Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

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

Module algebra (revised version)

Computer Science/Department of Software Technology Report CS-R8844 November

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

(revised version)

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.

1987 CR Categories: D.2.1 [**Software Engineering**]: Requirements/Specifications - *Languages;* D.2.2 [**Software Engineering**]: Tools and Techniques - *Modules and Interfaces;* D.3.3 [**Programming Languages**]: Language Constructs - *Abstract Data Types, Modules;* F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages - *Algebraic Approaches to Semantics.*

1985 Mathematics Subject Classification: 68Q55 [**Theory of Computing**]: Semantics; 68Q65 [**Theory of Computing**]: Abstract Data Types.

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.

Note: This is a revised version of report CS-R8617 (May 1986). In particular section 3.1 (basic module algebra), section 4 (models of module algebra), and section 5 (algebraic specifications from the viewpoint of module algebra) were extensively revised and contain new material.

Report CS-R8844
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

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:

(i) *Information hiding/abstraction.* Modules generally contain *hidden* (*auxiliary, local, internal, invisible,* . . .) items without which it would be 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 in specifications of data types in programming languages, and the hidden sorts and functions needed in initial algebra specifications of data types [BT87].

(ii) *Compositionality of module operations.* Modules can be adapted and combined by means of various operations like renaming of sorts and functions and importing a module in another one. Each such operation should preferably be a simple, effectively computable operation on the textual representation 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 a reflection of the textual one, i.e., the semantics of module operations should be *compositional* [J86]. If these two requirements can be met, computations involving modules become both practicable and meaningful. In our case compositionality is guaranteed by the fact that we use *algebraic semantics* (cf. [J86, Chapter II]).

(iii) *Reusability of modules.* Some modules can be used as part of many programs or specifications. These are said to be *reusable.* Such modules resemble the constructs in programming or specification languages, which are also highly reusable. Reusability of modules 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 performing some editing action on the text of an existing one is also a very general form of reuse, of course, but this will not always correspond to an acceptable 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* (+), *intersection* ($\cap$), and *supersignature* ($\supseteq$).

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 $BMA[fol]$, where *fol* is many-sorted first-order logic with equality. The main operators of this algebra are *taking the visible signature* (Σ), *renaming* (.), *combination/union* (+), and *export* ($\Box$). We do not discuss parametrization (actualization) in this paper. In Section 3.2 we prove a normal form theorem for closed module expressions. In Section 3.3 we introduce *hiding* (Δ) and *common export.* The former is complementary to export. 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 first-order 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 $\mathbb{M}(fol) \not\cong \mathbb{M}_C(fol) \not\cong \mathbb{T}(fol)$.

The implications of our results for algebraic specifications (viewed as equational theories or initial algebras) are discussed in SECTION 5.1. In SECTIONS 5.2-3 the expressive power of many-sorted equational logic, many-sorted conditional equational logic, many-sorted first-order logic with equality, and many-sorted equational logic in the presence of Booleans are compared with each other. SECTION 5.4 gives an overview of related results in the field of algebraic specification.

A more informal discussion (in Dutch) of many topics discussed in this paper can be found in [B87].

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 [Gau86]. 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. Further developments in this direction can, for instance, be found in papers by BLUM, EHRIG & PARISI-PRESICCE [BEPP87] and PARISI-PRESICCE [PP87].

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 [MS85, MVS85], who used it to characterize the composability of implementations. We obtain two distributive laws for the export operator $\Box$, both of which, in the context of the first-order theory interpretation $\mathbb{T}(fol)$ of module expressions, are equivalent to the CRAIG interpolation lemma.

In [BHK85] we experimented with the algebraic specification formalism ASF which is similar to OBJ or PLUSS. Our motivation for the present work was both dissatisfaction with the import and export mechanisms of ASF 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 $\mathbb{M}(fol)$ was discussed earlier 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 Booleans 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. We only consider specifications in many-sorted (conditional) equational logic and in many-sorted first-order logic with equality (but no other predicates). Signatures of such specifications are sets of declarations of *sorts*, *typed constants*, and *typed functions*. The equality predicate is part of the logical language and as such does not occur in the signature of any specification.

FIGURE 2.1 shows a simple example of a signature *Sig* both in textual and graphical form. Because the constant symbol 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 ($n \geq 2$) are not uniquely determined by the graphical representation, but only up to an arbitrary permutation of the argument sorts.

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

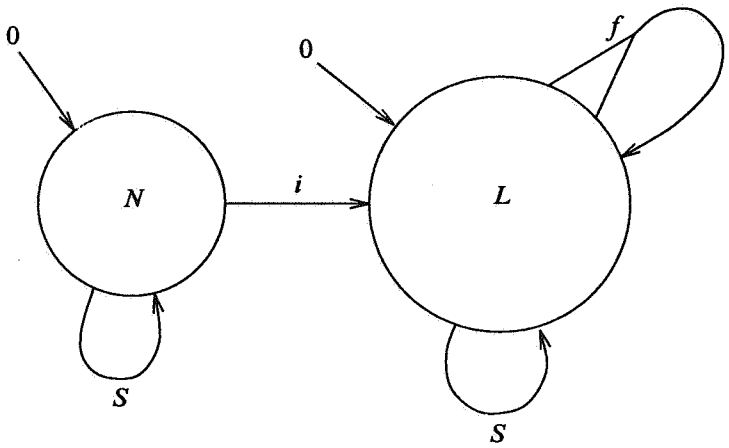

FIGURE 2.1. Example of a signature *Sig* - textual and (almost) equivalent graphical representation.

The ambiguity problems caused by overloading may be circumvented by attaching an explicit type to each non-logical symbol in a sentence. Let $\mathcal{T}(x)$ be the set of *correctly and explicitly typed* expressions (terms) which can be formed from the constant and function symbols declared in a signature x plus the first-order variable symbols declared in some separate variable declaration, and let $\mathcal{L}(x)$ be the set of correctly and explicitly typed first-order sentences over x. Some expressions belonging to $\mathcal{T}(Sig)$ (FIGURE 2.1) are

0^N

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

$S^{L \to L}(f^{L \times L \to L}(k^L, l^L))$,

where k and l are variables. Types are given by superscripts. 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 $\mathscr{L}(Sig)$-equations

$$S^{L\to L}(0^L) = 0^L$$
$$S^{L\to L}(i^{N\to L}(n^N)) = i^{N\to L}(S^{N\to N}(n^N))$$
$$S^{L\to L}(f^{L\times L\to L}(k^L,l^L)) = f^{L\times L\to L}(S^{L\to L}(k^L),S^{L\to L}(l^L)),$$

where k, l, n are variables, can in principle be abbreviated to

$$S(0^L) = 0$$
$$S(i(n)) = i(S(n))$$
$$S(f(k,l)) = f(S(k),S(l)),$$

because all types except that of 0 and S in the first equation 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. 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* (+), *intersection* ($\cap$), and *supersignature* ($\supseteq$).

Atomic signatures are constructed by means of the S-constructor (sort declaration) and the F-constructor (function declaration). Functions are typed, i.e., a non-empty sequence of (sort) names is attached to them. For reasons of brevity, names are natural numbers 0, $N(0)$, ... in the specification, but in the text we always use ordinary names for constants and functions. Functions whose type consists of a single name correspond to constants. Although their declaration is not forbidden, sorts that occur in the type of a constant or function need not be introduced explicitly. Sorts that do not occur in the type of any constant or function must be declared explicitly by means of the S-constructor, however. Signatures are constructed from atomic signatures by means of the +-operator. Because we allow overloaded constants and functions, unrestricted union of signatures is no problem.

Atomic renamings are constructed by means of *rs* (rename sort) and *rf* (rename function/constant). To avoid ambiguities due to overloading, the third argument of *rf* should contain the type of the name to be renamed. Atomic renamings can be applied to (atomic) signatures by means of the .-operator.

Renaming is *permutative*, i.e., if a is renamed to b, b is simultaneously renamed to a. Due to its permutative character, renaming never causes name clashes. Names that are different are never made equal by a renaming. Any injective renaming can be realized by an appropriate sequence of applications of atomic renamings.

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

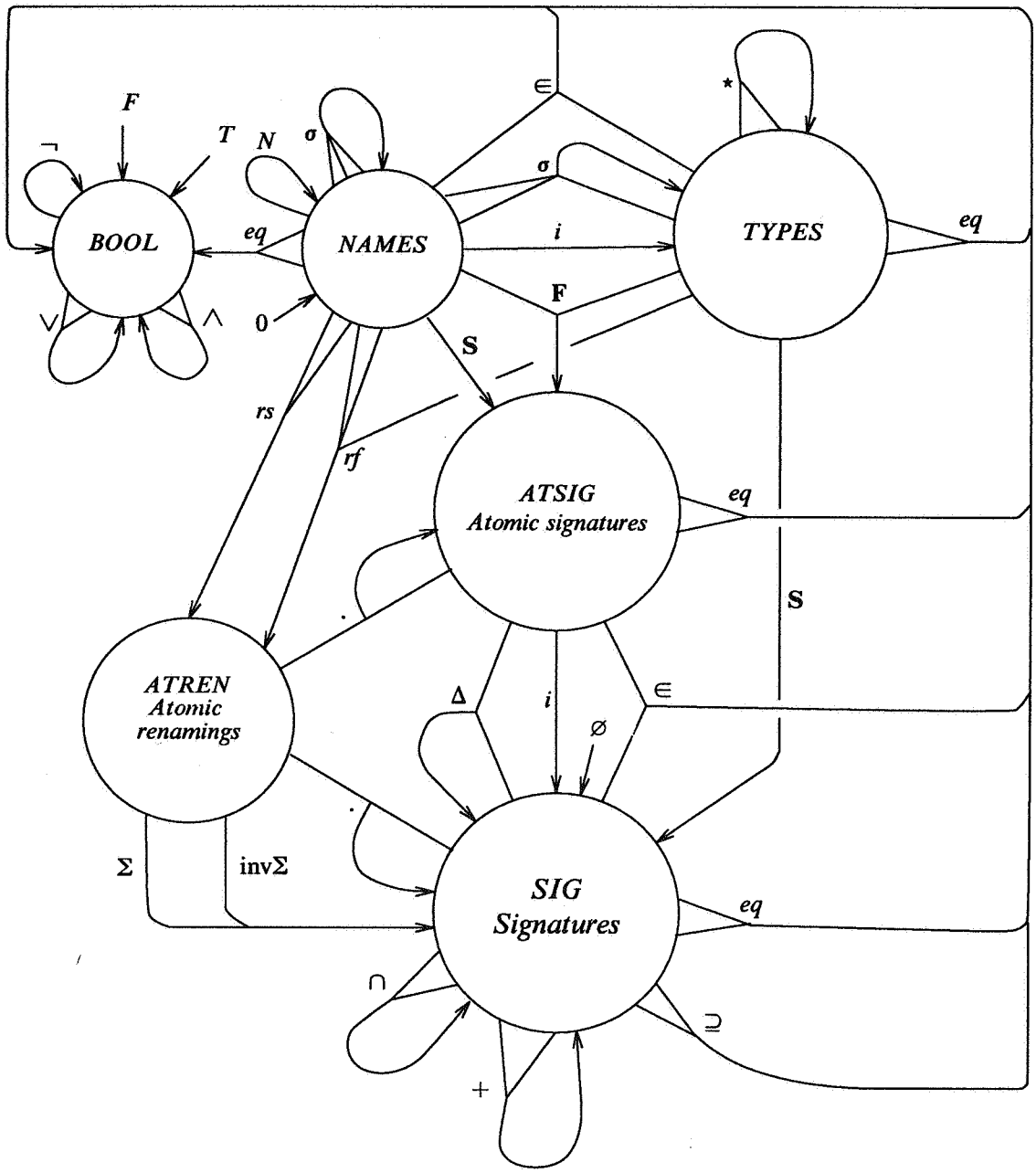

FIGURE 2.2. The signature of the algebra of signatures.

module *Booleans*
begin

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

end *Booleans*

module *Signatures*
begin

import *Booleans*
sort $NAMES$
functions
$0 : NAMES$
$N : NAMES \to NAMES$
$eq : NAMES \times NAMES \to BOOL$ (*Equality*)
$\sigma : NAMES \times NAMES \times NAMES \to NAMES$ (*Elementary renaming*)
variables $l,m,n : NAMES$
equations
$eq(0,0) = T$
$eq(0,N(l)) = F$

$eq(N(l),0) = F$
$eq(N(l),N(m)) = eq(l,m)$

$\sigma(l,m,l) = m$ (*Renaming is*
$\sigma(l,m,m) = l$ *permutative*)
$eq(l,n) = F \;\&\; eq(m,n) = F \Rightarrow \sigma(l,m,n) = n$

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

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

equations

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

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

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

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

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

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

equations

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

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

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

$l,m,n : NAMES$

$t,u : TYPES$

equations

$rs(l,l) = rs(m,m)$
$rs(m,m) = rf(l,l,t)$
$rf(l,l,t) = rf(m,m,u)$
} (*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 by 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*)

$\supseteq : SIG \times SIG \rightarrow BOOL$ (*Supersignature*)

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

variables

$l,m : NAMES$

t,u : *TYPES*
a : *ATSIG*
r : *ATREN*
x,y,z : *SIG*

equations

$x+\varnothing = x$
$x+x = x$
$x+y = y+x$
$(x+y)+z = x+(y+z)$
$i(\mathbf{F}(l,t)) = i(\mathbf{F}(l,t))+\mathbf{S}(t)$ *(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)*
$eq(l,m) = F \Rightarrow \Sigma(rs(l,m)) = i(\mathbf{S}(l))+i(\mathbf{S}(m))$
$eq(l,m) = F \Rightarrow \Sigma(rf(l,m,t)) = i(\mathbf{F}(l,t))+i(\mathbf{F}(m,t))$

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

$a\in\varnothing = F$
$\mathbf{S}(l)\in i(\mathbf{S}(m)) = eq(l,m)$
$\mathbf{S}(l)\in i(\mathbf{F}(m,t)) = l\in t$
$\mathbf{F}(l,t)\in i(\mathbf{F}(m,u)) = eq(l,m)\wedge eq(t,u)$
$\mathbf{F}(l,t)\in i(\mathbf{S}(m)) = F$
$a\in(x+y) = (a\in x)\vee(a\in y)$

$x\cap\varnothing = \varnothing$
$x\cap x = x$
$x\cap y = y\cap x$
$(x\cap y)\cap z = x\cap(y\cap z)$
$\mathbf{S}(l)\in x = F \Rightarrow i(\mathbf{S}(l))\cap x = \varnothing$
$\mathbf{F}(l,t)\in x = F \Rightarrow i(\mathbf{F}(l,t))\cap x = \mathbf{S}(t)\cap x$
$a\in x = T \Rightarrow i(a)\cap x = i(a)$
$(x+y)\cap z = (x\cap z)+(y\cap z)$

$a\in x = F \Rightarrow a\Delta x = x$
$a\Delta i(a) = \varnothing$
$l\in t = T \Rightarrow \mathbf{S}(l)\Delta i(\mathbf{F}(m,t)) = \mathbf{S}(l)\Delta\mathbf{S}(t)$
$a\Delta(x+y) = (a\Delta x)+(a\Delta y)$

$$x = y+z \Rightarrow x \supseteq y = T$$
$$a \in y = T \;\&\; a \in x = F \Rightarrow x \supseteq y = F$$
$$eq(x,y) = (x \supseteq y) \wedge (y \supseteq x)$$

end *Signatures*

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

The initial model of *Signatures* is a computable algebra [BT82]. 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 occurring in the type of any constant or function are declared explicitly. (Due to the equation

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

sorts that occur in the type of a constant or function need not be introduced explicitly by means of the S-constructor.) 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, 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*.

We call an equation *ω-derivable* if all of its closed instances are equationally derivable. An equation is ω-derivable if and only if it holds in the initial model (cf. [Hee86]). Some equations that are ω-derivable from *Signatures* are:

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

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

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

For instance,

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

becomes

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

3. Basic module algebra

3.1. *BMA[fol]*

In this section we concentrate on many-sorted first-order logic with equality (*fol*). The only predicates are **true**, **false**, and the equality predicate $=$. These are part of the logical language and as such do not occur in the signature of any *fol*-sentence.

Module expressions are modular *fol*-specifications. They basically consist 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 $\square$ (restriction of the visible signature of a module).

Each first-order sentence ϕ corresponds to a module constant $<\phi>$ whose associated signature $\Sigma(<\phi>) = \Sigma(\phi)$ is the smallest signature x such that $\phi \in \mathscr{L}(x)$. Remember that ϕ is explicitly typed (Section 2.1) so that x is uniquely determined by ϕ. We assume free variables in first-order sentences to be universally quantified. 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 constant. 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 set of *flat* module expressions consists of expressions involving only the constants $<\phi>$ and the operators $+$, . and T. These represent ordinary finite first-order theories. From the viewpoint of first-order logic, T(x) is equivalent to $<\phi>$ with ϕ a tautology and $\Sigma(<\phi>)=x$.

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

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

which is to be read as "export x from $<\phi_1> + <\phi_2>$". The intended meaning of this module expression is a module 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 modular *fol*-specifications (*BMA*[*fol*]) are given in FIGURE 3.2. A graphical representation of the corresponding signature is shown in FIGURE 3.1. While designing *BMA*[*fol*] we kept the following requirements in mind:

(A) All axioms 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 *fol*-specifications. In $\mathbb{M}(fol)$ the $+$-operator is interpreted as generalized intersection of model classes (*not* union of model classes!) and the export operator $\square$ is interpreted as restriction of the signature of the models in a class. $\mathbb{M}(fol)$ is discussed in more detail in SECTION 4.

(B) Persistency of signatures: as an extension of *Signatures* (SECTION 2.2) *BMA*[*fol*] would have to leave *Signatures* unaffected 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.

(C) Every closed module expression (closed term of sort *M*) would have to be provably equal to a normal form containing at most a single instance of the export operator $\square$. Normalization of module expressions is a basic operation which will have to be implemented in any system for manipulating specifications. In SECTION 3.2 we show that *BMA*[*fol*] satisfies this requirement.

(D) Let *X* and *Y* be closed module expressions. We call *Y* an *extension* of *X* if it says more than *X*, and we call it an *enrichment* of *X* if it says more than *X* but only about *new* signature elements. The axioms of *BMA*[*fol*] must guarantee that enrichment is a special case of extension. This is discussed in SECTION 3.4.

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 $\square$ (see (A) above), it turns out that:

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

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

(iii) 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.

Models of *BMA*[*fol*] (like $\mathbb{M}(fol)$) are *module algebras*. A *module* is an element of the carrier *M* of a module algebra. A module expression is a term of sort *M* over the signature of *BMA*[*fol*]. As such it is a textual representation (presentation) of a module.

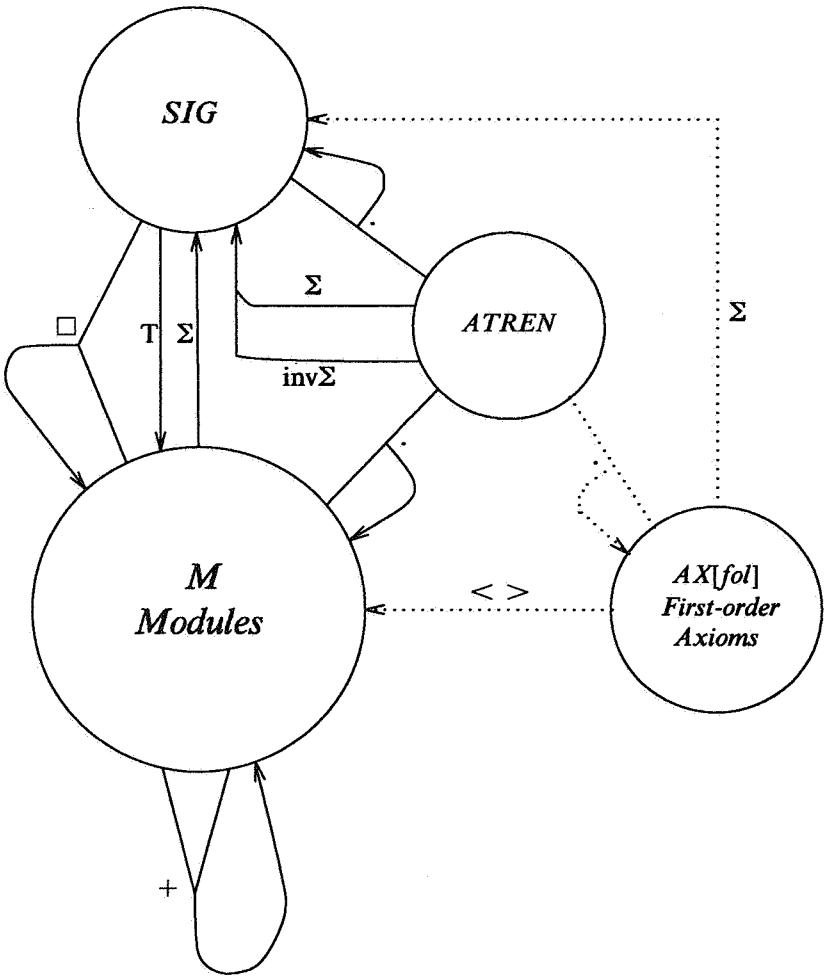

FIGURE 3.1. The signature of *BMA*[*fol*].
(The signature is only partially shown.
It is an extension of the signature of
Signatures shown in FIGURE 2.2.)

---

module *BMA*[*fol*]
begin

import *Signatures*

sort *M* (*Modules*)

constants

$<\phi> : M$ — *(For each first-order sentence $\phi \in \mathcal{L}(x)$ with signature x, $<\phi>$ is a constant of sort M; free variables in ϕ are assumed to be universally quantified)*

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

$r : ATREN$

$x, y : SIG$

$X, Y, Z : M$

equations

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

$x \supseteq (\Sigma(Y) \cap \Sigma(Z)) = T \Rightarrow$
$x \Box (Y+Z) = (x \Box Y)+(x \Box Z)$ (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) postulates unrestricted distribution of renaming over export. The permutative character of renaming is crucial here (see SECTION 2.2). Consider, for instance, the closed module expression

$$X = (\mathbf{S}{:}A + \mathbf{F}{:}a{:}A)\Box\langle a^A \neq b^A\rangle.$$

Whereas straightforward non-permutative renaming of a to b cannot be allowed as it leads to the inconsistent result

$$(\mathbf{S}{:}A + \mathbf{F}{:}b{:}A)\Box\langle b^A \neq b^A\rangle,$$

permutative renaming of a to b by means of (R4) does not cause a name clash:

$$rf(a,b,A).X \overset{(R4)}{=} (rf(a,b,A).(\mathbf{S}{:}A + \mathbf{F}{:}a{:}A))\Box(rf(a,b,A).\langle a^A \neq b^A\rangle) =$$
$$(\mathbf{S}{:}A + \mathbf{F}{:}b{:}A)\Box\langle b^A \neq a^A\rangle.$$

(R5) says that, due to its permutative character, renaming is an *involution*.

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

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

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*]:

$$\mathrm{inv}\Sigma(r) \supseteq (\Sigma(r) \cap \Sigma(X)) = T \Rightarrow r.X = X. \qquad \text{(R6')}$$

(C1) and (C2) together with the idempotent law $X + X = X$ 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 (requirement (D) — see SECTION 3.4).

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

(E3) says that hidden parts of the signature that are not used in any axiom may be deleted.

(E4) postulates restricted distribution of $\Box$ over $+$. Of course, it would be nice to have unrestricted distributivity, but this is simply not true in the models of *BMA*[*fol*] we have in mind. Consider the following simple counterexample:

$$x = \mathbf{S}{:}B + \mathbf{F}{:}T{:}B + \mathbf{F}{:}F{:}B$$
$$Y = \mathrm{T}(x + \mathbf{F}{:}c{:}B) + \langle T^B = c^B\rangle$$
$$Z = \mathrm{T}(x + \mathbf{F}{:}c{:}B) + \langle F^B = c^B\rangle.$$

Note that c^B is not exported by $x\Box Y$ and $x\Box Z$ due to the fact that $\mathbf{F}{:}c{:}B \notin x$. Now, on the one hand $x\Box(Y+Z)$ implies $T^B = F^B$ by way of $T^B = c^B = F^B$ and $\Sigma(T^B = F^B) \subseteq x$. On the other hand $(x\Box Y)+(x\Box Z)$ does not imply $T^B = F^B$, as one may choose $c^B = T^B$ in $x\Box Y$ and $c^B = F^B$ in $x\Box Z$. Hence $x\Box(Y+Z) \neq (x\Box Y)+(x\Box Z)$.

The following equations are equationally derivable from *BMA*[*fol*] and hence valid in all its models:

(1) $X+\mathrm{T}(\varnothing) = X$
(2) $x\Box\mathrm{T}(\varnothing) = \mathrm{T}(\varnothing)$
(3) $x\Box\mathrm{T}(y) = \mathrm{T}(x\cap y)$
(4) $x\Box(\mathrm{T}(y)+Z) = (x\Box\mathrm{T}(y))+(x\Box Z)$.

PROOF. (1) $X+\mathrm{T}(\varnothing) \overset{(C4)}{=} (X+\mathrm{T}(\Sigma(X)))+\mathrm{T}(\varnothing) \overset{(C2)}{=} X+(\mathrm{T}(\Sigma(X))+\mathrm{T}(\varnothing)) \overset{(C3)}{=} X+\mathrm{T}(\Sigma(X)+\varnothing) = X+\mathrm{T}(\Sigma(X)) \overset{(C4)}{=} X$.

(2) $x\Box\mathrm{T}(\varnothing) \overset{(E1)}{=} x\Box(\Sigma(\mathrm{T}(\varnothing))\Box\mathrm{T}(\varnothing)) \overset{(S2)}{=} x\Box(\varnothing\Box\mathrm{T}(\varnothing)) \overset{(E2)}{=} (x\cap\varnothing)\Box\mathrm{T}(\varnothing) = \varnothing\Box\mathrm{T}(\varnothing) = \mathrm{T}(\varnothing)$.

(3) $x\Box\mathrm{T}(y) = x\Box\mathrm{T}(y+\varnothing) \overset{(C3)}{=} x\Box(\mathrm{T}(y)+\mathrm{T}(\varnothing)) \overset{(E3)}{=} \mathrm{T}(x\cap y)+(x\Box\mathrm{T}(\varnothing)) \overset{(2)}{=} \mathrm{T}(x\cap y)+\mathrm{T}(\varnothing) \overset{(C3)}{=} \mathrm{T}((x\cap y)+\varnothing) = \mathrm{T}(x\cap y)$.

(4) $x\Box(\mathrm{T}(y)+Z) \overset{(E3)}{=} \mathrm{T}(x\cap y)+(x\Box Z) \overset{(3)}{=} (x\Box\mathrm{T}(y))+(x\Box Z)$. ■

Conversely, (E3) follows immediately from equations (3) and (4).

As we explained in SECTION 2.2, we are not interested in models containing non-standard signatures. Hence, when proving equations over M we may use equational deduction plus equations over *SIG* like $x+(y\cap x) = x$ which are valid in the initial algebra of *Signatures* (ω-derivable from *Signatures*) but not equationally derivable from *Signatures*. The following equations are valid in all models of *BMA*[*fol*] that do not contain non-standard signatures:

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

PROOF. (5) $(\Sigma(X)+y)\Box X \overset{(E1)}{=} (\Sigma(X)+y)\Box(\Sigma(X)\Box X) \overset{(E2)}{=} ((\Sigma(X)+y)\cap\Sigma(X))\Box X = (\Sigma(X)+(y\cap\Sigma(X)))\Box X =$ (with $x+(y\cap x) = x$) $\Sigma(X)\Box X = X$.

(6) $\Sigma(X)\Box(\mathrm{T}(y)+X) \overset{(E3)}{=} \mathrm{T}(\Sigma(X)\cap y)+(\Sigma(X)\Box X) \overset{(E1)}{=} \mathrm{T}(\Sigma(X)\cap y)+X \overset{(C4)}{=} \mathrm{T}(\Sigma(X)\cap y)+\mathrm{T}(\Sigma(X))+X \overset{(C3)}{=} \mathrm{T}((\Sigma(X)\cap y)+\Sigma(X))+X =$ (with $x+(y\cap x) = x$) $\mathrm{T}(\Sigma(X))+X \overset{(C4)}{=} X$.

(7) From $x+(x\cap y) = x$ follows that $\Sigma(X)\supseteq(\Sigma(X)\cap\Sigma(Y)) = T$. Hence, (7) is a special case of (E4).

(8) $\Sigma(X)\Box(X+Y) \overset{(7)}{=} X+(\Sigma(X)\Box Y) \overset{(E1)}{=} X+(\Sigma(X)\Box(\Sigma(Y)\Box Y)) \overset{(E2)}{=} X+((\Sigma(X)\cap\Sigma(Y))\Box Y) =$ (with the first part of the condition) $X+(\varnothing\Box Y) =$ (with the second part of the condition) $X+\mathrm{T}(\varnothing) \overset{(1)}{=} X$. ■

3.2. The normal form theorem

In this section we show that $BMA[fol]$ satisfies requirement (C) of the previous section, i.e., that every closed module expression is provably equal to a normal form containing at most a single instance of the export operator $\Box$. This means that, although using multiple levels of export in a module expression may be advantageous from the viewpoint of modularization, it is never essential as far as expressive power is concerned. The meaning of hiding is independent of the "depth" at which it occurs. The proof of the normal form result hinges on the conditional distributive law (E4).

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 $CME[fol] \subseteq ME[fol]$ will be the set of closed module expressions.

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

The set 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

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

such that $BMA[fol] \vdash X = X'$, where $\vdash$ means conditional equational provability.

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 $y \Box Z$ with y a signature and Z 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 a $Y \in W$ 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 as

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

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

LEMMA 3.2.1. Let x, x' be signatures and $Y \in FCME[fol]$. Then there is a $Y' \in FCME[fol]$ such that $BMA[fol] \vdash x \Box Y = x \Box Y' = (x + x') \Box Y'$.

PROOF. Transform $x \Box Y$ into $x \Box Y'$ by repeatedly applying (R6) in such a way that all names occurring in $\Sigma(Y)$ but not in x are replaced by names not occurring in $x + x'$. We then have

$$x \Box Y = x \Box Y' \overset{(E1)}{=} x \Box (\Sigma(Y') \Box Y') \overset{(E2)}{=} (x \cap \Sigma(Y')) \Box Y' =$$
$$((x + x') \cap \Sigma(Y')) \Box Y' = (x + x') \Box Y'. \blacksquare$$

PROOF OF THE NORMAL FORM THEOREM. Let the $\Box$-depth d of a closed module expression be defined inductively as follows:

$$d(X) = 0 \text{ if } X \in FCME[fol]$$
$$d(r.X) = d(X)$$
$$d(X + Y) = \max(d(X), d(Y))$$
$$d(x \Box Y) = d(Y) + 1.$$

We use induction with respect to the $\Box$-depth: if $d(X) = 0$, X is flat and X can simply be brought into the desired normal form by applying (E1):

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

Now assume that for some $n \geqslant 0$ all $X \in CME[fol]$ with $d(X) \leqslant n$ can be brought into the desired normal form and consider an $X \in CME[fol]$ with $d(X) = n+1$. Without loss of generality we may take X of the form

$$\sum_{i=1}^{k}(u_i \Box X_i) \qquad (k \geqslant 1)$$

with $d(X_i) \leqslant n$. (Flat summands can be brought into the form $u_i \Box X_i$ by means of (E1), and renamings encompassing any outermost $\Box$-operators can be moved inward by means of (R3) and (R4) without changing the $\Box$-depth.) By the induction hypothesis we may normalize X_i to $v_i \Box Y_i$ with $Y_i \in FCME[fol]$ $(1 \leqslant i \leqslant k)$, and we obtain

$$X = \sum_{i=1}^{k}(u_i \Box (v_i \Box Y_i)) \overset{(E2)}{=} \sum_{i=1}^{k}((u_i \cap v_i) \Box Y_i).$$

If $k = 1$ this is the desired normal form and we are finished. Assume $k \geqslant 2$ and let

$$y = \sum_{i=1}^{k}(u_i \cap v_i).$$

Using LEMMA 3.2.1 we can find for each i a $Y_i' \in FCME[fol]$ such that

$$(u_i \cap v_i) \Box Y_i = y \Box Y_i'.$$

Hence

$$X = \sum_{i=1}^{k}(y \Box Y_i').$$

If each term of the form $(y \Box Z_1) + (y \Box Z_2)$ can be written as $y \Box Z_3$ (with Z_1, Z_2, Z_3 flat), the desired normal form can be obtained in $k-1$ steps. So consider

$$Z = (y \Box Z_1) + (y \Box Z_2).$$

In general the condition

$$y \supseteq \Sigma(Z_1) \cap \Sigma(Z_2)$$

is not satisfied, so (E4) cannot be applied directly, but using LEMMA 3.2.1 we can transform Z_2 into $Z_2' \in FCME[fol]$ such that

$$y \Box Z_2 = y \Box Z_2'$$

and

$$y \Box Z_2' = (y + \Sigma(Z_1)) \Box Z_2'.$$

Taking the signature of both sides of the latter equation gives

$$y \cap \Sigma(Z_2') = (y + \Sigma(Z_1)) \cap \Sigma(Z_2') \supseteq \Sigma(Z_1) \cap \Sigma(Z_2'),$$

so

$$y \supseteq \Sigma(Z_1) \cap \Sigma(Z_2').$$

Now, if Z_2 is replaced by Z_2', (E4) can be applied:

$$Z = (y \Box Z_1) + (y \Box Z_2) = (y \Box Z_1) + (y \Box Z_2') \overset{(E4)}{=} y \Box (Z_1 + Z_2') = y \Box Z_3. \blacksquare$$

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

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

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

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

The Δ- and $\Box$-operators occurring in the right-hand side of (H) and (CE) are the deletion operator $\Delta: ATSIG \times SIG \to SIG$ and the export operator $\Box : SIG \times M \to M$ respectively (see FIGURE 3.3).

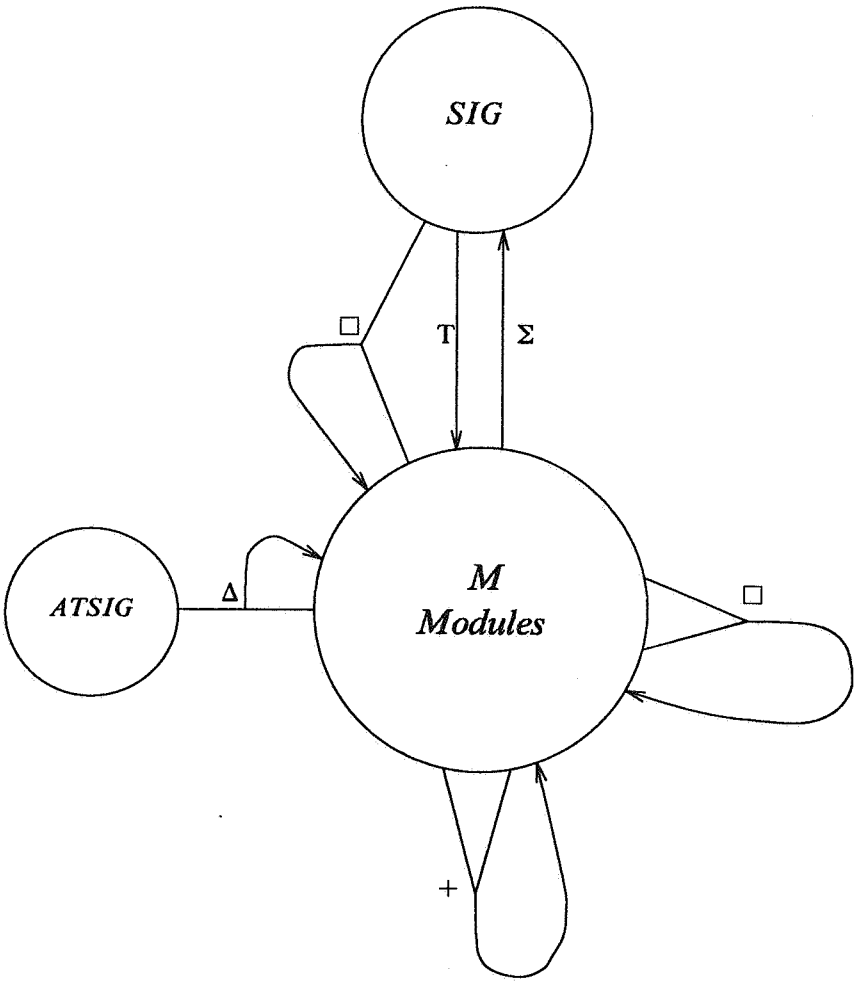

FIGURE 3.3. Extended signature for *BMA*[*fol*].
Note the overloading of $\Box$.

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, and common

export is a generalization of export in the sense that

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

PROOF. $\mathrm{T}(x)\Box Y \overset{(CE)}{=} (\Sigma(\mathrm{T}(x)) \cap \Sigma(Y)) \Box (\mathrm{T}(x)+Y) = (x \cap \Sigma(Y)) \Box (\mathrm{T}(x)+Y) \overset{(E3)}{=}$
$\mathrm{T}(x \cap \Sigma(Y)) + ((x \cap \Sigma(Y)) \Box Y) \overset{(C4)}{=} (x \cap \Sigma(Y)) \Box Y = x \Box Y.$ ■

As KOYMANS pointed out to us [Koy86], (E4) can be replaced by a remarkably symmetrical non-conditional equation if the common export operator is used in addition to the export operator:

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

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

$$(\Sigma(Y) \Box X) + (\Sigma(X) \Box Y) = (\Sigma(Y) \Box (\Sigma(X) \Box X)) + (\Sigma(X) \Box (\Sigma(Y) \Box Y)) =$$
$$((\Sigma(X) \cap \Sigma(Y)) \Box X) + ((\Sigma(X) \cap \Sigma(Y)) \Box Y) \overset{(E4)}{=} (\Sigma(X) \cap \Sigma(Y)) \Box (X+Y) \overset{(CE)}{=} X \Box Y.$$

Secondly, we show that $BMA[fol]-(\mathrm{E4})+(\mathrm{CE})+(\mathrm{E4}^*) \vdash (\mathrm{E4})$. Consider $x \Box (Y+Z)$ with $x \supseteq \Sigma(Y) \cap \Sigma(Z)$. Without loss of generality we may assume that

$$x \subseteq \Sigma(Y) + \Sigma(Z).$$

(Let $x' = x \cap (\Sigma(Y)+\Sigma(Z))$, then

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

We prove

$$x \Box (Y+Z) = (x \Box Y) + (x \Box Z)$$

by taking $Y' = \mathrm{T}(x)+Y$ and $Z' = \mathrm{T}(x)+Z$ and showing that

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

First observe that

$$\Sigma(Y') \cap \Sigma(Z') = (x+\Sigma(Y)) \cap (x+\Sigma(Z)) =$$
$$(x \cap x) + (x \cap \Sigma(Y)) + (x \cap \Sigma(Z)) + (\Sigma(Y) \cap \Sigma(Z)) = x \text{ (with } x \supseteq \Sigma(Y) \cap \Sigma(Z)),$$

and

$$Y'+Z' = \mathrm{T}(x)+Y+\mathrm{T}(x)+Z = \mathrm{T}(x)+Y+Z \overset{(C4)}{=}$$
$$\mathrm{T}(x)+\mathrm{T}(\Sigma(Y+Z))+Y+Z = \mathrm{T}(x+\Sigma(Y)+\Sigma(Z))+Y+Z =$$
$$\text{(with } x \subseteq \Sigma(Y)+\Sigma(Z)) \ \mathrm{T}(\Sigma(Y)+\Sigma(Z))+Y+Z = Y+Z.$$

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

$$Y' \Box Z' \overset{(CE)}{=} (\Sigma(Y') \cap \Sigma(Z')) \Box (Y'+Z') = x \Box (Y+Z).$$

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

$$Y' \Box Z' \overset{(E4^*)}{=} (\Sigma(Z') \Box Y') + (\Sigma(Y') \Box Z') = ((\Sigma(Y') \cap \Sigma(Z')) \Box Y') + ((\Sigma(Y') \cap \Sigma(Z')) \Box Z') =$$
$$(x \Box Y') + (x \Box Z') = (x \Box (\mathrm{T}(x)+Y)) + (x \Box (\mathrm{T}(x)+Z)) \overset{(E3)}{=}$$

$$T(x \cap x)+(x \square Y)+T(x \cap x)+(x \square Z) = T(x)+(x \square Y)+(x \square Z) =$$
(with $x \subseteq \Sigma(Y)+\Sigma(Z)$) $T(x \cap (\Sigma(Y)+\Sigma(Z)))+(x \square Y)+(x \square Z) =$
$T((x \cap \Sigma(Y))+(x \cap \Sigma(Z)))+(x \square Y)+(x \square Z) = T(\Sigma((x \square Y)+(x \square Z)))+(x \square Y)+(x \square Z) =$
$(x \square Y)+(x \square Z)$. ■

REMARK. By eliminating the common export operator by means of (CE) and putting $z = \Sigma(X) \cap \Sigma(Y)$, (E4*) is easily seen to be equivalent to

$$z = \Sigma(X) \cap \Sigma(Y) \Rightarrow z \square (X+Y) = (z \square X)+(z \square Y), \qquad (E4^-)$$

which is a special case of (E4). So the above result can be stated somewhat differently by saying that (E4) can be replaced by the slightly weaker axiom (E4$^-$) in *BMA*[*fol*].

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 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, i.e., $X = \Sigma(X) \square Y$;

(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

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

Hence, *BMA*[*fol*] satisfies requirement (D) of SECTION 3.1. Refinement is by definition a special case of extension.

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

LEMMA 3.4.1. (*Factorization 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*) $Y=\Sigma(X)\square Z=\Sigma(X)\square(Z+X)=$ (cf. equation (7) of SECTION 3.1) $(\Sigma(X)\square Z)+X=Y+X$.

(*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. Whether an extension is an enrichment or not depends on the semantics used. As we have not yet discussed the semantics of *BMA* we postpone further discussion of this point to SECTION 4.2.

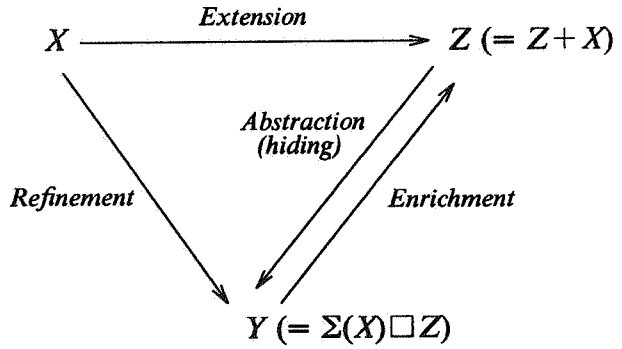

FIGURE 3.4. Graphical summary of DEFINITION 3.4.1 and the factorization lemma.

3.5. Notational conventions

From now on we will use the :-operator, which will allow us to drop the explicit typing from axioms in most cases (see also SECTION 2.1). Let a signature be called *unambiguous* if each of its function symbols is declared at most once as a function symbol of arity n ($n \geqslant 0$). Thus, the signature

$$\mathbf{F}:f:M\to M+\mathbf{F}:f:N\times N\to N$$

is unambiguous (f is declared once with arity 1 and once with arity 2), but

$$\mathbf{F}:f:M\to M+\mathbf{F}:f:N\to N$$

is ambiguous (f is declared twice with arity 1). Now $x:Y$ with unambiguous signature x 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.

We abbreviate axioms in module expressions still further by omitting universal quantifiers and variable declarations. The type of variables must be inferred from the context in which they occur. For instance,

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

and

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

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

Although we had the full model class interpretation $\mathbb{M}(fol)$ in mind while designing the axioms of *BMA* (requirement (A) of Section 3.1), the resulting system turns out to have other interesting and important interpretations. It should be emphasized that the normal form theorem (Theorem 3.2.2) can be applied independently of the particular interpretation chosen. There is no need to worry about semantics when calculating a normal form. Note, however, that everything in this section applies only to *fol*-specifications. Algebraic specifications (viewed as equational theories or initial algebras) are treated separately in Section 5.

4.1. Definitions

We first introduce some notation for classes of algebras and logically closed theories and then define suitable operators on them.

$\mathscr{L}(x)$ = the set of first-order sentences over signature x.
(Free variables in sentences are assumed to be universally quantified.
$\mathscr{L}(\varnothing)$ contains the 0-ary connectives **true** and **false**.)

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

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

$Alg_C(x)$ = the class of all countable x-algebras.

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

We only consider algebras with non-empty carriers.

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

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

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

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

$\Sigma(A)$ = x for $A \in Alg(x)$.
$x \square A$ = the restriction of A to $x \cap x'$ for $A \in Alg(x')$. (If $x \cap x' = \emptyset$, then $x \square A = A_{\emptyset}$.)
$x \square K$ = $\{x \square A \mid A \in K\}$ for $K \subseteq Alg(x')$.
$K + L$ = $\{A \in Alg(x_1 + x_2) \mid x_1 \square A \in K,\ x_2 \square A \in L\}$ for $K \subseteq Alg(x_1)$, $L \subseteq Alg(x_2)$. ($K + L = K \cap L$ if $x_1 = x_2$.)
$r.A$ = A renamed via r. For $A \in Alg(x)$ this yields an $A' \in Alg(r.x)$.
$r.K$ = $\{r.A \mid A \in K\}$. For $K \subseteq Alg(x)$ this yields a $K' \subseteq Alg(r.x)$.

$x \square T$ = $\mathscr{L}(x) \cap T$ for $T \in LCT(x')$.
$T + U$ = $\{\phi \in \mathscr{L}(x_1 + x_2) \mid T \cup U \vdash \phi\}$ for $T \in LCT(x_1)$, $U \in LCT(x_2)$.
$r.T$ = T renamed via r. For $T \in LCT(x)$ this yields a $T' \in LCT(r.x)$.

Using the above definitions, we further define three semantical mappings *Mod*, Mod_C, and *Th* on *CME*[*fol*] as follows:

$$X \mapsto Mod(X) \subseteq Alg(\Sigma(X))$$
$$X \mapsto Mod_C(X) \subseteq Alg_C(\Sigma(X))$$
$$X \mapsto Th(X) \in LCT(\Sigma(X)).$$

The precise inductive definitions are as follows:

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

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

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

An equivalence relation can now be associated with each of these three mappings in the following straightforward manner (with $X, Y \in CME[fol]$):

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

4.2. Four models of BMA[*fol*]

BMA[*fol*] is an algebraic specification and as such has an initial model $\mathbb{I}(BMA[fol])$. It is obtained by factorizing the free term algebra *CME*[*fol*], which consists of the textual representations (presentations) of modular first-order specifications, with respect to the congruence

$$X \equiv Y \Leftrightarrow BMA[fol] \vdash X = Y,$$

where $\vdash$ means conditional equational provability. Hence,

$$\mathbb{I}(BMA[fol]) = CME[fol] / \equiv.$$

Although rather weak, this congruence is 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 can 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 $\equiv_{Th}$ 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 $\mathbb{T}(fol) \models$ (E3) and $\mathbb{T}(fol) \models$ (E4), both of which turn out to be equivalent to the CRAIG interpolation lemma. This lemma states that two *fol*-sentences p and q with

$$\vdash p \Rightarrow q$$

always have an *interpolant*, i.e., a *fol*-sentence r with signature

$$\Sigma(r) \subseteq \Sigma(p) \cap \Sigma(q)$$

such that

$$\vdash p \Rightarrow r$$

and

$$\vdash r \Rightarrow q.$$

By using the deduction theorem for first-order logic, the following equivalent formulation of the interpolation lemma is obtained:
if

$$p \vdash q$$

there always is an interpolant r with signature

$$\Sigma(r) \subseteq \Sigma(p) \cap \Sigma(q)$$

such that

$$p \vdash r \vdash q.$$

See also [C57, Lemma 3], [BJ80, Chapter 23], or [S67, §5.4].

THEOREM 4.2.1. $\mathbb{T}(fol) \models$ (E3).

PROOF. We show that

$$x \Box (\mathrm{T}(y) + Z) \equiv_{Th} \mathrm{T}(x \cap y) + (x \Box Z).$$

(*a*) $\Sigma(x \Box (\mathrm{T}(y) + Z)) = x \cap (\Sigma(\mathrm{T}(y)) + \Sigma(Z)) = (x \cap y) + (x \cap \Sigma(Z)) = \Sigma(\mathrm{T}(x \cap y) + (x \Box Z))$.

(*b*) $Th(x \Box (\mathrm{T}(y) + Z)) \supseteq Th(\mathrm{T}(x \cap y) + (x \Box Z))$:
Let $p \in Th(\mathrm{T}(x \cap y) + (x \Box Z))$. Choose $q \subset Th(x \Box Z)$ with $q \vdash p$. Clearly, $q \in Th(x \Box (\mathrm{T}(y) + Z))$, hence $p \in Th(x \Box (\mathrm{T}(y) + Z))$.

(*c*) $Th(x\Box(\mathrm{T}(y)+Z))\subseteq Th(\mathrm{T}(x\cap y)+(x\Box Z))$:
Let $p\in x\Box(\mathrm{T}(y)+Z)$. Choose $q\in Th(Z)$ with $q\vdash p$. According to the interpolation lemma there is an interpolant r with $\Sigma(r)\subseteq\Sigma(q)\cap\Sigma(p)\subseteq\Sigma(Z)\cap x\cap(y+\Sigma(Z))=x\cap\Sigma(Z)$ such that $q\vdash r$ and $r\vdash p$. Hence, $r\in Th(x\Box Z)$ and $p\in Th(\mathrm{T}(x\cap y)+(x\Box Z))$. ■

REMARK. Conversely, (E3) implies the CRAIG interpolation lemma. Suppose $p\vdash q$ and let $x=\Sigma(q)$. From the fact that $q\in Th(x\Box(T(x)+<p>))=Th(T(x)+(x\Box<p>))$ follows that there is an $r\in Th(x\Box<p>)$ such that $r\vdash q$. Hence $p\vdash r\vdash q$ and $\Sigma(r)\subseteq\Sigma(q)\cap\Sigma(p)$, and r may be taken as interpolant. ■

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

PROOF. We show that

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

if $x\supseteq\Sigma(Y)\cap\Sigma(Z)$.

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

(*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*) If $x\supseteq\Sigma(Y)\cap\Sigma(Z)$ then $Th(x\Box(Y+Z))\subseteq Th((x\Box Y)+(x\Box Z))$:
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 $q_1\vdash q_2\Rightarrow p$. According to the interpolation lemma there is an interpolant r with

$$\Sigma(r)\subseteq\Sigma(q_1)\cap\Sigma(q_2\Rightarrow p)\subseteq\Sigma(Y)\cap(\Sigma(Z)+(x\cap(\Sigma(Y)+\Sigma(Z))))=$$
$$\Sigma(Y)\cap(\Sigma(Z)+(x\cap\Sigma(Y))+(x\cap\Sigma(Z)))=$$
$$(\Sigma(Y)\cap\Sigma(Z))+(x\cap\Sigma(Y))+(x\cap\Sigma(Y)\cap\Sigma(Z))\subseteq x$$

such that $q_1\vdash r$ and $r\vdash q_2\Rightarrow p$ or, equivalently, $q_1\vdash r$ and $q_2\vdash r\Rightarrow p$. 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. Like (E3), (E4) implies the CRAIG interpolation lemma. Suppose $\vdash p\Rightarrow q$. Let $x=\Sigma(p)\cap\Sigma(q)$. Now $x\Box(<p>+<\neg q>)\equiv_{Th}(x\Box<p>)+(x\Box<\neg q>)$. Consequently $\mathbf{false}\in Th((x\Box<p>)+(x\Box<\neg q>))$. Choose $r_1\in Th(x\Box<p>)$, $r_2\in Th(x\Box<\neg q>)$ with $r_1\wedge r_2\vdash\mathbf{false}$. Then $p\vdash r_1\vdash\neg r_2\vdash q$ and we may take r_1 as interpolant. ■

The importance of the CRAIG interpolation lemma in the case of (E3) was pointed out by RENARDEL DE LAVALETTE [RDL88a]. Interestingly, the equivalence of (E3) and (E4) in the case of modular first-order theories does not carry over to modular equational theories (see SECTION 5.1).

In SECTION 3.1 we gave an example showing that the condition $x\supseteq\Sigma(Y)\cap\Sigma(Z)$ of (E4) is essential. It may be instructive to consider the same example once again in the present context. Using the notational conventions introduced in SECTION 3.5 the explicit typing may be dropped, and the example looks as follows:

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

Clearly, x does not contain $\Sigma(Y)\cap\Sigma(Z)=x+\mathbf{F}{:}c{:}B$. Furthermore,

$$Th(x\Box(Y+Z))\neq Th((x\Box Y)+(x\Box Z))$$

as $Th(x\Box(Y+Z))$ contains $T=F$ whereas $Th((x\Box Y)+(x\Box Z))$ does not.

The relations between $\mathbf{M}(fol)$, $\mathbf{M}_C(fol)$, and $\mathbf{T}(fol)$ 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 $\mathbf{M}_C(fol)$ is a homomorphic image of $\mathbf{M}(fol)$ and that $\mathbf{T}(fol)$ is a homomorphic image of $\mathbf{M}_C(fol)$.

For $X, Y \in FCME[fol]$ 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 4.2.3. $\mathbf{M}(fol) \not\cong \mathbf{M}_C(fol) \not\cong \mathbf{T}(fol)$.

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

Let NA and NB be defined as follows (see SECTIONS 2.2 and 3.5 for the notation used):

$$NA = (\mathbf{S}{:}A + \mathbf{F}{:}0{:}A + \mathbf{F}{:}S{:}A \to A){:}(<S(x) = S(y) \Rightarrow x = y> + <S(x) \neq 0>)$$

$$NB = (\mathbf{S}{:}B + \mathbf{F}{:}0'{:}B + \mathbf{F}{:}S'{:}B \to B){:}(<S'(x) = S'(y) \Rightarrow x = y> + <S'(x) \neq 0'>).$$

NA and NB are identical up to renaming. Take

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

and construct Y from X by adding a *hidden* bijection from A to B to it:

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

$$Y = \Sigma(X)\Box Z.$$

Clearly, every (countable) model of Y is a (countable) model of X. Conversely, let M be a model of X. M is a model of Y if a bijection $f{:}A \to B$ and its inverse g can be added to it. This is only possible if the carriers A and B of M have the same cardinality. So only the models of X whose carriers have the same cardinality are models of Y and Y has no other models. Notice that although models of Y themselves do not contain f and g ($\Sigma(Y)$ does not contain them), it must be possible to add them to satisfy Z.

Now, if M is a countable model of X the carriers A and B of M are both countably infinite (X does not have finite models), and thus have the same cardinality. Hence, M is also a countable model of Y and $\mathbf{M}_C \models X = Y$.

On the other hand, let M be a structure whose carriers A and B are both infinite but of different cardinality, then $M \in Mod(X)$, but $M \notin Mod(Y)$. Hence, $\mathbf{M} \not\models X = Y$.

Secondly, we prove $\mathbf{M}_C(fol) \not\cong \mathbf{T}(fol)$ by giving $X, Y \in CME[fol]$ such that $\mathbf{T}(fol) \models X = Y$, but $\mathbf{M}_C(fol) \not\models X = Y$. Take

$$Z = (\mathbf{S}{:}N+\mathbf{F}{:}0{:}N+\mathbf{F}{:}S{:}N\to N+\mathbf{F}{:}add{:}N\times N\to N){:}$$
$$(<S(x)=S(y)\Rightarrow x=y>+<S(x)\neq 0>+<x\neq 0\Rightarrow \exists y\, S(y)=x>+$$
$$<add(x,0)=x>+<add(x,S(y))=S(add(x,y))>+<add(x,S(y))\neq x>)$$
$$X = (\mathbf{F}{:}add{:}N\times N\to N)\Delta Z,$$

where Δ is the hiding operator defined in SECTION 3.3, and construct Y by adding a hidden "non-standard" constant c to X:

$$bool = (\mathbf{S}{:}B+\mathbf{F}{:}T{:}B+\mathbf{F}{:}F{:}B){:}(<T\neq F>+<x=T\vee x=F>)$$
$$Z' = X+bool+((\Sigma(X)+\Sigma(bool)+\mathbf{F}{:}c{:}N+\mathbf{F}{:}standard{:}N\to B){:}$$
$$(<standard(0)=T>+<standard(x)=standard(S(x))>+<standard(c)=F>))$$
$$Y = \Sigma(X)\Box Z'.$$

The axiom $add(x,S(y))\neq x$ of X rules out models with cycles and the axiom $x\neq 0\Rightarrow \exists y\, S(y)=x$ eliminates models containing more than a single copy of the standard model $\mathbb{N}$ of X. Hence, all models of X of cardinality $\aleph_1$ consist of a single copy of the natural numbers $\mathbb{N}$ and $\aleph_1$ copies of the integers $\mathbb{Z}$. All these models are isomorphic, so X is $\aleph_1$-categorical. Furthermore, X has no finite models. As a consequence, $Th(X)$ is complete, i.e., for any ϕ with $\Sigma(\phi)\subseteq\Sigma(X)$ either $\phi\in Th(X)$ or $\neg\phi\in Th(X)$. (This is an immediate consequence of the upward LÖWENHEIM-SKOLEM theorem — or see [CK73].)

Now, by construction $Th(X)\subseteq Th(Y)$, but as Y is clearly consistent, the completeness of $Th(X)$ implies $Th(X)=Th(Y)$. Hence, $\mathbb{T}(fol)\models X=Y$.

On the other hand $Mod_C(Y)\subseteq Mod_C(X)$ by construction, but $Mod_C(Y)$ does not contain the standard model $\mathbb{N}$ of X or any model isomorphic to it. In fact,

$$Mod_C(Y) = Mod_C(X)-\{M\in Mod_C(X)\mid M\cong\mathbb{N}\}.$$

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

REMARK. An immediate consequence of THEOREM 4.2.3 is that there are homomorphisms $\Phi_1{:}\mathbb{M}(fol)\to\mathbb{M}_C(fol)$ and $\Phi_2{:}\mathbb{M}_C(fol)\to\mathbb{T}(fol)$ which are surjective but not injective.

REMARK. In SECTION 3.4 we stated without proof that whether an extension is an enrichment or not may depend on the particular semantics used. We are now in a position to give an example of this. In the first part of the above proof

$$Z+X = X+((\mathbf{F}{:}f{:}A\to B+\mathbf{F}{:}g{:}B\to A){:}(<gf(x)=x>+<fg(y)=y>))+X = Z,$$

so Z is an extension of X. In $\mathbb{M}_C(fol)$ it is an enrichment of X as well due to the fact that

$$\mathbb{M}_C(fol)\models \Sigma(X)\Box Z=X,$$

but in $\mathbb{M}(fol)$ it is not as

$$\mathbb{M}(fol)\not\models \Sigma(X)\Box Z=X.$$

Obviously, the logically closed theory $Th(X)$ of a closed module expression $X\in CME[fol]$ is recursively enumerable. By a theorem of KLEENE on the power of first-order logic with auxiliary (hidden) predicate symbols [Kle52], the converse is also true. For every recursively enumerable *fol*-theory T there is a closed module expression $X\in CME[fol]$ such that $Th(X)=T$. Thus the domain of $\mathbb{T}(fol)$ consists *precisely* of the recursively enumerable logically closed theories.

Can the characterization of $\mathbb{M}(fol)$, $\mathbb{M}_C(fol)$, or $\mathbb{T}(fol)$ be improved by adding to $BMA[fol]$ some *open* equation not valid in the initial algebra $\mathbb{I}(BMA[fol])$ (not ω-derivable from $BMA[fol]$)? In other words, is there an open (conditional) equation e over the signature of $BMA[fol]$ such that, for instance,

$$\mathbb{T}(fol) \models e,$$

but

$$\mathbb{I}(BMA[fol]) \not\models e?$$

Although this is still an open question, we suspect that there is no such e due to the fact that the signature of $BMA[fol]$ is "logic free" in the sense that it does not describe the structure of *fol*-sentences, but considers them as atomic entities (constants). As a result, any open equation e not valid in $\mathbb{I}(BMA[fol])$ is probably too general to be valid in $\mathbb{M}(fol)$, $\mathbb{M}_C(fol)$, or $\mathbb{T}(fol)$.

We have made no particular effort to add sufficiently many axioms to $BMA[fol]$ to guarantee that every open equation valid in the initial model $\mathbb{I}(BMA[fol])$ is equationally derivable, i.e., we have not attempted to make $BMA[fol]$ *ω-complete* (cf. [Hee86]). Although open module expressions and open equations valid in $\mathbb{I}(BMA[fol])$ do not play an important role in this paper, they come to the fore when module algebra is applied to parametrized specifications.

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. Algebraic specifications from the viewpoint of module algebra

We now return to algebraic specifications, which were the original motivation for studying module algebra. The main questions are whether algebraic specifications viewed as equational theories or initial algebras satisfy the axioms of *BMA*. These questions are discussed in Section 5.1. In Section 5.2 the expressive power of conditional equational logic and equational logic are compared with each other, and in Section 5.3 the same is done for first-order logic and equational logic. Finally, in Section 5.4 relations with earlier results on algebraic specifications are briefly summarized.

In the sequel *eql* means many-sorted equational logic and *ceql* means many-sorted (positive) conditional equational logic. In our setting a modular algebraic specification corresponds to an expression in $CME[eql]$ or $CME[ceql]$ depending on whether conditions are allowed or not. Clearly,

$$CME[eql] \subseteq CME[ceql] \subseteq CME[fol].$$

5.1. Why not base a model of BMA on equational logic or initial algebras?

In addition to *Mod*, Mod_C and *Th*, yet another semantic mapping *EqTh* may be considered which is like *Th* but produces an equational theory at the visible level rather than a first-order theory. The most appropriate domain for *EqTh* is the domain of modular algebraic specifications $CME[eql]$. *EqTh* is defined as follows (we denote the set of equations over a signature x by $Eq(x)$):

$$\begin{aligned}
&EqTh(<\phi>) = Th(<\phi>) \cap Eq(\Sigma(<\phi>)) \\
&EqTh(\mathrm{T}(x)) = Th(\mathrm{T}(x)) \cap Eq(x) \\
&EqTh(r.X) = r.EqTh(X) \\
&EqTh(X+Y) = (EqTh(X)+EqTh(Y)) \cap Eq(\Sigma(X+Y)) \\
&EqTh(x \,\square\, Y) = \mathscr{L}(x) \cap EqTh(Y).
\end{aligned}$$

Notice that $EqTh(X+Y) \neq EqTh(X)+EqTh(Y)$. The $+$-operator in the right-hand side produces the first-order deductive closure of $EqTh(X)$ and $EqTh(Y)$ rather than the equational closure (see Section 4.1). Hence, an additional filtering with $Eq(\Sigma(X+Y))$ is necessary to obtain $EqTh(X+Y)$.

Clearly, $EqTh(X) \subseteq Th(X)$. Let for $X, Y \in CME[eql]$

$$X \equiv_{EqTh} Y \Leftrightarrow \Sigma(X)=\Sigma(Y) \;\&\; EqTh(X)=EqTh(Y).$$

We write

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

Does $\mathbf{EQT}(eql)$ satisfy $BMA[eql]$? As was shown in SECTION 4.2, in the case of first-order logic (E3) and (E4) are equivalent to the CRAIG interpolation lemma. RODENBURG and VAN GLABBEEK have proved that equational logic has an interpolation property as well [RG88]. Two finite sets of equations E and F with $E \vdash F$ (i.e., $E \vdash e$ for every equation $e \in F$) always have an interpolant, i.e., a finite (possibly empty) set of equations I with

$$\Sigma(I) \subseteq \Sigma(E) \cap \Sigma(F)$$

such that

$$E \vdash I \vdash F.$$

This interpolation property turns out to be equivalent to (E3) in the case of $\mathbf{EQT}(eql)$ [RG88], so we have

THEOREM 5.1.1. $\mathbf{EQT}(eql) \models$ (E3).

In this respect modular equational theories behave in the same way as modular first-order theories. Unfortunately, this does not apply to (E4):

THEOREM 5.1.2. $\mathbf{EQT}(eql) \not\models$ (E4).

PROOF. The equation

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

is a special case of (E4) (cf. equation (7) in SECTION 3.1). We give a pair of closed module expressions $X, Y \in CME[eql]$ such that

$$EqTh(\Sigma(Y)\Box(X+Y)) \neq EqTh((\Sigma(Y)\Box X)+Y).$$

Let

$$X = (\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A+\mathbf{F}{:}c{:}A){:}(<f(c)=c>)$$
$$Y = (\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A+\mathbf{F}{:}h{:}A\times A\times A\to A+\mathbf{F}{:}a{:}A+\mathbf{F}{:}b{:}A){:}$$
$$(<h(x,x,y)=y>+<h(x,f(x),a)=h(x,f(x),b)>).$$

Clearly,

$$a=b \in EqTh(\Sigma(Y)\Box(X+Y))$$

as $a=h(c,c,a)=h(c,f(c),a)=h(c,f(c),b)=h(c,c,b)=b$ and $\Sigma(a=b)\subseteq\Sigma(Y)$. We will show, however, that

$$a=b \notin EqTh((\Sigma(Y)\Box X)+Y).$$

The first component

$$\Sigma(Y)\Box X = (\Sigma(Y)\cap\Sigma(X))\Box X = (\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A)\Box X$$

does not export c and we will show that

$$EqTh(\Sigma(Y)\Box X) = EqTh(\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A)).$$

By construction $EqTh(\Sigma(Y)\Box X) \supseteq EqTh(\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A))$, so we only have to show $EqTh(\Sigma(Y)\Box X) \subseteq EqTh(\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A))$. Let $e \in EqTh(\Sigma(Y)\Box X)$, then e is valid in all models

of X. Apart from the function symbol f, e contains only universally quantified variables. Hence, e is valid in the subalgebras of the models of X as well. Now, every model M of $\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A)$ is a subalgebra of the model M' of X obtained by adding an element c with $f(c)=c$ to M, so all models of $\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A)$ occur among the subalgebras of models of X. This means that e is valid in all models of $\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A)$ and as a consequence $e\in EqTh(\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A))$.

Because $EqTh(\Sigma(Y)\square X)$ is trivial, it contributes nothing to the total equational theory:

$$EqTh((\Sigma(Y)\square X)+Y) = (EqTh(\Sigma(Y)\square X)+EqTh(Y))\cap Eq(\Sigma((\Sigma(Y)\square X)+Y)) =$$
$$(EqTh(\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A))+EqTh(Y))\cap Eq(\Sigma(Y)) = EqTh(Y)\cap Eq(\Sigma(Y)) = EqTh(Y).$$

Now consider the $\Sigma(Y)$-algebra M with carrier $\{a,b\}$ and functions f and h defined as follows:

$f(a)=b$
$f(b)=a$
$h(x,y,z)=z$ if $x=y$
$h(x,y,z)=a$ if $x\neq y$.

$M\models EqTh(Y)$ by inspection, but $M\not\models a=b$. Therefore $a=b\notin EqTh(Y)$. ■

REMARKS. (i) The above proof fails for $\mathbf{T}(fol)$ due to the fact that, whereas $EqTh(\Sigma(Y)\square X)$ is trivial, $Th(\Sigma(Y)\square X)$ contains the non-trivial sentence $\exists x\, f(x)=x$. In conjunction with $Th(Y)$ this is enough to prove $a=b$. This also shows that for the particular X and Y used in the proof

$$EqTh((\Sigma(Y)\square X)+Y)\neq Th((\Sigma(Y)\square X)+Y)\cap Eq(\Sigma(Y)),$$

so in general we only have

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

(ii) Let *CondEqTh* be the semantic mapping which assigns to each conditional equational specification the corresponding conditional equational theory, and let

$$\mathbf{CEQT}(ceql) = CME[ceql]/\equiv_{CondEqTh},$$

then

$$\mathbf{CEQT}(ceql)\not\models \text{(E4)}.$$

The proof is identical to the proof of THEOREM 5.1.2, but with *EqTh* replaced everywhere by *CondEqTh*. Note in particular that $CondEqTh(\Sigma(Y)\square X)$ is still equal to $CondEqTh(\mathrm{T}(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A))$. Y may be replaced by the equivalent $CME[ceql]$ expression

$$(\mathbf{S}{:}A+\mathbf{F}{:}f{:}A\to A+\mathbf{F}{:}a{:}A+\mathbf{F}{:}b{:}A){:}(<f(x)=x\Rightarrow a=b>)$$

(cf. the proof of THEOREM 5.2.1 in the next section).

(iii) Another consequence of the proof of THEOREM 5.1.2 is that (E4) cannot be saved by considering the interpretation *ClEqTh* defined by

$$ClEqTh(X) = EqTh(X)\cap ClEq(\Sigma(X)),\qquad (X\in CME[eql])$$

where $ClEq(x)$ is the set of *closed* equations over a signature x.

RENARDEL DE LAVALETTE [RDL88b] and RODENBURG & VAN GLABBEEK [RG88] have pointed out that in general (E4) corresponds to a stronger interpolation property than (E3). In the case of equational logic this stronger property would be that for three finite sets of equations E_1, E_2 and F with

$$E_1\cup E_2\vdash F$$

there would always be a finite set of equations I with

$$\Sigma(I) \subseteq \Sigma(E_1) \cap (\Sigma(E_2) + \Sigma(F))$$

and such that

$$E_1 \vdash I$$

and

$$I \cup E_2 \vdash F.$$

By taking $E_2 = \varnothing$ the weaker form corresponding to (E3) is obtained as a special case. THEOREM 5.1.2 implies that equational logic lacks the stronger interpolation property, a fact proved earlier by MAIBAUM & SADLER [MS85].

In view of the foregoing we conclude that
(i) $\mathbb{EQT}(eql)$ is a semantics of $CME[eql]$ only in the weaker sense of $BMA[eql]-$(E4).
(ii) $\mathbb{EQT}(eql)$ is *not* a homomorphic image of $\mathbb{T}(eql) = CME[eql]/\equiv_{Th}$, which is the restriction of $\mathbb{T}(fol)$ to $CME[eql]$.
(iii) As it makes essential use of (E4), 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 the kind of effective normalization procedure required in a practical system. Although $\mathbb{EQT}(eql)$ may at first sight seem a very plausible semantics, the loss of (effective) normalization shows that it should be rejected.

Algebraic specifications are often interpreted as initial algebras. Does this lead to a model of $BMA[eql]$? Unfortunately, again the answer is *no*. Let $I(X)$ be the initial algebra of $X \in CME[eql]$. Actually, $I(X)$ is not a single algebra but an isomorphism class of algebras. $I(X)$ is well-defined provided $\Sigma(X)$ does not have void (empty) sorts (see for instance [MG85] or [EM85]). Consider the following two closed module expressions $X, Y \in FCME[eql]$

$$X = \mathrm{T}(\mathbf{S}{:}A + \mathbf{F}{:}a{:}A + \mathbf{F}{:}b{:}A)$$
$$Y = (\mathbf{S}{:}A + \mathbf{F}{:}a{:}A + \mathbf{F}{:}b{:}A){:}(<a=b>).$$

On the one hand,

$$I(X+Y) = I(\mathrm{T}(\Sigma(Y)) + Y) \overset{(C4)}{=} I(Y).$$

On the other hand, $I(X) \not\models a=b$ ("no confusion") and $I(Y) \models a=b$, so using the $+$-operator on classes of algebras defined in SECTION 4.1

$$I(X)+I(Y) = \varnothing.$$

This simple example shows that initial algebras of algebraic specifications cannot be combined in a straightforward way.

We can nevertheless define an initial algebra for specifications $X \in CME[eql]$ on the basis of the semantics $\mathbb{T}(eql)$ which interprets algebraic specifications as first-order theories rather than as equational theories. This is a consequence of the following theorem which we do not prove here:

THEOREM 5.1.3. Let x be a signature and $Y_1, Y_2 \in FCME[eql]$ such that

$$BMA[eql] \vdash x \square Y_1 = x \square Y_2.$$

Then, if $\Sigma(Y_1)$ and $\Sigma(Y_2)$ are not void and if $x \square I(Y_1)$ and $x \square I(Y_2)$ are both minimal algebras,

$$x \square I(Y_1) \cong x \square I(Y_2).$$

The initial algebra of an $X \in CME[eql]$ is now defined as follows: first normalize X, i.e., take some $Y \in FCME[eql]$ such that

$$BMA[eql] \vdash X = \Sigma(X) \square Y,$$

and then take

$$I(X) = \Sigma(X) \square I(Y).$$

According to THEOREM 5.1.3 the resulting $I(X)$ is determined uniquely up to isomorphism provided it is minimal.

COMMENTS. (i) Let $BMA[eql] \vdash X = \Sigma(X) \square Y$ with $Y \in FCME[eql]$, then

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

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

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

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

This shows that the construction of $I(X)$ is consistent with the $\mathbb{T}(eql)$-semantics.
(ii) The normalization step which has to be performed prior to taking the initial algebra is justified by the $\mathbb{T}(eql)$-semantics, which is not directly related to equational logic.

5.2. Conditional equations do not add expressive power

From the viewpoint of the full model class semantics $\mathbb{M}(fol)$ (and hence also from the viewpoint of the countable model semantics $\mathbb{M}_C(fol)$ and the theory semantics $\mathbb{T}(fol)$) positive conditional equations have the same expressive power as unconditional equations:

THEOREM 5.2.1. For every $X \in CME[eql]$ there is a $Y \in CME[ceql]$ such that $\mathbb{M}(fol) \models X = Y$ and, conversely, for every $X \in CME[ceql]$ there is a $Y \in CME[eql]$ such that $\mathbb{M}(fol) \models X = Y$.

Using the notation introduced in SECTION 3.2, the theorem can be expressed as

$$\mathbb{M}(fol) \models CME[ceql] = CME[eql].$$

PROOF. As $CME[eql] \subseteq CME[ceql]$, the first half of the theorem is trivial. To prove the second half take $X \in CME[ceql]$. We have to find a $Y \in CME[eql]$ such that $\mathbb{M}(fol) \models X = Y$. We only have to consider X of the form $<\phi>$ where ϕ is a conditional equation with a single condition. The case of multiple equations with multiple conditions can be dealt with in a similar manner.

Now let $\phi \equiv t_1 = t_2 \Rightarrow t_3 = t_4$ with t_1, t_2 terms of sort S and t_3, t_4 terms of sort U. We show that ϕ can be replaced by a *hidden* function $h : S \times S \times U \to U$ (with h a new symbol not in $\Sigma(\phi)$) satisfying two unconditional equations

$$e_1 \equiv h(x,x,u) = u,$$
$$e_2 \equiv h(t_1,t_2,t_3) = h(t_1,t_2,t_4).$$

Define

$$Z = (\mathbf{F}:h:S \times S \times U \to U):(<e_1> + <e_2>)$$
$$Y = \Sigma(<\phi>) \square Z.$$

Clearly, $Y \in CME[eql]$ and $\Sigma(Y) = \Sigma(<\phi>) \cap \Sigma(Z) = \Sigma(<\phi>)$. Now $\mathbb{M}(fol) \models <\phi> = Y$. Indeed, as $e_1, e_2 \vdash \phi$ we have on the one hand $Mod(<\phi>) \supseteq Mod(Y)$. On the other hand, each model M of $<\phi>$ can be extended to a model M' of Z by adding a function h satisfying e_1 and e_2 as follows:

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

where u_0 is some fixed element of carrier U of M. Hence,

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

and $\mathbb{M} \models <\phi> = Y$. ■

5.3. 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:

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

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*) $\varnothing \Box X = \varnothing \Box incons.$

Note that the "empty" algebra $A_\varnothing$ (SECTION 4.1) is a model of $\mathrm{T}(\varnothing)$ but not of $\varnothing \Box incons$. Hence, $\mathrm{T}(\varnothing)$ and $\varnothing \Box incons$ are different in $\mathbb{M}(fol)$. In case (*a*) we may say that X is consistent and in case (*b*) that it is inconsistent. Both *boolincons* itself as well as *boolem* + *boolincons* are consistent.

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

(1) There is no $X \in CME[eql]$ such that $\mathbb{M}(fol) \models X = boolcons$, or, equivalently, using the notation introduced in SECTION 3.2, $\mathbb{M}(fol) \not\models boolcons \in CME[eql]$.

PROOF. Every model of *boolcons* has a carrier B with at least two elements, whereas an $X \in CME[eql]$ always has a trivial model all of whose carriers have only a single element. Hence, $Mod(X) \neq Mod(boolcons)$ and $\mathbb{M}(fol) \not\models boolcons \in CME[eql]$. ■

(2) $\mathbb{M}(fol) \not\models boolem \in CME[eql]$.

PROOF. *Boolem* has a non-trivial model with two elements but it has no models with more than two elements. Now let $X = \Sigma(boolem)\Box X' \in CME[eql]$ with X' flat and assume that $Mod(X) = Mod(boolem)$. Then X' has a model M such that $M \not\models T = F$. Let $\mathbf{F}{:}c{:}B \notin \Sigma(X')$. $X' + \mathrm{T}(\mathbf{F}{:}c{:}B)$ is an equational specification so it has an initial model I. If $I \models c = T$, then $Th(X') \vdash c = T$ and $Th(X') \vdash x = T$ which implies $Th(X') \vdash F = T$ contradicting $M \not\models T = F$. Similarly, $I \not\models c = F$. Consequently, sort B of $\Sigma(X')\Box I$ has more than two elements. As $\Sigma(X')\Box I \in Mod(X')$ by construction, $\Sigma(boolem)\Box I \in Mod(X)$, but $\Sigma(boolem)\Box I \notin Mod(boolem)$. 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. An $X \in CME[fol]$ can be transformed to an equivalent $Y \in CME[eql,bool]$ by performing the following steps:
(*a*) Existential quantifiers in X are replaced by hidden Skolem functions. The resulting X' contains only universal axioms and is equivalent to X in $\mathbb{M}(fol)$.
(*b*) 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 among the axioms of X' are replaced by equations over B ($t_1 = t_2$ and $t_1 \neq t_2$ with t_1, t_2 terms of sort S are replaced by $eq_S(t_1,t_2) = T$ and $eq_S(t_1,t_2) = F$ respectively).
(*c*) Finally, the desired $Y \in CME[eql,bool]$ is obtained by replacing the universal axioms of X' by equations over *bool* using hidden *bool*-operators like $\neg$, $\wedge$, and $\vee$. ■

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

(ii) A more interesting proof of (5) would be based on a set of conditional rewrite rules (conditional equations) for transforming an arbitrary $X \in CME[fol]$ systematically into an equivalent $Y \in CME[eql,bool]$. An adequate presentation of such rules would require a detailed specification of first-order logic similar to the specification of signatures we gave in SECTION 2.2.

(iii) There are two minor open questions:

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

(*b*) The same question as (*a*) 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."

5.4. Relations with earlier results on algebraic specifications

In this section we summarize known results on the power of initial/final algebra specification using the language of module algebra. Like before, the initial algebra of an $X \in CME[eql]$ without void sorts is denoted by $I(X)$.

(1) For a minimal algebra A with signature x the following two properties are equivalent
(*a*) A is semicomputable;
(*b*) A has an initial algebra specification with hidden sorts and functions, i.e., $A \cong x\Box I(Y)$ for some $Y \in FCME[eql]$.
The implication (*b*) $\Rightarrow$ (*a*) is immediate. The converse is proved in detail in [BT87] for the single-sorted case. It is an open question whether Y can always be chosen in such a way that no hidden sorts are introduced, i.e., $sorts(x) = sorts(\Sigma(Y))$.

(2) If A is a minimal computable algebra with signature x, it has an initial algebra specification with hidden functions only, i.e., there is a $Y \in FCME[eql]$ such that
(*a*) $A \cong x\Box I(Y)$;
(*b*) $sorts(x) = sorts(\Sigma(Y))$.
See [BT82]. MAJSTER [M77] discovered that there are computable algebras for which there is no $Y \in FCME[eql]$ such that $A \cong I(Y)$. In addition to (*a*) and (*b*) Y can have several further properties (but not simultaneously):
(*c*) Y has a complete (i.e., confluent and terminating) term rewriting system. See [BT80] for a proof of the single-sorted case.
(*d*) Both the number of equations of Y and the number of constants and functions of $\Sigma(Y)$ are linearly bounded by the number of sorts of x. Moreover, $I(Y)$ is also the final Y-algebra which means that $I(Y)$ does not have non-trivial homomorphic images. See [BT82]. (*Signatures* of SECTION 2.2 is an example of such a Y for the algebra of signatures.)
(*e*) $\Sigma(Y)$ 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 x, there is a $Y \in FCME[ceql]$ such that
(*a*) Y has a *unique* final algebra $F(Y)$ (which in this case has the property that each of its homomorphic images satisfying Y is either $F(Y)$ itself or the trivial $\Sigma(Y)$-algebra);
(*b*) $A \cong x \square F(Y)$;
(*c*) $sorts(x) = sorts(\Sigma(Y))$.
See [BT83].

(4) Let $f:\omega \to \omega$ be a recursive function. There is an open module expression $Y(X) \in FME[eql]$ with free variable X of sort M such that for all $n \in \omega$

$$I(Y(\Sigma_\omega : <S^n(0) = c>)) \text{ is finite, and}$$
$$\text{card}(I(Y(\Sigma_\omega : <S^n(0) = c>))) > f(n),$$

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

(5) In the absence of hiding conditional equations are more powerful than unconditional ones from the viewpoint of initial algebra semantics. The following example illustrates this fact:

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

$\mathbb{N} = I(\mathrm{T}(\Sigma_N))$ 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 a Y with $I(Y) \cong A$, but that $FCME[eql]$ does not. Of course, in view of (2) (A is clearly computable) there also exists a $Y \in CME[eql]$ such that $I(Y) \cong A$.

(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 [BT87] it is shown that there is no $Y \in FCME[ceql]$ such that $I(Y) \cong \mathbb{N}_P$, so $\mathbb{N}_P$ has no initial algebra specification without hidden functions.

ACKNOWLEDGEMENTS
We would like to thank N.W.P. van Diepen, R.J. van Glabbeek, P.R.H. Hendriks, C.P.J. Koymans, E. Nieuwland, P.H. Rodenburg, and G.R. Renardel de Lavalette for their many helpful comments and suggestions.

REFERENCES

(References [E82], [GB84], [GM82], and [Hor85] are not cited in the text.)

[B87] J.A. BERGSTRA, *Terminologie van Algebraïsche Specificaties*, Kluwer, 1987 [In Dutch].

[BEPP87] E.K. BLUM, H. EHRIG & F. PARISI-PRESICCE, Algebraic specification of modules and their basic interconnections, *Journal of Computer and System Sciences*, **34** (1987), 293-339.

[BG80] R.M. BURSTALL & J.A. GOGUEN, The semantics of CLEAR, a specification language, in: *Abstract Software Specifications*, Lecture Notes in Computer Science, 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; also: The algebraic specification formalism ASF, in: J.A. BERGSTRA, J. HEERING & P. KLINT, eds., *Algebraic Specification*, The ACM Press in association with Addison-Wesley, to appear, 1989.

[BJ80] G. BOOLOS & R. JEFFREY, *Computability and Logic*, 2nd ed., Cambridge University Press, 1980.

[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.

[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, Lecture Notes in Computer Science, 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.

[BT87] J.A. BERGSTRA & J.V. TUCKER, Algebraic specifications of computable and semi-computable data types, *Theoretical Computer Science*, **50** (1987), pp. 137-181.

[C57] W. CRAIG, Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory, *Journal of Symbolic Logic*, **22** (1957), pp. 269-285.

[CK73] C.C. CHANG & H.J. KEISLER, *Model Theory*, North-Holland, 1973.

[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. JOUANNAUD & J. MESEGUER, Principles of OBJ2, in: *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.

[Gau86] M.-C. GAUDEL, Toward structured algebraic specifications, in *Esprit '85: Status Report of Continuing Work*, Vol. 1, North-Holland, 1986, pp. 493-510.

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

[GM82] J.A. GOGUEN & J. MESEGUER, Universal realization, persistent interconnection and implementation of abstract modules, in: M. NIELSEN & E.M. SCHMIDT, eds., *Proceedings 9th International Conference on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 140, Springer-Verlag, 1982, pp. 265-281.

[Hee86] J. HEERING, Partial evaluation and ω-completeness of algebraic specifications, *Theoretical Computer Science*, **43** (1986), pp. 149-167.

[Hor85] J.J. HORNING, Combining algebraic and predicative specifications in LARCH, in: H. EHRIG, C. FLOYD, M. NIVAT & J. THATCHER, eds., *Formal Methods for Software Development*, TAPSOFT Proceedings, Vol. 2, Lecture Notes in Computer Science, 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 [In French].

[Kla83] H.A. KLAEREN, *Algebraische Spezifikation*, Springer-Verlag, 1983 [In German].

[Kle52] S.C. KLEENE, Finite axiomatizability of theories in the predicate calculus using additional predicate symbols, *Memoirs of the American Mathematical Society*, No. 10 (1952), pp. 27-68.

[Koy86] C.P.J. KOYMANS, Personal communication.

[Lip83] U. LIPECK, Ein algebraischer Kalkül für einen strukturierten Entwurf von Datenabstraktionen, Dissertation, Forschungsbericht Nr. 148, Abteilung Informatik, Universität Dortmund, 1983 [In German].

[Loe85] J. LOECKX, A formal description of the specification language OBSCURE, Report A 85/15, Universität des Saarlandes, Saarbrücken, 1985.

[M77] M.E. MAJSTER, Limits of the "algebraic" specification of abstract data types, *SIGPLAN Notices*, **12** (1977), 10, pp. 37-42.

[MG85] J. MESEGUER & J.A. GOGUEN, Initiality, induction, and computability, in: M. NIVAT & J.C. REYNOLDS, eds., *Algebraic methods in Semantics*, Cambridge University Press, 1985, pp. 459-541.

[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: H. EHRIG, C. FLOYD, M. NIVAT & J. THATCHER, eds., *Formal Methods for Software Development*, TAPSOFT Proceedings, Vol. 2, Lecture Notes in Computer Science, 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.

[PP87] F. PARISI-PRESICCE, Union and actualization of module specifications: some compatibility results, *Journal of Computer and System Sciences*, **35** (1987), pp. 72-95.

[RDL88a] G.R. RENARDEL DE LAVALETTE, Modularisation, parameterisation, interpolation, Logic Group Preprint Series No. 32, Department of Philosophy, University of Utrecht, 1988.

[RDL88b] G.R. RENARDEL DE LAVALETTE, Preliminary remarks on theories and interpolation, unpublished note, July 20, 1988.

[RG88] P.H. RODENBURG & R.J. VAN GLABBEEK, An interpolation theorem in equational logic, Report CS-R8838, Department of Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[S67] 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.