MODULE ALGEBRA FOR RELATIONAL SPECIFICATIONS

J. A. Bergstra

Department of Computer Science, University of Amsterdam
Department of Philosophy, University of Utrecht*

Abstract

Module algebra is described as a parametrised data type that takes a three sorted parameter, involving signatures, renamings and atomic specifications. An actual parameter describing relational specifications is given and a biinterpretation model is made for the resulting specification.

1986 CR Categories: D.2.0 [software engineering]: requirements/ specifications- languages; D.2.2 [software engineering]: tools and techniques- modules and interfaces; D.3.3 [programming languages]: language constructs- modules; F.3.2 [logics and meanings of programs]: semantics of programming languages- algebraic approaches to semantics.

1980 Mathematics Subject Classification: 68B10[software]: analysis of programs- semantics.

Key words and phrases: signature, relational structure, relational specification, module algebra, biinterpretation of module objects.

Note: partial support was received from the European Communities under ESPRIT project 432 (An Integrated formal Approach to Industrial Software Development - METEOR)

* Department of Philosophy, University of Utrecht, Heidelberglaan 2, 3584 CS, Utrecht, The Netherlands.

1 Introduction.

In this paper we introduce the module algebra of [BHK86a], in a slightly generalised version. The generalisation is obtained by not specifying the algebra of signatures in detail but rather describing in an axiomatic way its most important features only. In this way module algebra is turned into an algebraic specification, with a three-sorted parameter. The sorts of the parameter are: SIG (signatures), ATREN (atomic renamings) and ATSPEC (atomic specifications).

This level of generality allows us to classify more and quite different structures as models of module algebra. In particular we describe the concept of a relational specification for a relational structure and we describe the semantics of structured relational specifications by means of (generalised) module algebra.

In section 5 we construct a model of this axiom system using biinterpretations. This construction follows as closely as possible the construction for biinterpretation models for BMA[fol] that was given in [BHK 86b].

Module algebra being a meta-theory of modularised specifications rather than an optimised an user friendly syntax we feel not obliged to use the format of module algebra in presenting the specifications BMA[Sig-Atspec] and BMA[rsig-ru] below. This specification will be presented in an ad hoc syntax that introduces minimal overhead.

2. signatures and atomic specifications

Sig-Atspec is a module that captures all operations on signatures, atomic renamings and atomic specifications that matter for us here.

module: Sig-Atspec

sorts:

SIG	signatures
ATREN	atomic renamings
ATSPEC	atomic

specifications

functions:

$\emptyset$: SIG	empty signature
$+$: SIG $\times$ SIG $\rightarrow$ SIG	union
$\cap$: SIG $\times$ SIG $\rightarrow$ SIG	intersection
Σ: ATSPEC $\rightarrow$ SIG	signature-of
Σ: ATREN $\rightarrow$ SIG	signature-of
invΣ: ATREN $\rightarrow$ SIG	invariant part of signature-of
$\cdot$: ATREN $\times$ SIG $\rightarrow$ SIG	appplication of renaming
$\cdot$: ATREN $\times$ ATSPEC $\rightarrow$ ATSPEC	application of renaming

variables:

$x, y, z \in$ SIG
$r \in$ ATREN
$\phi \in$ ATSPEC

abbreviation:

$x \subseteq y \equiv \exists z \in \text{SIG}\ (x + z = y)$

axioms:

$x + (y + z) = (x + y) + z$
$x + y = y + x$
$x + x = x$
$r \cdot (x + y) = r \cdot x + r \cdot y$
$r \cdot (x \cap y) = (r \cdot x) \cap (r \cdot y)$
$\Sigma(r \cdot x) = r \cdot \Sigma(x)$
$r \cdot r \cdot x = x$
$\Sigma(r \cdot \phi) = r \cdot \Sigma(\phi)$
$r \cdot r \cdot \phi = \phi$
$\Sigma(r) \cap \Sigma(x) \subseteq \text{inv}\Sigma(r) \rightarrow r \cdot x = x$

requirement:

For each $x, y, z \in$ SIG, with $x \subseteq y$ and $x \subseteq z$, there $n \in \omega$ and a sequence $r_1, \ldots, r_n \in$ ATREN such that

(i) for $i \in \{1, \ldots, n\}$ $\Sigma(r_i) \cap x \subseteq \text{inv}\Sigma(r_i)$, and

(ii) $(r_1 \cdot \ldots\ldots \cdot r_n \cdot y) \cap z = x$

comments:

Signatures will usually be structured families of

names in which a single name can occuur in several roles. Atomic renamings permute two names that occur with the same role. therefore we require that r is its own inverse.

end of: Sig-Atspec

3. Axioms of basic module algebra

Next we present the axioms of basic module algebra using an import of the previously described module Sig-Atspec. The various operators involved have been explained and motivated at length in [BHK86a, b].

module: BMA[Sig-Atspec]

parameter: Sig-Atspec

sort:	M	this sort will contain modules, or rather modular specifications
constants:	$\langle . \rangle$: Atspec → M	converts an atomic specification into a module
functions:	Σ: M → SIG	determines the exported, (external, visible) signature.
	T:SIG(N) → M	natural injection of signatures into modules.
	·: ATREN × M → M	application of atomic renamings
	+: M × M → M	combination (union)
	□: SIG × M → M	export
variables:	r ∈ ATREN	
	x, x',y ∈ SIG	
	X, Y, Z ∈ M	
axioms:	$\Sigma(\langle \phi \rangle) = \Sigma(\phi)$	S1
	$\Sigma(T(x)) = x$	S2

$$\Sigma(X + Y) = \Sigma(X) + \Sigma(Y) \qquad S3$$
$$\Sigma(x \,\square\, Y) = x \cap \Sigma(Y) \qquad S4$$
$$\Sigma(r \cdot X) = r \cdot \Sigma(X) \qquad S5$$

$$r \cdot \langle\phi\rangle = \langle r \cdot \phi\rangle \qquad R1$$
$$r \cdot T(x) = T(r \cdot x) \qquad R2$$
$$r \cdot (X + Y) = (r \cdot X) + (r \cdot Y) \qquad R3$$
$$r \cdot (x \,\square\, Y) = (r \cdot x) \,\square\, (r \cdot Y) \qquad R4$$
$$r \cdot (r \cdot X)) = X \qquad R5$$
$$\Sigma(r) \cap \Sigma(X) \subseteq \mathrm{inv}\Sigma(r) \rightarrow r \cdot X = X \qquad R6$$

$$X + Y = Y + X \qquad C1$$
$$(X + Y) + Z = X + (Y + Z) \qquad C2$$
$$T(x + y) = T(x) + T(y) \qquad C3$$
$$X = T(\Sigma(X)) + X \qquad C4$$
$$X + (y \,\square\, X) = X \qquad C5$$

$$\Sigma(X) \,\square\, X = X \qquad E1$$
$$x \,\square\, (y \,\square\, Z) = (x \cap y) \,\square\, Z \qquad E2$$
$$x \,\square\, (T(y) + Z) = T(x \cap y) + x \,\square\, Z \qquad E3$$
$$x \supseteq \Sigma(Y) \cap \Sigma(Z) \rightarrow x \,\square\, (Y + Z) = (x \,\square\, Y) + (x \,\square\, Z) \qquad E4$$

end of: BMA[Sig-Atspec]

comments: The bunch of axioms that has been presented in the modules Sig-Atspec and BMA[Sig-Atspec] above needs several comments.

(i) The algebra of signatures SIG over a given infinite set of names, together with the sort Ax[fol] of first order axioms that was introduced in [BHK86a] and described slightly differently in [BHK86a, b] constitutes a model of Sig-Atspec. The present description is more general in the sense that it does not require anything about the notion of a signature except the existence of some operations with reasonably plausible axioms.

(ii) The requirement in the specification of Sig-Atspec is not at all first order because it contains a quantifier over ω. Further it contains an unbounded number of existential quantifiers. We have no formal

semantics for this, other than the (obvious) statement that each realisation of the specification should satisfy this requirement. At an informal level this is perfectly clear. Moreover, we are not yet interested in all models of Sig-Atspec or BMA[Sig-Atspec] but rather in a few interesting ones.

(iii) We have not investigated the formal properties of BMA[Sig-Atspec]. For instance it would be interesting to know whether every model lof Sig-Atspec can be extended to a model of BMA[Sig-Atspec]. Also it is not clear to what extent a normal form theorem can be shown for arbitrary models of BMA[Sig-Atspec].

4 Relational specifications

In this section we will introduce relational specifications, a mechanism very closely related to PROLOG and for which no novelty is claimed here at all. Relational specifications constitute a counterpart to algebraic spoecifications in the following sense.

Algebraic specifications using equational logic are about **many sorted** algebras with total **functions**. The various sorts of such a many sorted algebra are thought to be disjoint, thus avoiding subsorts and greatly simplifying the problems involved in type checking and type inference in this case. The many sorted algebras provide a rudimentary but very effective typing mechanism, and the fact that all functions are total implies that in no way there is some additional type 'undefined' which might cause severe complications for a type checking mechanism and might even make type checking depending on the axioms (equations).

Relational specifications are at the opposite end of the spectrum. All objects are contained in an implicit supertype. On the type there are several partial **constructors**, and moreover, there are a family of **relations**. These relations also play the role of types and in this setting types need not to be disjoint. There are axioms that specify when the constructors are defined and when the relations are true. These axioms have the form of conditional rules and resemble very much a PROLOG program, be it that the constructors are partial. Semantically a relational specification defines a class of relational algebras with partial operations. This class also has an initial element.

We will now first formally describe what a signature is for a relational specification, then we describe the syntax and semantics of relational specifications with special emphasis on initial algebra semantics. (In the next section we will extend the specification mechanism with modularisation and hiding by means of module algebra.)

4. 1. Signatures

The signature of a flat relational specification consists of a finite set CONS of pairs (a, n) of a constructor name a and an arity $n \in \omega$, together with a finite set REL of pairs (b, n) of a relation name b and an arity $n \in \omega$. It is assumed that constructor names and relation names are different. Constructors with arity 0 are called constants, relations with arity 0 are also called propositions, and relations with arity 1 are also called types. Implicit in such a signature is the universal type U.

Let Σ = (CONS, REL) be a relational signature. Expressions over this signature are made as follows. There are object expressions and relational expressions. Constants and variables are object expressions, and if $t_1, \ldots, t_k$ are object expressions and a is a constructor name that occurs in CONS with arity k then $a(t_1, \ldots, t_k)$ is an object expression. Similarly propositions are relational expressions and given object expressions $t_1, \ldots, t_k$ and a relation name b that occurs in REL with arity k, also $b(t_1, \ldots, t_k)$ is a relational expression.

A relational structure $\mathfrak{A}$ of signature Σ is a set $A_{\mathfrak{A}}$ equipped with for each constructor name a with arity k a partial k-ary function $a_{\mathfrak{A}}$, and for each k-ary relation name b a k-ary relation $b_{\mathfrak{A}}$.

4. 2 Rules and specifications

A rule is a construct of the following form T_1 & ..... & $T_n \rightarrow T_{n+1}$. Here the T_i are either object expressions or relational expressions. Satisfaction of these rules in a relational structure is defined as follows. Let VAR be the collection of variables, and let $\mathfrak{A}$ be a given relational structure with domain A. Then a valuation is a mapping α: VAR $\rightarrow$ A $\cup$ {*},

and inductively we define $(\mathfrak{A},\alpha \models T)$, the value of T in $\mathfrak{A}$ under α, for object expressions T. $(\mathfrak{A},\alpha \models T)$ is either an element of $A \cup \{*\}$ which stands for undefined. We view the constructors of $\mathfrak{A}$ as total mappings on $A \cup \{*\}$ in a natural way by having value * in case of undefined and whenever any of the arguments is undefined.

$(\mathfrak{A},\alpha \models x) = \alpha(x)$ for $x \in VAR$

$(\mathfrak{A},\alpha \models a(x_1, \ldots .x_k)) = a_{\mathfrak{A}}(\alpha(x_1), \ldots ,\alpha(x_k))$, for a k-ary constructor name $a \in CONS$.

$(\mathfrak{A},\alpha \models a(T_1, \ldots , T_k)) =$

$(\mathfrak{A},\alpha[x_1/(\mathfrak{A},\alpha \models T_1), \ldots , x_k/(\mathfrak{A},\alpha \models T_k)] \models a(x_1, \ldots , x_k)$

For a k-ary constructor symbol a in CONS.

In addition and slightly unusually we also define satisfaction for object expressions T:

$\mathfrak{A},\alpha \models T$ if $(\mathfrak{A},\alpha \models T) \in A$

$\mathfrak{A},\alpha \not\models T$ if $(\mathfrak{A},\alpha \models T) = *$

For relational expressions $b(T_1, \ldots , T_k)$, we have simply

$\mathfrak{A},\alpha \models b(T_1, \ldots , T_k)$ iff all values $a_i = (\mathfrak{A},\alpha \models T_i) \in A$ (for $1 \leq i \leq k$) and $b_{\mathfrak{A}}(a_1, \ldots ,a_k)$.

For a rule $T_1 \& \ldots \& T_n \rightarrow T_{n+1}$ the satisfaction definitions are:

$\mathfrak{A},\alpha \models T_1 \& \ldots \& T_n \rightarrow T_{n+1} \Leftrightarrow$

$\mathfrak{A},\alpha \models T_1 \& \ldots \& \mathfrak{A},\alpha \models T_n \rightarrow \mathfrak{A},\alpha \models T_{n+1}$

$\mathfrak{A} \models T_1 \& \ldots \& T_n \rightarrow T_{n+1} \Leftrightarrow$ for all valuations α

$\mathfrak{A},\alpha \models T_1 \& \ldots \& T_n \rightarrow T_{n+1}$.

A (flat) relational specification is a pair (Σ, R) where Σ is a signature and R is a finite set of rules over Σ.

4. 3. Initial relational algebras

Consider a relational specification (Σ,R). Let us assume that Σ includes at least one constant. Now there is an initial relational algebra $T_I(\Sigma, R)$ for (Σ,R). The domain of this initial algebra consists of all closed object

expressions T that exist in all relational structures $\mathfrak{A}$ with $\mathfrak{A} \models R$, and a closed relational expression T is true in $\mathfrak{A}$ if it is true in all relational structures $\mathfrak{A}$ with $\mathfrak{A} \models R$. This initial algebra has a recursively enumerable domain and the interpretation of the relations are recursively enumerable predicates as well.

4. 5. Examples

In these examples we will use some notational conventions that must be explained here. Types and relations of arity 2 and more are decared separately. For relations the arity is explicitly given, just as for constructors. No distiction between constants and other constructors is made. It is ensured that no name occurs both as a constructor name and as a relation name. Further we allow conjunctions in the conclusions of rules as an abbreviation for a list of rules with the same conditions.

4. 5. 1. A stack of integers

constructors: 0/0, Ø/0, suc/1, push/2, pop/1, top/1
types: NAT, Stack
relations: eq/2
rules:
→ 0
→ Ø
→ NAT(0)
NAT(x) → NAT(suc(x))
→ Stack(Ø)
Stack(x) & NAT(y) → Stack(push(y, x))
push(x, y) → NAT(top(push(x, y))
push(x, y) → Stack(pop(push(x, y))
NAT(x) & NAT(y) → eq(suc(x), suc(y))
NAT(x) & NAT(x') & Stack(y) & Stack(y') & eq(x, x') &
→ eq(y, y') → eq(push(x, y), push(x', y'))
x → eq(x, x)
eq(x,y) → eq(y, x)
eq(x, y) & eq(y, z) → eq(x, z)
eq(x, push(y, z)) → eq(y, top(x)) & eq(z, pop(x))

4. 5. 2. Numerical expressions

constructors: 0/0, suc/1, +/2, ·/2, pred/2
types: NUM, NUMEX
relations: plus/3, mult/3, eval/2, lengthex/2,
length/2, diff/2
rules:
NUM(0)
NUM(x) → NUM(suc(x))
NUM(x) → NUMEX(x)
NUMEX(x) & NUMEX(y) → NUMEX(x + y) & NUMEX(x · y)
NUMEX(x) & eval(x,y) & diff(0, y) → NUMEX(pred(x))
NUM(x) → plus(x, 0, x)
NUM(x) & NUM(y) & plus(x, y, z) → plus(x, suc(y), suc(z))
NUM(x) → mult(x, 0, 0)
NUM(x) & NUM(y) & mult(x, y, z) & plus(x, z, u) →

→ mult(x, suc(y), u)
NUM(x) → diff(0, suc(x))
diff(x, y) → diff(suc(x), suc(y))
→ eval(0, 0)
eval(x, y) → eval(suc(x), suc(y))
NUMEX(x) & NUMEX(y) & eval(x, x') & eval(y, y') &
& plus(x', y', z) → eval(x + y, z)
NUMEX(x) & NUMEX(y) & eval(x, x') & eval(y, y') &
& mult(x', y', z) → eval(x', y', z)
NUMEX(x) & NUM(y) & eval(x, suc(y)) → eval(pred(x), y)
→ lengthex(0, suc(0))
NUMEX(x) & length(x, x') & NUM(y) & lengthex(y, y') →
→ lengthex(x + y, suc(x' + y')
& lengthex (x · y, suc(x' + y')
& lengthex(suc(x), suc(x'))
NUMEX(pred(x)) & lengthex(x, y) →
→ lengthex(pred(x), suc(y))
NUMEX(x) & length(x, y) & eval(y, z) → length(x, z)

Comment: the relations plus, mult, eval, lengthex and length are graphs of partial functions. lengthex first produces an expression that denotes the length of its argument. This expression is thereafter evaluated to a numeral in order to obtain the graph of 'length', which computes the actual length of an expression.

4. 6. rsig-ru, a model of Sig-Atspec.

Starting with two countably infinite collections of names CN (constructor names) and RN (relation names), we obtain a model (rsig-ru) of the axioms of Sig-Atspec as follows:

SIG: signatures as in 4. 2, with the understanding the constructor names must be taken from CN and that relation names must be taken from RN. Union and intersection are taken componentwise.

ATREN: Pairs of CN-names with corresponding arities plus pairs of RN-names with corresponding arities.

Application of an atomic renaming on a signature consists of permuting both names in the pair in the signature declaration.

ATSPEC: This sort consists of all rules ρ. Σ(ρ) determines the smallest signature over which rule ρ can be written. Application of an atomic renaming (a, b)/k interchanges the names a and b whenever these occur in Σ with arity k. Two rules are identified if they can be transformed into oneanother by means of renaming of variables (of course properly done in such a way that no name clash is introduced). In this simple situation invΣ(ρ) is just Σ(ρ).

Definition 4. 6. 1. With CME[ru] we denote the closed (module) expressions of sort M that can be written in the language of BMA[rsig-ru].

4. 7. BMA[rsig-ru] = BMA[Sig-Atspec] (Sig-Atspec := rsig-ru)

Of course we can pass the actual algebra rsig-ru for the parameter Sig-Atspec in BMA[Sig-Atspec]. It has to be verified that rsig-ru satisfies the requirements but that is no problem. It is of course not quite evident what the semantics of this parameter actualisation is. Rather than passing the algebra we pass the equational theory of its diagram. Thus we add so many constants as is necessary to make rsig-ru a minimal algebra and then we add all closed identities over the extended signature. In this way BMA[rsig-ru] becomes a specification that still can be provided with various models rather than some specific model already. In 5 we will provide a model for BMA[rsig-ru].

4. 8. Interpretation of rules in first order logic.

If we want to understand the relational specifications from the point of view of first order logic, we must add to each signature a unary relation U (the universal type) and a unary relation E (the existence predicate, or, the type of existing objects). For E there are these axioms:

$E(f(t_1, \ldots, T_k)) \rightarrow E(T_1) \,\&\, \ldots. \,\&\, E(T_k)$, for all $k \in \omega$ and all

constructors f with arity k. Translating a relational specification into this framework amounts to first of all adding these axioms and extending the signature as indicated, and secondly replacing in all rules object expressions T by relational expressions E(T) thus obtaining a usual Horn-clause (which is just a first order formula). So in a standard way relational specifications can be transformed to first order specifications, and in this way for instance we can define without difficulty the first order theory of a relational specification, its class of models, its closed theory etc. An example of extensive use of the existence predicate E can be found in the many-sorted partial logic (MPL) of [JKR86].

In the sequel we will assume that the sort ATSPEC contains universal closures of these first order translations of the rules rather than the rules themselves.

5. A model for BMA[rsig-ru]

5. 1. MO(ru), a domain of module objects

The model decribed in this section is of course based on the model rsig-ruof Sig-Atspec that was described above in 4. 5. Just as in [BHK86b] we will construct a model for BMA[rsig-ru], by means of biinterpretations. We need some auxiliary notions first. A <u>closed module object</u> is a triple (Σ, Γ, X) with Σ and Γ signatures, such that $\Sigma \subseteq \Gamma$, and X a finite set of rules over Γ. The intuition behind this notion is as follows. A module object (Σ, Γ, X) describes a relational specification of Σ structures (or theories) which has visible signature Σ, which is a subsignature of its internal signature Γ, over which the rules in X are written. $\Gamma-\Sigma$ contains the hidden relations and constructors of this specification. As the only role of the module is to contain information about Σ structures it is evident from the intuition that a bijective renaming of hidden constructors and relations together with a corresponding renaming of the rules leads to an equivalent module.

MO[ru] is the class of closed module objects in the above sense. The operations $\langle . \rangle \Sigma$, $\cdot$, $\square$, and + can be interpreted on MO[ru], where + turns out to be a partial mapping initially.

$\langle . \rangle$: $\langle \phi \rangle = (\Sigma(\phi), \Sigma(\phi), \{\phi\})$

Σ: $\Sigma(\Sigma, \Gamma, X) = \Sigma$

T: $T(\Sigma) = (\Sigma, \Sigma, \emptyset)$

$\cdot$: $r \cdot (\Sigma, \Gamma, X) = (r \cdot \Sigma, r \cdot \Gamma, r \cdot X)$
where $r \cdot \{\phi_1, \ldots, \phi_n\} = \{r \cdot \phi_1, \ldots, r \cdot \phi_n\}$

$\square$: $\Delta \square (\Sigma, \Gamma, X) = (\Delta \cap \Sigma, \Gamma, X)$

+: $(\Sigma_1, \Gamma_1, X_1) + (\Sigma_2, \Gamma_2, X_2) = (\Sigma_1 + \Sigma_2, \Gamma_1 + \Gamma_2, X_1 + X_2)$
provided that $\Gamma_1 \cap \Gamma_2 = \Sigma_1 \cap \Sigma_2$.

The operator Σ_I defines the internal signature of a module object as follows. $\Sigma_I(\Sigma, \Gamma, X) = \Gamma$. So the condition, for + being defined, can be read as follows, with P, Q ranging over MO[ru]:
$\Sigma_I(P) \cap \Sigma_I(Q) = \Sigma(P) \cap \Sigma(Q)$.

Next we introduce an equivalence relation $\approx$ on MO[ru] with the following properties:

(i) for each pair A, B $\in$ MO[ru] there are A' $\approx$ A and B' $\approx$ B such that A' + B' is defined,

(ii) intuitively A $\approx$ A' implies that A and A' are equivalent module descripions,

(iii) if A $\approx$ A' then

- $\Sigma(A) = \Sigma(A')$
- $\Sigma \square A \approx \Sigma \square A'$
- $r \cdot A \approx r \cdot A'$

(iv) if A + B is defined and A' + B' is defined (with A $\approx$ A' and B $\approx$ B') then A + B $\approx$ A' + B'.

The relation $\approx$ that satisfies these properties is found very simply indeed as follows: let $A = (\Sigma, \Gamma, X)$ and $B = (\Sigma', \Gamma', X')$ then A $\approx$ A' if $\Sigma = \Sigma'$ and there is a bijective signature morphism $\phi : \Gamma \to \Gamma'$ such that:

(i) ϕ is the identity mapping on Σ,

(ii) $\phi(X) = X'$ (the effect of a bijective signature morphism extends in an unproblematic way to sets of rules).

Working on congruence classes modulo $\approx$ the operation + becomes a total one and we obtain a structure for the signature of BMA[rsig-ru]. We will denote this structure with MO(ru). (So ignoring the fact that + is partial

on MO[ru] we may say that MO(ru) = MO[ru]/≈). MO(ru) is a model of the basic modularisation operators for algebraic specifications.

Proposition 5. 1. 1. Let $P = (\Sigma, \Gamma, \{\phi_1, \ldots, \phi_n\}) \in MO(ru)$, then [P] is the interpretation of $cme(P) = \Sigma \square (T(\Gamma) + \langle\phi_1\rangle + \ldots + \langle\phi_n\rangle) \in CME[ru]$.

PROOF. Immediate.

5. 2. The syntactic biinterpretation model, $MO_{sb}(ru)$

Departing from the structure MO(ru) that was outlined above we will describe the syntactic biinterpretation structure $MO_{sb}(ru)$. This structure is obtained as a quotient after defining a congruence on MO(ru), which in turn is generated by a notion of equivalence on module objects that extends the notion of isomorphism which was used in section 3 to obtain MO(ru).

Definition 5. 2. 1. Let Σ_1 and Σ_2 be signatures, a morphism from Σ_1 to Σ_2 is an arity preserving mapping f that maps relation names of Σ_1 into relation names of Σ_2 and constructor names of Σ_1 into constructor names of Σ_2. Morphisms can be total, partial, surjective, injective and bijective.

Definition 5. 2. 2. Let two module objects $A_1 = (\Sigma, \Gamma_1, X_1)$ and $A_2 = (\Sigma, \Gamma_2, X_2)$ be given . An interpretation $f: A_1 \hookrightarrow A_2$ is a partial morphism from Γ_1 to Γ_2 which satisfies the following conditions:

(i) The restriction of f to Σ is the identity mapping. So in particular $\Sigma \subseteq Dom(f)$. (It is allowed however that f maps elements of Γ_1 outside Σ on Σ.)

(ii) for every rule ϕ in X_1, the transformed rule $f(\phi)$ is a member of X_2.

Definition 5. 2. 3. Let A and B be two module objects. We say that A and B are syntactically biinterpretation equivalent if the following three

conditions are satisfied:

(i) $\Sigma(A) = \Sigma(B)$,

(ii) there is an interpretation $f: A \hookrightarrow B$,

(iii) there is an interpretation $g: B \hookrightarrow A$.

We write $A =_{sb} B$ in this case.

Notational convention: we say that (f, g) is a syntactic biinterpretation for the pair (A, B).

Proposition 5. 2. 4.

(i) $=_{sb}$ is an equivalence relation on MO[fol],

(ii) $M \approx M'$ implies $M =_{sb} M'$,

(iii) $=_{sb}$ induces an equivalence relation (also written $=_{sb}$) on the equivalence classes of $\approx$ as follows: $[M] =_{sb} [M']$ if $M =_{sb} M'$,

(iv) $=_{sb}$ is a congruence on MO(ru).

Proof. All of these proofs are quite boring and have been omitted from this text.

Definition 5. 2. 5. The syntactic biinterpretation model is as follows:

$MO_{sb}(ru) = MO(ru)/=_{sb}$.

Theorem 5. 2. 6. $MO_{sb}(ru)$ is a model of BMA[rsig-ru].

Proof. Boring and omitted, see [BHK 86b] for some details on this construction in the case of modularised algebraic specifications.

Theorem 5. 2. 7. In BMA[rsig-ru] each closed module expression can be shown equal to an expression in normal form of the following shape: $\Sigma \, \square \, (\, T(\Gamma) + \langle \phi_1 \rangle + \ldots + \langle \phi_n \rangle)$ with $\Sigma \subseteq \Gamma$ and $\Sigma(\phi_i) \subseteq \Gamma$, for $1 \leq i \leq n$.

Proof. Just as the proof of of the normal form theorem for BMA[fol] in [BHK86a].

6. Conclusions. The objectives of this paper are twofold. First a generalisation of module algebra (as presented in[BMHK86a]) has been made by allowing a modification of the signatures involved. This

flexibility of the formalism regarding signatures is important in view of the fact that in the context of the ESPRIT project METEOR we hope to apply module algebra also on the more complex syntax of MPL (many sorted partial logic) from [JKR86].

Secondly we have presented a modular structure for a simple subsystem of logic programming (see [L84] for a survey of logic programming). Indeed we feel that this subsystem, (which we prefer to call 'relational specifications' rather than logic programming in order to avoid the confusion implied by the procedural interpretation of logic programming) is perhaps an important addition to algebraic specifications. In relational specifications emphasis is on partial constructors and relations rather that on functions and types as is the case in algebraic specifications. By avoiding an equality predicate one need not bother about the existense of four different interpretations of the equality sign in the context of partial operations. This unpleasant problem being removed nothing stands in the way of an interpretation of the constructors as partial functions. The unary relations now play the role of types and again there is no problem concerning subtypes because types may be related in an arbitrary complex way. Just as in the case of algebraic specifications relational specifications admit an initial interpretation (the initial relational structure).

7. References

[BHK86a] J. A. Bergstra, J. Heering, P. Klint, Module algebra, Report, CS-R8617, Department of Software Technology, Centre for Mathematics and Computer Science, Amsterdam 1986.

[BHK86b] J. A. Bergstra, J. Heering, P. Klint, Biinterpretation models for the axioms of module algebra, report CS-R86??, Department of Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1986

[GM84] J. A. Goguen, J. Meseguer, Equality, types and (why not?) generics for logic programming, Journal of Logic programming 2 (1984) pp. 179-210

[JKR86] H. B. M. Jonkers, C. P. J. Koymans, G. R. Renardel de Lavalette, A semantic framework for the COLD-family of languages, Logic

Group Preprint Series No. 9 Department of Philosophy, University of Utrecht, 1986

[L84] J. W. Lloyd, foundations of logic programming, Springer Verlag Berlin, Heidelberg, New York, Tokyo, 1984