RAIRO. INFORMATIQUE THÉORIQUE

MANFRED BROY

MARTIN WIRSING

Algebraic definition of a functional programming language and its semantic models

RAIRO. Informatique théorique, tome 17, nº 2 (1983), p. 137-161

<http://www.numdam.org/item?id=ITA_1983__17_2_137_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 17, n° 2, 1983, p. 137 à 161)

ALGEBRAIC DEFINITION OF A FUNCTIONAL PROGRAMMING LANGUAGE AND ITS SEMANTIC MODELS (*) (**)

by Manfred BROY and Martin WIRSING [1]

Communiqué par J.-F. PERROT

Abstract. — *In the usual framework of abstract types programming languages including a definition mechanism for partial recursive functions cannot be specified sufficiently complete because of the termination problem. Therefore the algebraic concepts of abstract types are extended to partial algebras leading to "total" homomorphisms for partial algebras. In this framework an abstract type is given defining a functional programming language. The category of models of that type can be structured with the help of a partial order induced by the total homomorphisms. This order shows the relationship between the different semantic models and the well-known notions of fixed point theory. Initial and weakly terminal models correspond directly to least fixed points, the subcategories of optimal and maximal models correspond to optimal and maximal fixed points. Finally, strong terminality and initiality of the subcategory of minimally defined models can be connected to mathematical and operational equivalence of recursive functions.*

Résumé. — *En raison du problème de terminaison la notion usuelle des types abstraits ne permet pas de spécifier (d'une manière suffisamment complète) des fonctions partiellement récursives ayant un domaine non récursif. De ce fait nous élargissons les concepts algébriques des types abstraits par la notion d'algèbre partielle et la notion d'homomorphisme entre algèbres partielles. Nous appliquons cette méthode de spécification à l'exemple d'un langage de programmation fonctionnelle. La catégorie des modèles de ce type peut être analysée et structurée à l'aide d'un ordre partiel induit par les homomorphismes faibles. Cet ordre montre les relations entre les différents modèles sémantiques d'un type et les notions bien connues de la théorie des points fixes. Les modèles initiaux et faiblement terminaux correspondent exactement aux plus petits points fixes et les sous-catégories des modèles optimaux et maximaux correspondent aux points fixes optimaux et aux points fixes maximaux. En plus les modèles terminaux de la sous-catégorie des modèles minimalement définis décrivent l'équivalence mathématique entre fonctions récursives pendant que les modèles initiaux de cette sous-catégorie caractérisent une équivalence opérationnelle.*

1. INTRODUCTION

Abstract data types are a uniform, powerful tool for formal specifications. The meaning of such types is generally explained by particular models such as initial or terminal ones, or by the class of all possible models, which may be described by sets of congruences on the term algebra (*cf.* Wirsing, Broy [34], Broy *et al.* [13]).

(*) Received June 1980, revised October 1982.
(**) The work was supported in part by the Sonderforschungsbereich 49 Programmiertechnik, Munich.
[1] Technische Universität München, Institut für Informatik, Arcisstrasse 21, 8000 München 2, R.F.A.

Of course it would be of interest to use the tool of abstract types to specify the semantics of recursive functions being part of functional programming languages thus considering programs as abstract objects of such a type. Unfortunately in the framework of "equationally definable classes" the termination problem of partial recursive functions can not be specified "sufficiently complete", since it is recursively unsolvable.

In particular using only equations as axioms it is impossible to specify sufficiently complete abstract types including models containing partial recursive functions with non-recursive domains. Therefore such types cannot have terminal nor initial models (*cf.* Broy, Wirsing [9]).

To cope with these problems we have to extend the notion and theory of abstract data types. Abstract data types consist of sorts and function symbols — called signature — and of equations. The function symbols generate an algebra of terms, called term algebra. On this term algebra the equations induce congruence relations which may be described by homomorphisms. The homomorphisms induce a lattice structure, a complete partial ordering or at least a partial ordering on the classes of isomorphic structures.

There were numerous efforts to extend the abstract type approach to describe types with models containing functions with nonrecursive domains (*cf.* Wand [32], "rational theories" in ADJ [31] and "algebraic semantics" in Courcelle, Nivat [16]).

These approaches study either least fixed points by assuming continuous interpretations (which is equivalent to the implicit introduction of existential quantifiers) or they consider infinite terms or nonstandard objects (thus extending the carrier set of data which may cause difficulties with structural induction).

In place of that we propose two other extensions of the basic theory:

— an "algebraic" one using partial algebras and homomorphisms for partial algebras instead of total ones;

— a "logical" one using explicitly existential quantifiers in the axioms (*cf.* Broy *et al.* [7]).

In section 2.1 we give the basic notions for the theory of partial algebras. In section 2.2 we discuss partial abstract types, that are classes of partial algebras, and prove characterization and existence theorems for initial and weakly terminal algebras of partial abstract types.

As an important application we define an abstract type MAP in section 3.1. The type MAP describes a functional programming language including an evaluation operator for functional programs, i. e. applications of partial recursive functions.

MAP is not and cannot be sufficiently complete but it is "weakly" sufficiently complete.

The weakly terminal models of MAP accord to least fixed point semantics with the mathematical equality between the functionals. Each model of type MAP can be viewed as a particular semantics of the functional programming language. Two semantic models S_1 and S_2 of MAP can be defined to be extensionally equivalent, if there is a model M of type MAP, such that there are (strong) homomorphisms from both S_1 and S_2 to M. Consequently for a programming language all semantic models are extensionally equivalent, if and only if there exists a strongly terminal model for the corresponding type.

If we consider the class C_M of all models being extensionally equivalent to a given model M, then for every model A in C_M the interpretation of a functional f in A yields the same fixed point of f as the interpretation of f in M. C_M forms a lattice (*cf.* Wirsing, Broy [34]). In particular, for every initial model I the class C_I contains all models which describe least fixed point semantics.

With the help of existential quantifiers we can restrict the type MAP to a type MAP″ such that C_I represents the class of finitely generated models of MAP″. On the other hand, by introducing a specific approximation function for recursively defined objects of sort <u>map</u> we can restrict MAP to a type MAP* such that the weakly terminal models Z^* of MAP* correspond directly to a notion of operational equivalence while the weakly terminal models Z of MAP correspond to mathematical (functional) equivalence. This leads to a proper formal definition of the different notions of equivalence of recursively defined functions.

2. BASIC DEFINITIONS AND RESULTS

2.1. Partial Σ-Algebras

We give briefly the basic notions concerning partial heterogeneous algebras and their connection to abstract data types:

A *partial heterogenous algebra A* is a pair $(\{s^A\}_{s \in S}, \{f^A\}_{f \in F})$ of families where the (countable) index sets S and F are called *sorts* and *functions symbols* and where

– $\{s^A\}_{s \in S}$ is a family of *carrier sets*;

– $\{f^A\}_{f \in F}$ is a family of partial *operations* on the carrier sets, i. e. for every $f \in F$ the function f^A is a partial mapping $f^A : s_1^A \times \ldots \times s_n^A \to s_{n+1}^A$ with some $n \in N$, $s_1, \ldots, s_{n+1} \in S$.

$s_1^A \times \ldots \times s_n^A$ is called *domain* of f^A, s_{n+1}^A is called *range* of f^A and $s_1 \times \ldots \times s_n \to s_{n+1}$ is called *functionality* of f. If $n=0$ then f is called *constant*. The pair of index sets $\Sigma=(S, F)$ is called *signature*. A then is called *Σ-algebra* and we abbreviate $\{s^A\}_{s \in S}$ by S^A and $\{f^A\}_{f \in F}$ by F^A.

A Σ-algebra A' is called *Σ-subalgebra* of a Σ-algebra A, if for all $s \in S : s^{A'} \subseteq s^A$ and for all $f \in F$ $f^{A'}$ is the restriction of f^A to A' and if A' is closed under all the operations of A. For every Σ-algebra A there exists a smallest Σ-subalgebra A_{Gen} of A (with respect to set inclusion). A_{Gen} is the subalgebra of A finitely generated by the constants and function symbols named in Σ. A is called *finitely generated* (or *Σ-structure*) if $A_{\mathrm{Gen}}=A$.

From the function symbols itself we obtain particular Σ-algebras: Let $x_1, \ldots, x_k$ be (free) variables of sort $s_1, \ldots, s_k \in S$. Then $W_\Sigma(x_1, \ldots, x_k)$ is the least set (with respect to set inclusion) M such that $x_1, \ldots, x_k$ and every constant $f \in F$ belong to the appropriate carrier sets of M and, whenever $f: s_1 \times \ldots \times s_n \to s_{n+1} \in F$ and $t_1, \ldots, t_n$ of sort $s_1, \ldots, s_n$ are in M, then $f(t_1, \ldots, t_n)$ belongs to the carrier set s_{n+1}^M of M. The elements of $W_\Sigma(x_1, \ldots, x_k)$ are called terms. $W_\Sigma(x_1, \ldots, x_k)$ is made into a Σ-algebra by defining:

$$f^{W_\Sigma}(t_1, \ldots, t_n) \underset{\mathrm{def}}{=} f(t_1, \ldots, t_n) \qquad \text{for} \quad t_i \in s_i^{W_\Sigma}.$$

This algebra is a total algebra. We denote it also by $W_\Sigma(x_1, \ldots, x_k)$. If $k=0$ we write W_Σ and call it the *term algebra* of Σ. Clearly, W_Σ is finitely generated and Σ-subalgebra of all $W_\Sigma(x_1, \ldots, x_k)$. The elements of W_Σ are called *ground terms*.

For a ground term $t \in W_\Sigma$ and a Σ-algebra A the *interpretation* t^A of t in A is obtained by substituting all symbols f in t by f^A. Then either t^A denotes an element of a carrier set of A or t^A is undefined.

Homomorphisms for partial algebras are simply partial operations which satisfy the usual homomorphism-property on the domain of every function in the signature:

Let A, B be partial heterogeneous Σ-algebras.

A family $\{\varphi_s : s^A \to s^B\}_{s \in S}$ of possibly partial operations is called <u>Σ-homomorphism</u> from A into B (denoted by $\varphi : A \to B$) iff

$$\text{for all } f: s_1 \times \ldots \times s_n \to s \in F \quad \text{and all } x_1 \in s_1^A, \ldots, x_n \in s_n^A:$$

$$f^A(x_1, \ldots, x_n) \text{ defined} \Rightarrow \varphi_s(f^A(x_1, \ldots, x_n)) = f^B(\varphi_{s_1}(x_1), \ldots, \varphi_{s_n}(x_n)).$$

Therefore on total algebras this notion of homomorphism coincides with the usual one. We distinguish different kinds of homomorphisms which will be important in the sequel (*cf.* Grätzer [19], Broy, Wirsing [8] where a slightly different terminology is used).

Total Σ-homomorphisms are called "Σ-homomorphisms" by Grätzer and preserve the definedness of terms. In particular, if a Σ-homomorphism $\varphi : A \to B$ is total, then

$$\text{for all } f: s_1 \times \ldots \times s_n \to s \in F \quad \text{and all } x_1 \in s_1^A, \ldots, x_n \in s_n^A:$$

$$f^A(x_1, \ldots, x_n) \text{ defined } \Rightarrow f^B(\varphi_{s_1}(x_1), \ldots, \varphi_{s_n}(x_n)) \text{ defined.}$$

A total Σ-homomorphism satisfying also the converse condition is called *strong*. A Σ-homomorphism $\varphi : A \to B$ is said to be strong, if it is total and

$$\text{for all } f: s_1 \times \ldots \times s_n \to s \in F \quad \text{and} \quad x_1 \in s_1^A, \ldots, x_n \in s_n^A:$$

$$f^B(\varphi_{s_1}(x_1), \ldots, \varphi_{s_n}(x_n)) \text{ defined } \Rightarrow f^A(x_1, \ldots, x_n) \text{ defined.}$$

PROPOSITION 1: *Let $\varphi : A \to B$ be a Σ-homomorphism between two Σ-structures A und B.*

(1) *φ is total iff for all terms $t \in W_\Sigma$ such that t^A is defined t^B is defined as well and $\varphi(t^A) = t^B$;*

(2) *φ is strong iff for all terms $t \in W_\Sigma$ (t^A is defined $\Leftrightarrow t^B$ is defined) and $\varphi(t^A) = t^B$.*

Proof: (1) φ total implies that

$$f^A(x_1, \ldots, x_n) \text{ defined } \Rightarrow$$
$$\varphi_s(f^A(x_1, \ldots, x_n)) = f^B(\varphi_{s_1}(x_1), \ldots, \varphi_{s_n}(x_n)) \text{ defined}$$

holds for all $f: s_1 \ldots s_n \to s$ and all $x_1 \in s_1^A, \ldots, x_n \in s_n^A$. A simple induction on the length of terms concludes the proof.

(2) is proven analogously. $\square$

Properties of Σ-algebras are expressed by formulas. For simplicity we consider only *conditional universal-existential* formulas, i. e. formulas of the form:

$$\forall s_1 x_1 \ldots \forall s_k x_k \quad \exists s'_1 y_1 \ldots \exists s'_l y_l,$$

$$\bigwedge_{1 \leq i \leq m} [D(p_i) \wedge p_i = q_i] \quad \Rightarrow \quad C,$$

where k, l, $m \geqq 0$ and C is of the form $D(t)$, $\neg D(t)$ or $t = t'$. If $D(p_i)$ can be inferred from $p_i = q_i$ then it will be omitted.

A formula without free identifiers is called *sentence*. The *interpretation* of a formula in a Σ-algebra is defined as usual in a classical two-valued first-order logic. Truth of G in A is denoted by $A \models G$. In particular for ground terms t_1, t_2 we have $A \models D(t_1)$ iff t_1^A is defined, and $A \models t_1 = t_2$ iff t_1 and t_2 are of the same sort and both undefined in A or t_1 and t_2 are both defined and $t_1^A = t_2^A$. Furthermore, $A \models \forall s x : G$ iff for all $a \in s^A$ $A \models G[a/x]$.

The only predicate symbols occurring in formulas are the definedness predicate D and the identity symbol. Since we are working with heterogenous algebras this is not an essential restriction: A predicate $p \subseteqq s_1 \times \ldots \times s_n$ may be seen as boolean function $\bar{p} : s_1 \times \ldots \times s_n \to \underline{\text{bool}}$ where $\underline{\text{bool}}$ denotes the sort $\{\text{true, false}\}$ of the truth-values. Note, however, that there is a difference concerning the notions of homomorphisms between boolean functions and predicates (*cf.* Chang, Keisler [15]).

Due to the strong interpretation of "$=$", $t_1 = t_2$ is always defined and the axioms and rules of inference for classical first order logic (*see* e. g. Chang, Keisler [15]) as well as a structural induction (*cf.* Guttag [20]) can be generalized to types (*cf.* Wirsing, *et al.* [33]). Because of our approach of partial algebras all functions $f : s_1 \times \ldots \times s_n \to s_{n+1}$ are *strict*: For all terms $t_1, \ldots, t_n$ of sort $s_1, \ldots, s_n$ the following axiom holds:

$$D(f(t_1, \ldots, t_n)) \Rightarrow D(t_i) \quad \text{for} \quad i = 1, \ldots, n.$$

Moreover the following "undefined element axiom" holds:

$$\neg D(t_1) \wedge \neg D(t_2) \Rightarrow t_1 = t_2.$$

2.2. Classes of Σ-structures and Abstract Types

An *abstract type* $T = (\Sigma, E)$ consists of a signature Σ and a (countable) set E of sentences, called *axioms*. A subtype (Σ', E') with $\Sigma' \subseteqq \Sigma$, $E' \subseteqq E$ may be designated as *primitive*. Then $T = (\Sigma, E, (\Sigma', E'))$ is called a *hierarchical type*, *cf.* Wirsing *et al.* [33]. This definition can easily be generalized to several primitive subtypes $P_1, \ldots, P_n$ and to hierarchical subtypes. For simplicity, however, we consider in the following only types (Σ, E, P) with a nonhierarchical primitive subtype $P = (\Sigma', E')$; all definitions and properties carry easily over in the general case (*cf.* also Wirsing *et al.* [33]).

Moreover, we assume in the following that Σ' contains the sort $\underline{\text{bool}}$ of truth-values and the constants true, false of sort $\underline{\text{bool}}$.

Let $T = (\Sigma, E, P)$ with $P = (\Sigma', E')$ be a hierarchical type. A term $t \in W_{\Sigma'}(x_1, \ldots, x_n)$ is called primitive. If $t \hat{=} f(t_1, \ldots, t_n)$ where

$f: s_1 \times \ldots \times s_n \to s'$ and $s' \in S'$, then t is called of primitive sort. The only primitive terms of sort bool are assumed to be "true" and "false".

If T is hierarchical and P is the primitive type of T, then a Σ-algebra A is called (*fg-model*) of *type* T, if A is finitely generated, $A \models G$ for all $G \in E$ and the restriction of A to the carrier sets of P satisfies true$\neq$false and is a *fg*-model of P; i. e. these carrier sets must be finitely generable by primitive functions only.

The class of all *fg*-models of a type T is denoted by Gen_T. If $E = \emptyset$ we write Gen_Σ for $\mathrm{Gen}_{(\Sigma, \emptyset)}$. We say that a sentence G is *fg-valid* in T ($T \models G$) if for all $A \in \mathrm{Gen}_T : A \models G$. T is *fg-satisfiable* if $\mathrm{Gen}_T \neq \emptyset$ and T is called *monomorphic* if all elements of Gen_T are isomorphic.

We call a formula G *provable* in T ($T \vdash G$) if G can be deduced from the axioms of T and from the logical axioms and rules including induction. The provability implies *fg*-validity. But the completeness theorem of first-order logic does not hold for all types. If we consider, for instance, a type NAT of natural numbers (*cf.* ADJ [18]) with addition and multiplication then due to the restriction to finitely generated models Gödel's incompleteness theorem applies to NAT i. e. there exists a sentence $\forall \underline{\mathrm{nat}}\, x : G$ such that:

- for all natural numbers n $G[n/x]$ is provable;
- $\forall \underline{\mathrm{nat}}\, x : G$ is *fg*-valid, but not provable.

Let C be a class of Σ-structures. A Σ-structure A of C is called *strongly terminal* in C, iff for all Σ-structures B of type T there is a strong homomorphism $\varphi : B \to A$.

A is called *initial* (*strongly initial resp.*) in C if for all $B \in C$ there exists a unique total (strong resp.) Σ-homomorphism $\varphi : A \to B$.

In fixed point theory the semantics of a recursive definition is determined by the least fixed point fulfilling the recursive equation. According to this one class of models is of particular interest: For a class C of Σ-algebras the class $M\mathrm{def}(C)$ of minimally defined models is given by:

$$M\mathrm{def}(C) \underset{\mathrm{def}}{=} \{ A \in C : \forall t \in W_\Sigma : A \models D(t) \Leftrightarrow C \models D(t) \}.$$

A Σ-algebra Z in a class C of Σ-algebras is called *weakly terminal in* C if it is strongly terminal in $M\mathrm{def}(C)$.

An algebra $A \in \mathrm{Gen}_T$ is called *initial in* T if it is initial in Gen_T, and *weakly terminal*, if it is strongly terminal in $M\mathrm{def}(\mathrm{Gen}_T)$.

Analogously A is called strongly initial and strongly terminal resp. if it is so in Gen_T. Initial and weakly terminal algebras can be characterized as follows:

PROPOSITION 2: *Let A be a model of T. Then:*

(*a*) *A is initial in T iff:*

(1) *A is minimally defined, i.e. for all $t \in W_\Sigma$: $A \models D(t) \Leftrightarrow T \vdash D(t)$ and*

(2) *for all t, $t' \in W_\Sigma$ of the same sort:*

$$A \models t' = t \quad \Leftrightarrow \quad (T^* \nvdash D(t) \quad \text{and} \quad T^* \vdash D(t')) \text{ or } T \nvdash t' = t.$$

(*b*) *Let the primitive type P of T be monomorphic and the terms p_i, q_i in the antecedenses of the axioms be of primitive sort. Then A is weakly terminal in T iff:*

(1) *A is minimally defined, i.e.:*
for all $t \in W_\Sigma$: $A \models D(t) \Leftrightarrow T \vdash D(t)$ and

(2) *for all t, $t' \in W_\Sigma$ of the same sort:*

$$A \models t = t' \quad \Leftrightarrow \quad \forall K(x) \in W(\Sigma, x) \text{ of primitive sort:}$$
$$A \models K[t/x] = K[t'/x].$$

Proof: Proposition 1 and 2 of Broy, Wirsing [11]. □

In order to prove the existence of initial and weakly terminal algebras we define two further notions:

A type T is called *weakly sufficiently complete* iff for every ground term t of primitive sort such that $D(t)$ is provable there exists a primitive term p such that $t = p$ is provable. If either $\neg D(t)$ is provable or such a primitive term p exists, then T is called *sufficiently complete.*

An axiom $\forall s_1 x_1 \ldots \forall s_k x_k \exists s'_1 y_1 \ldots \exists s'_l y_l : G$ where G is quantifier-free is called *uniform in T*, if for all ground terms $t_1, \ldots, t_k$ such that $D(t_1), \ldots, D(t_k)$ is provable there exist ground terms $t'_1, \ldots, t'_l$ such that:

$$D(t'_1) \wedge \ldots \wedge D(t'_l) \wedge G[t_1/x_1, \ldots, t_n/x_n, t'_1/y_1, \ldots, t'_l/y_l]$$

is valid in T.

THEOREM 1: *Let T be a satisfiable, weakly sufficiently complete type the primitive subtype of which is monomorphic.*

If all nonprimitive axioms of T are uniform with antecedenses p_i, q_i of primitive sort, then T has an initial and a weakly terminal algebra.

Proof: For axioms without existential quantifiers apply corollaries 1 and 4 of Broy, Wirsing [11]. The case of axioms with existential quantifiers reduces to the one without existential quantifiers since the uniformity implies that every axiom containing existential quantifiers can be replaced by (infinitely many) axioms without existential quantifiers. □

3. APPLICATIONS

A programming language can be described by an abstract type in the following way:

– The context-free syntax corresponds to the term algebra of the type's signature. The sorts denote the syntactic entities.

– The context-conditions correspond to particular definedness predicates specifying that certain terms are not defined.

– The semantics of the language is given by the equational axioms.

In the following we design a simple functional programming language as a hierarchical type. Starting with a basic "primitive" data type (including $\underline{\text{bool}}$, $\underline{\text{nat}}$, etc.) we define a type MAP containing (besides the sorts of the basic type) a sort $\underline{\text{map(ping)}}$ the terms of which represent the programs of a functional programming language (*cf.* Backus [1]).

3.1. The type MAP

As primitive sorts for the data type MAP we use a sort $\underline{\text{data}}$ and a sort $\underline{\text{id}}$ of identifiers (for functions). For simplicity we assume that $\underline{\text{data}}$ comprises the sort $\underline{\text{bool}}$ of the truth-values "true" and "false". In analogy to the classical theory of recursive functions (*cf.* Shoenfield [30]), we define recursive functions relatively to a given family $\{f_i\}_{i \in I}$ of primitive functions on $\underline{\text{data}}$. For simplicity we assume, that the sorts $\underline{\text{id}}$ and $\underline{\text{data}}$ and the functions $\{f_i\}_{i \in I}$ are specified by some monomorphic types ID and DATA, i. e. by types having only isomorphic models.

The terms (objects) of sort $\underline{\text{map}}$ are functions constructed from the primitive functions $\{f_i\}_{i \in I}$, selector functions s_j, and the functionals fcond for conditionals, comp for functional composition and def for the recursive definition of functions. As in Backus [1] no formal arguments are needed. For every "functional program" f, i.e. for every term f of sort <u>map</u>, an apply-operation defines the application of f to a tuple of objects of sort <u>data</u>. The laws (A1)-(A6) for apply simply describe the behavior of a (call-by-value-) text substitution machine.

The function sub is used as an auxiliary (hidden) function, where sub(i, f, g) replaces all free occurrences of i in f by g. The equations (S) for sub are the usual ones known from the substitution operation in the λ-calculus.

The function occurs is used as a "syntactic" auxiliary (hidden) function for formulating the axioms of local renaming (α-reduction).

<u>type</u> MAP ≡

<u>primitives</u>: ID, DATA,

<u>sort</u>: <u>map</u>,

<u>functions</u>:

<u>map</u> f'_i,	for the primitive functions $f_i \in F_{DATA}$, $i \in I$
<u>map</u> s_j,	selector function $j \in N \setminus \{0\}$
(<u>id</u>) <u>map</u> fid,	interpretation of identifiers as functions
(<u>id</u>, <u>map</u>) <u>map</u> def,	(recursive) function declaration
(<u>map</u>, <u>map</u>, <u>map</u>) <u>map</u> fcond,	conditional functional
(<u>map</u>, ..., <u>map</u>) <u>map</u> comp,	composition of functions
(<u>id</u>, <u>map</u>, <u>map</u>) <u>map</u> sub,	substitution operator
(<u>map</u>, <u>data</u>, ..., <u>data</u>) <u>data</u> apply,	apply-operator
(<u>id</u>, <u>map</u>) <u>bool</u> occurs,	free occurrence of identifiers,

<u>laws</u>

$$\text{occurs}(i, f'_k) = \text{false},$$
$$\text{occurs}(i, s_k) = \text{false},$$
$$\text{occurs}(i, \text{fid}(j)) = (i = j),$$
$$\text{occurs}(i, \text{comp}(g_0, \ldots, g_n)) = \bigvee_{0 \le k \le n} \text{occurs}(i, g_k),$$
$$\text{occurs}(i, \text{fcond}(p, h, g)) = \text{occurs}(i, p) \vee \text{occurs}(i, h) \vee \text{occurs}(i, g),$$
$$\text{occurs}(i, \text{def}(j, g)) = (i \ne j \wedge \text{occurs}(i, g)))$$

$$(S)\left\{\begin{array}{l}
\text{sub}(i, f'_j, x) = f'_j, \\
\text{sub}(i, s_j, x) = s_j, \\
\text{sub}(i, \text{fid}(k), x) = \underline{\text{if}}\ i = k\ \underline{\text{then}}\ x \\
\qquad\qquad \underline{\text{else}}\ \text{fid}(k)\ \underline{\text{fi}}, \\
\neg\, \text{occurs}(i, g) \Rightarrow \text{def}(j, g) = \text{def}(i, \text{sub}(j, g, \text{fid}(i))), \\
\neg\, \text{occurs}(k, x) \Rightarrow \text{sub}(i, \text{def}(k, g), x) = \underline{\text{if}}\ i = k\ \underline{\text{then}}\ \text{def}(i, g) \\
\qquad\qquad \underline{\text{else}}\ \text{def}(k, \text{sub}(i, g, x))\ \underline{\text{fi}}, \\
\text{sub}(i, \text{fcond}(p, g, h), x) = \text{fcond}(\text{sub}(i, p, x), \text{sub}(i, g, x), \text{sub}(i, h, x)), \\
\text{sub}(i, \text{comp}(g_0, \ldots, g_m), x) = \text{comp}(\text{sub}(i, g_0, x), \ldots, \text{sub}(i, g_m, x)),
\end{array}\right.$$

$$(A1)\quad \text{apply}(f'_j, d_1, \ldots, d_n) = \begin{cases} f_j(d_1, \ldots, d_n) & \text{if } f_j \text{ is a } n\text{-ary primitive function} \\ \underline{\text{error}} & \text{otherwise} \end{cases}$$

(A2) $\text{apply}(s_j, d_1, \ldots, d_n) = \begin{cases} d_j & \text{if } 1 \leq j \leq n \\ \underline{\text{error}} & \text{otherwise} \end{cases}$

(A3) $\text{apply}(\text{fid}(i), \quad d_1, \ldots, d_n) = \underline{\text{error}},$

(A4) $\text{apply}(\text{def}(i, g), d_1, \ldots, d_n) = \text{apply}(\text{sub}(i, g, \text{def}(i, g)), d_1, \ldots, d_n),$

(A5) $\text{apply}(p, d_1, \ldots, d_n) = \text{true} \Rightarrow \text{apply}(\text{fcond}(p, g, h), d_1, \ldots, d_n) = \text{apply}(g, d_1, \ldots, d_n),$

(A5a) $\text{apply}(p, d_1, \ldots, d_n) = \text{false} \Rightarrow \text{apply}(\text{fcond}(p, g, h), d_1, \ldots, d_n) = \text{apply}(h, d_1, \ldots, d_n),$

(A5c) $\text{apply}(p, d_1, \ldots, d_n) \neq \text{true} \wedge \text{apply}(p, d_1, \ldots, d_n) \neq \text{false} \Rightarrow$

$$\text{apply}(\text{fcond}(p, g, h), d_1, \ldots, d_n) = \underline{\text{error}},$$

(A6) $\text{apply}(\text{comp}(g_0, \ldots, g_m), d_1, \ldots, d_n) =$

$$\text{apply}(g_0, \text{apply}(g_1, d_1, \ldots, d_n), \ldots, \text{apply}(g_m, d_1, \ldots, d_n)),$$

Note that for a nullary primitive function f, apply $(\text{comp}(f')) = \text{apply}(f')$

<u>end of type</u>

The function occurs is specified sufficiently completely. Every term occurs(i, x) can be reduced either to true or to false, i. e. occurs is a total function in every structure of type MAP.

The equation $t = \underline{\text{error}}$ is an abbreviation for $\neg D(t)$ and indicates, that the interpretation of the term t is undefined in every model. For the sake of simplicity we omit subscripts in the function(scheme)s comp and apply (giving the number of parameters) and give the respective conditions for well-formedness (the context conditions) only verbally.

Based on a suitable abstract type for natural numbers the factorial function can be specified as follows:

$\text{def}(\text{fac}, \text{fcond}(\text{eq0}', 1', \text{comp}(*', s_1, \text{comp}(\text{fid}(\text{fac}), -1'))))$

where eq0 denotes the test for zero and -1 the predecessor operation.

Note that we did not introduce an explicit tupling operator like:

$(\underline{\text{map}}, \ldots, \underline{\text{map}})\, \underline{\text{map}}\, [., \ldots, .]$

as a constructor function to generate objects of sort <u>map</u> with tuples of sort <u>data</u> as results (*cf.* [1]).

Of course such a tupling operator with:

$\text{apply}([g_1, \ldots, g_k], d_1, \ldots, d_n) =$

$$\text{apply}(g_1, d_1, \ldots, d_n) \,\&\, \ldots \,\&\, \text{apply}(g_k, d_1, \ldots, d_n)$$

could well be specified if the range of the apply operation is extended to sequences ("&" denotes the concatenation of sequences). The resp. axioms for occurs, sub and fcond are obvious. We tried to avoid to do so because we did not want to introduce any additional structure on the domains and ranges.

In fact the tupling-operator is implicitly contained in our particular composition operator, which actually is a specific combination of the composition operator "." (for binary functions) and the tupling operator:

$\text{comp}(g_0, \ldots, g_k) = [g_1, \ldots, g_k] . g_0.$

In fact, a slightly syntactically sugared version of the abstract syntax given by the signature of type MAP could be taken as a small functional programming language:

Denote

fid (i)	by i	
fcond (p, g, h)	by $\underline{\text{if}}\ p \to g, h\ \underline{\text{fi}}$	
comp (g_0, g_1)	by $g_1 . g_0$	
comp $(g_0, g_1, \ldots, g_n)$	by $[g_1, \ldots, g_n] . g_0$	for $n \geqq 2$
def (i, g)	by $i :: g$	

Example:

(1) Factorial Function:

$\text{fac} :: \underline{\text{if}}\ \text{eq}0' \to 1', [s_1, -1'.\text{fac}].*'\ \underline{\text{fi}}$

(2) Ackermann Function:

$$\begin{aligned}\text{ack} :: \underline{\text{if}}\ & s_1.\text{eq}0' \to s_2.+1', \\ & \underline{\text{if}}\ s_2.\text{eq}0' \to [s_1.-1', 1'].\text{ack}, \\ & \qquad [s_1.-1', [s_1, s_1.-1'].\text{ack}].\text{ack} \\ & \underline{\text{fi}} \\ \underline{\text{fi}} &\end{aligned}$$

As can be seen immediately, using the axioms (S), every term t of sort $\underline{\text{map}}$ containing the substitution operator sub can be reduced to a term not containing sub. The function sub is only used as an auxiliary function for defining the axioms for the function apply.

Due to the strictness of the operations each term t of sort $\underline{\text{map}}$ must be defined in all models (which follows from the definition of occurs, since occurs is total).

Furthermore, for all terms t of sort map the term apply $(t, d_1, \ldots, d_n)$ can be reduced by the rules in (A) and (S) resp. But not every reduction yields a unique or defined result.

For example if f is the (trivial) recursive definition def$(g, \mathrm{fid}(g))$ we obtain

$$\mathrm{apply}(\mathrm{def}(g, \mathrm{fid}(g)), d_1, \ldots, d_n) \overset{(A4)}{=}$$
$$\mathrm{apply}(\mathrm{sub}(g, \mathrm{fid}(g), \mathrm{def}(g, \mathrm{fid}(g))), d_1, \ldots, d_n) \overset{(S)}{=}$$
$$\mathrm{apply}(\mathrm{def}(g, \mathrm{fid}(g)), d_1, \ldots, d_n).$$

Since no other axioms are applicable to these terms, the value of them is not fixed in MAP. For every primitive term d of sort $\underline{\mathrm{data}}$ there exists a fg-model A_d of MAP with $\mathrm{apply}(\mathrm{def}(g, \mathrm{fid}(g)), d_1, \ldots, d_n)=d$ in A_d. Thus suppose MAP has a strongly initial model. Then in any such model of MAP $\mathrm{apply}(\mathrm{def}(g, \mathrm{fid}(g)), d_1, \ldots, d_n)$ must be different from any primitive term d of sort $\underline{\mathrm{data}}$. Analogously in any strongly terminal fg-model it would be equal to any d, i. e. $\underline{\mathrm{data}}^A$ could contain at most one element. Consequently MAP cannot have any strongly terminal fg-model and MAP cannot have any strongly initial fg-model which is a persistent enrichment (*cf.* ADJ [18]) of (initial or terminal) fg-models of DATA. MAP is neither sufficiently complete nor persistent.

3.2. Partial initial and weakly terminal models of the type MAP

Now we give two particular models for the type MAP: A term-model I and a model Z based on functional abstraction.

The model I is defined by a quotient structure on the term algebra W_{MAP}, where $\equiv_I$ is defined for the terms of sort $\underline{\mathrm{map}}$ by

$t1 \equiv_I t2$ iff $\tilde{t}1 \hat{=} \tilde{t}2$ where $\tilde{t}1$ and $\tilde{t}2$ originate from $t1$ and $t2$ resp. by eliminating all occurrences of the function subst and by local renaming of all identifiers such that the local identifiers in $\tilde{t}1$ and $\tilde{t}2$ appear in some fixed linear order, i. e. $\mathrm{MAP} \vdash t1 = t2$.

According to this definition all functions with range $\underline{\mathrm{map}}^I$ are simple term-constructor functions apart from the function sub^I, which can always be eliminated. The values of the function occurs^I are uniquely determined, since occurs is defined sufficiently complete.

So it remains to define the function apply^I

$$\mathrm{apply}^I(f^I, x_1, \ldots, x_n) = \begin{cases} y^I & \text{if } \exists y \in W_{\mathrm{DATA}} : \mathrm{MAP} \vdash \mathrm{apply}(f, x_1, \ldots, x_n) = y \\ \text{undefined} & \text{otherwise} \end{cases}$$

As is well-known from λ-calculus fully abstract (*cf.* Milner [25]) models are much harder to define. To avoid the technical details for the introduction of *environment* used to cope with terms f of sort $\underline{\text{map}}$ for which some object i of sort $\underline{\text{id}}$ the predicate occurs(i, f) yields true we restrict ourselves to give a semantic representation for "closed" objects of sort $\underline{\text{map}}$, i. e. for objects f such that $\forall\, \underline{\text{id}}\, i$: occurs(i, f) = false. In this case an object f of sort $\underline{\text{map}}$ can be represented by a family of functions $(\underline{\text{data}}^Z)^n \to \underline{\text{data}}^Z$, i. e.

$$f^Z = \{\lambda x_1, \ldots, x_n : \text{apply}^I(f^I, x_1, \ldots, x_n)\}_{n \in N}.$$

With this definition the application of such a semantic object f^Z to n arguments of sort $\underline{\text{data}}^Z$ is simply defined by applying the n-ary function form the family f^Z to the arguments.

PROPOSITION 3: *I and Z (appropriately extended to "nonclosed" objects) are models of* MAP.

Proof: By checking the validity of the axioms of MAP. □

These two models show that the type MAP is satisfiable. MAP is also weakly sufficiently complete:

LEMMA 1: *The type* MAP *is weakly sufficiently complete.*

Proof: A simple structural induction shows that the operation occurs is sufficiently completely defined.

By changing axiom (A6) into the following essentially equivalent one

$$\text{(A6}') \ \text{apply}(g_1, d_1, \ldots, d_n) = y_1 \wedge \ldots \wedge \text{apply}(g_m, d_1, \ldots, d_n) = y_m \Rightarrow$$

$$\text{apply}(\text{comp}(g_0, \ldots, g_m), d_1, \ldots, d_n) = \text{apply}(g_0, y_1, \ldots, y_m)$$

all axioms involving apply have conclusions of the form:

(i) apply(t) = apply(t') or apply$(t) = t'$ or

(ii) apply(t) = $\underline{\text{error}}$

where t and t' do not contain any nonprimitive function symbol with range in a primitive sort. Terms on the left-hand side of (ii) satisfy trivially the weak sufficient completeness whereas for terms on the left-hand side of (i) we can apply corollary 2 of Broy, Wirsing [11].

The sufficient completeness of occurs together with the weak sufficient completeness of apply yield the weak sufficient completeness of MAP. □

THEOREM 2: MAP *has initial and weakly terminal models.*

Proof: Due to proposition 3 and lemma 1 MAP is satisfiable and weakly sufficiently complete. The primitive types ID and DATA are assumed to be monomorphic. Every antecedens $p=q$ of an axiom is of primitive sort and $D(p)$ can be inferred (since q is of the form true, false or is a variable). Thus according to theorem 1 in section 2.4 MAP has initial and weakly terminal models. □

In particular, I is an initial and Z is a weakly terminal model.

THEOREM 3: *I is an initial and Z is a weakly terminal model of* MAP.

Proof: I and Z are minimally defined. Thus proposition 2(a) together with the definition of I implies immediately that I is an initial model of MAP.

The definition of Z implies that for closed ground terms f and g of sort $\underline{\text{map}}$:

$$Z \models f=g \text{ iff } \forall \underline{\text{data}}\, x_1, \ldots, x_n:$$

$$Z \models \text{apply}(f, x_1, \ldots, x_n)=\text{apply}(g, x_1, \ldots, x_n).$$

Hence according to proposition 2(b) the congruence of Z is coarser than the one of the weakly terminal models. Hence since Z as well as the weakly terminal models are minimally defined there exists a strong homomorphism from every weakly terminal model onto Z. Then the definition of weak terminality implies that this homomorphism is an isomorphism. Therefore Z is weakly terminal. □

3.3. Comparing the models of MAP

In the following let us fix two single models ID and D of ID and DATA (as subalgebras) for all models of MAP. This is possible since ID and DATA are assumed to be monomorphic.

Then we can introduce the following quasi-ordering $\sqsubseteq$ for two models A and B of type MAP:

$A \sqsubseteq B$ (A is *extensionally weaker* than B) if for all ground terms t of primitive sort:

$$[A \models D(t)] \quad \Rightarrow \quad t^A=t^B.$$

This quasi-ordering induces an equivalence relation $\sim$:

$$A \sim B \ (A \textit{ extensionally equivalent} \text{ to } B) \text{ if}$$

$$A \sqsubseteq B \quad \text{and} \quad B \sqsubseteq A$$

By C_A we denote the class of all models B of MAP with $B \sim A$.

Note: In Broy, Wirsing [11] and Broy, Pair, Wirsing [13] $A \models D(t) \Leftrightarrow B \models D(t)$ for all ground terms t is also required to have extensional equivalence. This property can be deduced for extensional equivalent models of type MAP since all nonprimitive terms are defined.

PROPOSITION 4: *The class* $M\mathrm{def}(\mathrm{Gen}_{\mathrm{MAP}})$ *of all minimally defined algebras of* MAP *is a class of extensional equivalence. In particular, the initial model I and the weakly terminal model Z are extensionally equivalent.*

Proof: Obvious. □

On the level of terms the quasi-ordering $\sqsubseteq$ corresponds to the "less defined"-ordering in Manna [23]:

Let f and g be two ground terms of sort $\underline{\mathrm{map}}$.

$$f^A \sqsubseteq g^B \underset{\mathrm{def}}{\Leftrightarrow} \text{for all contexts } K(x) \in W_{\mathrm{MAP}}(x) \text{ of sort } \underline{\mathrm{map}}$$

$$\text{and all } d_1, \ldots, d_n \in W_{\underline{\mathrm{DATA}}}$$

$$[A \models D(\mathrm{apply}(K[f/x], d_1, \ldots, d_n)] \Rightarrow$$

$$\mathrm{apply}(K[f/x], d_1, \ldots, d_n)^A = \mathrm{apply}(K[g/x], d_1, \ldots, d_n)^B$$

For closed objects f and g of sort $\underline{\mathrm{map}}$ this is equivalent to:

$$f^A \sqsubseteq g^B \Leftrightarrow \text{for all } d_1, \ldots, d_n \in W_{\underline{\mathrm{DATA}}}:$$

$$[A \models D(\mathrm{apply}(f, d_1, \ldots, d_n))] \Rightarrow$$

$$\mathrm{apply}(f, d_1, \ldots, d_n)^A = \mathrm{apply}(g, d_1, \ldots, d_n)^B.$$

Defining $f^A \sim g^B$ by $f^A \sqsubseteq g^B \wedge g^B \sqsubseteq f^A$, we can analyse the classes C_A:

PROPOSITION 5:

(1) *For all* $B, B' \in C_A$ *and every closed ground term* f *of sort* map $f^B \sim f^{B'}$.

(2) *Every total homomorphism* $\varphi : B \to B'$ *for* $B, B' \in C_A$ *is strong.*

Proof: (1) Obvious. (2) For all nonprimitive ground terms $t \in W_{\underline{\mathrm{map}}}$ $\mathrm{MAP} \vdash D(t)$. Together with the extensional equivalence of B and B' this implies $B \models D(t) \Leftrightarrow B' \models D(t)$ for all ground terms t.

Thus according to proposition 1 every homomorphism $\varphi : B \to B'$ is strong. □

THEOREM 4: (1) *There exists a strongly initial fg-model* I_A *in* C_A *such that the equality in* $\underline{\mathrm{map}}$ *is nearly the syntactic equality:*

$I_A \models f = g$ iff $f \overset{(S)}{=} g$, *where* $\overset{(S)}{=}$ *denotes the congruence relation induced by* (S).

(2) *There exists a strongly terminal fg-model Z_A in C_A such that for all terms f, g of sort* $\underline{\text{map}}$ *without free identifiers:*

$Z_A \models f = g$ iff $f^{Z_A} \sim g^{Z_A}$.

(3) *The congruences $\sim_B$ associated to the elements $B \in C_A$ form a complete lattice with respect to the following ordering $\subseteqq$:*

$\sim_B \subseteqq \sim_{B'} \underset{\text{def}}{\Leftrightarrow}$ *there exists a strong homomorphism $\varphi : B' \to B$;*

$\sim_{I_A}$ *is the maximum and $\sim_{Z_A}$ the minimum of this lattice.*

Proof: (1) If terms g of sort $\underline{\text{map}}$ contain no applications of "sub", then no further equivalence is induced by the axioms. For every B in C_A and every term t, $h(t^{I_A}) = t^B$ defines a homomorphism.

(2) The $\sim$-equivalence is an equivalence relation on the term algebra of MAP and compatible with the laws of type MAP. Hence the term algebra modulo $\sim$ is a model of MAP. For every B in C_A and every term t, $h(t^B) = t^{Z_A}$ defines a strong homomorphism.

(3) Theorem 5 of Broy, Pair, Wirsing [13]. □

Corollary: *The initial model I of* MAP *is strongly initial and the weakly terminal model is strongly terminal in the class* $C_I = C_Z = M\text{def}(\text{Gen}_{\text{MAP}})$. □

Therefore the equality for terms of sort $\underline{\text{map}}$ in I_A is decidable (provided the equality between identifiers is decidable), whereas the one in Z_A is not even recursively enumerable. Due to the termination problem for "apply" no model of MAP will be computable in the sense of Bergstra, Tucker [4]. The strongly terminal *fg*-models of every C_A provide a fully abstract semantics for MAP in the sense of Milner [25].

The reduction:

def $(i, g) = \text{sub}(i, g, \text{def}(i, g))$,

holds in the strongly terminal algebras of C_A but not in the strongly initial ones. To compare the classes C_A of extensional equivalence we define:

$C_A \sqsubseteq C_B \underset{\text{def}}{\Leftrightarrow}$ for all $M \in C_A$, $\exists M' \in C_B : M \sqsubseteq M'$.

C_A is *optimal* $\underset{\text{def}}{\Leftrightarrow}$ C_A is the maximum of all classes C_M which are

extensionally less defined than all maximal classes i. e.
$\forall$ maximal $C_B : C_A \sqsubseteq C_B$ and

$\forall C_M : (\forall$ maximal $C_B : C_M \sqsubseteq C_B) \Rightarrow C_M \sqsubseteq C_A$.

THEOREM 5: *The extensional equivalence classes of type* MAP *form a semilattice w.r.t.* $\sqsubseteq$ *with an optimal element and where the class* C_Z *of the weakly terminal model* Z *of* MAP *is the least element.*

Proof: Analogously to the proof of lemma 1 and theorem 2 one can prove the assumptions of theorem 7 in Broy, Pair, Wirsing [13]. Thus the classes of extensional equivalence form a semilattice. To get an optimal element we have only to take the greatest lower bound of all maximal elements. □

3.4. Fixed points and functionals

In analogy to the theory of recursive functions a *functional* is an element of $W_{\mathrm{MAP}}(x)$ of sort $\underline{\mathrm{map}}$ where x denotes a variable of sort $\underline{\mathrm{map}}$.

An object f^A of sort $\underline{\mathrm{map}}$ is called *A-fixed point* of the functional $\tau[x]$ iff $f^A \sim \tau^A[f^A]$, here A is an arbitrary model of type MAP.

This definition is justified by the following properties:

LEMMA 2: (1) $\mathrm{def}(i, g)^A$ *is an A-fixed point of* $\mathrm{sub}(g, i, .)$ *for every model* A, *i. e.* $\mathrm{def}(i, g)^A \sim \mathrm{sub}(i, g, \mathrm{def}(i, g))^A$ *by* (A4).

(2) *If* f^A *is an A-fixed point of* $\mathrm{sub}(i, g, .)$ *then there exists a model* B *of type* MAP *such that* $f^A \sim \mathrm{def}(i, g)^B$. □

According to the definitions of Manna, Shamir [24] we call a fixed point f^A of the functional $\mathrm{sub}(i, g, .)$ (where f is a ground term of sort $\underline{\mathrm{map}}$ and A a model of MAP):

- $\underline{\text{least fixed point}}$, if for all fixed points h^B of $\mathrm{sub}(i, g, .) : f^A \sqsubseteq h^B$;
- $\underline{\text{maximal fixed point}}$, if for all fixed points h^B of $\mathrm{sub}(i, g, .)$ $f^A \sqsubseteq h^B \Rightarrow f^A \sim h^B$;
- $\underline{\text{optimal fixed point}}$, if for all maximal fixed points h^B of $\mathrm{sub}(i, g, .)$ $f^A \sqsubseteq h^B$ and for all fixed points f'^C with $f'^C \sqsubseteq h^B$ for all maximal fixed points h^B we have $f'^C \sqsubseteq f^A$.

Then we obtain the following connection to the ordering on model classes introduced.

PROPOSITION 6: *Let* C_{opt} *be an optimal and* $C_{\max}$ *be a maximal class of* MAP. *Then for every model* $L \in C_{M\mathrm{def}\,(\mathrm{Gen}_{\mathrm{MAP}})}$, $O \in C_{\mathrm{opt}}$ *and* $M \in C_{\max}$:

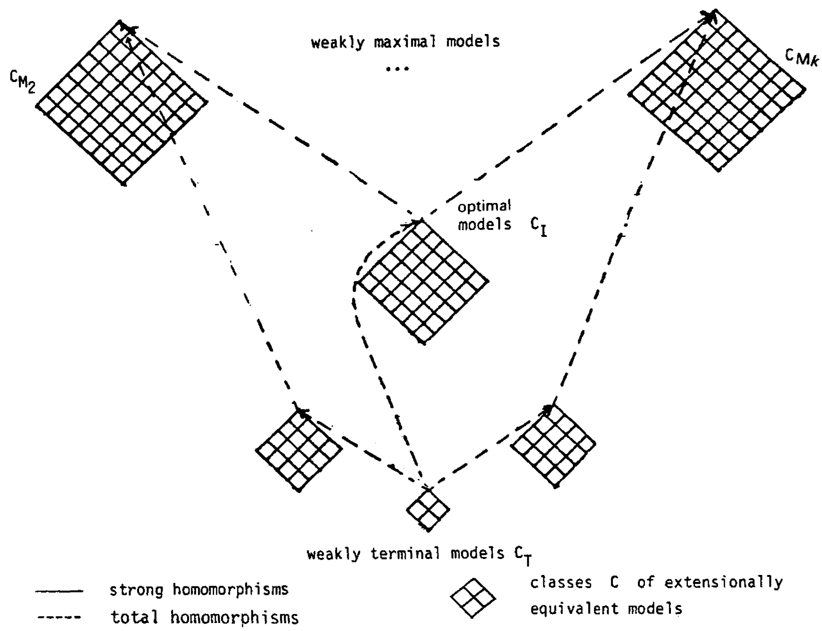
(1) $\mathrm{def}(i, g)^L$ *is a least fixed point;*

(2) $\mathrm{def}(i, g)^O$ *is an optimal fixed point;*

(3) $\mathrm{def}(i, g)^M$ *is a maximal fixed point*

of the functional $\mathrm{sub}(i, g, .)$.

The theorems give us a good impression of the structure of the category of models of type MAP. This structure may be illustrated by the following figure:



Call-by-value, call-by-name

All models A of type MAP correspond to call-by-value semantics, since our functions (especially the selector functions) are assumed to be strict (*cf.* de Bakker [3], de Roever [28]).

3.5. Provability and computability: operational semantics

So far we have payed much attention to the extensional equivalence. In the category C_M the extensional equivalence corresponds to the equality of the strongly terminal *fg*-models in C_M. Now we want to discuss the other *fg*-models in C_M. To do this we restrict ourselves to C_Z, where Z is a weakly terminal *fg*-model of type MAP. For every *fg*-model A in C_Z and every closed ground term t of sort $\underline{\text{map}}$ a term apply$(t, d_1, \ldots, d_n)$ is defined if and only if there is a primitive term r of sort $\underline{\text{data}}$ such that apply$(t, d_1, \ldots, d_n)=r$ is provable in type MAP. Thus we may equate computability with provability. For every term.

(*) apply $(t, d_1, \ldots, d_n)$ with primitive terms $d_1, \ldots, d_n$

with a ground t of sort $\underline{\text{map}}$ in which an application of sub does not occur only one nonprimitive axiom is applicable. If sub occurs in t, the laws (S) always allow to transform t uniquely (modulo renaming) in an equivalent term t' such that $t = t'$ is provable, where sub does not occur in t'. If we assume left-to-right evaluation of apply in the axiom (A6) concerning the function "comp", then for every term (*) a deduction sequence is uniquely determined. If the generated sequence is infinite, then (*) is not defined in all models of C_Z.

C_Z corresponds to least fixed point semantics. The strongly terminal *fg*-models of C_Z specify the mathematical equality between two (relatively) partial recursive functions whereas the strongly initial models I_Z of C_Z correspond to the syntactic equality (*cf.* the first lemma and the theorem of the last section II.2), i. e. two programs f, g of sort $\underline{\text{map}}$ are equal in I_Z iff they have the same Gödel-Number (*cf.* Rogers [29]). Considering the termination problem for apply we see that every *fg*-model in C_Z is cosemicomputable (*cf.* Bergstra, Tucker [4]).

In the strongly initial *fg*-model I in C_Z the equality of two objects of $\underline{\text{map}}$ corresponds exactly to operational equivalence, if we consider only terms t where no free identifiers occur, i. e. $\forall\, \underline{\text{id}}\, i : \neg\, \text{occurs}(i, t)$ holds (programs with free identifiers are excluded generally by context conditions in the formal definition of programming languages).

Using axioms with existential quantifiers one can specify a type MAP'' the *fg*-models of which belong to C_Z. We enrich type MAP by two further functions:

$\underline{\text{funct}}\ \underline{\text{map}}\ \Omega,$

$\underline{\text{funct}}\ (\underline{\text{id}}, \underline{\text{map}}, \underline{\text{nat}})\ \underline{\text{map}}\ \text{iter},$

with the laws (here $\underline{\text{nat}}$ denotes the sort of natural numbers):

$$\text{sub}(i, \Omega, x) = \Omega,$$
$$\text{occurs}(i, \Omega) = \text{false},$$
$$\text{apply}(\Omega, d_1, \ldots, d_n) = \underline{\text{error}},$$
$$\text{iter}(i, g, 0) = \Omega,$$
$$\text{iter}(i, g, m+1) = \text{sub}(i, g, \text{iter}(i, g, m)).$$

Then we can define approximations for objects def(i, g) by iter(i, g, n) with $n \in N$. It is trivial to show that for every model A of MAP″:

$$\text{iter}(i, g, n)^A \sqsubseteq \text{iter}(i, g, n+1)^A$$

and that for all $i, g, m, d_1, \ldots, d_n$ there exists some term $d \in W_{\text{DATA}} \cup \{\underline{\text{error}}\}$ such that apply(iter(i, g, m), $d_1, \ldots, d_n) = d$ is provable, provided no further application of def occurs in g.

By adding the axiom (*cf.* de Bakker [3]):

$$(\forall\, \underline{\text{nat}}\, m : \text{apply}(\text{iter}(i, g, m), d_1, \ldots, d_n) = \underline{\text{error}})$$
$$\Rightarrow \quad \text{apply}(\text{def}(i, g), d_1, \ldots, d_n) = \underline{\text{error}}$$

to type MAP″, the resulting type contains only fg-models in C_Z of type MAP.

If all basic functions f_i of MAP are continuous, then MAP″ is consistent and possesses a strongly initial fg-model, which is a strongly initial fg-model in C_Z. Furthermore it has a terminal model in which two terms of sort $\underline{\text{map}}$ without free identifiers are equal iff they are mathematically (i. e. extensionally) equivalent.

Now we may introduce a function:

$\underline{\text{funct}}$ ($\underline{\text{map}}$, $\underline{\text{nat}}$) $\underline{\text{map}}$ approx

specified by:

$$\begin{aligned}
&\text{approx}(f'_j, n) = f'_j,\\
&\text{approx}(s_j, n) = s_j,\\
&\text{approx}(\text{fid}(h), n) = \text{fid}(h),\\
&\text{approx}(\text{fcond}(p, g, h), n) = \text{fcond}(\text{approx}(p, n), \text{approx}(g, n),\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{approx}(h, n)),\\
&\text{approx}(\text{comp}(g_0, \ldots, g_m), n) = \text{comp}(\text{approx}(g_0, n), \ldots,\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{approx}(g_m, n)),\\
&\text{approx}(\text{def}(k, g), 0) = \Omega,\\
&\text{approx}(\text{def}(k, g), n+1) =\\
&\qquad\qquad \text{sub}(k, \text{approx}(g, n+1), \text{approx}(\text{def}(k, g), n)),
\end{aligned}$$

The introduction of approx allows to distinguish extensionally equivalent functions, if they are not "operationally equivalent". Moreover approx defines approximations for "recursively defined" objects of sort $\underline{\text{map}}$; i. e. we have:

$$\forall\, \underline{\text{map}}\, f, \underline{\text{data}}\, d_1, \ldots, d_n, d, \underline{\text{nat}}\, n :$$
$$\text{apply}(\text{approx}(f, n), d_1, \ldots, d_n) = d \quad \Rightarrow \quad \text{apply}(f, d_1, \ldots, d_n) = d.$$

Moreover for each term t of sort $\underline{\text{map}}$ the values of:

$$\text{apply}(\text{approx}(t, n), d_1, \ldots, d_n)$$

are uniquely determined (independently from the particular model of $\underline{\text{MAP}}$).

In the type MAP″ which is generated by enriching MAP by the function approx we again have weakly terminal *fg*-models, which now characterize some notion of "operational equivalence".

4. CONCLUDING REMARKS

If the meaning of programming languages is to be specified by algebraic theories using first order equations, the fixed point properties can be expressed quite straight forward (*cf.* also [32]). However, generally it is impossible to specify in first order, that a function has to be the least (defined) fixed point of a functional, since this is a second order property. If the concept of initiality, which is a second order characterization, is extended appropriately, however, by considering partial homomorphisms, then it also captures the notion of least definedness and thus of least fixed points.

For abstract types specifying programming languages by first order conditional equations such initial models can never be fully abstract in the sense of Milner [25]: the extensional equality of recursive functions cannot be expressed. However, fully abstract models, that give the "real" mathematical meaning, can be found by taking the terminal algebra in the class of models that are extensionally equivalent to the initial model.

Apart from being interested to find "the" meaning of a programming language, it is also appealing to consider the class of all models of it as specified by some algebraic theories. If the fixed point properties of the resp. programs are expressed properly, the class of models has a similar structure as the class of fixed points of some function. However, in addition each class of extensionally equivalent models (corresponding to particular fixed points) can be further structured by considering various congruence relations on the terms representing the functions.

Note, that we did not introduce any explicit notion of monotonicity or continuity for defining the semantics of our language. This is possible because partial functions and thus partial homomorphisms are monotonic if naturally extended to flat domains (*cf.* [23]).

The algebraic approach to language semantics is not restricted to applicative languages, i. e. recursive functions, but can also be applied for procedural languages to define the semantics of program variables and assignments (*cf.* Pepper [27], Broy, Wirsing [8]), procedures (*cf.* Gaudel [17], Pair [26]), parallelism (*cf.* Broy [6], Broy, Wirsing [12]), or nondeterminism (*cf.* Broy, Wirsing [10]).

Different notions of equivalences of programs ranging from mathematical ("functional" or "extensional") equivalence to algorithmic (*cf.* Broy [6]) and to operational (computational) equivalence and at last to syntactic equality can contribute to the better understanding of the concepts of programming languages (*cf.* Broy *et al.* [14]).

Algebraic definitions of programming languages of the kind of type MAP may be viewed as restricting the class of possible semantic models of some programming language by specific axiomatic rules. If these axioms are weakly sufficiently complete and a weakly terminal *fg*-model exists, then such an algebraic definition can even be considered as a complete semantic definition by taking the weakly terminal *fg*-model as "mathematical semantics". Further complementary semantic definitions then can be verified to be consistent with the algebraic definition by showing their "extensional equivalence" to the partially initial *fg*-model.

ACKNOWLEDGEMENT

We like to thank our colleages, P. Pepper for interesting discussions, and W. Dosch for reading carefully drafts.

REFERENCES

Note: L.N.C.S., Lecture Notes in Computer Science;
I.C.A.L.P., Int. Colloquium on Automata, Languages and Programming.

1. J. Backus, *Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs*, Comm. A.C.M., Vol. 21, No. 8, 1978, pp. 613-641.
2. F. L. Bauer and H. Wössner, *Algorithmische Sprache und Programmentwicklung*, Berlin-Heidelberg-New York, Springer, 1981.
3. J. W. de Bakker, *Least Fixed Points Revisited*. In C. Böhm, Ed., λ-Calculus and Computer Science Theory, Roma, L.N.C.S., Vol. 37, pp. 27-62, Berlin, Springer, 1975.
4. J. A. Bergstra and J. V. Tucker, *Algebraic Specification of Computable and Semi-Computable Data Structures*, Afdeling Informatica Amsterdam, IW 115/79, 1979.
5. G. Birkhoff and J. D. Lipson, *Heterogeneous Algebras*, J. Comb. Theory, Vol. 8, 1970, pp. 115-133.
6. M. Broy, *Transformation Parallel Ablaufender Programme*, Dissertation, Technische Universität München, Fakultät für Mathematik, 1980.

7. M. BROY, W. DOSCH, H. PARTSCH, P. PEPPER and M. WIRSING, *Existential Quantifiers in Abstract Data Types*. In H. A. MAURER, Ed., Proc. of the Sixth I.C.A.L.P., Graz, L.N.C.S., Vol. 71, pp. 73-87, Berlin: Springer 1979.
8. M. BROY and M. WIRSING, *Programming Languages as Abstract Data Types*. In M. DAUCHET, Ed., 5e Colloque de Lille sur les arbres en algèbre et en programmation, Lille, February 1980, pp. 160-177, Université de Lille, 1980.
9. M. BROY and M. WIRSING, *Partial-Recursive Functions and Abstract Data Types*, Bulletin of the E.A.T.C.S., Vol. 11, June, 1980, pp. 34-41.
10. M. BROY and M. WIRSING, *On the Algebraic Specification of Nondeterministic Programming Languages*. In E. ASTESIANO and C. BÖHM, Eds., 6th Colloquium on Trees in Algebra and Programming, Genova, L.N.C.S., Vol. 112, pp. 162-179, Berlin, Springer, 1981.
11. M. BROY and M. WIRSING, *Partial Abstract Types*, Acta Informatica, Vol. 18, 1982, pp. 47-64.
12. M. BROY and M. WIRSING, *On the Algebraic Specification of Finitary Infinite Communicating Processes*. In D. BJORNER, Ed., I.F.I.P. Working Conference on Formal Description of Programming Concepts II, Garmisch 1982 (to appear).
13. M. BROY, C. PAIR and M. WIRSING, *A Systematic Study of Models of Abstract Data Types*, Centre de Recherche en Informatique de Nancy, Report 81-R-042, 1981.
14. M. BROY, P. PEPPER and M. WIRSING, *On Relations Between Programs*. In B. ROBINET, Ed., 4th International Symposium on Programming, Paris, April, 22nd-24th 1980, L.N.C.S., Vol. 83, pp. 59-78, Berlin, Springer.
15. C. C. CHANG and H. J. KEISLER, *Model Theory*, Studies in Logic and the Foundations of Mathematics, Vol. 73, Amsterdam, North-Holland, 1973.
16. B. COURCELLE and M. NIVAT, *The Algebraic Semantics of Program Schemas*. In J. WINKOWSKI, Ed., Proc. Math. Foundations of Comp. Science, Zakopane, L.N.C.S., Vol. 64, pp. 16-30, Berlin, Springer, 1978.
17. M. C. GAUDEL, *Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation*, Thèse d'État, Nancy, March, 1980.
18. G. GOGUEN, J. W. THATCHER and E. G. WAGNER, *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types*. In R. T. YEH, Ed., Current trends in programming methodology, Vol. 4, Data Structuring, pp. 80-149, N.J., Prentice Hall, 1978.
19. G. GRÄTZER, *Universal Algebra*, Princeton, Van Nostrand, 1968.
20. J. V. GUTTAG, *The Specification and Application to Programming of Abstract Data Types*, Ph. D. Th., Univ. of Toronto, Dept. of Comp. Sc., Rep. CSRG-59, 1975.
21. P. HITCHCOCK and D. PARK, *Induction Rules and Termination Proof*. In M. NIVAT, Ed., Proc. of the first I.C.A.L.P., I.R.I.A., pp. 225-251, Amsterdam, North-Holland, 1973.
22. S. MCLANE, *Categories for the Working Mathematician*, Berlin: Springer, 1971.
23. Z. MANNA, *Mathematical Theory of Computation*, New York, McGraw Hill, 1974.
24. Z. MANNA and A. SHAMIR, *The Theoretical Aspects of the Optimal Fixed Point*, S.I.A.M. J. Comp., Vol. 5, No. 3, 1978, pp. 414-426.
25. R. MILNER, *Fully Abstract Models of Typed λ-Calculi*, T.C.S., Vol. 4, 1977, pp. 1-22.
26. C. PAIR, *Types abstraits et sémantique algébrique des langages de programmation*, Centre de Recherche en Informatique de Nancy, Rapport 80-R-011, 1980.

27. P. Pepper, *A Study on Transformational Semantics.* In F. L. Bauer and M. Broy, Eds., Proc. International Summer School on Program Construction, Marktoberdorf, 1978, L.N.C.S., Vol. 69, Berlin, Springer, 1979, pp. 322-405.

28. W. de Roever, *Recursion and Parameter Mechanism: an Axiomatic Approach.* In J. Loeckx, Ed., Proc. of the second I.C.A.L.P., Saarbrücken, L.N.C.S., Vol. 14, Berlin, Springer, 1975, pp. 34-65.

29. H. Rogers Jr., *Theory of Recursive Functions and Effective Computability*, New York, McGraw-Hill Book Company, 1967.

30. J. R. Shoenfield, *Mathematical Logic*, Reading (Massachusetts): Addison-Wesley, 1969.

31. E. G. Wagner, J. W. Thatcher and J. B. Wright, *Programming Languages as Mathematical Objects.* In J. Winkowski, Ed., Proc. Math. Foundations of Computer Science, Zakopane, L.N.C.S., Vol. 64, Berlin, Springer, 1978, pp. 84-101.

32. M. Wand, *First-Order Identities as a Defining Language*, Indiana University, Comp. Science Department, Technical Report No. 29, 1977.

33. M. Wirsing, P. Pepper, H. Partsch, W. Dosch and M. Broy, *On Hierarchies of Abstract Types*, Acta Informatica (to appear). Preliminary version: Technische Universität München, Institut für Informatik, TUM-I 8007.

34. M. Wirsing and M. Broy, *Abstract Data Types as Lattices of Finitely Generated Models.* In P. Dembinski, Ed., Proc. Math. Foundations of Computer Science, Rydzyna, L.N.C.S., Vol. 88, Berlin, Springer, 1980, pp. 673-685.