}(n)$

sort *SIG* (*Signatures*)

constant $\varnothing : SIG$ (*Empty signature*)

functions

$i : ATSIG \rightarrow SIG$ (*Injection*)

$+ : SIG \times SIG \rightarrow SIG$ (*Combination/Union*)

$\mathbf{S} : TYPES \rightarrow SIG$ (*Convert type to set of sorts*)

$. : ATREN \times SIG \rightarrow SIG$ (*Apply atomic renaming*)

$\Sigma : ATREN \rightarrow SIG$ (*Signature affected by atomic renaming*)

$\mathrm{inv}\Sigma : ATREN \rightarrow SIG$ (*Signature used but invariant under atomic renaming*)

$\in : ATSIG \times SIG \rightarrow BOOL$ (*Membership*)

$\cap : SIG \times SIG \rightarrow SIG$ (*Intersection*)

$\Delta : ATSIG \times SIG \rightarrow SIG$ (*Deletion*)

$\subseteq : SIG \times SIG \rightarrow BOOL$ (*Subsignature*)

$eq : SIG \times SIG \rightarrow BOOL$ (*Equality*)

variables

$l,m : NAMES$

$t,u : TYPES$

$a : ATSIG$

$r : ATREN$

$x,y,z : SIG$

axioms

$x+\varnothing = x$

$x+x = x$

$x+y = y+x$

$(x+y)+z = x+(y+z)$

$i(\mathbf{F}(l,t)) = i(\mathbf{F}(l,t))+\mathbf{S}(t)$ *(A constant or function implicitly declares the sort(s) occurring in its type)*

$\mathbf{S}(i(l)) = i(\mathbf{S}(l))$

$\mathbf{S}(t*u) = \mathbf{S}(t)+\mathbf{S}(u)$

$r.\varnothing = \varnothing$

$r.i(a) = i(r.a)$

$r.(x+y) = (r.x)+(r.y)$

$\Sigma(rs(l,l)) = \varnothing$ *(This catches all identity renamings)*

$eq(l,m) = F \Rightarrow \Sigma(rs(l,m)) = i(\mathbf{S}(l))+i(\mathbf{S}(m))$

$eq(l,m) = F \Rightarrow \Sigma(rf(l,m,t)) = i(\mathbf{F}(l,t))+i(\mathbf{F}(m,t))$

$\mathrm{inv}\Sigma(rs(l,m)) = \varnothing$

$eq(l,m) = F \Rightarrow \mathrm{inv}\Sigma(rf(l,m,t)) = \mathbf{S}(t)$

$a \in \varnothing = F$

$\mathbf{S}(l) \in i(\mathbf{S}(m)) = eq(l,m)$

$\mathbf{S}(l) \in i(\mathbf{F}(m,t)) = l \in t$

$\mathbf{F}(l,t) \in i(\mathbf{F}(m,u)) = eq(l,m) \wedge eq(t,u)$

$\mathbf{F}(l,t) \in i(\mathbf{S}(m)) = F$

$a \in (x+y) = (a \in x) \vee (a \in y)$

$x \cap \varnothing = \varnothing$

$x \cap x = x$

$x \cap y = y \cap x$

$(x \cap y) \cap z = x \cap (y \cap z)$

$\mathbf{S}(l) \in x = F \Rightarrow i(\mathbf{S}(l)) \cap x = \varnothing$

$\mathbf{F}(l,t) \in x = F \Rightarrow i(\mathbf{F}(l,t)) \cap x = \mathbf{S}(t) \cap x$

$a \in x = T \Rightarrow i(a) \cap x = i(a)$

$(x+y) \cap z = (x \cap z)+(y \cap z)$

$a \in x = F \Rightarrow a \Delta x = x$

$a \Delta i(a) = \varnothing$

$l \in t = T \Rightarrow \mathbf{S}(l) \Delta i(\mathbf{F}(m,t)) = \mathbf{S}(l) \Delta \mathbf{S}(t)$

$a \Delta (x+y) = (a \Delta x)+(a \Delta y)$

$x+z = y \Rightarrow x \subseteq y = T$

$$a \in x = T \;\&\; a \in y = F \;\Rightarrow\; x \subseteq y = F$$

$$eq(x,y) = (x \subseteq y) \wedge (y \subseteq x)$$

end *Signatures*

FIGURE 2.3. Initial/final algebra specification of the algebra of signatures.

COMMENTS:
(1) Functions whose type consists of a single name correspond to constants.

(2) Although sorts are implicitly declared by constants and functions, a sort that does not occur in the type of any constant or function still has to be declared explicitly.

(3) Because we allow overloaded constants and functions, unrestricted union of signatures is no problem.

(4) In general, $\mathcal{T}(\Sigma_1 + \Sigma_2) \neq \mathcal{T}(\Sigma_1) \cup \mathcal{T}(\Sigma_2)$.

(5) Due to its permutative character, renaming never causes name clashes.

(6) Because of overloading, atomic renamings must completely specify the atomic signature they should work on when applied.

(7) The use of functions Σ and invΣ on *ATREN* will become clear later on when restricted renameability of hidden functions in modules is discussed (SECTION 3.1).

(8) *Signatures* has formal parameter *Names* simply because we preferred to suppress most details of the associated data type. For all suitable actual parameters, the initial model of *Signatures* is a computable algebra. Every closed signature expression of sort *SIG* can be brought in the normal form

$$\sum_{k=1}^{m} i(\mathbf{S}(s_k)) + \sum_{k=1}^{n} i(\mathbf{F}(f_k,t_k)) \qquad (m,n \geqslant 0),$$

with $s_k \neq s_l$ $(k \neq l)$, $(f_k,t_k) \neq (f_l,t_l)$ $(k \neq l)$, and $s_k \notin t_l$, i.e., only sorts not occurrring in the type of any constant or function are declared explicitly. Two signatures are equal if and only if the corresponding normal forms are syntactically identical modulo associativity and commutativity of $+$ and modulo associativity of $\star$.

Furthermore, for all suitable actual parameters the initial and final model of *Signatures* are isomorphic, i.e., the initial model does not have non-trivial homomorphic images and all non-trivial minimal models are isomorphic (cf. [BT82]). There are two reasons for this. First, on all sorts except *ATREN* an *eq*-function is defined with the property that for all closed expressions x and y

$$\vdash eq(x,y) = T \;\Leftrightarrow\; \vdash x = y$$
$$\vdash eq(x,y) = F \;\Leftrightarrow\; \nvdash x = y.$$

On these sorts any equality which is stronger than provable equality immediately leads to inconsistency. Secondly, all atomic renamings with the same behavior are provably equal and hence no stronger equality on *ATREN* is possible without inducing a stronger equality on *ATSIG* as well.

We are not interested in "non-standard signatures", i.e., we only consider the non-trivial minimal model of *Signatures*.

(9) We call an equation *ω-derivable* if all of its closed instances are derivable. An equation is ω-derivable if and only if it holds in the initial model (cf. [HEE85]). Some equations that are ω-derivable from *Signatures* are:

$$(x \cap y) + z = (x \cap y) + (x \cap z)$$

$$(x+y)\cap x = x$$
$$x+(x\cap y) = x$$
$$r.(r.x) = x$$
$$r.(x\cap y) = (r.x)\cap(r.y)$$
$$r.(a\Delta x) = (r.a)\Delta(r.x).$$

For reasons of readability we will from now on use a somewhat different notation for signatures. Instead of

$$i(\mathbf{S}(n))$$
$$i(\mathbf{F}(c,i(n)))$$
$$i(\mathbf{F}(f,(\cdots(i(n_1)*\cdots*i(n_{k-1}))*i(n_k)))) \qquad (k>1)$$

we will write respectively

$$\mathbf{S}{:}n$$
$$\mathbf{F}{:}c{:}n$$
$$\mathbf{F}{:}f{:}n_1\times\cdots\times n_{k-1}\to n_k.$$

For instance,

$$i(\mathbf{S}(n))+i(\mathbf{S}(m))+i(\mathbf{F}(f,(i(n)*i(m))*i(n)))$$

becomes

$$\mathbf{S}{:}n+\mathbf{S}{:}m+\mathbf{F}{:}f{:}n\times m\to n.$$

3. Basic module algebra

3.1. BMA[fol]

In this section we consider *module expressions*. These are basically expressions consisting of module constants/variables and the operators Σ (the visible signature of a module), . (renaming of a module), T (conversion of a signature to a module without axioms), + (combination/union of modules), and $\Box$ (restriction of the visible signature of a module).

Each (closed) first-order sentence ϕ over a signature x corresponds to a module constant $<\phi>$ with signature $\Sigma(<\phi>) = \bigcap_{\phi\in\mathscr{L}(y)} y$. A finite first-order theory corresponds to a module expression

$$<\phi_1>+\cdots+<\phi_n>,$$

where + is the above-mentioned combination operator on modules. The signature of such a theory is

$$\Sigma(<\phi_1>+\cdots+<\phi_n>) = \Sigma(<\phi_1>)+\cdots+\Sigma(<\phi_n>),$$

where the +-operator occurring in the right-hand side is the + on signatures.

Renaming of signatures is extended in the natural way to renaming of first-order sentences. So $r.\phi$ is the sentence obtained from ϕ by applying atomic renaming r to it, and $<r.\phi>$ is the corresponding module. Clearly, $\Sigma(<r.\phi>) = r.\Sigma(<\phi>)$.

In addition to (infinitely many) constants $<\phi>$, there are module expressions $T(x)$ for each signature x. These represent modules that do not impose any constraint on x-algebras.

The class of *flat* module expressions consists of expressions involving only the constants $<\phi>$ and the operators +, . and T. These represent ordinary finite first-order theories. $T(x)$ is equivalent to $<\phi>$ with ϕ a tautology and $\Sigma(<\phi>)=x$.

Non-flat expressions involve the export operator $\Box$. Consider, for instance,

$$x\Box(<\phi_1>+<\phi_2>),$$

which is to be read as "export x from $<\phi_1>+<\phi_2>$". The intended meaning of this module

expression is a specification whose visible signature is restricted to those sorts and functions of $\Sigma(<\phi_1>+<\phi_2>)$ which also occur in x, i.e.,

$$\Sigma(x\,\square(<\phi_1>+<\phi_2>)) = x \cap \Sigma(<\phi_1>+<\phi_2>).$$

Sorts and functions not occurring in x become hidden, i.e., inaccessible from the outside. One of the main properties of hidden sorts and functions is that they can be renamed without affecting the meaning of the specification in which they occur, as long as name clashes between hidden names as well as between hidden and visible names are avoided.

The axioms of basic module algebra for first-order logic specifications (*BMA*[*fol*]) are given in FIGURE 3.2. A graphical representation of the corresponding signature is shown in FIGURE 3.1. The axioms of *BMA*[*fol*] mainly describe the interaction between the $+$- and $\square$-operators. Although this cannot be done without aiming at a specific semantics for $+$ and $\square$, it turns out that:

(1) The axioms of *BMA*[*fol*] are convincing on *a priori* grounds even without such a semantics;

(2) *BMA*[*fol*] has several different semantics including the "natural" ones;

(3) The $+$-, $\square$- and Σ-operators cannot be treated separately from each other. General axioms describing their interrelation are necessary if a useful interpretation of these operators is to be obtained. Trying to find a meaning for the structuring operators of modular specifications without any axiomatic preliminaries is not a well-defined problem.

While designing *BMA*[*fol*] we kept the following requirements in mind:

(1) All equations of *BMA*[*fol*] would have to hold in the algebra $\mathbb{M}(fol)$ of full model classes of modules which we consider to be a natural standard model for modular specifications. $\mathbb{M}(fol)$ is discussed in SECTION 4.2.

(2) As an extension of *Signatures* (SECTION 2.2) *BMA*[*fol*] would have to leave *Signatures* untouched in the sense that every closed *SIG*-term over the signature of *BMA*[*fol*] would have to be provably equal to a closed *SIG*-term over the signature of *Signatures*, and no new identities between closed terms over the signature of *Signatures* would be introduced.

(3) Every closed module expression should be provably equal to a normal form containing at most a single instance of the export operator $\square$, i.e., the normal form theorem (THEOREM 3.3.2) would have to hold for *BMA*[*fol*]. Normalization of module expressions is a basic operation which will have to be implemented in any system for manipulating specifications.

(4) The axioms of *BMA*[*fol*] would have to guarantee that *refinement* and *enrichment* are both special cases of *extension* (see SECTION 3.4).

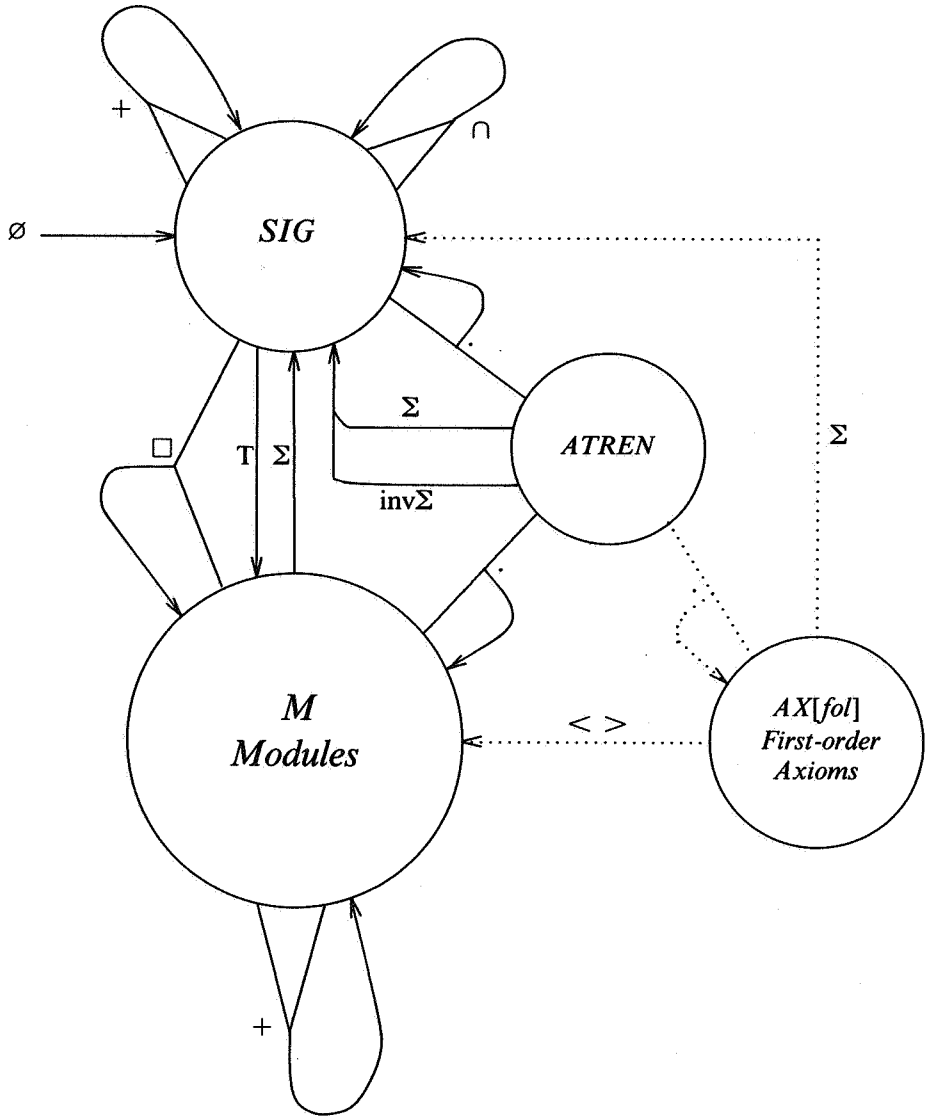

FIGURE 3.1. The signature of *BMA*[*fol*].
(The signature is only partially shown.
It is an extension of the signature of
Signatures shown in FIGURE 2.2.)

---

specification *BMA*[*fol*]
begin

import *Signatures*

sort *M* (*Modules*)

constants

$<\phi> : M$ — *(For each closed first-order formula $\phi \in \mathcal{L}(x)$ with signature x, $<\phi>$ is a constant of sort M)*

functions

$\Sigma : M \to SIG$ — *(Signature)*

$\mathrm{T} : SIG \to M$ — *(Injection)*

$. : ATREN \times M \to M$ — *(Apply atomic renaming)*

$+ : M \times M \to M$ — *(Combination/Union)*

$\Box : SIG \times M \to M$ — *(Export)*

variables

$r : ATREN$

$x, x', y : SIG$

$X, Y, Z : M$

axioms

$\Sigma(<\phi>) = \Sigma(\phi)$ (S1)

$\Sigma(\mathrm{T}(x)) = x$ (S2)

$\Sigma(X+Y) = \Sigma(X)+\Sigma(Y)$ (S3)

$\Sigma(x \Box Y) = x \cap \Sigma(Y)$ (S4)

$\Sigma(r.X) = r.\Sigma(X)$ (S5)

$r.<\phi> = <r.\phi>$ (R1)

$r.\mathrm{T}(x) = \mathrm{T}(r.x)$ (R2)

$r.(X+Y) = (r.X)+(r.Y)$ (R3)

$r.(x \Box Y) = (r.x) \Box (r.Y)$ (R4)

$r.(r.X) = X$ (R5)

$\Sigma(r) \cap \Sigma(X) = \mathrm{inv}\Sigma(r) \Rightarrow r.X = X$ (R6)

$X+Y = Y+X$ (C1)

$(X+Y)+Z = X+(Y+Z)$ (C2)

$\mathrm{T}(x+y) = \mathrm{T}(x)+\mathrm{T}(y)$ (C3)

$X+\mathrm{T}(\Sigma(X)) = X$ (C4)

$X+(y \Box X) = X$ (C5)

$\Sigma(X) \Box X = X$ (E1)

$x \Box (y \Box Z) = (x \cap y) \Box Z$ (E2)

$x \Box (\mathrm{T}(y)+Z) = \mathrm{T}(x \cap y)+(x \Box Z)$ (E3)

$x = (\Sigma(Y) \cap \Sigma(Z))+x' \Rightarrow$
$x \Box (Y+Z) = (x \Box Y)+(x \Box Z)$ (E4)

end *BMA[fol]*

FIGURE 3.2. Basic Module Algebra.

COMMENTS:
(S1)-(S5) are the natural identities for Σ.

(R1)-(R3) are self-evident.

(R4) is valid because of the permutative character of renaming.

(R5) says that renaming is an *involution*.

(R6) postulates (restricted) renameability of hidden items. The condition

$$\Sigma(r) \cap \Sigma(X) = \text{inv}\Sigma(r)$$

does not allow renaming of items that are visible, or renaming of hidden items causing a clash between hidden and visible names. Clashes between hidden names cannot happen due to the permutative character of renaming. The following equation is equivalent to (R6) and derivable from *BMA*[*fol*]:

$$(\Sigma(r) \cap \Sigma(X)) + y = \text{inv}\Sigma(r) \Rightarrow r.X = X. \qquad \text{(R6')}$$

(C1) and (C2) together with the equation $X+X = X$ express the fact that modules are *sets* of axioms. The idempotent law for $+$ is a special case of (C5) (take $y = \Sigma(X)$ and apply (E1)).

(C3)-(C4) are self-evident.

(C5) is a generalization of the idempotent law for $+$, expressing the fact that enrichment is a special case of extension (see SECTION 3.4).

(E1)-(E2) are self-evident.

(E3) is closely related to equation (7) below which is a special case in which $\Box$ distributes over $+$. It has been included for technical purposes and is used in SECTION 3.3.

(E4) postulates (restricted) distribution of $\Box$ over $+$. Of course, it would be nice if we would have unrestricted distributivity. But this is simply not true in the models of *BMA*[*fol*] we have in mind. Consider the following simple counterexample (see SECTIONS 2.2 and 3.5 for the notation used):

$$x = \mathbf{S}{:}B + \mathbf{F}{:}T{:}B + \mathbf{F}{:}F{:}B$$
$$Y = (x + \mathbf{F}{:}c{:}B){:}(<T{=}c>)$$
$$Z = (x + \mathbf{F}{:}c{:}B){:}(<F{=}c>).$$

In this case we have $x \Box (Y+Z) \neq (x \Box Y) + (x \Box Z)$ both from an intuitive point of view as well as in all models of *BMA*[*fol*] discussed in SECTION 4.2. The point is that $x \Box (Y+Z)$ implies $T = F$ while $(x \Box Y) + (x \Box Z)$ does not, as one may choose $c = T$ in $x \Box Y$ and $c = F$ in $x \Box Z$.

Some equations that are equationally derivable or ω-derivable (SECTION 2.2) from *BMA*[*fol*] are:

(1) $x \Box \mathrm{T}(\varnothing) = \mathrm{T}(\varnothing)$
(2) $x \Box \mathrm{T}(y) = \mathrm{T}(x \cap y)$
(3) $X + \mathrm{T}(\varnothing) = X$
(4) $\Sigma(X) \cap \Sigma(Y) = \varnothing \;\&\; \varnothing \Box Y = \mathrm{T}(\varnothing) \Rightarrow \Sigma(X) \Box (X+Y) = X$
(The second part of the condition means that Y is consistent. See SECTION 5.2.)
(5) $(\Sigma(X) + y) \Box X = X$
(6) $\Sigma(X) \Box (\mathrm{T}(y) + X) = X$
(7) $x \Box (\mathrm{T}(y) + Z) = (x \Box \mathrm{T}(y)) + (x \Box Z)$.

If the equation $x + (x \cap y) = x$ is added to *Signatures* (this does not affect its initial model), all of the

above equations become equationally derivable. Hence, they are valid in all models of *BMA*[*fol*] that do not contain non-standard signatures, and in particular in the models discussed in SECTION 4.2.

3.2. The normal form theorem

In the sequel *ME*[*fol*] will be the set of *module expressions*, i.e., expressions of sort *M* over the signature of *BMA*[*fol*], and *CME*[*fol*] will be the set of closed module expressions in *ME*[*fol*].

DEFINITION 3.2.1: A term $X \in CME[fol]$ is *flat* if it does not contain the $\Box$-operator.

The class of flat closed module expressions will be called *FCME*[*fol*].

THEOREM 3.2.1: For every $X \in FCME[fol]$ there is an $X' \in FCME[fol]$ of the form

$$\sum_{i=1}^{n} <\phi_i> + \mathrm{T}(x) \qquad (n \geqslant 0, \text{ the summand } \mathrm{T}(x) \text{ may be absent})$$

such that $BMA[fol] \vdash X = X'$.

PROOF: By structural induction using axioms (R1)-(R3) and (C2)-(C4).

DEFINITION 3.2.2: A term $X \in CME[fol]$ is in *normal form* if X has the form $\Sigma \Box Y$ with Σ a signature and Y flat.

THEOREM 3.2.2: (*Normal form theorem*) Each $X \in CME[fol]$ has a normal form $X' \in CME[fol]$ such that $BMA[fol] \vdash X = X'$.

For $V, W \subseteq CME[fol]$ we write

$BMA[fol] \vdash V \subseteq W$, if for all $X \in V$ there is an $Y \in W$ with $BMA[fol] \vdash X = Y$;
$BMA[fol] \vdash V = W$, if $BMA[fol] \vdash V \subseteq W$ and $BMA[fol] \vdash W \subseteq V$.

Using this notation the normal form theorem can be restated very simply:

$$BMA[fol] \vdash CME[fol] = SIG \Box FCME[fol].$$

For the proof of the normal form theorem we first need a lemma:

LEMMA 3.2.1: Let Σ, Σ' be signatures and $X \in FCME[fol]$. Then there is an $X' \in FCME[fol]$ such that $BMA[fol] \vdash \Sigma \Box X = (\Sigma + \Sigma') \Box X'$.

PROOF: $\Sigma \Box X$ is transformed into $\Sigma \Box X'$ by means of a series of atomic renamings which successively replace all names occurring in $\Sigma(X)$ but not in Σ by names not occurring in Σ'. For such X' we have $BMA[fol] \vdash \Sigma \Box X = (\Sigma + \Sigma') \Box X'$.

PROOF OF THE NORMAL FORM THEOREM: Let the $\Box$-depth of an $X \in CME[fol]$ be defined as follows:

$d(X) = 0$ if $X \in FCME[fol]$
$d(X + Y) = \max(d(X), d(Y))$
$d(\Sigma \Box X) = 1 + d(X)$.

We use induction on the $\Box$-depth. If $d(X) = 0$, X is flat and $X = \Sigma(X) \Box X$. Now assume that for some $n \geqslant 0$ all $X \in CME[fol]$ with $d(X) \leqslant n$ can be brought into the required normal form.
If $X = \Sigma \Box X'$ with $d(X') = n$, first normalize X' to $\Sigma' \Box X''$ with $X'' \in FCME[fol]$. This immediately leads to normalization of X:

$$X = \Sigma \Box (\Sigma' \Box X'') = (\Sigma \cap \Sigma') \Box X''.$$

The remaining case is

$$X = \sum_{i=1}^{k}(\Sigma_i \square X_i)$$

with $d(X_i) \leqslant n$. (If one of the summands X_i of X is flat, we write $X_i = \Sigma(X_i) \square X_i$.) Using induction we may normalize X_i to $\Sigma_i' \square X_i'$ with $X_i' \in FCME[fol]$ $(1 \leqslant i \leqslant k)$, and we obtain

$$X = \sum_{i=1}^{k}(\Sigma_i \cap \Sigma_i') \square X_i'.$$

Let $\Sigma = \sum_{i=1}^{k}(\Sigma_i \cap \Sigma_i')$. Using lemma 3.2.1 we find for each i an X_i'' such that $(\Sigma_i \cap \Sigma_i') \square X_i' = \Sigma \square X_i''$, hence

$$X = \sum_{i=1}^{k}(\Sigma \square X_i'').$$

In order to normalize this sum it suffices to write any term of the form $(\Sigma \square Y_1) + (\Sigma \square Y_2)$ as $\Sigma \square Y_3$ (with Y_1, Y_2, Y_3 flat). Performing this transformation $k-1$ times leads to the required normal form. So consider

$$Y = (\Sigma \square Y_1) + (\Sigma \square Y_2).$$

We want to apply (E4). Using lemma 3.2.1 we obtain a Y_2' such that

$$\Sigma \square Y_2 = (\Sigma + \Sigma(Y_1)) \square Y_2'.$$

Taking the signature of both sides gives

$$\Sigma \cap \Sigma(Y_2) = (\Sigma + \Sigma(Y_1)) \cap \Sigma(Y_2') \supseteq \Sigma(Y_1) \cap \Sigma(Y_2'),$$

so

$$\Sigma \supseteq \Sigma(Y_1) \cap \Sigma(Y_2').$$

Hence

$$\begin{aligned}
&(\Sigma + \Sigma(Y_1)) \square Y_2' = \\
&(\Sigma + \Sigma(Y_1)) \square (\Sigma(Y_2') \square Y_2') = \\
&((\Sigma + \Sigma(Y_1)) \cap \Sigma(Y_2')) \square Y_2' = \\
&((\Sigma \cap \Sigma(Y_2')) + (\Sigma(Y_1) \cap \Sigma(Y_2'))) \square Y_2' = \\
&(\Sigma \cap \Sigma(Y_2')) \square Y_2' = \\
&\Sigma \square (\Sigma(Y_2') \square Y_2') = \\
&\Sigma \square Y_2'.
\end{aligned}$$

Finally,

$$Y = (\Sigma \square Y_1) + (\Sigma \square Y_2) = (\Sigma \square Y_1) + (\Sigma \square Y_2') \overset{(E4)}{=} \Sigma \square (Y_1 + Y_2') = \Sigma \square Y_3.$$

3.3. Two additional module operators: hiding and common export

Two useful operators for constructing specifications are the *hiding operator* $\Delta : ATSIG \times M \to M$ defined by

$$a \Delta X = (a \Delta \Sigma(X)) \square X, \qquad \text{(H)}$$

and the *common export operator* $\square : M \times M \to M$ defined by

$$X \square Y = (\Sigma(X) \cap \Sigma(Y)) \square (X + Y). \qquad \text{(CE)}$$

The Δ- and $\square$-operators occurring in the right-hand side of (H) and (CE) are respectively the deletion operator $\Delta : ATSIG \times SIG \to SIG$ and the export operator $\square : SIG \times M \to M$. Hence, both

operators are defined in terms of operators of *BMA*[*fol*]. As such they are superfluous from a theoretical viewpoint and adding them to *BMA*[*fol*] would only complicate the theoretical development. They are useful in practice, however. Hiding is complementary to export, while common export is a generalization of export in the sense that

$$x \square Y = \mathrm{T}(x) \square Y.$$

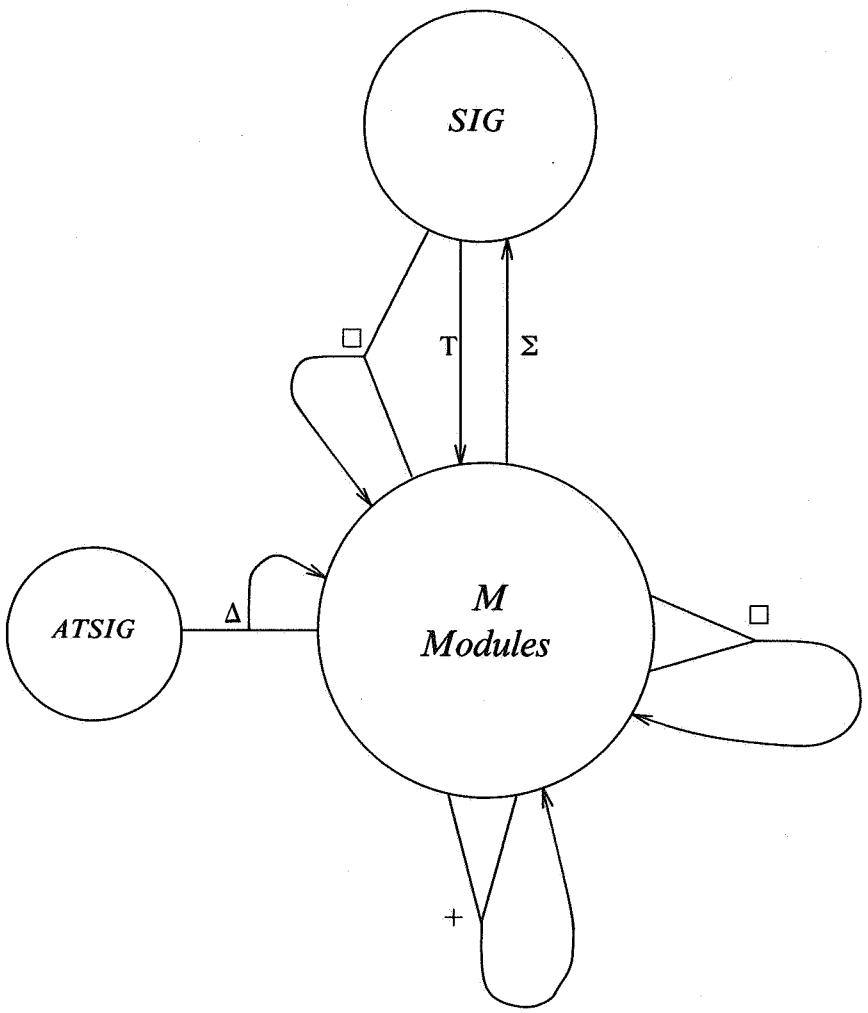

FIGURE 3.3. Extended signature for *BMA*[*fol*].
The hiding operator $\Delta : ATSIG \times M \rightarrow M$, and
the common export operator $\square : M \times M \rightarrow M$
have been added. Note the overloading of $\square$.

As C.P.J. Koymans pointed out to us, (E4) can be replaced by a remarkably symmetrical non-conditional equation if the common export operator is used in addition to the export operator:

$$(\Sigma(X) \square Y) + (\Sigma(Y) \square X) = X \square Y. \qquad \text{(E4*)}$$

PROOF: We first show that *BMA*[*fol*]+(CE) $\vdash$ (E4*).

$$(\Sigma(X)\Box Y)+(\Sigma(Y)\Box X) =$$
$$(\Sigma(X)\Box(\Sigma(Y)\Box Y))+(\Sigma(Y)\Box(\Sigma(X)\Box X)) =$$
$$((\Sigma(X)\cap\Sigma(Y))\Box Y)+((\Sigma(X)\cap\Sigma(Y))\Box X) = \text{(using (E4) with } x'=\varnothing)$$
$$(\Sigma(X)\cap\Sigma(Y))\Box(X+Y) \overset{(CE)}{=} X\Box Y.$$

Secondly, we show that $BMA[fol]-(E4)+(E4^*)+(CE) \vdash (E4)$.
Suppose $x = (\Sigma(Y)\cap\Sigma(Z))+x'$. Let $u' = x'\cap(\Sigma(Y)+\Sigma(Z))$ and $u = (\Sigma(Y)\cap\Sigma(Z))+u'$. Now

$$x\Box Y = x\Box(\Sigma(Y)\Box Y) = (x\cap\Sigma(Y))\Box Y =$$
$$(u\cap\Sigma(Y))\Box Y = u\Box(\Sigma(Y)\Box Y) = u\Box Y.$$

Similarly, $x\Box Z = u\Box Z$ and $x\Box(Y+Z) = u\Box(Y+Z)$, so we only have to prove that

$$u\Box(Y+Z) = (u\Box Y)+(u\Box Z).$$

To this end let $Y' = Y+\mathrm{T}(u)$ and $Z' = Z+\mathrm{T}(u)$.
It will be shown that

(*a*) $Y'\Box Z' = u\Box(Y+Z)$
(*b*) $Y'\Box Z' = (u\Box Y)+(u\Box Z)$.

First observe that

$$\Sigma(Y'\Box Z') = (\Sigma(Y')\cap\Sigma(Z'))\cap(\Sigma(Y')+\Sigma(Z')) =$$
$$\Sigma(Y')\cap\Sigma(Z') = (\Sigma(Y)+u)\cap(\Sigma(Z)+u) =$$
$$(\Sigma(Y)\cap\Sigma(Z))+(\Sigma(Y)\cap u)+(\Sigma(Z)\cap u)+u =$$
$$u+((\Sigma(Y)+\Sigma(Z))\cap u) = u.$$

To prove (*a*) we apply (CE):

$$Y'\Box Z' = u\Box(Y'+Z') = u\Box(Y+\mathrm{T}(u)+Z+\mathrm{T}(u)) =$$
$$u\Box(\mathrm{T}(u)+Y+Z) \overset{(E3)}{=} \mathrm{T}(u\cap u)+u\Box(Y+Z) = \mathrm{T}(u)+u\Box(Y+Z) =$$
$$\mathrm{T}(\Sigma(u\Box(Y+Z)))+(u\Box(Y+Z)) = u\Box(Y+Z).$$

To prove (*b*) we apply (E4*):

$$Y'\Box Z' = (\Sigma(Y')\Box Z')+(\Sigma(Z')\Box Y') =$$
$$((\Sigma(Y')\cap\Sigma(Z'))\Box Z')+((\Sigma(Y')\cap\Sigma(Z'))\Box Y') =$$
$$(u\Box Y')+(u\Box Z') =$$
$$(u\Box(Y+\mathrm{T}(u)))+(u\Box(Z+\mathrm{T}(u))) =$$
$$(u\Box(\mathrm{T}(u)+Y))+(u\Box(\mathrm{T}(u)+Z)) \overset{(E3)}{=}$$
$$\mathrm{T}(u\cap u)+(u\Box Y)+\mathrm{T}(u\cap u)+(u\Box Z) =$$
$$\mathrm{T}(u)+(u\Box Y)+\mathrm{T}(u)+(u\Box Z) =$$
$$\mathrm{T}(u)+(u\Box Y)+(u\Box Z) =$$
$$\mathrm{T}(u+(\Sigma(Y)\cap\Sigma(Z)))+(u\Box Y)+(u\Box Z) =$$
$$\mathrm{T}((u\cap\Sigma(Y))+(u\cap\Sigma(Z)))+(u\Box Y)+(u\Box Z) =$$
$$\mathrm{T}(\Sigma(u\Box Y)+\Sigma(u\Box Z))+(u\Box Y)+(u\Box Z) =$$
$$\mathrm{T}(\Sigma((u\Box Y)+(u\Box Z)))+(u\Box Y)+(u\Box Z) =$$
$$(u\Box Y)+(u\Box Z).$$

REMARK: It follows from the above proof that we may replace (E4) in $BMA[fol]$ by the slightly weaker axiom

$$x = \Sigma(Y)\cap\Sigma(Z) \Rightarrow x\Box(Y+Z) = (x\Box Y)+(x\Box Z). \qquad (E4^-)$$

3.4. Abstraction, enrichment, extension, and refinement

The theory of modular specification is relevant to the study of transformational program development. Both require a classification of the various possible construction/development steps. We will first discuss such a classification informally, and then give precise definitions of the notions involved in the context of module algebra.

Let $S: X \mapsto Y$ be a transformation step from a specification X to some other specification Y. In accordance with more or less established terminology we may say that

(1) S is an *abstraction* (Y is an abstraction of X) if Y is obtained by deleting (hiding) information from X.
(2) S is an *enrichment* (Y is an enrichment of X) if Y covers more issues than X without in any way changing or constraining the meaning of X.
(3) S is an *extension* (Y is an extension of X) if Y describes more than X in a way consistent with X and perhaps even in a more specific way than X. (An enrichment is a *conservative extension*.)
(4) S is a *refinement* (Y is a refinement of X) if Y describes the same as X but in a more specific way (essentially by adding additional constraints).

These informal definitions can be translated into precise ones for specifications $X, Y \in CME[fol]$ as follows.

DEFINITION 3.4.1: For $X, Y \in CME[fol]$ we say that

(1) Y is an *abstraction* of X if $Y = \Sigma(Y) \square X$;
(2) Y is an *enrichment* of X if X is an abstraction of Y;
(3) Y is an *extension* of X if $Y = Y + X$;
(4) Y is a *refinement* of X if Y is an extension of X, and $\Sigma(Y) = \Sigma(X)$.

COMMENTS:

(1) If $Y = \Sigma(Y) \square X$ (Y is an abstraction of X), Y is obtained by hiding information (in this case a part of the signature) from X.

(2) If $X = \Sigma(X) \square Y$ (Y is an enrichment of X), Y says more about new signature elements (i.e., sorts and functions in $\Sigma(Y) - \Sigma(X)$), but does not add any constraints to X.

(3) If $Y = Y + X$ (Y is an extension of X), Y says more than X.

(4) If $Y = Y + X$ and $\Sigma(Y) = \Sigma(X)$ (Y is a refinement of X), Y says more than X about the same signature.

The combination operation $X, Z \mapsto X + Z$ can be viewed as producing an extension $Y = X + Z$ of X. Furthermore, both enrichment and refinement are (simpler) forms of extension. Indeed, if Y is an enrichment of X then

$$X = \Sigma(X) \square Y,$$

and hence

$$Y + X = Y + (\Sigma(X) \square Y) \overset{(C5)}{=} Y.$$

Refinement is by definition a special case of extension.

Every extension can be split into a refinement and an enrichment:

LEMMA 3.4.1: (*Factorisation lemma*) For any extension $X \mapsto Z$ with $X, Z \in CME[fol]$ there is a $Y \in CME[fol]$ such that $X \mapsto Y$ is a refinement and $Y \mapsto Z$ is an enrichment. (See FIGURE 3.4).

PROOF: Let $Y = \Sigma(X) \square Z$. We must verify that

(*a*) $Y = Y + X$
(*b*) $\Sigma(Y) = \Sigma(X)$
(*c*) $Y = \Sigma(Y) \square Z$.

(*a*) $Y = \Sigma(X) \square Z = \Sigma(X) \square (Z + X) = (\Sigma(X) \square Z) + (\Sigma(X) \square X) = (\Sigma(X) \square Z) + X = Y + X$.

(*b*) $\Sigma(Y) = \Sigma(X) \cap \Sigma(Z) = \Sigma(X) \cap \Sigma(X + Z) = (\Sigma(X) \cap \Sigma(X)) + (\Sigma(X) \cap \Sigma(Z)) = \Sigma(X)$.

(*c*) Immediate from (*b*) and the definition of Y.

REMARK: Of course, whether an extension is an enrichment or not depends on the semantics used. It is quite possible that an extension $X \mapsto Y$ is an enrichment in $\mathbb{M}_C(fol)$ but not in $\mathbb{M}(fol)$ (see SECTION 4.2).

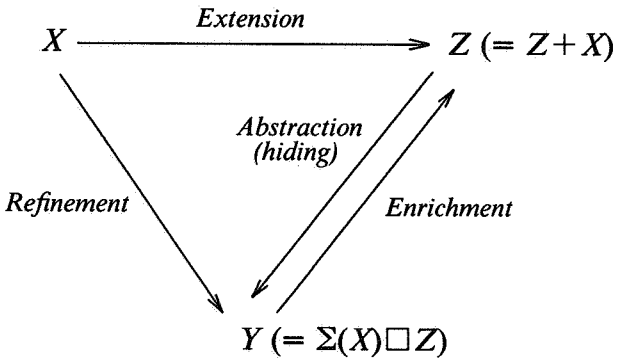

FIGURE 3.4. Graphical summary of DEFINITION 3.4.1 and the factorization lemma.

3.5. Notational conventions

In the examples and proofs in this paper we will use the operators :, +:, □: and Δ: in addition to +, □ and Δ. These do not occur in *BMA*[*fol*], but in many cases allow us to drop the explicit typing from axioms (see SECTION 2.1) and to avoid having to duplicate signatures in module expressions.

For $x:Y$ to be meaningful, x must be an *unambiguous* signature in which each name occurs at most once as a function name of arity n ($n \geqslant 0$). Thus, the signature $\mathbf{F}:S:M \to M + \mathbf{F}:S:N \to N$ is ambiguous, but $\mathbf{F}:S:M + \mathbf{F}:S:N \to N$ is not. Now $x:Y$ means that whenever a function symbol f of arity n occurs without explicit typing in an axiom of Y and x contains $\mathbf{F}:f:A_1 \times \cdots \times A_n \to A$, then f is an abbreviation of

$$f^{A_1 \times \cdots \times A_n \to A}.$$

Here Y must be viewed purely syntactically rather than as an expression subject to the laws of module algebra. Note, that the arity of a function symbol can always be determined uniquely from the context in which the symbol occurs.

In addition to using the above abbreviation operators, we omit universal quantifiers from equations and do not declare variables. The type of variables must be inferred from the context in which they occur. For instance,

$$(\mathbf{S}{:}B+\mathbf{F}{:}T{:}B+\mathbf{F}{:}F{:}B){:}(<T\neq F>+<x=T\vee x=F>) \equiv$$
$$<T^B\neq F^B>+<\forall x^B\ x^B=T^B\vee x^B=F^B>.$$

$$(\mathbf{S}{:}N+\mathbf{F}{:}0{:}N+\mathbf{F}{:}S{:}N\to N+\mathbf{F}{:}+{:}N\times N\to N){:}$$
$$(<x+0=0>+<x+S(y)=S(x+y)>) \equiv$$
$$<\forall x^N\ x^N+^{N\times N\to N}0^N=x^N>+$$
$$<\forall x^N\forall y^N\ x^N+^{N\times N\to N}S^{N\to N}(y^N)=S^{N\to N}(x^N+^{N\times N\to N}y^N)>.$$

Now we can define $+{:}$, $\square{:}$ and $\Delta{:}$ as follows:

$$X+{:}Y = X+(\Sigma(X){:}Y)$$
$$x\square{:}Y = x\square(x{:}Y)$$
$$a\Delta{:}Y = a\Delta(i(a){:}Y).$$

4. Semantics of *BMA*[*fol*]

4.1. Notational conventions

$\mathscr{L}(\Sigma)$ = the set of closed first-order formulas of signature Σ.
($\mathscr{L}(\varnothing)$ contains the 0-ary connectives **true** and **false**.)

$Alg(\Sigma)$ = the class of all Σ-algebras.
($Alg(\varnothing) = \{A_\varnothing\}$, where $A_\varnothing$ is the unique "empty" algebra.)

$Alg(\Sigma,\phi)$ = the class of all Σ-algebras satisfying a formula $\phi\in\mathscr{L}(\Sigma)$.

$Alg_C(\Sigma)$ = the class of all countable Σ-algebras.

$Alg_C(\Sigma,\phi)$ = the class of all countable Σ-algebras satisfying a formula $\phi\in\mathscr{L}(\Sigma)$.
($Alg_C(\Sigma,\phi) = Alg(\Sigma,\phi)\cap Alg_C(\Sigma)$.)

$LCT(\Sigma)$ = the set of logically closed theories over Σ,
i.e., subsets of $\mathscr{L}(\Sigma)$ which are closed under logical deduction.
(Note that $T\in LCT(\Sigma)$ always contains **true** and hence is never empty.
Furthermore, Σ can always be completely recovered from T as all
constant and function symbols occur in the various tautologies
which must always be present in T.)

$Th(\Sigma)$ = $\{\phi\in\mathscr{L}(\Sigma) \mid \vdash\phi\}$ (= the smallest element of $LCT(\Sigma)$).

$Th(\Sigma,\phi)$ = $\{\psi\in\mathscr{L}(\Sigma) \mid \phi\vdash\psi\}$ for $\phi\in\mathscr{L}(\Sigma)$ (= the smallest element of $LCT(\Sigma)$ containing ϕ).

$Th(\Sigma,K)$ = $\{\phi\in\mathscr{L}(\Sigma) \mid \forall A\in K\ A\models\phi\}$ with $K\subseteq Alg(\Sigma)$.

We only consider algebras with non-empty carriers. It is assumed that Σ is exported from each Σ-algebra.

$\Sigma\square A$ = the restriction of A to $\Sigma\cap\Sigma'$ with $A\in Alg(\Sigma')$. (If $\Sigma\cap\Sigma'=\varnothing$, then $\Sigma\square A = A_\varnothing$.)

$\Sigma\square K$ = $\{\Sigma\square A \mid A\in K\}$ with $K\subseteq Alg(\Sigma')$.

$K+L$ = $\{A\in Alg(\Sigma_1+\Sigma_2) \mid \Sigma_1\square A\in K,\ \Sigma_2\square A\in L\}$ with $K\subseteq Alg(\Sigma_1)$, $L\subseteq Alg(\Sigma_2)$.

$\Sigma\square T$ = $\mathscr{L}(\Sigma)\cap T$ for $T\in LCT(\Sigma')$.

$T+U$ = $\{\phi\in\mathscr{L}(\Sigma_1+\Sigma_2) \mid T\cup U\vdash\phi\}$ for $T\in LCT(\Sigma_1)$, $U\in LCT(\Sigma_2)$.

Using the above notation we define three semantical mappings *Mod*, Mod_C, and *Th* on *CME*[*fol*] as follows (with $X\in CME[fol]$):

(1) $X\mapsto Mod(X) \subseteq Alg(\Sigma(X))$

(2) $X\mapsto Mod_C(X) \subseteq Alg_C(\Sigma(X))$

(3) $X\mapsto Th(X) \subseteq LCT(\Sigma(X))$.

The precise inductive definitions are as follows:

(1) $Mod(<\phi>) = Alg(\Sigma(<\phi>),\phi)$
$Mod(\mathrm{T}(x)) = Alg(x)$
$Mod(X+Y) = Mod(X)+Mod(Y)$
$Mod(\Sigma\Box X) = \Sigma\Box Mod(X)$

(2) $Mod_C(<\phi>) = Alg_C(\Sigma(<\phi>),\phi)$
$Mod_C(\mathrm{T}(x)) = Alg_C(x)$
$Mod_C(X+Y) = Mod_C(X)+Mod_C(Y)$
$Mod_C(\Sigma\Box X) = \Sigma\Box Mod_C(X)$

(3) $Th(<\phi>) = Th(\Sigma(<\phi>),\phi)$
$Th(\mathrm{T}(x)) = Th(x)$
$Th(X+Y) = Th(X)+Th(Y)$
$Th(\Sigma\Box X) = \Sigma\Box Th(X)$.

An equivalence relation can be associated with each of these three mappings in the following straightforward manner:

$$X \equiv_{Mod} Y \Leftrightarrow \Sigma(X)=\Sigma(Y) \;\&\; Mod(X)=Mod(Y)$$
$$X \equiv_{Mod_c} Y \Leftrightarrow \Sigma(X)=\Sigma(Y) \;\&\; Mod_C(X)=Mod_C(Y)$$
$$X \equiv_{Th} Y \Leftrightarrow \Sigma(X)=\Sigma(Y) \;\&\; Th(X)=Th(Y).$$

4.2. Four models of BMA[fol]

Being an algebraic specification, *BMA*[*fol*] has an initial model $\mathbb{I}(BMA[fol])$. It essentially consists of the textual representations (presentations) of modular specifications modulo a rather weak congruence which is nevertheless strong enough to make the normal form theorem (THEOREM 3.2.2) work. As a result, $\mathbb{I}(BMA[fol])$ is a computable algebra which could actually be implemented as part of a system for manipulating specifications.

Three further models of *BMA*[*fol*] can be obtained by factorizing *CME*[*fol*] with respect to the three equivalence relations $\equiv_{Mod}$, $\equiv_{Mod_c}$, and $\equiv_{Th}$ on *ME*[*fol*] introduced in the previous section. In fact, all of them are congruences on *CME*[*fol*], so we may write

$$\mathbb{M}(fol) = CME[fol]/\equiv_{Mod}$$
$$\mathbb{M}_C(fol) = CME[fol]/\equiv_{Mod_c}$$
$$\mathbb{T}(fol) = CME[fol]/\equiv_{Th}.$$

Furthermore, it can be verified that each of these three constructions is a (minimal) model of *BMA*[*fol*]. All verifications involved are straightforward except the verification of $\mathbb{T}(fol) \models$ (E4), which turns out to be equivalent to the CRAIG interpolation lemma. We will return to this later (THEOREM 4.2.2).

The relations between $\mathbb{M}(fol)$, $\mathbb{M}_C(fol)$, and $\mathbb{T}(fol)$ are as follows. For $X \in CME[fol]$

(*a*) $Mod_C(X) = Mod(X) \cap Alg_C(\Sigma(X))$

(*b*) $Th(X) = Th(\Sigma(X), Mod_C(X))$.

The proof of (*a*) uses the downward LÖWENHEIM-SKOLEM theorem and (*b*) is based on the completeness theorem. Both proofs use the normal form theorem by assuming that X is in normal form.

Furthermore, it follows that

$$X \equiv_{Mod} Y \Rightarrow X \equiv_{Mod_C} Y \Rightarrow X \equiv_{Th} Y,$$

which implies that $\mathbb{M}_C(fol)$ is a homomorphic image of $\mathbb{M}(fol)$ and that $\mathbb{T}(fol)$ is a homomorphic image of $\mathbb{M}_C(fol)$.

Let $X, Y \in FCME[fol]$. We have trivially

$$X \equiv_{Th} Y \Rightarrow X \equiv_{Mod} Y.$$

Hence, for flat module expressions the three semantics are equivalent. For non-flat expressions they are different, however:

THEOREM 4.2.1: $\mathbb{M}(fol) \not\cong \mathbb{M}_C(fol) \not\cong \mathbb{T}(fol)$.

Hence, there are homomorphisms $\phi_1 : \mathbb{M}(fol) \rightarrow \mathbb{M}_C(fol)$ and $\phi_2 : \mathbb{M}_C(fol) \rightarrow \mathbb{T}(fol)$ which are surjective but not injective.

PROOF: We first prove $\mathbb{M}(fol) \not\cong \mathbb{M}_C(fol)$ by giving a pair of closed module expressions $X, Y \in CME[fol]$ such that

$$\mathbb{M}_C \models X = Y,$$

but

$$\mathbb{M} \not\models X = Y.$$

Let NA and NB be defined as follows:

$$NA = (\mathbf{S}{:}A + \mathbf{F}{:}S_A{:}A \rightarrow A + \mathbf{F}{:}0_A{:}A): \\ (<\forall x^A \forall y^A\ S_A(x) = S_A(y) \Rightarrow x = y> + <\forall x^A\ S_A(x) \neq 0_A>)$$

$$NB = (\mathbf{S}{:}B + \mathbf{F}{:}S_B{:}B \rightarrow B + \mathbf{F}{:}0_B{:}B): \\ (<\forall x^B \forall y^B\ S_B(x) = S_B(y) \Rightarrow x = y> + <\forall x^B\ S_B(x) \neq 0_B>).$$

(See SECTIONS 2.2 and 3.5 for the notation used.)
Take

$$X = (\mathbf{S}{:}A + \mathbf{S}{:}B) \square (NA + NB).$$

Let

$$Z = X + ((\mathbf{F}{:}f{:}A \rightarrow B + \mathbf{F}{:}g{:}B \rightarrow A){:}(<\forall x^A\ gf(x) = x> + <\forall x^B\ fg(x) = x>))$$

and

$$Y = (\mathbf{S}{:}A + \mathbf{S}{:}B) \square Z.$$

We must verify that

(*a*) $\mathbb{M}_C(fol) \models X = Y$ and
(*b*) $\mathbb{M}(fol) \not\models X = Y$.

(*a*) By construction $Alg_C(Y) \subseteq Alg_C(X)$, and if $\mathfrak{A} \in Alg_C(X)$ its carriers A and B are both countable and infinite. Therefore $\mathfrak{A}$ can be enriched with a function $f{:}A \rightarrow B$ and its inverse g, so $Alg_C(X) \subseteq Alg_C(Y)$.
(*b*) We show that $Alg(Y) \neq Alg(X)$. Indeed let $\mathfrak{A}$ be a structure in which A and B are both infinite but of different cardinality, then $\mathfrak{A} \in Alg(X)$; but as the carriers A' and B' of an $\mathfrak{A}' \in Alg(Y)$ have the same cardinality, $\mathfrak{A} \notin Alg(Y)$.

Secondly, we prove $\mathbb{M}_C(fol) \not\cong \mathbb{T}(fol))$ by giving $X, Y \in CME[fol]$ such that

$$\mathbb{T}(fol) \models X = Y,$$

but

$$\mathbb{M}_C(fol) \not\models X = Y.$$

Take

$$X = (\mathbf{S}{:}N + \mathbf{F}{:}0{:}N + \mathbf{F}{:}S{:}N{\to}N){:}(<\forall x^N \forall y^N\ S(x){=}S(y) \Rightarrow x{=}y> + <\forall x^N\ S(x){\neq}0>)$$
$$bool = (\mathbf{S}{:}B + \mathbf{F}{:}T{:}B + \mathbf{F}{:}F{:}B){:}(<T{\neq}F> + <\forall x^B\ x{=}T \vee x{=}F>)$$
$$Z = X + {:}bool + {:}((\mathbf{F}{:}ST{:}N{\to}B + \mathbf{F}{:}c{:}N){:}$$
$$(<ST(0){=}T> + <\forall x^N\ ST(x){=}ST(S(x))> + <ST(c){=}F>))$$
$$Y = \Sigma(X)\Box Z.$$

By construction $Alg_C(Y) \subseteq Alg_C(X)$. In fact,

$$Alg_C(Y) = Alg_C(X) - \{\mathbb{N}\},$$

where $\mathbb{N}$ is the standard model of X. Hence,

$$\mathbb{M}_C(fol) \not\models X = Y.$$

For

$$\mathbb{T}(fol) \models X = Y$$

first notice that by construction

$$Th(X) \subseteq Th(Y).$$

Now suppose $p \in Th(Y)$. We show that $p \in Th(X)$. For this it suffices to verify $\mathbb{N} \models p$. Let $\mathbb{N}'$ be any extension of $\mathbb{N}$ which is elementarily equivalent to $\mathbb{N}$. $\mathbb{N}'$ must be non-standard, therefore $\mathbb{N}' \in Alg_C(Y)$ and $\mathbb{N}' \models p$. But this implies $\mathbb{N} \models p$.

THEOREM 4.2.2: $\mathbb{T}(fol) \models$ (E4).

PROOF: (Throughout this proof $p, q, q_1, \ldots$ denote first-order sentences.)
Suppose $x \supseteq \Sigma(Y) \cap \Sigma(Z)$. We show that

$$x\Box(Y+Z) \equiv_{Th} (x\Box Y) + (x\Box Z).$$

(*a*) $\Sigma(x\Box(Y+Z)) = x \cap (\Sigma(Y) + \Sigma(Z)) = \Sigma((x\Box Y) + (x\Box Z))$.

(*b*) $Th(x\Box(Y+Z)) \supseteq Th((x\Box Y) + (x\Box Z))$:
Let $p \in Th((x\Box Y) + (x\Box Z))$. Choose $q \in Th(x\Box Y)$, $r \in Th(x\Box Z)$ with $q \wedge r \vdash p$. Clearly, $q, r \in Th(x\Box(Y+Z))$, hence $p \in Th(x\Box(Y+Z))$.

(*c*) $Th(x\Box(Y+Z)) \subseteq Th((x\Box Y) + (x\Box Z))$:
Let $p \in Th(x\Box(Y+Z))$, choose $q_1 \in Th(Y)$, $q_2 \in Th(Z)$ with $q_1 \wedge q_2 \vdash p$, then $q_1 \vdash q_2 \Rightarrow p$. Now $q_1 \in \mathscr{L}(\Sigma(Y))$, $q_2 \Rightarrow p \in \mathscr{L}(x + \Sigma(Z))$. According to the CRAIG interpolation lemma [SHO67, §5.4] there is an $r \in \mathscr{L}(x)$ such that $q_1 \vdash r$, $r \vdash q_2 \Rightarrow p$. Hence $q_1 \vdash r$ and $q_2 \vdash r \Rightarrow p$, and therefore $r \in Th(x\Box Y)$ and $r \Rightarrow p \in Th(x\Box Z)$ which implies $p \in Th((x\Box Y) + (x\Box Z))$.

REMARK: Conversely, (E4) implies the CRAIG interpolation lemma. Suppose $p \in \mathscr{L}(x)$, $q \in \mathscr{L}(y)$ such that $\vdash p \Rightarrow q$. Let $z = x \cap y$. Now $z\Box(<p> + <\neg q>) \equiv_{Th} (z\Box<p>) + (z\Box<\neg q>)$. Consequently $\mathbf{false} \in Th((z\Box<p>) + (z\Box<\neg q>))$. Choose $r_1 \in Th(z\Box<p>)$, $r_2 \in Th(z\Box<\neg q>)$ with $r_1 \wedge r_2 \vdash \mathbf{false}$. Then $p \vdash r_1 \vdash \neg r_2 \vdash q$ and we may use r_1 as interpolant.

In summary we may say that each of the four semantics discussed in this section has some interesting property. The initial semantics $\mathbb{I}(BMA[fol])$ is close to an implementation of the formalism; $\mathbb{M}(fol)$ corresponds to what seems to be the most general intuition of module composition; $\mathbb{M}_C(fol)$ is different from $\mathbb{M}(fol)$ showing that first-order logic with hidden sorts and functions is strictly more powerful than conventional "flat" first-order logic; and, finally, $\mathbb{T}(fol)$ is mathematically manageable and a potential candidate for becoming a standard semantics of module composition operators.

5. Equational logic from the viewpoint of module algebra

5.1. Conditional equations do not add expressive power

In the sequel *eql* means equational logic, and *ceql* means conditional equational logic.

Theorem 5.1.1: $\mathbb{M}(fol) \models CME[ceql] = CME[eql]$. (See Section 3.2 for the notation used.)

Proof: Evidently $CME[eql] \subseteq CME[ceql]$, which leaves the case $CME[eql] \supseteq CME[ceql]$ to be proved. For a $T \in CME[ceql]$ we have to find a $T' \in CME[eql]$ such that

$$\mathbb{M}(fol) \models T = T'.$$

We only have to consider T of the form $<\phi>$ where ϕ is a conditional equation. We assume ϕ has only a single condition. Multiple conditions can be dealt with in a similar way.
Let

$$\phi \equiv t_1^S = t_2^S \Rightarrow t_3^U = t_4^U.$$

Choose a new function symbol $h : S \times S \times U \to U$. Let

$$\begin{aligned} p &\equiv h(x,x,u) = u \\ q &\equiv h(t_1,t_2,t_3) = h(t_1,t_2,t_4) \\ X &= (\mathbf{F}{:}h{:}S \times S \times U \to U){:}(<p> + <q>) \\ Y &= \Sigma(<\phi>)\Box X. \end{aligned}$$

Now $\mathbb{M}(fol) \models <\phi> = Y$ and $Y \in CME[eql]$. Indeed, because $p,q \vdash \phi$ we have on the one hand $Mod(<\phi>) \supseteq Mod(Y)$. On the other hand, each $A \in Mod(<\phi>)$ can be extended to a model $B \in Mod(X)$ as follows:

$$h(s_1,s_2,u) = \begin{cases} u \text{ if } s_1 = s_2 \\ u_0 \text{ if } s_1 \neq s_2 \end{cases}$$

where u_0 is some fixed element of U (in A). It follows that

$$Mod(<\phi>) \subseteq \Sigma(<\phi>)\Box Mod(X) = Mod(Y).$$

5.2. A comparison of the expressive power of first-order logic and equational logic

What is the precise difference between equational logic and first-order logic from the viewpoint of module algebra? The following observations on this problem are somewhat informal. We only give sketches of the proofs involved.

We first need the following five definitions:

$$\begin{aligned} boolcons &= (\mathbf{S}{:}B + \mathbf{F}{:}T{:}B + \mathbf{F}{:}F{:}B){:}(<T \neq F>) \\ boolem &= (\mathbf{S}{:}B + \mathbf{F}{:}T{:}B + \mathbf{F}{:}F{:}B){:}(<\forall x^B\ x = T \vee x = F>) \\ bool &= boolcons + boolem \\ boolincons &= (\mathbf{S}{:}B + \mathbf{F}{:}T{:}B + \mathbf{F}{:}F{:}B){:}(<T = F>) \\ incons &= boolcons + boolincons. \end{aligned}$$

Boolcons expresses consistency, *boolem* expresses the law of the excluded middle, and *boolincons* expresses inconsistency. The following disjunction holds in $\mathbb{M}(fol)$ (and hence also in $\mathbb{M}_C(fol)$ and $\mathbb{T}(fol)$) for each $X \in CME[fol]$:

(*a*) $\varnothing\Box X = \mathrm{T}(\varnothing)$, or
(*b*) $\varnothing\Box X = \varnothing\Box incons.$

In case (*a*) we may say that X is consistent and in case (*b*) that it is inconsistent. Note that both *boolincons* itself as well as *boolem* + *boolincons* are consistent.

We now prove that *boolcons* and *boolem* are in some well-defined sense the only first-order specifications that do not have algebraic equivalents:

(1) $\mathbb{M}(fol) \not\models boolcons \in CME[eql]$.
PROOF: Let $X \in CME[eql]$. To see that $\mathbb{M}(fol) \not\models boolcons = X$ note that X always has a model which is trivial in the sense that each carrier has only a single element. Clearly, such a trivial $A \in Mod(X)$ is not an element of $Mod(boolcons)$.

(2) $\mathbb{M}(fol) \not\models boolem \in CME[eql]$.
PROOF: Let $X = \Sigma(bool) \square X' \in CME[eql]$ with X' flat. Assume $Mod(X) = Mod(boolem)$. Then $Mod(X')$ contains a model A with $A \not\models T = F$. Now let $\mathbf{F}:c:B \notin \Sigma(X')$. Consider the initial algebra I of $X' + \mathrm{T}(\mathbf{F}:c:B)$. If $I \models T = c$, then $Th(X') \vdash T = c$ and $Th(X') \vdash \forall x^B (x = T)$ which implies $Th(X') \vdash F = T$ contradicting $A \not\models T = F$. Similarly, $I \not\models F = c$. Consequently, sort B of $\Sigma(X') \square I$ has more than two elements. As $\Sigma(X') \square I \in Mod(X')$ by construction, $\Sigma(bool) \square I \in Mod(X)$, but $\Sigma(bool) \square I \notin Mod(boolem)$. This contradicts the assumption.

(3) $\mathbb{M}(fol) \not\models boolcons \in CME[eql, boolem]$.
PROOF: Similar to (1).

(4) $\mathbb{M}(fol) \not\models boolem \in CME[eql, boolcons]$.
PROOF: Similar to (2).

(5) $\mathbb{M}(fol) \models CME[fol] = CME[eql, bool]$.
PROOF: Existential quantifiers in $X \in CME[fol]$ are replaced by hidden Skolem functions. The resulting X' contains only universal axioms and is equivalent to X in $\mathbb{M}(fol)$.
Next, a hidden equality function $eq_S:S \times S \to B$ is introduced for each (hidden or visible) sort S of X'. Atomic formulae in the axioms of X' are replaced by equations over B ($t_1^S = t_2^S$ is replaced by $eq_S(t_1^S, t_2^S) = T$, $t_1^S \neq t_2^S$ is replaced by $eq_S(t_1^S, t_2^S) = F$).
Finally, the universal axioms of X' are replaced by equations over *bool* by means of hidden *bool*-operators like $\neg$, $\wedge$, and $\vee$.

COMMENTS:
(1) *Boolcons* and *boolem* are independent from the viewpoint of equational logic.

(2) A more interesting proof of (5) is based on a set of conditional rewrite rules (conditional equations) for transforming an arbitrary $X \in CME[fol]$ systematically into an $X' \in CME[eql, bool]$. An adequate presentation of such rules requires a thorough specification of first-order logic similar to our specification of signatures.

(3) There are two minor open questions:

(I) Let $X \in CME[eql, boolcons]$. Suppose that $\mathbb{M}(fol) \not\models X \in CME[eql]$. Does this imply $\mathbb{M}(fol) \models boolcons \in CME[eql, X]$?

(II) The same question as (I) but with *boolem* instead of *boolcons*.

What these questions amount to is whether *boolcons* and *boolem* are "primitive" or "minimal" if one works "modulo equational logic."

5.3. Why not base a model of BMA on equational logic?
Another semantic mapping may be considered which is like *Th* but produces an equational theory at the visible level rather than a full first-order theory. The most appropriate domain for this fifth semantic mapping *EqTh* is $CME[eql]$. *EqTh* is defined as follows ($Eq(\Sigma)$ denotes the set of Σ-equations):

$$EqTh(<\phi>) = Th(<\phi>) \cap Eq(\Sigma(<\phi>))$$
$$EqTh(X+Y) = (EqTh(X)+EqTh(Y)) \cap Eq(\Sigma(X+Y))$$
$$EqTh(\Sigma \square X) = \mathscr{L}(\Sigma) \cap EqTh(X).$$

Clearly, $EqTh(X) \subseteq Th(X)$. Let $X \equiv_{EqTh} Y$ if $\Sigma(X) = \Sigma(Y)$ and $EqTh(X) = EqTh(Y)$. We write

$$\mathbb{EQT}(eql) = CME[eql] / \equiv_{EqTh}.$$

It turns out that

$$\mathbb{EQT}(eql) \models BMA^{-}[eql],$$

where $BMA^{-} = BMA - (E4)$.

THEOREM 5.3.1: $\mathbb{EQT}(eql) \not\models (E4)$.

PROOF: This is a somewhat tricky consequence of the failure of the interpolation property for equational logic. Let

$$X_0 = (\mathbf{S}{:}A + \mathbf{F}{:}c{:}A + \mathbf{F}{:}f{:}A \to A){:}(<f(c)=c> + <ff(x)=x>)$$
$$X = (\mathbf{S}{:}A + \mathbf{F}{:}f{:}A \to A) \square X_0$$
$$Y = (\mathbf{S}{:}A + \mathbf{F}{:}f{:}A \to A + \mathbf{F}{:}h{:}A \times A \times A \to A + \mathbf{F}{:}c_1{:}A + \mathbf{F}{:}c_2{:}A){:}$$
$$(<h(x,x,y)=y> + <h(x,f(x),c_1)=h(x,f(x),c_2)>).$$

Now, if (E4) were valid we would have

$$\Sigma(Y) \square (X_0 + Y) = (\Sigma(Y) \square X_0) + (\Sigma(Y) \square Y) = ((\Sigma(Y) \cap \Sigma(X_0)) \square X_0) + Y = X + Y,$$

and

$$EqTh(X_0 + Y) \vdash h(c,f(c),c_1) = h(c,f(c),c_2)$$
$$EqTh(X_0 + Y) \vdash f(c) = c,$$

hence

$$EqTh(\Sigma(Y) \square (X_0 + Y)) \vdash c_1 = c_2.$$

We will show, however, that

$$EqTh(X+Y) \not\vdash c_1 = c_2.$$

LEMMA 5.3.1: $EqTh(X) \subseteq EqTh((\mathbf{S}{:}A + \mathbf{F}{:}f{:}A \to A){:}(<ff(x)=x>))$.

To see that $EqTh(X+Y) \not\vdash c_1 = c_2$ consider the following structure A:

$$|A| = \{a,b\},\ f(a)=b,\ f(b)=a,\ c_1 \to a,\ c_2 \to b,$$
$$h(x,y,z)=z \text{ if } x=y,\ h(x,y,z)=a \text{ if } x \neq y.$$

It follows from lemma 5.3.1 that $A \models EqTh(X)$, and $A \models EqTh(Y)$ by inspection, hence $A \models EqTh(X+Y)$, but $A \not\models c_1 = c_2$. Therefore $EqTh(X+Y) \not\vdash c_1 = c_2$.

Finally, lemma 5.3.1 has to be verified. Consider the structure D: $|D| = \{a,b,c\}$, $f(a)=b$, $f(b)=a$, $f(c)=c$, $c \to c$. $D \in Mod(X_0)$. Therefore

$$EqTh(X_0) \subseteq Th(D) \cap Eq(\Sigma(X_0)) \text{ and}$$
$$EqTh(X) \subseteq Th(\Sigma(X) \square D) \cap Eq(\Sigma(X)).$$

We will list all possible equations over $\Sigma(X) \square D$ and show that the ones that are valid in $\Sigma(X) \square D$ are derivable from $ff(x)=0$.

(*a*) $f^n(x) = f^m(y)$. These equations are not valid in $\Sigma(X) \square D$ as can be seen by taking $x=a$ and $y=c$.

(*b*) $f^n(x) = f^m(x)$ with $n \bmod 2 \neq m \bmod 2$. These equations are not valid in $\Sigma(X) \square D$ either, as

can be seen by taking $x = a$.

(c) $f^n(x) = f^m(x)$ with $n \bmod 2 = m \bmod 2$. These equations are valid in $\Sigma(X)\Box D$ and can easily be derived from $ff(x)=x$.

In view of the foregoing it can be concluded that

(1) $\mathbb{EQT}(eql)$ is a semantics of $CME[eql]$ in the weaker sense of BMA^-.

(2) The proof of the normal form theorem (THEOREM 3.2.2) does not apply to $\mathbb{EQT}(eql)$. This does not mean that the normal form theorem is not valid for expressions in $CME[eql]$. It may still be provable using recursion theoretic methods, but such a proof is unlikely to lead to an effective normalization procedure which is essential in a practical system.

(3) $\mathbb{EQT}(eql)$ is not a homomorphic image of any of the other models.

Although $\mathbb{EQT}(eql)$ may at first sight seem a very plausible semantics, the loss of (effective) normalization shows that it should be rejected.

Note that $EqTh(X) = Eq(\Sigma(X)) \cap Th(X)$ cannot always hold, because otherwise $\mathbb{EQT}(eql)$ would be a homomorphic image of $\mathbb{T}(eql)$ after all. So in general we have

$$EqTh(X) \subseteq Eq(\Sigma(X)) \cap Th(X).$$

Now the question is: can we define an initial algebra for specifications $X \in CME[eql]$ on the basis of the semantics $\mathbb{T}(eql)$? The answer is given by the following theorem which we do not prove here.

THEOREM 5.3.2: Let $X \in CME[eql]$, and let $Y_1, Y_2 \in FCME[eql]$ such that

$$BMA[eql] \vdash X = \Sigma(X)\Box Y_1$$
$$BMA[eql] \vdash X = \Sigma(X)\Box Y_2.$$

Let $A = I(Y_1)$, $B = I(Y_2)$, then

$$\Sigma(X)\Box A \cong \Sigma(X)\Box B.$$

COMMENTS:

(1) $I(Y)$ denotes the initial algebra of a $Y \in FCME[eql]$ provided $\Sigma(Y)$ does not have void (empty) sorts (see for instance [EM85] or [MG86]).

(2) The initial algebra of an $X \in CME[eql]$ is found as follows. Take some $Y \in FCME[eql]$ such that $BMA(eql) \vdash X = \Sigma(X)\Box Y$. Then $I(X) = \Sigma(X)\Box I(Y)$.

(3) Let $BMA(eql) \vdash X = \Sigma(X)\Box Y$ with $Y \in FCME[eql]$, then

(a) $Th(X) = Th(\Sigma(X)\Box Y) = \mathscr{L}(\Sigma(X)) \cap Th(Y)$

(b) $I(Y) \models Th(Y)$

(c) $\Sigma(X)\Box I(Y) \models \mathscr{L}(\Sigma(X)) \cap Th(Y)$

(d) $I(X) \models Th(X)$.

This indicates that the construction of $I(X)$ is consistent with the $\mathbb{T}(eql)$-semantics.

(4) The normalizing transformation which has to precede taking initial algebras is justified by the $\mathbb{T}(fol)$-semantics, which is not directly related to equational logic. This leads to open questions regarding rewrite rules and separate compilation of modular term rewriting systems.

5.4. Relations with earlier results on algebraic specification

(1) For a minimal algebra A with signature Σ the following two properties are equivalent

(a) A is semicomputable;

(b) A has an initial algebra specification with hidden sorts and functions, i.e., $A \cong \Sigma\Box I(X)$ for some $X \in FCME[eql]$.

The implication (b) $\Rightarrow$ (a) is immediate. The converse is proved in detail in [BT79] for the single-sorted case. It is an open question whether X can always be chosen in such a way that no hidden sorts are introduced, i.e., that $sorts(\Sigma) = sorts(\Sigma(X))$.

(2) If A is a minimal computable algebra with signature Σ, it has an initial algebra specification with

hidden functions only, i.e., there is an $X \in FCME[eql]$ such that
(*a*) $A \cong \Sigma \square I(X)$;
(*b*) $sorts(\Sigma) = sorts(\Sigma(X))$.
See [BT82]. MAJSTER [MAJ77] discovered that there are computable algebras for which there is no $X \in FCME[eql]$ such that $A \cong I(X)$.

In addition to (*a*) and (*b*) X can have several further properties (but not simultaneously):
(*c*) X has a complete (= confluent and terminating) term rewriting system. See [BT80] for a proof of the single-sorted case.
(*d*) Both the number of equations of X and the number of constants and functions of $\Sigma(X)$ are linearly bounded by the number of sorts of Σ. Moreover, $I(X)$ is also the final X-algebra which means that $I(X)$ does not have non-trivial homomorphic images. See [BT82]. (*Signatures* of SECTION 2.2 is an example of such an X for the algebra of signatures.)
(*e*) $\Sigma(X)$ has only unary hidden functions. A proof of the single-sorted case was given in [BKN80]. A special case involving finite algebras was discussed in [BM82].

(3) If A is a minimal cosemicomputable algebra with signature Σ, there is an $X \in FCME[ceql]$ such that
(*a*) X has a *unique* final algebra $F(X)$ (which in this case has the property that each of its homomorphic images satisfying X is either $F(X)$ itself or the trivial $\Sigma(X)$-algebra);
(*b*) $A \cong \Sigma \square F(X)$;
(*c*) $sorts(\Sigma) = sorts(\Sigma(X))$.
See [BT83].

(4) Let $f:\omega \to \omega$ be a recursive function. There is a term $T(X) \in FCME[eql](X)$ with a free variable X of sort M such that for all $n \in \omega$:

(*a*) $I(T(\Sigma_\omega : <S^n(0) = c>))$ is finite

(*b*) $\text{card}(I(T(\Sigma_\omega : <S^n(0) = c>))) > f(n)$,

where $\Sigma_\omega = \mathbf{S}:N + \mathbf{F}:0:N + \mathbf{F}:S:N \to N + \mathbf{F}:c:N$. See [BM81].

(5) The following example illustrates the fact that in the absence of hiding conditional equations are more powerful than non-conditional ones. Let

$$\Sigma_N = \mathbf{S}:N + \mathbf{F}:0:N + \mathbf{F}:S:N \to N$$
$$\Sigma_{SON} = \Sigma_N + \mathbf{S}:SETS + \mathbf{F}:\varnothing:SETS + \mathbf{F}:ins:N \times SETS \to SETS + \mathbf{F}:\#:SETS \to N.$$

$\mathbb{N} = I(\mathrm{T}(\Sigma_N))$ is the structure of natural numbers. It is enriched to a Σ_{SON}-algebra A by interpreting *SETS* as the collection of *finite* subsets of $\mathbb{N}$, $\varnothing$ as the empty set, *ins* as insertion, and # as the cardinality of a set. In [BM84] it is shown that $FCME[ceql]$ contains an X with $I(X) \cong A$, but that $FCME[eql]$ does not. Of course, in view of (2) (A is clearly computable) there also exists an $X \in CME[eql]$ such that $I(X) \cong A$.

(6) Let

$$\Sigma_N^P = \Sigma_N + \mathbf{F}:P:N \to N$$

where Σ_N is borrowed from (5). Enrich $\mathbb{N} = I(\mathrm{T}(\Sigma_N))$ to a Σ_N^P-algebra $\mathbb{N}_P$ by defining $P(n) = 1$ if n is prime and $P(n) = 0$ otherwise. In the revised version of [BT79] it is shown that there is no $X \in FCME[ceql]$ such that $I(X) \cong \mathbb{N}_P$, so $\mathbb{N}_P$ has no initial algebra specification without hidden functions.

6. Examples using equational logic

In this section we give a series of modular algebraic specifications using the operators of *BMA* and the hiding operator Δ of Section 3.3. See Sections 2.2 and 3.5 for the notation used.

$bool0 = \mathrm{T}(\mathbf{S}{:}B+\mathbf{F}{:}T{:}B+\mathbf{F}{:}F{:}B).$

$bool1 = bool0+{:}(\mathbf{F}{:}\neg{:}B\to B){:}(<\neg T=F>+<\neg F=T>).$

$bool2 = bool1+{:}(\mathbf{F}{:}\vee{:}B\times B\to B){:}(<T\vee x=T>+<F\vee x=x>+<x\vee y=y\vee x>).$

$bool3 = bool2+{:}(\mathbf{F}{:}\wedge{:}B\times B\to B){:}(<x\wedge y=\neg(\neg x\vee\neg y)>).$

$nat0 = \mathrm{T}(\mathbf{S}{:}N+\mathbf{F}{:}0{:}N+\mathbf{F}{:}S{:}N\to N).$

$nat1 = nat0+{:}(\mathbf{F}{:}+{:}N\times N\to N){:}(<x+0=x>+<x+S(y)=S(x+y)>).$

$nat2 = nat1+{:}(\mathbf{F}{:}\,.\,{:}N\times N\to N){:}(<x.0=0>+<x.S(y)=(x.y)+x>).$

$nat3 = nat2+{:}(\mathbf{F}{:}\exp{:}N\times N\to N){:}(<\exp(x,0)=S(0)>+<\exp(x,S(y))=x.\exp(x,y)>).$

$nateq0 = bool0+{:}nat0+{:}(\mathbf{F}{:}eq{:}N\times N\to B){:}$
$(<eq(0,0)=T>+<eq(S(x),0)=F>+<eq(0,S(x))=F>+<eq(S(x),S(y))=eq(x,y)>).$

$nateq1 = bool2+{:}nateq0.$

$setnateq = nateq1+{:}(\mathbf{S}{:}S+\mathbf{F}{:}\varnothing{:}S+\mathbf{F}{:}ins{:}N\times S\to S+\mathbf{F}{:}\in{:}N\times S\to B){:}$
$(<x\in\varnothing=F>+<x\in ins(y,z)=eq(x,y)\vee(x\in z)>+$
$<ins(x,ins(y,z))=ins(y,ins(x,z))>+<ins(x,ins(x,y))=ins(x,y)>).$

$setnat0 = (\Sigma(bool0)+\Sigma(nat0)+\mathbf{S}{:}S+\mathbf{F}{:}\varnothing{:}S+\mathbf{F}{:}ins{:}N\times S\to S+\mathbf{F}{:}\in{:}N\times S\to B)\Box setnateq.$

$nateq2 = nateq1+{:}(\mathbf{F}{:}+{:}B\times N\to N){:}(<F+x=x>+<T+x=S(x)>).$

$setnat1 = (\Sigma(setnat0)+\mathbf{F}{:}card{:}S\to N)\Box{:}$
$(nateq2+{:}setnat0+{:}(<card(\varnothing)=0>+<card(ins(x,y))=\neg(x\in y)+card(y)>)).$

$setnat2 = (\Sigma(nat0)+\mathbf{S}{:}S+\mathbf{F}{:}\varnothing{:}S+\mathbf{F}{:}ins{:}N\times S\to S+\mathbf{F}{:}card{:}S\to N)\Box setnat1.$

$setnat3 = (nateq0+setnat2)+{:}(\mathbf{F}{:}\in{:}N\times S\to B){:}(<x\in y=eq(card(y),card(ins(x,y)))>).$

$setnat4 = setnat1+setnat3.$

$binnat\,0 = \mathrm{T}(\mathbf{S}{:}BN+\mathbf{F}{:}0_b{:}BN+\mathbf{F}{:}1_b{:}BN+\mathbf{F}{:}A_0{:}BN\to BN+\mathbf{F}{:}A_1{:}BN\to BN).$
Note: $A_0(A_0(1_b))$ corresponds to 100, etc.

$natbinnat = (binnat\,0+nat\,0)+{:}(\mathbf{F}{:}i{:}BN\to N+\mathbf{F}{:}j{:}N\to BN){:}$
$((\mathbf{F}{:}S_b{:}BN\to BN)\Delta{:}$
$(nat\,1+{:}(<S_b(0_b)=1_b>+<S_b(1_b)=A_0(1_b)>+<S_b(A_0(x))=A_1(x)>+$
$<S_b(A_1(x))=A_0(S_b(x))>+<i(0_b)=0>+<i(S_b(x))=S(i(x))>+$
$<i(A_0(x))=i(x)+i(x)>+<j(0)=0_b>+<j(S(x))=S_b(j(x))>))).$

$binnat\,1 = (\Sigma(binnat\,0)+\mathbf{F}{:}add_b{:}BN\times BN\to BN+\mathbf{F}{:}mult_b{:}BN\times BN\to BN)\Box{:}$
$(natbinnat+{:}nat\,2+{:}(<add_b(x,y)=j(i(x)+i(y))>+<mult_b(x,y)=j(i(x).i(y))>)).$

$binnatset = (\Sigma(binnat\,0)+\mathbf{S}{:}S+\mathbf{F}{:}\varnothing_b{:}S+\mathbf{F}{:}ins_b{:}BN\times S\to S+\Sigma(bool\,0)+\mathbf{F}{:}\in_b{:}BN\times S\to B)\Box{:}$
$(setnat\,0+{:}natbinnat+{:}(<\varnothing_b=\varnothing>+<ins_b(x,y)=ins(i(x),y)>+<x\in_b y=i(x)\in y>)).$

$binnat\,2 = (\Sigma(binnat\,0)+\mathbf{F}{:}stand{:}BN\to BN)\Box{:}(natbinnat+{:}<stand(x)=j(i(x))>).$

$binnateqb = (\Sigma(binnat\,0)+\Sigma(bool\,0)+\mathbf{F}{:}eq_b{:}BN\times BN\to B)\Box{:}$
$(natbinnat+{:}nateq\,0+{:}(\mathbf{F}{:}lth{:}BN\to N){:}$
$(<lth(0_b)=S(0)>+<lth(1_b)=S(0)>+<lth(A_0(x))=S(lth(x))>+$
$<lth(A_1(x))=S(lth(x))>+<eq_b(x,y)=eq(lth(x),lth(y))\wedge eq(i(x),i(y))>)).$

$binnateq = (binnat\,0+bool\,0)+{:}(\mathbf{F}{:}eq{:}BN\times BN\to B)\Box{:}$
$(binnat\,2+{:}binnateqb+{:}<eq(x,y)=eq_b(stand(x),stand(y))>).$

$stackbool\,0 = (\mathbf{S}{:}ST+\mathbf{F}{:}\varnothing{:}ST+\mathbf{F}{:}push{:}B\times ST\to ST+\mathbf{F}{:}pop{:}ST\to ST+\mathbf{F}{:}top{:}B\times ST\to B){:}$
$(bool\,0+{:}(<pop(\varnothing)=\varnothing>+<pop(push(x,u))=u>+<top(x,\varnothing)=x>+$
$<top(x,push(y,u))=y>)).$

$stackboolbinnat = stackbool\,0+{:}binnat+{:}(\mathbf{F}{:}k{:}ST\to BN+\mathbf{F}{:}l{:}BN\to ST){:}$
$(<k(\varnothing)=0_b>+<k(push(T,x))=A_1(k(x))>+<k(push(F,u))=A_0(k(x))>+$
$<l(0_b)=\varnothing>+<l(1_b)=push(T,\varnothing)>+<l(A_0(x))=push(F,l(x))>+$
$<l(A_1(x))=push(T,l(x))>).$

$stackbool\,1 = (\Sigma(stackbool\,0)+\mathbf{F}{:}add_s{:}ST\times ST\to ST+\mathbf{F}{:}mult_s{:}ST\times ST\to ST)\Box{:}$
$(stackboolbinnat+{:}binnat\,1+{:}(<add_s(x,y)=l(add_b(k(x),k(y)))>+$
$<mult_s(x,y)=l(mult_b(k(x),k(y)))>)).$

ACKNOWLEDGMENTS

We would like to thank N.W.P. van Diepen, P.R.H. Hendriks, C.P.J. Koymans, and E. Nieuwland for their many helpful comments and suggestions.

REFERENCES

(References [E82], [GB84], [GM84], and [HOR85] are not cited in the text.)

[BG80] R.M. BURSTALL & J.A. GOGUEN, The semantics of CLEAR, a specification language, in: D. BJØRNER, ed., *Abstract Software Specifications*, LNCS, Vol. 86, Springer-Verlag, 1980, pp. 292-332.

[BHK85] J.A. BERGSTRA, J. HEERING & P. KLINT, Algebraic definition of a simple programming language, Report CS-R8504, Department of Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1985.

[BKN80] J.A. BERGSTRA, H.C.M. KLEIJN & P. NOUWT, On the algebraic specification of infinite data types using monoidal auxiliary functions, Report 80-43, Institute of Applied Mathematics and Computer Science, University of Leiden, 1980.

[BM81] J.A. BERGSTRA & J.-J. CH. MEYER, Small specifications for large finite data structures, *International Journal of Computer Mathematics*, **9** (1981), 4, pp. 305-320.

[BM82] J.A. BERGSTRA & J.-J. CH. MEYER, The equational specification of finite minimal unoids using unary hidden functions only, *Fundamenta Informaticae*, **V** (1982), 2, pp. 143-170.

[BM84] J.A. BERGSTRA & J.-J. CH. MEYER, On specifying sets of integers, *Elektronische Informationsverarbeitung und Kybernetik*, **20** (1984), 10/11, pp. 531-541.

[BT79] J.A. BERGSTRA & J.V. TUCKER, *Algebraic specifications of computable and semi-computable data structures*, Report IW 115/79, Department of Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1979; revised version to appear in *Theoretical Computer Science*.

[BT80] J.A. BERGSTRA & J.V. TUCKER, A characterisation of computable data types by means of a finite equational specification method, in: J.W. DE BAKKER & J. VAN LEEUWEN, eds, *Automata, Languages and Programming*, 7th Colloquium, LNCS, Vol. 85, Springer-Verlag, 1980, pp. 76-90.

[BT82] J.A. BERGSTRA & J.V. TUCKER, The completeness of the algebraic specification methods for computable data types, *Information and Control*, **54** (1982), 3, pp. 186-200.

[BT83] J.A. BERGSTRA & J.V. TUCKER, Initial and final algebra semantics for data type specifications: two characterization theorems, *SIAM Journal on Computing*, **12** (1983), 2, pp. 366-387.

[E82] H.-D. EHRICH, On the theory of specification, implementation, and parametrization of abstract data types, *Journal of the ACM*, **29** (1982), 1, pp. 206-227.

[EM85] H. EHRIG & B. MAHR, *Fundamentals of Algebraic Specifications*, Vol. I, *Equations and Initial Semantics*, Springer-Verlag, 1985.

[FGJM85] K. FUTATSUGI, J.A. GOGUEN, J.P. JOUANNNAUD & J. MESEGUER, Principles of OBJ2, *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, ACM, 1985, pp. 52-66.

[GAN83] H. GANZINGER, Increasing modularity and language-independency in automatically generated compilers, *Science of Computer Programming*, **3** (1983), pp. 223-278.

[GAU85] M.-C. GAUDEL, An introduction to PLUSS, to appear in *Proceedings of the Esprit Technical Week*, 1985.

[GB84] J.A. GOGUEN & R.M. BURSTALL, Introducing institutions, in: E. CLARKE & D. KOZEN, eds, *Logics of Programs*, LNCS, Vol. 164, Springer-Verlag, 1984, pp. 221-255.

[GM84] J.A. GOGUEN & J. MESEGUER, Equality, types, modules, and (why not?) generics for logic programming, *Journal of Logic Programming*, **2** (1984), pp. 179-210.

[HEE85] J. HEERING, Partial evaluation and ω-completeness of algebraic specifications, Report CS-R8501, Department of Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1985; to appear in *Theoretical Computer Science*, **43** (1986), pp. 1-19.

[HOR85] J.J. HORNING, Combining algebraic and predicative specifications in LARCH, in: *Formal Methods for Software Development*, TAPSOFT Proceedings, Vol. 2, LNCS, Vol. 186, Springer-Verlag, 1985, pp. 12-26.

[J86] T.M.V. JANSSEN *Foundations and Applications of Montague Grammar*, Part 1: *Philosophy, Framework, Computer Science*, Tract 19, Centre for Mathematics and Computer Science, Amsterdam, 1986.

[KAP83] S. KAPLAN, Un langage de spécification de types abstraits algébriques, Thèse de 3ème cycle, Université de Paris-Sud, 1983.

[KLA83] H.A. KLAEREN, *Algebraische Spezifikation*, Springer-Verlag, 1983.

[LIP83] U. LIPECK, Ein algebraischer Kalkül für einen strukturierten Entwurf von Datenabstraktionen, Dissertation, Forschungsbericht Nr. 148, Abteilung Informatik, Universität Dortmund, 1983.

[LOE85] J. LOECKX, A formal description of the specification language OBSCURE, Report A 85/15, Universität des Saarlandes, Saarbrücken, 1985.

[MAJ77] M.E. MAJSTER, Limits of the "algebraic" specification of abstract data types, *SIGPLAN Notices*, **12** (1977), 10, pp. 37-42.

[MG86] J. MESEGUER & J.A. GOGUEN, Initiality, induction, and computability, in: M. NIVAT & J. REYNOLDS, eds, *Algebraic methods in Semantics*, Cambridge University Press, 1986.

[MS85] T.S.E. MAIBAUM & M.R. SADLER, Axiomatising specification theory, in: H.-J. KREOWSKI, ed., *Recent Trends in Data Type Specification*, 3rd Workshop on Theory and Applications of Abstract Data Types, Informatik-Fachberichte 116, Springer-Verlag, 1985, pp. 171-177.

[MVS85] T.S.E. MAIBAUM, P.A.S. VELOSO & M.R. SADLER, A theory of abstract data types for program development: bridging the gap?, in: *Formal Methods for Software Development*, TAPSOFT Proceedings, Vol. 2, LNCS, Vol. 186, Springer-Verlag, 1985, pp. 214-230.

[P72] D.L.PARNAS, On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, **15** (1972), pp. 1053-1058.

[SHO67] J.R. SHOENFIELD, *Mathematical Logic*, Addison-Wesley, 1967.

[W83] M. WIRSING, Structured Algebraic Specifications: A Kernel Language, Thesis, Institut für Informatik, Technische Universität, München, 1983.