Theoretical Computer Science 33 (1984) 139-174
North-Holland

A SYSTEMATIC STUDY OF MODELS OF ABSTRACT DATA TYPES

M. BROY and M. WIRSING

Department of Computer Science, University of Passau, 8390 Passau, Fed. Rep. Germany

C. PAIR

Centre de Recherche en Informatique de Nancy (CNRS LA 262), France;
and Ministère de l'Education Nationale, 75007 Paris, France

Communicated by M. Nivat
Received September 1982
Revised February 1984

Abstract. The term-generated models of an abstract data type can be represented by congruence relations on the term algebra. Total and partial heterogeneous algebras are considered as models of hierarchical abstract data types.

Particular classes of models are studied and it is investigated under which conditions they form a complete lattice. This theory allows also to describe programming languages (and their semantic models) by abstract types. As example we present a simple deterministic stream processing language.

1. Introduction

Abstract (data) types are formal specifications of (classes of) *heterogeneous algebras* which are called the *models* of abstract types. Such a view of the semantics of abstract types is important, in particular, for their applications; for instance, for the incorporation of abstract types into programming languages [16, 15], the use of abstract types for the definition of programming languages and for the specification of compilers [3, 4, 8, 9, 18, 33, 34, 42], or for the joint development of programs and data structures [37, 17].

However, the study of the models of a type has rarely been done very systematically. The point of view of the ADJ-group [2] is that an abstract type specifies an *initial algebra* of a category of algebras. Wand [42] is interested in *final algebras*, the Milano-group (Bertoni et al. [7]) in *monoinitial algebras*. The study of *all term-generated models* of a type has been initiated by a group of Pisa (Giarratana et al. [19]) and independently proposed by Bauer and then continued by the CIP-group of München [11, 45]. The present study is based on this work; however, the presentation (based on [34]) and several results are new.

One often cited advantage of abstract types is their modularized, hierarchical construction from already predefined types (cf. [14]). We call such types hierarchical types and have made particular efforts for their definition and the study of their models for total (Section 4) and partial algebra semantics (Section 5) as it is needed for the specification of programming languages (cf. [8, 9, 34]).

We prove theorems of the following kind: under certain assumptions the models of an abstract type form a complete lattice, or a semilattice; or if these assumptions cannot be satisfied, we single out 'well-behaved' subclasses of models such as the classes of extensionally equivalent models.

The assumptions concern the properties of a type. Let us first recall the definition of a type used in this paper: an abstract type consists of a signature (Section 2) and of axioms (Section 3).[1] The assumptions may concern *the form of the axioms*. We do not try here to reach a maximal generality (cf. for this [11]), but only a generality appropriate for particular applications. Moreover, we introduce other hypotheses, also of 'syntactic nature', since evidently the interesting results connect syntactic properties of a formal system, i.e., the abstract type, with semantic properties concerning its models. For example, for hierarchical types, the *sufficient completeness* of Guttag [23] and Guttag and Horning [24] ensures the extensional equivalence of all models. But, for programming languages in particular, it is necessary (cf. [10, 34]) to study models of abstract types which are not sufficiently complete (Section 5).

A *complete lattice* is a nonempty *set together with an ordering relation* such that every nonempty subset has a least upper bound and a greatest lower bound. In this study the set will be the class of term-generated models and the ordering will be based on the notion of *homomorphism*: $A \sqsubseteq B$ if there exists a homomorphism from A into B. In fact, this is not an ordering between models but between isomorphism classes of models. The isomorphism classes of term-generated models can be represented by the congruence relations on the *term algebra*. We will use this tool systematically, for it permits to work with the set (the lattice) of congruences. So we need not to make any use of category theory as, e.g., ADJ or Wand. In particular, we prove that the isomorphism classes of the models of a consistent and sufficiently complete hierarchical type T form a lower semilattice w.r.t. $\sqsubseteq$ (see Theorem 4.6); they form a complete lattice if, furthermore, the premises of the axioms of T are of primitive sort (see Theorem 4.7). But, if T is not sufficiently complete, then the isomorphism classes split into disjoint and incomparable lattices of extensionally equivalent models (see Theorem 4.8). By allowing partial functions as interpretations of incompletely specified operations with primitive range, the situation remains the same (see Proposition 5.9), but we can distinguish the class of *locally computable models*.

Such models exist in the more general framework of hierarchical partial types, under the assumption of consistency and partial sufficient completeness (see Theorem 5.11). But there a class of extensionally equivalent models does form only

[1] For many authors, e.g., [1], an abstract data type is indeed a class of algebras specified by a signature and axioms, what is called here the class of models of the type.

an upper semilattice and not a complete lattice (see Theorem 5.10). Furthermore, in the class $\mathscr{C}$ of all classes of extensional equivalence the locally computable models are an initial element w.r.t. the generalised 'less-definedness' ordering of the fixed point theory (see Proposition 5.7). We give syntactic conditions under which the class $\mathscr{C}$ forms a lower semilattice w.r.t. this ordering (see Theorem 5.15).

Finally, as example we present a simple deterministic stream processing language. This language can be used to write programs which consecutively read and consecutively output finite or infinite sequences of integer numbers.

2. Signature and algebras

In this section we review the notions of signature, term algebra and term-generated heterogeneous algebra. In particular, we recall that the set of congruences associated to the algebras of a certain signature forms a complete lattice with respect to set inclusion. Its greatest element is called terminal element and is associated to the algebras where every carrier set contains (at most) one element whereas its least element is called initial element and is associated to the term algebra.

A *signature* $\Sigma = (S, F)$ consists of

- a (finite) set S of *sorts* $s_1, \ldots, s_m$, which will be interpreted as *carrier sets*,
- a (finite) set F of *operation symbols* $f_1, \ldots, f_n$ together with their *functionalities*, $s_{i_1} \times \cdots \times s_{i_p} \to s_j$ ($p \geq 0$, $s_{i_k}, s_j \in S$), where every operation symbol f_i will be interpreted as a function $S_{i_1} \times \cdots \times S_{i_p} \to S_j$ where S_i is the interpretation (or carrier set) associated with s_i. In Sections 2, 3 and 4 we assume that these functions are total.

The operation symbols composed in accordance with the usual rules generate the *terms* of the different sorts (unless stated otherwise, a 'term' does not contain any variable). If $W(\Sigma)_s$ is the set of terms of sort s, then the n-tuple $W(\Sigma) = (W(\Sigma)_{s_1}, \ldots, W(\Sigma)_{s_m})$ is a heterogeneous algebra with the operations $f^{W(\Sigma)}$ for the symbols f of the following signature:

$$f^{W(\Sigma)}: (u_1, \ldots, u_p) \in (W(\Sigma)_{s_{i_1}}, \ldots, W(\Sigma)_{s_{i_p}}) \mapsto f(u_1, \ldots, u_p) \in W(\Sigma)_{s_j}.$$

$W(\Sigma)$ is called the *term algebra* of the abstract type.

An interpretation is an epimorphism from the term algebra; it uniquely determines a *term-generated heterogeneous* Σ-algebra A: A heterogeneous Σ-algebra A is called *term-generated* if there exists an epimorphism from $W(\Sigma)$ into A. $W(\Sigma)$ is *initial* in the class of *all* such algebras. We denote by t^A the image of term t in the algebra A w.r.t. the interpretation homorphism $W(\Sigma) \to A$. Obviously, at most one homomorphism can exist from A into B, if A and B are term-generated Σ-algebras: if this homomorphism exists, it transforms t^A into t^B.

The notion of *congruence over* $W(\Sigma)$ will be the tool for studying the category of term-generated Σ-algebras. Every term-generated Σ-algebra is isomorphic to the

quotient $W(\Sigma)/\sim_A$ of the term algebra $W(\Sigma)$ w.r.t. the congruence $\sim_A$:

$$t \sim_A t' \underset{\text{def}}{\Leftrightarrow} t^A = t'^A.$$

Conversely, every quotient of $W(\Sigma)$ w.r.t. a congruence is a term-generated Σ-algebra.

Therefore, we can reduce the study of this category of algebras to the study of the set $\mathcal{C}(\Sigma)$ of congruences over $W(\Sigma)$. In particular we have

$$\sim_A \subseteq \sim_B \Leftrightarrow \text{there exists a homomorphism from } A \text{ to } B,$$

where $\subseteq$ is the set inclusion (also called 'finer than', 'stronger than').

Proposition 2.1 (cf., for instance, Grätzer [22]). *$\mathcal{C}(\Sigma)$ is a complete lattice w.r.t. $\subseteq$.*

Its *minimum* (or *initial element*) is the identity relation between terms (associated to $W(\Sigma)$).

Its *maximum* (or *terminal element*) is the 'universal congruence' U where all terms of a sort are congruent (associated to the algebras A where every carrier set s^A is a singleton, or empty if $W(\Sigma)_s = \emptyset$). The greatest lower bound $\bigsqcap_{i \in I} \sim_i$ is the conjunction of the congruences

$$t \sim_{\bigsqcap} t' \underset{\text{def}}{\Leftrightarrow} t \sim_i t \quad \text{for all } i \in I,$$

making $\mathcal{C}(\Sigma)$ a complete lower semilattice. Therefore, $\mathcal{C}(\Sigma)$ is a complete lattice (cf. Fig. 1) according to the following well-known lemma.

Lemma 2.2. *A complete lower semilattice having a maximum forms a complete lattice.*

Effectively a set E has a least upper bound

$$\bigsqcup E = \bigsqcap E',$$

where E' is the set of upper bounds.

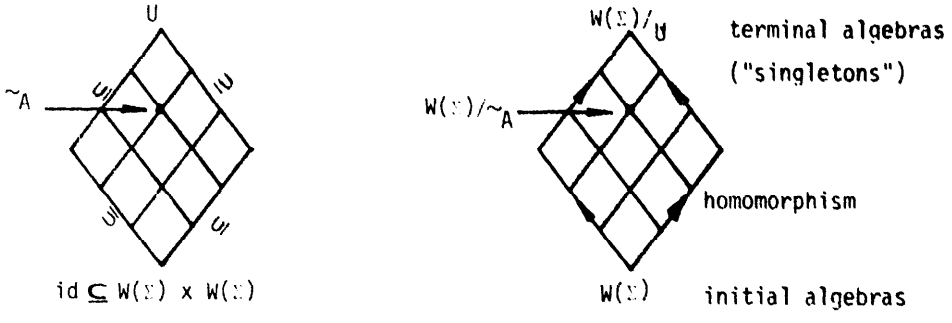

Fig. 1. The lattice $\mathcal{C}(\Sigma)$ of term-generated Σ-algebras.

3. Axioms and models

In this section, the form of the axioms for algebraic types is discussed. It is shown that the set of congruences associated to a type with positive conditional axioms forms a complete lattice. Moreover, a sound and complete proof system (with respect to ground equations) is given.

In algebra the axioms are equations of the form $t \equiv t'$ where t and t' are terms of the same sort: we refer to this as the *equational case.* For the applications in computer science, however, at least axioms with preconditions are reasonable (cf. [25]):

$$b \equiv \text{true} \Rightarrow t \equiv t'$$

if one of the sorts is BOOL, the sort of boolean values; and in order to avoid this particularisation one accepts conditions

$$t_1 \equiv t_1' \Rightarrow t \equiv t'$$

or, more generally, conditional equations (cf. [1, 11]) also called Horn clauses [26]:

$$\bigwedge_{1 \le i \le q} t_i \equiv t_i' \Rightarrow t \equiv t'$$

($q \geqslant 0$, for $q = 0$ we have $t \equiv t'$).

We will work with this kind of axioms: the $t_i \equiv t_i'$ are called the *premises* and $t \equiv t'$ the *conclusion* of the axiom.

Thus we obtain a *first order formal system with equality.* Its theorems are called *theorems of the abstract type.*

A Σ*-algebra* A *satisfies* an axiom of the preceding form if $t_i^A = t_i'^A$ for $i = 1, \ldots, q$ implies $t^A = t'^A$; in other words, if $t_i \sim_A t_i'$ for $i = 1, \ldots, q$, then $t \sim_A t'$: we say also that the *congruence* $\sim_A$ *satisfies* the axiom.

A *model* A of a type is a term-generated Σ-algebra satisfying the axioms. The theorems of the type are also satisfied in A.

Remark 3.1. In Sections 3 and 4 we only use axioms without variables. Indeed, a formula

$$\bigwedge_{1 \le i \le q} t_i \equiv t_i' \Rightarrow t \equiv t'$$

containing at most the free variables $x_1, \ldots, x_u$ of sort $s_{j_1}, \ldots, s_{j_u}$ is satisfied by a Σ-algebra A if, for all $a_1, \ldots, a_u$ of the carrier sets $S_{i_1}^A, \ldots, S_{i_u}^A$,

$$t_j^A[a_1/x_1, \ldots, a_u/x_u] = t_j'^A[a_1/x_1, \ldots, a_u/x_u] \quad \text{for } j = 1, \ldots, q$$

$$\text{implies} \quad t^A[a_1/x_1, \ldots, a_u/x_u] = t'^A[a_1/x_1, \ldots, a_u/x_u].$$

In the case of term-generated algebras this exactly means that A satisfies all the axioms obtained by replacing each variable by term of the same sort. Thus, using axioms with variables does not change the classes of models specified by abstract types.

Proposition 3.2. *The congruences satisfying the axioms E form a complete lattice* $\mathscr{C}(\Sigma)/E$.

Proof. $\mathscr{C}(\Sigma)/E$ is a complete sub—lower—semilattice of $\mathscr{C}(\Sigma)$ since if some congruences satisfy a Horn clause, then their conjunction satisfies this formula, too (cf. for instance [1]). Obviously, U satisfies the axioms; thus we can apply Lemma 2.2. □

By the way, the lattice $\mathscr{C}(\Sigma)/E$ of congruences (cf. Fig. 2) satisfying the axioms E has an initial element the corresponding model of which is an initial algebra in the category of models (and even in the category of possibly not term-generated models, cf. [11]).

Proposition 3.3. *The minimal congruence satisfying the axioms is the 'syntactic congruence'* $\equiv_{sy}$ *which is defined by*

$$t \equiv_{sy} t' \underset{\text{def}}{\Leftrightarrow} t \equiv t' \text{ is a theorem of the type.}$$

Proof. (a) $\equiv_{sy}$ is a congruence and satisfies the axioms since they are Horn clauses.

(b) If $\sim$ is a congruence satisfying the axioms, then $\equiv_{sy} \subseteq \sim$: indeed, if $t \equiv t'$ is a theorem, then $t^A = t'^A$ in every model A, and thus $t \sim t'$ if $\sim$ satisfies the axioms. □

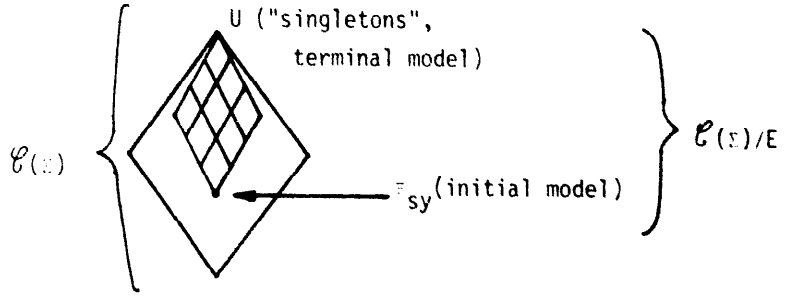

Fig. 2. The lattice $\mathscr{C}(\Sigma)/E$ which is contained in $\mathscr{C}(\Sigma)$.

The proof is done considering the type as a first order formal system. However, it is also true for a simpler formal system (11):

- its formulas are the $t \equiv t'$ where t, t' are terms of the same sort,
- its axioms are the $t \equiv t$,
- its rules of inference are: for each axiom $\bigwedge_{1 \le i \le q} t_i \equiv t_i' \Rightarrow t \equiv t'$, the rule

$$\frac{t_1 \equiv t_1', \ldots, t_q \equiv t_q'}{t \equiv t'}$$

$$\text{(COMP)}\ \frac{t \equiv t', \quad t \equiv t''}{t' \equiv t''} \qquad \text{(SUBST)}\ \frac{t \equiv t'}{f(t_1, \ldots, t, \ldots, t_p) \equiv f(t_1, \ldots, t', \ldots, t_p)}.$$

Symmetry and transitivity of $\equiv_{sy}$ result from (COMP): for instance, symmetry can be derived with $t'' =^{def} t$:

$$\frac{t \equiv t' \quad t \equiv t}{t' \equiv t}.$$

Transitivity then results from symmetry combined with (COMP).

Consequently, this system is also complete for the proof of $t \equiv t'$ which is satisfied by every model. In fact, there exists a model, associated with the congruence $\equiv_{sy}$, where only the equalities which are provable are satisfied: this situation is rather remarkable for a formal system.

We note that these rules are sound for heterogeneous types because of the absence of variables. In the presence of variables the transitivity may cause some problems and therefore, the rules have to be modified (according to Goguen and Meseguer [21] and Huet and Oppen [28]). We also note that because of the restriction to term-generated models no complete proof system can exist w.r.t. formulas containing free variables (cf., e.g., [44]).

Remark 3.4. The converse of part (b) of the proof of Proposition 3.3 is not true: if $\equiv \subsetneq \sim$, then $\sim$ does not necessarily satisfy the axioms. Only if one restricts to the equational case, then every epimorphic image of the initial model is a model too: in the equational case, $\mathscr{C}(\Sigma)/E$ is a complete sublattice of $\mathscr{C}(\Sigma)$. This is not true in the general case. If two congruences satisfy the axioms, their least upper bound does not necessarily do the same. For example, let single, married, widowed, true, false, be 0-ary operations (i.e., constants) with the axiom

$$\text{single} \equiv \text{married} \Rightarrow \text{true} \equiv \text{false}.$$

Then two congruences where

$$\text{single} \sim \text{widowed}, \text{single} \not\sim \text{married}, \text{true} \not\sim \text{false},$$
$$\text{married} \sim \text{widowed}, \text{single} \not\sim \text{married}, \text{true} \not\sim \text{false}$$

satisfy the axioms but not their least upper bound, for which

$$\text{single} \sim \text{widowed} \sim \text{married}, \text{true} \not\sim \text{false}.$$

4. Hierarchical abstract types

The abstract types of the preceding chapter do not exclude trivial models, i.e., models defined by the congruence U, with (at most) one element for every carrier set. In particular, if one uses the boolean values, nothing specifies that, in every model of the type BOOL, true is different from false.

On the other hand, one often constructs types based on (known) data types the axiomatisation of which is presupposed to be known—cf. for instance finite sets of

integers [24], arrays of natural numbers [43], primitive recursive functions over natural numbers [44] or statements of a programming language over the expressions of the same language [38].

For these *primitive types* (integers, natural numbers, expressions) we assume a given model, or in other words a given congruence $=_P$, called ***primitive congruence.*** This congruence could be the initial congruence of the primitive type (cf. [43, 11]), but also other choices are feasible.

A *hierarchical type T* is an abstract type (Σ, E) together with a subsignature $\Sigma_P \subseteq \Sigma$, called *primitive signature* and a ***primitive congruence*** $=_P$ for this subsignature Σ_P.

4.1. Primitive signature and hierarchical algebras

In this section, hierarchical algebras (these are algebras respecting the primitive congruence) and their associated hierarchical congruences are discussed. Using the notion of primitive context we define whether two (hierarchical) congruences are extensionally equivalent. It is shown that the set $\mathscr{C}(\Sigma)$ of all congruences (of signature Σ) is partitioned into disjoint sublattices of extensionally equivalent congruences.

First we define the notions of primitive sort, operation and term.

One designates in the signature Σ of a type a primitive signature $\Sigma_P = (S_P, F_P)$. The sorts $s \in S_P$ are called *primitive sorts*, the operations $f \in F_P$ are called *primitive operations* and must have a functionality which uses only primitive sorts.

The terms which are formed by the primitive operations are called *primitive terms.* Thus, every primitive term is an element of $W(\Sigma_P)_s$ for some $s \in S_P$, and also of $W(\Sigma)_s$. But, in general, $W(\Sigma)_s$ contains $W(\Sigma_P)_s$ properly; there exist terms of primitive sort s which are not primitive terms (see Fig. 3).

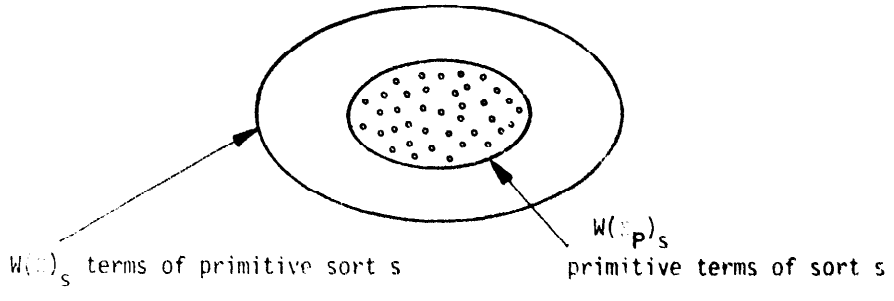

Fig. 3.

An algebra A of a hierarchical type is called *hierarchical algebra* if A is a term-generated Σ-algebra such that the primitive carrier sets and operations form an algebra $A|\Sigma_P$ which is isomorphic to $W(\Sigma_P)/=_P$, i.e., which is an element of the isomorphism class of those term-generated Σ_P-algebras which satisfy exactly $=_P$. The congruence relation induced by a hierarchical algebra on the term algebra is called a *hierarchical congruence.*

Lemma 4.1. *The restriction of a hierarchical congruence* $\sim_A$ *to the primitive terms coincides with the primitive congruence* $=_P$.

Proof. For all primitive terms p and p':

$$p =_P p' \Leftrightarrow p^{A|\Sigma_P} = p'^{A|\Sigma_P} \Leftrightarrow p^A = p'^A \Leftrightarrow p \sim_A p'. \quad \square$$

But not every congruence satisfying the condition of Lemma 4.1 is hierarchical (cf. Fig. 4). The following proposition will give an exact characterisation of such congruences.

Proposition 4.2. *The hierarchical congruences are exactly those congruences over* $W(\Sigma)$ *which satisfy that the primitive terms of every (congruence) class of a term of primitive sort represents exactly and only one class of the primitive congruence* $=_P$.

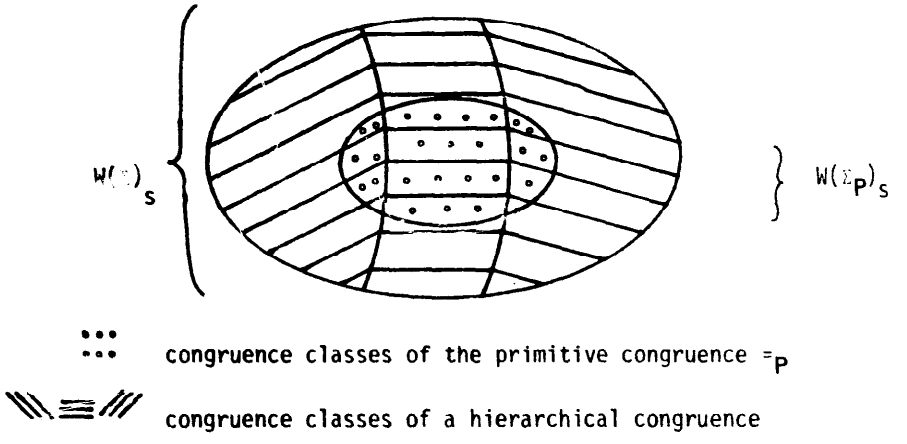

Fig. 4.

Proof. Let A be a hierarchical algebra. Then, according to Lemma 4.1, every congruence class c_t of a term t of primitive sort contains at most one class of $=_P$. c_t contains at least one class since t^A is an element of $A|\Sigma_P$ which is isomorphic to $W(\Sigma_P)/=_P$; hence, there exists a primitive term p ($\in W(\Sigma_P)$) such that

$$t^A = p^{A|\Sigma_P} = p^A.$$

Conversely if every class of a congruence $\sim$ contains one and only one class of $=_P$, then the restriction of $W(\Sigma)/\sim$ to Σ_P is isomorphic to $W(\Sigma_P)/=_P$. $\square$

Corollary. *If two hierarchical congruences* $\sim_A$ *and* $\sim_B$ *verify* $\sim_A \subseteq \sim_B$, *then they have the same restriction to the primitive sorts.*

Definition. Two congruences are *extensionally equivalent* if they have the same restriction to the primitive sorts. Corresponding algebras are also said to be extensionally equivalent.

In some applications, an element of an algebra (e.g., a stack st) is only considered through the functions of primitive sort (e.g., top(st), top(pop(st)), ...). We call every term cn of primitive sort a (primitive) *context* of a term t if cn contains exactly one occurrence of one variable x, the sort of t and x being the same: $cn[t/x]$ will be simply written as $cn[t]$. Note that a context may contain nonprimitive functions, too, and that only the outermost function must range in a primitive sort.

Extensionally equivalent algebras are undistinguishable through the (primitive) contexts, i.e., they have the same 'input–output' behavior.

Lemma 4.3. *The lattice $\mathscr{C}(\Sigma)$ is partitioned in sublattices of extensionally equivalent congruences: for each of them, $\mathscr{C}'$, the maximum is the congruence $\ominus$ defined by*

$$t \ominus t' \Leftrightarrow \textit{for all contexts } cn \textit{ of } t,\ cn[t] \overset{\mathrm{P}}{=} cn[t'],$$

where $\overset{\mathrm{P}}{=}$ is the restriction to the primitive sorts of the congruences of $\mathscr{C}'$ (thus, for hierarchical congruences, $=_{\mathrm{P}}$ is the restriction of $\overset{\mathrm{P}}{=}$ to the primitive terms).

Proof. It is obvious that $\ominus$ is a congruence. Let us study its restriction to terms t, t' of a primitive sort:

$$t \ominus t' \Rightarrow t \overset{\mathrm{P}}{=} t' \quad \text{(with } x \text{ for } cn\text{)},$$

$$t \overset{\mathrm{P}}{=} t' \Rightarrow cn[t] \overset{\mathrm{P}}{=} cn[t'] \quad \text{for all } cn \Rightarrow t \ominus t'.$$

Thus, $\ominus$ belongs to $\mathscr{C}'$.

Now, if $\sim$ is a congruence belonging to $\mathscr{C}'$,

$$t \sim t' \Rightarrow cn[t] \sim cn[t'] \Rightarrow cn[t] \overset{\mathrm{P}}{=} cn[t']$$

for every (primitive) context cn of $t \Rightarrow t \ominus t'$. □

4.2. *Hierarchical models*

A model of a hierarchical type is called *hierarchical model* if it is a hierarchical algebra satisfying the axioms. We shall study the hierarchical congruences satisfying the axioms and, from now, 'hierarchical congruence' will mean 'hierarchical congruence satisfying the axioms'.

The formal system associated with a hierarchical type is the system of the nonhierarchical type extended by the axioms $p = p'$ for all p and p' that are primitive terms with $p =_{\mathrm{P}} p'$. Every hierarchical algebra satisfies these axioms. We shall study the hierarchical congruences. In particular, a question is whether the syntactic congruence $=_{\mathrm{SY}}$ defined by this formal system is a hierarchical congruence. We will see that 'consistency' and 'sufficient completeness' guarantee this. Then, all hierarchical congruences are extensionally equivalent and form a complete lower semilattice with $=_{\mathrm{SY}}$ as initial element (see Theorem 4.6). If, moreover, the premises of the axioms are of primitive sort, then the hierarchical congruences form even a

complete lattice (see Theorem 4.7). On the other hand we show that for a consistent but not sufficiently complete type the set of hierarchical congruences is partitioned into disjoint and (w.r.t. $\subseteq$) incomparable complete lower semilattices of extensional equivalent congruences.

First, we can observe two simple facts.

Fact 4.4. *Every primitive class of* $\equiv_{sy}$ *contains* ***at most*** *one class of* $=_P$ *iff for all primitive terms p and p'*

$$p \equiv_{sy} p' \Rightarrow p =_P p',$$

i.e., the hierarchical type is ***(hierarchy-) consistent*** [23]. *This means that an equality between primitive terms is provable in the whole hierarchical type only if these terms are congruent in the primitive congruence.*

Fact 4.5. *Every primitive class of* $\equiv_{sy}$ *contains* ***at least*** *one class of* $=_P$ *iff for every term of primitive sort t there exists a primitive term p with* $t \equiv_{sy} p$, *i.e., the hierarchical type is* ***sufficiently complete*** [23]. *Equivalently one can say that every term of primitive sort can be proved to be congruent to a primitive term by the proof system of the type.*

Moreover, if $\equiv_{sy}$ is a hierarchical congruence, it is the least one; then, by the consequence of Proposition 4.2, all hierarchical congruences have the same restriction to the primitive sorts: they are extensionally equivalent.

Theorem 4.6. *The syntactic congruence is associated with a hierarchical model iff the hierarchical type is consistent and sufficiently complete. In this case, the hierarchical congruences are extensionally equivalent and form a complete lower semilattice with the syntactic congruence as initial element.*

Proof. The first part of the theorem follows from Facts 4.4 and 4.5. Let us consider the second part. The hierarchical congruences are the elements of the complete lattice $\mathscr{C}(\Sigma)/E$ which are extensionally equivalent to $\equiv_{sy}$. Thus the intersection in the lattice $\mathscr{C}(\Sigma)/E$ of hierarchical congruences is a hierarchical congruence, too. □

Consistency and sufficient completeness do not guarantee that the hierarchical congruences form a complete lattice. Consider e.g., the example of Remark 3.4 and assume that true and false are different elements of primitive sort whereas single, married, widowed are not primitive. This type is consistent and sufficiently complete but does not have any terminal algebra and, hence, cannot form a complete lattice.

In the equational case, however, $\mathscr{C}(\Sigma)/E$ is the sublattice of $\mathscr{C}(\Sigma)$ constituted by the congruences greater than $\equiv_{sy}$ (see Remark 3.4); the congruences extensionally equivalent to $\equiv_{sy}$ also form a sublattice $\mathscr{C}'$ of $\mathscr{C}(\Sigma)$; thus, the hierarchical congruences are those of the intersection of these two sublattices, which is a sublattice of $\mathscr{C}(\Sigma)$. Its maximum is that of $\mathscr{C}'$, given by Lemma 4.3.

All this remains true if the terms t_i, t'_i of the premises of the axioms are of primitive sort (we briefly say that *premises are of primitive sort*); indeed such a type has the same models as an equational one, its axioms are the conclusions $t \equiv t'$ of the axioms the premises of which are satisfied by $\equiv_{sy}$. We have thus proved the following theorem.

Theorem 4.7. *Let T be a hierarchical type such that all premises of axioms are of primitive sort. Then if T is consistent and sufficiently complete, the hierarchical congruences form a complete lattice. The terminal congruence is the* ***extensional congruence*** $\ominus$:

$$t \ominus t' \Leftrightarrow \textit{for all (primitive) contexts } cn \textit{ of } t,\ cn[t] \equiv_{sy} cn[t'].$$

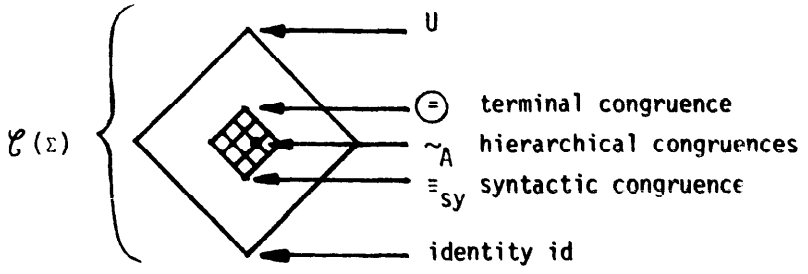

Fig. 5. The lattice of hierarchical congruences of a consistent, sufficiently complete hierarchical type.

Therefore, the hierarchical type T has initial models determined by the syntactic equality $\equiv_{sy}$ and terminal models determined by the extensional congruence $\ominus$ (cf. Fig. 5). The terminal models are *fully abstract* in the sense of [32].

As a consequence of Theorems 4.6 and 4.7, a method can be given to prove an equality $t^{TM} = t'^{TM}$ in a terminal model TM of a type T verifying the hypotheses of Theorem 4.7: add $t \equiv t'$ to the axioms of T and prove the consistency of the obtained type T'. Indeed, if T' is consistent, it possesses a model A which is also a model of T: $t^A = t'^A$ and thus $t^{TM} = t'^{TM}$. Tools to prove consistency are well known, essentially by proving the confluence of an equivalent rewriting system [31, 27].

If a type is *inconsistent*, then no hierarchical congruence can exist which contains $\equiv_{sy}$. An inconsistent type does not have any hierarchical model.

On the other hand, consider consistent hierarchical types which are not sufficiently complete—as, e.g., a type SET over integers together with a function some: **set** → **integer** and the axiom isempty(s) = false ⇒ some(s) ∈ s.

The syntactic congruence of such a type does not correspond to a hierarchical model (cf. Theorem 4.6). Every congruence ~ associated with a hierarchical model must properly contain the syntactic congruence; the restriction of ~ to the primitive sorts is obtained by grouping together every class of $\equiv_{sy}$ without primitive term ('nonstandard class') together with a class containing a primitive term (standard class), to verify the assumptions of Proposition 4.2 (see Fig. 6).

Every regrouping ρ determines a class EXT_ρ of extensionally equivalent hierarchical models. If we add the equations for the regrouping to the axioms, we obtain

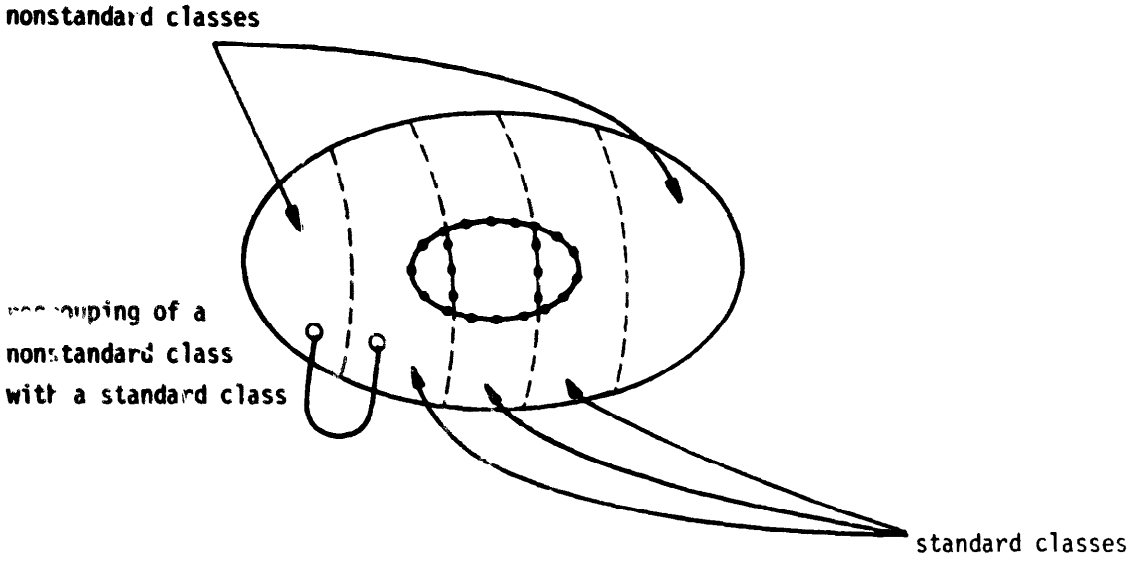

Fig. 6.

a sufficiently complete hierarchical type having exactly the elements of EXT_ρ as hierarchical models, EXT_ρ being nonempty iff this type is consistent. According to Theorems 4.6 and 4.7 the congruences associated with EXT_ρ form a complete lower semilattice and, if the terms of the premises of the axioms are of primitive sort, then they even form a complete lattice. If ρ and ρ' are two different regroupings, the congruences of ρ and ρ' are incomparable w.r.t. the set inclusion $\subseteq$ (cf. Fig. 7). We obtain the following theorem.

Theorem 4.8. *Let T be a consistent hierarchical type. Then every class of extensionally equivalent hierarchical congruences forms a complete lower semilattice. These semilattices are disjoint and incomparable w.r.t. $\subseteq$. If the axioms have premises of primitive sort, then every class is a complete sublattice of $\mathcal{C}(\Sigma)/E$.*

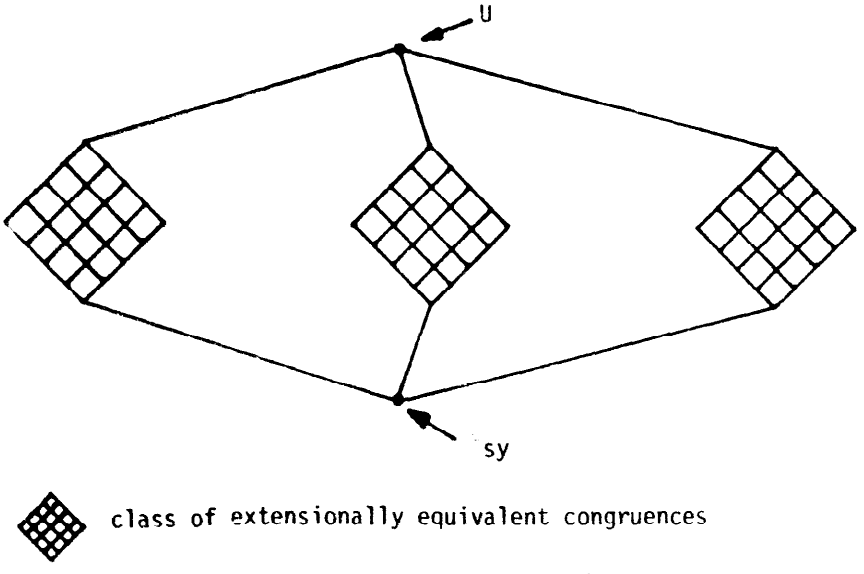

Fig. 7. The classes of congruences associated with a not sufficiently complete type.

Now, unlike the cases of sufficient completeness, a first order proof system, or the proof system Π (cf. Proposition 3.3), is no more complete: if $t \equiv t'$ is satisfied by every model, it is not always provable. For example, let us consider a type with:

- a primitive sort S_1 having one constant p,
- a nonprimitive sort S_2 and operations $a: \to S_2, f: S_2 \to S_1$ without any axiom. Every hierarchical model satisfies $f(a) = p$, but $f(a) \equiv p$ is not provable (for a study of these questions, see [6]).

5. Partial abstract types

Roughly speaking, a type is not sufficiently complete, if an external operation f, i.e., an operation with values in a primitive sort, is not completely specified. Then there exist terms t such that $f(t)$ is not syntactically equivalent to any primitive term: one can say that the value of $f(t)$ is not significant and, therefore, may be interpreted as undefined. This leads us to considering *partial heterogeneous algebras*; the only difference with total ones is that the operations can be interpreted as partial functions. For simplicity we assume that the given model of the primitive type is a total algebra.

5.1. Partial models

In this section the interpretation of the equality symbol $\equiv$ in partial algebras as well as the interpretation of universal quantifier in partial algebras will be discussed.

For a partial algebra A, we write $t^A = t'^A$ if t^A and t'^A are both defined and equal or if they are both undefined. And we write $t^A \stackrel{e}{=} t'^A$ if t^A and t'^A are both defined and equal; otherwise $t^A \stackrel{e}{=} t'^A$ does not hold. ($=$ is the so-called *strong equality* (cf. [45]) and $\stackrel{e}{=}$ the *existential equality* (cf. [5]).) An axiom $t \equiv t'$ is satisfied by a partial algebra A if $t^A = t'^A$. This verification condition is strong: axioms $t \equiv t'$ are excluded where t^A and t'^A are not both defined or both undefined as e.g., mult$(t, 0) = 0$ where t is a term which is undefined in A. In order to avoid such situations one can employ preconditions (cf. also errors in [3]). In contrast to Broy and Wirsing [11] we take here the following position: The undefinedness of terms in the preconditions should not imply the equality of two terms in the conclusion; otherwise, one could obtain (partial) initial algebras which are not recursively enumerable.[2] Thus, we say that a partial algebra A satisfies an axiom

$$\bigwedge_{1 \le i \le q} t_i \equiv t_i' \Rightarrow t \equiv t'$$

without variables if $t_i^A \stackrel{e}{=} t_i'^A$ for $i = 1, \ldots, q$ implies $t^A = t'^A$. As pointed out in [11],

[2] Informally this can be understood by the following axiom: $t \equiv t' \Rightarrow a \equiv b$. If the evaluation of t and t' loops in some algebra A (thus t, t' are undefined in A), then a and b must be equal in A. Thus the noncomputability of some term implies an equality with the immediate effect that in general this equality will not be recursively enumerable.

the strong equality is expressible by the existential equality together with a definedness predicate and vice versa. However, allowing the strong equality in the premises we get a more powerful specification method leading to specifications with noncomputable (hyperarithmetic) algebras.

Let us now examine the case where axioms may contain variables. Free variables in a formula are interpreted as universally quantified. They must hold for all elements of the carrier set of the algebra: a (term-generated) partial algebra A satisfies a formula F with free variables $x_1, \ldots, x_n$ if A satisfies $F(t_1/x_1, \ldots, t_n/x_n)$ for all terms $t_1, \ldots, t_n$ of appropriate sort which are *defined in* A: $t_1, \ldots, t_n$ are *ground terms*, i.e., terms without variables.

Remark 5.1. Partial functions are strict: $f(t)$ is undefined when the interpretation of t is undefined. It is, however, possible to use 'conditionals' in the sense of programming languages. Instead of introducing an operation **if-then-else** into the type (which we do not want to be strict) we may consider an axiom

$$u \equiv \textbf{if } b \textbf{ then } t \textbf{ else } t'$$

as abbreviation of the two axioms

$$b \equiv \text{true} \Rightarrow u \equiv t \quad \text{and} \quad b \equiv \text{false} \Rightarrow u \equiv t'.$$

Another possibility is to define the semantics of **if–then–else** by an evaluation function (as in denotational semantics). This has been done in [8, 9] where the semantics of simple programming languages is completely algebraically defined. In general, fixed point theory (cf. [41]) considers also models with nonstrict operations. This can be done within the algebraic approach by considering generalized heterogeneous partial algebras (cf. [12]).

5.2. The associated total type

In this section, the connections between partial types and total types are studied: to any partial type T (for short, PAT T) a total type $\bar{T}$ is associated by introducing a so-called 'definedness predicate'. Similarly to any partial algebra a total algebra is associated by introducing new 'bottom' elements. Then the totalisations of the models of T are exactly the models of $\bar{T}$ (see Proposition 5.2). Moreover, we use the total type $\bar{T}$ to give a criterion whether a term is defined in T (see Proposition 5.4) and even to establish a sound and complete proof system (with respect to ground atomic formulas) for T.

Now, let a partial type T be given. We extend it to a total type $\bar{T}$ in the following way:

- $\bar{T}$ is a hierarchical type on a unique new[3] primitive sort **Bool** with two 0-ary operations **true** and **false**, interpreted as the boolean values true and false;

[3] This means that **Bool** is assumed to be different of all sorts of T.

- for every sort s of T, an operation $D: s \to \mathbf{BOOL}$; $D(t) \equiv \mathbf{true}$ (abbreviated in the sequel by $D(t)$) expresses the definedness of the term t;
- axioms for D:

(ST) *Strictness*:

$$D(f(t_1, \ldots, t_n)) \Rightarrow D(t_i)$$

for every operation $f: s_1 \times \cdots \times s_n \to s$, and $i = 1, \ldots, n$.

(DP) *Definedness of primitive terms*:

$$D(p)$$

for all primitive terms p.

(UN) *Unicity of undefinedness*:

$$D(t) \equiv \mathbf{false} \wedge D(t') \equiv \mathbf{false} \Rightarrow t \equiv t'$$

for t, t' of the same sort.

- transformation of the axioms of T:

Every axiom

$$(\alpha) \qquad \bigwedge_{1 \le i \le q} t_i \equiv t_i' \Rightarrow t \equiv t'$$

with the free variables $x_1, \ldots, x_n$ is replaced by

$$(\bar{\alpha}) \qquad \bigwedge_{1 \le k \le p} D(x_k) \wedge \bigwedge_{1 \le i \le q} (D(t_i) \wedge t_i \equiv t_i') \Rightarrow t \equiv t'.$$

Then every algebra A of the partial type T can be made into an algebra $\bar{A}$ of $\bar{T}$ in the following way:

- for every sort $s \in S$,

$$s^{\bar{A}} = \begin{cases} s^A \cup \{er_s\} & \text{if there exists a term } t \text{ of sort } s \text{ with } t^A \text{ undefined,} \\ s^A & \text{otherwise,} \end{cases}$$

where er_s is a new element;

- $$D^{\bar{A}}(a) = \begin{cases} \text{true} & \text{if } a \in s^A, \\ \text{false} & \text{if } a = er_s; \end{cases}$$

- for every operation $f: s_1 \times \cdots \times s_n \to s$,

$$f^{\bar{A}}(a_1, \ldots, a_n) = \begin{cases} a & \text{if } (a_1, \ldots, a_n) \in (s_1^A \times \cdots \times s_n^A) \\ & \text{and } f^A(a_1, \ldots, a_n) = a, \\ er_s & \text{if there exists an } i \in \{1, \ldots, n\} \text{ with } a_i = er_s \\ & \text{or } f^A(a_1, \ldots, a_n) \text{ is undefined.} \end{cases}$$

The total algebra $\bar{A}$ is called the *totalisation* of the partial algebra A: two distinct algebras have two distinct totalisations.

Conversely, every algebra B of $\bar{T}$ verifying (ST) and (UN) is the totalisation of an algebra $\tilde{B}$ of T:

- if there exists an element er_s of s^B such that $D^B(er_s) = \text{false}$, then, from (UN), er_s is unique: $s^{\tilde{B}} = s^B - \{er_s\}$; otherwise, $s^{\tilde{B}} = s^B$.
- $f^{\tilde{B}}(a_1, \ldots, a_n)$ is defined, and equal to $f^B(a_1, \ldots, a_n)$ iff $D^B(f^B(a_1, \ldots, a_n)) = \text{true}$.

A structural induction using (ST) shows that $t^{\tilde{B}}$ is defined, and $t^{\tilde{B}} = t^B$, iff $D^B(t^B) = \text{true}$. The verification that $\tilde{\bar{B}} = B$ is then immediate.

Proposition 5.2. *Let T be a PAT and $\bar{T}$ the total type associated with T.*

(1) *If A is an algebra of T, $\bar{A}$ its totalisation, and t a term of sort s, then t^A defined iff $D^{\bar{A}}(t^A) = \text{true}$ and $t^A = t^{\bar{A}}$.*

(2) *The models of $\bar{T}$ are the totalisations of the models of T.*

Proof. Let A be an algebra of T. Its totalisation $\bar{A}$ obviously satisfies (ST), (DP) and (UN).

(1) Already proved with $A = \tilde{B}$ and $\bar{A} = B$.

(2) From (1), A verifies axiom (α) iff $\bar{A}$ verifies ($\bar{\alpha}$): if A is a model of T, then $\bar{A}$ is a model of $\bar{T}$; if B is a model of $\bar{T}$, then $B = \tilde{\bar{B}}$ and $\tilde{B}$ is a model of T. □

Because of the 1–1 correspondence between (partial) models of T and (total) models of $\bar{T}$, the study of the former can be replaced by that of the latter. By definition, the partial (hierarchical) congruence $\sim_A$ associated to a partial (hierarchical) algebra A will be the congruence $\sim_{\bar{A}}$ associated to $\bar{A}$. Thus, $\sim_A \subseteq \sim_B$ means that there exists an homomorphism φ from $\bar{A}$ into $\bar{B}$: $\varphi(f^A(x_1, \ldots, x_n))$ and $f^B(\varphi(x_1), \ldots, \varphi(x_n))$ are strongly equal, i.e., both defined and equal or both undefined.

In the sequel we have to prove that some relations $\sim$ on terms of $\bar{T}$ are congruences verifying axioms. They will be given by:

- an equivalence on the terms of sort **Bool**, hierarchical in the sense that every term is equivalent to **true** or **false**: it is completely defined by the condition for which $D(t) \sim \textbf{true}$; this will be expressed by a unary relation R associated to $\sim$,

$$D(t) \sim \textbf{true} \Leftrightarrow R(t);$$

- $t \sim t' \Leftrightarrow t \equiv_{sy} t'$ or ($D(t) \sim \textbf{false}$ and $D(t') \sim \textbf{false}$) where $\equiv_{sy}$ is the syntactic congruence of $\bar{T}$.

Lemma 5.3. *The previous relation is a congruence verifying* (ST) *if* ($t \equiv_{sy} t'$ *and* $R(t) \Rightarrow R(t')$) *and* ($R(f(t_1, \ldots, t_i, \ldots, t_n)) \Rightarrow R(t_i)$). *It verifies* (DP) *if $R(p)$ for all primitive terms p. It verifies* (UN). It verifies ($\bar{\alpha}$) *if* $R(t) \Rightarrow D(t) \equiv_{sy} \textbf{true}$.

Proof. Relation $\sim$ is obviously reflexive and symmetric. It is transitive because

$t \equiv_{\overline{sy}} t'$ and $\neg R(t) \Rightarrow \neg R(t')$. It is a congruence because $\equiv_{\overline{sy}}$ is, and

$$\neg R(t_i) \Rightarrow \neg R(f(t_1, \ldots, t_i, \ldots, t_n)),$$

$$t \equiv_{\overline{sy}} t' \Rightarrow f(t_1, \ldots, t, \ldots, t_n) \equiv_{\overline{sy}} f(t_1, \ldots, t', \ldots, t_n),$$

$$t \equiv_{\overline{sy}} t' \Rightarrow (R(t) \Leftrightarrow R(t')).$$

The other results are immediate. □

A first application is the following proposition.

Proposition 5.4. *$D(t)$ is a theorem of $\bar{T}$ iff there exists a primitive context cn of t and a primitive term p such that $cn[t] \equiv p$ is a theorem of $\bar{T}$.*

Proof. Since all primitive terms p are assumed to be defined, if $cn[t] \equiv p$ is a theorem, then $D(cn[t])$ is a theorem and also $D(t)$ by repeated applications of (ST).

For the converse let us consider the relation $\sim$ defined as before with $R(t) \Leftrightarrow \exists cn \exists p,\ cn[t] \equiv_{\overline{sy}} p$. Then $R(t) \Rightarrow D(cn[t]) \equiv_{\overline{sy}} \mathbf{true}$ for some context cn and by (ST) we obtain $R(t) \Rightarrow D(t) \equiv_{\overline{sy}} \mathbf{true}$. From Lemma 5.3, it is a congruence satisfying axioms, thus stronger than $\equiv_{\overline{sy}}$. □

Definition. A ground term t of a primitive sort of T is *reducible* if $t \equiv_{sy} p$ for some primitive term p.

Thus, $D(t)$ is a theorem iff t is a subterm of a reducible term.

Remark 5.5. The proof system (II) of Section 3 extended by (ST), (DP) and (UN) is also a proof system for $\bar{T}$. But this system can be simplified. For example, no $D(t) \equiv \mathbf{false}$ can be proved since only the rule (COMP) can lead to such a formula, but with another $D(t) \equiv \mathbf{false}$ as a premise; therefore, (UN) is unuseful. We shall now see that a new system (II1) is sufficient; its formulae are of the form $t \equiv t'$ and $D(t)$, for t, t' ground terms of T:

- *axioms*:

(REF) $t \equiv t$

(DP) $D(p)$ for p primitive term.

- *rules of inference*:

$$\text{(COMP)} \quad \frac{t \equiv t' \quad t \equiv t''}{t' \equiv t''}$$

$$\text{(SUBST)} \quad \frac{t \equiv t'}{f(t_1, \ldots, t, \ldots, t_n) \equiv f(t_1, \ldots, t', \ldots, t_n)}$$

for f operation of T (i.e., $f \neq D$)

$$\text{(SUBSD)}\quad \frac{t \equiv t' \quad D(t)}{D(t')}$$

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

$$(\bar{\alpha}_1)\quad \frac{D(u_1) \ldots D(u_r)\ D(t_1) \ldots D(t_q)\quad t_1 \equiv t_1' \ldots t_q \equiv t_q'}{t \equiv t'}$$

if $\bigwedge_{1 \le i \le q} t_i \equiv t_i' \Rightarrow t \equiv t'$ is obtained from an axiom of T by substituting $u_1, \ldots, u_r$ for the free variables.

It is clear that this system is sound, i.e., its theorems are theorems of $\bar{T}$:

$$(\mathrm{III} \vdash t \equiv t') \Rightarrow t \equiv_{sy} t' \qquad (\mathrm{III} \vdash D(t)) \Rightarrow D(t) \equiv_{sy} \text{true}.$$

The congruence induced by (III) is the least congruence verifying the axioms (ST), (DP) and ($\bar{\alpha}$) since (UN) can be removed without changing theorems.

We now prove that (III) *is a* (*sound and complete*) *proof system for* $\bar{T}$, i.e., for t, t', t'' ground terms, $t \equiv t'$ and $D(t'')$ are theorems of $\bar{T}$ iff they are provable by (III). Note that (III) is a proof system for ground atomic formulas. In general, there does not exist a complete proof system for algebraic types w.r.t. formulas containing variables (cf., e.g., [44]).

The soundness of (III) has already been proved. For the completeness, let us consider the following congruence, hierarchical for **BOOL**, uniquely defined by

$$t \sim t' \Leftrightarrow \mathrm{III} \vdash t \equiv t'$$

$$D(t) \sim \textbf{true} \Leftrightarrow \mathrm{III} \vdash D(t).$$

It is a congruence because of (SUBST) and (SUBSD). It satisfies (ST) because of (STi), (DP), and ($\bar{\alpha}$) because of ($\bar{\alpha}_1$). Thus $\equiv_{sy} \subseteq \sim$:

$$t \equiv_{sy} t' \Rightarrow \mathrm{III} \vdash t \equiv t' \qquad D(t) \equiv_{sy} \text{true} \Rightarrow \mathrm{III} \vdash D(t).$$

5.3. *Nonhierarchical partial models*

Every total nonhierarchical type admits an initial algebra (see Proposition 3.2). For partial types we would like to have a similar property. Since initiality depends crucially on the notion of homomorphism we define in this section two notions of homomorphisms for partial algebras—the so-called 'total' and 'weak' ones.

We show that the algebras which are initial (in the sense of total algebras) in the class of 'minimally defined' algebras of a partial type T (see Proposition 5.6) are also initial in all models of T with respect to total homomorphisms (see Proposition 5.7). Moreover, definedness and equality (with respect to ground terms) in these initial models coincides with the provable definedness and equality in T (see Proposition 5.7). Hence this notion of initiality is also general as the one for total types.

More specifically, let T be a nonhierarchical partial type. Using only axioms of the form $\bigwedge t_i \equiv t_i' \Rightarrow t \equiv t'$ one cannot express any definedness. Hence we assume that at least certain terms are defined such as **true** and **false** of sort **bool**.

The totalisations of the (nonhierarchical models) of T are the models of $\bar{T}$, hierarchical on **Bool**. In general, $\bar{T}$ is not sufficiently complete on **Bool**: it would be the case only if $D(t)$ is provable for all t. The syntactic congruence $\equiv_{\bar{sy}}$ is thus in general not associated with a model (see Theorem 4.6). The congruences associated with models are divided into disjoint lower semilattices: each semilattice is formed with those where the same terms are defined (see Theorem 4.8).

One of these semilattices corresponds to the *minimally defined models*, where t is defined only if $D(t) \equiv_{\bar{sy}}$ **true**. Let $\sim_A$ be a congruence of the following class:

- $t \equiv_{sy} t' \Rightarrow t \sim_A t'$;
- $D(t) \not\equiv_{sy}$ **true** and $D(t') \not\equiv_{sy}$ **true** $\Rightarrow D(t) \sim_A$ **false** and $D(t') \sim_A$ **false**
$\Rightarrow t \sim_A t'$ (from (UN)).

A candidate for the initial congruence of the class is then $\equiv_I$:

- $D(t) \equiv_I$ **true** $\Leftrightarrow D(t) \equiv_{\bar{sy}}$ **true**;
- $t \equiv_I t' \Leftrightarrow t \equiv_{\bar{sy}} t'$ or ($D(t) \not\equiv_{\bar{sy}}$ **true** and $D(t') \not\equiv_{\bar{sy}}$ **true**).

Because of Lemma 5.3 it is actually associated with a model.

Proposition 5.6. *$\equiv_I$ is a congruence associated with a model I of T, initial in the semilattice of minimally defined models.*

For comparing congruences where defined terms are not the same, and thus noncomparable by $\subseteq$, we consider two other orderings corresponding to two different kinds of generalized homomorphisms (cf. [5, 39, 11]). For two partial algebras A and B:

- $\sim_A \sqsubseteq \sim_B$ iff $t \sim_A t'$ and t^A defined $\Rightarrow t \sim_B t'$ and t^B defined (in other words, $t^A \overset{e}{=} t'^A \Rightarrow t^B \overset{e}{=} t'^B$)
- $\sim_A \preccurlyeq \sim_B$ iff $t \sim_A t'$ and t^B defined $\Rightarrow t \sim_B t'$ and t^A defined (this condition is equivalent to ($t \sim_A t' \Rightarrow t \sim_B t'$) and ($t^B$ defined $\Rightarrow t^A$ defined)).

A homorphism $\varphi: A \to B$ for partial algebras is a partial operation which satisfies the usual homorphism property on its domain:

$$f^A(x_1, \ldots, x_n) \text{ defined} \Rightarrow \varphi(f^A(x_1, \ldots, x_n)) = f^B(\varphi(x_1), \ldots, \varphi(x_n)).$$

A *total homorphism* also satisfies

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

Total homorphisms correspond to the ordering $\sqsubseteq$ which preserves defined terms.

An initial congruence for $\sqsubseteq$ defines an initial algebra w.r.t. total homomorphisms and will also be said *initial.*

The relation $\preccurlyeq$ preserves undefined terms: for $\sim_A \preccurlyeq \sim_B$, if t^A is undefined, t^B is undefined. It corresponds to the existence of a *weak homomorphism* φ, i.e., an homomorphism which satisfies a condition converse to the previous one:

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

Thus a weak homomorphism can be a partial operation but it is surjective for term-generated algebras.

Proposition 5.7. (1) *$D(t)$ and $t \equiv t'$ are theorems of $\bar{T}$ iff $t^I \stackrel{e}{=} t'^I$.*

(2) *$D(t)$ is a theorem of $\bar{T}$ iff t^I is defined.*

(3) *$\equiv_I$ is initial in the class of congruences associated with models of T.*

(4) *Every sound and complete proof system π for T (e.g., III, Section 5.2) satisfies the following properties: (for all terms t, t')*

(a) *$\pi \vdash D(t) \Leftrightarrow t^A$ is defined for all models A.*

(b) *$\pi \vdash t \equiv t' \Leftrightarrow t^A \stackrel{e}{=} t'^A$ holds in all models A.*

In particular, property 4(b) *says that only the existential equality can be proved (cf. Remark* 5.5).

Proof

(1) $t^I \stackrel{e}{=} t'^I \Leftrightarrow D(t) \equiv_I \textbf{true}$ and $t \equiv_I t'$

$\Leftrightarrow D(t) \equiv_{sy} \textbf{true}$ and $t \equiv_{sy} t'$.

(2) Particular case of (1) with t, t' identical.

(3) $t^I \stackrel{e}{=} t'^I \Rightarrow D(t) \sim_A \textbf{true}$ and $t \sim_A t'$ for every model A.

(4) Results from (2) and (1).

5.4. Hierarchical partial models

Hierarchical total types admit initial models if they are consistent and sufficiently complete. If the premises of the axioms are of primitive sort, then the hierarchical congruences form a complete lattice. In the case of hierarchical partial types we will see that the existence of initial models is ensured already under weaker conditions: apart from consistency only the so-called 'partial completeness' is needed (see Theorem 5.8). The lattice property, however, is not so easily reached. For a consistent partial type with premises of primitive sort any class of extensionally equivalent congruences only forms a complete upper semilattice with respect to weak homomorphisms (see Theorem 5.10). Only if all functions of nonprimitive range are total, then any class of extensionally equivalent congruences forms a complete $\subseteq$-lattice (see Proposition 5.9). Finally we show in this section that the initial models of a partially complete type are *locally computable.*

First let us recall that a partial model A is hierarchical for a given primitive congruence $=_P$ iff:

(a) if t is a term of primitive sort and t^A is defined, there exists a primitive term p such that $t^A = p^A$.

(b) for p, p' primitive terms $p^A = p'^A \Rightarrow p =_P p'$.

We recall that to study hierarchical models we add $p \equiv p'$ to the axioms whenever p, p' are primitive terms verifying $p =_P p'$ (4.2).

The role played by the initial congruence $\equiv_{\overline{sy}}$ in (4.2) can here be played by the initial congruence $\equiv_I$: it verifies condition (b) iff T is consistent; it verifies condition (a) iff, for t of primitive sort,

$$D(t) \equiv_{\overline{sy}} \textbf{true} \Leftrightarrow \exists p \text{ primitive: } t \equiv_{\overline{sy}} p.$$

Definition. T is *partially complete* if every ground term t of primitive sort with $D(t) \equiv_{sy}$ **true** is reducible.

'Partially complete' is a weaker condition than 'sufficiently complete'. It is sufficient for the existence of a model. From Proposition 5.4, a type is partially complete iff every subterm of primitive sort of a reducible term is reducible, too. We have thus proved the following theorem.

Theorem 5.8. (a) *The initial congruence $\equiv_I$ is associated with a hierarchical partial model of a PAT T iff T is consistent and partially complete.*

(b) *T is partially complete iff, in a reducible term, every subterm of primitive sort is also reducible.*

Let us now study the structure of the class of hierarchical partial congruences, or more exactly of their totalisations. This class can be divided in subclasses of extensionally equivalent congruences relatively to the primitive sorts and **Bool**, each subclass being a lower semilattice w.r.t. $\subseteq$, or even a lattice if the premises of axioms are of primitive sort.

Two models are extensionally equivalent for $\bar{T}$ when

- they are extensionally equivalent for the primitive sorts of T,
- they have the same defined terms.

If only the first condition is verified, the models (and their associated congruences) are said to be extensionally equivalent for T.

As an example of application, let us consider the models where only the functions having a primitive range can be interpreted as partial. Then, the defined terms are those which contain no undefined terms of primitive sort. Thus, two extensionally equivalent models for T have the same defined terms, and are extensionally equivalent for $\bar{T}$. We have therefore proved the following.

Proposition 5.9. *Let T be a consistent hierarchical PAT such that premises of the*

axioms are of primitive sort. The set of congruences associated with models where functions having a nonprimitive range are total, is divided into disjoint complete $\subseteq$-lattices of extensionally equivalent congruences for T.

A class of congruences associated with models extensionally equivalent for T is constituted, in general, of several semilattices corresponding to different defined nonprimitive terms (see Fig. 8). It is uniquely defined by the common restriction ρ

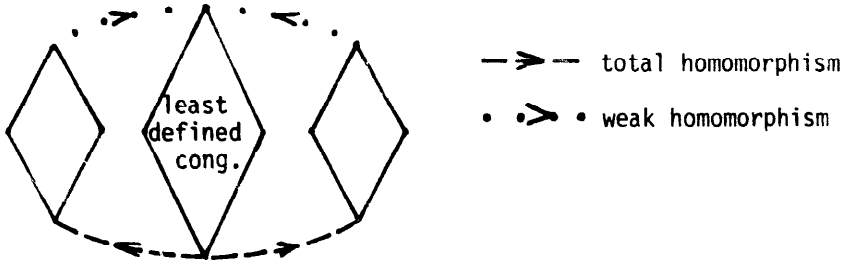

Fig. 8. A class of extensionally equivalent partial congruences.

of the congruences to the primitive sorts (i.e., by a regrouping of classes of syntactically equivalent primitive terms, such that some classes without primitive terms are grouped together and any other class is grouped together with a class of reducible terms, see Fig. 9). Such a class of congruences is denoted by EXT_{ρ}.

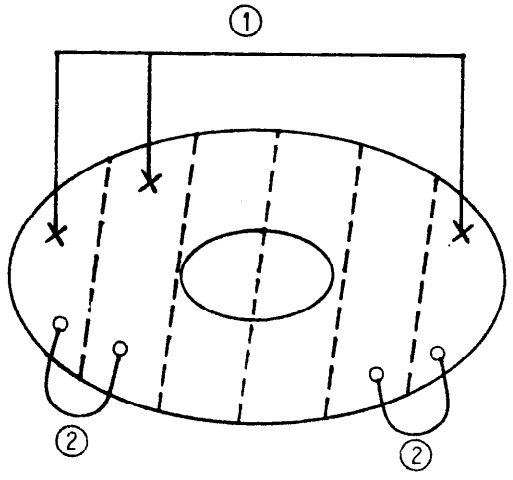

Fig. 9. Two different regroupings into $\stackrel{P}{=}$; ① leads to locally computable models.

Theorem 5.10. *A class of extensionally equivalent models of a type T has an initial element. If the premises of the axioms are of primitive sort, then the congruences associated to the models of the class form a complete upper semilattice w.r.t.* $\leqslant$; *the terminal element is the terminal minimally defined congruence of the class.*

Proof. We consider a class EXT_{ρ}.

(a) If p is a primitive term and t a term of the same sort, the models of EXT_ρ satisfy the equation $t \equiv p$ if $t \rho p$. Adding these equations to the axioms, we obtain a partially complete type T_ρ: indeed, if $D(t)$ is a theorem, t^A is defined for every model A of EXT_ρ, thus $t \rho p$ for some primitive term p. Therefore, T_ρ has an initial congruence (which is the initial congruence of the semilattice of the minimally defined congruences of T_ρ).

(b) We suppose now that premises of axioms are of primitive sort. Extensionally equivalent congruences satisfying axioms (ST), (DP) and (UN) form a complete lattice w.r.t. $\subseteq$ (see Theorem 4.7). If two of these congruences verify $\sim_A \leqslant \sim_B$ and if $\sim_A$ satisfies axiom ($\bar{\alpha}$), then $\sim_B$ satisfies this axiom. Thus, EXT_ρ is an upper semilattice w.r.t. $\leqslant$.

(c) There exists in EXT_ρ a $\leqslant$-terminal congruence which is one of the terminal congruences $\ominus_i$ of the semilattices $\mathscr{C}_i$ constituting EXT_ρ: from Theorems 4.7 and 4.8,

$$t \ominus_i t' \Leftrightarrow \text{for all contexts } cn \text{ of } t \text{ and } t',\ cn[t] \rho\, cn[t'] \text{ and } t,\ t' \text{ both defined or both undefined in the models of } \mathscr{C}_i$$

(because, for $\bar{T}$, D is a context). For the minimally defined models, t is defined if, for some context, $cn[t]$ is reducible in T_ρ: then, for the terminal congruence $\ominus$:

$$t \ominus t' \Leftrightarrow \text{for all contexts } cn \text{ of } t \text{ and } t',\ cn[t] \rho\, cn[t'].$$

Consequently, $t \ominus_i t' \Rightarrow t \ominus t'$. Moreover $D(t) \ominus \mathbf{true} \Rightarrow D(t) \ominus_i \mathbf{true}$. Therefore, $\ominus_i \leqslant \ominus$. □

An interesting class of extensionally equivalent models is defined by grouping in ρ all terms of primitive sort which are not reducible (Fig. 9). They are the models A verifying for p primitive and t of the same sort,

$$t^A = p^A \Rightarrow t \equiv_{\mathrm{sy}} p$$

(and therefore $t^A = p^A \Leftrightarrow t \equiv_{\mathrm{sy}} p$ since the converse is always true). Such models (and congruences) can be called *locally computable* in the following sense: if t^A is defined, and thus equivalent to a primitive term p, $t \equiv p$ is provable.[4] If the type is partially complete, the initial model I is locally computable.

Theorem 5.11. *A consistent hierarchical PAT has a locally computable model iff it is partially complete. Then, the initial model I is locally computable. If moreover the premises of axioms are of primitive sort, the locally computable congruences form a complete upper semilattice w.r.t. $\leqslant$; the terminal congruence $\ominus$ of this semilattice verifies*

$$t \ominus t' \Leftrightarrow \textit{for all contexts } cn \textit{ of } t \textit{ and } t',\ cn[t] \equiv_{\mathrm{sy}} cn[t'] \textit{ or } cn[t],\ cn[t'] \textit{ not reducible,}$$

$$D(t) \ominus \mathbf{true} \Leftrightarrow D(t) \equiv_{\mathrm{sy}} \mathbf{true} \Leftrightarrow \textit{for some context } cn \textit{ of } t,\ cn[t] \textit{ is reducible.}$$

[4] Therefore, every function with primitive range can be seen as a partial recursive function (cf. [6]).

Proof. If the type is partially complete, I is a locally computable model. Conversely, if A is a locally computable model and $D(t) \equiv_{sy}$ **true**, t is defined for A, $t^A = p^A$ for some primitive term p and therefore $t \equiv_{sy} p$. The last part of the theorem results from Theorem 5.10 and from Proposition 5.4. □

5.5. Structure of the set of classes of extensionally equivalent models

For total algebras, all lattices of extensionally equivalent models were disjoint and incomparable. For partial algebras the 'less defined' ordering of the fixed point theory would be a natural candidate for structuring such classes. We say that a class EXT_ρ of extensionally equivalent models is *less defined* than $\mathrm{EXT}_{\rho'}$ (for short $\mathrm{EXT}_\rho \sqsubseteq \mathrm{EXT}_{\rho'}$) if for some congruence of EXT_ρ and some congruence of $\mathrm{EXT}_{\rho'}$, ρ is partially weaker than ρ' ($\rho \sqsubseteq \rho'$).

Since the restrictions to the primitive sorts of all congruences of one class are the same, this definition is independent of the particular models. If there exist total models, their classes are maximal.

Proposition 5.12. *The following properties are equivalent:*

(1) $\mathrm{EXT}_\rho \sqsubseteq \mathrm{EXT}_{\rho'}$.

(2) *The initial congruence of* $\mathrm{EXT}_{\rho'}$: $\equiv_{I\rho} \sqsubseteq \equiv_{I\rho'}$.

(3) EXT_ρ *contains a congruence partially weaker than some congruence of* $\mathrm{EXT}_{\rho'}$.

Moreover, for a consistent and partially complete hierarchical PAT, the class of all locally computable models is $\sqsubseteq$*-initial in the class of all nonempty classes of extensionally equivalent models.*

Proof. (1 $\Rightarrow$ 2): The models of EXT_ρ are obtained by adding axioms $t \equiv p$ for every primitive term p with $t \rho p$; if $\rho \sqsubseteq \rho'$, $t \rho p$ implies $t \rho' p$. Therefore, the syntactical congruence obtained by adding these axioms is contained in the syntactical congruence associated with ρ' in the same way. Then, from the definition of the initial congruence (Proposition 5.6), $\equiv_{I\rho} \sqsubseteq \equiv_{I\rho'}$.

(2 $\Rightarrow$ 3): Obvious.

(3 $\Rightarrow$ 1): If $\sim_\rho \in \mathrm{EXT}_\rho$ is partially weaker than $\sim_{\rho'} \in \mathrm{EXT}_{\rho'}$, it is also true for their restrictions ρ and ρ' to primitive sorts.

In particular, the class of locally computable models of a partially complete PAT is initial for $\sqsubseteq$. □

Unfortunately, the general form of our axioms—even restricted to premises of primitive sort—does not imply a $\sqsubseteq$-lower semilattice structure as the following example shows.

Example. Let T be a PAT with primitive subtype P such that P consists of two 0-ary operations $a, b: \to P$ with $a \neq b$ and T extends P by three 0-ary operations

$t_0, t_1, t_2 : \to P$ and a function $f: P \to P$. Furthermore, f is defined by the axioms $f(t_1) \equiv t_1$, $f(t_2) \equiv t_2$ and $t_1 \equiv t_1 \Rightarrow f(t_0) \equiv t_2$ (i.e., if t_1 is defined, $f(t_0)$ is equal to t_2).

Consider the following congruences of T:

$$C_1 : \{a, t_0, t_1, t_2, f(t_0), f(t_1), f(t_2)\}, \{b\},$$

$$C_2 : \{a, t_1, t_2, f(t_0), f(t_1), f(t_2)\}, \{b, t_0\},$$

$$C_3 : \{a, t_1, f(t_1)\}, \{b\}, \{t_0, f(t_0), t_2, f(t_2)\},$$

$$C_4 : \{a, t_2, f(t_2)\}, \{b\}, \{t_1, f(t_1), t_0, f(t_0)\}.$$

For C_1 and C_2, every term is defined; for C_3 and C_4, there exists a class of undefined terms. C_3 and C_4 are two incomparable maximal $\sqsubseteq$-lower bounds of C_1 and C_2. The reason is that t_0 and thus $f(t_0)$ must be undefined in every lower bound (since t_0 is identified with two inequivalent primitive terms in C_1 and C_2); but then, according to the last axiom, either t_1 has to be undefined or t_2 must be identified with $f(t_0)$ which is undefined. The undefined terms of the greatest lower bound cannot be uniquely determined.

By restricting the form of the axioms, however, sufficient conditions for the existence of a semilattice structure will be obtained in the next paragraph. The idea is that no term t occurring in the conclusion of an axiom should contain any proper subterm the undefinedness of which could be created by 'ambiguity', as t_0 in the example above where $t_0 = a$ in C_1 and $t_0 = b$ in C_2 where a and b are different primitive terms. A sufficient syntactic condition is that t does not contain any nonprimitive proper subterm of primitive sort.

5.6. *The case of simple axioms*

Partial completeness is a syntactic property which in general is only semidecidable. In this section we will give a (linearly) decidable criterion for partial completeness: the 'simple' form of the axioms (see Proposition 5.14). Consistent partial types with such axioms have an additional property: the set of their hierarchical congruences forms a complete lower semilattice with respect to total homomorphisms, the initial congruence being the least element (see Theorem 5.15). The example of the previous section (Section 5.5) shows that the 'simple axiom condition' is also the weakest condition w.r.t. the form of axioms ensuring the semi-lattice-property.

Definition. We call a term t *simple* if every proper subterm of primitive sort of t is primitive. An *axiom* is called *simple* if both terms of its conclusion are simple.

In the following we will prove that the models of a consistent partial type T with simple axioms forms a complete lower semilattice w.r.t. $\sqsubseteq$; T contains locally computable models and its classes of extensionally equivalent models form a lower semilattice w.r.t. the 'less defined' ordering $\sqsubseteq$.

The proof will proceed in three steps. First we show that T is partially complete, then that the models form a semilattice and consequently that the classes of extensional equivalence form a semilattice.

Definition. A ground term t is called *fully reducible* if every subterm of primitive sort of t is reducible.

Lemma 5.13. *Let T be a hierarchical PAT with simple axioms and let t be a fully reducible ground term. Then, for all ground terms t',*

$$t \equiv_{\overline{sy}} t' \Rightarrow t' \text{ is fully reducible.}$$

Proof. Let FR the set of fully reducible terms and let us consider the congruence, hierarchical for **Bool**, uniquely defined by

$$t \sim t' \Leftrightarrow t \equiv_{\overline{sy}} t' \text{ and } ((t \in \mathrm{FR} \text{ and } t' \in \mathrm{FR}) \text{ or } (t \notin \mathrm{FR} \text{ and } t' \notin \mathrm{FR})),$$

$$D(t) \sim \mathbf{true} \Leftrightarrow D(t) \equiv_{\overline{sy}} \mathbf{true} \text{ and } t \in \mathrm{FR}.$$

It is a congruence because

$$t \equiv_{\overline{sy}} t' \text{ and } t \in \mathrm{FR} \text{ and } t' \in \mathrm{FR} \text{ and } f(t) \notin \mathrm{FR}$$

$$\Rightarrow f(t) \equiv_{\overline{sy}} f(t') \text{ and } f(t), f(t') \text{ nonreducible,}$$

$$t \notin \mathrm{FR} \Rightarrow f(t) \notin \mathrm{FR}.$$

We shall prove that this congruence satisfies axioms (ST), (DP) and ($\bar{\alpha}$) of Section 5.2. Then the proposition results of the fact that $\equiv_{sy}$ is the least congruence verifying these axioms (in fact, $\sim$ and $\equiv_{\overline{sy}}$ are identical).

- (ST) and (DP) are obviously satisfied.
- For ($\bar{\alpha}$): if t is obtained by substituting, into a simple term, fully reducible terms for the variables, t is fully reducible or is a nonreducible term of primitive sort; the same holds for t'.

If moreover $t \equiv_{sy} t'$, then t and t' are both fully reducible or both nonreducible. Thus ($\bar{\alpha}$) is also satisfied by the congruence. □

Proposition 5.14. *A hierarchical PAT with simple axioms is partially complete.*

Proof. Since a primitive term is fully reducible, from Lemma 5.13, a reducible ground term is fully reducible; each of its subterms of primitive sort is reducible. Therefore the type is partially complete (see Theorem 5.8). □

Theorem 5.15. *The set of hierarchical congruences of a consistent hierarchical PAT with simple axioms is a complete lower semilattice w.r.t. $\sqsubseteq$; its initial element is the initial congruence.*

Proof. Let $\mathcal{M}$ be a nonempty set of congruences associated to models of T. To show the existence of a greatest lower bound for $\mathcal{M}$ w.r.t. $\sqsubseteq$, we consider the type $T_{\mathcal{M}}$ obtained from T by adding the axioms $t \equiv t'$ for t, t' simple terms and $t^A \stackrel{c}{=} t'^A$ in all models A such that $\sim_A \in \mathcal{M}$. The models associated to $\mathcal{M}$ are models of $T_{\mathcal{M}}$. The axioms of $T_{\mathcal{M}}$ are all simple. Hence, by Proposition 5.14, $T_{\mathcal{M}}$ is partially complete. Let FR be the set of fully reducible terms of $T_{\mathcal{M}}$.

Let us consider the congruence $\sim$ defined by

- $D(t) \sim \mathbf{true} \Leftrightarrow D(t) \sim_A \mathbf{true}$ for all $\sim_A \in \mathcal{M}$ and $t \in \mathrm{FR}$,
- $t \sim t' \Leftrightarrow t \equiv t'$ is a theorem of $T_{\mathcal{M}}$ or ($D(t) \sim \mathbf{false}$ and $D(t') \sim \mathbf{false}$).

From Lemma 5.4 it is actually a congruence satisfying (ST), (DP) and (UN): indeed, $t \equiv t'$ is a theorem of $T_{\mathcal{M}}$ and $D(t) \sim_A \mathbf{true}$ and $t \in \mathrm{FR} \Rightarrow D(t') \sim_A \mathbf{true}$ and $t' \in \mathrm{FR}$ (see Lemma 5.13).

Relation $\sim$ verifies ($\bar{\alpha}$), too, because, for all $\sim_A \in \mathcal{M}$:

(1) $D(u_k) \sim \mathbf{true} \Rightarrow D(u_k) \sim_A \mathbf{true}$ and $u_k \in \mathrm{FR}$;

(2) $t_i \sim t_i'$ and $D(t_i) \sim \mathbf{true} \Rightarrow t_i \sim_A t_i'$ and $D(t_i) \sim_A \mathbf{true}$.

Then, $t \sim_A t'$ for $\sim_A \in \mathcal{M}$; t and t' are obtained by substituting, in simple terms, fully reducible terms for variables: they are equivalent to simple terms and therefore $t \equiv t'$ is a theorem of $T_{\mathcal{M}}$. Thus $t \sim t'$.

Condition (2) above also means that $\sim$ is a lower bound of $\mathcal{M}$ w.r.t. $\sqsubseteq$.

Moreover, let $\sim_M$ be partially weaker than every congruence of $\mathcal{M}$. We first prove that

$$t^M \text{ defined} \Rightarrow t \in \mathrm{FR}$$

by structural induction on the ground term $t = f(u_1, \ldots, u_n)$: if t^M is defined, then all u_i^M are defined and $u_i^M \in \mathrm{FR}$. Thus, $t \in \mathrm{FR}$ unless it is a nonreducible term of primitive sort. But t^M defined implies t^A defined for all $A \in \mathcal{M}$. Hence, the partial completeness of $T_{\mathcal{M}}$ implies $t \sim_A p$ for some primitive term p in all $A \in \mathcal{M}$. Thus t is reducible.

Now, $t \sim_M t'$ and t^M defined $\Rightarrow \forall \sim_A \in \mathcal{M}$, $t \sim_A t'$ and t^A defined and $t \in \mathrm{FR}$ and $t' \in \mathrm{FR}$. Then $t \equiv t'$ is a theorem of $T_{\mathcal{M}}$; therefore, $t \sim t'$, and $D(t) \sim \mathbf{true}$. Relation $\sim$ is the greatest lower bound of $\mathcal{M}$.

The hierarchical congruences form a complete lower semilattice. Since T is partially complete (see Proposition 5.14), the initial congruence belongs to this lattice and is its initial element. □

Corollary. *The classes of extensionally equivalent models of a consistent hierarchical PAT with simple axioms form a complete lower semilattice w.r.t. the 'less defined' ordering; its initial element is the class of locally computable models* (*see Fig.* 10).

Proof. Let $\mathcal{S}$ be a set of classes of extensionally equivalent models, and $\mathcal{M}$ the set of their initial congruences. $\mathcal{M}$ has a greatest lower bound $\sim$, which belongs to class $\mathrm{EXT}_{\rho 0}$: $\mathrm{EXT}_{\rho 0}$ is a lower bound of $\mathcal{S}$ (Proposition 5.12).

Conversely, if EXT_ρ is a lower bound of $\mathcal{S}$, its initial congruence is partially weaker than those of $\mathcal{M}$, thus than $\sim$, and $\mathrm{EXT}_\rho \sqsubseteq \mathrm{EXT}_{\rho 0}$. □

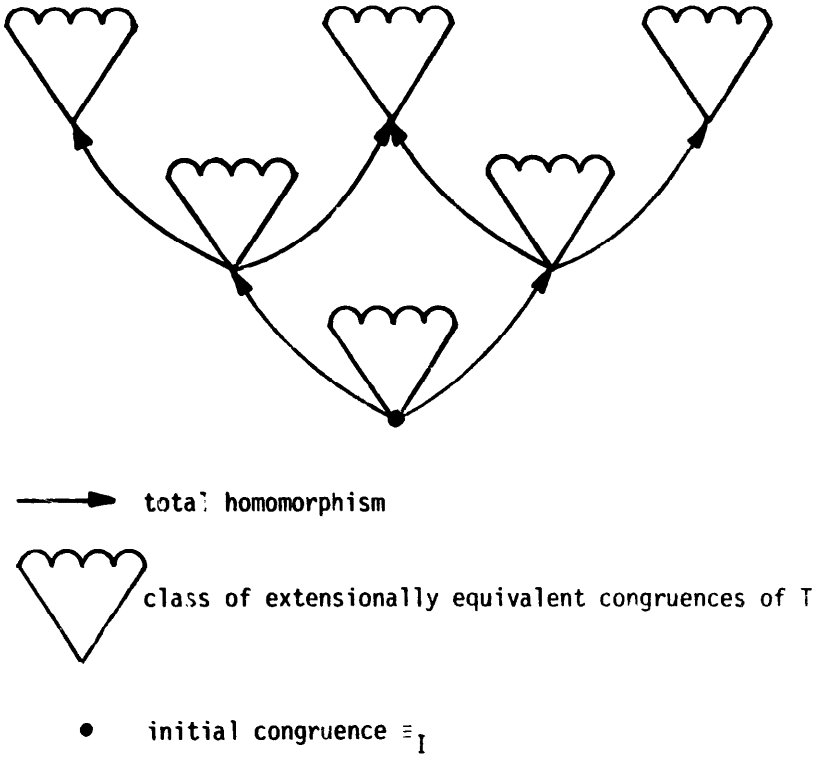

Fig. 10. The $\sqsubseteq$-semilattice of classes of extensional equivalence.

6. Algebraic specification of a deterministic stream processing language

To demonstrate how the algebraic specification of a simple, nontrivial programming language looks like, a specification of a deterministic stream processing language DSPL is given. This language can be used to write programs which consecutively read (possibly infinite) sequences of integers and consecutively output (possibly infinite) sequences of integers.

Two sorts are supposed as primitive:

- **int**, the sort of integers together with the usual operations.
- **id**, the sort of identifiers, with an equality

$$\text{eq}: \mathbf{id} \times \mathbf{id} \to \mathbf{int}: \text{eq}(x, x) \equiv 1, \text{eq}(x, y) \equiv 0 \text{ for } x, y \text{ distinct.}$$

From these two primitive sorts the sort **exp** is firstly constructed. **exp** is the sort of expressions built from integers (the operation intexp: **int** → **exp** converting an integer into an expression), identifiers and the usual operations, together with a substitution function

$$\text{esubst}: \mathbf{exp} \times \mathbf{exp} \times \mathbf{id} \to \mathbf{exp}$$

(where esubst(e, e', x) denotes the substitution of every occurrence of x by e' in e) and an evaluation function

$$\text{val}: \mathbf{exp} \to \mathbf{int}$$

with axioms like

$$\text{val}(\text{intexp}(n)) = n$$

$$\text{val}(\text{add}(e, e')) = \text{val}(e) + \text{val}(e') \quad \text{etc.}$$

For identifiers x the value of val(x) remains unspecified. Hence the sort **exp** is not sufficiently complete: in locally computable models the interpretation val(e) for expressions e containing free identifiers is undefined whereas in other models the interpretation might take an integer value.

Now, the type contains two other nonprimitive sorts **seq** and **agent** which are simultaneously defined. Intuitively, an agent transforms an input sequence into an output sequence.

The sort **seq** comprises the following operations and axioms (which are typical for sequences !):

sort seq

empty: → **seq**

isempty: **seq** → **int**

append: **int** × **seq** → **seq**

top: **seq** → **int**

rest: **seq** → **seq**

isempty(empty) ≡ 1

isempty(append(n, s)) ≡ 0

top(append(n, s)) ≡ n

rest(append(n, s)) ≡ s.

Note, that the value of top(empty) and rest(empty) is not specified. It will be interpreted to undefined in locally computable models.

Only finite sequences can be generated in this way. However, agents will also generate infinite sequences. Hence, sort **seq** cannot be taken as primitive since the set of finite and infinite sequences is not a (finitely generated) model of **seq**.

sort agent

stop: → **agent**

input: **id** × **agent** → **agent**

output: **exp** × **agent** → **agent**

def: **id** × **exp** × **agent** → **agent**

if: **exp** × **agent** × **agent** → **agent**

rec: **id** × **agent** → **agent**

call: **id** → **agent**

process: **agent** × **seq** → **seq**.

The language of agents can be viewed as a procedural language by writing

stop	for stop
read(x); a	for input(x, a)
print(e); a	for output(e, a)
$x := e$; a	for def(x, e, a)
if e **then** $a1$ **else** $a2$ **fi**	for if(e, $a1$, $a2$)
p:: a	for rec(p, a)
call p	for call(p).

The language allows to write just mutually recursive procedures in tail-recursion. So we have a classical sequential input/output stream oriented, iterative, procedural programming language.

Examples. (1) The following agent computes the infinite sequence of the numbers 2^i;

$$\text{def}(x, 1, \text{rec}(p, \text{output}(x, \text{def}(x, 2 * x, \text{call}(p))))).$$

(2) The following agent merges the infinite sequence of numbers 2^i with every ordered (infinite) sequence:

$$\text{def}(x, 1, \text{input}(y, \text{rec}(p, \text{if}(x - y, \text{output}(y, \text{input}(y, \text{call}(p))),$$
$$\text{output}(x, \text{def}(x, 2 * x, \text{call}(p))))))).$$

(3) The agents may be sequentially composed. Assuming $a1$ and $a2$ are agents, then the function

comp: **agent** × **agent** → **agent**

is specified partially completely by

$$\text{process}(\text{comp}(a1, a2), s) \equiv \text{process}(a2, \text{process}(a1, s)).$$

We give now the axioms:

$$\text{process}(\text{stop}, s) \equiv \text{empty}$$

$$\text{process}(\text{input}(x, a), \text{append}(n, s)) \equiv \text{process}(\text{def}(x, \text{intexp}(n), a), s)$$

$$\text{val}(e) \equiv n \Rightarrow \text{process}(\text{output}(e, a), s) \equiv \text{append}(n, \text{process}(a, s))$$

$$\text{def}(x, e, \text{stop}) \equiv \text{stop}$$

$$\mathrm{def}(x, e, \mathrm{input}(x, a)) \equiv \mathrm{input}(x, a)$$

$$\mathrm{eq}(x, y) \equiv 0 \Rightarrow \mathrm{def}(x, e, \mathrm{input}(y, a)) \equiv \mathrm{input}(y, \mathrm{def}(x, e, a))$$

$$\mathrm{def}(x, e, \mathrm{output}(e', a)) \equiv \mathrm{output}(\mathrm{esubst}(e', e, x), \mathrm{def}(x, e, a))$$

$$\mathrm{def}(x, e, \mathrm{if}(e', a1, a2)) \equiv \mathrm{if}(\mathrm{esubst}(e', e, x), \mathrm{def}(x, e, a1), \mathrm{def}(x, e, a2))$$

$$\mathrm{val}(e) > 0 \Rightarrow \mathrm{if}(e, a1, a2) \equiv a1$$

$$\mathrm{val}(e) \leqslant 0 \Rightarrow \mathrm{if}(e, a1, a2) \equiv a2$$

$$\mathrm{rec}(p, a) \equiv \mathrm{asubst}(a, \mathrm{rec}(p, a), p).$$

Relation asubst: **agent** × **agent** × **id** → **agent** is a hidden auxiliary operation with the axioms

$$\mathrm{asubst}(\mathrm{stop}, a, p) = \mathrm{stop}$$

$$\mathrm{asubst}(\mathrm{call}(p), a, p) \equiv a$$

$$\mathrm{eq}(p, q) = 0 \Rightarrow \mathrm{asubst}(\mathrm{call}(q), a, p) \equiv \mathrm{call}(q)$$

$$\mathrm{asubst}(\mathrm{input}(x, a'), a, p) \equiv \mathrm{input}(x, \mathrm{asubst}(a', a, p))$$

$$\mathrm{asubst}(\mathrm{output}(e, a'), a, p) \equiv \mathrm{output}(e, \mathrm{asubst}(a', a, p))$$

$$\mathrm{asubst}(\mathrm{if}(e, a1, a2), a, p) \equiv \mathrm{if}(e, \mathrm{asubst}(a1, a, p), \mathrm{asubst}(a2, a, p)).$$

DSPL defines an abstract type that can be seen as a specification of a programming language. However, DSPL provides only an abstract syntax (i.e., the term algebra). The relationship to existing programming languages may not be seen immediately. Actually, there are several ways of classifying DSPL and relating it to more common notations. It can be seen as an 'assignment-oriented' language, if we write

$$x := e\,;a$$

for $\mathrm{def}(x, e, a)$. However, it can also be seen as an 'applicative' language, if we write

$$(\lambda x.a)(e)$$

for $\mathrm{def}(x, e, a)$. In any case it is rather a 'data flow' language, since it is not required that $\mathrm{val}(e)$ is defined for instance in the axiom

$$\mathrm{def}(x, e, \mathrm{input}(x, a)) = \mathrm{input}(x, a)$$

which certainly does not hold in a classical procedural language like PASCAL.

Objects of sort **agent** are programs that take finite or infinite sequences as input and produce finite or infinite sequences. Terms of sort **agent** built without using the functions rec and call can be seen as trees with stop at the terminal leaves. A computation is a path through such a tree, using an input sequence, chosen in accordance with the if-statements. After elimination of the if-statements, a sequence of applications of input, output and def remains, applied to stop (the ending agent).

Using the functions rec and call also infinite trees can be represented. They may be obtained via an iterated replacement of all occurrences of call(p) in the agent a by the agent rec(p, a).

The axioms of the type are simple. Therefore, the type DSPL is partially complete.

Moreover, the rewriting system obtained by orienting axioms from left to right is confluent [27]. Indeed, it is easy to see that the conflicting left-hand sides are directly confluent; for example, if val(e) $\equiv n$ and val(e') > 0

$$\begin{array}{ccc} & \text{def}(x, e, \text{if}(e', a1, a2)) & \\ \swarrow & & \searrow \\ \text{if}(\text{esubst}(e', e, x), \text{def}(x, e, a1), \text{def}(x, e, a2)) & \longrightarrow & \text{def}(x, e, a1) \end{array}$$

since, if val(e') is defined,

$$\text{val}(\text{esubst}(e', e, x)) \equiv \text{val}(e').$$

Therefore, a reducible term cannot be equivalent to two different primitive terms: the type is consistent. Thus, there exist models.

But we have to avoid models where infinite sequences, like

(1) process(def(x, 1, rec(p, output(x, def(x, 2 * x, call(p))))), empty),

would be interpreted as undefined. More generally, we intend that every process gives a defined result.

To ensure that, we introduce two definedness functions

$$D1 : \textbf{seq} \to \boldsymbol{b} \qquad D2 : \textbf{agent} \to \boldsymbol{b}$$

where $\boldsymbol{b}$ is a primitive sort with one element tr, and the (simple) axioms

$$D1(\text{process}(a, s)) \equiv \text{tr}, \qquad D2(\text{call}(p)) \equiv \text{tr},$$

$$D2(\text{stop}) \equiv \text{tr}, \qquad D2(\text{input}(i, a)) \equiv \text{tr}, \qquad D2(\text{output}(e, a)) \equiv \text{tr},$$

$$D2(\text{def}(i, e, a)) \equiv \text{tr}, \qquad D2(\text{if}(e, a1, a2)) \equiv \text{tr}, \qquad D2(\text{rec}(p, a)) \equiv \text{tr}.$$

Thus, process and the operations generating agents have to be interpreted as total functions.

For example, these axioms ensure definedness of term (1) and therefore

$$\text{top}(\text{process}(\text{def}(x, 1, \text{rec}(p, \text{output}(x, \text{def}(x, 2 * x, \text{call}(p))))), \text{empty}))$$

can be reduced to 1.

The new axioms keep consistency, because they can be only applied in proofs of equations of sort $\boldsymbol{b}$.

Theorems 5.11 and 5.15 then imply:

(1) The type DSPL has an initial model which is locally computable (in fact, the only partial functions in this model are val, top and rest).

(2) The set of the hierarchical partial congruences of DSPL forms a complete lower semilattice w.r.t. $\sqsubseteq$.

(3) Every class of extensionally equivalent congruences is a complete upper semilattice w.r.t. $\preccurlyeq$.

(4) The classes of extensionally equivalent models of DSPL form a complete lower semilattice w.r.t. the 'less-definedness ordering' with the locally computable models as initial element.

The different models define different semantics for the language. Another proof for partial completeness and consistency of the language DSPL can be found in [29] where a term-rewrite system for DSPL is derived for which the confluence has been checked by machine.

7. Concluding remarks

It has been one of the contributions of denotational semantics to demonstrate that programs can be viewed as functions and, therefore, mapped onto particular function spaces in mathematical (denotational) semantics. Abstract types, however, specify a class of possible semantic models. In this class, particular models may be distinguished, such as terminal or initial ones. They may be used to characterize classes of isomorphic semantic models, i.e., to specify the semantics up to isomorphism. In particular, the algebraic approach promises several advantages. First, in this way the description of data structures and programs can be done in one coherent formal framework by a hierarchy of abstract types. Second, it allows for a proper formal definition without considering any unwanted details of concrete representations just talking about the intended functions and their characteristic properties. Third, on such a basis several different concrete semantic definitions (mathematical semantics, operational semantics, axiomatic semantics, etc.) may conveniently be compared and proved to be extensionally equivalent. Fourth, a description of a programming language can be given without defining an explicit domain and partial ordering, which is genrally necessary, if a fixed point semantics is looked for. Note that in some important cases (such as for nondeterministic and concurrent programs) the respective domains and orderings are extremely difficult to be found.

Of course, we do not say that the algebraic, abstract approach to the definition of a programming language makes other methods for description superfluous. We rather propose abstract types as an important tool for the stepwise design and semantic specification of programming languages, independent of particular syntactic or semantic representation, only oriented towards the basic concepts, which may be expressed by algebraic properties. If one has finished the experimental design phase, where several possible closely related languages might be considered, and a specification by abstract types is completed, one should try to give denotational, operational and axiomatic semantics. This second design phase can be used to discuss particular aspects of the language. Simultaneously a concrete syntax can be given.

Finally, the algebraic approach allows for a systematic classification and comparison of data structures and control structures in programming languages. This may lead into an algebraic theory of language concepts (cf. the work of Peter Mosses).

References

[1] J.W. Thatcher, E.G. Wagner and J.B. Wright, Specification of abstract data types using conditional axioms, IBM Res. Rep. RC-6214, 1976.

[2] J.A. Goguen, J.W. Thatcher and E.G. Wagner, An initial algebra approach to the specification, correctness and implementation of abstract data types, in: R. Yeh, ed., *Current Trends in Programming Methodology IV: Data Structuring* (Prentice-Hall, Englewood Cliffs, NJ, 1978) pp. 80–144.

[3] E.G. Wagner, J.W. Thatcher and J.B. Wright, Programming Languages as mathematical objects, *7th MFCS*, Lecture Notes in Computer Science **64** (Springer, Berlin, 1978).

[4] J.W. Thatcher, E.G. Wagner and J.B. Wright, More on advice on structuring compilers and proving their correctness, in: H. A. Maurer, ed., *Proc. 6th Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **71** (Springer, Berlin, 1979) pp. 596–615.

[5] H. Andreka, B. Burmeister and I. Nemeti, Quasivarieties of partial algebras—a unifying approach towards a two-valued model theory for partial algebras, Preprint Nr. 557, FB Mathematik, TH Darmstadt, 1980.

[6] J.A. Bergstra, M. Broy, J.V. Tucker and M. Wirsing, On the power of algebraic specifications, *10th MFCS*, Lecture Notes in Computer Science **118** (Springer, Berlin, 1981) pp. 193–204.

[7] A Bertoni, G. Mauri and P.A. Miglioli, A characterisation of abstract data types as model-theoretic invariants, in: H.A. Maurer, ed., *Proc. 6th Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **71** (Springer, Berlin, 1979) pp. 26–27.

[8] M. Broy and M. Wirsing, Algebraic definition of a functional programming language, TUM 18008, TU München, Institut für Informatik, 1980; Revised version: *R.A.I.R.O. Informatique Theorique* **17** (2) (1983) 137–161.

[9] M. Broy and M. Wirsing, Programming languages as abstract data types, *5ème Coll. Les Arbres en Algèbre et Programmation*, Lille (1980) pp. 160–177.

[10] M. Broy and M. Wirsing, Partial recursive functions and abstract data types, *EATCS Bulletin* **11** (1980) 34–41.

[11] M. Broy and M. Wirsing, Initial versus terminal algebra semantics for partially defined abstract types, TUM-I 8018, Technische Universität München, Institut für Informatik, 1980; Revised version: M. Broy and M. Wirsing, Partial abstract types, *Acta Informatica* **18** (1) (1982) 47–64.

[12] M. Broy and M. Wirsing, Generalized heterogeneous algebras, in: G. Ausiello and M. Protasi, eds., *8th Coll. on Trees and Algebra in Programming*, Lecture Notes in Computer Science **159** (Springer, New York 1983) pp. 1–34.

[13] M. Broy, W. Dosch, H. Partsch, P. Pepper and M. Wirsing, Existential quantifiers in abstract data types, in: H.A. Maurer, ed., *Proc. 6th Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **71** (Springer, Berlin 1979) pp. 73–87.

[14] R.M. Burstall and J.A. Goguen, An informal introduction to specifications using CLEAR, in: R. Boyer and J. Moore, eds., *The Correctness Problem in Computer Science* (Academic Press, New York, 1981).

[15] F.L. Bauer, M. Broy, W. Dosch, B. Krieg-Brückner, A. Laut, M. Luckmann, T. Matzner, B. Möller, H. Partsch, P. Pepper, K. Samelson, R. Steinbrüggen, M. Wirsing and H. Wössner, Programming in a wide spectrum language: A collection of examples, *Sci. Comput. Programm.* **1** (1) (1981) 73–114.

[16] B. Liskov, A. Snydes, R. Atkinson and C. Schaffert, Abstraction mechanism in CLU, *Comm. ACM* **20** (1977) 546–576.

[17] W. Dosch, M. Wirsing, G. Ausiello and G.F. Mascari, Polynomials—the specification, analysis and development of an abstract data type, in: R. Wilhelm, ed., GI-10 Jahrestagung Saarbrücken, *Informatik Fachbericht* **33** (Springer, Berlin, 1980) pp. 306–320.

[18] M.C. Gaudel, Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation, Thèse d'Etat, Nancy, 1980.

[19] V. Giarratana, F. Gimona and U. Montanari, Observability concepts in abstract data type specifications, *5th MFCS*, Lecture Notes in Computer Science **45** (Springer, Berlin, 1976) pp. 576–587.

[20] J.A. Goguen, Abstract errors for abstract data types, UCLA Semantics and Theory of Computation Report 46, 1977; *Proc. IFIP Working Conf. on Formal Description of Programming Language Concepts*, 1977.

[21] J.A. Goguen and J. Meseguer, Completeness of many-sorted equational logic, *SIGPLAN Notices* **16** (7) (1981) 24–32.

[22] H. Grätzer, *Universal Algebra* (Van Nostrand, Princeton, 1968).

[23] J.V. Guttag, The specification and application to programming of abstract data types, Ph.D. Thesis, University of Toronto, 1975.

[24] J.V. Guttag and J.J. Horning, The algebraic specification of abstract data types, *Acta Informatica* **10** (1978) 27–52.

[25] J.V. Guttag, Notes on type abstraction, in: F.L. Bauer and M. Broy, eds., *Program Construction*, Lecture Notes in Computer Science **69** (Springer, Berlin, 1979) pp. 593–616.

[26] A. Horn, On sentences which are true of direct unions of algebras, *J. Symbolic Logic* **16** (1951) 14–21.

[27] G. Huet, Confluent reductions: abstract properties and applications to term rewriting systems, *18th IEEE Symp. on Foundations of Computer Science* (1977) 30–45.

[28] G. Huet and D. Oppen, Equations and rewrite rules: A survey, in: R. Book, ed., *Formal Languages: Perspectives and Open Problems* (Academic Press, New York, 1980).

[29] H. Hussmann, Operativitätskriterien für algebraische Typen, Diplomarbeit, Technische Universität München, Institut für Informatik, 1983.

[30] S. Kamin, Some definitions for algebraic data type specifications, *SIGPLAN Notices* **14** (3) (1979) 28–37.

[31] D. Knuth and P. Bendix, Simple word problems in abstract algebras, in: Leech, ed., *Computational Problems in Abstract Algebras* (Pergamon, Oxford, 1970) pp. 263–297.

[32] R. Milner, Fully abstract models of typed λ-calculi, *Theoret. Comput. Sci.* **4** (1977) 1–22.

[33] P. Mosses, A constructive approach to compiler correctness, in: J. de Bakker and J.v.d. Leeuwen, eds., *Proc. 7th Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **85** (Springer, Berlin, 1979).

[34] C. Pair, Types abstraits et sémantique algébrique des langages de programmation, Rept. 80-R-011, Centre de Recherche en Informatique de Nancy, 1980.

[35] C. Pair, Sur les modèles de types abstraits algébriques, Rept. 80-P-052, Centre de Recherche en Informatique de Nancy, 1980.

[36] C. Pair, Abstract data types and algebraic semantics of programming languages, *Theoret. Comput. Sci.* **18** (1982) 1–13.

[37] H. Partsch and M. Broy, Examples for change of types and object structures, in: F.L. Bauer and M. Broy, eds., *Program Construction*, Lecture Notes in Computer Science **69** (Springer, Berlin, 1979) pp. 421–463.

[38] P. Pepper, A study on transformational semantics, in: F.L. Bauer and M. Broy, eds., *Program Construction*, Lecture Notes in Computer Science **69** (Springer, Berlin, 1979) pp. 322–405.

[39] H. Reichel, Theorie der Aequoide, Dissertation, B. Humboldt Universität Berlin, 1979.

[40] K. Schütte, *Beweistheorie* (Springer, Berlin, 1960).

[41] D. Scott, Continuous lattices, *Proc. 1971 Dalhousie Conf.*, Lecture Notes Mathematics **274** (Springer, Berlin, 1971) pp. 97–136.

[42] M. Wand, First order identities as a defining language, Tech. Rept. 29, Indiana University, Computer Science Department, 1977 (see also: *Acta Informatica* **14** (1980) 337–357).

[43] M. Wand, Final algebra semantics and data type extensions, Tech. Rept. 65, Indiana University, 1978 (see also: *J. Comput. Systems Sci.* **19** (1979) 27–44).

[44] M. Wirsing and M. Broy, Abstract data types as lattices of finitely generated models, *9th MFCS*, Lecture Notes in Computer Science **88** (Springer, Berlin, 1980) pp. 673–685.

[45] M. Wirsing, P. Pepper, H. Partsch, W. Dosch and M. Broy, On hierarchies of abstract data types, Tech. Rept. TUM-18007, TU München, 1980; revised version: *Acta Informatica* **20** (1983) 1–33.